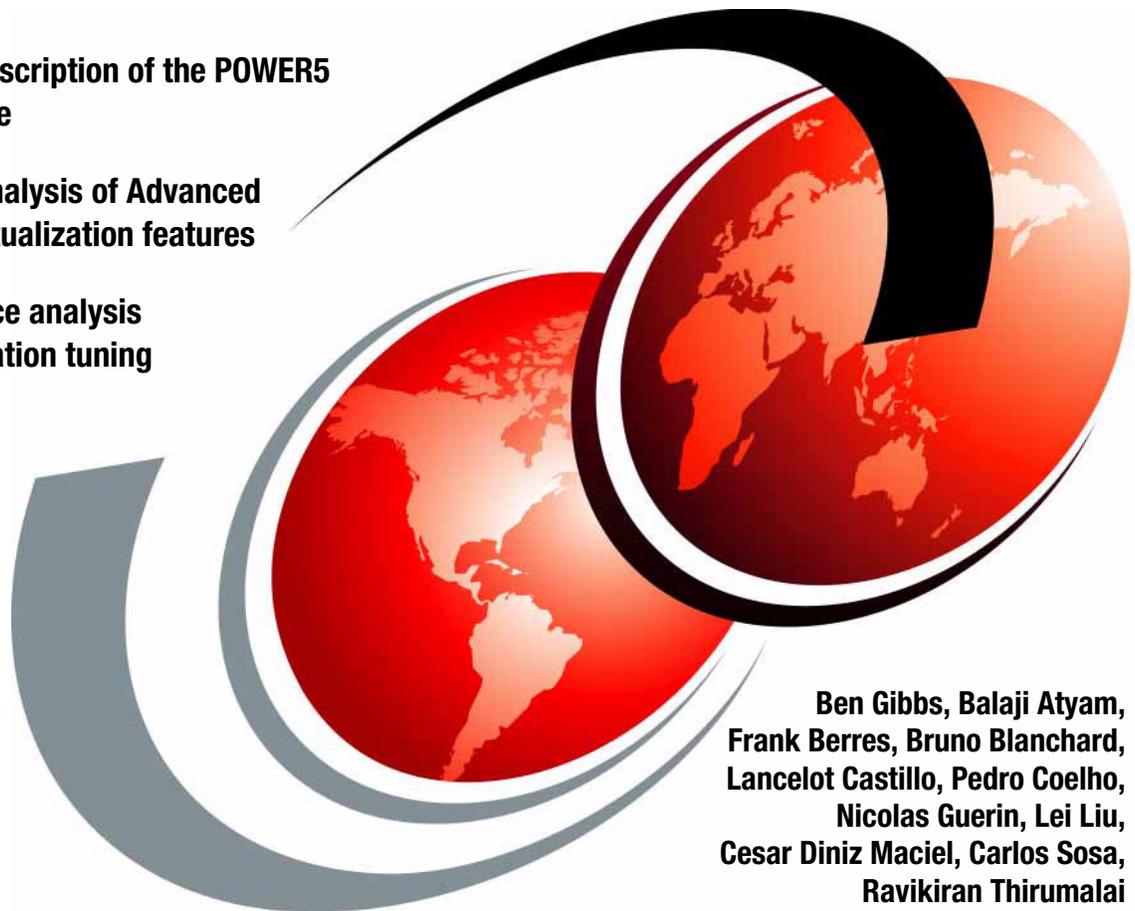


Advanced POWER Virtualization on IBM **e**server p5 Servers: Architecture and Performance Considerations

Detailed description of the POWER5
architecture

In-depth analysis of Advanced
POWER Virtualization features

Performance analysis
and application tuning



Ben Gibbs, Balaji Atyam,
Frank Berres, Bruno Blanchard,
Lancelot Castillo, Pedro Coelho,
Nicolas Guerin, Lei Liu,
Cesar Diniz Maciel, Carlos Sosa,
Ravikiran Thirumalai



International Technical Support Organization

**Advanced POWER Virtualization on IBM @server
p5 Servers: Architecture and Performance
Considerations**

November 2005

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Second Edition (November 2005)

This edition applies to IBM @server p5 servers that include the POWER5 microprocessor architecture and the IBM AIX 5L Version 5.3 operating system.

© Copyright International Business Machines Corporation 2005. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
The specialists who wrote this redbook	xii
Become a published author	xv
Comments welcome	xv
Chapter 1. Introduction	1
1.1 Performance tuning redefined	3
1.1.1 Understanding performance	3
1.1.2 Performance considerations	6
Chapter 2. IBM POWER5 architecture	9
2.1 Introduction	10
2.2 Chip design	12
2.3 POWER5 enhancements	13
2.4 POWER5 instruction pipelines	14
2.4.1 Instruction fetching	15
2.4.2 Branch prediction	16
2.4.3 Instruction decoding and preprocessing	17
2.4.4 Group dispatch	17
2.4.5 Register renaming	18
2.4.6 Instruction execution	19
2.5 Caches	21
2.5.1 Level 2 (L2) cache	25
2.5.2 Level 3 (L3) cache	27
2.5.3 Summary of caches on POWER5	30
2.5.4 Address translation resources	30
2.6 Timing facilities	31
2.7 Dynamic power management	33
2.8 Processor Utilization Resource Register (PURR)	34
2.9 Large POWER5 SMPs	36
2.10 Summary	40
Chapter 3. Simultaneous multithreading	41
3.1 What is multithreading?	42
3.2 POWER5 simultaneous multithreading features	44
3.2.1 Dynamic switching of thread states	46

3.2.2 Snooze and snooze delay	47
3.3 Controlling priority of threads	49
3.3.1 Dynamic resource balancing (DRB)	49
3.3.2 Adjustable thread priorities	50
3.3.3 Thread priority implementation	52
3.4 Software considerations	55
3.4.1 Simultaneous multithreading aware scheduling	55
3.4.2 Thread priorities on AIX 5L V5.3	56
3.4.3 Thread priorities on Linux	58
3.4.4 Cache effects	58
3.5 Simultaneous multithreading performance	59
3.5.1 Engineering and scientific applications	59
3.5.2 Simultaneous multithreading benchmarks	61
3.6 Summary	71
Chapter 4. POWER Hypervisor	73
4.1 POWER Hypervisor implementation	76
4.1.1 POWER Hypervisor functions	79
4.1.2 Micro-Partitioning extensions	85
4.1.3 POWER Hypervisor design	87
4.2 Performance considerations	90
Chapter 5. Micro-Partitioning	93
5.1 Partitioning on the IBM eServer p5 systems	94
5.2 Micro-Partitioning implementation	96
5.2.1 Virtual processor dispatching	104
5.2.2 Phantom interrupts	112
5.3 Performance considerations	115
5.3.1 Micro-Partitioning considerations	116
5.3.2 Locking considerations	121
5.3.3 Memory affinity considerations	126
5.3.4 Idle partition consideration	127
5.3.5 Application considerations in Micro-Partitioning	128
5.3.6 Micro-Partitioning planning guidelines	133
5.4 Summary	142
Chapter 6. Virtual I/O	143
6.1 Introduction	144
6.2 POWER Hypervisor support for virtual I/O	145
6.2.1 Virtual I/O infrastructure	147
6.2.2 Types of connections	149
6.3 The IBM Virtual I/O Server	152
6.3.1 Providing high availability support	156
6.4 Virtual Serial Adapter (VSA)	163

6.5 Virtual Ethernet	164
6.5.1 Virtual LAN	164
6.5.2 Virtual Ethernet connections	169
6.5.3 Benefits of virtual Ethernet	170
6.5.4 Limitations and considerations	171
6.5.5 POWER Hypervisor switch implementation	171
6.5.6 Performance considerations	174
6.5.7 VLAN throughput at different processor entitlements	176
6.5.8 Comparing throughput of VLAN to physical Ethernet	178
6.5.9 Comparing CPU utilization	180
6.5.10 Comparing transaction rate and latency	182
6.5.11 VLAN performance	183
6.5.12 VLAN implementation guidelines	185
6.6 Shared Ethernet Adapter	186
6.6.1 Shared Ethernet Adapter performance	190
6.6.2 Request/response time and latency	193
6.7 Implementation guidelines	196
6.7.1 Guidelines for Shared Ethernet Adapter sizing	197
6.7.2 Guidelines for physical Ethernet sizing	202
6.7.3 Control of threading in the Shared Ethernet Adapter	204
6.8 Virtual SCSI	205
6.8.1 Client and server interaction	209
6.8.2 AIX 5L V5.3 device configuration for virtual SCSI	210
6.8.3 Interpartition communication	212
6.8.4 Disk considerations	215
6.8.5 Configuring for redundancy	217
6.8.6 Performance considerations	220
6.8.7 Sizing a virtual SCSI server	226
6.9 Summary	230
Chapter 7. AIX 5L Version 5.3 operating system support	233
7.1 Introduction	234
7.1.1 Processors	234
7.1.2 Dynamic re-configuration	239
7.1.3 Existing performance commands enhancement	239
7.1.4 New performance commands	248
7.1.5 Paging space	251
7.1.6 Logical Volume Manager (LVM)	252
7.1.7 Virtual local area network (VLAN)	254
7.1.8 EtherChannel	255
7.1.9 Partition Load Manager	255
Chapter 8. POWER5 system performance	257

8.1 Performance commands	258
8.1.1 lparstat command	258
8.1.2 mpstat command	264
8.1.3 vmstat command	268
8.1.4 iostat command	270
8.1.5 sar command	272
8.1.6 topas command	275
8.1.7 xmperf command	278
8.2 Performance tuning approach	283
8.2.1 Global performance analysis	283
8.2.2 CPU analysis	289
8.2.3 Memory analysis	294
8.2.4 Disk I/O analysis	296
8.2.5 Network I/O analysis	304
Chapter 9. Application tuning	311
9.1 Performance bottlenecks identification	312
9.1.1 Time commands, time utilities, and time routines	314
9.2 Tuning applications using only the compiler	317
9.2.1 Compiler brief overview	317
9.2.2 Most commonly used flags	321
9.2.3 Compiler directives for performance	327
9.2.4 POWER5 compiler features	332
9.3 Profiling applications	336
9.3.1 Hardware performance monitor	336
9.3.2 Profiling utilities	343
9.4 Memory management	350
9.5 Optimization of critical sections in the code	351
9.5.1 General rules for optimization strategies	353
9.5.2 Array optimization	353
9.5.3 Loop optimization	355
9.6 Optimized libraries	360
9.6.1 MASS Library	361
9.6.2 ESSL library	368
9.7 Parallel programming general concepts	370
Chapter 10. Partition Load Manager	373
10.1 When and how should I use Partition Load Manager?	374
10.1.1 Partition Load Manager and other load-balancing tools	374
10.1.2 When to use Partition Load Manager	376
10.1.3 How to deploy Partition Load Manager	382
10.2 More about Partition Load Manager installation and setup	383
10.2.1 Overview of Partition Load Manager behavior	383

10.2.2	Management versus monitoring modes	385
10.2.3	Configuration file and tunables	386
10.3	Managing and monitoring with Partition Load Manager	390
10.3.1	Managing multiple partitions	391
10.3.2	Extra tips about the xlplm command	392
10.3.3	Examples of Partition Load Manager commands output	393
10.4	Partition Load Manager performance impact	396
10.4.1	Partition Load Manager resource requirements	396
10.4.2	Partition Load Manager impact on managed partitions.	397
	Related publications	401
	IBM Redbooks	401
	Other publications	402
	Online resources	402
	How to get IBM Redbooks	403
	Help from IBM	403
	Index	405

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law. INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in

any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®	HACMP™	POWER4+™
@server®	IBM®	POWER5™
ibm.com®	Lotus®	PTX®
iSeries™	Micro-Partitioning™	Redbooks™
i5/OS™	Perform™	Redbooks (logo)™
pSeries®	Power Architecture™	RS/6000®
zSeries®	PowerPC Architecture™	Tivoli®
AIX 5L™	PowerPC®	TotalStorage®
AIX®	POWER™	Tracer™
Domino®	POWER2™	WebSphere®
DB2®	POWER3™	
Electronic Service Agent™	POWER4™	

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM® Redbook provides an insight into the performance considerations of Advanced POWER™ Virtualization on the IBM System p5 and IBM @server p5 servers. It discusses the major hardware, software, benchmarks, and various tools that are available.

This book is suitable for professionals who want a better understanding of the IBM POWER5™ architecture and Micro-Partitioning™ that is supported by the IBM System p5 and @server p5 servers. It targets clients, sales and marketing professionals, technical support professionals, and IBM Business Partners.

The Advanced POWER™ Virtualization feature is a combination of hardware and software that supports and manages the virtual I/O environment on POWER5™-based systems. The main technologies are:

- ▶ POWER5 microprocessor architecture with simultaneous multithreading support
- ▶ Micro-Partitioning™
- ▶ Virtual SCSI Server
- ▶ Virtual Ethernet
- ▶ Shared Ethernet Adapter
- ▶ Partition Load Manager

This redbook is intended as an additional source of information that, together with existing sources referenced throughout this document, enhances your knowledge of IBM solutions for the UNIX® marketplace. It does not replace the latest marketing materials and tools.

While the focus in this publication is IBM @server p5 hardware and the AIX® 5L™ operating system, the concepts and methodology can be extended to the i5/OS™ and Linux® operating systems as well as the IBM System p5 and IBM @server i5 platform.

A basic understanding of logical partitioning is required.

The specialists who wrote this redbook

This redbook was the result of two separate residencies and was made up of specialists from around the world working at the International Technical Support Organization, Austin Center.

Ben Gibbs is a Senior Consulting Engineer with Technonics, Inc. (<http://www.technonics.com>) in Austin, Texas. He has more than 20 years of experience with UNIX-based operating systems and started working with the AIX operating system in November 1989. His areas of expertise include performance analysis and tuning, operating system internals, and device driver development for the AIX and AIX 5L™ operating systems. He was the project leader for this IBM Redbook.

Dr. Balaji Atyam has been a Senior Software Engineer in the Systems and Technology Group of IBM since 2000. His responsibilities are porting, benchmarking, performance tuning, parallel programming, and technical consulting services to key Independent Software Vendors in the area of High Performance Computing on IBM @server systems. He received his Ph.D. in Applied Mathematics from Indian Institute of Technology, Roorkee, India. He was a Scientist/Engineer for the Indian Space Research Organization (ISRO) prior to joining IBM.

Frank Berres is a Senior Architect with SerCon GmbH in Germany. SerCon is an IBM company that is assigned to IBM Business Consulting Services (BCS). Frank has more than five years of experience in IT consulting and support on AIX-based systems. He holds a degree in Electrical Engineering from the University of Applied Sciences in Bingen, Germany.

Bruno Blanchard is a Certified IT Specialist working for IBM France in the IGS Strategy Design and Authority team in La Gaude. He has been with IBM since 1983, starting as a System Engineer for VM. He started working with AIX in 1988, using AIX 5L on the IBM RT/PC, PS/2, RS/6000®, SP/2, and pSeries®. He has written many Redbooks™, and is currently working as an Architect in projects deploying @server cluster 1600 and pSeries servers infrastructures for server consolidation and on demand environments.

Lancelot Castillo is an IBM Certified Advanced Technical Expert – pSeries and AIX 5L. He works as a pSeries Product Manager at Questronix Corporation, an IBM Business Partner in the Philippines, and has more than six years of experience in AIX and pSeries servers. Castillo holds a Bachelor's degree in Electronics and Communications Engineering from Mapua Institute of Technology. His areas of expertise include AIX performance tuning and sizing, RS/6000 SP, and HACMP™.

Pedro Coelho is an IT Specialist with IBM Global Services in Portugal. He has five years of experience in AIX, AIX 5L, and Linux in the area of post-sales support and services. He holds a degree in Computer Science from COCITE, Lisbon. His areas of expertise include HACMP and performance analysis and tuning. He is also working with IBM Learning Services teaching beginners and advanced classes on AIX 5L and Linux.

Nicolas Guerin is an IT Specialist working for IBM France in La Gaude. He has eight years of experience in the Information Technology field. His areas of expertise include AIX and AIX 5L, system performance and tuning, HACMP, pSeries, SP, ESS, and SAN. He has worked for IBM for 10 years and is an IBM Certified Advanced Technical Expert - pSeries and AIX 5L. This is his second redbook.

Lei Liu is a Senior IT Specialist working for IBM China at the Technical Sales Support Center in Beijing, where she is responsible for large-account support for telecom clients, including both pre-sale and post-sale technical support. She has more than 13 years of working experience on UNIX systems. She joined IBM in 1998, and her areas of expertise include AIX and AIX 5L, system performance analysis and tuning, and HACMP. She is an IBM Certified Advanced Technical Expert - pSeries and AIX 5L.

Cesar Diniz Maciel is a Certified IT Specialist with the pSeries division in IBM Brazil. He works in pre-sales technical support for pSeries, AIX 5L, and Linux on pSeries, and is a Regional Designated Specialist for Latin America for High End systems and Linux on pSeries. He has worked for IBM since 1996. He has nine years of experience on AIX, AIX 5L, and pSeries systems and holds a degree in Electrical Engineering from UFMG, Belo Horizonte.

Dr. Carlos Sosa is a Senior Technical Staff Member in the Systems and Technology Group of IBM, where he has been a member of the Chemistry and Life Sciences high-performance effort since 2001. For the past 18 years, he has focused on scientific applications with emphasis in Life Sciences, parallel programming, benchmarking, and performance tuning. He received a Ph.D. degree in Physical Chemistry from Wayne State University and completed his post-doctoral work at the Pacific Northwest National Laboratory. His area of interest is future POWER architectures and cellular molecular biology.

Ravikiran Thirumalai is a Software Engineer at IBM India Software Labs. He works for the IBM Linux Technology Center as a kernel developer for the baseOS team. His main areas of interest in the kernel are SMP scalability, locking algorithms, lock-free techniques, and the virtual file system. He has worked in the IT industry for more than 6 years and holds a Bachelor's degree in Electrical and Electronics engineering from Bangalore University and an MS in Software Systems from BITS Pilani.

Thanks to the following people for their contributions to this project:

IBM Austin

Bret Olszewski, Dr. Joel Tendler, Larry Brenner, Luke Browning, Herman D. Dierks, Octavian F. Herescu, Bruce D. Hurley, Harry Mathis, Sujatha Kashyap, Bob Kovacz, Kiet H. Lam, Stephen Nasypany, Frank O'Connell, Tony Ramirez, Sergio Reyes, Jorge D Rodriguez, Luc Smolders, Mysore Srinivas, Suresh Warriar, Erin Burke

IBM Atlanta

Tommy Todd

IBM Brazil

Claudio Garrido, Leonardo Vidal

IBM Dallas

Hari Reddy

IBM Mount Laurel

David Chisholm

IBM Poughkeepsie

David Wootton

IBM Raleigh

Matthew Cali

IBM Somers

Jim McGaughan

IBM Toronto

Robert Enenkel, Arie Tal

IBM Watson

David Klepacki, James B. Shearer

Groupe Bull France

Jez Wain

Technonics, Inc.

Sandra Lopez-Martin

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will team with IBM technical professionals, IBM Business Partners, and/or clients.

Your efforts will help increase product acceptance and client satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

<http://www.redbooks.ibm.com/>

- ▶ Send your comments in an e-mail to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. JN9B Building 905
11501 Burnet Road
Austin, Texas 78758-3493



Part 1

Virtualization technology

In this part, we provide an in-depth look at the technology behind the virtualization capabilities of the IBM @server p5 systems. Detailed information is provided about the IBM POWER5 microprocessor architecture, the POWER Hypervisor, simultaneous multithreading, Micro-Partitioning, and virtual I/O.



Introduction

This book takes an in-depth look at the performance considerations of the IBM @server p5 servers, advancing the concepts and terminology provided in the redbook entitled *Advanced POWER Virtualization on IBM eServer p5 Servers: Introduction and Basic Configuration*, SG24-7940. If you are not familiar with the terminology and concepts of virtualization, we strongly suggest that you read that book before starting the material in this book, as this book assumes that you are already familiar with the concepts of virtualization.

As a quick review, the following terms are defined here:

Virtualization	The pooling of system resources via the POWER Hypervisor to access processors, memory, and I/O devices across logical partitions
POWER Hypervisor	Supports partitioning and dynamic resource movement across multiple operating system environments
Micro-Partitioning	Enables you to allocate less than a full physical processor to a logical partition
Virtual LAN	Provides network virtualization capabilities that enable you to prioritize traffic on shared networks
Virtual I/O	Provides the ability to dedicate I/O adapters and devices to a virtual server, enabling the on demand allocation and management of I/O devices

Capacity on Demand Enables inactive processors and memory to be activated on an as-needed basis

Simultaneous multithreading

Enables applications to increase overall resource utilization by virtualizing multiple physical CPUs through the use of multithreading

Why would the information presented in this book be of importance to you?

Some of the reasons are:

- ▶ Reduce costs by increasing asset utilization.
- ▶ Re-deploy talent to manage your business, not the infrastructure.
- ▶ Rapidly provision new servers.
- ▶ Drive new levels of IT staff productivity.
- ▶ Simplify server management and operations.
- ▶ Communicate more securely with virtual Ethernet.

The IBM @server p5 family of servers includes powerful new capabilities such as the partitioning of processors to 1/10th of a processor, sharing processor resources in a pool to drive up utilization, sharing physical disk storage and communications adapters between partitions, and taking advantage of cross-partition workload management, to mention a few.

Upcoming chapters include an in-depth discussion about each component that makes up the new IBM capabilities available on the POWER5 family of systems, such as POWER Hypervisor, simultaneous multithreading, Micro-Partitioning, virtual LAN, and Virtual I/O.

We hope that by the end of this book you will have a more complete understanding of virtualization as it applies to the POWER5 architecture and the @server p5 systems.

1.1 Performance tuning redefined

With the advent of this new technology and functionality, our traditional concepts and methods of system and applications performance tuning must accommodate the virtual dimension. In addition to defining and explaining these concepts and methods, this book also covers traditional performance issues as well as performance as a function of a system environment with virtual capabilities.

1.1.1 Understanding performance

Technological improvements in microprocessors, disks, and networking equipment have dramatically changed the landscape of server computing. While those improvements have more often than not reduced the incidence of performance problems in client environments, they have also increased the capabilities of systems such that more complex problems may be solved. Thus, performance tuning has tended to change in nature from simple hardware and software bottleneck analysis toward evaluation of more complex interactions. Performance evaluation and tuning of complex systems requires discipline and exactness. Frequently, the solution to a problem is not obvious. Often the steps along the journey toward the solution may seem inconclusive or even counterproductive. But, with a systematic rigor, nearly every bottleneck can be alleviated in some way. To help the reader achieve the goal of making system tuning rewarding and beneficial, we have dedicated one chapter to provide you with tools to help in the effort.

In addition, it is important to note that performance tuning can be subdivided into *system tuning* and *application tuning*. The objectives of these two tuning areas are very different. System tuning relies on the ability to modify system parameters in order to provide faster throughput measurements. This throughput consists of the amount of work performed over a period of time and normally corresponds to a series of identical or different jobs running simultaneously and competing for the same system resources. Application tuning looks at the source code for a particular application and requires tailoring or optimizing the code to a particular architecture or common architectural features.

An IBM @server p5 system is subjected to various types of loads. The load can vary widely depending on the number of applications used and the type of applications being run. Obviously the number of loads and type of applications will vary widely over the period of the server's working life. Consequently, changes have to be made to the server's hardware and software setup to accommodate these changing conditions. Applications require tuning as well.

System administrators often refer to any degradation of service as a *bottleneck* in the server system. Bottlenecks must be understood and compensated for if the

system administrators are to keep the users satisfied with performance. Similarly, programmers must identify bottlenecks within the source code of certain applications that form part of the system load.

System tuning

Within a server there are limited resources that can affect the performance of a given system. Each of these resources work together hand-in-hand and are capable of influencing the behavior of one another. If performance modifications are not carefully administered, the overall effect could be a deterioration of server performance.

Here we distinguish between three types of resources:

- ▶ Logical resources

The resources as seen by the operating system. For example, a central processing unit (CPU) may be available to the operating system as `cpu0`, but it may not be the first CPU installed in the system. The operating system could be installed in the third partition of the system and using the third physical CPU in the system as `cpu0`.

- ▶ Virtual resources

The resources that *appear* to be available for the operating system to use. For example, virtual storage provides the appearance that there is more memory available than is actually installed in the system. With the use of paging space, the operating system no longer has to be limited by the amount of physical memory installed in the system.

- ▶ Physical resources

The actual hardware resources found in the system, such as processors, memory, disk drives, and network adapters. The efficiency of the operating system will maximize the hardware performance.

As server performance is distributed throughout each server component and type of resource, it is essential to identify the most important factors or bottlenecks that will affect the performance for a particular activity.

Detecting the bottleneck within a server system depends on a range of factors such as:

- ▶ Configuration of the server hardware
- ▶ Software applications workload
- ▶ Configuration parameters of the operating system
- ▶ Network configuration and topology

File servers need fast network adapters and fast disk subsystems. In contrast, database server environments typically produce high processor and disk

utilization, requiring fast processors or multiple processors and fast disk subsystems. Both file and database servers require large amounts of memory for caching by the operating system.

Traditionally there has been a simplified approach to performance tuning: If the performance bottleneck is the processor, then either a faster processor or more processors could be installed. An alternative to processor upgrade is to offload processing requirements by using workload management techniques. If the bottleneck is memory, then additional memory could be installed. Memory bottlenecks often result in excessive disk I/O as a result of paging (swapping) between paging space and memory. If the bottleneck is the disk subsystem, then either additional disk drives, disk adapters, or both can be installed. In addition, a specialized high-performance disk subsystem could be used. If the bottleneck is the network adapter then a faster network interface could be installed. Another optimization technique that can be employed is to utilize multiple network adapters in the server increasing throughput onto one or multiple segments.

Before any tuning is actually performed, it is worth understanding the framework within which performance testing is done. Follow a set of simple guidelines to assist in any type of performance analysis.

There are many trade-offs related to performance tuning that have to be considered. In order to choose the best set of options it is vital to ensure that there is a balance between them. The trade-offs are:

- ▶ *Cost versus performance.* In some situations, the only way performance can be improved is by using more or faster hardware while keeping in mind, “Does the additional cost result in a proportional increase in performance?”
- ▶ *Conflicting performance requirements.* When more than one application is used simultaneously, there may be conflicting performance requirements.
- ▶ *Speed versus functionality.* Here, for example, resources may be increased to improve a particular section, but serve as an overall detriment to the system. Using a methodical approach you can obtain improved server performance, such as by:
 - Understanding the factors that can affect server performance, for the specific server functional requirements and for the characteristics of the particular system
 - Measuring the current performance of the server
 - Identifying a performance bottleneck
 - Upgrading the component that is causing the bottleneck

- Measuring the new performance of the server to check for improvement

Although we cover this material in this book, additional information may be found in this redbook: *AIX 5L Performance Tools Handbook*, SG24-6039.

Application tuning

Application tuning (or application optimization) requires careful analysis of the source code to tailor it to a particular hardware architecture. In other words, it is the goal of the programmer to make the application aware of the hardware features that are accessible to the application. For instance, we shall see in the applications tuning chapter how to optimize *do loops* to take advantage of the L2 cache on @server p5 architectures.

We shall see that, in general, any tuning that we carry out at the application level will leverage systems with and without virtual environments. The following redbooks cover this subject on POWER3™ and POWER4™:

- ▶ *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, SG24-5155
- ▶ *The POWER4 Processor Introduction and Tuning Guide*, SG24-7041

1.1.2 Performance considerations

The goal of virtualization is to allow the deployment of resources in a flexible manner. This flexibility enables best use of resources and, when correctly used, should improve performance and the end-user experience. However, virtualization alters the way we look at system performance. We still follow the same rules for identifying existing or potential bottlenecks, but the remedy can be different and difficult to obtain. Virtualization is a flexible resource model for the on demand world. The focus here is more on increasing resource utilization and responding to changing workloads. Resources are dynamically allocated, including fractional, on an as-needed basis. Capacity on Demand (CoD) enables the allocation of additional resources as needed, and Workload Manager (WLM) enables the optimization of resources to respond to changing workloads.

This book follows the model of a system consisting of four major subsystems: *processor complex, memory hierarchy, storage model* and *network topology*.

- ▶ Beginning with Chapter 2, “IBM POWER5 architecture” on page 9, an in-depth look at the IBM POWER5 microprocessor architecture is provided.
- ▶ Chapter 3, “Simultaneous multithreading” on page 41 takes a detailed look at the simultaneous multithreading feature of the POWER5 microprocessor.

Application and system programmers will find the information in chapters 2 and 3 useful to their programming efforts.

- ▶ Chapter 4, “POWER Hypervisor” on page 73 is dedicated to the POWER Hypervisor™ and its role in the system.
- ▶ Chapter 5, “Micro-Partitioning” on page 93 provides detailed information about Micro-Partitioning.
- ▶ Chapter 6, “Virtual I/O” on page 143 focuses on virtual input/output, which includes virtual Ethernet, virtual SCSI, and the Shared Ethernet Adapter.

Part 2 of this book addresses performance tuning and application tuning:

- ▶ Chapter 7 looks at the support provided by the AIX 5L Version 5.3 operating system and the changes to the performance analysis tools.
- ▶ Chapter 8 focuses on application tuning with information about compiler options, profiling, memory management, and optimization techniques. This chapter will benefit those who are involved in benchmarks and providing solutions enablement.
- ▶ Chapter 9 looks at support provided by the Partition Load Manager and its implementation into the virtualization model.



IBM POWER5 architecture

The POWER5 system is the next generation of POWER processor-based microprocessors. It builds on the IBM POWER4 architecture, providing new and improved functional support designed to meet a variety of client needs and requirements.

This chapter provides an in-depth overview of the POWER5 design and discusses various aspects of the functional enhancements that the POWER5 system is designed to support. The chapter is intended to provide you with a look at the POWER5 microprocessor technology. It includes information about the instruction pipelines and the L1, L2, and L3 caches.

2.1 Introduction

The POWER5 processor is the latest 64-bit implementation of the PowerPC® AS architecture (Version 2.02). This dual core processor with *simultaneous multithreading* technology is fabricated using silicon-on-insulator (SOI) devices and copper interconnects. SOI technology is used to reduce the device capacitance and increase transistor performance. Wire resistance is lower in copper interconnects and results in reduced delays in wire-dominated chip timing paths. The chip is implemented using 130 nm lithography with eight metal layers and a die that measures 389 mm². The chip is made up of 276 million transistors.

The primary design objectives of POWER5 technology are:

- ▶ Maintain binary and structural compatibility with existing POWER4 systems
- ▶ Enhance and extend symmetric multiprocessor (SMP) scalability
- ▶ Continue superior performance
- ▶ Provide additional server flexibility
- ▶ Deliver power-efficient design
- ▶ Enhance reliability, availability, and serviceability

The POWER5 microprocessor is downward binary compatible with all PowerPC and PowerPC AS application-level code. The POWER5 has been designed for very high frequency operations with operating frequencies of up to 1.9 GHz. POWER5 consists of a deeply pipelined design with 16 stages for fixed-point register-to-register operations, 18 stages for most load and store operations (with L1 data cache hits), and 21 stages for most floating-point operations.

The processor exhibits a speculative superscalar inner core organization with aggressive branch prediction, out-of-order issues, register renaming, a large number of instructions in flight, and fast selective flush of incorrect speculative fetched instructions and results. There has been a specific focus on storage latency management where the core can issue out-of-order load operations with support for up to eight outstanding L1 data cache line misses. There is hardware-initiated or software-initiated instruction prefetching from the L2, L3, and memory along with hardware-initiated data stream prefetching, and software instruction prefetching based on branch prediction hints.

The POWER5 architecture is an enhancement over the POWER4 architecture, but it maintains binary and structural compatibility. The identical pipeline structure enables compiler optimizations targeted for POWER4 to work equally well on POWER5-based systems.

Each POWER5 processor core is designed to support both *simultaneous multithreading* and single-threaded modes. Software (an operating system using POWER Hypervisor calls) can switch the processor from simultaneous multithreading mode to single-threaded mode. Chapter 3, “Simultaneous

multithreading” on page 41 offers detailed information about simultaneous multithreading.

Figure 2-1 shows the layout of the POWER5 processor.

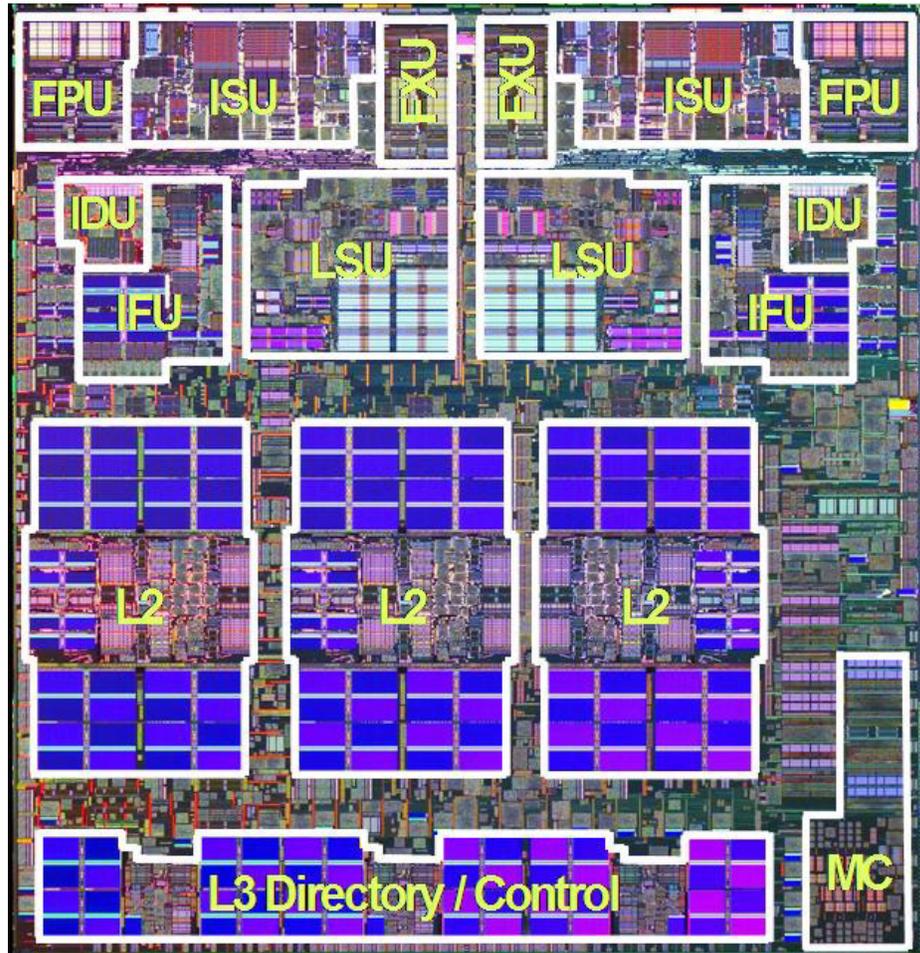


Figure 2-1 POWER5 processor chip

FXU - Fixed-Point (Integer) Unit	FPU - Floating-Point Unit
ISU - Instruction Sequencing Unit	IDU - Instruction Decoding Unit
LSU - Load Store Unit	IFU - Instruction Fetch Unit
L2 - Level 2 Cache	L3 - Level 3 Cache Controller
MC - Memory Controller	

2.2 Chip design

Two identical processor cores are found in a single POWER5 chip. Figure 2-2 shows the high-level layout of a POWER5 processor, including the L3 cache and memory. Because of the dual-core design and support for simultaneous multithreading (two hardware threads per core), a single POWER5 chip appears as a four-way microprocessor system to the operating system.

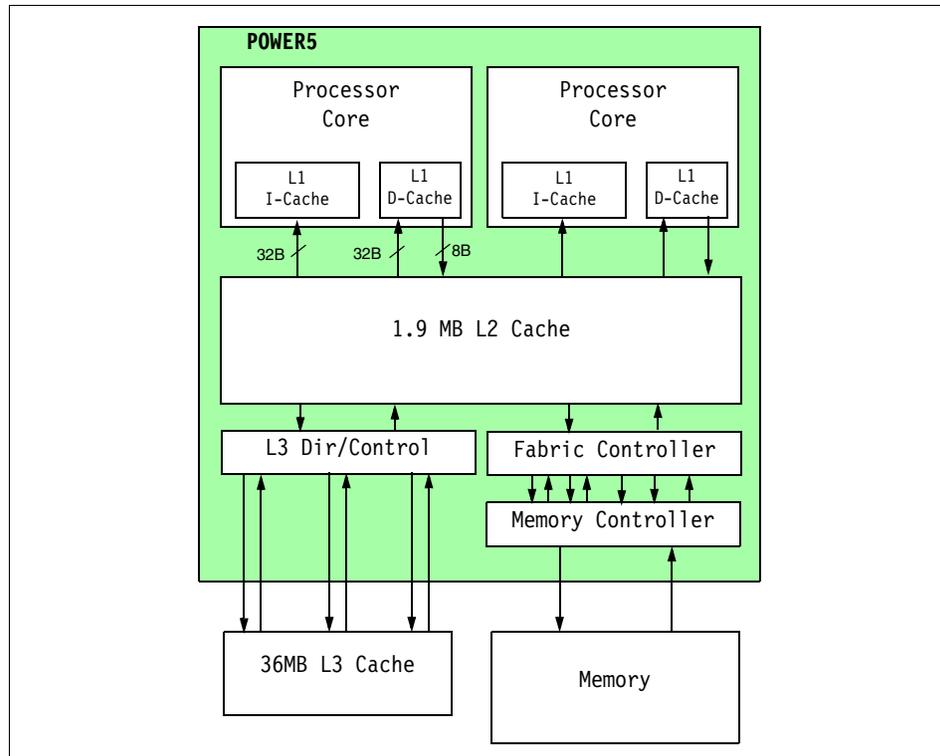


Figure 2-2 High-level structure of POWER5

Simultaneous multithreading is a hardware multithreading¹ technology that can greatly improve the utilization of the processor's hardware resources, resulting in better system performance. Superscalar processors can issue multiple instructions in a single cycle from a single code path (hardware thread), but processors using simultaneous multithreading can issue multiple instructions from multiple code paths (hardware threads) in a single cycle. POWER5 provides for two hardware threads per processor core. Hence, multiple instructions from

¹ The terminology *multithreading* used here refers to the hardware execution of threads provided on a processor core as used in the computer architecture community. It is not same as the software use of the term.

both the hardware threads can be issued in a single processor cycle on the POWER5.

2.3 POWER5 enhancements

Table 2-1 shows a comparison between the POWER4 architecture and the POWER5 architecture. Most of the enhancements were made to accommodate simultaneous multithreading support in the POWER5 processors. (See Chapter 3, “Simultaneous multithreading” on page 41.)

Table 2-1 Differences between POWER4 and POWER5

Unit	POWER4	POWER5
Instruction Fetch Unit (IFU)	Direct-mapped 64 KB Level 1 Instruction Cache	Two-way 64 KB Level 1 Instruction Cache
	4-entry direct mapped prefetch buffer	Split, 2-entry per thread prefetch buffer
	16-entry Branch Information Queue (BIQ)	Split, 8-entry per thread BIQ
	Branch prediction control	Replicated branch prediction control
	Link stack	Replicated link stack
Instruction Decode Unit (IDU)	8-entry Instruction Fetch Buffer (IFB)	6-entry IFB per thread
Instruction Issue Unit (ISU)	20-entry FIFO Global Completion Table (GCT)	20-entry linked list GCT
	80 General Purpose Registers (GPR), 72 Floating-point Registers (FPR) mapper	120 GPR, 120 FPR mapper
	32-entry Condition Register (CR) mapper	40-entry CR mapper
	24-entry Fixed-point Exception Register (XER) mapper	32-entry XER mapper
	20-entry Floating-point Issue Queue (FPQ)	24-entry packed FPQ
Fixed-point Unit (FXU)	80-entry GPR	120-entry GPR

Unit	POWER4	POWER5
Floating-point Unit (FPU)	72-entry FPR	120-entry FPR
Load/Store Unit (LSU)	32 K, two-way set-associative Data Cache	32 K, four-way set-associative Data Cache
	128-entry two-way set-associative Effective to Real Address Translation (ERAT)	128-entry fully associative ERAT
	64-entry Segment Lookaside Buffer (SLB)	Replicated 64-entry SLB per thread
	32-entry Load Reorder Queue (LRQ)	16-entry real and 16-entry virtual LRQ per thread
	32-entry Store Reorder Queue (SRQ)	16-entry real and 16-entry virtual SRQ per thread
	8-entry Load Miss Queue (LMQ)	8-entry LMQ with thread control
	One set of Special Purpose Registers (SPR)	Replicated Special Purpose Registers (SPRs) with thread ID
L2	1.45 MB on chip	1.9 MB on chip
L3	16 MB Cache	36 MB, directory, controller on-chip

The POWER5 architecture provides for two threads of execution in parallel. To do this, some of the processor resources had to be replicated. For example, the 16-entry Branch Information Queue (BIQ) in POWER4 has been split into two 8-entry queues, one per each thread.

2.4 POWER5 instruction pipelines

The POWER5 instruction pipeline can be subdivided into a master pipeline and several different execution pipelines. Figure 2-3 on page 16 depicts the POWER5 instruction master pipeline. Each box in the diagram represents a pipeline stage. The POWER5 pipeline structure is very similar to the POWER4 pipeline structure. Even the pipeline latencies including penalties for mispredicted branches and load-to-use latencies for L1 data cache hits remain the same. This

design lets the compiler optimizations designed for POWER4 to work equally well on POWER5.

The master pipeline presents speculative in-order instructions to the mapping, sequencing and dispatch functions, and ensures an orderly completion of the real execution path. The master pipeline (in-order processing) throws away any potential speculative results associated with mispredicted branches. The execution pipelines allow out-of-order issuing of speculative and non-speculative instructions. The execution unit pipelines progress independently from the master pipeline and each other.

The POWER5 processor consists of the following instruction pipeline features:

- ▶ Deeply pipelined design
 - 16 stages of execution for most fixed-point (integer) register-to-register operations. (IF to CP in Figure 2-3 on page 16)
 - 18 stages for most load and store operations
 - 21 stages for most floating-point operations
- ▶ Out of order issue of up to 8 instructions into 8 execution pipelines
 - Two load or store instruction pipelines
 - Two fixed-point instruction pipelines
 - Two floating-point instruction pipelines
 - One branch instruction pipeline
 - One condition register operation instruction pipeline

2.4.1 Instruction fetching

In simultaneous multithreading, the POWER5 core uses two separate *Instruction Fetch Address Registers* (IFAR) to store the program counter for the two threads from the same program or different programs. Instructions are fetched every alternate cycle for each hardware thread (see IF - instruction fetch stage in Figure 2-3 on page 16). In single-threaded mode, instructions are fetched from the active thread every cycle, and the program counter corresponding to that hardware thread is used. The POWER5 core can fetch an eight-word (32-byte) aligned block of eight instructions per cycle. Keep in mind that all instructions in POWER and PowerPC are 32 bits (one word). The two threads share the instruction cache and the instruction address translation facility (L1 I-cache and I-ERAT). POWER5 also provides a four-entry instruction prefetch queue above the I-cache for hardware initiated prefetching. The first two entries of the instruction prefetch queue are dedicated for thread 0 and the remaining two entries for thread 1 regardless of whether the core is running in simultaneous multithreading or single-threaded mode.

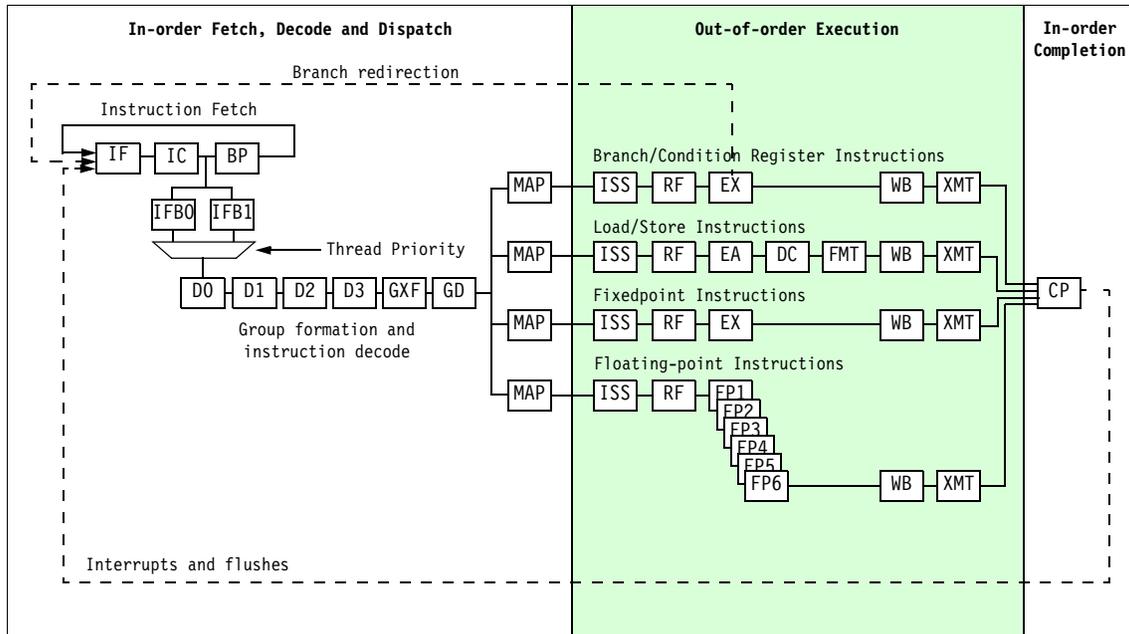


Figure 2-3 POWER5 instruction pipeline

IF - Instruction Fetch	IC - Instruction Cache Access
BP - Branch Prediction	IFB - Instruction Fetch Buffers
D0-D3 - Decode Stages	GXF - Group Transfer
GD - Group Dispatch	MAP - Register Mapping
ISS - Instruction Issue	RF - Register File Access
EX - Execution	EA - Effective Address Generation
DC - Data Cache Access	FMT - Data Formatting
WB - Write Back to Register	FP1 - Floating-point Alignment and Multiply
FP2 - Floating-point Multiply	FP3 - Floating-point Add
FP4 - Floating-point Add	FP5 - Floating-point Normalize Result
FP6 - Floating-point Round Result	XMT - Finish and Transmit
CP - Group Completion	

2.4.2 Branch prediction

The eight fetched instructions are scanned for branch instructions each cycle (BP stage in Figure 2-3). If branch instructions are found, the branch direction is predicted using three *Branch History Tables* (BHT). The tables are shared by the two threads, and two of the tables use bimodal and path-correlated branch prediction mechanisms to predict branches. The third table is used as a selector designed to predict which of these prediction mechanisms is more likely to predict the right instruction path. The BP stage can predict all of the branches at the same time in the fetched instruction group. If the fetched instructions contain

multiple branches, the core logic has the capability to track up to eight outstanding branches per thread in simultaneous multithreading and 16 outstanding branches in single-threaded mode. The core logic also predicts the target of a taken branch in the current cycle's eight instruction group. The target address of most branches is calculated from the instruction's address plus an offset as described by the Power Architecture™ and PowerPC Architecture™. For predicting targets of subroutine returns, the core logic uses a per-thread, eight-entry *Link Stack* (return stack). For predicting targets of the **bcctr** (branch conditional to address in the Count Register) instruction, a 32-entry target cache shared by both the threads is used. If a branch is taken, the core logic loads the program counter with the target address of the branch. If the branch is not predicted as taken, the address of the next sequential instruction (current instruction address + 4) is loaded into the program counter.

2.4.3 Instruction decoding and preprocessing

Instructions in the predicted path from BP stage are placed in the per-thread *Instruction Fetch Buffers* (IFBs). This happens in the D0 stage (see Figure 2-3 on page 16). The core has two 6-entry IFBs, one for each thread. Each IFB entry can hold four instructions. Up to eight instructions can be placed in one of the two IFBs every cycle. Up to five instructions can be taken out from either of the two IFBs every cycle. Based on the thread priorities, instructions from one of the IFBs are selected, split into internal instructions in some cases (*instruction cracking*), and an *instruction group* is formed. This corresponds to the D1 to D3 stages.

Because instructions are later executed out of order, it is necessary to remember the program order of all instructions in flight. Instruction groups are formed to minimize the logic for tracking large numbers of instructions in flight. Groups of these instructions are tracked instead. Care is taken during group formation so that internal instructions that resulted from the cracking of an instruction do not end up in different groups. All instructions in a group belong to the same thread and are decoded in parallel. Each group can have a maximum of five instructions.

2.4.4 Group dispatch

The process of moving the instructions belonging to a group formed in the D0 to D3 stages into the *issue queues* is known as *group dispatch* (GD). Before a group can be dispatched, the processor must ensure that resources required by the instructions in the group are available:

- ▶ Each instruction in the group needs an available entry in an appropriate issue queue.
- ▶ Each load instruction and store instruction needs an entry in the *load reorder queue* and *store reorder queue* respectively to be able to detect out-of-order execution hazards.

- ▶ Each dispatched group needs an available entry in the *Global Completion Table* (GCT). The GCT is used to track the groups of five instructions formed in the D0-D3 stage. The core logic allocates GCT entries in program order for each thread.

When all of the necessary resources are available for the group, the group is dispatched (GD stage). Note in Figure 2-3 on page 16 that the instruction flow from the IF stage to the GD stage happens in program order.

2.4.5 Register renaming

To facilitate out-of-order and parallel execution of instructions in a group, the architected registers (the ones specified in the instruction) are renamed by utilizing a large physical register file provided in the core. Each register that is renamed must have a corresponding physical register. The *Rename Mapper* serves this purpose, and renaming takes place in the MAP stage of the instruction pipeline. Example 2-1 shows a code example where register renaming is needed in this parallel execution environment.

Example 2-1 Register renaming example

```
mulw  r4, r5, r8    ; Multiply contents of GPR 5 to GPR8, result in GPR4
addi  r5, r6, r7    ; Add contents of GPR6 and GPR7, result in GPR5
lwzx  r7, r1, r9    ; Load 32-bit word at address determined by adding
                          ; contents of GPR1 and GPR9 into GPR7
```

In this code example, the three instructions can execute in parallel. Referring to Figure 2-3 on page 16, the `mulw` and `addi` instructions would be issued to the fixed-point instruction pipelines (two fixed-point pipelines in each core) and the `lwzx` instruction would be issued to the load/store pipeline. If GPR5 was not remapped in the `addi` instruction during execution, it could change the source operand for the `mulw` instruction if the `mulw` instruction would stall for some reason. Rename registers are necessary for other situations in a parallel execution (*superscalar*) environment such as supporting precise interrupts. However, these discussions are beyond the scope of this book.

Table 2-2 on page 19 summarizes the rename resources that are available to the POWER5 core. For example, the compiler has 32 GPRs that are available for integer operations in the program. The POWER5 core has a total of 120 registers for renaming. With simultaneous multithreading, both threads can dynamically share the physical register files (rename resources). Instruction-level parallelism exploited for each thread is limited by the physical registers available for each thread. Certain workloads such as scientific applications exhibit high instruction-level parallelism. To exploit instruction-level parallelism of such applications, the

POWER5 makes all of the physical registers available to a single thread in single-threaded mode, enabling higher-instruction level parallelism.

Table 2-2 *Rename resources in the POWER5 core*

Resource type	Available to each thread	Physically in the core
GPRs	32 (36 ^a)	120
FPRs	32	120
XER	4 fields ^b	32
CTR	2	16
LR	2	16
CR	8 (9 ^c) 4-bit fields	40
FPSCR	1	20

a. The POWER5 architecture uses four extra scratch registers known as eGPRs and one additional 4-bit CR field known as eCR for instruction cracking and grouping. These are not the architected registers and are not available for the programming environment.

b. The XER has four mappable fields and one non-mappable field per thread.

c. Eight CR fields plus one non-architected eCR field for instruction cracking and grouping.

2.4.6 Instruction execution

After the MAP stage, instructions enter the issue queues shared by the two threads. These issue queues feed the execution pipelines.

Each POWER5 processor core contains:

- ▶ Two fixed-point (integer) execution pipelines²
 - Both capable of basic arithmetic, logical, and shifting operations
 - Both capable of multiplies
 - One capable of divides and the other capable of Special Purpose Register (SPR) operations
- ▶ Two 6-stage load/store execution pipelines
- ▶ Two 9-stage floating-point execution pipeline (6-stage execution)
 - Both capable of the full set of floating-point instructions
 - All data formats supported in hardware including IEEE 754

² Figure 2-3 on page 16 does not illustrate the number of execution units. See Figure 2-4 on page 21 instead.

- ▶ One branch execution pipeline
- ▶ One condition register logical pipeline

The following instruction issue queues are built into the POWER5 core:

- ▶ Combined, two 18-entry issue queues to feed the fixed-point and load/store execution pipelines
- ▶ Two 12-entry issue queues to feed the floating-point execution pipelines
- ▶ One 12-entry issue queue for branch execution pipeline
- ▶ One 10-entry issue queue for condition register logical execution pipeline

In summary, each POWER5 processor core has eight execution units, each of which can issue instructions out of order, with bias toward the oldest instructions first. Each execution unit can issue an instruction each cycle and complete an instruction every cycle. Keep in mind that the total latency of an instruction depends on the number and nature of each pipeline stage of execution.

Instructions in the issue queue become eligible for issue when all of the input operands for that instruction become available. The issue logic selects an eligible instruction from the issue queue and issues it (ISS stage). While in simultaneous multithreading mode, the issue logic does not differentiate between instructions from the two threads. Therefore, instructions from either of the threads can be issued at any given time, simultaneous to the execution units, thus making the core truly simultaneous multi-threaded. Upon issue of an instruction, the source operand registers for that instruction are read (RF stage), executed on the proper execution unit (EX stage), and results written back to the target register (WB stage). In each load/store unit (LSU), an adder is used to compute the effective address to read from (load) or write to (store) in the EA stage, and the data cache is subsequently accessed in the DC stage. For load instructions, when data is returned from the data cache, a formatter selects the correct bytes from the cache line (FMT stage) and writes them to the register (WB stage).

When all of the instructions in a group have executed without generating an exception and the group is the oldest of a given thread, the group is completed (CP stage). Completion is when the results are moved from the temporary rename registers into the registers that are specified in the program. The processor core can complete two groups per cycle, one from each thread. The GCT entry allocated to the group during the GD stage is deallocated when the group is committed. Each POWER5 processor core has a 20-entry GCT shared by the two threads. Figure 2-4 on page 21 provides some additional detail.

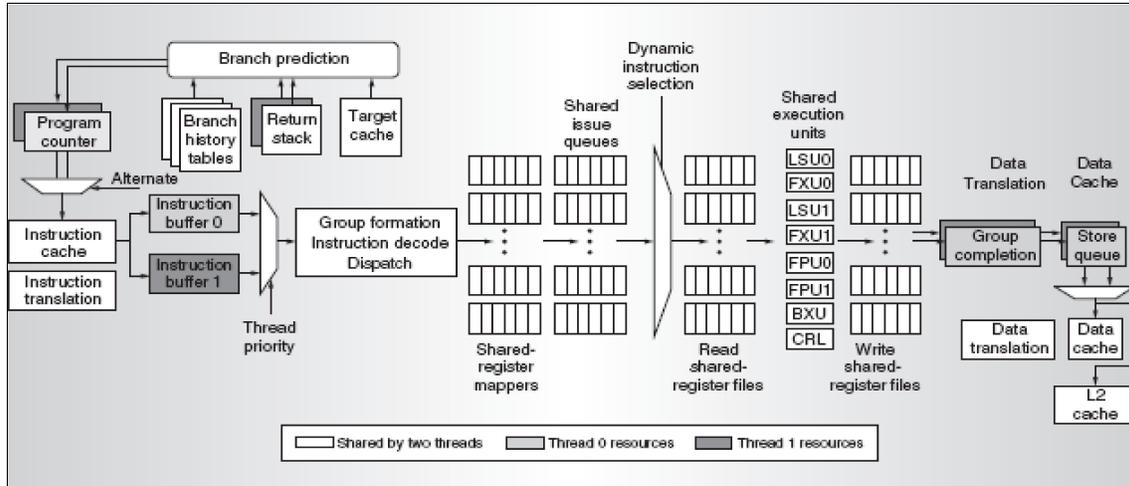


Figure 2-4 POWER5 instruction and data flow

2.5 Caches

The POWER5 microprocessor contains up to three levels of cache hierarchy. The level 1 (L1) cache employs a Harvard cache organization in which instructions are separate from the data. The L1 instruction cache is 64 KB and the L1 data cache is 32 KB in size. The L1 caches are private to each processor core. The cache line size for both caches is 128 bytes. In simultaneous multithreading, the caches are shared by the two hardware threads running in the core.

Both processor cores in a chip share a 1.9 MB unified level 2 (L2) cache. The processor chip houses a level 3 (L3) cache controller that provides for an L3 cache directory on the chip. However, the L3 cache itself is on a separate Merged Logic DRAM (MLD) cache chip. The L3 is a 36 MB victim cache of the L2 cache. The L3 cache is shared by the two processor cores in the POWER5 chip. Needless to say, the L2 and L3 caches are shared by all of the hardware threads of both processor cores on the chip. Table 2-3 on page 30 lists the cache characteristics of the POWER5 processor architecture.

L1 instruction cache

The 64 KB L1 instruction cache is two-way set associative cache for instructions of programs running in the core. This cache uses a *least recently used* (LRU) replacement policy and is indexed using 15 bits of the effective address, as shown in Figure 2-5 on page 24. The 15 bits consist of a 7-bit byte offset and an 8-bit set number. Two-way set associative refers to the fact that there are two address comparators per each set. For any address in memory, the item being

referenced has a predetermined set membership in the cache. Instructions are not fetched on an individual basis from memory as this would be very inefficient. An instruction cache line is fetched from memory. The size of a cache line in the instruction cache is 128 bytes (32 instructions) and is aligned on a 128-byte boundary. Therefore, as shown in Figure 2-5 on page 24, the right-most seven bits (bits 25:31) of the effective address represent the byte offset within the cache block. Based on bits 17 to 24 of the effective address, the line that comes from memory is placed into one of the two ways for the specified set number based on the LRU replacement policy.

When instructions are fetched from the cache, the address of the requested instruction is compared to the addresses in each of the two ways at the specified set number. If the address matches one of the ways, then it is considered to be a *cache hit* and the requested instructions are returned to the fetch unit. If a *cache miss* occurs, the instructions have to be obtained from one of the other memories in the hierarchy.

For both the instruction cache and the data cache the bus width to the L2 cache is 256 bits (32 bytes). Therefore it takes a minimum of four “beats” or cycles to transfer the 128-byte cache line. Each 32-byte beat is referred to as a *sector*.

When a cache miss occurs for instruction fetch, instructions will be returned from the L2 cache if they are present there; otherwise they will come from L3 or system memory. When the instructions arrive, they will take a bypass path so that the instructions will be sent to the fetch unit as quickly as possible. A state machine is used when the bypass path is being used and writes whatever sectors that have arrived into the prefetch buffer. If the instructions are arriving from the L2, then all the sectors will be written. If the data is not from the L2, then it will arrive at a later time and, after the last sector arrives, the state machine will be initiated again to write all of the sectors into the instruction cache.

The instruction cache is single ported, enabling either a read or write operation to occur. Writes to the instruction cache occur in a cycle when the instruction cache cannot be read. The state machine uses a pattern of fetching to try to reduce the impact of these cycles. For example, instead of writing on four consecutive cycles, the state machine spreads the writes out such that each thread does not miss consecutive fetch opportunities. (Each thread is allocated every other cycle of the instruction cache so that if a write occurs on cycle n it will not occur on cycle $n+2$). Because the instruction cache fetch unit can fetch more instructions then it can execute, the performance impact of these writes is very small.

L1 data cache

The 32 KB L1 data cache is a four-way set associative cache for data used by programs running in the core. This cache uses a *least-recently used* (LRU) replacement policy and is indexed using 13 bits of the effective address as shown

in Figure 2-5 on page 24. The 13 bits consist of a 7-bit byte offset and a 6-bit set number. *Four-way set associative* refers to the fact that there are four address comparators per each set. For any address in memory, the item being referenced has a predetermined set membership in the cache. Data is not typically fetched on an individual basis from memory as this would be very inefficient. Instead, a data cache line is fetched from memory. The size of a data cache line is identical to an instruction cache line of 128 bytes and is aligned on a 128-byte boundary. Therefore, as shown in Figure 2-5 on page 24, the rightmost seven bits (bits 25 to 31) of the effective address represent the byte offset within the cache block. Based on bits 19 to 24 of the effective address, the line that arrives from memory is placed into one of the four ways for the specified set number based on the LRU replacement policy.

When data is loaded from or stored into the cache, the address of the data item is compared to the addresses in each of the four ways at the specified set number. If the address matches one of the ways, then it is considered to be a *cache hit* and the data is either returned from (load operation) or written to (store operation) the data cache. If a *cache miss* occurs, the data will have to be obtained from one of the other memories in the hierarchy.

The data cache is a *write-through* cache and therefore never holds modified data. When a store occurs to an existing L1 data cache line, the L1 data cache line is updated as well as a write to the L2 cache using an independent 8-byte (64-bit) data bus.

The L1 data cache provides two read ports and one write port to the core. On a cache miss, data is returned on the L2 cache interface in four 32-byte beats. Like instruction cache misses, the L2 always returns the “critical sector” (the sector containing the specific data item address that referenced the cache line) in the first beat, and the *load miss queue* (LMQ) forwards these load requests into the pipeline as quickly as possible. This is called *critical data forwarding* (CDF). As each 32-byte beat is received it is written to the cache. When all four 32-byte beats are received, the data cache directory is updated.

L1 instruction cache

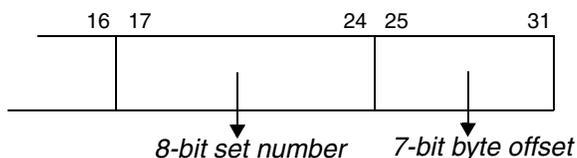
The L1 instruction cache is 64 KB in size and is two way set-associative.

Two way set-associative means there are two cache lines managed per set.

Number of set in the cache can be determined by:

$$64KB / (128 \text{ bytes per block} * 2 \text{ ways}) = 256$$

Effective address determines location in cache.



	Way 0	Way 1
Set 0	cache line	cache line
Set 1		
Set 2		
Set 3		
Set 4		
Set 5		
.		
.		
.		
Set 252		
Set 253		
Set 254		
Set 255	128 bytes	128 bytes

L1 data cache

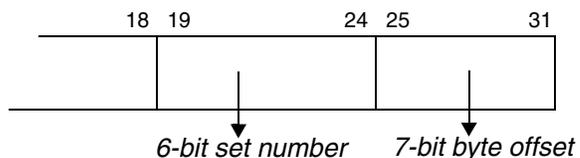
The L1 data cache is 32 KB in size and is four way set-associative.

Four way set-associative means there are four cache lines managed per set.

Number of set in the cache can be determined by:

$$32KB / (128 \text{ bytes per block} * 4 \text{ ways}) = 64$$

Effective address determines location in cache.



	Way 0	Way 1	Way 2	Way 3
Set 0				
Set 1				
Set 2				
Set 3				
Set 4				
Set 5				
.				
.				
.				
Set 60				
Set 61				
Set 62				
Set 63	128B	128B	128B	128B

Figure 2-5 L1 caches

2.5.1 Level 2 (L2) cache

The L2 is a unified cache (contains both instructions and data) shared by both cores on the POWER5 chip. In addition, it maintains full hardware memory coherency within the system and can supply modified data to the cores on other POWER5 processors and I/O devices. Logically, the L2 is an in-line cache. Unlike the L1 data cache, which is write-through, it is a copy-back (store-in) cache. A copy-back cache will not propagate changes to the next levels in the memory hierarchy such as L3 and system memory. By doing this, bus traffic is kept to a minimum and avoids bottlenecks due to memory contention. The L2 cache will respond to other processors and I/O devices requesting any modified data that it currently has.

The L2 cache is fully inclusive of the L1 instruction and data caches located in the two processor cores on one POWER5 chip.

The L2 is a total of 1.9 MB and is physically partitioned into three symmetrical *slices* with each slice holding 640 KB of instructions or data. As shown in Figure 2-7 on page 26, each slice is comprised of 512 associative sets. Each set contains ten 128-byte cache lines. Each of the slices has a separate L2 cache controller. Either processor core of the chip can independently access each L2 controller. The correct slice is determined by a hashing algorithm involving bits 36 to 55 of the physical address, as shown in Figure 2-6 on page 26.

When the slice has been determined, the indexing of the cache by the L2 controller is performed using the address bits as shown in Figure 2-6 on page 26. Using the address of either the requested instruction or data, bits 57 through 63 are used to represent the byte offset within the cache line. Address bits 48 through 56 are used to select the congruence class. A physical tag comparison (that is, real address bits 14 through 47) is used to determine if the desired cache line is resident within one of the 10 ways for that congruence class.

Each slice has a castout/intervention/push bus (16 bytes wide) to the fabric controller and operates at half the core frequency. *Error Correction Control* (ECC) provides single-bit error recovery. To aid performance, eight 64-byte-wide store queues are provided per slice supporting simultaneous multithreading. To minimize bus contention, *store gathering* is also supported. Store gathering is a performance enhancement that is used when storing to non-cachable memory areas such as memory-mapped I/O. When stores are to contiguous memory, the individual stores by the program are gathered into one bus operation, instead of a complete bus transaction for each individual store.

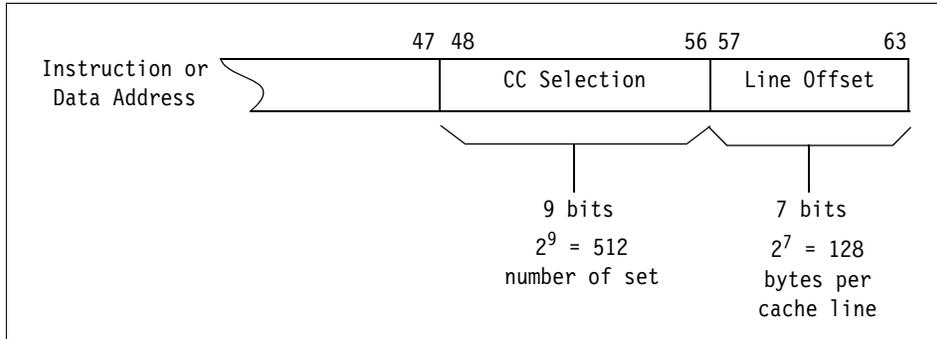


Figure 2-6 Cache indexing bits

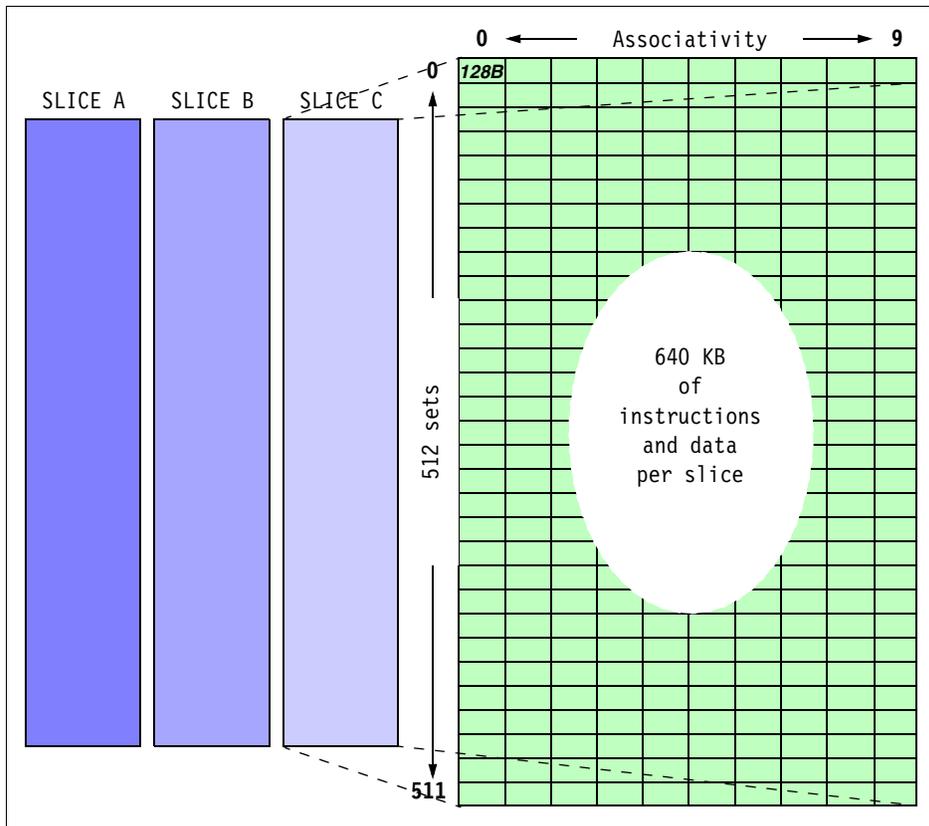


Figure 2-7 L2 cache organization

2.5.2 Level 3 (L3) cache

The L3 cache is a unified 36 MB cache accessed by both cores on the POWER5 processor chip. It maintains full memory coherency with the system and can supply intervention data to cores on other POWER5 processor chips. The L3 is a *victim cache* and is not inclusive of the L2. This means that the same cache line will never reside in both caches simultaneously and a valid, modified cache line cast out from the L2 due to being least-recently used is written into the L3 cache associated by its set number.

This cache is implemented off-chip as a separate *Merged Logic DRAM* (MLD) cache chip. However, the L3 cache directory and control is on the POWER5 processor chip itself. Having the L3 directory on the processor chip itself helps the processor check the directory after an L2 miss without experiencing off-chip delays. Figure 2-8 shows a high-level diagram of this design.

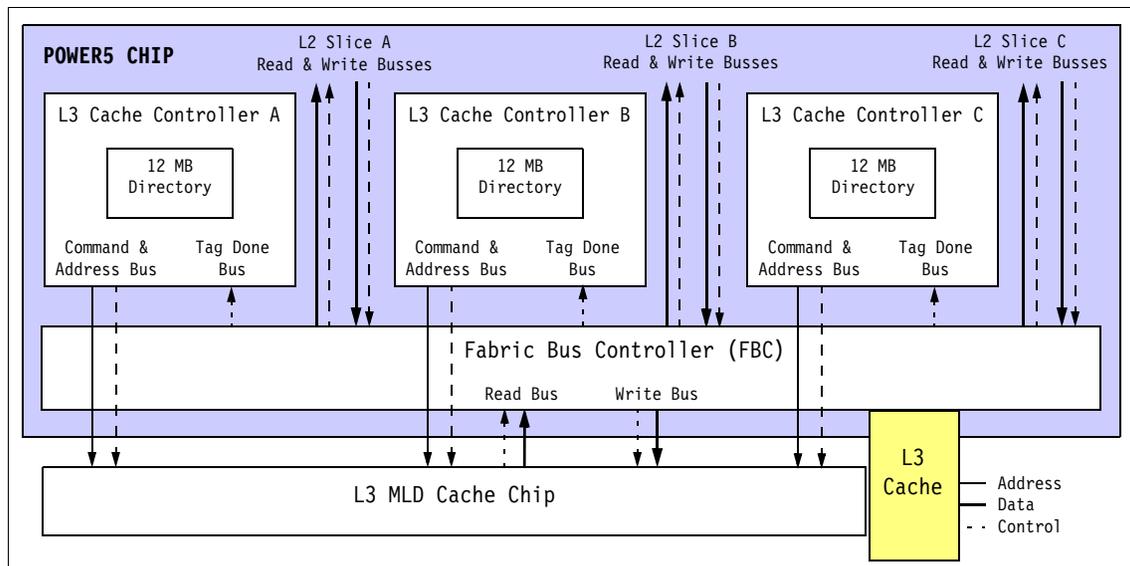


Figure 2-8 L3 cache high-level design

The cache is split into three identical 12 MB slices on the cache chip. The same hashing algorithm for selecting the L2 slices is used to select the L3 slices for a given physical address. A slice is 12-way set-associative. There are 4096 sets that are two-way sectored (which means that the directory manages two 128-byte cache lines per entry). Each of the 12 MB slices can be accessed concurrently. Figure 2-9 on page 28 depicts the L3 cache organization graphically.

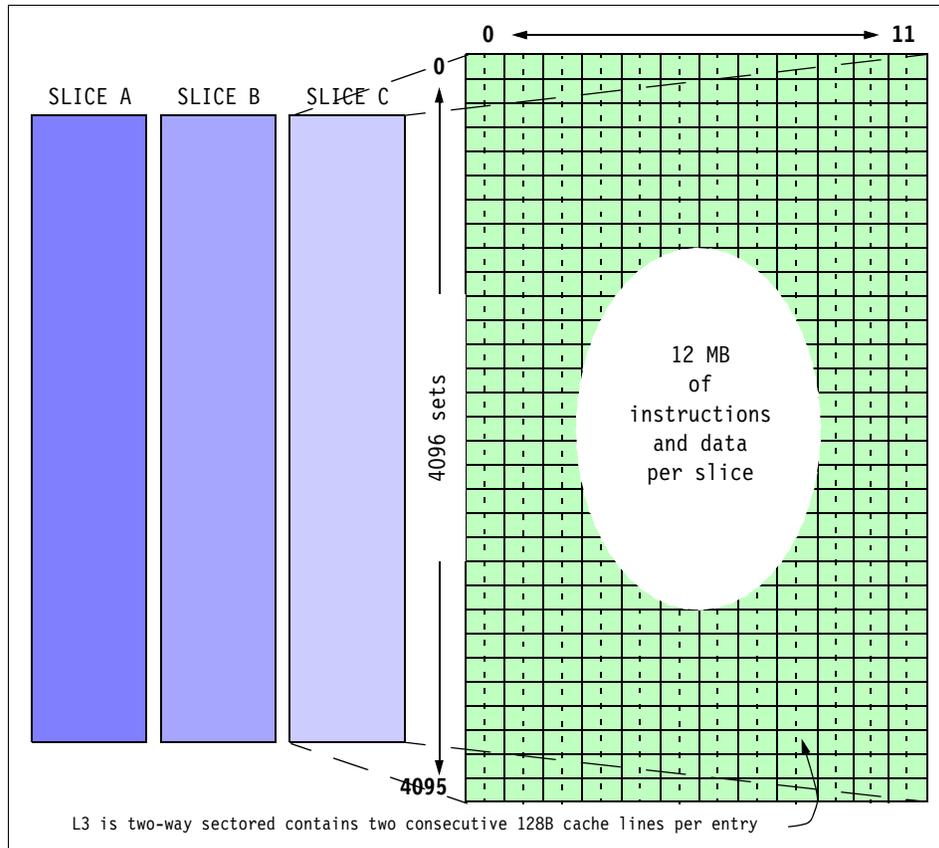
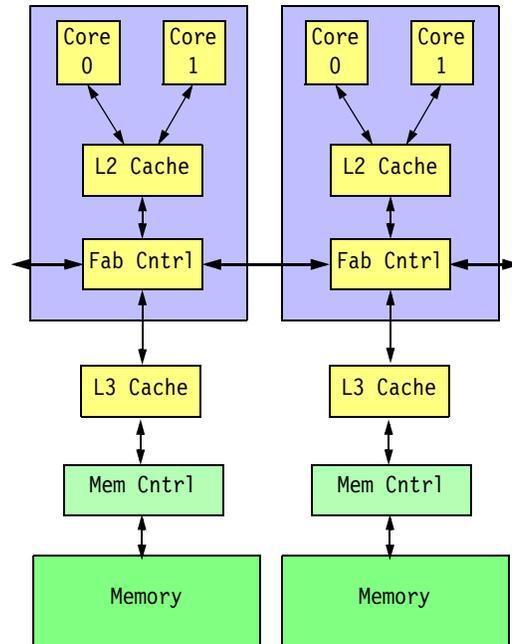


Figure 2-9 L3 cache organization

Unlike in the POWER4 microprocessors, the L3 cache is on the processor side and not on the memory side of the fabric. This is depicted in Figure 2-10 on page 29. This design lets the POWER5 satisfy L2 cache misses more efficiently with hits on the off-chip cache, thus avoiding traffic on the interchip fabric. References to data not on the L2 cause the system to check the L3 cache before sending requests onto the interchip fabric. The L3 operates as a back door with separate 128-bit (16-byte) data busses for reads and writes that operate at one-half the processor speed. Because of higher transistor density of the POWER5 fabrication technology, the memory controller has now been moved onto the chip, eliminating the need for a separate memory controller chip as in POWER4 systems. These architectural changes to the POWER5 processor have the significant benefits of reducing latency to the L3 and main memory as well as the number of chips necessary to build a system. The result is a higher level of SMP scaling. Initial POWER5 systems support 64 physical processors.

POWER4 Systems



POWER5 Systems

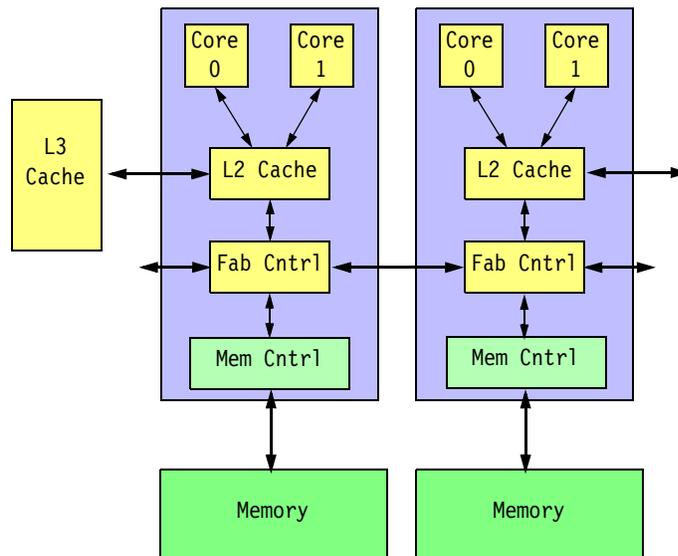


Figure 2-10 Comparison between POWER4 and POWER5

2.5.3 Summary of caches on POWER5

Table 2-3 Cache characteristics of the POWER5 processor

Cache characteristics	L1 instruction cache	L1 data cache	L2 cache	L3 cache
Contents	Instructions only	Data only	Instructions and data	Instructions and data
Size	64 KB	32 KB	1.9 MB	36 MB
Associativity	two-way	Four-way	10-way	12-way
Replacement Policy	LRU	LRU	LRU	LRU
Line size	128 B	128 B	128 B	256 B
Indexed by	Effective address	Effective address	Physical address	Physical address
Tags	Physical address	Physical address	Physical address	Physical address
Inclusivity	N/A	N/A	Inclusive of L1 instruction and data caches	<i>Not</i> inclusive of L2 cache (victim cache of L2)
Hardware Coherency	Yes	Yes	Yes (separate snoop ports)	Yes (separate snoop ports)
Store policy	N/A	Write-through No allocate on store miss	Copy-back Allocate on store miss	Copy-back

2.5.4 Address translation resources

The POWER5 chip supports translation from a 64-bit *effective address* (EA) to a 65-bit *virtual address* (VA) and then to a 50-bit *real address* (RA). The processor architecture specifies a *Translation Lookaside Buffer* (TLB) and a *Segment Lookaside Buffer* (SLB) to translate the effective address used by software to a real address (physical address) used by the hardware. Each processor core contains a unified, 1024 entry, four-way set associative TLB. The TLB is a cache of recently accessed page table entries that describe the pages of memory.

There are two effective-to-real address translation (ERATs) caches. They are called the I-ERAT (for instruction address translation) and the D-ERAT (for data address translation).

The I-ERAT is a 128-entry, two-way set associative translation cache that uses a FIFO-based replacement algorithm. In this algorithm, one bit is kept per congruence class and is used to indicate which of the two entries was loaded first. As the name implies, the first entry loaded is the first entry targeted for replacement when a new entry has to be loaded into that congruence class.

Each entry in the I-ERAT provides translation for a 4 KB block of storage. In the event that a particular section of storage is actually mapped by a large page TLB entry (16 MB), each referenced 4 KB block of that large page will occupy an entry in the I-ERAT (that is, large page translation is not directly supported in the I-ERAT).

The D-ERAT is a 128-entry, fully associative translation cache that uses a binary LRU replacement algorithm. As with the I-ERAT, the D-ERAT provides address translations for 4 KB and 16 MB pages of storage.

2.6 Timing facilities

The Time Base, Decrementer, and the POWER Hypervisor Decrementer provide timing functions for the system. The `mftb` instruction is used to read the Time Base; the `mtspr` and `mfspr` instructions are used to write the Time Base and Decrementers and to read the Decrementers.

Time Base (TB) The Time Base provides a long-period counter driven at 1/8 the processor clock frequency.

Decrementer (DEC) The Decrementer, a counter that is updated at the same rate as the Time Base, provides a means of signaling an interrupt after a specified amount of time has elapsed unless the Decrementer is altered by software in the interim, or the Time Base update frequency changes.

POWER Hypervisor Decrementer

The POWER Hypervisor Decrementer (HDEC) provides a means for the POWER Hypervisor to manage timing functions independently of the Decrementer, which is managed by virtual partitions. Similar to the Decrementer, the HDEC is a counter that is updated at the same rate as the Time Base, and it provides a means of signaling an interrupt after a specified amount of time has elapsed. Software must have POWER Hypervisor privilege to update the HDEC.

Time Base

The Time Base (TB in Figure 2-11) is a 64-bit register and contains a 64-bit unsigned integer that is incremented by one every eight processor clock cycles, as shown in Figure 2-11. Each increment adds 1 to the low-order bit (bit 63). The Time Base increments until its value becomes 0xFFFF_FFFF_FFFF_FFFF ($2^{64} - 1$). At the next increment, its value becomes 0x0000_0000_0000_0000. There is no interrupt or other indication when this occurs.

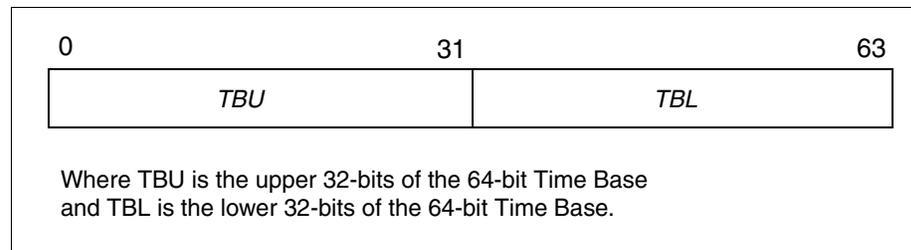


Figure 2-11 Time Base register

If we consider the IBM @server p5 570 model with 1.65 GHz processors, we can determine the time base, as shown in Example 2-2.

Example 2-2 Calculating the Time Base period

$$TBP = \frac{2^{64} \times 8}{1.65 \text{ GHz}} = 8.94 \times 10^{10} \text{ seconds or approx. 2,836 years}$$

Decrementer

The Decrementer (DEC) is a 32-bit decrementing counter that provides a mechanism for causing a Decrementer interrupt after a programmable delay. The contents of the Decrementer are treated as a signed integer. The Decrementer is driven by the same frequency as the Time Base. The period of the Decrementer depends on the driving frequency, but if the same values are used as given above for the Time Base and if the Time Base update frequency is constant, the period would be as shown in Example 2-3.

Example 2-3 Calculating the Decrementer period

$$DP = \frac{2^{32} \times 8}{1.65 \text{ GHz}} = 20.82 \text{ seconds}$$

Whenever bit 0 (most significant bit) of the Decrementer changes from 0 to 1, an interrupt request is signaled. If multiple Decrementer interrupt requests are received before the first can be reported, only one interrupt is reported. The

occurrence of a Decrementer interrupt cancels the request. If the Decrementer is altered by software and the contents of bit 0 are changed from 0 to 1, an interrupt request is signaled.

POWER Hypervisor Decrementer

The POWER Hypervisor Decrementer (HDEC) is a 32-bit decrementing counter and POWER Hypervisor resource that provides a mechanism for causing a POWER Hypervisor decrementer interrupt after a programmable delay. The contents of the Decrementer are treated as a signed integer.

The HDEC is driven by the same frequency as the Time Base. The period of the HDEC will depend on the driving frequency, but if the same values are used as given above for the Time Base and if the Time Base update frequency is constant, the period would be as shown in Example 2-4.

Example 2-4 Calculating the POWER Hypervisor decrementer

$$\text{HDEC} = \frac{2^{32} \times 8}{1.65 \text{ GHz}} = 20.82 \text{ seconds}$$

2.7 Dynamic power management

Chip power is a very important and limiting factor in modern processor designs. A nice side benefit of *complementary metal oxide semiconductor* (CMOS) technology is that if the logic is not clocking, there is no switching of the gates, and if there is no switching, there is negligible power consumption. To reduce power consumption, POWER5 chips use a fine-grained dynamic clock-gating mechanism to gate off clocks to a local clock buffer, if the dynamic power management logic knows that the set of latches driven by that clock buffer will not be used in the next cycle. For example, if the floating-point registers will not be read on the next cycle, the dynamic power management logic detects it and turns off the clocks to the read ports of the floating-point registers. A minimum amount of logic implements the clock gating function. Special care has been taken to ensure clock gating logic does not cause performance loss or create a critical timing path for the chip.

While in simultaneous multithreading mode, the number of instructions executed per cycle goes up, thus increasing the chip's total power consumption. In addition to power consumption, leakage of power has become a performance limiter. POWER5 uses transistors with low threshold voltage only in critical paths such as floating-point register read ports. Figure 2-12 on page 34 shows photographs taken with thermal sensitive cameras on prototype POWER5 chips, with and

without dynamic power management, and single-threaded versus simultaneous multithreading. From the picture, it is evident that dynamic power management reduces power consumption below the standard single-threaded level without power management enabled.

POWER5 also provides for the software environment to control low-power modes. When the thread priority is set to *low priority*, the POWER5 dispatches instructions every 32 cycles, thus saving power. Thread priorities are discussed in 3.3.2, “Adjustable thread priorities” on page 50.

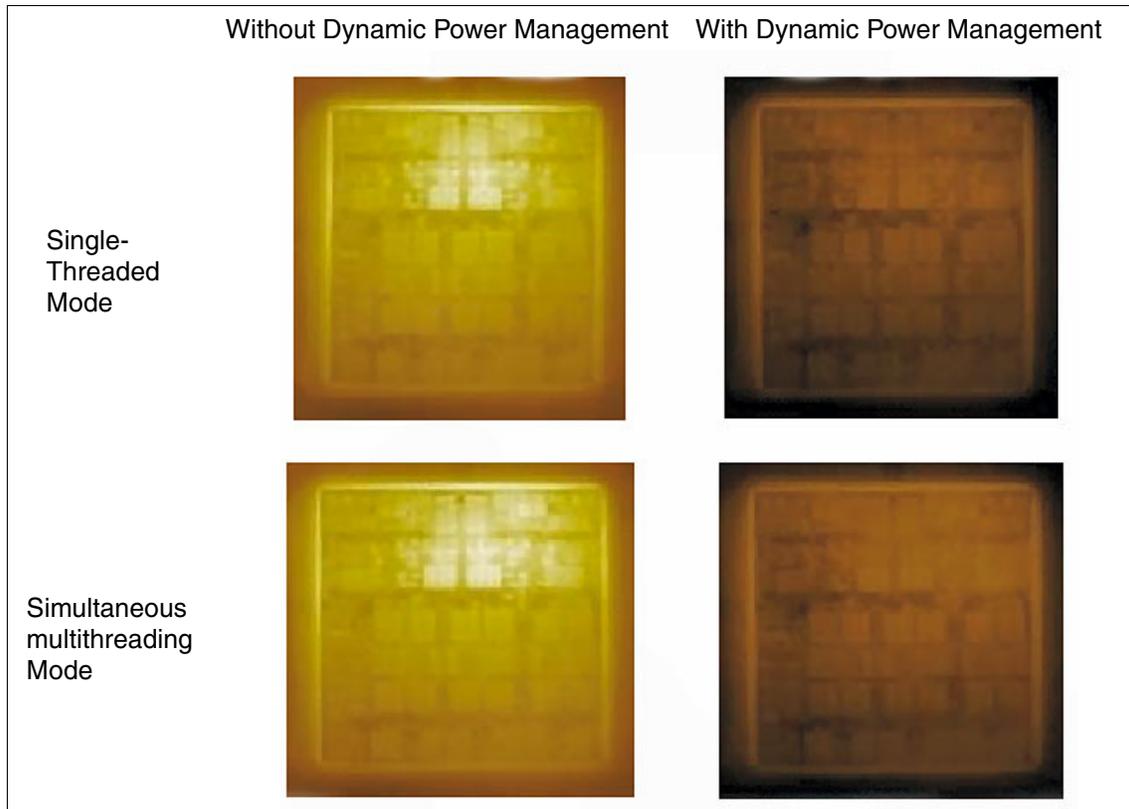


Figure 2-12 POWER5 photos using thermal-sensitive camera

2.8 Processor Utilization Resource Register (PURR)

Previously, a local timer tick (10 ms in AIX 5L, 1 ms in Linux with HZ=1000) was charged to the current running process that was preempted by the timer interrupt. If the process was executing code in the kernel via a system call, the entire tick was charged to the process’s system time. If the process was

executing application code, the entire tick was charged to the process's user time. Otherwise, if the current running process was the operating system's idle process, the tick was charged in a separate variable. UNIX commands such as **iostat** and **vmstat** show these as %usr, %sys, and %idle. Through the outputs, it was possible to determine the utilization of the processor. The problem with this method is that the process receiving the tick most likely has not run for the entire timer period and, unfortunately, was executing when the timer expired. The issue becomes more complicated using simultaneous multithreading as threads from perhaps two different processes share the physical processor resources.

To address these issues and to provide more accurate details of processor utilization, the POWER5 architecture introduces a Processor Utilization Resource Register. This is a special-purpose register that can be read or written by the POWER Hypervisor but is read-only by the operating system (supervisor mode). There are two registers, one for each hardware thread. As with the timebase register, it increments by one every eight processor clock cycles when the processor is in single-threaded mode. When the processor is in simultaneous multithreading mode, the thread that dispatches a group of instructions in a cycle will increment the counter by 1/8 in that cycle. In no group dispatch occurs in a given cycle, both threads increment their PURR by 1/16. Over a period of time, the sum of the two PURR registers when running in simultaneous multithreading mode should be very close but not greater than the number of timebase ticks.

AIX 5L Version 5.3 uses the PURR for process accounting. Instead of charging the entire 10 ms clock tick to the interrupted process as before, processes are charged based on the PURR delta for the hardware thread since the last interval, which is an approximation of the computing resource that the thread actually received. This makes for a more accurate accounting of processor time in the simultaneous multithreading environment.

For example, in simultaneous multithreading mode, the operating system sees the two hardware threads as two separate processors, and dispatches two separate tasks (processes), one on each logical processor. If the old method of charging the current running thread a tick every 10 ms, each logical processor reports a utilization of 100%, representing the portion of time that the logical processor was busy. Using the PURR method, each logical processor reports a utilization of 50%, representing the proportion of physical processor resources that it used, assuming equal distribution of physical processor resources to both the hardware threads.

2.9 Large POWER5 SMPs

Somewhat like the POWER4, the POWER5 uses *Dual Chip Modules (DCMs)*³ and *Multi-Chip Modules (MCMs)* as the basic building blocks for low-/mid-range and high-end servers respectively.

Figure 2-13 depicts a POWER5 DCM, and an actual POWER5 DCM is shown in Figure 2-14 on page 37. The chips in POWER5 are designed to support multiple system configurations ranging from a low-end uniprocessor up through a 64-way (with MCMs).

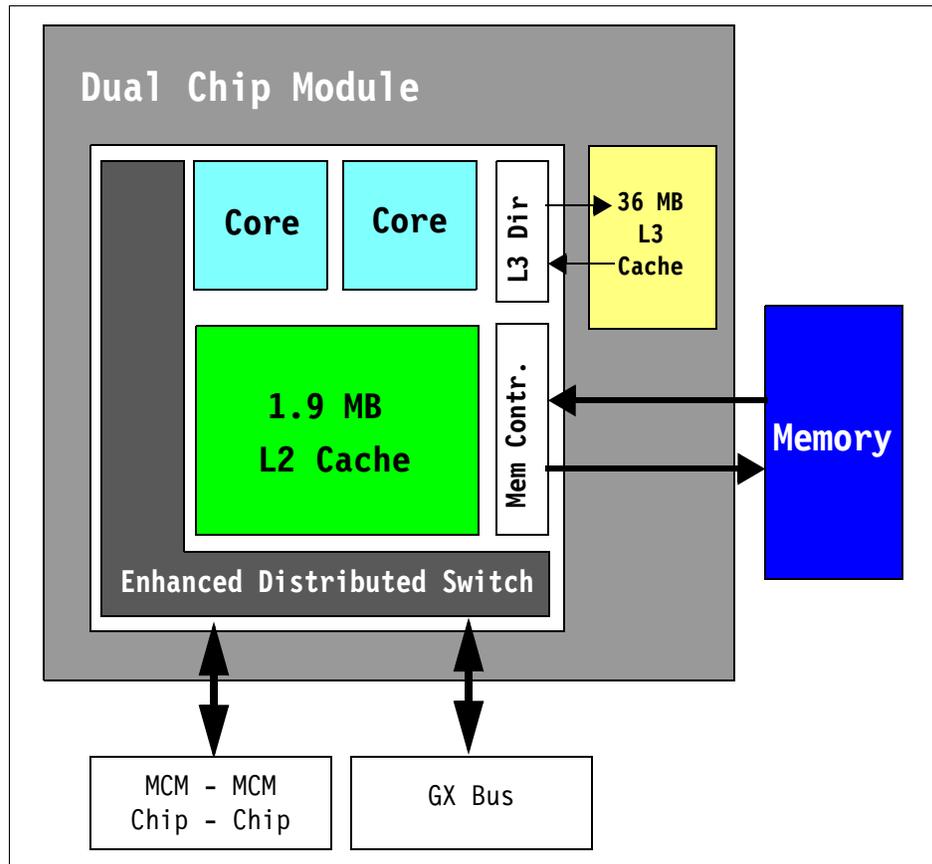


Figure 2-13 POWER5 Dual Chip Module (DCM)

³ DCM has one POWER5 chip and one L3 MLD cache chip, hence the name *dual* chip module. The DCM has only one POWER5 chip with two cores.

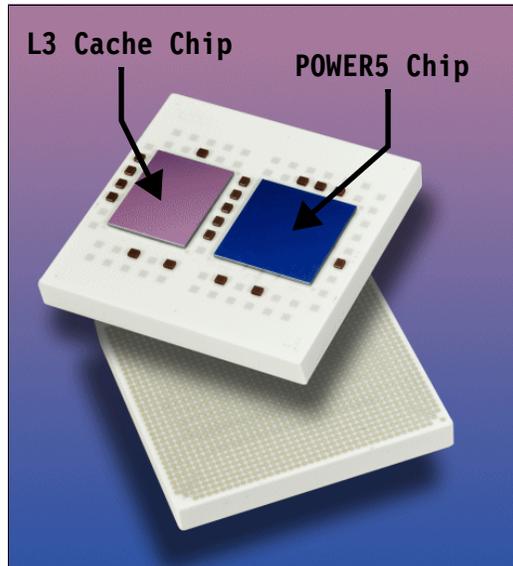


Figure 2-14 Actual DCM

As with the POWER4, POWER5 exploits the enhanced distributed switch for interconnects. All chip interconnects operate at half the processor frequency and scale with processor frequency.

Figure 2-15 on page 38 depicts the logical view of a POWER5 MCM. MCMs are used as basic building blocks on high-end SMPs. MCMs have four POWER5 chips and four L3 cache chips each. Each MCM is an eight-way building block. Figure 2-16 on page 38 shows an actual picture of a POWER5 MCM.

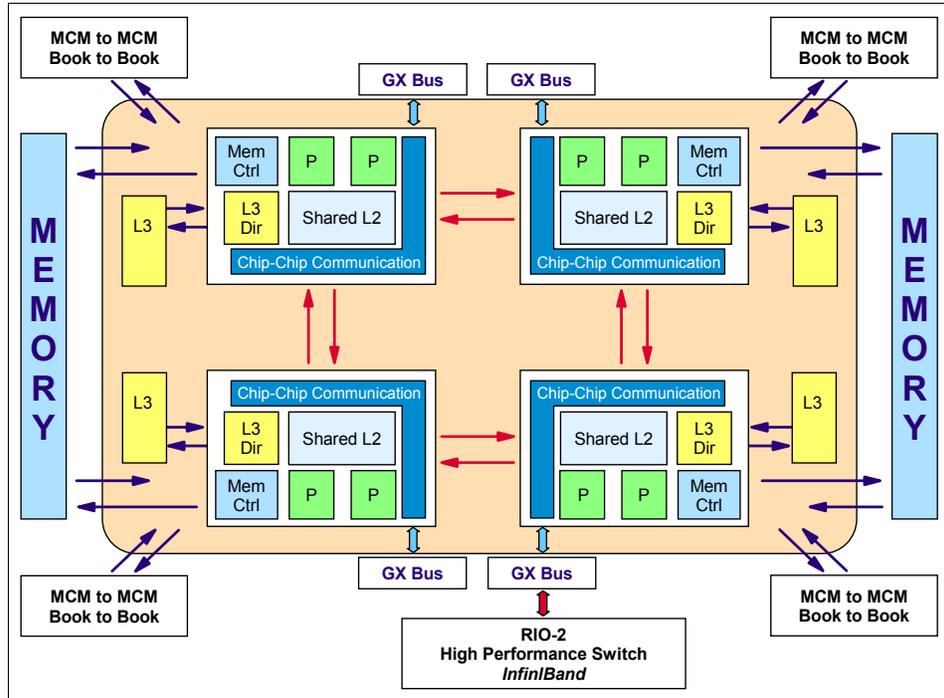


Figure 2-15 Logical view of the POWER5 multi-chip module

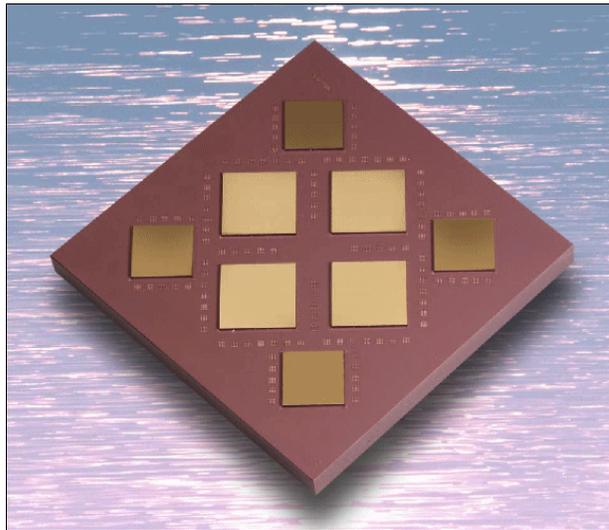


Figure 2-16 POWER5 multi-chip module

Two POWER5 MCMs can be tightly coupled to form a book, as shown in Figure 2-17. These books are interconnected again to form larger SMPs, up to 64-way. The MCMs and books can be interconnected to form eight-way, 16-way, 32-way, 48-way, and 64-way SMPs with one, two, four, six, and eight MCMs respectively.

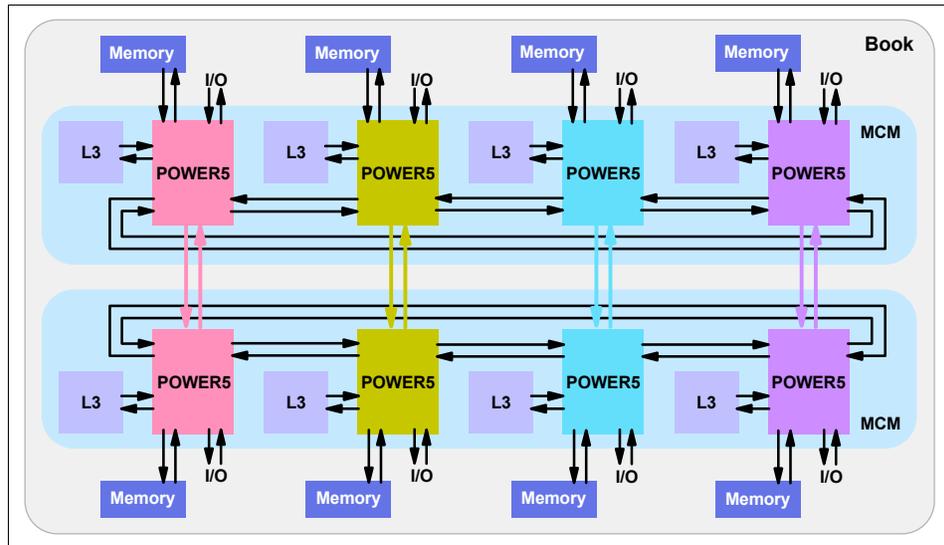


Figure 2-17 16-way POWER5 building block

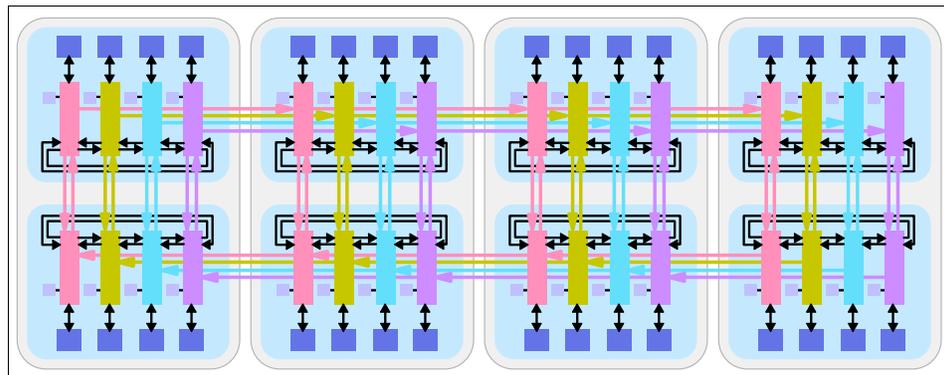


Figure 2-18 64-way POWER5 SMP interconnection

2.10 Summary

POWER5 processor-based systems provide excellent flexibility and performance. Many of the features that enable flexibility and performance challenge existing notions of how systems *look and feel*. IBM has already invested in ensuring that software can exploit the increased performance levels POWER5 systems will be offering, and is continuing in its pursuit to produce system-level enhancements to provide even greater performance increases over time.



Simultaneous multithreading

A very-high-frequency processor can spend more than half of its execution time waiting for cache and TLB misses. Given the trend for advances in processor cycle time and performance to increase faster than DRAM performance, it is expected that memory access delays will make up an increasing proportion of processor cycles per instruction. This is often referred to as the *memory wall*. One technique for tolerating memory latency that has been known for several years is *multithreading*. There are several different forms of multithreading. A traditional form called *fine grain multithreading* keeps N threads, or states, in the processor and interleaves the threads on a cycle-by-cycle basis. This eliminates all pipeline dependencies if N is large enough that instructions from the same thread are separated by a sufficient number of cycles in the execution pipelines.

The form of multithreading implemented in the POWER5 architecture is called *simultaneous multithreading* and is a hardware design enhancement that enables two separate instruction streams (threads) to execute simultaneously on the processor. It combines the multiple instruction-issue capabilities of superscalar processors with the latency-addressing capabilities of multithreading.

3.1 What is multithreading?

In general, the evolution of multithreading can be broadly divided into:

- ▶ Single threading
- ▶ Coarse grain threading
- ▶ Fine grain threading
- ▶ Simultaneous multithreading

Figure 3-1 provides an overview of these four types of multithreading. Each box in the diagram represents an execution stage in the respective instruction pipeline. The acronyms provided on the left of each block represent the fixed-point execution (FX) units, the load store (LS) units, the float-point (FP) units, the branch execution (BRX) units, and the condition register logical execution unit (CRL).

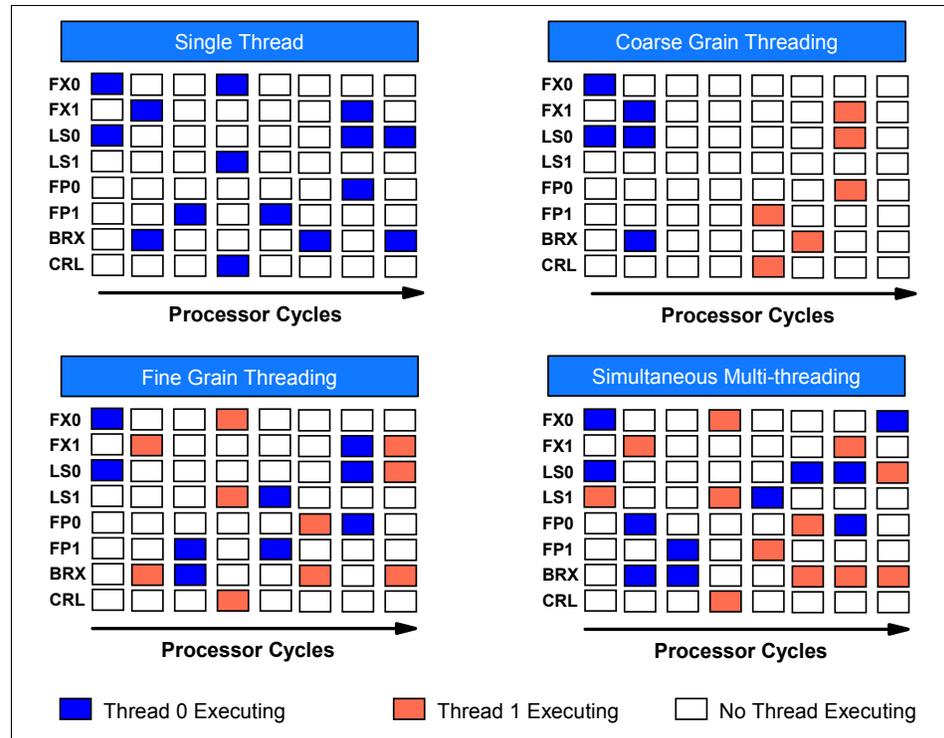


Figure 3-1 multithreading techniques

In single-threaded mode, we see a thread executing two instructions per cycle. Note that in the single-threaded mode, just two execution units (FX0 and load/store unit 0 (LS0)) are utilized in the first cycle (vertical column). In this mode, execution unit utilization is dependent on instruction-level parallelism

produced by the compiler or assembly language programmer to take advantage of the eight instruction pipelines (FX0 - CRL) in this *superscalar* processor.

The IBM STAR series of processors utilized in RS/6000 Model S85 servers used a hardware multithreading technique called *course grain threading* that, when enabled, enabled multiple threads to run in parallel. In coarse grain threading, one thread known as the *active thread* executes on the processor while the other threads are *dormant*. If the active thread experiences a long latency event such as a cache miss, the processor places the active thread into the dormant state and switches to one of the other dormant threads waiting on the processor. For such threading mechanisms to work efficiently, the latency of switching from one thread to the other must be shorter than the latency of the event (servicing of a cache miss) that caused the switch. For example in Figure 3-1 on page 42, a branch instruction (subroutine call) executed in the BRX unit causes an instruction cache miss. While the instructions are being fetched from memory, another thread that was dormant is allowed to execute and its instructions start in the FP1 and CRL execution units. However, as processor pipelines become more complex, efficiency of thread switching is diminished.

Fine-grain threading is a hardware multithreading technique in which threads take turns every processor clock cycle executing their instructions. While fine grain threaded processors tolerate long latency operations better and utilize the execution units better, all instruction pipelines may not be utilized. Therefore, similar to single threaded processors, efficiency of fine grained threaded processors is also limited by the instruction level parallelism.

In a *simultaneous multi-threaded* processor, the processor fetches instructions from more than one thread. Since instructions from any of the threads can be fetched by the processor in a given cycle, the processor is no longer limited by the instruction level parallelism of the individual threads. What differentiates this implementation is its ability to schedule instructions for execution from all threads concurrently. With simultaneous multithreading, the system dynamically adjusts to the environment, enabling instructions to execute from each thread if possible, and allowing instructions from one thread to utilize all of the execution units if the other thread encounters a long latency event. For instance, when one of the threads has a cache miss, the second thread can continue to execute.

More information about simultaneous multithreading may be found in the following references:

- ▶ *Simultaneous Multi-threading: Maximizing On-Chip Parallelism*, 22nd International Symposium on Computer Architecture
- ▶ *Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor*, 23rd Annual International Symposium on Computer Architecture

3.2 POWER5 simultaneous multithreading features

The POWER5 simultaneous multithreading implementation is a natural extension to the eight instruction pipeline superscalar POWER4 design. When in simultaneous multithreading mode, instructions from either thread can use the eight instruction pipelines in a given clock cycle. By duplicating portions of logic in the instruction pipeline and increasing the capacity of the register rename pool, the POWER5 processor can execute two instruction streams, or threads, concurrently. The POWER5 also features dynamic resource balancing (DRB) and adjustable thread priorities for efficient utilization of the resources shared by both threads. Through hardware and software thread prioritization, greater utilization of the hardware resources can be realized without an impact to application performance. Figure 3-2 on page 45 illustrates the increased processor resource utilization using simultaneous multithreading in POWER5 compared with POWER4. Notice the increased utilization of the instruction pipelines, shown by the shaded boxes. Processor utilization can also be seen in Figure 2-12 on page 34, which shows the thermal image comparison.

Note: Each POWER5 processor core appears to the operating system as a two-way symmetric multiprocessor (SMP).

Each hardware thread is supported as a separate logical processor by AIX 5L V5.3. So, a dedicated partition that is created with one physical processor is configured by AIX 5L V5.3 as a logical two-way by default. This is independent of the partition type, so a shared partition with two virtual processors is configured by AIX 5L V5.3 as a logical four-way by default. When simultaneous multithreading is disabled, at least half of the logical processors will be offline.

Characteristics of the POWER5 simultaneous multithreading implementation are as follows:

- ▶ Eight priority levels for each thread that can be raised or lowered by the POWER Hypervisor, operating system, or application
- ▶ Processor resources optimized for best simultaneous multithreading performance, providing the ability to reduce priority of a thread that is consuming maximum resources or hold decode of a thread with long latency events
- ▶ Dynamic feedback of shared resources, enabling balanced thread execution
- ▶ Software-controlled thread priority
- ▶ Dynamic thread switching capabilities

IBM has estimated the performance benefit of simultaneous multithreading at 30% for commercial transaction processing workloads. Read more about this at: http://www.ibm.com/servers/eserver/pseries/hardware/system_perf.html

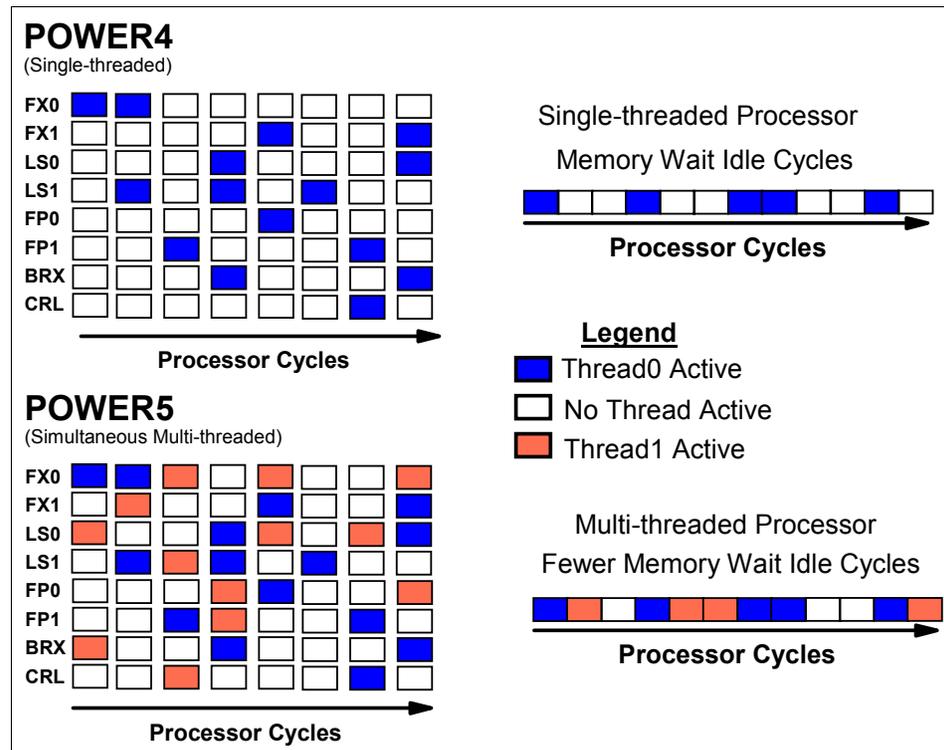


Figure 3-2 Single-threaded versus simultaneous multithreading

In simultaneous multithreading mode, the POWER5 processor uses two separate instruction fetch address registers (IFARs) to store the program counters for the two threads. Instruction fetches (IF stage) alternate between the two threads. Up to eight instructions can be fetched from the instruction cache (IC stage) and placed into one of the two instruction fetch buffers every cycle. Up to five instructions can be taken out of the instruction fetch buffer per cycle for execution. The two threads share the instruction cache and the instruction translation facility. In a given cycle, all fetched instructions are unique for each thread.

Important: Not all applications benefit from simultaneous multithreading.

Not all applications benefit from simultaneous multithreading. Having two threads executing on the same processor will not increase the performance of

applications with execution-unit-limited performance or applications that consume all of the processor's memory bandwidth. For this reason, the POWER5 supports *single-threaded* execution mode. In this mode, the POWER5 gives all physical resources to the *active thread*, enabling it to achieve higher performance than a POWER4 system at equivalent frequencies. In single-threaded mode, the POWER5 uses only one *instruction fetch address register* and fetches instructions for one thread every cycle.

3.2.1 Dynamic switching of thread states

The POWER5 processor provides for the software to dynamically switch from simultaneous multithreading mode to single-threaded mode and vice versa. There are instances when this could be useful, such as real-time applications where guaranteed latency is more important than overall throughput, or scientific applications that are limited by execution resources (for example, when sharing of execution resources will prove counterproductive).

There may also be instances when there are not enough processes ready-to-run on all available hardware threads. For example, in simultaneous multithreading mode, one hardware thread of execution is the operating system's idle process and the other hardware thread is application code. Because the hardware thread of the idle process also needs to map registers from the rename register pool, there may be a performance impact for a task when it is run in simultaneous multithreading mode compared to when it is run in single-threaded mode. In single-threaded mode and as designed, the operating system's idle process would not execute until there were no other processes in the ready-to-run state.

When the POWER5 processor is operating in single-threaded mode, the inactive thread will be in one of two possible states, *dormant* or *null*, as shown in Figure 3-3 on page 47. From a hardware perspective, the only difference between these states is whether the thread awakens on an external or decremented interrupt.

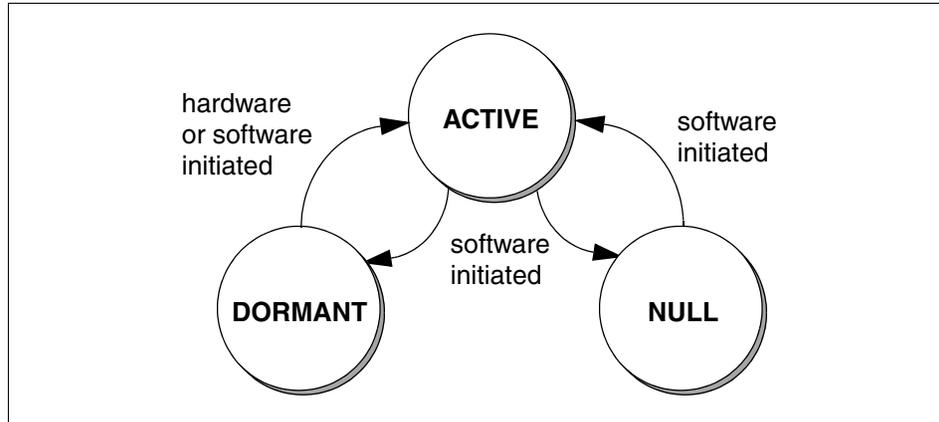


Figure 3-3 POWER5 thread states

When the POWER5 processor is powered on, each core is brought up in single-threaded mode with thread 0 active, and thread 1 is *dormant* by default. To define these three states:

- | | |
|----------------------|--|
| Active state | Thread is active and running as seen by software and hardware. Hardware maintains the architected state of the thread. |
| Dormant state | Thread is inactive in hardware but active in software (processor structures are maintained by software). The POWER5 processor does not make any distinction between dormant and null and behaves the same way. |
| Null state | The hardware thread is inactive in hardware and inactive in software. This is true single-threaded mode. |

As an example, we previously noted that to the AIX 5L V5.3 and Linux operating systems, each POWER5 processor core appears as a two-way (two logical processors) system. In the null state, only one logical processor would exist. In the dormant state, two logical processors would exist to the operating system, but only one physical hardware thread (Thread 0) would be used. The second hardware thread would have to be activated in order to use the second logical processor.

3.2.2 Snooze and snooze delay

In the *dormant* state, the architected register state is not maintained in the hardware, but the software maintains knowledge of the logical processor, such as per-processor data. (The term *software* refers to either the operating system or the POWER Hypervisor.) The processor is set up so that the dormant thread can return to the active state by a decremter or external interrupt.

The process of putting an active thread into a dormant state is known as *snoozing*. If there are not enough tasks available to run on both hardware threads, the operating system's idle process will be selected to run on the available hardware thread. It is better for the operating system to snooze the idle process' thread and switch to single-threaded mode. Doing so enables all of the processor resources to be available to the task doing meaningful work.

To snooze a thread, the operating system will invoke the H_CEDE POWER Hypervisor call (refer to Table 4-1 on page 81). The thread then goes to the dormant state. A snoozed thread is brought alive when a decremter, external interrupt, or an H_PROD POWER Hypervisor call is received. When other tasks become ready to run, the processor transitions from single-threaded mode to simultaneous multithreading mode through any of the means mentioned earlier. This involves the snoozed thread coming to life at the system reset interrupt vector for the thread and having the POWER Hypervisor restore the operating system state, and then returning from the original H_CEDE POWER Hypervisor call made by the thread to snooze. This means several thousand cycles of thread startup latency.

Therefore, it does not make sense to snooze a thread as soon as the idle condition is detected. There could be another thread in the ready-to-run state in the *run queue* by the time the snooze occurs, resulting in wasted cycles due to the thread start-up latency. It is good for performance if the operating system waits for a small amount of time for work to come in before snoozing a thread. This short idle spinning time is known as *simultaneous multithreading snooze delay*. An operating system can optionally make this delay tunable.

Both AIX 5L and Linux incorporate changes to snooze an idle thread. They also provide snooze delay tunables.

When the system is set to operate in single-threaded mode, by use of the **smtctl** AIX 5L command, the inactive thread is put into the null state, and the operating system is unaware of the hardware thread's existence. No system resources are allocated to the second hardware thread. This mode is advantageous if all the system's executing tasks perform better in single-threaded mode.

The AIX 5L V5.3 **smtctl** command, which controls enabling and disabling of simultaneous multithreading mode, provides privileged users and applications with a means to enable or disable simultaneous multithreading for all processors in a partition either immediately or on a subsequent boot of the system.

The two flags associated with **smtctl** are **-m** and **-w**; they are defined as follows:

- m off** Sets simultaneous multithreading mode to disabled
- m on** Sets simultaneous multithreading mode to enabled

- w boot** Makes the simultaneous multithreading mode change effective on the next and subsequent reboots
- w now** Makes the mode change effective immediately, but will not persist across reboot

The `smtctl` command does not rebuild the boot image. To change the default simultaneous multithreading mode of AIX 5L and Linux, the `bosboot` command must be used to rebuild the boot image. The boot image in AIX 5L V5.3 and Linux has been extended to include an indicator that controls the default simultaneous multithreading mode.

3.3 Controlling priority of threads

Because the POWER5 processor core is capable of fetching instructions from two separate instruction paths, contention arises between the two threads for the processor's resources. There are also times when the code executing in the processor is not doing any meaningful work, such as running the operating system's idle process. There is also the case where one thread is currently holding a lock and another thread wants the lock. If a spin-lock is implemented, the thread that holds the lock would be forced to contend with the thread asking for the lock, delaying the release of the lock. In addition, critical sections of code in the operating system or real-time applications must be able to execute with some guaranteed latency. To address these issues, the POWER5 processor provides:

- ▶ Dynamic resource balancing (DRB)
- ▶ Adjustable thread priorities

In this section, we discuss each of these two features of controlling threads in the simultaneous multithreading environment.

3.3.1 Dynamic resource balancing (DRB)

The purpose of this resource is to ensure smooth flow of both threads through the processor. If either of the two hardware threads start dominating the processor resources and depriving the other thread, the DRB logic throttles down the dominating thread so that the other thread can flow smoothly without stalling. For example, if one thread experiences multiple L2 cache misses for loading of data, the dependant load instructions can block in the issue queue slots, preventing the other thread from dispatching instructions. (Refer to the processor pipeline discussion in 2.4, "POWER5 instruction pipelines" on page 14.) To prevent such stalls, the DRB logic monitors the miss queues, and if a particular thread reaches a threshold for L2 cache misses, it throttles that thread down so that the other thread can progress smoothly. Similarly, one thread could start

using too many *Global Completion Table* (GCT) entries, preventing the other thread from dispatching instructions. DRB logic then detects this condition and throttles down the thread dominating the GCT.

Important: DRB is done at the processor level and is not tunable by software.

POWER5 DRB can throttle down a thread in three different ways, with the choice of the throttling mechanism depending on the situation:

1. Reducing the thread's priority.

This is used in situations in which a thread has used more than a predetermined number of GCT entries.

2. Holding the thread from decoding instructions until resource congestion is cleared.

This applies to when the number of L2 misses incurred by a thread reaches a threshold.

3. Flushing all of the dominating thread's instructions waiting for dispatch and holding the thread's decoding unit until congestion clears.

This is used if a long latency instruction such as memory ordering instructions (for example, **sync**) causes dominating of the issue queues.

Studies have shown that higher performance is realized when resources are balanced across the threads using DRB.

3.3.2 Adjustable thread priorities

The DRB logic is built into the hardware to ensure balanced resource utilization by the threads. However, there are instances when software knows that a process running on a hardware thread might not be doing any computational work, such as spinning for a lock or executing the operating system's idle process. The operating system might also want to quickly dispatch a process, such as a process holding a critical spinlock, and needs to elevate its priority. For better utilization of processor resources under such scenarios, the POWER5 features adjustable thread priorities, where software can specify whether the hardware thread running the process can have more or fewer execution resources.

The POWER5 supports the eight levels of thread priorities (0-7) shown in Table 3-1 on page 51. The thread priority is independent of the AIX 5L and Linux thread priorities. Each thread has a 64-bit *thread status register* (TSR) associated with it.

Table 3-1 POWER5 thread priority levels

Thread priority level	Priority level	Privilege level for software to set this priority ^a	Equivalent nop instruction
0	Thread shut-off	POWER Hypervisor Mode ^b	-
1	Very low	Supervisor Mode	or 31,31,31
2	Low	User/Supervisor Mode	or 1,1,1
3	Medium low	User/Supervisor Mode	or 6,6,6
4	Normal	User/Supervisor Mode	or 2,2,2
5	Medium high	Supervisor Mode	or 5,5,5
6	High	Supervisor Mode	or 3,3,3
7	Extra high	POWER Hypervisor Mode	or 7,7,7

a. Certain fields in a thread control register (TCR) affect the privilege level. This column assumes recommended setting and setups, which is usually the case with well-behaved software.

b. The POWER Hypervisor is the highest privilege level followed by supervisor (usually the O/S) and user applications.

Important: The thread priorities mentioned here are independent of the operating system's concept of thread priority.

The POWER5 processor supports three processor states:

- ▶ *POWER Hypervisor mode*: All thread priority values can be set.
- ▶ *Supervisor mode* (AIX 5L or Linux kernel code): Only priority levels one through six can be set.
- ▶ *User mode* (application programs): Restricted to levels two through four.

By default, threads execute at *normal* priority in both kernel mode and user mode.

The priority level can be set in two ways. The thread in the correct mode can execute an `mtspr` instruction to set the three-bit priority field in the thread status register to the desired thread priority. The second method uses the equivalent no-operation (`nop`) instruction. In the POWER and PowerPC architectures, there is no actual `nop` instruction. However, if the `or` instruction is executed with the two source registers and the destination register being the same register, it is considered a `nop`. The POWER5 architecture takes it one step further by providing the ability to control thread priority. Which GPR is used with the `or`

instruction affects the priority of the thread. The last column in Table 3-1 on page 51 shows the equivalent **nop** instructions that set the thread priority.

Thread priority adjustment can be performed in C/C++ code with the use of the **#pragma** compiler directives. Example 3-1 shows how an application programmer can adjust priorities of the application. Keep in mind that the three priorities shown in the code example are the only priorities available to applications running in user mode. The other priorities are reserved for kernel code (supervisor mode) or the POWER Hypervisor.

Example 3-1 C/C++ code example of setting thread priorities

```
void  smt_low_priority(void);          /* The three priorities available to */
void  smt_mediumlow_priority(void);   /* application programs (user mode) */
void  smt_normal_priority(void);

#pragma mc_func smt_low_priority  { "7c210b78" } /* or r1, r1, r1 */
#pragma mc_func smt_medium_priority{ "7cc63378" } /* or r6, r6, r6 */
#pragma mc_func smt_normal_priority{ "7c421378" } /* or r2, r2, r2 */

int main(int argc, char **argv)
{
    .
    .
    smt_low_priority();
    .
    .
    smt_normal_priority();
}
```

3.3.3 Thread priority implementation

When the priority of thread execution is manipulated by software, the effect is to throttle the execution of the lower priority threads. This is done by holding the instructions of the thread in their instruction fetch buffers. As described in 3.3.1, “Dynamic resource balancing (DRB)” on page 49, the lower priority thread is kept from entering the decode stage of the pipeline, thus yielding the decode resources to the higher priority thread.

Most applications will not be concerned with manipulating their priority. However, there may be instances where the application programmer might want to use the priority adjustment for synchronization. For example, your application is either multi-threaded or multi-tasking. Each thread or task processes its own data, but the application as a whole cannot proceed until all threads or tasks are complete. As each thread or task finishes its part of the work, it can lower its priority to enable the others to catch up.

Table 3-2 shows the effect of thread priority on obtaining execution time in the instruction pipeline. If both threads have a priority of 0, the processor is essentially stopped and an I/O or decremter interrupt will be required to resume execution. If thread 0 has the priority of 0 and thread 1 has a priority of 1 (very low priority), then a group of up to five instructions is started every 32 processor clock cycles for thread 1. Having a priority of 1 is really intended for the operating system's idle process and locking mechanism. If one thread holds a lock and the other thread wants the lock, you want the thread that holds the lock to use the processor resources and not have to share cycles with the thread that keeps asking whether the lock is available yet (spin lock). There are other scenarios for using priority manipulation, but the discussion of these scenarios is beyond the scope of this book.

Table 3-2 Effect of thread priorities on decode slot usage

Thread 0 priority (X)	Thread 1 priority (Y)	Decode slots status
0	0	Both Thread 0 and Thread 1 are stopped.
0	1	Thread 1 begins decoding up to five instructions every 32 processor cycles for power savings. Thread 0 is stopped.
0	>1	Thread 1 uses all processor resources and will be fetching and executing instructions every clock cycle.
1	1	Every 64 cycles, each thread will start up to five instructions for power saving.
1	>1	Thread 1 gets all of the execution resources and thread 0 gets any leftover resources. Thread 1 should have the performance similar to single-threaded mode.
>1	>1	How many cycles each thread gets before yielding to the other is determined by the equation $1/(2^{x-y} + 1)$

When both threads have a priority greater than 1, the following equation is used:

$$\frac{1}{2^{(|X - Y| + 1)}}$$

For example, if thread 0 has a priority of 4 and thread 1 has a priority of 2, then thread 1 gets $1/(2^{(4 - 2) + 1}) = 1/8$ cycles or one processor cycle out of every eight. Thread 0 gets the other seven cycles.

Figure 3-4 on page 54 depicts the effects of thread priorities on instructions executed per cycle. The x-axis labels with comma separators represent actual thread priority pairs. For example, 7, 0 implies that thread 0 has a priority of 7 and

thread 1 has been stopped. The numbers without comma separators represent the value of $(X - Y)$ where X is the priority of thread 0 and Y is the priority of thread 1. A value of 5 on the x-axis indicates either (7,2) or (6,1) for X and Y .

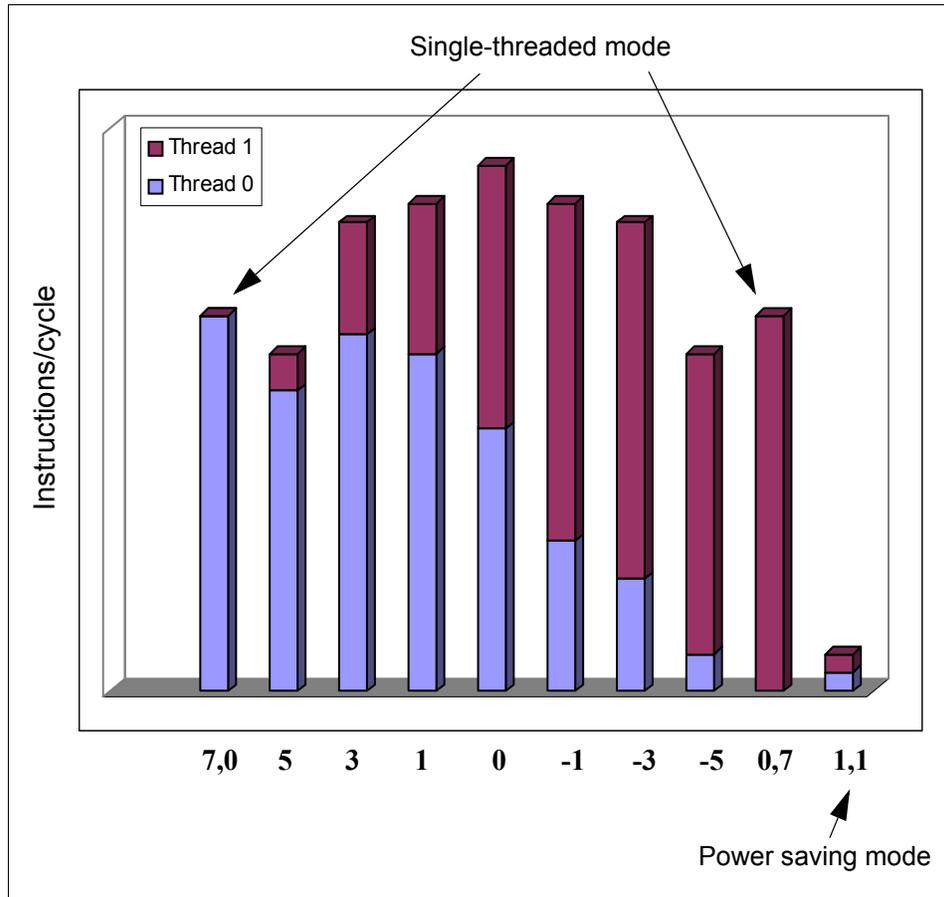


Figure 3-4 Effect of adjusting thread priorities

3.4 Software considerations

The goal of simultaneous multithreading is to increase overall throughput of the system by executing two threads that when run individually on the processor in single-threaded mode may not utilize the processor execution resources to the desired level. Simultaneous multithreading performance depends on the type of application; however, for most cases some general rules can be introduced:

- ▶ Simultaneous multithreading does not speed up individual threads of execution, but overall throughput should improve.
- ▶ If applications care about real-time responses rather than overall system performance, they are better off running in single-threaded mode.
- ▶ For workloads that are limited by the processor execution resources, such as technical workloads that exhibit high instruction level parallelism and consume large amount of rename resources such as floating-point registers (FPRs), simultaneous multithreading will not help much.

Important: The POWER5 provides facilities for the operating system to dynamically switch simultaneous multithreading on and off for applications and workloads that might benefit from simultaneous multithreading.

In general, based on previous work, the following rules can be summarized for application performance on simultaneous multithreading environments:

- ▶ Applications found in commercial environments showed higher simultaneous multithreading gain than scientific applications.
- ▶ Experiments on different workloads have shown varying degrees of simultaneous multithreading gain ranging from -11% to 43%. On average, most of the workloads showed a positive gain running in simultaneous multithreading mode.

Applications that showed a negative simultaneous multithreading gain may be attributed to L2 cache thrashing and increased local latency under simultaneous multithreading.

3.4.1 Simultaneous multithreading aware scheduling

A multi-processor kernel can run on a POWER5 simultaneous multithreading enabled system without modifications, since the kernel will see the two hardware threads as two separate logical processors. For example, on an IBM @server p5 system with two physical processors (four hardware threads) and two ready-to-run processes, the *scheduler* could schedule the processes to run on the two hardware threads of the same processor core, resulting in the other processor core being idle. If the operating system is not simultaneous

multithreading aware, there would be no way for the scheduler to distinguish between threads on the same processor or different processors. Obviously, this does not lead to efficient utilization of system processing capacity. Given this background, the most obvious optimization for simultaneous multithreading is to make sure that work is distributed to all of the primary threads (thread 0) before work is dispatched to secondary threads (thread 1). Secondary threads can be snoozed or put at very low priorities if they are idle. The AIX Version 5L V5.3 and Linux 2.6 kernel are simultaneous multithreading aware.

Note: Both AIX 5L V5.3 and the Linux 2.6 kernel are simultaneous multithreading aware.

Another optimization is to consider the two threads of a core as one affinity (AIX 5L V5.3) or scheduling (Linux) domain, so that the domain reflects sharing of resources such as the translation look-aside buffer (TLB) used to map virtual addresses to real addresses, between the two hardware threads. It might be beneficial for software threads of the same process to run in the same domain so that the shared processor caches (L1, TLB, and so on.) are effectively utilized by the software. It also makes sense to maintain the affinity of software tasks to domains where they ran earlier, so that they get a warmer cache.

The **bindprocessor** command has been enhanced in AIX 5L V5.3 to accept command line options to display all primary threads or all secondary threads. This is to help applications that use binding to bind to one physical processor.

These two optimization techniques are meant to illustrate that simultaneous multithreading awareness helps the operating system perform better. As both the AIX 5L and Linux operating systems evolve, more optimization techniques can be expected.

3.4.2 Thread priorities on AIX 5L V5.3

AIX 5L V5.3 does not lower the priority of the idle thread if simultaneous multithreading is enabled. It searches for work in its own run queue and other run queues for threads with normal priorities. This ensures that any information about available work is current and can be acted on with least latency. If no work is available and if the snooze delay is not over, it will spin in a loop for a tunable number of times in a very low priority loop (for power savings only), checking for work only on its own run queue. After that it returns to top of the idle process and repeats the search for work until snooze delay expires.

Normally, AIX maintains both hardware threads at the same priority but will boost or lower thread priorities in a few key places to optimize performance, and lowers thread priorities when the thread is doing non-productive work such as spinning

in the idle process or on a kernel lock. When a thread is holding a critical kernel lock, AIX boosts the thread's priority. These priority adjustments are made only with code executing in kernel mode (device drivers, system calls, and so on.).

AIX has tunable options (such as the `schedo` command) to enable the boosting of priority for hot locks to *medium-high* (5) priority. There is code in the first level interrupt handlers to reset the priority back to *normal* (4) priority, so the priority boost for hot locks does not boost interrupt handlers and exception handlers. There is a tunable to enable the priority boost to be preserved across the interrupts and to be kept until the job gets dispatched, but that is not the default.

Dedicated partition implementation

For instances where the processor is dedicated to a partition, simultaneous multithreading is enabled, and if there are no ready-to-run tasks, the idle process is started and invokes the POWER Hypervisor's H_CEDE call. AIX will set the priority of the idle process to *low* (2) and wait for the simultaneous multithreading snooze delay to decrement to zero.

If kernel code is waiting for a spinlock, AIX changes the waiting thread's priority to *low* (2). Therefore the spinning thread yields processor resources to the thread holding the lock. Using the formula mentioned in 3.3.3, "Thread priority implementation" on page 52, this would result in the idle thread getting one cycle compared to eight cycles for the thread that holds the lock, assuming normal priority for the thread that holds the lock.

Micro-Partitioning implementation

For instances where the processor is used in a Micro-Partitioning environment and a wait on a spinlock occurs, a wait for a tunable spin delay occurs after setting the priority to *low* (2). After the spin delay, the POWER Hypervisor's H_CONFER call is made only if the task is running with interrupts disabled, perhaps to serialize with interrupt service routines. Since it has interrupts disabled, we cannot dispatch another job (or thread) from the run queue on the processor. The H_CONFER is used to release the hardware thread. If the second hardware thread were also to H_CONFER later on, then the whole processor is freed up and POWER Hypervisor can redispach the physical processor on another partition. Note that this is a requirement because the hardware threads are bound and must run in the same partition. If the task is running with interrupts enabled, the task is placed onto the sleep queue and the dispatcher is called to dispatch another task. The POWER Hypervisor will control priority management and redispach the physical processor if the other hardware thread also cedes or confers.

3.4.3 Thread priorities on Linux

The Linux 2.6 kernel for POWER lowers the thread priority for the idle process. For dedicated LPARs, the priority of the idle process is set to low and then waits for the snooze delay period before snoozing the idle thread by means of the POWER Hypervisor's H_CEDE call. For Linux instances in a Micro-Partitioning environment, Linux always invokes H_CEDE for the idle process.

The Linux kernel also lowers the priority of a thread spinning for a lock to be released. The priority of the waiting thread is set to low (2) and, assuming the other hardware thread in the core has a normal (4) priority, like AIX 5L V.53 the waiting thread will get one processor clock cycle to every eight cycles of the other thread.

3.4.4 Cache effects

With simultaneous multithreading, thread-level parallelism is used to compensate for low instruction level parallelism by having two possibly different tasks share the same processor core and caches. This means there could be more associativity misses in the caches. (See 2.5, "Caches" on page 21.) To compensate for this, POWER5 has increased POWER4's associativity of the L1 instruction cache from direct-mapped (one-way) to two-way set associative, and increased the data cache from two-way to four-way set associative.

The L2 cache on POWER5 is now a 1.9 MB, 10-way set associative cache, compared to the 1.5 MB eight-way set associative L2 cache on POWER4.

The L3 on the POWER5 is 36 MB 12-way set-associative cache compared to the 32 MB eight-way set-associative on POWER4. The L3 cache on the POWER5 is now a victim cache of L2, unlike an inline cache in POWER4. On POWER5, the L3 cache runs at half the processor speed, compared to one-third the processor speed on POWER4. The L3 cache being a victim cache of L2, it behaves like a large (albeit a bit slower) L2 extension. These processor enhancements help offset the cache effects due to simultaneous multithreading, resulting in overall improved application performance.

3.5 Simultaneous multithreading performance

Performance measurements for various standard industrial benchmarks were made with AIX 5L V5.3 on four-way @server p5 570 POWER5 systems to validate gains from simultaneous multithreading. The measurements were made with simultaneous multithreading enabled and disabled.

Figure 3-5 illustrates simultaneous multithreading gains for various workloads for a four-way 1.65 GHz p5 570 POWER5 system. As the chart shows, throughput improvement varies from 10% to 50% depending on the workload.

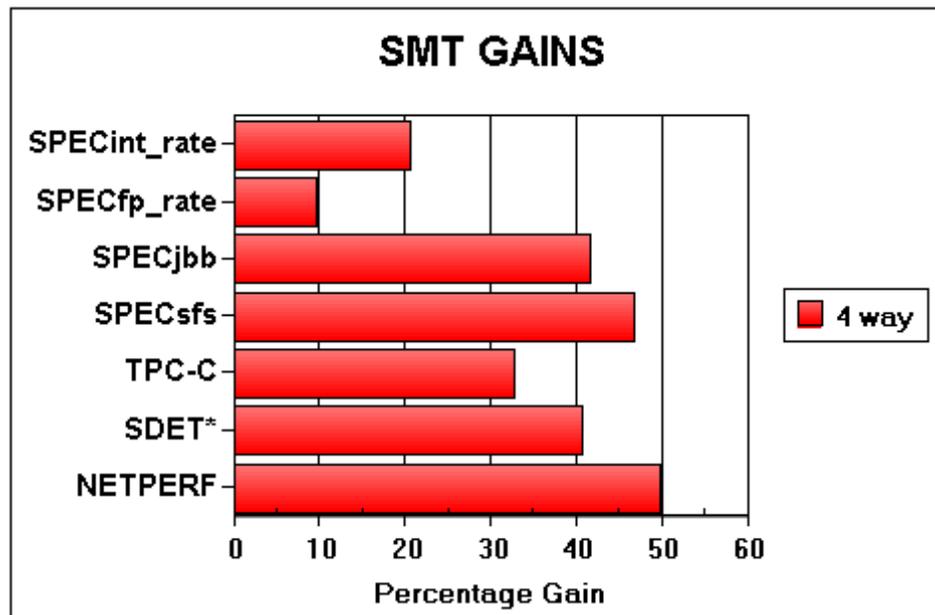


Figure 3-5 Simultaneous multithreading gains for various workloads

3.5.1 Engineering and scientific applications

In this section, we present a series of examples that involve applications in the area of High Performance Computing (HPC). These applications correspond to the Life Sciences and Computer Aided Engineering (CAE) industry:

- ▶ Gaussian03

Gaussian (Gaussian03, Rev.C.01, Gaussian Inc., Wallingford, CT) is a connected series of programs that can be used for performing a variety of electronic structure calculations; molecular mechanics, semi-empirical, ab initio, and density functional theory.

▶ Assisted Model Building with Energy Refinement (AMBER)

AMBER is a flexible suite of programs for performing molecular mechanics and molecular dynamics calculations based on force fields. AMBER is the primary program used for molecular dynamics simulations and is the only program considered in our current study. The version used for our tests correspond to AMBER7 for IBM systems, and the test that was selected to run AMBER is the Joint AMBER-CHARMM (JAC) benchmark.

▶ Basic Local Alignment Search Tool (BLAST)

BLAST is a set of similarity search programs designed to explore all of the available sequence databases regardless of whether the query is protein or nucleic acid. The BLAST programs have been designed for speed, with a minimal sacrifice of sensitivity to distant sequence relationships. The scores assigned in a BLAST search have a well-defined statistical interpretation, making real matches easier to distinguish from random background hits. BLAST uses a heuristic algorithm that seeks local (as opposed to global) alignments and is therefore able to detect relationships among sequences that share only isolated regions of similarity.

▶ FLUENT

FLUENT V6.1.22 (FLUENT, Inc.) is a leading computational fluid dynamics (CFD) application program for modeling fluid flow and heat transfer in complex geometries. FLUENT provides complete mesh flexibility, solving flow problems with unstructured meshes that can be generated about complex geometries with relative ease. CFD applications allow for high parallelization.

Although all of these applications are in the same area of High Performance Computing, the algorithms that are utilized to carry out their simulations are not necessarily the same. This provides a good test for the performance of simultaneous multithreading under different conditions or workloads.

3.5.2 Simultaneous multithreading benchmarks

We ran two sets of test cases for each of these applications: one set for single-threaded mode and one set for simultaneous multithreading.

The system used to conduct these tests was an IBM *@server* p5 570 (9117-570) with four 1.65 GHz processors and 16 GB of memory. The operating system was AIX 5L V5.3. Fortran xlf 9.1 and the xlc 7.0 compilers were installed.

Although this is a benchmarks section, we hope it can also provide basic information for sizing and capacity planning for this type of application. The objective of capacity planning is to provide an estimate of future systems resource requirements based on the present knowledge of the system utilization.

An extensive discussion of sizing and capacity planning independent from scientific applications can be found in the redbook *IBM @server pSeries Sizing and Capacity Planning*, SG24-7071.

Gaussian03 benchmark tests

The first benchmark test corresponds to Gaussian03. For both single-threaded and simultaneous multithreading modes, we ran our test case 1-way (sequential), and two-way, four-way, and eight-way (parallel). In other words, within the Gaussian notation we ran with `nproc` using one, two, four, and eight processors. It is important to note that the system only had four physical processors.

These benchmarks are important because they show that there is almost no difference running single-threaded versus simultaneous multithreading when utilizing parallel jobs with the total number of physical processors or less. We also show that parallel jobs running two times the number of physical processors, when running on simultaneous multithreading mode, still show additional scalability; that does not happen when running in single-threaded mode.

Figure 3-6 on page 62 illustrates the performance of Gaussian03 using multiple processors under single-threaded and simultaneous multithreading modes. In this figure we can identify three trends:

- ▶ The first trend corresponds to the performance when running with one and two processors. In this case we see that when running Gaussian03 under any of these two modes, the performance is basically identical (less than 1% difference).
- ▶ The second trend may be observed when running with four processors. In this case we see that when running in single-threaded mode there is a slight advantage in performance as there is no sharing of processor resources with another hardware thread. The percentage difference is approximately 4%.

- ▶ The last trend may be seen when requesting a run with eight processors. Clearly, this case is requesting more than the physical number of processors available on this machine. However, in simultaneous multithreading mode (abbreviated as SMT in the following figures), the two hardware threads appear as two logical processors. Because of the more efficient use of the processor, we see more than a 40% improvement in performance when compared to running in single-threaded mode.

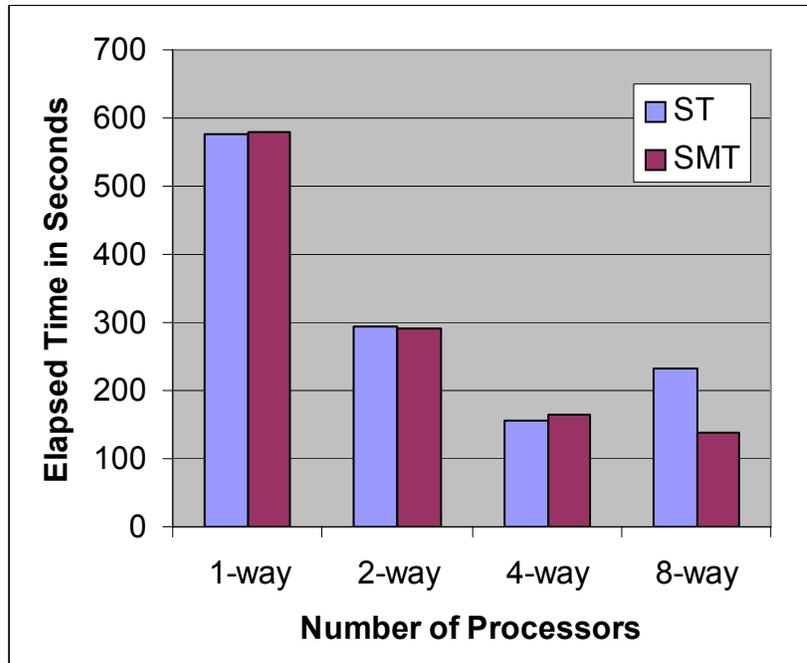


Figure 3-6 Gaussian03 benchmarks

Next we look at a performance comparison of the two modes, single-threaded and simultaneous multithreading, in a series of throughput benchmarks. We ran the throughput benchmarks by carrying out a single calculation on a standalone system; this was the only process running. We refer to this scenario as a single job. When the job was done, we simultaneously submitted two jobs, three jobs, and on to eight simultaneous jobs. Figure 3-7 shows the performance of Gaussian03 with a series of throughput benchmarks using single-threaded and simultaneous multithreading modes.

As in the first set of tests, we can identify two trends. The first trend corresponds to the throughput benchmarks consisted of one, two, and three simultaneous jobs, where there is basically no difference in performance (less than 1%) between single-threaded and simultaneous multithreading.

The second trend begins when the number of processors is the same as the number of jobs submitted. In our case where there is four physical processors, we start seeing the benefit of the simultaneous multithreading mode. Clearly the simultaneous multithreading mode outperforms the single-threaded mode. As the number of simultaneous jobs is increased the effect becomes more dramatic.

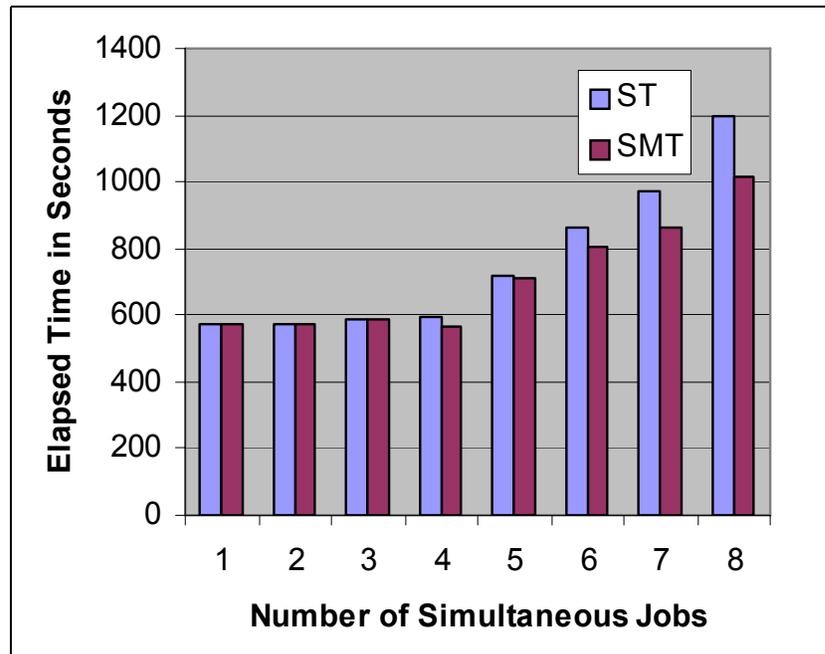


Figure 3-7 Throughput comparison of Gaussian03 tests

The difference between single-threaded and simultaneous multithreading can be seen in Figure 3-8. This figure clearly shows that from one to three simultaneous jobs, there is not much difference between single-threaded and simultaneous multithreading. However, from four to eight simultaneous jobs, the advantage of simultaneous multithreading is clear.

The difference that we see in the case of five jobs running simultaneously, compared to four and six simultaneous jobs, might be caused by the operating system running daemons and kernel processes in the background; therefore for a certain period of time they both were competing for resources. Explicitly binding to processor may alleviate this behavior.

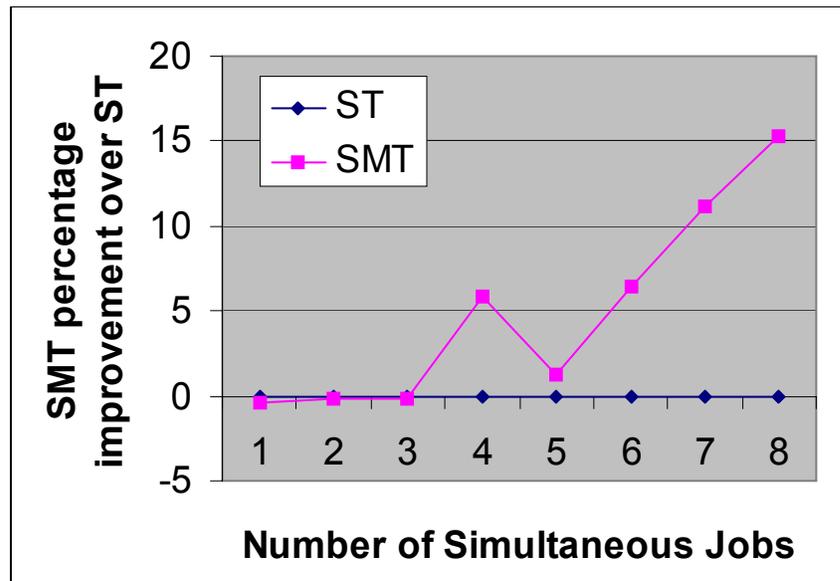


Figure 3-8 Performance advantage of the simultaneous multithreading

AMBER7 benchmark tests

The second application that we tested was AMBER7. Figure 3-9 shows results similar to the case of the Gaussian03 benchmark tests. Again we observe exactly the same three trends. For the first trend, we see that there is no difference between single-threaded mode and simultaneous multithreading mode. For the second trend, the performance improvement in single-threaded mode again is only about 4%. Finally, as in the case of Gaussian03, we see a large performance improvement using simultaneous multithreading when running with eight logical processors. The gain is of the order of 25%.

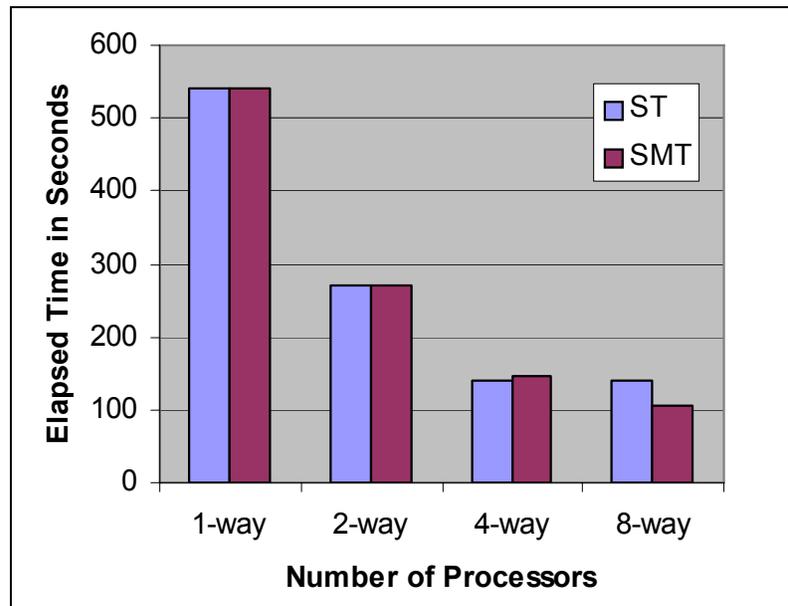


Figure 3-9 AMBER7 benchmark test comparison

Using the same procedure as with the Gaussian03 tests, we ran throughput benchmarks with one, two, four, and eight processors running simultaneous copies of the Joint AMBER-CHARMM benchmark input. The results presented in Figure 3-10 are similar to the results discussed for Gaussian03.

Perhaps the largest qualitative difference between this case and Gaussian03 is for four simultaneous jobs. AMBER7 does not seem to be taking as much advantage of simultaneous multithreading as Gaussian did in this case.

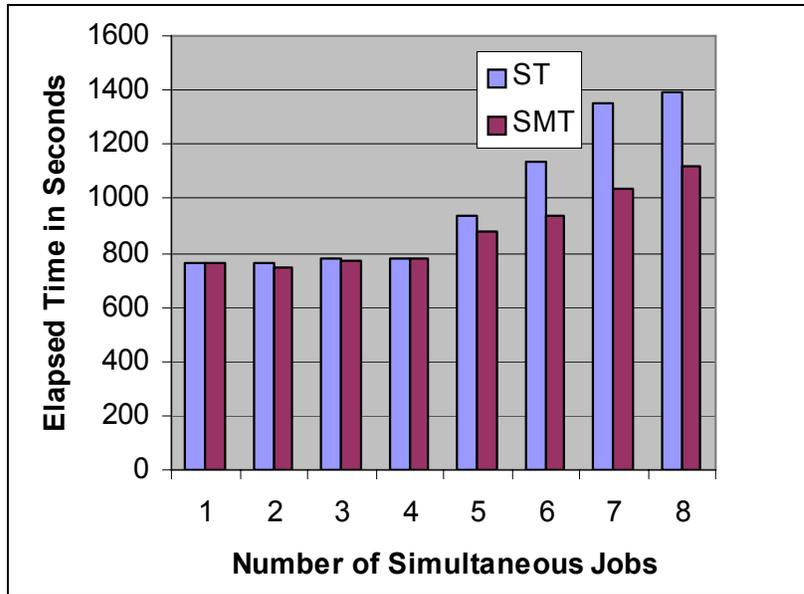


Figure 3-10 AMBER7 results for simultaneous jobs

In Figure 3-11, AMBER7 takes advantage of simultaneous multithreading as a function of the number of simultaneous jobs running on the machine. From one to four simultaneous jobs, we see AMBER7 taking slight advantage of simultaneous multithreading.

However, as the number of simultaneous jobs increases, so does the advantage of using simultaneous multithreading. We see that in this case, when running seven simultaneous jobs, the improvement when compared to single-threaded is as high as 25%. The behavior of the case with eight simultaneous jobs may be explained as operating system noise.

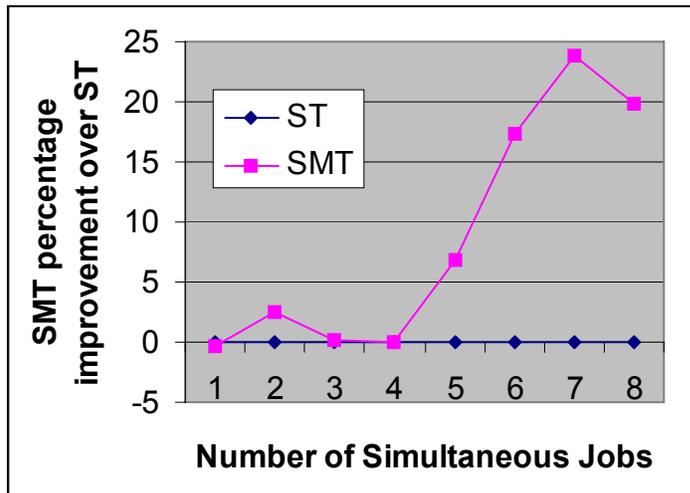


Figure 3-11 AMBER7 simultaneous multithreading improvement

BLAST benchmark tests

Figure 3-12 shows the results of the BLAST benchmark tests, with the same trends as for Gaussian03 and AMBER7. However, it appears that BLAST tends to favor simultaneous multithreading more than the other two applications. In the two-way test, unlike the other two benchmark tests, simultaneous multithreading has about a 2% advantage. Trend one is the same, except that for the two-way run, BLAST favors the simultaneous multithreading by about 2%.

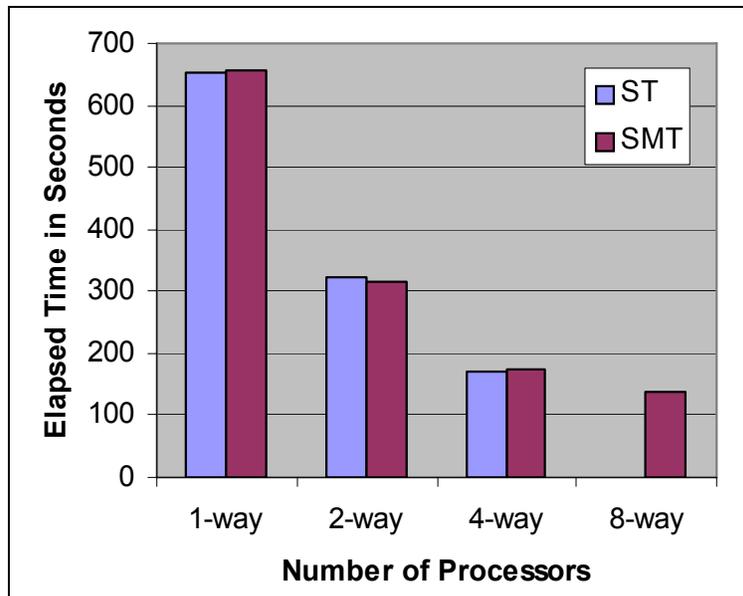


Figure 3-12 BLAST benchmark test comparison

Figure 3-13 on page 69 summarizes the results for BLAST. BLAST is different from the other two applications, Gaussian03 and AMBER7, which are floating-point intensive applications, while BLAST relies on pattern matching.

Again, BLAST shows behavior similar to the two previous applications. For cases with one to four jobs running simultaneously, little use is made of simultaneous multithreading. However, as soon as the number of jobs exceeds the number of physical processors, the advantage of simultaneous multithreading is clear.

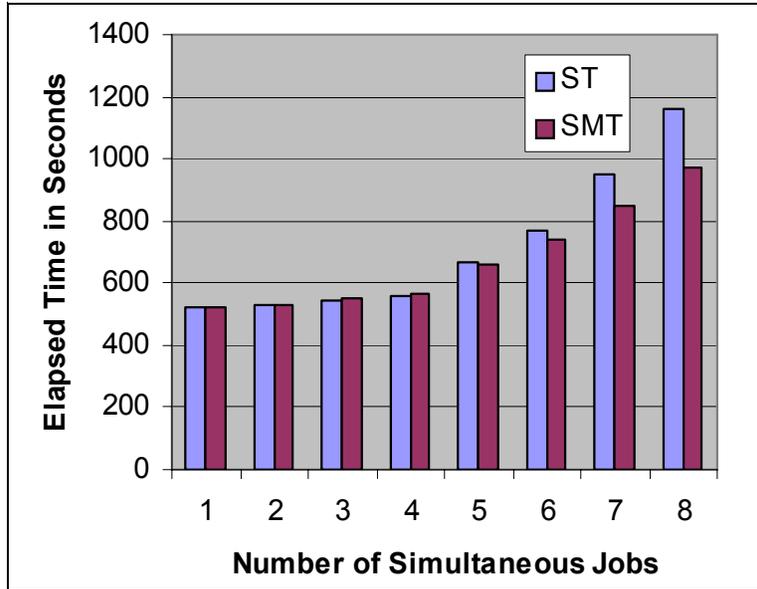


Figure 3-13 BLAST results for simultaneous jobs

A more dramatic difference showing the benefit of simultaneous multithreading can be seen in Figure 3-14. We see that in the case where we have doubled the number of jobs compared to the number of physical processors, simultaneous multithreading shows a performance improvement over single-threaded by as much as 16% difference.

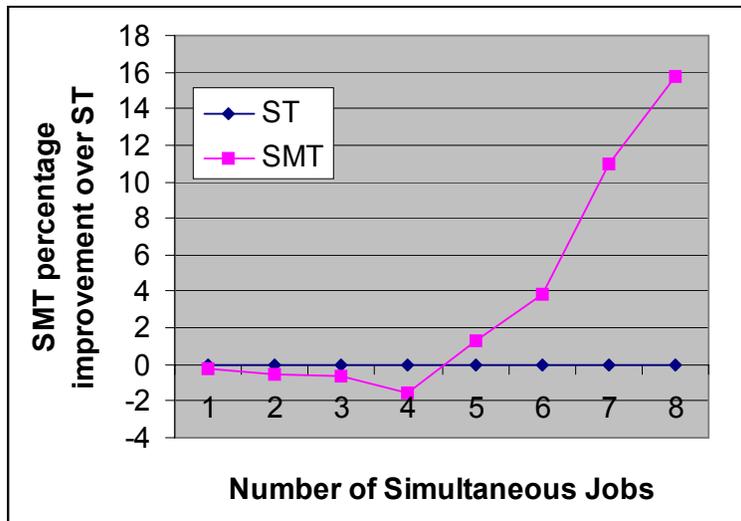


Figure 3-14 BLAST simultaneous multithreading improvement

FLUENT benchmark tests

The purpose of these tests was to determine whether the FLUENT application would benefit from simultaneous multithreading. The measure of performance used in this experiment is *FLUENT rating*, which is the number of FLUENT jobs that can be completed in a 24-hour time period. Higher values of FLUENT rating indicate better performance.

The application is submitted requesting one-way (sequential), and two-way and four-way (parallel) when the system is configured in single-threaded mode. To ensure that each thread is running on a different physical or logical processor, we use the **bindprocessor** command. When the system was configured in simultaneous multithreading mode, the single parallel job is submitted using one, two, four, and eight processes. In the experiments where simultaneous multithreading was used, one processor was assigned to two processes of the parallel job. When the parallel job contained one process, complete resources of a processor were assigned to the process under both single-threaded and simultaneous multithreading.

Table 3-3 shows the results of running a single parallel job on the single-threaded and simultaneous multithreading configurations. When the parallel job contains one process, the results for both single-threaded and simultaneous multithreading are almost identical, indicating that running in simultaneous multithreading mode does not affect performance. The performance of single-threaded and simultaneous multithreading is compared for a given number of physical processors. The number processes in the parallel job is equal to the number of physical processors in single-threaded and it is double the number of physical processors in simultaneous multithreading mode. Based on the results, running in simultaneous multithreading mode gives a 33% boost in performance using one physical processor. When the system is fully loaded, the improvement is slightly less at 23%. This improvement resulted in super-linear speed-up when the single process run is used to compute the speed-up.

Table 3-3 Performance of parallel FLUENT test case

Physical CPUs	Single-thread mode			Simultaneous multi-thread mode			
	Processes	FLUENT Rating jobs/day (A)	Speedup	Processes	FLUENT Rating jobs/day (B)	Speedup	Simultaneous multithreading vs single-threaded (B)/(A)
1				1	166.6	1.0	
1	1	166.8	1.0	2	221.4	1.3	1.33
2	2	334.3	2.0	4	415.4	2.5	1.24
4	4	625.0	3.8	8	768.2	4.6	1.23

In order to evaluate the performance of single-threaded and simultaneous multithreading features on a throughput benchmark, a set of several serial jobs was submitted simultaneously and the FLUENT rating for each job was measured. The total throughput was computed by multiplying the number of processes by the average throughput for the set of jobs. In simultaneous multithreading mode, three sets of jobs were used. These sets contained one, two, and four jobs, respectively. Each job in each of these sets was assigned to a processor. For the simultaneous multithreading configuration, four sets of jobs were submitted. These four sets contained one, two, four, and eight jobs. One processor was used for the jobs in each these sets.

Table 3-4 shows the results of running several serial jobs on the single-threaded and simultaneous multithreading configurations. The performance of single-threaded and simultaneous multithreading is compared for a given number of physical processors. The number of jobs in the parallel run is equal to the number of physical processors in single-threaded mode, and it is double the number of physical processors in simultaneous multithreading mode. Based on the results, simultaneous multithreading mode gives a 35% boost in performance for a single physical processor.

Table 3-4 Throughput performance of serial FLUENT for test case: FL5M3

Single-thread mode				Simultaneous multi-thread mode				
Jobs	CPUs	FLUENT Rating for single job (B)	Total FLUENT Rating for all jobs (A)*(B)	Jobs	CPUs	FLUENT Rating for Single job (E)	Total FLUENT Rating for all jobs (D)*(E)	SMT vs single-threaded (D)*(E)/(A)*(B)
				1	1	166.6	166.6	
1	1	166.8	166.8	2	1	112.2	224.4	1.35
2	2	167.3	334.6	4	2	109.7	436.0	1.31
4	4	163.6	654.4	8	4	108.5	871.2	1.33

3.6 Summary

We have tried to illustrate a series of scenarios where scientific applications can take full advantage of simultaneous multithreading. We included the throughput benchmarks in order to replicate the workloads that a supercomputing center might experience on a day-to-day basis. These throughput benchmarks were carried out by running multiple copies of a single application. Of course, if the input is identical, all of the particular jobs will be competing for the same resources. However, in order to provide a more balanced representation of a real

workload, we combined the three applications into one throughput benchmark. Figure 3-15 summarizes the results from this benchmark.

With this benchmark we are trying to measure which mode will provide the best throughput results. However, prior to discussing the results, it is important to define how we ran this benchmark. Here the amount of time that the benchmarks were going to run was predefined based on how long the individual runs take. Given that constraint, our threshold of 90 minutes was arbitrary. We wrote a script that would submit Gaussian03, AMBER7, and BLAST jobs simultaneously. When the script reached 90 minutes, all jobs were stopped and the total number of completed jobs in this period was used as the measurement of performance.

For these types of benchmarks, we see that the simultaneous multithreading benefit is clear. We see performance improvements from 20% to almost 60% difference when compared to single-threaded. The largest improvement corresponds to AMBER7 with almost 60% in comparison with single-threaded.

Will using simultaneous multithreading benefit your environment? There is no straightforward answer to that question and it depends on the application or applications in the system. Later in this book, we describe the performance tools that can assist you in determining whether simultaneous multithreading is desirable.

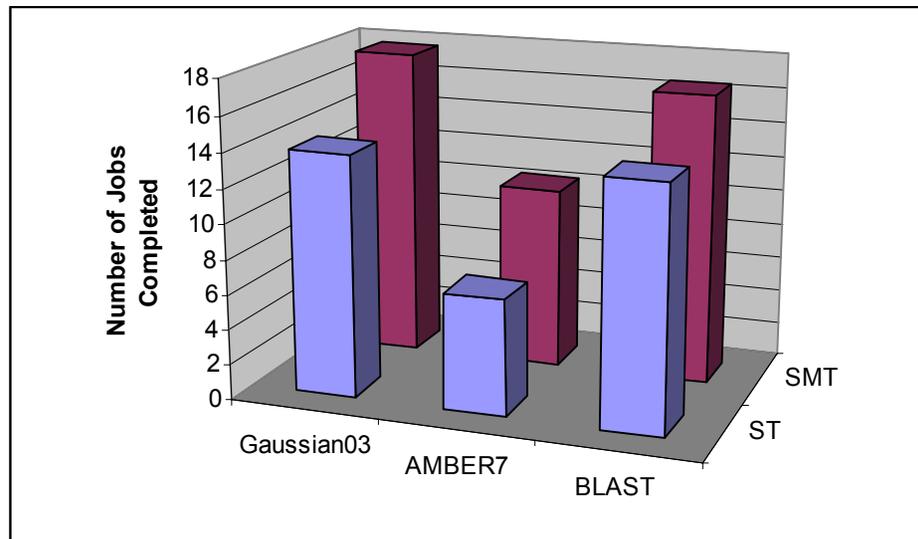


Figure 3-15 Benchmark comparison of Gaussian03, AMBER7, and BLAST



POWER Hypervisor

The technology behind the virtualization of the IBM @server p5 systems is provided by a piece of firmware known as the POWER Hypervisor, which resides in flash memory. This firmware performs the initialization and configuration of the POWER5 processor, as well as the virtualization support required to run up to 254 partitions concurrently on the IBM @server p5 servers.

The POWER Hypervisor supports many advanced functions when compared to the previous version found in POWER4 processor-based systems. This includes sharing of processors, virtual I/O, and high-speed communications among partitions using a virtual LAN, and it enables multiple operating systems to run on the single system. Currently, the AIX 5L, Linux, and i5/OS™ operating systems are supported, as shown in Figure 4-1 on page 74.

With support for dynamic resource movement across multiple environments, clients can move processors, memory, and I/O between partitions on the system as workloads are moved between the partitions.

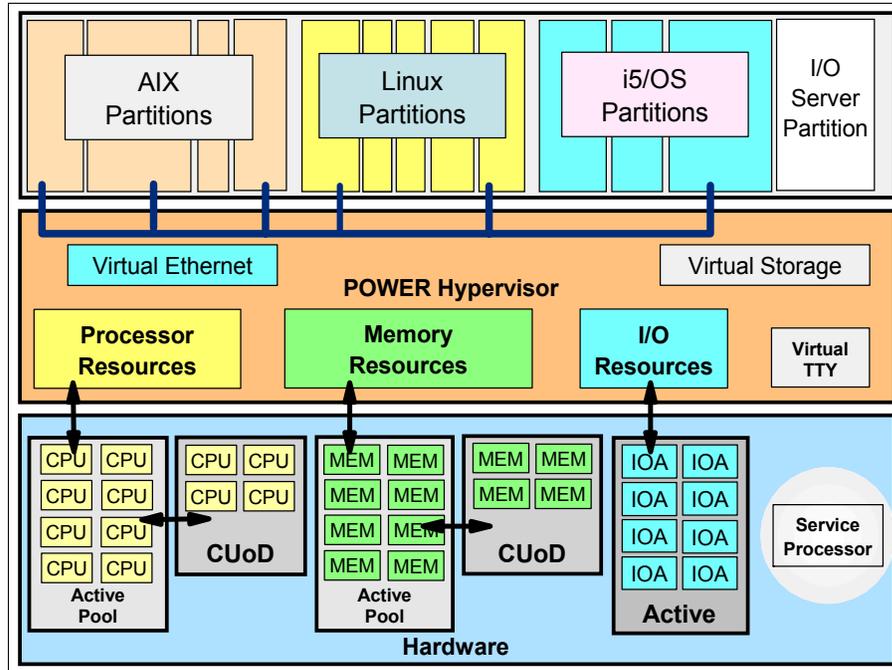


Figure 4-1 Virtualization technologies implemented on POWER5 servers

The POWER Hypervisor is the underlying control mechanism that resides below the operating systems but above the hardware layer (Figure 4-2). It owns all system resources and creates partitions by allocating and sharing them.

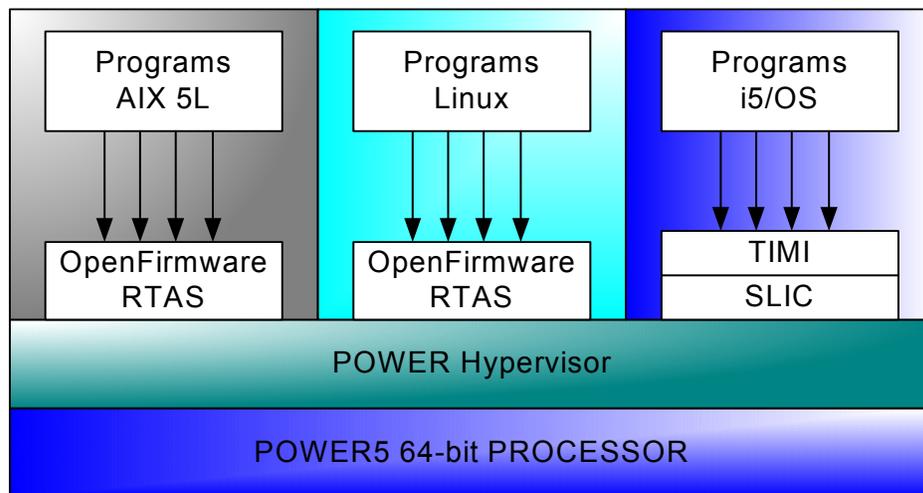


Figure 4-2 IBM eServer p5 system layers

The layers above the POWER Hypervisor are different for each supported operating system. For the AIX 5L and Linux operating systems, the layers above the POWER Hypervisor are similar but the contents are characterized by each operating system. The layers of code supporting AIX 5L and Linux consist of system firmware and Run-Time Abstraction Services (RTAS).

System firmware is composed of *low-level firmware* (code) that performs server unique input/output (I/O) configurations and the Open Firmware that contains the boot-time drivers, boot manager, and the device drivers required to initialize the PCI adapters and attached devices. RTAS consists of code that supplies platform-dependent accesses and can be called from the operating system. These calls are passed to the POWER Hypervisor that handles all I/O interrupts.

The distinction between RTAS and Open Firmware is important. Open Firmware and RTAS are both platform-specific firmware and both are tailored by the platform developer to manipulate the specific platform hardware. RTAS encapsulates some of the machine-dependent operations of the IBM @server p5 systems into a machine-independent package. The operating system can call RTAS to do things such as start and stop processors in an SMP configuration, display status indicators (such as LEDs), and read/write NVRAM without having to know the intricate details of how the low-level functions are implemented on particular platforms. Open Firmware, on the other hand, does not have to be present when the operating system is running. Open Firmware is defined by the IEEE 1275 standard and is a specification for machine-independent BIOS that is capable of probing and initializing devices that have IEEE-1275 compliant Forth code in their ROMs. The device tree produced by Open Firmware can then be passed to the operating system when control is passed to the operating system during boot. Read more about the IEEE 1275 Open Firmware standard at:

<http://www.openfirmware.org>

For i5/OS, Technology Independent Machine Interface (TIMI) and the layers above the POWER Hypervisor are still in place. System Licensed Internal Code (SLIC), however, is changed and enabled for interfacing with the POWER Hypervisor. The POWER Hypervisor code is based on the iSeries™ Partition Licensed Internal Code (PLIC) code that is enhanced for use with the IBM @server i5 hardware. The PLIC is now part of the POWER Hypervisor.

Attention: The POWER Hypervisor is mandatory on all POWER5 processor-based systems. This includes any single-LPAR system.

4.1 POWER Hypervisor implementation

The POWER4 processor introduced support for logical partitioning with a new privileged processor state called *POWER Hypervisor mode*. It is accessed using POWER Hypervisor calls, which are generated by the operating system's kernel running in a partition. POWER Hypervisor mode allows for a secure mode of operation that is required for various system functions where logical partition integrity and security are required. The POWER Hypervisor validates that the partition has ownership of the resources it is attempting to access, such as processor, memory, and I/O, then completes the function. This mechanism allows for complete isolation of partition resources.

In the POWER5 processor, further design enhancements are introduced that enable the sharing of processors by multiple partitions. The POWER Hypervisor Decrementer (HDEC) is a new hardware facility in the POWER5 design that is programmed to provide the POWER Hypervisor with a timed interrupt independent of partition activity. The HDEC is described in "POWER Hypervisor Decrementer" on page 33. HDEC interrupts are routed directly to the POWER Hypervisor, and use only POWER Hypervisor resources to capture state information from the partition. The HDEC is used for fine-grained dispatching of multiple partitions on shared processors. It also provides a means for the POWER Hypervisor to dispatch physical processor resources for its own execution.

The POWER5 processor supports special machine instructions and are exclusively used by the POWER Hypervisor. If an operating system instance in a partition requires access to hardware, it first invokes the POWER Hypervisor by using POWER Hypervisor calls. The POWER Hypervisor allows privileged access to the operating system for dedicated hardware facilities and includes protection for those facilities in the processor and memory locations.

The primary POWER Hypervisor calls used by the operating system in the dispatch of a virtual processor are:

H_CEDE	Used when a virtual processor or thread becomes idle, enabling the POWER Hypervisor to dispatch other work.
H_CONFER	Used to grant the remaining cycles in a dispatch interval to another virtual processor in the partition. It may be used when one virtual processor cannot make forward progress because it is waiting on an event to complete on another virtual processor, such as a lock miss.
H_PROD	Used to activate a virtual processor that has ceded or conferred processor cycles.

A virtual processor will always be in one of four logical states. These states are:

Runnable	Ready to run, waiting for dispatch
Running	Currently dispatched on a physical processor
Not-runnable	Has ceded or conferred its cycles
Expired	Consumed its full entitled cycles for the current dispatch window

Architecturally, the POWER Hypervisor, a component of global firmware, owns the partitioning model and the resource abstractions that are required to support that model. Each partition is presented with the resource abstraction for its partition and other required information through the Open Firmware device tree, which is created by firmware and copied into the partition before the operating system is started. In this way, operating systems receive resource abstractions. They also participate in the partitioning model by making POWER Hypervisor calls at key points in their execution as defined by the model.

The introduction of shared processors did not fundamentally change this model. New virtual processor objects and POWER Hypervisor calls have been added to support shared processor partitions. Actually, the existing physical processor objects have just been refined to not include physical characteristics of the processor, because there is not a fixed relationship between a virtual processor and the physical processor that actualizes it. These new POWER Hypervisor calls are intended to support the scheduling heuristic for minimizing idle time.

The POWER Hypervisor is entered by the way of three interrupts:

► System reset interrupt

A non-maskable, asynchronous interrupt that is caused by a command for *soft reset* invoked from the service processor. The POWER Hypervisor code saves all processor state by saving the contents of the processor's registers (multiplexing the use of this resource with the operating system). The processor's stack and data are found by processing the Processor Identification Register (PIR). The PIR is a read-only register. During power-on reset, it is set to a unique value for each processor in a multi-processor system.

► Machine Check Interrupt

The following causes of machine check interrupts are precise and synchronous with the instruction that caused the operation that encountered the error:

- The detection of a parity error in the L1 data cache, the data effective-to-real (D_ERAT), the translation lookaside buffer, or the segment lookaside buffer during the execution of a load or store instruction. If the

interrupt is caused by a soft error, executing the appropriate sequence of instructions in the Machine Check Handler program will clear the error condition without causing any loss of state.

- The detection of an uncorrectable *error-correcting code* (ECC) error in the L2 cache when a Load instruction is executed.
- The detection of an uncorrectable ECC error in the L2 cache while the Page Table is being searched in the process of translating an address.
- The detection of corrupt data that is being returned to satisfy a Load instruction for which the effective address specified a location in caching inhibited memory.

The POWER Hypervisor code saves all processor state by saving the contents of the processor's registers (multiplexing the use of this resource with the operating system). The processor's stack and data are found by processing the Processor Identification Register (PIR).

The POWER Hypervisor investigates the cause of the machine check. The cause may be either a recoverable event on the current processor or one of the other processors in the logical partition. Also the POWER Hypervisor must determine whether the machine check has corrupted its own internal state (by looking at the footprints, if any, that were left in the per processor data area of the errant processor).

► System (Hypervisor) call interrupt

The POWER Hypervisor call interrupt is a special variety of the `sc` (system call) instruction. The parameters to a POWER Hypervisor call are passed in registers using the PowerPC Application Binary Interface (ABI) definitions. This ABI specifies an interface for compiled application programs to system software. A copy of the ABI specification can be found at:

<http://www.linuxbase.org/spec/ELF/ppc64>

In contrast to the PowerPC ABI, passing parameters by reference are avoided in POWER Hypervisor calls. This minimizes the address translation problem that parameters passed by reference would cause because address translation is disabled automatically when interrupts are invoked. Input parameters may be indexes. Output parameters may be passed in the registers and require special in-line assembler code on the part of the caller. The first parameter in the POWER Hypervisor call function table to POWER Hypervisor call is the function token. The assignment of function token is designed such that a single mask operation can be used to validate the value to be within the range of a reasonable-size branch table. Entries within the branch table can handle unimplemented code points. Some of the POWER Hypervisor calls indicate whether the system is in LPAR mode and which ones are available. The Open Firmware property is provided in the `/rtas` node of the partition's device tree. The property is present if the system is in LPAR

mode while its value specifies which function sets are implemented by a given implementation. If the system implements any POWER Hypervisor call of a function set, it implements the entire function set. Additionally, certain values of the Open Firmware property indicate that the system supports a given architecture extension to a standard POWER Hypervisor call.

The POWER Hypervisor routines are optimized for execution speed. In some rare cases, locks will have to be taken, and short wait loops will be required due to specific hardware designs. However, if a needed resource is truly busy, or processing is required by an agent, the POWER Hypervisor returns to the caller, either to have the function retried or continued later.

4.1.1 POWER Hypervisor functions

The POWER Hypervisor provides the following functions. Table 4-1 on page 81 shows the list of POWER Hypervisor calls.

Attention: This information is not intended to be a programming reference and these calls may change in future levels of firmware. However, these definitions may provide a better understanding of the mechanics within the POWER Hypervisor.

► Page frame table

The page frame table describes the pages of memory. The access functions to the page frame table carefully update a Page Table Entry (PTE) with at least 64-bit store operations because an invalid update sequence could result in machine check. The POWER Hypervisor protects the system from a checkstop condition (a condition where the processor becomes architecturally frozen) by allocating bits associated with PTE locks and reserved by the operating system to indicate that the PTE is in use.

For logical addressing, an additional level of virtual addresses translation is managed by the POWER Hypervisor. The operating system is not allowed to use the physical address for its memory; this includes main storage, memory-mapped I/O (MMIO) space, and NVRAM. The operating system sees main storage as regions of contiguous logical memory. Each logical region is mapped by the POWER Hypervisor into a corresponding block of contiguous physical memory on a specific node. All regions on a specific system are the same size, though different systems with different amounts of memory may have different region sizes because they are the amount of memory allocation to partitions. That is, partitions are granted memory in region-size chunks, and if a partition's operating system gives up memory, it is in units of a full region.

- ▶ Translation control entry

Translation control entry (TCE) access is provided by a POWER Hypervisor call and take as a parameter, the Logical I/O Bus Number (LIOBN), which is the logical bus number value derived from the property that is associated with the particular I/O adapter. TCE is responsible for the I/O address to memory address translation in order to perform direct memory access (DMA) transfers between memory and PCI adapters. The TCE tables are allocated in the physical memory.
- ▶ Debugger support

Debugger support provides the capability for the real mode debugger to be able to get to its serial port and beyond the real mode limit register without turning on virtual address translation.
- ▶ Virtual Terminal support

The POWER Hypervisor provides console access to every logical partition without a physical device assigned. The console emulates a *vt320* terminal that can be used to access the partition system using the Hardware Management Console (HMC). A partition's device tree that contains one or more nodes notifies that it has been assigned to one or more virtual terminal (vterm) client adapters. The unit address of the node is used by the partition to map the virtual device (or devices) to the operating system's corresponding logical representations and notify the partition that the virtual adapter is a vterm client adapter. The node's interrupts property specifies the interrupt source number that has been assigned to the client vterm I/O adapter for receive data.
- ▶ Dump support

This enables the operating system to dump POWER Hypervisor data areas in support of field problem diagnostics. The dump function set contains the POWER Hypervisor call **H_HYPERVISOR_DATA**. This call is enabled or disabled (default disabled) via the Hardware Management Console.
- ▶ Memory Migration Support

The Memory Migration Support POWER Hypervisor call was provided to assist the operating system in the memory migration process. It is the responsibility of the operating system not to change the DMA mappings referenced by the translation buffer. Failure of the operating system to serialize relative to the logical bus numbers may result in DMA data corruption within the caller's partition.
- ▶ Performance Monitor Support

The performance registers will be saved when a virtual processor yields or is preempted. They will be restored when the state of the virtual processor is restored on the hardware. A bit in one of the performance monitor registers

enables the partition to specify whether the performance monitor registers count when a POWER Hypervisor call (except yield) is made. When a virtual processor yields or is preempted, the performance monitor registers will not count, enabling a partition to query the POWER Hypervisor for appropriate information regarding POWER Hypervisor code and data addresses.

Table 4-1 POWER Hypervisor calls

Hypervisor call	Definition
H_REGISTER_VPA	Provides a data area registered with the Hypervisor by the operating system for each virtual processor. The VPA is the control area that holds information used by the POWER Hypervisor and the OS in cooperation with each other.
H_CEDE	Has the virtual processor, which has no useful work to do, enter a wait state, ceding its processor capacity to other virtual processors until some useful work appears, signaled either through an interrupt or an H_PROD call.
H_CONFER	Enables a virtual processor to give its cycles to one or all other virtual processors in its partition.
H_PROD	Makes the specific virtual processor runnable.
H_ENTER	Adds an entry into the page frame table. PTE high and low order bytes of the page table contain the new entry.
H_PUT_TCE	Provides mapping of a single 4096-byte page into the specified TCE.
H_READ	Returns the contents of a specific PTE into GPR4 and GPR5.
H_REMOVE	Invalidates an entry in the page table.
H_BULK_REMOVE	Invalidates up to four entries in the page frame table.
H_GET_PPP	Returns the partition's performance parameters.
H_SET_PPP	Enables the partition to modify its entitled processor capacity percentage and variable processor capacity weight within limits.
H_CLEAR_MODE	Clears the modified bit in the specific PTE. The second double word of the old PTE is returned in GPR4.

Hypervisor call	Definition
H_CLEAR_REF	Clears the reference bit in the specific PTE from the partition's node page frame table.
H_PROTECT	Sets the page protects bits in the specific PTE.
H_EOI	Incorporates the interrupt reset function when specifying an interrupt source number associated with an interpartition logical I/O adapter.
H_IPI	Generates an interprocessor interrupt.
H_CPPR	Sets the processor's current interrupt priority.
H_MIGRATE_DMA	This call is extended to serialize the sending of a logical LAN message to allow for migration of TCE mapped DMA pages.
H_PUT_RTCE	Maps the number of contiguous TCEs in an RTCE to the same number of contiguous I/O adapter TCEs.
H_PAGE_INIT	Initializes pages in real mode either to zero or to the copied contents of another page.
H_GET_TCE	This standard call is used to manage the interpartition logical LAN adapters's I/O translations.
H_COPY_RDMA	Copies data from an RTCE table mapped buffer in one partition to an RTCE table mapped buffer in another partition, with the length of the transfer being specified by the transfer length parameter in the call.
H_SEND_CRQ	Sends one 16-byte message to the partner partition's registered Command / Response Queue (CRQ). The CRQ facility provides ordered delivery of messages between authorized partitions.
H_SEND_LOGICAL_LAN	Sends a logical LAN message.
H_ADD_LOGICAL_LAN_BUF	Adds receive buffers to the logical LAN receive buffer pool.
H_PIC	Returns the summation of the physical processor pool's idle cycles.
H_XIRR	This call is extended to report the source number associated with virtual interrupts from an interpartition logical LAN I/O adapter.

Hypervisor call	Definition
H_POLL_PENDING	Provides the operating system with the ability to perform background administrative functions and the implementation with indication of pending work so that it may more intelligently manage the use of hardware resources.
H_PURR ^a	This call is a new resource provided for Micro-Partitioning and simultaneous multithreading. It provides an actual count of ticks that the shared resource has used on a per virtual processor or per thread basis. In the case of Micro-Partitioning, the virtual processor's Processor Utilization Resource Register begins incrementing when the virtual processor is dispatched onto a physical processor. Therefore, comparisons of elapsed PURR with elapsed Time_Base provides an indication of how much of the physical processor a virtual processor is getting. The PURR will also count Hypervisor calls made by the partition.

a. See "Processor Utilization Resource Register (PURR)" on page 34.

Monitoring POWER Hypervisor calls

In AIX 5L Version 5.3, the **lparstat** command using the **-h** and **-H** flags displays Hypervisor statistical data about many POWER Hypervisor calls, including **cede**, **confer**, and **prod**. Using the **-h** flag adds summary POWER Hypervisor statistics to the default **lparstat** output. The following shows an example of this command, collecting statistics for one five-second interval.

Example 4-1 *lparstat -h* command

```
# lparstat -h 5 1

System configuration: type=Dedicated mode=Capped smt=0n lcpu=4 mem=3808

%user  %sys  %wait  %idle  %hypv  hcalls
-----  ----  -----  ----  -----  -----
  0.0   0.1   0.0   99.8   61.2  2440007
#
```

Using the **-H** flag displays detailed POWER Hypervisor information, including statistics for many POWER Hypervisor call functions. The output in Example 4-2 on page 84 shows the following for each of these POWER Hypervisor calls:

Number of calls	Number of POWER Hypervisor calls made
Total Time Spent	Percentage of total time spent for this type of call

Hypervisor Time Spent Percentage of POWER Hypervisor time spent for this type of call

Average Call Time Average call time for this type of call in nanoseconds

Maximum Call Time Maximum call time for this type of call in nanoseconds

Example 4-2 lparstat -H command

```
# lparstat -H 5 1
```

```
System configuration: type=Dedicated mode=Capped smt=0n lcpu=4 mem=3808
```

Detailed information on Hypervisor Calls

Hypervisor Call	Number of Calls	%Total Time Spent	%Hypervisor Time Spent	Avg Call Time(ns)	Max Call Time(ns)
remove	0	0.0	0.0	1	714
read	0	0.0	0.0	1	193
nclear_mod	0	0.0	0.0	1	0
page_init	3	0.0	0.0	624	2212
clear_ref	0	0.0	0.0	1	0
protect	0	0.0	0.0	1	0
put_tce	0	0.0	0.0	1	613
xirr	8	0.0	0.0	598	1535
eoi	8	0.0	0.0	601	932
ipi	0	0.0	0.0	1	0
cpr	0	0.0	0.0	1	0
asr	0	0.0	0.0	1	0
others	0	0.0	0.0	1	0
enter	2	0.0	0.0	335	521
cede	12236005	61.5	100.0	499	83573
migrate_dma	0	0.0	0.0	1	0
put_rtce	0	0.0	0.0	1	0
confer	0	0.0	0.0	1	0
prod	31	0.0	0.0	446	1081
get_ppp	1	0.0	0.0	1477	2550
set_ppp	0	0.0	0.0	1	0
purr	0	0.0	0.0	1	0
pic	1	0.0	0.0	386	690
bulk_remove	0	0.0	0.0	1	0
send_crq	0	0.0	0.0	1	0
copy_rdma	0	0.0	0.0	1	0
get_tce	0	0.0	0.0	1	0
send_logical_lan	0	0.0	0.0	1	0
add_logical_lan_buf	0	0.0	0.0	1	0

```
#
```

4.1.2 Micro-Partitioning extensions

A new virtual processor is dispatched on a physical processor when one of the following conditions happens:

- ▶ The physical processor is idle and a virtual processor was made ready to run (interrupt or process).
- ▶ The old virtual processor exhausted its time slice (HDEC interrupt).
- ▶ The old virtual processor ceded or conferred its cycles.

When one of these conditions occurs, the POWER Hypervisor, by default, records all the virtual processor architected state including the Time Base and Decrementer values and sets the POWER Hypervisor timer services to wake the virtual processor per the setting of the decrementer. The virtual processor's Processor Utilization Resource Register (PURR) value for this dispatch is computed. The Virtual Processor Area (VPA) dispatch count is incremented (such that the result is odd). Then the POWER Hypervisor selects a new virtual processor to dispatch on the physical processor using an implementation-dependent algorithm having the following characteristics given in priority order:

1. The virtual processor is "ready to run" (has not ceded or conferred its cycles or exhausted its time slice).
2. Ready-to-run virtual processors are dispatched prior to waiting in excess of their maximum specified latency.
3. Of the non-latency critical virtual processors ready to run, select the virtual processor that is most likely to have its working set in the physical processor's cache or for other reasons will run most efficiently on the physical processor.

If no virtual processor is ready to run at this time, start accumulating the Pool Idle Count (PIC) of the total number of idle processor cycles in the physical processor pool.

Virtual I/O

Virtual input/output (I/O) support is one of the advanced features of the new POWER Hypervisor. Virtual I/O provides a given partition with the appearance of I/O adapters that do not necessarily have direct correspondence with a physical adapter. Virtual I/O is covered in detail in Chapter 6, "Virtual I/O" on page 143.

Memory considerations

POWER5 processors use memory to temporarily hold information. Memory requirements for partitions depend on partition configuration, I/O resources assigned, and applications used. Memory can be assigned in increments of 16 MB.

Depending on the overall memory in your system and the maximum memory values you choose for each partition, the server firmware must have enough memory to perform logical partition tasks. Each partition has a Hardware Page Table (HPT); its size is based on an HPT ratio and determined by the maximum memory values you establish for each partition. The HPT ratio is 1/64.

When selecting the maximum memory values for each partition, consider the following:

- ▶ Maximum values affect the HPT size for each partition.
- ▶ The logical memory map size of each partition.

When you create a logical partition on your managed system, the managed system reserved an amount of memory to manage the logical partition. Some of this physical partition is used for POWER Hypervisor page table translation support. The current memory available for partition usage as displayed by the HMC is the amount of memory that is available to the logical partitions on the managed system (Figure 4-3). This is the amount of active memory on your managed system minus the estimated memory needed by the managed system to manage the logical partitions defined on your system. Therefore, the amount in this field decreases for each additional logical partition you create.

The screenshot shows a configuration window titled "Specify desired, minimum and maximum amounts of memory for this profile using a combination of the gigabyte and megabyte fields below." It displays the following information:

- Installed memory (MB): 4096
- Current memory available for partition usage (MB): 3824

Below this information are three columns of input fields for memory configuration:

Minimum memory	Desired memory	Maximum memory
0 GB	0 GB	0 GB
128 MB	128 MB	128 MB

At the bottom of the window are buttons for "Help", "?", "< Back", "Next >", "Finish", and "Cancel".

Figure 4-3 Current memory available for partition usage using HMC

When you assess changing performance conditions across system reboots, it is important to know that memory allocations might change based on the availability of the underlying resources. Memory is allocated by the system across the system. Applications in partitions cannot determine where memory has been physically allocated.

4.1.3 POWER Hypervisor design

The POWER Hypervisor is primarily responsible for affinity in a Micro-Partitioning system. The physical processors in the shared processor pool are grouped within natural hardware boundaries, such that all processors within the pool have the same affinity characteristics and the partition is guaranteed to only execute on that pool of processors, barring events such as a processor being GUARD'ed off due to predictive failures, and possibly replaced with a spare processor from another affinity domain. See Figure 5-12 on page 119 for the relationship between virtual and physical processors.

The POWER Hypervisor will continue to provide affinity domain information in the device tree for processors, which are actually virtual processors in a Micro-Partitioning configuration. The side effect of Micro-Partitioning might be limits to the depth of hierarchy of affinity domain information that can be provided—that is, instead of going down to the physical processor it might stop at the lowest common layer of all processors in the shared pool. The POWER Hypervisor attempts to maintain physical processor affinity when dispatching virtual processors. It will always try first to dispatch the virtual processor on the same physical processor as it last ran on, and depending on resource utilization will broaden its search out to the other processor on the POWER5 chip, then to another chip on the same MCM, then to a chip on another MCM.

Save and restore registers

The POWER Hypervisor will save the following registers when a state is saved for a virtual processor: GPRs, FPRs, CR, XER, LR, CTR, ACCR, SPRG0, SPRG1, SPRG2, SPRG3, ASR, SLB state, DAR, DEC, DSISR, SRR0, SRR1, PMCs, MMCR0/1/A, SDAR, DABR and SDR1.

Preemption of a virtual processor

The POWER Hypervisor is responsible for time slicing and managing the dispatching of the partitions across the physical processors. One of the features of the POWER4+™ and POWER5 that makes this possible is the POWER Hypervisor Decrementer (HDEC). This is a clock interrupt source utilized by the POWER Hypervisor to preempt a dispatched partition and regain control of the physical processors. This interrupt occurs even if external interrupts are disabled and cannot be masked by the partition. The POWER Hypervisor utilizes this HDEC to drive its partition dispatcher, so in reality, the POWER Hypervisor is managing the execution of multiple partition images across the same physical resources, just as an operating system manages the execution of multiple processes / threads within its partition instance.

The POWER4+ processor does not have support for the POWER Hypervisor decrementer. The SRR0 and SRR1 registers are used to present an HDEC

interrupt to the processor. To avoid loss of partition state, a pending HDEC interrupt will be held off for N (programmable hardware value) cycles if $MSR[RI]=0$. The number of cycles (N) has to be large enough to enable a partition to safely execute instructions until $SRR0$ and $SRR1$ are saved and indicated by the setting of $MSR[RI]=1$. If not, taking the HDEC interrupt would result in the corresponding loss of state because these registers are updated when an interrupt or exception occurs.

Note: For those not familiar with the POWER and PowerPC architecture, at the time of an exception or interrupt, $SRR0$ is loaded with either the address of the instruction that caused the exception, or the address of the instruction that would have been dispatched had the interrupt not occurred. $SRR1$ contains the Machine State Register (MSR) contents at the time of the exception or interrupt. For example, the thread being preempted may have been in user mode ($MSR[PR]=1$). For the interrupt to be serviced, the processor must be in supervisory (system) mode and this bit has to be cleared (0). If interrupts are not masked or held off, then the processor automatically saves off the current MSR into $SRR1$ and produces a new MSR value with appropriate bit settings, which is placed into the MSR. Therefore, if these registers are not saved, recovery may be impossible. The $MSR[RI]$ bit is not affected by exceptions or interrupts and can be used by the operating system to indicate recovery.

This places the requirement on the operating system to use the $MSR[RI]$ bit to avoid fatal failures that could occur because of POWER Hypervisor preemption of a virtual processor.

A POWER5-based server provides complete HDEC support that enables preemption with an unsaved $SRR0$ and $SRR1$.

The POWER Hypervisor issues a **sync** instruction on the processor when it preempts a virtual processor. This ensures that a storage access sequence (in particular, a Memory Mapped I/O sequence) by the preempted virtual processor is seen by the devices on the system in the order it was intended. The POWER Hypervisor will also do the equivalent of a dummy **stwcx** instruction to cancel a reservation that may be held by the yielding or preempted virtual processor.

Cache invalidations

The *segment lookaside buffer* (SLB) that was saved when the virtual processor yielded or was preempted is restored on each dispatch of a virtual processor. There is one SLB per thread (two per processor core). Information derived from the SLB may also be cached in the instruction, possibly with *Data Effective to Real Address Translation* (D_ERAT), along with information from the translation lookaside buffer (TLB).

The TLB of a processor is invalidated every time the partition ID of a virtual processor switched in on a processor is different from the partition ID of the virtual processor that last ran on it. The POWER4 family of processors provides an instruction to flush the TLB of a processor, avoiding the need for a broadcast of TLB invalidations.

Since the number of partitions exceeds the number of hardware partition IDs, shared processor partitions may share a hardware partition ID. This can lead to false invalidations of TLB entries. Since the TLB is flushed in many instances on a dispatch of a virtual processor dispatch, the false invalidations are not a concern.

When a partition is IPLed (rebooted) in the shared pool, all processors in the pool flush their instruction cache prior to switching in a virtual processor from the partition being IPLed.

POWER Hypervisor dispatching algorithm

Each shared pool has its own instantiation of the POWER Hypervisor dispatcher. The POWER Hypervisor uses the POWER5 HDEC, which is programmed to generate an interrupt every 10 ms (1/100 second), as a timing mechanism for controlling the dispatch of physical processors to system partitions. Each virtual processor is guaranteed to get its entitled share of processor cycles during each 10 ms dispatch window. Each shared processor partition is configured with a specific processor entitlement, based on a quantity of processing units, which is referred to as the partition's *entitled capacity*. The entitled capacity, along with a defined number of virtual processors, defines the physical processor resource that will be allotted to the partition. If a partition does not use its allocation of cycles in a scheduling window, it will lose the unused cycles. The minimum allocation of resource is 1 ms per processor; the POWER Hypervisor calculates number of ms using the capacity entitlement and the number of virtual processors for each shared pool. When a *capped* shared processor has received its capacity entitlement within a dispatch interval, it becomes not-runnable. An *uncapped* partition may get more than its allocation of cycles in a scheduling window. Virtual processors are time-sliced through the use of the Hypervisor Decrementer much like the operating system time slices threads. The POWER Hypervisor HDEC and time base will be used by the POWER Hypervisor dispatcher for virtual processor accounting.

The physical processor resource in a shared pool may become overcommitted (with respect to uncapped partitions). A suitable variation of the Time Function History Scheduling (TFHS) algorithm will be used for making dispatch decisions when the pool is overcommitted. The algorithm requires some notion of priority when making scheduling decisions.

4.2 Performance considerations

The POWER Hypervisor uses some system processor and memory resources (a small percentage). These resources are associated with virtual memory management (VMM), the POWER Hypervisor dispatcher, virtual processor data structures (including save areas for virtual processor), and for queuing of interrupts. The impact on performance should be minor for most workloads, but the impact increases with extensive amounts of page-mapping activity. Partitioning may actually help performance in some cases for applications that do not scale well on large SMP systems by enforcing strong separation between workloads running in the separate partitions.

Other areas where performance can be affected by the POWER Hypervisor are:

- ▶ Increasing path length
- ▶ Dispatching of virtual processors (saving and restoring state)
- ▶ TLB flush when a virtual processor is dispatched
- ▶ Increased misses in a shared processor's caches

Dispatching and interrupt latencies

Virtual processors have dispatch latency, because they are scheduled. When a virtual processor is made runnable, it is placed on a run queue by the POWER Hypervisor, where it sits until it is dispatched. The time between these two events is referred to as *dispatch latency*.

The dispatch latency of a virtual processor is a function of the partition entitlement and the number of virtual processors that are online in the partition. Entitlement is equally divided among these online virtual processors, so the number of online virtual processors affects the length of each virtual processor's dispatch. The smaller the dispatch cycle, the greater the dispatch latency.

Timers also have latency issues. The POWER5 Decrementer is virtualized by the POWER Hypervisor at the virtual processor level, so that timers will interrupt the initiating virtual processor at the designated time. If a virtual processor is not running, then the timer interrupt has to be queued with the virtual processor, as it is delivered in the context of the running virtual processor.

External interrupts have latency issues as well. External interrupts are routed directly to a partition. When the operating system makes the *accept pending interrupt* POWER Hypervisor call, the POWER Hypervisor, if necessary, dispatches a virtual processor of the target partition to process the interrupt. The POWER Hypervisor provides a mechanism for queuing up external interrupts that is also associated with virtual processors. Whenever this queuing mechanism is used, latencies are introduced.

These latency issues are not expected to cause functional problems, but they may present performance problems for real-time applications. To quantify matters, the worst case virtual processor dispatch latency is 18 ms, since the minimum dispatch cycle that is supported at the virtual processor level is 1 ms. This figure is based on the POWER Hypervisor dispatch wheel. It can be visualized by imagining that a virtual processor is scheduled in the first and last portions of two 10-ms intervals. In general, if these latencies are too great, then clients may increase entitlement, minimize the number of online virtual processors without reducing entitlement, or use dedicated processor partitions.

The output of `lparstat` with the `-h` flag displays the percentage spent in POWER Hypervisor (`%hypv`) and the number of POWER Hypervisor calls. Note from the example output shown in Example 4-3 that the `%hypv` is around 61% on this idle system. As was shown in Example 4-2 on page 84, this is the result of the `H_CEDD` call being made to place the virtual processor into a wait state because there is no meaningful work to do after servicing interrupts, and so on.

Example 4-3 lparstat -h output

```
# lparstat -h 1 16
System configuration: type=Dedicated mode=Capped smt=0n lcpu=4 mem=3808

%user  %sys  %wait  %idle  %hypv  hcalls
-----  ----  -----  -----  -----  -----
  0.1   0.6   0.0   99.4   61.2  2439926
  0.0   0.0   0.0  100.0   60.8  2442449
  0.0   0.0   0.0  100.0   61.2  2442355
  0.0   0.0   0.0  100.0   61.6  2439577
  0.0   0.0   0.0  100.0   60.8  2442471
  0.0   0.2   0.0   99.8   61.7  2436181
  0.0   0.0   0.0  100.0   61.2  2443133
  0.0   0.1   0.0   99.9   61.2  2448492
  0.0   0.0   0.0  100.0   61.2  2447438
  0.0   0.0   0.0  100.0   61.2  2446917
#
```

To provide input to the capacity planning and quality of service tools, the POWER Hypervisor reports certain statistics to an operating system. These include the number of virtual processors that are online, minimum processor capacity that the operating system can expect (the operating system may cede any unused capacity back to the system), the maximum processor capacity that the partition will grant to the operating system, the portion of spare capacity (up to the maximum) that the operating system will be granted, variable capacity weight, and the latency to a dispatch via a POWER Hypervisor call. The output of the `lparstat` command with the `-i` flag, shown in Example 4-4 on page 92, will report the logical partition related information.

Example 4-4 lparstat -i output

```
# lparstat -i
Node Name                : aix_lpar01
Partition Name           : AIX 5L Version 5.3 Gold
Partition Number         : 1
Type                     : Dedicated-SMT
Mode                     : Capped
Entitled Capacity        : 2.00
Partition Group-ID       : 32769
Shared Pool ID           : -
Online Virtual CPUs      : 2
Maximum Virtual CPUs     : 2
Minimum Virtual CPUs     : 1
Online Memory            : 3808 MB
Maximum Memory           : 4096 MB
Minimum Memory           : 128 MB
Variable Capacity Weight : -
Minimum Capacity         : 1.00
Maximum Capacity         : 2.00
Capacity Increment       : 1.00
Maximum Dispatch Latency : -
Maximum Physical CPUs in system : 2
Active Physical CPUs in system : 2
Active CPUs in Pool      : -
Unallocated Capacity     : -
Physical CPU Percentage   : 100.00%
Unallocated Weight       : -
#
```



Micro-Partitioning

In this chapter we discuss the detailed implementation for Micro-Partitioning, which is one of the key features provided in the IBM @server p5 systems.

The following topics are included:

- ▶ Partitioning on POWER5
- ▶ Micro-Partitioning implementation
- ▶ Performance considerations
- ▶ Configuration guidelines

5.1 Partitioning on the IBM eServer p5 systems

With technology inspired by IBM zSeries® heritage, logical partitioning (LPAR) appeared on IBM @server pSeries POWER4 processor-based systems supporting AIX 5L Version 5.1 in 2001. Logical partitioning of a system allows more than one operating system to reside on the same platform simultaneously without interfering with each other. With POWER4 technology, the smallest granularity of partitioning was the assignment of one processor to a partition. All partitions were considered *dedicated*, where an entire processor is dedicated to the partition and not allowed to be shared among other partitions. This means a 32-way IBM @server pSeries 690 can host up to 32 independent partitions for running a combination of AIX 5L and Linux.

Continuing the evolution of partitioning technology, the IBM @server p5 systems extends its capabilities by further improving the flexibility of LPARs. There are two types of partitions in the IBM @server p5 systems, and both types of partitions can coexist in the same system at any given time.

- ▶ Dedicated processor partitions
- ▶ Shared processor partitions or micro-partitions

In addition to sharing the processor, the IBM @server p5 systems provide sharing of devices through virtual I/O, virtual terminals, and virtual Ethernet. These topics will be covered later in this book.

Dedicated processor partitions

A *dedicated processor partition*, like the partitions used on servers based on the POWER4 processor-based servers, cannot share the processor with other partitions. These processors are owned by the partition where they are running. The amount of processing capacity on the partition is limited by the total processing capacity of the processors configured in that partition, and it cannot go over this capacity (unless you add more processors inside the partition using a dynamic LPAR operation). By default, a powered-off logical partition using dedicated processors will have its processors available for use by other partitions in the system.

Micro-Partitioning

Shared processor partitions or Micro-Partitioning¹ provides the ability to share processors among other partitions in the system. This allows a system to perform more efficiently than would be required with dedicated processor partitions.

¹ Some publications refer to this technology as *shared processor partitions*, but the terms micro-partitions and Micro-Partitioning in used this book.

Micro-Partitioning is the mapping of *virtual processors* to *physical processors*. The virtual processors are assigned to the partitions, not physical ones. With the assistance of the POWER Hypervisor, an *entitlement* or percentage of processor usage is granted to the shared partitions. The minimum processor entitlement is 1/10 of a processor for a partition. By dividing up processor usage in this manner, a system can have multiple partitions sharing the same physical processor, and dividing the processing capacity among themselves, as shown in Figure 5-1.

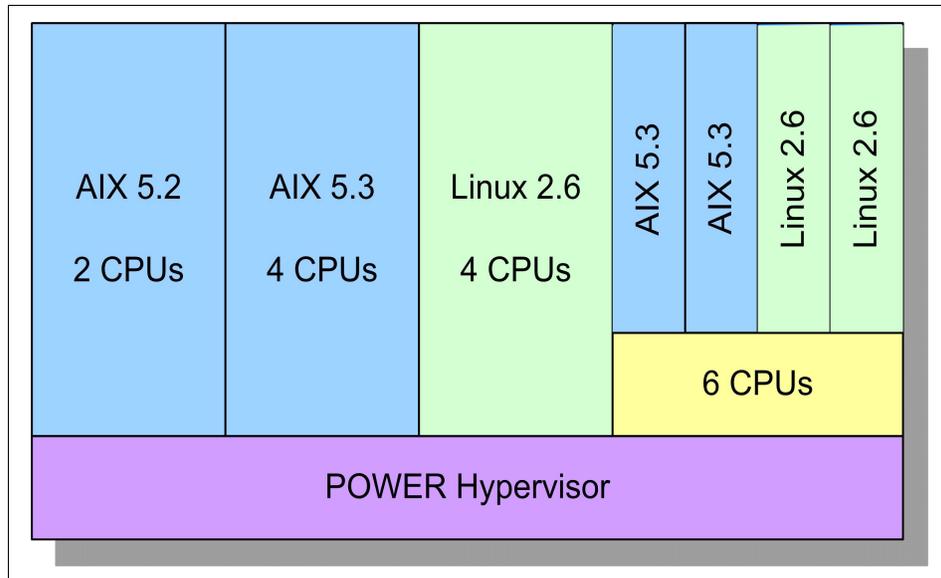


Figure 5-1 System with dedicated and shared partitions

With fractional processor allocations, more partitions can be created on a given platform, which enables clients to maximize the number of workloads that can be supported on a server simultaneously. Micro-Partitioning enables both optimized use of processing capacity while preserving the isolation between applications provided by separate operating system images.

There are several scenarios where the use of Micro-Partitioning can bring advantages such as optimal resource utilization, rapid deployment of new servers and application isolation:

Server consolidation Consolidating small systems onto a large and robust server brings advantages in management and performance, usually together with reduced total cost of ownership. Micro-Partitioning enables the consolidation from small and large systems without the burden of dedicating very powerful processors to a small partition. You can divide the processing power

	between several partitions with the adequate processing capacity for each one.
Server provisioning	With Micro-Partitioning and virtual I/O, a new partition can be deployed rapidly, to accommodate unplanned demands, or to be used as a test environment.
Virtual server farms	In environments where applications scale with the addition of new servers, the ability to create several partitions sharing processing resources is very useful and contributes to better use of processing resources by the applications deployed on the server farm.

5.2 Micro-Partitioning implementation

Micro-Partitioning enables several operating system images to share the physical processor resources in a time-sliced manner. From an operating system perspective, a virtual processor is indistinguishable from a physical processor. The key benefit of implementing partitioning in the POWER Hypervisor firmware and POWER5 chip architecture is to provide a transparent interface to the operating system.

Optionally, for increased resource flexibility, the operating system can be enhanced to exploit Micro-Partitioning. For instance, an operating system may voluntarily relinquish processor cycles to the Hypervisor when they are not needed. AIX 5L V5.3 is the first version of AIX 5L to support Micro-Partitioning. SUSE LINUX Enterprise Server 9 for POWER systems and Red Hat Enterprise Linux AS 3 for POWER Update 3 also include such optimizations.

The virtualization of physical processors on POWER5-based servers requires a new partitioning model because it is fundamentally different from the partitioning model used on POWER4-based servers. Several new terminologies and concepts are introduced in Micro-Partitioning.

Layers of Processor Abstraction

The following terminology represents the three types of processors used in Micro-Partitioning:

Logical Processor	A hardware thread; an operating system view of a managed processor unit. In the AIX 5L V5.3 operating system, each hardware thread appears as a unique processor (for example, <code>bindprocessor -q</code>). The number of logical processors will be double the number of virtual processors with simultaneous multithreading enabled. Both hardware threads on one virtual processor must be
--------------------------	---

in the same partition at the same time. Currently, 128 logical processors per partition is the maximum.

Virtual Processor Defines the way that a partition's entitlement may be spread over physical processor. The virtual processor is the unit of POWER Hypervisor dispatch and the granularity of processor dynamic reconfiguration. Currently, the maximum number of virtual processors is 64 per partition.

Physical Processor The actual physical hardware resource. Currently, the maximum number of physical processors in the POWER5 systems is 64. This definition is the number of unique processor cores, not the number of processor chips (each of which contains two processing cores).

Virtual processors

Virtual processors are the whole number of concurrent operations that the operating system can use. The processing power that is available to the operating system on the partition can be conceptualized as being spread equally across these virtual processors.

In Micro-Partitioning, the partitions are defined using the Hardware Maintenance Console (HMC). When you create a partition, you have to choose between a shared processor partition and a dedicated processor partition. You cannot mix shared processors and dedicated processors in one partition. Using the HMC menu shown in Figure 5-2 on page 98, selecting the optimal number of virtual processors depends on the workload in the partition.

To enable sharing of physical processors in Micro-Partitioning, you have to configure these additional options:

- ▶ Minimum, desired, and maximum processing units of capacity
- ▶ The processing sharing mode, either capped or uncapped
- ▶ Minimum, desired, and maximum virtual processors

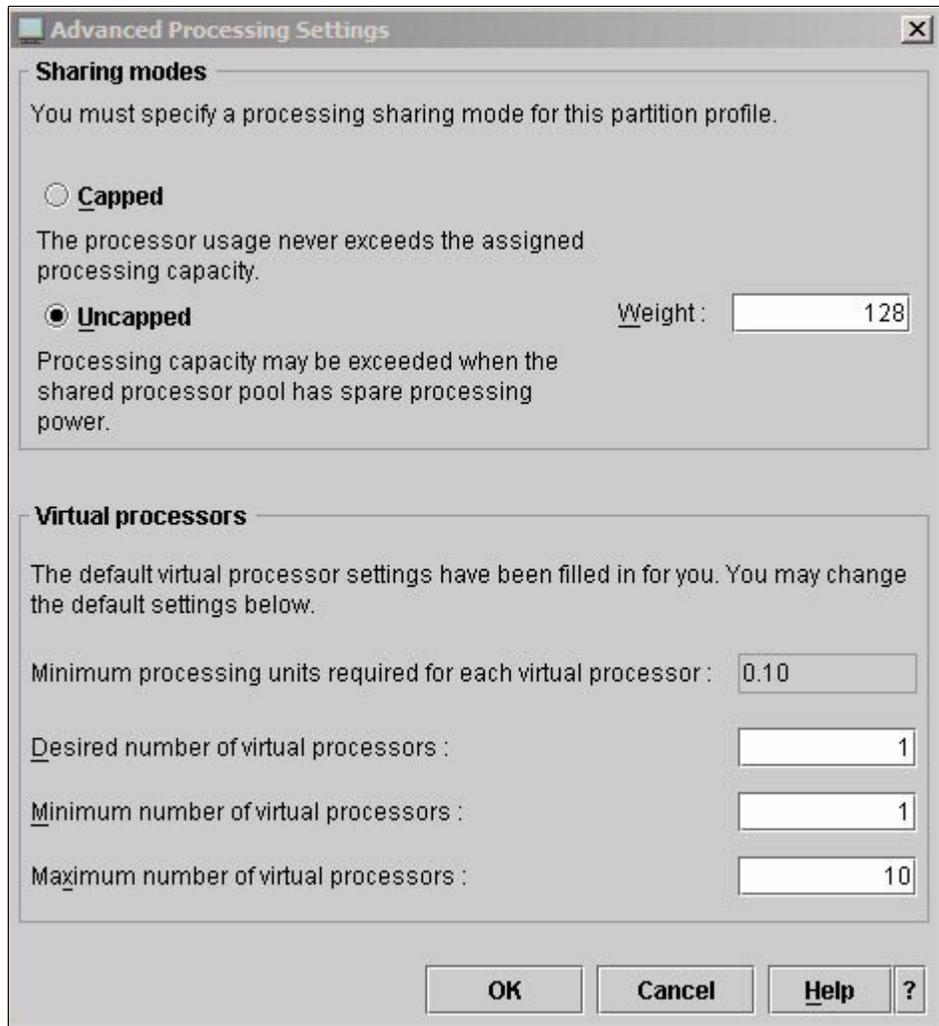


Figure 5-2 HMC console for virtual processor management

You also can use the Advanced tab in your partitions profile to change the default configuration and to assign more virtual processors. At the time of publication, the maximum number of virtual processors per partition is 64.

Processing capacity is specified in terms of *processing units*. Processing units can be configured in fractions of 1/100 of a processor. The *minimum* capacity of 1/10 of a processor is specified as 0.1 processing units. To assign a processing capacity representing 50% of a processor, 0.50 processing units are specified on the HMC.

On a system with two processors, a maximum of 2.0 processing units can be assigned to a partition. After a partition is activated, processing capacity is usually referred to as *capacity entitlement* or *entitled capacity*. Figure 5-3 shows a graphical view of the definitions of processor capacity.

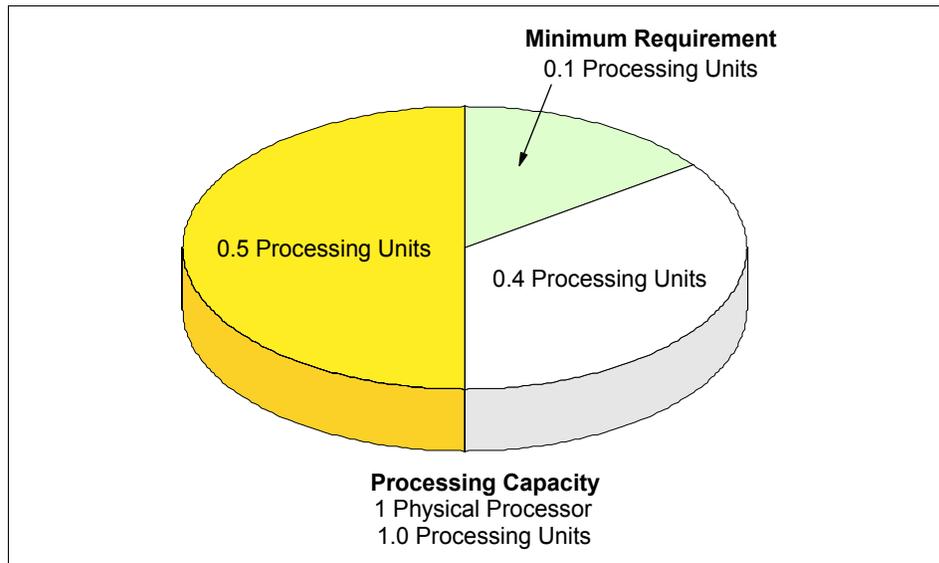


Figure 5-3 Processing units of capacity

By default, the number of processing units that you specify is rounded up to the minimum number of virtual processors needed to satisfy the assigned number of processing units. The default settings maintain a balance of virtual processors to processor units. For example:

- ▶ If you specify 0.50 processing units, one virtual processor will be assigned.
- ▶ If you specify 2.25 processing units, three virtual processors will be assigned.

A logical partition will have at least as many virtual processors as its assigned processing capacity. By making the number of virtual processors too small, you limit the processing capacity of an uncapped partition. If you have a partition with 0.50 processing units and one virtual processor, the partition cannot exceed 1.00 processing units because it can run only one job at a time, which cannot exceed 1.00 processing units. However, if the same partition with 0.50 processing units was assigned two virtual processors and processing resources were available, the partition could use an additional 1.50 processing units.

Figure 5-4 on page 100 shows the relationship between two partitions using a shared processor pool of a single physical CPU. One partition has two virtual processors and the other a single one. The figure also shows how the capacity entitlement is evenly divided over the number of virtual processors.

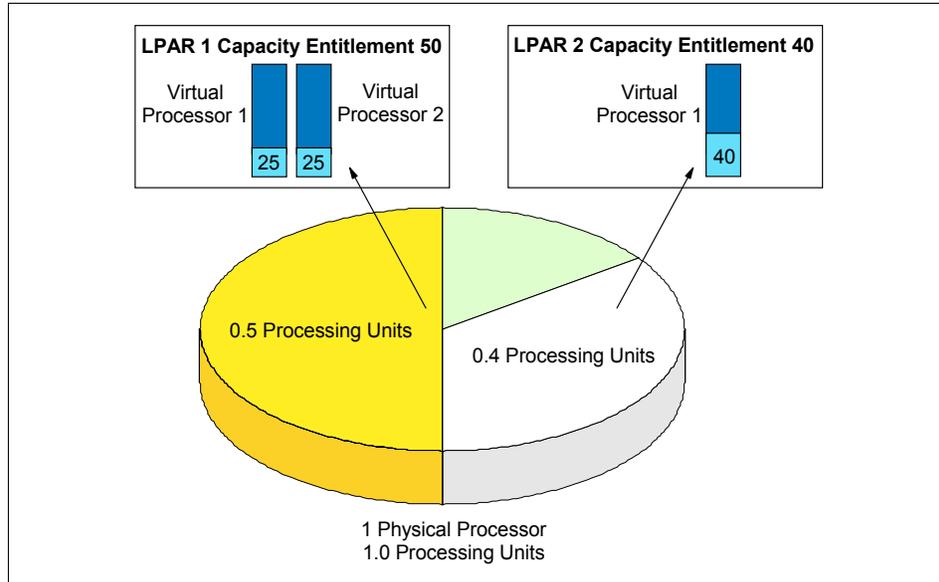


Figure 5-4 Distribution of capacity entitlement on virtual processors

When a partition is started, preference is given to the *desired* value, but this value cannot always be used because there may not be enough unassigned capacity in the system. In that case, a different value is chosen, which must be greater than or equal to the *minimum* capacity attribute. The value that is chosen represents a commitment of capacity that is reserved for the partition. This capacity cannot be used to start another shared partition; otherwise, capacity could be overcommitted.

The entitled processor capacity is distributed to the partitions in the sequence in which the partitions are started. For example, consider a shared pool that has 2.0 processing units available. Partitions 1, 2, and 3 are activated in sequence:

- ▶ Partition 1 activated
Min. = 1.0, max = 2.0, desired = 1.5
Allocated capacity entitlement: 1.5
- ▶ Partition 2 activated
Min. = 1.0, max = 2.0, desired = 1.0
Partition 2 cannot be activated because the minimum capacity is not met.
- ▶ Partition 3 activated
Min. = 0.1, max = 1.0, desired = 0.8
Allocated capacity entitlement: 0.5

The maximum value is used only as an upper limit for dynamic operations.

Capped and uncapped mode

In the configuration of Micro-Partitioning, two types are available, *capped* and *uncapped*. The difference is in defining the ability of a partition to use extra capacity available in the system. If a processor donates unused cycles back to the shared pool, or if the system has idle capacity (because there is not enough workload running), the extra cycles may be used by other partitions, depending on their type and configuration.

Capped mode The processing capacity never exceeds the assigned processing capacity.

Uncapped mode The processing capacity may be exceeded when the shared processing pool has available resources.

A *capped partition* is defined with a hard maximum limit of processing capacity. That means that it cannot go over its defined maximum capacity in any situation, unless you change the configuration for that partition (either by modifying the partition profile or by executing a dynamic LPAR operation). Even if the system is otherwise idle, the capped partition cannot exceed its entitled capacity.

With an uncapped partition, you must specify the uncapped weight of that partition. If multiple uncapped logical partitions require idle processing units, the managed system distributes idle processing units to the logical partitions in proportion to each logical partition's uncapped weight. The higher the uncapped weight of a logical partition, the more processing units the logical partition gets.

Figure 5-5 on page 102 shows the usage of a capped partition of the shared processor pool. Partitions using the capped mode are not able to assign more processing capacity from the shared processor pool than the capacity entitlement will allow.

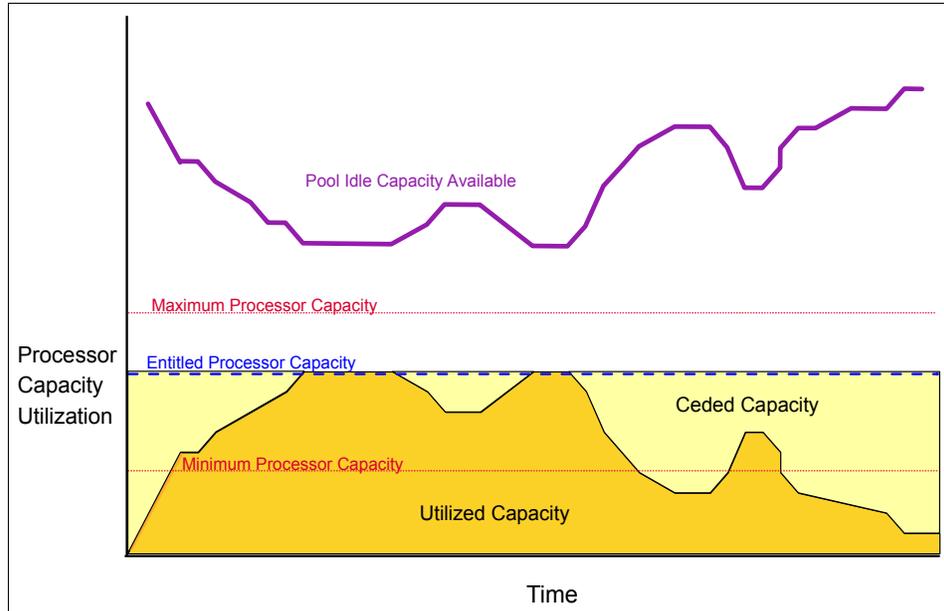


Figure 5-5 Capped shared processor partitions

Figure 5-6 on page 103 shows the usage of the shared processor pool by an uncapped partition. The uncapped partition can assign idle processing capacity if it needs more than the entitled capacity.

In general, the value of the *minimum*, *desired*, and *maximum* virtual processor attributes should parallel those of the minimum, desired, and maximum capacity attributes in some fashion. A special allowance should be made for uncapped partitions, as they are allowed to consume more than their entitlement.

If the partition is uncapped, then the administrator may want to define the desired and maximum virtual processor attributes $x\%$ above the corresponding entitlement attributes. The exact percentage is installation-specific, but 25% to 50% is a reasonable number.

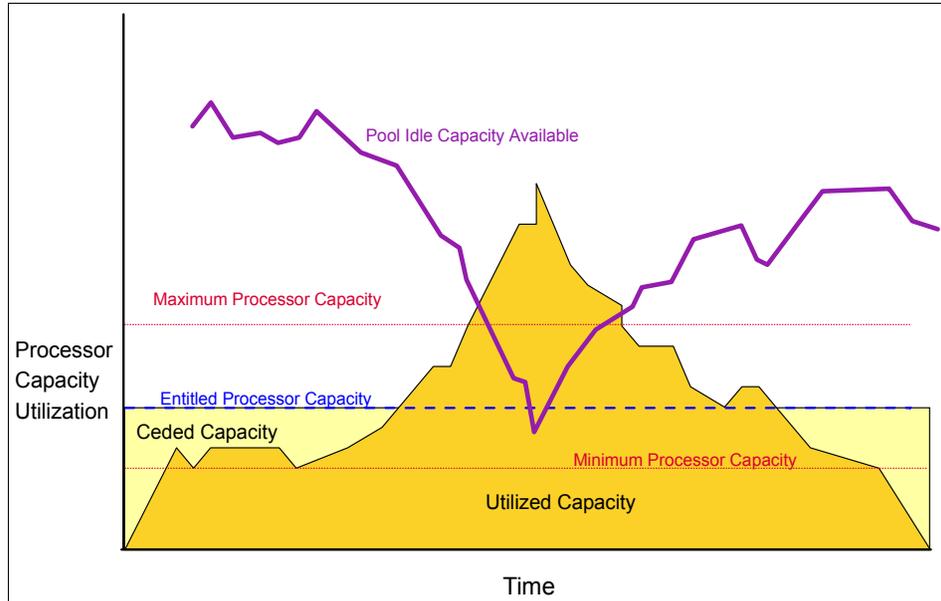


Figure 5-6 Uncapped shared processor partition

Table 5-1 shows several reasonable settings for number of virtual processors, processing units, and the capped and uncapped mode.

Table 5-1 Reasonable settings for shared processor partitions

Min VPs ^a	Desired VPs	Max VPs	Min PU ^b	Desired PU	Max. PU	Capped
1	2	4	0.1	2.0	4.0	Y
1	3 or 4	6 or 8	0.1	2.0	4.0	N
2	2	6	2.0	2.0	6.0	Y
2	3 or 4	8 or 10	2.0	2.0	6.0	N

a - Virtual processors, b - Processing units

Operating systems and applications running in shared partitions need not be aware that they are sharing processors. However, overall system performance can be significantly improved by minor operating system changes. AIX 5L V5.3 provides support for optimizing overall system performance of shared processor partitions.

Weight for uncapped partitions

You can determine how the POWER Hypervisor should distribute the extra cycles between different uncapped partitions. When configuring an uncapped partition on the HMC, you are presented with an option to set the *variable capacity weight*. It is a number between 0 and 255 that represents the relative share of extra capacity that the partition is eligible to receive. For any uncapped partition, its eligible share is calculated by dividing its own variable capacity weight by the sum of the variable capacity weights for all uncapped partitions.

The default uncapped weight for uncapped logical partitions is *128*. A partition's share is computed by dividing its variable capacity weight by the sum of the variable capacity weights for all uncapped partitions. Setting the uncapped weight to 0 will result in the logical partition being treated as capped. A logical partition with an uncapped weight of 0 cannot use more processing units than those that are committed to the logical partition.

5.2.1 Virtual processor dispatching

There are four logical states that a virtual processor could be in:

Running	Currently dispatched onto a physical processor.
Runnable	Currently not running, but ready to run. The queue of runnable virtual processors represents a <i>first-in, first out</i> (FIFO) queue for selecting the next virtual processor to be dispatched to a physical processor.
Not-runnable	The state of a virtual processor that has released its cycles either by calling H_CEDE or H_CONFER POWER Hypervisor calls. In the cede case, either an interrupt or an H_PROD call from another virtual processor makes this virtual processor runnable again. In the confer case, a H_PROD call or a dispatch cycle granted to the conferred targets will make the virtual processor runnable again.
Entitlement expired	The state of all virtual processors that have received full entitlement for the current dispatch window.

The POWER Hypervisor schedules virtual processors from a set of physical processors that is called the *pool*. There are up to three pools of physical processors on a system: one is for *dedicated processor partitions*, one is for *shared processor pool*, and the other is for *unallocated* processors.

Each partition is presented with the resource abstraction for its partition and other required information through the *Open Firmware device tree*. The device tree is created by firmware and copied into the partition before the operating system is started. Operating systems receive resource abstractions and

participate in the partitioning model by making POWER Hypervisor calls at key points in their execution as defined by the model.

For Micro-Partitioning, the POWER Hypervisor schedules virtual processors from a set of physical processors in the *shared processor pool*. By definition, these processors are not associated with dedicated partitions.

In Micro-Partitioning there is no fixed relationship between virtual processors and physical processors. The POWER Hypervisor can use any physical processor in the shared processor pool when it schedules a virtual processor. By default, it attempts to use the same physical processor that was last used by the partition, but this cannot always be guaranteed. The POWER Hypervisor uses the concept of a home node for virtual processors, enabling it to select the best available physical processor from a *processor affinity* perspective for the virtual processor that is to be scheduled.

Processor affinity As an application runs, the instruction cache fills with the instructions and the data cache fills with the data associated with the application. If the application is momentarily preempted by another higher priority task that only executes for a short period of time, the caches may still have instructions and data related to the application that was preempted when it is redispached, and it would be optimal for that application to be dispatched back onto that processor.

Affinity is actively managed by the POWER Hypervisor because each partition has a completely different context. Currently, there is one shared processor pool, so all virtual processors are implicitly associated with the same pool.

Operating systems use their virtual processors by being dispatched in time-sliced manner onto physical processors under the control of the POWER Hypervisor, much like the operating system time slices software threads.

The POWER Hypervisor utilizes the HDEC register to drive its partition dispatcher. HDEC is a clock interrupt source utilized by the POWER Hypervisor to preempt a dispatched partition and regain control of the physical processor. For the details of HDEC, refer to “POWER Hypervisor Decrementer” on page 33.

Dispatch wheel

The POWER Hypervisor uses the architectural metaphor of a *dispatch wheel* with a fixed timeslice of 10 milliseconds (1/100 seconds) to guarantee that each virtual processor receives its share of the entitlement in a timely fashion. This means that the entitled processing unit of each partition is distributed to one or more virtual processors, which will then be dispatched onto physical processors in a time-slice manner during every 10 ms dispatch wheel. The time that each

virtual processor gets dispatched depends on the number of virtual processors and the *entitled processing capacity* that has been assigned to that partition from HMC by the system administrator. When a partition is completely busy, the partition entitlement is evenly distributed among its online virtual processors.

The POWER Hypervisor manages a dispatch wheel for each physical processor in the shared pool. Figure 5-7 illustrates the assignment of virtual processors to a physical processor.

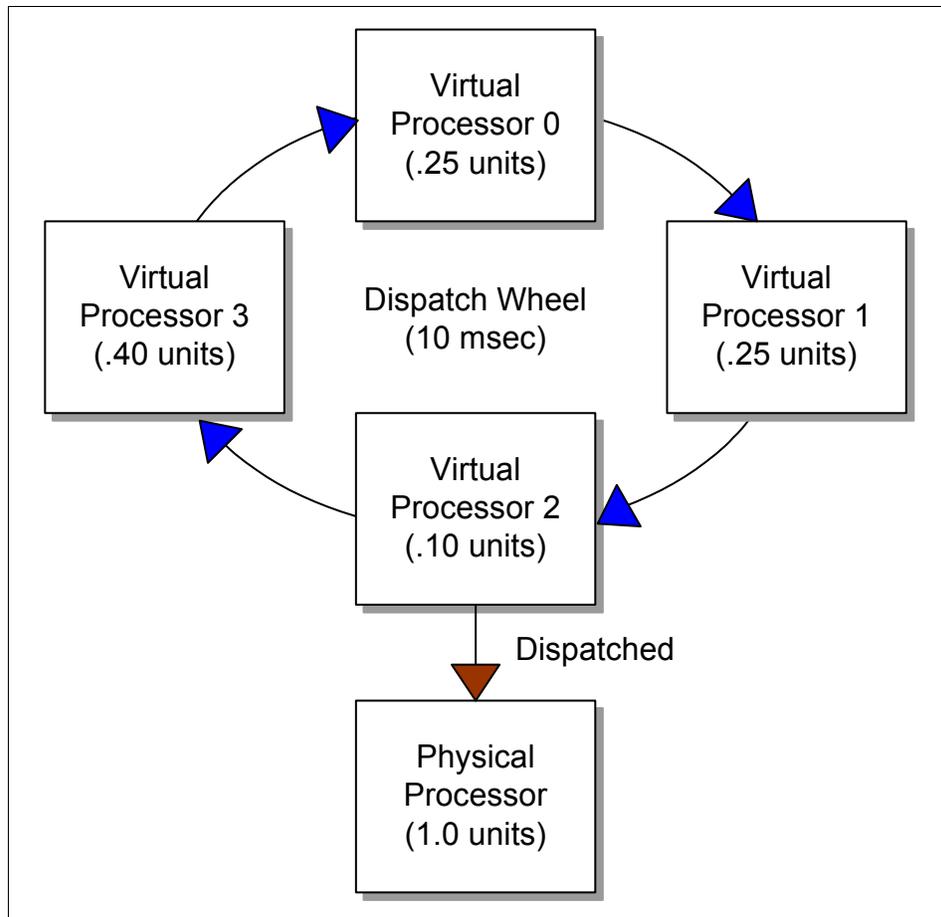


Figure 5-7 Dispatch wheel

Initially, if the available physical processor entitlement in the whole system meets the requirement defined for the partition, the partition will be started and the POWER Hypervisor will begin to dispatch the required virtual processors to each physical processor evenly. For every subsequent time slice, the POWER Hypervisor does not guarantee that all virtual processors will be dispatched in

the same order, nor does it guarantee that the virtual processors in a given partition are dispatched together. However, it does ensure that every partition gets its entitlement if needed.

The dispatch wheel works the same way when simultaneous multithreading is enabled for a processor or group of processors. The two logical processors (hardware threads) are dispatched together whenever the POWER Hypervisor schedules the virtual processor to run. The amount of time that each virtual processor runs is split between the two logical processors.

Figure 5-8 shows a diagram for a case when simultaneous multithreading is enabled.

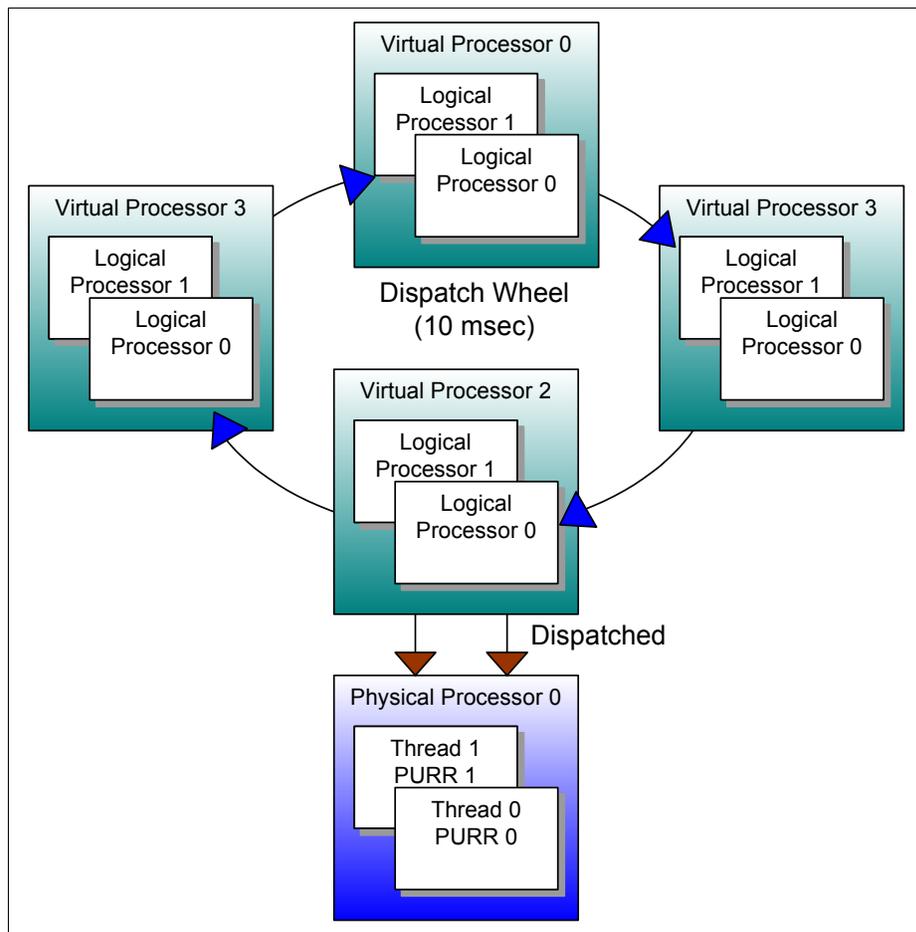


Figure 5-8 Dispatch wheel when simultaneous multithreading is enabled

Processor affinity policy

The POWER Hypervisor attempts to dispatch work in a way that maximizes processor affinity. When the POWER Hypervisor is dispatching a virtual processor, it first attempts to use the same physical processor this virtual processor was previously dispatched on. Otherwise, it will be dispatched to the first available processor in the following order, same chip, same multi-chip module (MCM), or same node. When a physical processor becomes idle, the POWER Hypervisor will look for a virtual processor that requires processing time. Priority will be given to virtual processors in this order:

1. Virtual processors that have an affinity for that processor
2. Virtual processors with no affinity to a real processor
3. Virtual processors that are uncapped

In IBM AIX 5L V5.3, the **mpstat** command using the **-d** flag displays detailed affinity and migration statistics for AIX 5L threads and dispatching statistics for logical processors (Example 5-1).

Example 5-1 The `mpstat -d` command

```
# mpstat -d
System configuration: lcpu=4 ent=0.5

cpu   cs    ics  bound  rq  push  S3pull  S3grd  S0rd  S1rd  S2rd  S3rd  S4rd  S5rd  ilcs  vlcs
0 68598 38824 0 0 0 0 0 95.6 0.0 0.0 4.4 0.0 0.0 174110 237393
1 291 244 0 0 0 0 0 90.9 7.4 0.0 1.7 0.0 0.0 1092 237759
2 54514 30174 1 1 0 0 0 94.0 0.1 0.0 6.0 0.0 0.0 2756 71779
3 751 624 0 0 0 0 0 91.3 2.9 0.0 5.8 0.0 0.0 1192 72971
ALL 124154 69866 1 1 0 0 0 94.8 0.1 0.0 5.1 0.0 0.0 89575 309951
```

The POWER Hypervisor dispatch affinity domains are defined as follows, and statistics for virtual processor dispatch across these domains is given by the **mpstat** command.

- cpu** Logical CPU (processor) number.
- cs** The number of context switches.
- ics** The number of involuntary context switches; typically caused by the thread's time slice expiring.
- bound** Total number of threads bound to a particular processor.
- rq** The number of threads on the run queue.
- push** The number of thread migrations to other processors due to starvation load balancing.
- S3pull** Number of thread migrations outside the S3rd affinity domain due to idle stealing.
- S3grd** Number of dispatches from the global run queue outside the S3rd affinity domain.

- S0rd** The process redispach occurs within the same logical processor. This happens in the case of simultaneous multithreading enabled systems.
- S1rd** The process redispach occurs within the same physical processor, among different logical processors. This involves sharing of the L1, L2, and L3 cache.
- S2rd** The process redispach occurs within the same processor chip, but among different physical processors. This involves sharing of the L2 and L3 cache.
- S3rd** The process redispach occurs within the same MCM module, but among different processor chips.
- S4rd** The process redispach occurs within the same central processing complex (CPC) plane, but among different MCM modules. This involves access to the main memory or L3-to-L3 transfer.
- S5rd** The process redispach occurs outside of the CPC plane.
- ilcs** Total number of involuntary logical CPU context switches.
- vlcs** Total number of voluntary logical CPU context switches.

As previously stated, the POWER Hypervisor always tries first to dispatch the virtual processor onto the same physical processor that it last ran on and, depending on resource utilization, will broaden its search out to the other processor on the POWER5 chip, then to another chip on the same MCM, then to a chip on another MCM.

Operating system support

In general, operating systems and applications do not have to be aware that they are sharing processors in a Micro-Partitioning environment. However, overall system performance can be improved significantly by minor operating system changes. The main issue here is that the POWER Hypervisor cannot distinguish between the operating system doing useful work such as numerical computations and non-useful work such as spinning while waiting for a lock to be released. The result is that the operating system may waste much of its entitlement doing nothing of value.

AIX 5L V5.3 provides support for optimizing overall system performance of Micro-Partitioning. These optimizations are built around the idea that an operating system can provide hints to the POWER Hypervisor about scheduling. For example, an operating system can signal to the POWER Hypervisor when it is no longer able to schedule work and yield the remaining time slice. This results in better utilization of the physical processors in the shared processor pool.

The dispatch mechanism may utilize POWER Hypervisor calls to communicate between the operating system and the POWER Hypervisor. The major three

POWER Hypervisor calls used by operating systems are H_CEDE, H_CONFER, and H_PROD. For the definition of POWER Hypervisor calls, refer to Table 4-1 on page 81.

When the operating system detects an inability to utilize processor cycles, it may cede or confer its cycles back to the POWER Hypervisor, enabling it to schedule another virtual processor for the remainder of the dispatch cycle. Reasons for a cede or confer may include the operating system entering its idle task or an application entering a spin loop to wait for a resource to free. There is no concept of credit for cycles that are ceded or conferred. Entitled cycles not used during a dispatch interval are lost.

A virtual processor that has ceded cycles back to the POWER Hypervisor can be reactivated using a H_PROD POWER Hypervisor call. If the operating system running on another virtual processor within the logical partition detects that work is available for one of its idle processors, it can use H_PROD to signal the POWER Hypervisor to make the virtual processor runnable again. Once dispatched, this virtual processor would resume execution at the return from the H_CEDE POWER Hypervisor call.

Dispatching example

Table 5-2 shows an example configuration that will be used to illustrate dispatching of virtual processors. In this Micro-Partitioning example, three logical partitions share two physical processors, and all partitions are capped.

Table 5-2 Micro-Partitioning definition:

LPAR	Capacity entitlement	Virtual processors	Capped or uncapped?
1	0.8	2	capped
2	0.2	1	capped
3	0.6	3	capped

Figure 5-9 on page 111 shows each of these partitions running during two POWER Hypervisor dispatch windows.

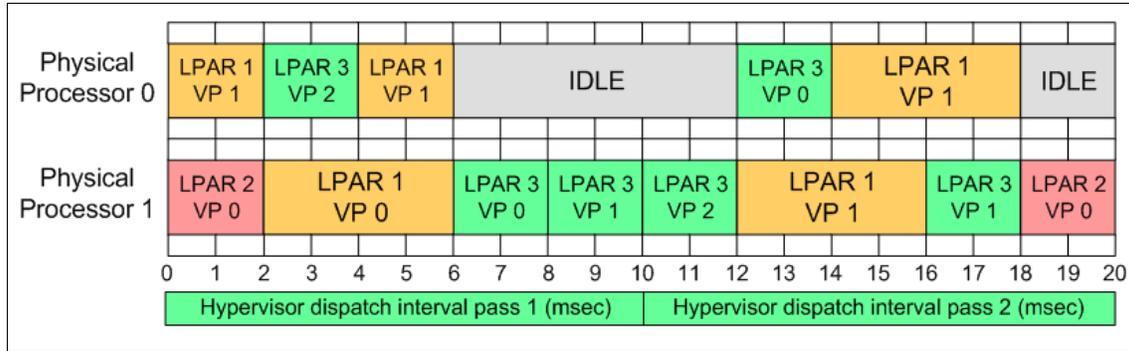


Figure 5-9 Dispatching processors in Micro-Partitioning

Logical partition 1 (LPAR1) is defined with an entitlement capacity of 0.8 processing units and two virtual processors (VP0 and VP1). This means that the partition is entitled to 80% of physical processor capacity from the shared processor pool for each 10 ms dispatch window. The figure shows that the workload is evenly distributed (40% each) between the two physical processors. Note that it is possible for a virtual processor (for example, VP1) to be dispatched more than once during a dispatch interval. The figure shows LPAR1 with VP1 running for two cycles on physical processor 0; LPAR3 with VP0 running for the next two cycles; then LPAR1 is redispached. This may happen if the operating system confers (H_CONFER) cycles and then is reactivated by the POWER Hypervisor call, H_PROD.

Logical partition 2 (LPAR2) is configured with one virtual processor (VP0) and a capacity of 0.2 processing units, entitling it to use 20% of the physical processor resources during each dispatch interval. In this example, the virtual processor dispatched during the two dispatch wheels is assigned to the same physical processor according to the affinity policy.

Logical partition 3 (LPAR3) is configured with three virtual processors (VP0, VP1, and VP2) and has an entitled capacity of 0.6 processing units. Each virtual processor receives 20% of a physical processor in each dispatch interval, but in the case of VP0 and VP2, the physical processor they run on is changed in the two dispatch intervals. The POWER Hypervisor does attempt to maintain physical processor affinity when dispatching virtual processors. As described previously, dispatch logic will always attempt to dispatch the virtual processor onto the same physical processor it last ran on. Depending on resource utilization, it will broaden its search out to the other processor on the POWER5 chip, then to another chip on the same MCM, then to a chip on another MCM.

Tracing virtual processor dispatch

The dispatching of virtual processors by the POWER Hypervisor does not involve the operating system running in the partition. The operating system cannot directly monitor the rate or characteristics of context switching from the POWER Hypervisor. However, there is a communication area that is shared between the POWER Hypervisor and each virtual processor in a partition so that an operating system such as AIX 5L V5.3 can implement tracing of the virtual processor context switching.

The trace facility in AIX 5L V5.3 supports the trace hook value of 419. This trace hook represents the information that is available about context switching. Example 5-2 shows how a system administrator or systems programmer can trace virtual processor dispatching.

Example 5-2 Tracing virtual processor dispatching

```
# trace -aj 419
# trcstop
# trcrpt
ID  PROCESS NAME  CPU  PID  TID  I  SYSTEM CALL  ELAPSED_SEC  DELTA_MSEC  APPL  SYSCALL  KERNEL  INTERRUPT
419 -229498-      3   229498  876731          I  SYSTEM CALL  0.020218416  5.926309          cpu preemption data Preempted vProcIndex=0005
                                     rtrdelta=0000 enqdelta=17321 exdelta=202DC
                                     start wait=2D33E3B52A87 end wait=2D33E3CE5F7B
                                     SRR0=000000000017B770
                                     SRR1=8000000000009032

419 -229498-      2   229498  819359          I  SYSTEM CALL  0.020325289  0.101956          cpu preemption data Preempted vProcIndex=0004
                                     rtrdelta=0000 enqdelta=1732A3 exdelta=202DB
                                     start wait=2D33E3B529FC end wait=2D33E3CE5F7A
                                     SRR0=000000000017B74C
                                     SRR1=8000000000009032
```

The time when a virtual processor is removed from a physical processor is encoded in the trace hook as well as in the start wait field. The time that the virtual processor is redispached is encoded as the end wait field. We can see the statistics from the following sequence of an AIX trace.

Some measure of virtual processor to physical processor affinity is possible as well. The trace hook shows an index to the physical processor (vProcIndex field). The index is fixed over time, and not necessarily an indicator of the physical processor number of the system. If a virtual processor number is dispatched to the same vProcIndex as the previous dispatch, affinity is maintained.

5.2.2 Phantom interrupts

In order to speed the processing of I/O interrupts, the delivery of interrupts to physical processors can happen without direct execution of the POWER Hypervisor. Rather, interrupts are delivered by the hardware directly to a physical processor running a partition's virtual processor. In the event an interrupt is delivered to a virtual processor for a partition that does not own the hardware, the

interrupt is ignored by the currently executing partition and is queued by the POWER Hypervisor for servicing by the correct partition. An interrupt that is mistakenly delivered to an incorrect partition is termed a *phantom interrupt*.

Figure 5-10 shows the interrupt servicing logic and can be described as follows. If the processor is idle (running in POWER Hypervisor mode), then the POWER Hypervisor handles the interrupt and identifies the correct partition to make ready to run. The interrupt is queued to be delivered when a virtual processor for the correct partition runs. If the physical processor is running a virtual processor for a partition, the virtual processor receives the I/O interrupt. The operating system running on the virtual processor calls the POWER Hypervisor (via the H_XIRR call) to determine the interrupt source. If the interrupt source is not for this partition, the interrupt is queued in the POWER Hypervisor for delivery to the correct partition. If the interrupt source is for this partition, the correct device driver is invoked.

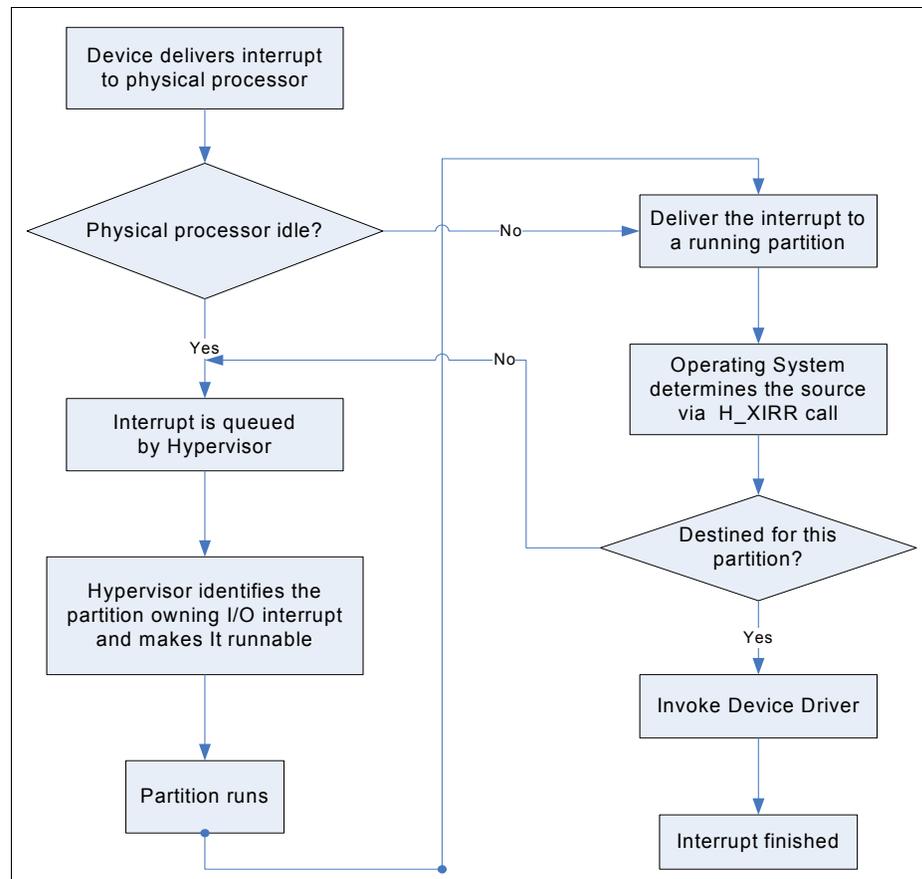


Figure 5-10 Interrupt servicing in POWER5 systems

For dedicated processor partitions, phantom interrupts are extremely rare, as the I/O hardware can be relatively certain of which physical processors a partition is running on. In Micro-Partitioning, phantom interrupts happen with statistical likelihood. The latency for interrupt servicing can become extended, due to nuances of partition dispatch. But normally interrupt latency will have an upper limit of the duration of the dispatch wheel (10 ms).

Under normal and even heavy I/O load, the performance degradation of handling phantom interrupts is very low. That is because the CPU cost to process a phantom interrupt is small. In order to understand the rate of phantom interrupts, we allow extraction of their rates by the AIX 5L command **mpstat**. In Example 5-3, the *ph* field shows the number of phantom interrupts for each logical processor.

Example 5-3 mpstat command

```
(localhost:) # mpstat -i 1 100
```

```
System configuration: lcpu=8   ent=1.0
```

cpu	mpcs	mpcr	dev	soft	dec	ph
0	0	1	1939	47	183	2735
1	0	1	1946	1	83	2225
2	0	1	1815	1	100	2912
3	0	1	1870	1	89	1510
4	0	1	2000	11	102	3096
5	0	1	1951	1	100	1715
6	0	1	2093	1	112	1942
7	7	0	2101	0	100	2527
ALL	7	7	15715	63	869	18662
0	0	1	1767	49	148	4131
1	0	1	1809	1	100	1843
2	0	1	1951	1	101	4062
3	0	1	1974	1	85	2602
4	0	1	1918	11	101	2264
5	0	1	1868	1	101	3492
6	0	1	1980	1	110	5461
7	7	0	1971	0	100	1841
ALL	7	7	15238	65	846	25696

You can trace phantom interrupts in AIX 5L V5.3 using the **trace** command with trace hook values of 492 and 47F. In Example 5-4 on page 115, the tracing of hook values 100 and 200 are included to show when the interrupt occurs and when the preempted process (in this case, *wait*) resumes.

Example 5-4 AIX 5L trace for phantom interrupt

```
# trace -aj 100,200,492,47F
# trcstop
# trcrpt
ID  PROCESS CPU  PID      ELAPSED_SEC DELTA_MS  APPL SYSCALL KERNEL INTERRUPT
100  wait   0   8197      0.340638   0.337172  I/O INTERRUPT iar=2C514 cpuid=00
492  wait   0   8197      1  0.340638   0.337173  h_call:start H_XIRR iar=3B6B100
                                           p1=1857D50 p2=234E70 p3=9C6B508A638
492  wait   0   8197      1  0.340642   0.337177  h_call:end H_XIRR iar=3B6B100 rc=0000
47F  wait   0   8197      1  0.340642   0.337177  phantom interrupt cpuid=00
200  wait   0   8197      1  0.340643   0.337178  resume wait iar=2C514 cpuid=00
```

5.3 Performance considerations

Micro-Partitioning adds a layer of abstraction by the creation of virtual processors. This virtualization promotes greater flexibility and increased processor utilization. However, in some cases, improper system configuration utilizing Micro-Partitioning can negatively affect performance. The intention of this book is to assist system administrators with avoiding those pitfalls.

The impact on performance can be positive or negative and may be defined or measured in a number of different ways:

- ▶ A decrease in maximum throughput for a fixed entitlement due to workloads in other partitions. The impact can be measured in partitions with high CPU utilization levels.
- ▶ A change in processing time used by a partition to complete a fixed task due to workloads in other partitions. This impact may be measured at any utilization level.
- ▶ A change in processing time for a software thread or process to complete a fixed task due to workloads in other partitions. This impact may be measured at any utilization level.

With Micro-Partitioning, the impact on performance tends to be isolated to each partition. In other words, there is no unmeasured partition or time unaccounted for that reflects the impact on performance. Rather, the impact on performance appears as changing amounts of CPU time to complete work. The changing amounts can be either positive or negative. Better processor utilization through load balancing, efficient programs and Micro-Partitioning aware operating systems can have a positive impact. However, high CPU utilization by all active partitions can be negative. In most cases, the impact on performance only occurs when multiple partitions are running on the system. There is very little performance impact when running a single partition by itself.

Note: The impact on performance when implementing Micro-Partitioning can be both positive and negative.

For example, in a test using Network File System (NFS), throughput was measured on four dedicated processor partitions, each with one physical processor. The result was compared to a Micro-Partitioning environment implementing four partitions with 1.0 entitlement per partition, each essentially running on its own physical processor. The throughput in each partition was the same in both cases. Processor usage was about 2% higher in the case of Micro-Partitioning.

5.3.1 Micro-Partitioning considerations

In Chapter 3, “Simultaneous multithreading” on page 41, simultaneous multithreading was defined by the POWER5 architecture as having two hardware threads of execution occurring simultaneously. These hardware threads could be from independent programs or programs that are multi-threaded. Each hardware thread is considered a unique processor by the operating system. Except for the few differences mentioned here, the behavior of simultaneous multithreading is independent of whether the partition is configured with dedicated processors or Micro-Partitioning.

The purpose of simultaneous multithreading is to increase the number of instructions that can execute in a unit of time through the microprocessor. Each thread uses microprocessor resources such as registers to execute instructions. Under almost all circumstances, there are sufficient resources to have more throughput with two threads executing than with a single thread executing. However, the simultaneous execution of two threads results in the sharing of some microprocessor resources. This implies that the time to execute a fixed number of instructions by a single thread may increase when two threads are active in the processor core. However, over that same measured interval, the total instructions executed by both threads normally will be greater than those that could be executed by a single thread. If a partition is executing a low-to-medium CPU utilization, there may not be enough software threads or software processes to keep all of the hardware threads busy. In this case, it is beneficial to be able to apply all of the microprocessor resources to a single thread.

Note: AIX 5L V5.3 classifies the two hardware threads as primary and secondary. Dispatch preference is given to the primary thread.

AIX 5L V5.3 classifies the two threads on a microprocessor as a primary thread and a secondary thread. In a partition with simultaneous multithreading enabled,

threads of a process are dispatched to the primary hardware threads before the secondary threads. This helps to optimize performance for single-threaded applications running on a microprocessor. Because the secondary threads do not get work to execute, they go into a snooze state and the primary thread runs at almost single-thread performance.

In dedicated processor partitions, the POWER Hypervisor can dynamically transition the processor from simultaneous multithreading to single-threaded when requested by the operating system. When a single hardware thread running on a processor becomes idle, the processor is changed to single-threaded mode, and the running thread benefits from single-threaded performance. When the other thread is runnable again, the processor returns to simultaneous multithreading mode and runs both threads.

Micro-Partitioning does not support automatic changing between simultaneous multithreading and single-threaded; this is controlled by the `smtctl` command. In Micro-Partitioning, if a hardware thread becomes idle, it spins in an idle loop at low priority. This enables the other running thread to get a large part of the processing capacity to itself.

Effect of simultaneous multithreading on processor usage

For a processor to cede its idle cycles to the POWER Hypervisor in the case of Micro-Partitioning, both hardware threads must be idle. If one thread is idle while the other is running, some idle capacity remains in the partition and cannot be given back to the POWER Hypervisor. This effect is noted by comparing the CPU utilization of the partition versus the fraction of its entitlement used.

The behavior is more perceptible when CPU usage within a partition is between 40% and 70% of processing capacity. You can observe this effect by looking at the difference between partition entitlement utilization (processing capacity consumed by the partition) and partition processor utilization (processing capacity consumed by the threads in the partition). The AIX 5L command `vmstat` shows this information in the `ec` and `pc` columns. Figure 5-11 on page 118 illustrates this effect as observed when running a Java™-based application server with WebSphere® and DB2®.

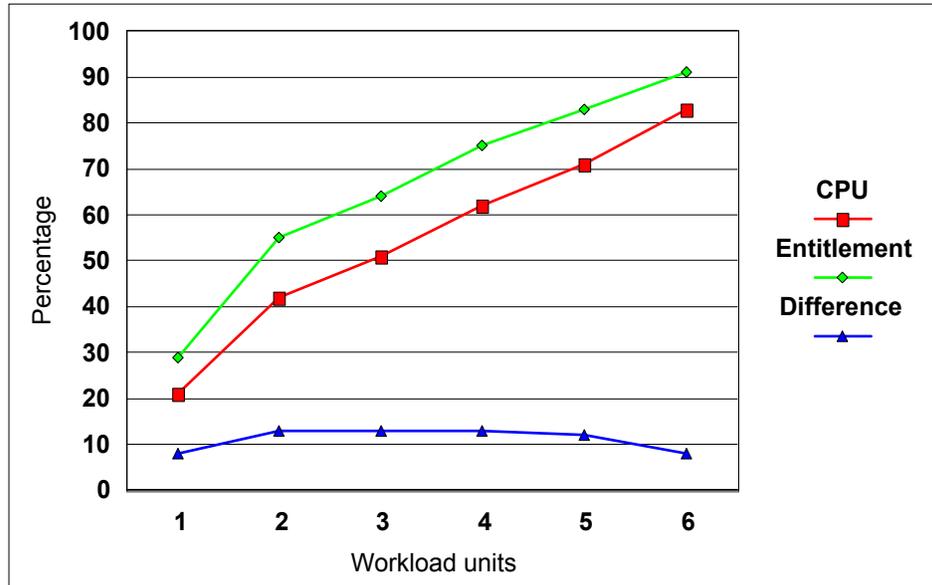


Figure 5-11 Effects of simultaneous multithreading

As partition utilization increases, this effect decreases because the hardware threads get more work to be done so the idle time for each thread decreases. Obviously, this effect is not present in partitions running in single-threaded mode.

Micro-Partitioning effects on caching

On POWER5 systems, the two processor cores on the same chip share the L2 and L3 caches. A system running with dedicated partitions and with each core assigned to a different partition, the caches are still shared. This results in the two cores competing for cache capacity. Naturally, each core can access only the cache lines correspondent to its memory addresses. But, the competition for cache capacity has direct impact on the performance each microprocessor can achieve.

In Micro-Partitioning, the same situation can occur, with the additional factor that during a given interval a physical processor may have executed code from several different partitions. When a virtual processor is dispatched onto a physical processor, all of the memory addresses are relative to the partition the virtual processor is assigned. Cache usage becomes dependent on the memory access behavior of different applications running on different partitions. The competition for shared caches is a significant factor in Micro-Partitioning performance, as the cache hit ratios for a measured partition may change over time as other partitions run at varying levels of activity.

The POWER Hypervisor is responsible for maintaining affinity between virtual and physical resources in Micro-Partitioning. When dispatching a virtual processor onto a physical processor, the POWER Hypervisor tries to redispach a virtual processor to the same physical processor that it ran on previously. This attempts to maximize cache affinity and reduce the need for reloading data from main memory.

Nevertheless, in Micro-Partitioning there is the potential of having several partitions sharing a processor, resulting in several different memory contexts. Moreover, because of dispatching requirements, a physical processor may not be available when a virtual processor makes the transition from not-runnable to runnable. When a virtual processor is ready to run, the POWER Hypervisor checks whether the physical processor that ran this virtual processor for the last time is idle. If it is busy, then it starts looking in increasing levels of affinity scope for an idle processor (other cores on same chip, other processors within same MCM, and any other processor in the system) until one is found. If no processor is available, the virtual processor is queued onto the runnable queue. Figure 5-12 depicts the flow of actions described.

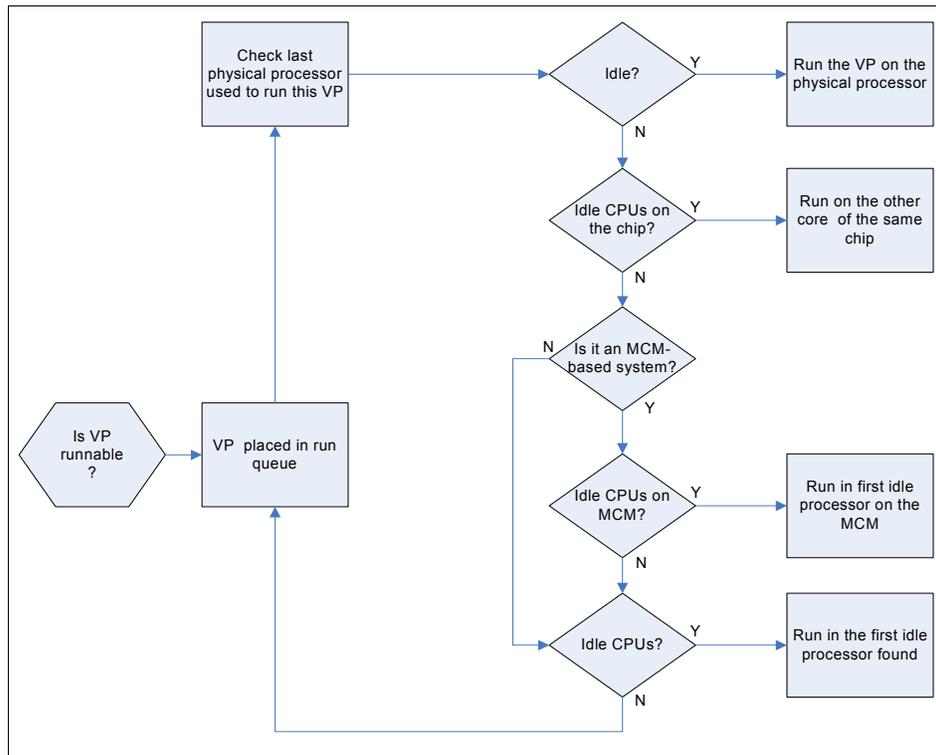


Figure 5-12 Affinity between virtual and physical processors

Even if the virtual processor is dispatched on the same physical processor from its last run, data in cache may have been replaced by previous virtual processors dispatched in the same physical processor.

The amount of leftover cache context depends on the amount of data read from other applications running on the same processor and the ratio of virtual processors to physical processors. If an application running on a virtual processor is memory intensive, data in the caches belonging to other virtual processors is replaced. The caches are reloaded when other virtual processors are later dispatched. Therefore, an application whose performance depends on cache efficiency will be affected when running in a micro-partition along with other partitions that do intensive memory access.

Number of virtual processors

When the number of virtual processors is much larger than the number of physical processors on the system, the time slice given to each virtual processor on the physical processors tends to get smaller. One way to calculate the size of the virtual processor time slice is to divide the partition entitlement by the number of virtual processors. Increasing the number of virtual processors increases the probability that a cache line will be flushed for a virtual processor that is not running, and thus reduces the physical processor's cache efficiency.

When virtual processor capacity is small, the impact on performance of reading data from memory is significantly high, due to the fact that the time to fetch data from memory is constant and the time slice is small for small capacity entitlements. Therefore, the impact is more significant in virtual processors with small capacity.

Keep in mind the purpose of a cache is to hold data that is referenced frequently. If applications running on the system are processing data by reading it, modifying it, and then writing it back to memory, the virtual processor time slice effects mentioned above would be no different than dedicated processor partitions.

The performance impact of increasing cache miss rates in the partition due to competition with other partitions depends on the size of the partition, the number of virtual processors, the nature of the other partitions, and the *type* of application.

A worst-case scenario is where one partition uses the caches moderately and another partition uses the caches extensively. Both partitions run on the same processor. For example, application A, which is composed of small but numerous tasks, fits well in the cache by itself. Application B uses memory heavily for reading and writing large blocks of data. In all cases, each partition has two virtual processors, each with 0.1 processor capacity. The partition running application A is uncapped, and the partitions running application B are capped.

Figure 5-13 shows the results of three test cases.

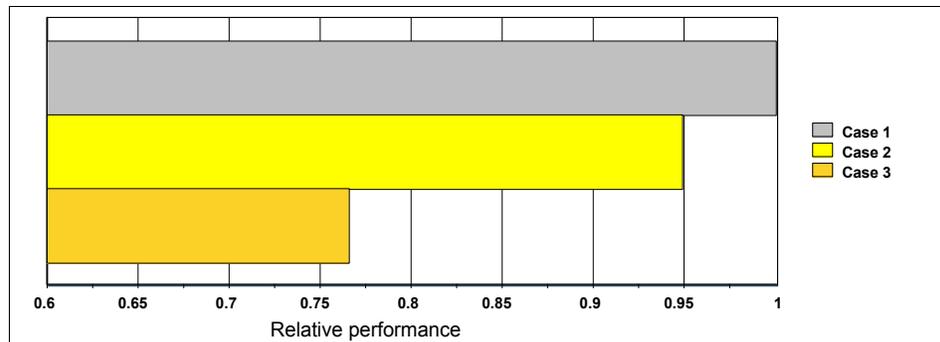


Figure 5-13 Measurements of cache effects in different partitions

Case 1 shows the throughput for application A running without any other partitions active, and serves as a reference point. Case 2 shows the throughput when application A runs in one partition, and one other partition runs application B. Even with the effects of application B reducing caching efficiency, application A runs well, with slightly more than a 5% penalty for sharing the same physical processor with application B. Case 3 shows the throughput for application A while seven other partitions run benchmark B. Due to seven partitions and 14 virtual processors running memory-intensive workloads, we can see that caching efficiency drops to around 76%. This example was an extreme case where the workloads were selected so that the effect on cache usage would have the most impact on performance. Most applications that will run on these systems, including commercial and technical workloads, should observe a smaller impact in performance.

In addition to the changes in maximum throughput achieved by a partition for a fixed entitlement, changes in the CPU time to perform a task will occur. For example, consider a case where a partition requires 100 CPU seconds to complete a database sort when it runs on a processor while the other partitions on the system are relatively idle. When the exact same sort is run again at a time when a number of other partitions are active, the resulting CPU time is 130 seconds. It is important to understand this phenomenon in environments where billing for CPU usage is performed.

5.3.2 Locking considerations

Most operating systems and sophisticated applications use spin locks to serialize read/write access to shared memory. The effectiveness of spin locks is based on the idea that the locks are not held for long periods of time. In Micro-Partitioning, it is possible for a virtual processor holding a lock to be undispached for several

milliseconds. This increases the likelihood of lock contention when a partition is spread over several virtual processors.

For example, in AIX 5L V5.3, kernel locks that run with interrupts disabled benefit from special handling in Micro-Partitioning. They are handled differently from locks that run with interrupts enabled, since having interrupts disabled prohibits the undischarging of the blocked software thread and running another.

Consider the case where a virtual processor owning the lock is not running (for example, it used up the entitled time slice), and there is another virtual processor that needs the same lock to run on the system concurrently. Without optimization, a blocked virtual processor will spin waiting until the lock is released by the owner. To effectively solve this situation without spending unnecessary cycles, the virtual processor waiting for the lock uses the POWER Hypervisor call H_CONFER to give its cycles to the virtual processor owning the lock. The POWER Hypervisor dispatches the lock owner to continue processing and eventually release the lock. Note that in a simultaneous multithreading enabled partition, this mechanism is relatively less effective. If there is heavy locking, running a partition in single-threaded mode may reduce the impact.

Lock contention can be monitored with the AIX 5L trace facility. Example 5-5 shows a case where a thread on virtual processor 0 attempts to acquire a lock. When it determines that there is lock contention with another thread, it confers its processor cycles. After the lock is released, it acquires the lock.

1. Thread on virtual processor 0 attempts to acquire a lock.

Example 5-5 AIX 5L trace of lock contention - step 1

ID	PROCESS NAME	CPU	PID	TID	I	SYSTEM CALL	ELAPSED_SEC	DELTA_MSEC	APPL	SYSCALL	KERNEL	INTERRUPT
112	-229498-	0	229498	1294461			0.033375354	0.000376			lock: dmiss lock	
											addr=F1000600234F0100 lock	
											status=B7060000000000	
											requested_mode=LOCK_SWRITE	
											etern addr=3CCF1EC	
											name=00000000.00000000	

2. After identifying lock contention with another thread, the lock becomes a spin lock.

Example 5-6 AIX 5L trace of lock contention - step 2

112	-229498-	0	229498	700529			0.033376227	0.000019			krlock: cpuid=00 addr=F100060004006B80	action=spin
-----	----------	---	--------	--------	--	--	-------------	----------	--	--	--	-------------

3. Instead of actively spinning, it confers its cycles to CPU 2, which is ahead of it in the lock queue.

Example 5-7 AIX 5L trace of lock contention - step 3

```
112 -229498- 0 229498 700529 0.033382946 0.000019 krlock: cpuid=00 addr=F100060004006B80 action=confer (target cpuid=0002)
```

4. The POWER Hypervisor runs to confer cycles and dispatches the thread on CPU 2.

Example 5-8 AIX 5L trace of lock contention - step 4

```
492 -229498- 0 229498 700529 0.033383307 0.000014 h_call: start H_CONFER iar=17A8F0 p1=0002 p2=52904FF p3=2D33E3F7FA44
419 -229498- 2 229498 819359 0.033598712 0.215405 cpu preemption data Preempted vProcIndex=0004
rtrdelta=0000 enqdelta=4471D exdelta=A66B
start wait=2D33E3F3B790 end wait=2D33E3F8A518
SRR0=000000000017B770
SRR1=8000000000009032
```

5. Virtual CPU 2, which was waiting on the lock, acquires it. After it is no longer needed, it is handed off to the thread on CPU 3, which also was waiting for it.

Example 5-9 AIX 5L trace of lock contention - step 5

```
112 -229498- 2 229498 819359 0.033599436 0.000724 krlock: cpuid=02 addr=F100060004006B80 action=acquire
112 -229498- 2 229498 819359 0.033600580 0.001144 lock: dlock lock addr=F1000600234F0100 lock status=B70800000C809F
requested_mode=LOCK_SWRITE return addr=3CCF1EC
name=00000000.00000000
112 -229498- 2 229498 819359 0.033601444 0.000864 krlock: cpuid=02 addr=F100060004006B80
action=handoff (target cpuid=0003)
419 -229498- 3 229498 876731 0.033602912 0.001468 cpu preemption data Preempted vProcIndex=0005
rtrdelta=0000 enqdelta=446AF exdelta=A670
start wait=2D33E3F3B7FE end wait=2D33E3F8A51D
SRR0=000000000017B74C SRR1=8000000000009032
```

6. Virtual CPUs 2 and 3 continue processing.

Example 5-10 AIX 5L trace of lock contention - step 6

```
254 -229498- 2 229498 819359 0.033603515 0.000603 MBUF m_copydata mbuf=F100061008250C00 offset=0 len=26
cpaddr=F100061001480000
254 -229498- 3 229498 819359 0.033604287 0.000763 MBUF return from m_copydata
254 -229498- 2 229498 819359 0.033604697 0.000410 MBUF m_copydata mbuf=F100061008250C00 offset=26 len=8
cpaddr=F10006100148001A
254 -229498- 2 229498 819359 0.033605015 0.000318 MBUF return from m_copydata
```

7. On the next pass of the POWER Hypervisor dispatch wheel, virtual CPU 0 is dispatched to run again. Note that approximately 7 ms have passed.

Example 5-11 AIX 5L trace of lock contention - step 7

```
419 -229498- 0 229498 700529 0.040380275 .069589 cpu preemption data Unblocked vProcIndex=0007
rtrdelta=AA7D enqdelta=12875A exdelta=2E448
start wait=2D33E3F7FC15 end wait=2D33E40E1234
SRR0=00000000001EB274 SRR1=8000000000009032
492 -229498- 0 229498 700529 0.040381096 0.000821 h_call: end H_CONFER iar=17A8F0 rc=0000
```

8. The thread running on virtual CPU 3 hands off the lock to the thread running on CPU 0, which resumes and acquires the lock.

Example 5-12 AIX 5L trace of lock contention - step 8

112	-229498-	3	229498	819359	0.033601444	0.000864	krlock: cpuid=03 addr=F100060004006B80 action=handoff (target cpuid=0000)
419	-229498-	0	229498	819359	0.040382945	0.001207	cpu preemption data Unblocked vProcIndex=0006 rtrdelta=AA2E enqdelta=12882B exdelta=2E447 start wait=2D33E3F7FB93 end wait=2D33E40E1233 SRR0=00000000001EB274 SRR1=8000000000009032
112	-229498-	0	229498	819359	0.040381738	0.000642	krlock: cpuid=00 addr=F100060004006B80 action=acquire
112	-229498-	0	229498	819539	0.040381882	0.000144	lock: dlock lock addr=F1000600234F0100 lock status=B70800000AB071 requested mode=LOCK_SWRITE return addr=3CCF1EC name=00000000.00000000

Since lock contention is statistical, reducing the number of virtual processors in a partition will usually decrease lock contention just as increasing the number of virtual processors in a partition usually increases lock contention. Environments that have responsiveness issues without full utilization of entitled capacity should be evaluated for possible lock contention issues. AIX 5L kernel lock contention can be analyzed with the use of the **curt** tool.

There are two types of virtual processor context switches, *voluntary* and *involuntary*. Context switches initiated by H_CEDE, H_CONFER, and H_PROD POWER Hypervisor calls are voluntary context switches, while timeslice-related context switches are involuntary. The number of voluntary and involuntary context switches can be extracted from the output fields `vics` and `ilcs` by the AIX 5L command **mpstat**.

The number of virtual processor context switches is important because it is one measure of POWER Hypervisor activity. In some cases it is best to minimize the number of virtual processors in each partition, if there are many partitions activated. On the other hand, if more virtual processors are needed to satisfy peak load conditions and the capacity requirements vary greatly over time, it may be best to take virtual processors offline when they are not needed. In such a situation, the Partition Load Manager may be used to automate this process as a function of load. The detailed explanation of Partition Load Manager is discussed in Chapter 10, “Partition Load Manager” on page 373.

The context switch statistics and the number of POWER Hypervisor calls can also be extracted from AIX 5L high-level commands such as **lparstat** and **mpstat**. The detailed explanation for the commands is discussed in 8.1, “Performance commands” on page 258. Example 5-13 on page 125 illustrates **lparstat**, which shows the name of the POWER Hypervisor call and its elapsed execution time.

Example 5-13 lparstat command

```
# lparstat -H 1 1
```

System configuration: type=Shared mode=Uncapped smt=0n 1cpu=4 mem=256 ent=0.20

Detailed information on Hypervisor Calls

Hypervisor Call	Number of Calls	%Total Time Spent	%Hypervisor Time Spent	Avg Call Time(ns)	Max Call Time(ns)
remove	0	0.0	0.0	1	709
read	0	0.0	0.0	1	376
nclear_mod	0	0.0	0.0	1	0
page_init	4	0.0	0.1	655	1951
clear_ref	0	0.0	0.0	1	0
protect	0	0.0	0.0	1	0
put_tce	0	0.0	0.0	1	1671
xirr	6	0.0	0.1	638	1077
eoi	6	0.0	0.1	447	690
ipi	0	0.0	0.0	1	0
cppr	6	0.0	0.1	265	400
asr	0	0.0	0.0	1	0
others	0	0.0	0.0	1	0
enter	4	0.0	0.0	272	763
cede	357	1.3	98.4	7106	641022
migrate_dma	0	0.0	0.0	1	0
put_rtce	0	0.0	0.0	1	0
confer	0	0.0	0.0	1	0
prod	55	0.0	0.8	391	1168
get_ppp	1	0.0	0.1	1738	2482
set_ppp	0	0.0	0.0	1	0
purr	0	0.0	0.0	1	0
pic	1	0.0	0.0	260	656
bulk_remove	0	0.0	0.0	1	0
send_crq	0	0.0	0.0	1	2395
copy_rdma	0	0.0	0.0	1	0
get_tce	0	0.0	0.0	1	0
send_logical_lan	1	0.0	0.1	2685	4602
add_logical_lan_buf	6	0.0	0.2	686	859

From an operating system point of view, there are software context switches to make a different thread execute. The AIX 5L and Linux command **vmstat** can be used to check context switches at the operating system level.

The example shown in Figure 5-14 on page 126 represents the relative performance of various configurations when executing an NFS benchmark. It shows both the SMP scaling effect and the performance considerations when

running several virtual processors. When the configuration changes from a four-way dedicated processor partition to four 1-way dedicated processor partitions, aggregate throughput is increased by a small margin. This is due to both decreased data movement between processors and locking.

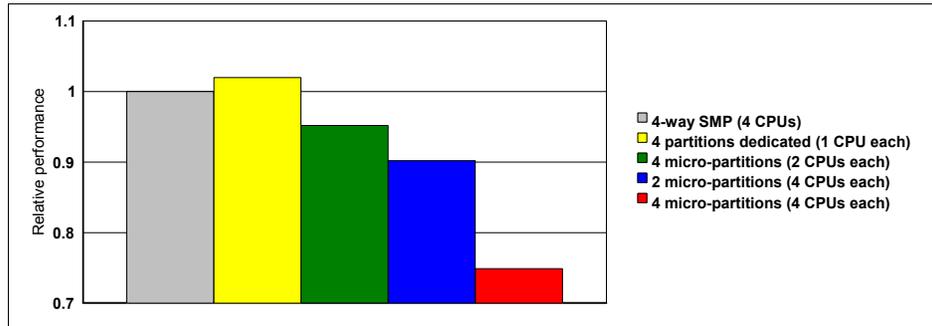


Figure 5-14 The effect of multiple virtual processors in overall performance

With Micro-Partitioning, four partitions with two virtual processors each see a reduction in performance compared to four dedicated processor partitions. Micro-Partitioning using two partitions with four virtual processors was tested and it was found that using four virtual processors increases the lock contention.

Finally, we have four partitions, each with four virtual processors. This case has the lowest performance, due to increasing cache interference and locking requirements. As we increase the number of virtual processors, the relative performance is more affected because of the SMP scaling inside the partition and cache interference due to dispatching the multiple virtual processors in the system.

Note: It is recommended that you have as few virtual processors configured as possible for each partition. It is better to have few virtual processors with higher capacity than a large number of virtual processors each with a small amount of processing power. If it is necessary for expanding the partition to handle more workload, you can add more virtual processors by executing a dynamic LPAR operation.

5.3.3 Memory affinity considerations

In the POWER5 processor-based servers, memory is attached to processor modules and it has the same access characteristics for any processor within the module. This does not differ from POWER4 processor-based servers. Memory and processors that are connected directly are said to fall within a single affinity domain. A processor can access memory attached to its local memory domain

faster (that is, lower latency) than it can access memory attached to other memory domains. AIX 5L V5.3 has optional support for organizing its memory management strategies around these affinity domains.

With memory affinity support enabled, AIX 5L attempts to satisfy page faults from the memory closest to the processor that generated the page fault. This is of benefit to the application because it is now accessing memory that is local to the MCM rather than memory scattered among different affinity domains. This is true for dedicated processor partitions. When using Micro-Partitioning, however, virtual processors may be dispatched on different physical processors during the time a partition is running. As a result, there is no way to implement affinity domains, so memory affinity has no meaning in Micro-Partitioning. Memory is allocated to partitions in a round-robin way, and this tends to reduce processor utilization due to variation in memory allocation.

Important: Applications that benefit from memory affinity should not be implemented in Micro-Partitioning environments.

5.3.4 Idle partition consideration

In Micro-Partitioning, the POWER Hypervisor manages virtual processor dispatching between different partitions so that each partition gets the deserved processing entitlement. In the case of partitions running in the system in the idle state (no work being done), the unused processing cycles may be conferred to other partitions by the POWER Hypervisor, leading to more efficient usage of the CPU resources. There are some activities that consume processor resources even when the partition is idle. System activity such as interrupts and daemons polling for events are some examples of activities that use processing resources. Because of these activities, an idle partition still presents some load to the physical processor. Moreover, the POWER Hypervisor also needs some processing resources to manage these idle partitions and the virtual processors running on them. Normally, a system is not expected to have a large number of idle virtual processors. If there are many, you should analyze whether they are really needed for the work that has to be done. AIX 5L V5.3 implements some timer-management functions to minimize resource utilization by the idle partitions. Performance affected by idle partition management should be minimal. Figure 5-15 on page 128 shows the impact of adding idle partitions to a system running a workload in one uncapped partition.

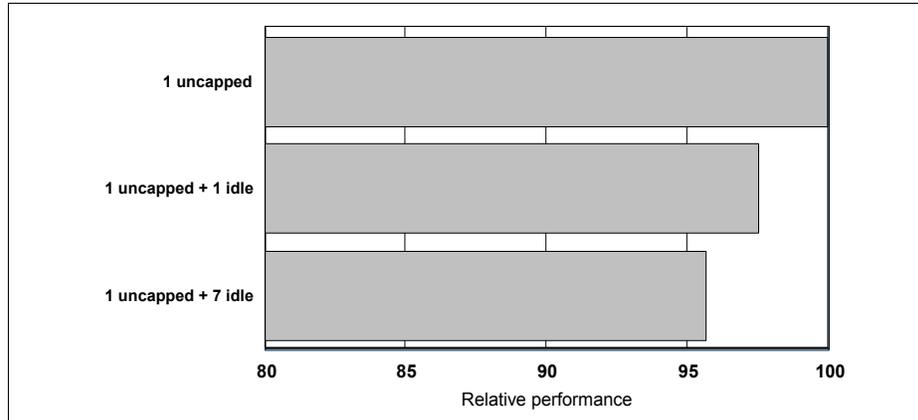


Figure 5-15 Uncapped partition performance example

Because idle partitions are not doing any productive work, to further reduce the performance impact associated with having idle partitions in the system, AIX 5L V5.3 introduces the idea of *slow ticks*. This is an operation mode for idle processors with a reduced timer tick rate. In AIX 5L, a clock interrupt has always occurred every 10 ms (1/100 of a second). This is still the case for busy partitions. For idle partitions, the period of the clock interrupt is changed dynamically to 1/10 of a second. Slow ticks are enabled in partitions running independently as a function of load average on each processor of a system. Note that daemons that run periodically for polling activities, or applications that present similar behavior, can prevent the change to slow ticks; because there are threads running periodically, the partition is not technically idle.

5.3.5 Application considerations in Micro-Partitioning

Applications do not have to be aware of Micro-Partitioning because it is completely transparent from the application perspective. However, there are some considerations that should guide a decision about which applications are suitable for Micro-Partitioning and which are not.

Applications with response time requirements

The Micro-Partitioning environment is dynamic, especially when capped and uncapped partitions are running on the same system.

As stated in 5.2.1, “Virtual processor dispatching” on page 104, the POWER Hypervisor attempts to dispatch all virtual processors in an interval of 10 milliseconds. It does not guarantee, however, that the elapsed time between one dispatch and the next one is fixed. Virtual processors can therefore be dispatched any time between immediately (smallest latency) and 18 ms (largest

latency) after the last dispatch, based on the virtual processor configured capacity and the number of virtual processors in the shared pool. Figure 5-16 illustrates the case for the smallest capacity (10% of a physical processor), where the time slice is 1 millisecond.

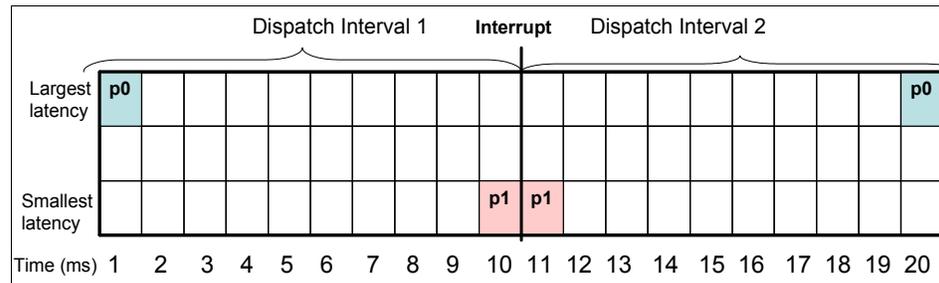


Figure 5-16 Dispatch latencies for virtual processors

Applications that have strong response time requirements for transactions also may not be good candidates for Micro-Partitioning. You can configure the processing capacity inside a partition by tuning for the optimal number of virtual processors, depending on the specific needs for the partition. If an application depends on the individual processing capacity of a processor to run efficiently, it will probably have higher response times when running on a partition with smaller (but more) virtual processors. In order to meet quality of service requirements, care must be taken when configuring the system to support response time critical workloads. For planning purposes, if you decide to run applications that must have predictable response times, or applications that have transactions whose individual performance is a performance factor, you should consider configuring the partition with extra capacity (perhaps 2-5% CPU per partition), in order to compensate for these effects.

Applications in Micro-Partitioning environments, like those running in dedicated processor partitions, see their response times as a function of the CPU utilization. In Micro-Partitioning, if an application is run and the CPU utilization within the partition becomes very high, response time will suffer. The problem is magnified for small virtual processors, since each virtual processor is logically a slower CPU. In laboratory tests, it is frequently difficult to drive small virtual processor partitions to high utilizations on heavy CPU transactions with acceptable quality of service. Applications without strong quality of service requirements are good candidates for small-scale Micro-Partitioning.

Applications with polling behavior

Applications that perform polling may or may not be good candidates for Micro-Partitioning. Because they need to periodically poll to detect whether the resource is available or condition is satisfied, they spend cycles that otherwise

would be available for other partitions (because they are not actually doing work). If the application needs to periodically wake up a thread to do the polling, that means that a virtual processor must be dispatched to run that thread, and spend physical processor cycles, even if it is not producing work. This behavior is the same regardless of the application being run on a partitioned server or not. What might make a differences that in Micro-Partitioning spare cycles can be conferred to other partitions with the help of the POWER Hypervisor.

Applications with low average utilization and high peaks

Applications where average usage of processor resources is low with peaks of usage during a short period of time are good candidates for Micro-Partitioning. More than one application can share the processor resources and run with the required performance, exploiting the benefits of sharing otherwise unused resources. Applications that perform online transaction processing (OLTP) generally fit into this category because they are based on user input, and may vary throughout the day depending on user activity. Usually there are distinct but independent peaks of utilization and an average use significantly lower than the peaks. Examples of such applications are mail servers, Web-based applications, and directory servers.

Figure 5-17 shows the user distribution for a system on a real client scenario. You can clearly identify the peak times.

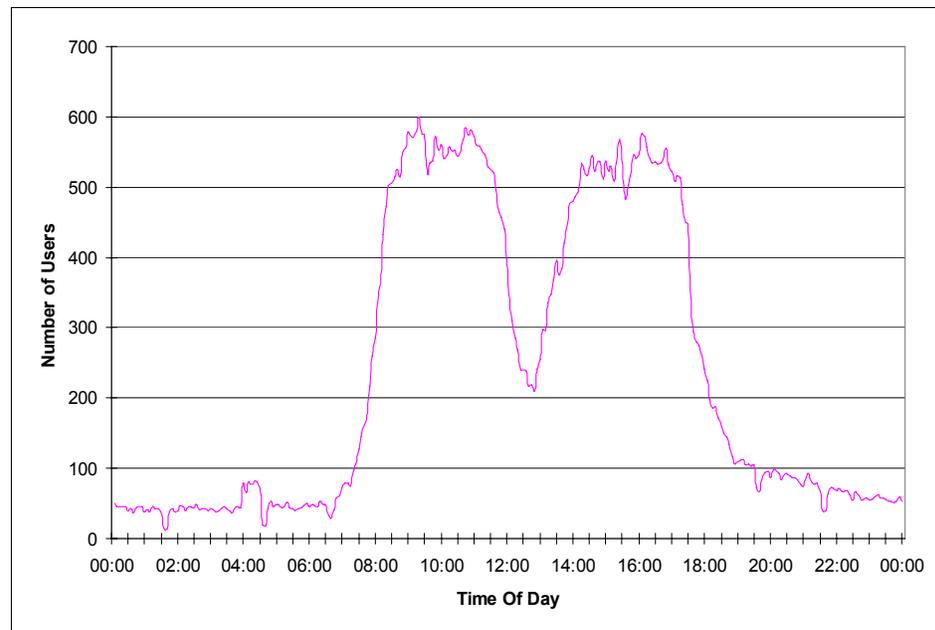


Figure 5-17 User distribution on an application server

For OLTP applications, the processor usage usually follows a similar distribution, as shown on Figure 5-18 for the same system.

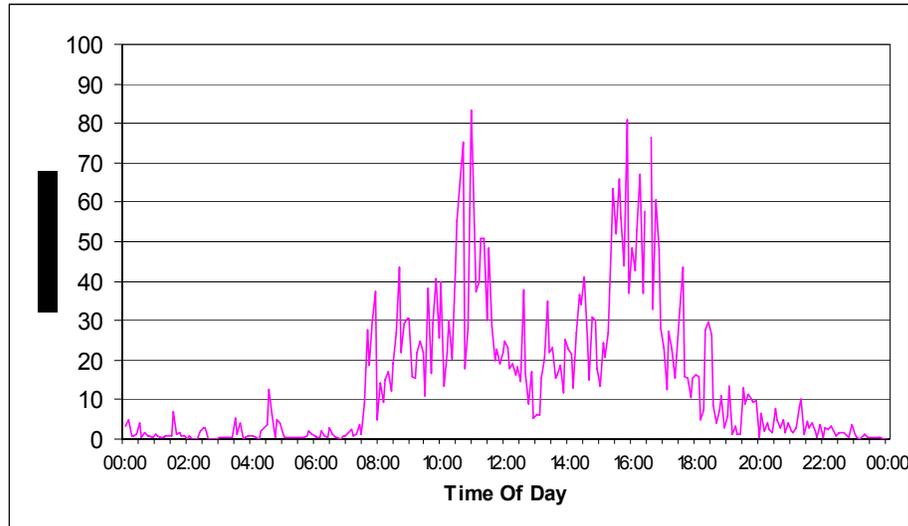


Figure 5-18 Processor utilization by the application server

The same behavior can be seen on mail servers. An analysis of a Lotus® Domino® server rendered a similar shape for number of users and processor usage.

If you have several workloads that have peak activity at different times, you can have each one running on a separate partition, and all partitions sharing the same physical processors. By adjusting each partition entitlement and the partition mode (capped or uncapped), you can run the system at a higher average utilization while fulfilling the processing requirements for each application.

Figure 5-19 on page 132 illustrates a typical scenario in which different applications are running in a Micro-Partitioning environment, with different peak times, and a mixed of capped and uncapped partitions. The system is running with four physical processors, virtualized into 20 virtual processors distributed between five partitions. Three partitions run OLTP types of applications, and two partitions run batch processing.

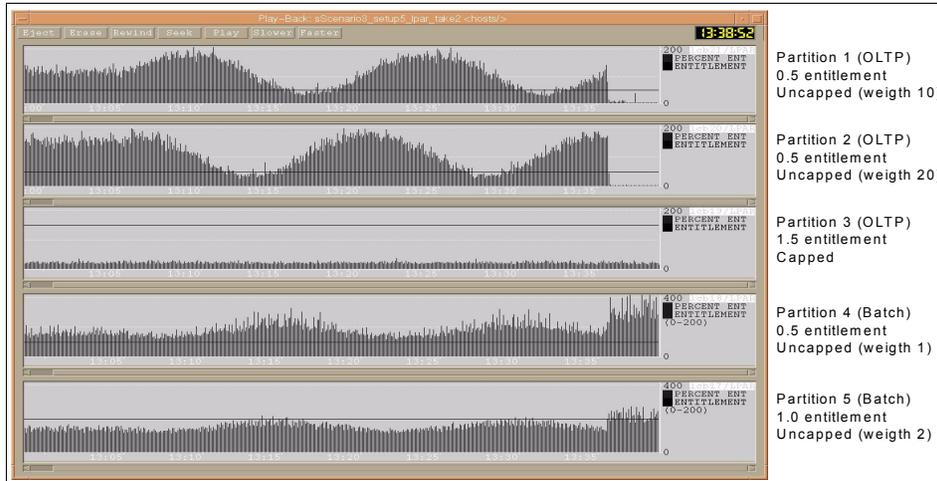


Figure 5-19 Processor utilization between five partitions

From this chart, we can see that partitions 1 and 2 have peak utilization at different times. Therefore, there is no need to duplicate the amount of resources to satisfy both partitions at peak processing. Partition 3 is capped and at a low utilization, so it remains constant during the time, and cedes the extra cycles not needed to other partition. Partitions 4 and 5 also benefit from the shared resources, receiving extra cycles whenever there are idle processors. And because of the nature of the applications (online and batch), the partition weight is a key factor to allocate the extra cycles to the uncapped partitions.

CPU-intensive applications

If an application uses most of its entitled processing capacity during its execution, it may not be suitable to run in a Micro-Partitioning environment. Because the requirements for the application are high and constant during execution, a dedicated processor partition is a better choice for this application. In a dedicated processor partition it will receive the processing capacity it needs, and it is less susceptible to interference from other workloads running in the system.

However, if the partition's entitled processing capacity does not own most of the physical processor's capacity, it will be beneficial to run these applications in an *uncapped* partition as they have the ability to use the extra cycles that may be available eventually. In this case, the application can execute more work on a system that would otherwise be idle. That would be the case when running online applications in a system during daytime and batch applications at night.

Typical applications in this scenario are decision support systems (DSS) and high-performance computing applications.

5.3.6 Micro-Partitioning planning guidelines

When planning for Micro-Partitioning, it is important to identify the application requirements and behavior in order to correctly size the partitions and maximize the system performance.

Planning for future applications is often a case where estimates are the only information available. In these cases, Micro-Partitioning can help, since partitions can be adjusted for required capacities in a very flexible way. On the other hand, an estimate can always be larger than the actual requirements, or smaller. Because of this, you must always consider having reserve capacity to accommodate unexpected resource requirements.

When the application environment is already in production or test, the task of planning for Micro-Partitioning becomes more direct. You can measure the resource utilization by the application on the running system and use this as a base for Micro-Partitioning performance requirements. Based on the detailed information you measure, you can plan the Micro-Partitioning environment to make the most effective use of the physical resources.

When planning for Micro-Partitioning, there are three main strategies for defining configurations:

Idle Resource Reallocation

A careful analysis of application resource usage and peak processing requirements, in order to deploy applications and substantially increase system utilization. You should run most of the partitions in uncapped mode.

Harvested Capacity

The definition of partitions that have quality of service requirements, and allowing other partitions to run on the system with the resources eventually idle. You may have some partitions running uncapped when you use this approach so that they can use available resources in the system.

Guaranteed Capacity

A basic definition, based on the sum of capacities from all servers being migrated, or based on sizing estimates using any published performance unit. In general, the partitions are running in capped mode when using this strategy.

Each strategy applies to different situations, depending on the amount of information you have for planning.

Idle Resource Reallocation

This is the strategy where you make the most efficient use of the processing capacity in the system. It is also the strategy that requires the most accurate planning and detailed knowledge about the applications behavior.

Idle Resource Reallocation involves an accurate knowledge of resource utilization over time by applications, in order to share resources and deliver quality of service. Instead of planning by summing up the peak utilization for each application, you plan a processing capacity sufficient for the sum of the usage of each application at all times. When one application does not consume resources up to the peak, these resources are reallocated to other applications that peak at that moment. Under normal circumstances, all partitions have their requirements fulfilled. If a few partitions consume resources up to the peak, the system still fulfills all partition requirements. However, if most or all of the application peaks at the same time in an unplanned manner, then the system is overcommitted and partitions will have performance constraints.

With adequate planning, a system can be configured with applications that do not overlap their peaks in processing, and therefore never overcommit the system. Total system usage will be high, and quality of service will be maintained, with maximum efficiency in resource usage.

Figure 5-20 on page 135 shows the processor usage (in percent over time) for three different applications during the same period. From the charts you can see that the peaks in processing for each application are not at the same time.

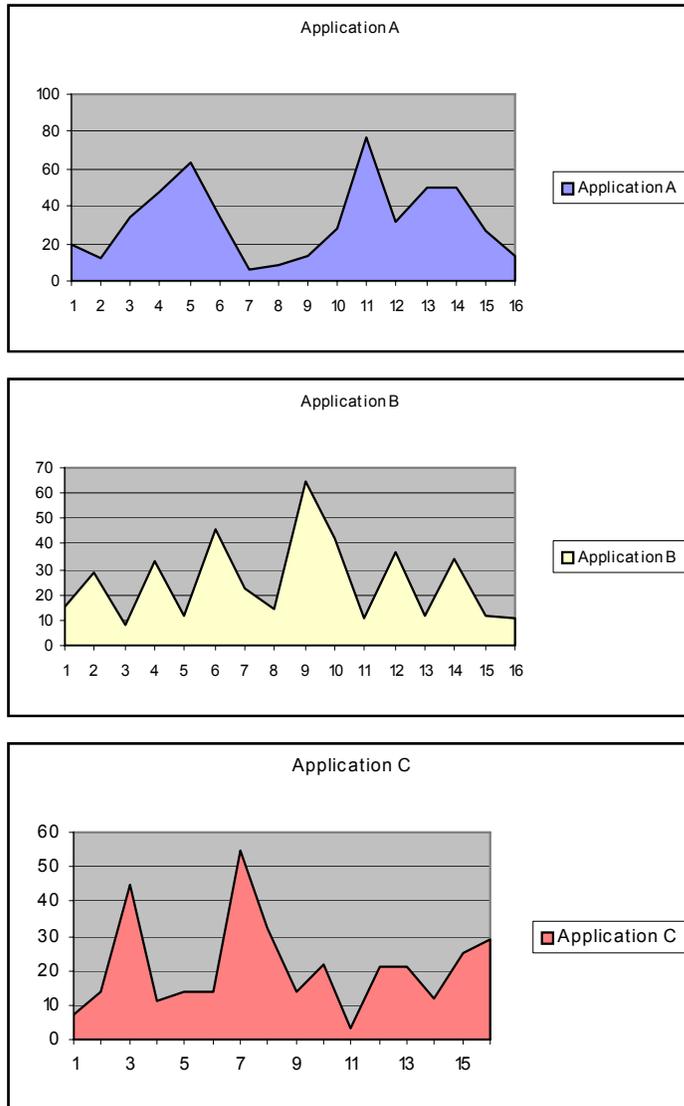


Figure 5-20 Resource utilization for three different applications

For clarity, the core of this example is simplicity. We hypothetically consider that for each of these applications, each percentage of resources is equivalent to 0.1 rPerf.

We can therefore show the peak utilization for each partition and the sum of the peaks, as well as the smaller server model that would be required to host these

applications in a dedicated server or dedicated partition environment (Table 5-3 on page 136):

Table 5-3 Peak utilization per partition

Application	Peak processing (%)	Capacity requirement in rPerf	Smallest server required
A	77	7.7	p5-510, 2 proc, 1500 MHz
B	65	6.5	p5-510, 2 proc, 1500 MHz
C	55	5.5	p5-510, 2 proc, 1500 MHz
TOTAL	197	19.7	p5-570, 6 proc, 1500 MHz (2 spare CoD)

The workload of these three applications could be fulfilled by three 2-way standalone servers, or one 8-way server with three dedicated partitions (two processors each) and two inactive processors.

If we consolidate these applications on a server with Micro-Partitioning, we can benefit from their behavior and size a system with less capacity than the sum of all peaks. First, we need to sum the usage for the three applications, at a given time.

Figure 5-21 shows the result of this sum, and we can see that the maximum peak processing for the sum is 96 percent (using the same consideration that for each application the ratio is 0.1 rPerf for each 1 percent of utilization). We therefore reach a requirement of 9.6 rPerf for all three applications.

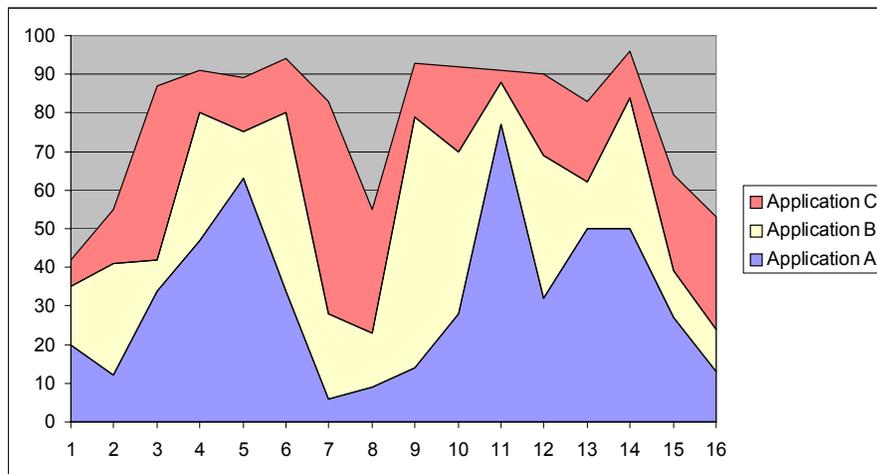


Figure 5-21 Application resource utilization example

By adding a 20 percent contingency to accommodate several factors such as uncertainties in the exact time of workload peaks, and system effects due to Micro-partitioning, we calculate to a requirement of 11.52 rPerf. This is about half of the capacity that we would need if we sized for peak capacity of each application independently (and also including the contingency).

This workload can be satisfied with one 4-way p5-550 at 1500 MHz, with three active processors and one inactive (Cod) processor, when compared to the deployment on dedicated partitions or servers that would have required six processors.

As previously discussed, this is the most efficient strategy for consolidating running systems using Micro-Partitioning. It is important to note that all partitions must be uncapped, so they can receive the resources needed for peak processing. Also, if for some reason the peaks in processing change, the partition entitlements must be recalculated and a new planning effort should be made. Otherwise, partitions may not be able to get the resources they need, and application performance will not be as optimal as it could be.

Harvested capacity

When you have a mix of partitions that have a response time requirement (such as OLTP applications) and partitions that do not have response time requirements (such as batch applications or test partitions), and you have some knowledge of the applications behavior, Micro-Partitioning is designed to run the workloads without providing capacity for the peak processing of each partition. You can provide capacity for the partitions that have the response time requirements, up to peak capacity. Because they do not normally run at peak processing, the extra resources can be used by the partitions that do not have response time requirements. For these partitions, instead of specifying a peak capacity, you define a minimum capacity for them to run and let them run uncapped, using the resources available from the other partitions.

In the case you run both production, test, and development partitions on the same server, you can, for example, configure a DB server and application server partitions so they have their processing requirements guaranteed. The development and test partitions can be configured as uncapped partitions and use any available resources on the system.

Another recommended application of this strategy is the case of a server farm running an application that receives load from load balancers. Normally the load is balanced among the servers executing the application. In case one server receives more workload than others, it can use more resources from the processor pool, then return to normal behavior when the extra workload finishes.

Guaranteed capacity

This is the simplest algorithm of capacity planning for Micro-Partitioning. When you are planning a system for new applications, typically no performance data is available about the resource utilization by the applications. Therefore, you should rely on application sizing and performance requirements estimates to size the partitions, and add extra capacity as a contingency for when the application needs more than initially planned.

This is also the case where you have the applications running, but cannot identify capacity utilization behavior (because of either insufficient metrics or random behavior).

For these situations, the simplest approach is to size a system based on the required capacities, up to the peak capacity, and add additional capacity for contingency. This method offers the smallest risk and is fairly simple to estimate. Moreover, since the system was planned based on the peak requirements for each application, you do not need a substantial effort in performance management, since there is installed capacity for all of the application performance requirements.

The drawback of this strategy is that it does not optimize resource usage based on application behavior, so a large fraction of the processing resources may be unused during hours of less activity, and if also when applications present complementary processing requirements (one application has a peak and the other has a valley).

An application of this strategy can be a server consolidation environment in an outsourcing contract, where each client pays for a guaranteed capacity, and there is not a possibility for over-commitment of resources. Another case is the consolidation of many applications with very small workloads, requiring less than one physical POWER5 processor. Take note that the entry server p5-510 at 1.65 GHz with only one processor has approximately the same processing power as a 12-way 7017-S7A that was considered the top high end enterprise class server less than ten years ago. Hence, many applications exist that only need a fraction of one POWER5 processor. Using guaranteed capacity algorithms for consolidating several of these applications on one processor using Micro-Partitioning technology is far more efficient than dedicating even the slowest available processor to each of these applications.

Consider the example of a three-tiered ERP system. Based on the functional requirements from the client, a sizing tool generates an estimate for system requirements based on peak requirements for each component of the solution.

A typical ERP solution is based on several servers running different functions; a database server, one or more application servers, one development system, and

one test system. A hypothetical example of a new system installation would be similar to the requirements listed in Table 5-4.

Note: rPerf is an estimate of relative commercial processing performance between IBM @server p5 systems. It is derived from an IBM analytical model that uses characteristics from IBM internal workloads and industry transaction processing and Web processing benchmarks. The rPerf model is not meant to represent any specific public benchmark result. It is used here as an indication of the required performance in IBM systems for this specific scenario.

Table 5-4 An example of an ERP system requirements

Function	Estimated capacity in rPerf	Estimated plus contingency in rPerf
DB Server	4.0	4.4
Application Server 1	3.3	3.7
Application Server 2	3.3	3.7
Development	1.8	2.0
Test	1.3	1.5
Total	13.7	15.3

If we were to use separate systems for each function, we would use five systems, with an adequate capacity to provide system usage within the performance requirements (Table 5-5).

Table 5-5 Implementation with separate servers

Function	Capacity requirement in rPerf	Server	Capacity provided in rPerf
DB Server	4.4	p5-510, 1 proc, 1650 MHz	5.24
App Server 1	3.7	p5-510, 1 proc, 1650 MHz	5.24
App Server 2	3.7	p5-510, 1 proc, 1650 MHz	5.24
Development	2.0	p5-510, 1 proc, 1500 MHz	3.25
Test	1.5	p5-510, 1 proc, 1500 MHz	3.25
Total	15.3	5 servers, 5 processors	22.22

The amount of rPerf required for the application is 15.3. The amount of rPerf configured into the systems is 22.22 due to physical constraints. (The number of processors must be an integer number.) Although extra capacity is being configured, it cannot be allocated wherever it is needed because these systems are separate. The DB server application, for example, can request extra processing power equivalent to 0.84 rPerf, while the Test partition can get an extra 1.75 rPerf.

If we use a more sophisticated approach by configuring a dedicated server, we will have more flexibility in moving extra resources among partitions, but still need to provide extra capacity that can be utilized. Table 5-6 shows the same example using dedicated processor partitioning with a 1.65 GHz server.

Table 5-6 Dedicated processor partitioning with 1.65 GHz mid-range server

Function	Required capacity in rPerf	Server	Capacity provided in rPerf
DB Server	4.4	p5-510, 1 proc, 1650 MHz	4.6
App Server 1	3.7	p5-510, 1 proc, 1650 MHz	4.6
App Server 2	3.7	p5-510, 1 proc, 1650 MHz	4.6
Development	2.0	p5-510, 1 proc, 1650 MHz	4.6
Test	1.5	p5-510, 1 proc, 1650 MHz	4.6
Total	15.3	1 servers, 5 activated 1650 MHz processors, 3 offline CoD processors	23.0 Activated 37.22 total available

Of the servers available at the time of writing, an 8-way p5-570 server matches the requirements. By ordering some of the processors as CoD features, it is possible to activate only five of the processors to satisfy the workload.

Again in this case, the provided processing power is more than needed (7.7 rPerf), but this extra processing power cannot be freely reused where it is required. For example, the needs of the DB server can only request an extra 0.2 rPerf equivalent, while the test partition can request up to 3.1 rPerf. When using CoD, it is possible to satisfy the DB extra resource needs by activating one of the CoD processors.

When using a server with Micro-Partitioning, you can accommodate the different functions with more effective utilization. A single IBM @server p5 550 can deliver up to 19.66 rPerf with four POWER5 processors running at 1.65 GHz.

Since the behavior of each system is not known, so to accommodate the requirements of a single system using Micro-Partitioning, we sum the peak performance requirements for each function, and apply a 20 percent sizing contingency. This contingency accommodates several factors, including the fact that rPerf is only an approximate indicator of performance between systems, uncertainties in workload peaks, and system effects due to Micro-partitioning technology.

For this workload, we would have the configuration as provided in Table 5-7 using a 4-way p5-550 server.

Table 5-7 .Implementation with Micro-Partitioning

Function	Requested capacity in rPerf	Capacity using Micro-Partitioning technology with a 20% contingency	% of physical processor requirement
DB Server	4.4	5.28	1.07
App Server 1	3.7	4.44	0.90
App Server 2	3.7	4.44	0.90
Development	2.0	2.4	0.49
Test	1.5	1.8	0.37
Total	15.3	18.36	3.73

The extra resources on the machine can then be allocated to any of the partitions whenever they require capacity. Moreover, when a partition is not using its total capacity, the remainder of its entitlement is automatically available in the shared processing pool. Also, when the applications are running, resource allocation can be fine-tuned and allocated according to the partition needs.

This example shows that even when using the guaranteed capacity sizing algorithm, and taking a 20 percent contingency for the solution using Micro-Partitioning technology over the dedicated server solutions, micro-partitioning allows replacing five servers, each with one processor, with one 4-way server.

Sizing partitions for a virtualized environment is not fundamentally different than sizing for dedicated systems. The ultimate efficiency of sizing depends heavily on the knowledge of the workload, degree of risk assumed in sizing, and the expected attention to capacity monitoring. Idle resource reallocation provides the most optimized environment. However, it requires good knowledge of the workload and likely the closest monitoring of system capacity. Harvested capacity allows very high system utilizations, if workloads with relaxed response time requirements can exploit otherwise idle cycles. Guaranteed capacity, while

the least effective at maximizing the overall hardware utilization, works extremely well for very small partitions. Guaranteed capacity also generally requires the least attention to capacity monitoring, as there are no consequences of workloads peaking concurrently.

5.4 Summary

There are some performance considerations to take into account when implementing Micro-Partitioning technology. AIX 5L V5.3 and, to a lesser extent, Linux do a good job of sharing the computing resources across the workloads they are running. With Workload Manager or Partition Load Manager, it is possible to make sure that an organization's priorities are respected when there is a conflict. Because of the overhead of scheduling virtual processors, there should be an objective of keeping the number of partitions to a minimum.

Important: To maximize performance, keep the number of partitions to a minimum.

If organizational policies do not require separate partitions, you must ask, "What are the technical and performance reasons for creating a new partition, rather than adding a new workload to an existing one and providing it with the same amount of additional resources?" A consolidation project should have higher objectives than replacing n -machines by n -partitions.

Some good reasons for using Micro-Partitioning technology include:

- ▶ Tuning the operating system for a given application; for example, 32-bit or 64-bit kernel, large pages, threaded Ethernet adapters, Linux or AIX 5L.
- ▶ A Network Install Manager (NIM) server, which must always be at the latest level of AIX to be installed. The partition can be activated when required; otherwise its resources can be made available to the shared pool.
- ▶ Ad hoc partition creation for an on-off occasion, demo, trial software, training, and so on.
- ▶ Containing an unpredictable or runaway application that prevents it from affecting other applications, although this can also be achieved with WLM.
- ▶ Provide application isolation for security or organizational reasons; for example, you may want a firewall application to be isolated from your Web servers. Isolating development, test, and training activities from production.

Careful planning should be done to satisfy application resource requirements. This enables the system to be utilized efficiently with satisfactory performance from the application point of view.



Virtual I/O

This chapter provides an introduction to virtual input/output (I/O), as well as a close look at how the POWER Hypervisor handles transactions between the partitions. This chapter also addresses performance aspects for each of the components of the virtual I/O system. With respect to virtual I/O, the components covered in this chapter are:

- ▶ POWER Hypervisor
- ▶ Virtual Serial Adapter
- ▶ Virtual Ethernet
- ▶ Shared Ethernet Adapter
- ▶ Virtual SCSI

The virtual I/O product documentation can be found at:

Using the Virtual I/O Server

<http://publib.boulder.ibm.com/infocenter/eserver/v1r3s/index.jsp?lang=en>

6.1 Introduction

Virtual I/O provides the capability for a single I/O adapter to be used by multiple logical partitions on the same server, enabling consolidation of I/O resources and minimizing the number of required I/O adapters. The driving forces behind virtual I/O are:

- ▶ The advanced technological capabilities of today's hardware and operating systems such as POWER5 and IBM AIX 5L Version 5.3.
- ▶ The value proposition enabling on demand computing and server consolidation. Virtual I/O also provides a more economic I/O model by using physical resources more efficiently through sharing.

As we write this, the virtualization features of the POWER5 platform support up to 254 partitions, while the server hardware provides only up to 160 I/O slots per machine. With each partition typically requiring one I/O slot for disk attachment and another one for network attachment, this puts a constraint on the number of partitions. To overcome these physical limitations, I/O resources have to be shared. Virtual SCSI provides the means to do this for SCSI storage devices.

Furthermore, virtual I/O enables attachment of previously unsupported storage solutions. As long as the Virtual I/O Server supports the attachment of a storage resource, any client partition can access this storage by using virtual SCSI adapters.

For example, if there is no native support for EMC storage devices on Linux, running Linux in a logical partition of a POWER5 server makes this possible. A Linux client partition can access the EMC storage through a virtual SCSI adapter. Requests from the virtual adapters are mapped to the physical resources in the Virtual I/O Server. Therefore, driver support for the physical resources is needed only in the Virtual I/O Server.

Typically, a small operating system instance needs at least one slot for a Network Interface Connector (NIC) and one slot for a disk adapter (SCSI, Fibre Channel, and so on), but more robust configurations often consist of two redundant NIC adapters and two disk adapters.

Virtual I/O devices are intended as a complement to physical I/O adapters (also known as dedicated or local I/O devices). A partition can have any combination of local and virtual I/O adapters.

Supported levels

Although IBM @server p5 servers support AIX 5L Version 5.2, it is not possible to run an AIX 5L V5.2 partition with Micro-Partitioning, virtual SCSI, virtual

Ethernet, or shared ethernet adapters. However, a mixed environment between AIX 5L V5.2 and AIX 5L V5.3 partitions on @server p5 servers is supported.

Figure 6-1 shows a sample configuration with mixed operating systems and mixed AIX 5L versions. The first five partitions are using dedicated processors. The AIX 5L V5.2 partition cannot join the virtual I/O paths, but it provides all known LPAR and dynamic LPAR features. It has to be configured with dedicated I/O adapters. The AIX 5L V5.3 partitions using shared processors likewise may use dedicated storage and dedicated networking.

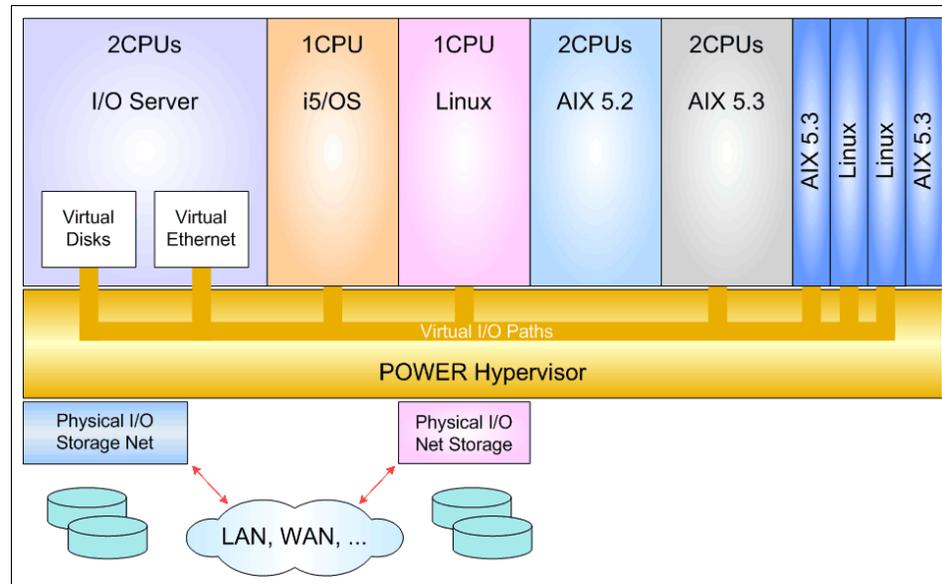


Figure 6-1 Mixed operating system environment

6.2 POWER Hypervisor support for virtual I/O

As Figure 6-1 illustrates, the POWER Hypervisor provides the interconnection for the partitions. To use the functionalities of virtual I/O, a partition uses a *virtual adapter*. The POWER Hypervisor provides the partition with a view of an adapter that has the appearance of an I/O adapter, which may or may not correspond to a physical I/O adapter. The POWER Hypervisor provides two classifications of virtual adapters:

- ▶ *Hypervisor simulated class*. This classification of virtual adapter, shown in Figure 6-2 on page 146, is where the POWER Hypervisor simulates an I/O adapter. This class is used in virtual Ethernet support (see 6.5, “Virtual

Ethernet” on page 164). This technique provides reliable and fast communication between partitions using network protocols.

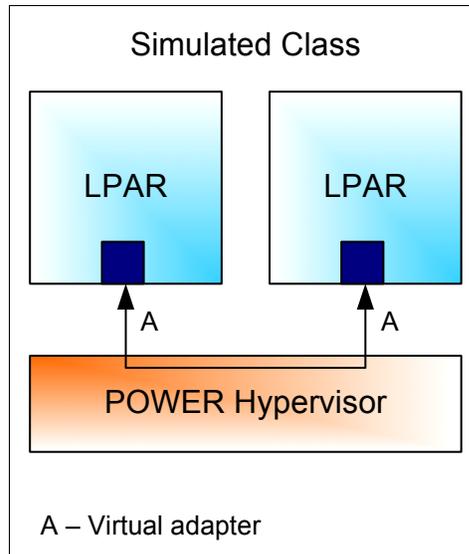


Figure 6-2 POWER Hypervisor simulated class

- ▶ *Partition managed class.* In this class, shown in Figure 6-3 on page 147, a server partition provides the services of one of its physical I/O adapters to client partition. A server partition provides support to handle I/O requests from the client partition and perform those requests on one or more of its devices, targeting the client partition’s direct memory access (DMA) buffer areas using LRDMA facilities. (See “Logical Remote Direct Memory Access (LRDMA)” on page 150.) It then passes I/O responses back to the client partition. This classification of virtual adapter is used by virtual SCSI as described in 6.8, “Virtual SCSI” on page 205.

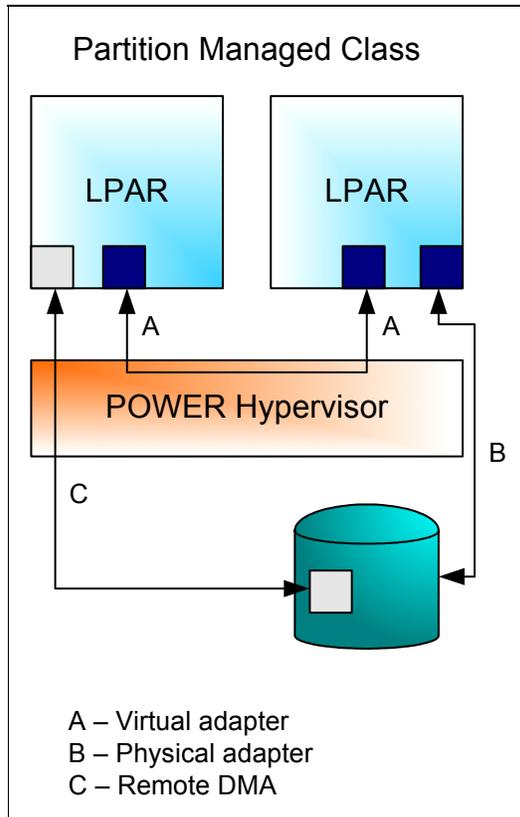


Figure 6-3 POWER Hypervisor partition managed class

6.2.1 Virtual I/O infrastructure

The virtual I/O infrastructure is a complex subject, and it is not the purpose of this book to address it extensively. We briefly present some of the components that are relevant to understanding the performance issues.

The Open Firmware device tree

The virtual I/O adapters and associated inter-partition communication paths are defined using the HMC during the creation of the partition's profile.

When a partition is booted, it receives from the POWER Hypervisor the definition of all of its available hardware resources as *device nodes* in what is called the partition *Open Firmware device tree*.

Depending on the specific virtual device, their device tree node may be found as a child of / (the root node) or in the virtual I/O subtree.

In addition to the virtual I/O devices, the Open Firmware device tree also contains the definition of the *virtual host bridge* and the *virtual interrupt source controller*. These definitions enable the partition to communicate with the virtual devices in the same way that it communicates with physical devices.

Each virtual device node in the Open Firmware device tree contains the properties defined in Table 6-1.

Table 6-1 Required attributes of the /vdevice node

Property name	Req?	Definition
name	Yes	Standard property name per IEEE 1275 specifying the virtual device name; the value shall be "vdevice"
device_type	Yes	Standard property name per IEEE 1275 specifying the virtual device type; the value shall be "vdevice"
model	No	Property not present
compatible	Yes	Standard property name per IEEE 1275 specifying the virtual device programming models; the value shall include "IBM,vdevice"
used-by-rtas	No	Property not present
ibm,loc-code	No	Location code
reg	No	Property not present
#size-cells	Yes	Standard property name per IEEE 1275; the value shall be 0. No child of this node takes space in the address map as seen by the owning partition.
#address-cells	Yes	Standard property name per IEEE 1275; the value shall be 1.
#interrupt-cells	Yes	Standard property name per IEEE 1275; value shall be 2. The first cell contains the interrupt# as will appear in the XIRR and is used as input to interrupt RTAS calls. The second cell contains the value 0, indicating a positive edge sense.
interrupt-map-mask	No	Property not present.
interrupt-ranges	Yes	Standard property name that defines the interrupt number(s) and range(s) handled by this unit.
interrupt map	No	Property not present

Property name	Req?	Definition
interrupt-controller	Yes	The /vdevice node appears to contain an interrupt controller.
ranges	No	Used by virtual adapters.
ibm,drc-indexes	for DR	For <i>Dynamic Reconfiguration (DR)</i> . Refers to the DR slots: the number provided is the maximum number of slots that can be configured. This is limited by, among other things, the RTCE tables allocated by the POWER Hypervisor.
ibm,drc-power-domains	for DR	Value of -1 to indicate that no power manipulation is possible or needed.
ibm,drc-types	for DR	Value of "SLOT"—any virtual IOA can fit into any virtual slot.
ibm,drc-names	for DR	The virtual location code.

6.2.2 Types of connections

The virtual I/O infrastructure provides several primitives that are used to build connections between partitions for various purposes. These primitives include:

- ▶ A *Command/Response Queue (CRQ)* facility that provides a *pipe* between partitions. A partition can enqueue a command to the target partition's CRQ for processing by that partition. The partition can set up the CRQ to receive an interrupt when an entry is placed in the queue.
- ▶ An extended *Translation Control Entry (TCE)* table called the *Remote TCE (RTCE)* table, which enables a partition to provide *windows* to its memory for other partitions to use, while maintaining addressing and access control to its memory.
- ▶ *Remote DMA* services that enable a server partition to transfer data to another partition's memory via the RTCE windows. This enables a device driver in a server partition to efficiently transfer data to and from another partition. This is key to sharing a virtual I/O adapter in the server partition.

The Command/Response Queue

The CRQ facility provides a communications pipeline for ordered delivery of messages between authorized partitions. The facility is reliable in the sense that the messages are delivered in sequence. The sender is notified if the transport facility in the POWER Hypervisor able to deliver the message or was unable to provide the data associated with the message, or if the target partition either fails

or deregisters its half of the transport connection. Optionally, the CRQ owner may choose to be notified via an interrupt when a message is added to their queue.

The CRQ facility does not police the contents of the payload portions (after the one-byte header) of messages that are exchanged between the communicating partitions. The architecture does provide means (via the Format Byte) for self-describing messages so that the definitions of the content and protocol between the partitions may evolve over time without change to the CRQ architecture or its implementation.

Remote Translation Control Entry (RTCE)

The TCE and RTCE tables are used to translate direct memory access (DMA) operations and provide protection against improper operations.

The RTCE table is analogous to the TCE table for dedicated I/O, and Table 6-2 shows a comparison. The RTCE table has more information in it provided by the POWER Hypervisor. This enables the POWER Hypervisor to create links to the TCEs on the partition that owns the device. An entry in the RTCE table is never accessed directly by the operating system; only through POWER Hypervisor calls as described 4.1.1, “POWER Hypervisor functions” on page 79.

Table 6-2 TCE and RTCE comparison

TCE (Translation Control Entry)	RTCE (Remote TCE)
In POWER4 processor-based pSeries servers	In POWER5 processor-based pSeries servers
Translation table for logical to dedicated I/O bus addresses	Needed for Remote DMA
Managed by the POWER Hypervisor	Managed by the POWER Hypervisor
Addressed by the operating system	<i>Never</i> addressed directly by the operating system. Addressed only through POWER Hypervisor calls.

Logical Remote Direct Memory Access (LRDMA)

The virtual I/O infrastructure can take advantage of different types of Direct Memory Access (DMA). The virtual SCSI feature only uses Logical Remote Direct Memory Access (LRDMA).

LRDMA enables an I/O server to securely target memory pages within an I/O client for virtual I/O operations. The I/O server uses the POWER Hypervisor call of the Logical Remote DMA facility to manage the movement of commands and data associated with the client requests. The server driver may use this service if

it has a connection established via a Command/Response Queue. Virtual SCSI defines two modes of LRDMA:

Copy RDMA

The I/O devices target DMA buffers in the I/O server's memory. After the DMA transfer completes, the POWER Hypervisor copies the data between the DMA buffers and the client's memory.

Redirected RDMA

This mode allows for an I/O device to securely perform DMA transfers directly into the client partition's memory.

Example 6-4 shows how data is transferred using redirected RDMA.

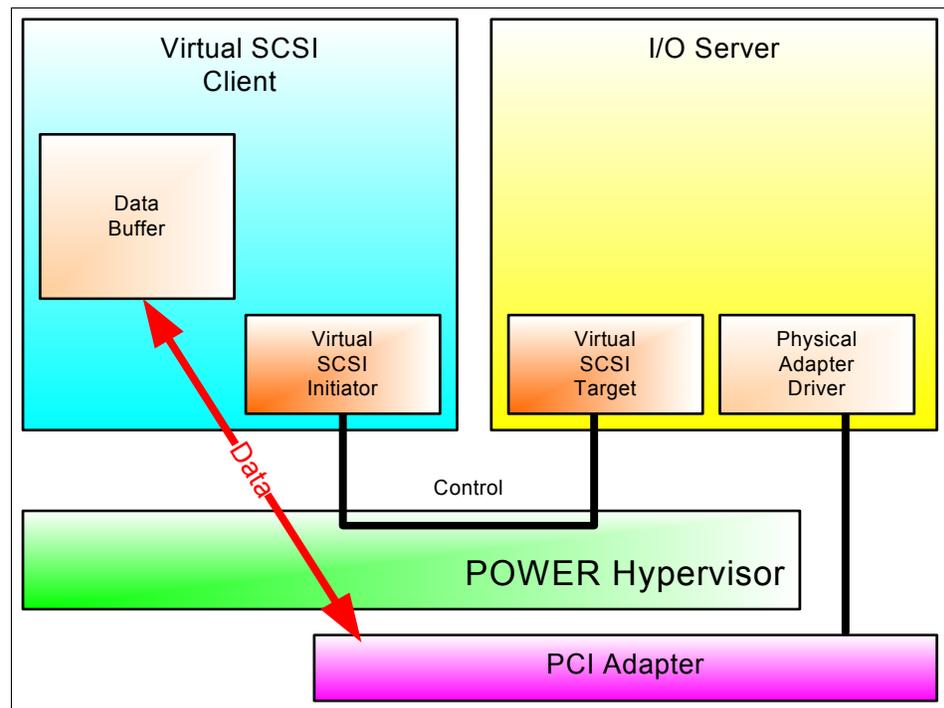


Figure 6-4 Logical Remote Direct Memory Access (LRDMA)

Redirected RDMA provides better overall system performance because the data is transferred to and from the data buffer by the DMA controller on the adapter card. This performance degradation by using copy RDMA may be offset if the I/O server's DMA buffer is being used as cache for multiple virtual I/O operations.

6.3 The IBM Virtual I/O Server

The IBM Virtual I/O Server is the link between the virtual resources and physical resources. It is a specialized partition that owns the physical I/O resources, and is supported only on POWER5 processor-based servers. This server runs in a special partition that cannot be used for execution of application code.

It mainly provides two functions:

- ▶ Serves virtual SCSI devices to client partitions
- ▶ Provides a Shared Ethernet Adapter for VLANs

Installation of the Virtual I/O Server partition is performed from a special **mksysb** CD-ROM that is provided to clients who order the Advanced POWER Virtualization feature, at an additional charge¹. This is dedicated software only for the Virtual I/O Server operations, so the Virtual I/O Server software is supported only in Virtual I/O Server partitions.

You can install the Virtual I/O Server from CD or using NIM on Linux (NIMoL) from the Hardware Maintenance Console (HMC).

The Virtual I/O Server supports the following operating systems as Virtual I/O clients:

- ▶ IBM AIX 5L V5.3
- ▶ SUSE LINUX Enterprise Server 9 for POWER
- ▶ Red Hat Enterprise Linux AS 3 for POWER, update 3
- ▶ Red Hat Enterprise Linux AS 4 for POWER

The I/O Server operating system is hidden to simplify transitions to further versions. No specific operating system skills are required for administration of the I/O Server.

Performance considerations for the Virtual I/O Server are addressed in “Virtual I/O Server performance results” on page 191.

Command line interface

The Virtual I/O Server provides a restricted scriptable command line interface (CLI). All aspects of Virtual I/O Server administration are accomplished through the CLI, including:

- ▶ Device management (physical, virtual, LVM)
- ▶ Network configuration
- ▶ Software installation and update
- ▶ Security

¹ Included with @server p5-590 and @server p5-595.

- ▶ User management
- ▶ Installation of OEM software
- ▶ Maintenance tasks

For the initial logon to the Virtual I/O Server, use the user ID **padmin**, which is the prime administrator. When logging on, you are prompted for a new password, so there is no default password to remember.

Upon logging on to the I/O server, you will be placed into a restricted Korn shell. The restricted Korn shell works the same way as a regular Korn shell with some restrictions. Specifically, users cannot do the following:

- ▶ Change the current working directory.
- ▶ Set the value of the SHELL, ENV, or PATH variables.
- ▶ Specify the path name of the command that contains a redirect output of a command with a `>`, `>|`, `<>`, or `>>`.

As a result of these restrictions, you cannot execute commands that are not accessible to your PATH. In addition, these restrictions prevent you from directly sending the output of the command to a file, requiring you to pipe the output to the **tee** command instead.

The Virtual I/O Server CLI supports two execution modes: *traditional* and *interactive*.

The traditional mode is for single command execution. In this mode, you execute one command at a time from the shell prompt. For example, to list all virtual devices, enter the following:

```
#ioscli lsdev -virtual
```

To reduce the amount of typing required in traditional shell level mode, an alias has been created for each subcommand. With the aliases set, you are not required to type the **ioscli** command. For example, to list all devices of type adapter, you can enter the following:

```
#lsdev -type adapter
```

You can type **help** for an overview of the supported commands, as shown in Example 6-1 on page 154.

Example 6-1 Commands available in Virtual I/O Server environment

```
$ help
```

Install Commands	Physical Volume Commands	Security Commands
updateios	lspv	lsgcl
lssw	migratepv	cleargcl
ioslevel		lsfailedlogin
remote_management	Logical Volume Command	
oem_setup_env	lslv	UserID Commands
oem_platform_level	mklv	mkuser
license	extendlv	rmuser
	rmlvcopy	lsuser
LAN Commands	rmlv	passwd
mktcpip	mklvcopy	chuser
hostname		
cfglnagg		
netstat	Volume Group Commands	Maintenance Commands
entstat	lsvg	chlang
cfgnamesrv	mkvg	diagmenu
traceroute	chvg	shutdown
ping	extendvg	fck
optimizenet	reducevg	backupios
lsnetsvc	mirrorios	savevgstruct
	unmirrorios	restorevgstruct
Device Commands	activatevg	starttrace
mkvdev	deactivatevg	stoptrace
lsdev	importvg	cattracerpt
lsmap	exportvg	bootlist
chdev	syncvg	snap
rmdev		startsysdump
cfgdev		topas
mkpath		mount
chpath		unmount
lspath		showmount
rmpath		startnetsvc
errlog		stopnetsvc

In interactive mode, the user is presented with the **ioscli** command prompt by executing the **ioscli** command without any subcommands or arguments. From this point on, **ioscli** commands are executed one after the other without having to retype **ioscli**. For example, to enter interactive mode, type:

```
#ioscli
```

When in interactive mode, to list all virtual devices, enter:

```
#lsdev -virtual
```

External commands, such as **grep** or **sed**, cannot be executed from the interactive mode command prompt. You must first exit interactive mode by entering **quit** or **exit**. A measurement of virtual I/O in conjunction with the Shared Ethernet Adapter functionality is discussed in 6.6, “Shared Ethernet Adapter” on page 186.

Limitations and considerations

The Virtual I/O Server software is dedicated software only for the Virtual I/O Server operations, and there is no possibility of running other applications in the Virtual I/O Server partition.

There is no option to get the Virtual I/O Server partition pre-installed on new systems. As this is written, the pre-install manufacturing process does not allow the Virtual I/O Server partition to be pre-installed.

Other limitations can occur because of resource shortages. The Virtual I/O Server should be properly configured with enough resources. The most important are the processor resources. If a Virtual I/O Server has to host a lot of resources to other partitions, you must ensure that enough processor power is available. In case of high load or high traffic across virtual Ethernet adapters and virtual disks, partitions can observe delays in accessing resources.

Logical volume limitation

The Virtual I/O Server software enables you to define up to 1024 logical volumes per volume group, but the actual number you can define depends on the total amount of physical storage defined for that volume group and the size of the logical volumes you configure.

Table 6-3 shows the limitations for logical storage management.

Table 6-3 Limitations for logical storage management

Category	Limit
Volume group	4,096 per system
Physical volume	1,024 per volume group
Physical partition	2,097,152 per volume group
Logical volume	4,096 per volume group
Logical partition	Based on physical partitions

6.3.1 Providing high availability support

When we talk about providing high availability for the Virtual I/O Server we are talking about incorporating the I/O resources (physical and virtual) on the Virtual I/O Server as well as the client partitions into a configuration that is designed to eliminate single points of failure.

The Virtual I/O Server is a single point of failure. In case of a crash of the Virtual I/O Server, the client partitions will see I/O errors and not be able to access the adapters and devices that are hosted by the Virtual I/O Server.

However, redundancy can be built into the configuration of the physical and virtual I/O resources at several stages.

Since the Virtual I/O Server is a customized AIX 5L OS-based appliance, redundancy for physical devices attached to the Virtual I/O Server can be provided by using capabilities such as LVM mirroring, Multipath I/O, and EtherChannel.

Note: When activating the EtherChannel you may see some `Unsupported ioctl in device driver` errors if you are using virtual Ethernets in your Link Aggregation. These errors can be ignored.

Figure 6-5 on page 157 shows a single Virtual I/O Server configuration with disk and network attachment. The disks are mirrored through LVM. The two physical network adapters are configured as a Link Aggregation in Network Interface Backup (NIB) mode.

While this kind of configuration protects you from the failure of one of the physical components, such as a disk or network adapter, it will still cause the client partition to lose access to its devices if the Virtual I/O Server fails.

The Virtual I/O Server itself can be made redundant by running a second instance of it in another partition. When running two instances of the Virtual I/O Server, you can use LVM mirroring, Multipath I/O, Link Aggregation, or Multipath routing with dead gateway detection in the client partition to provide highly available access to virtual resources hosted in separate Virtual I/O Server partitions. Many configurations are possible; they depend on the available hardware resources as well as your requirements.

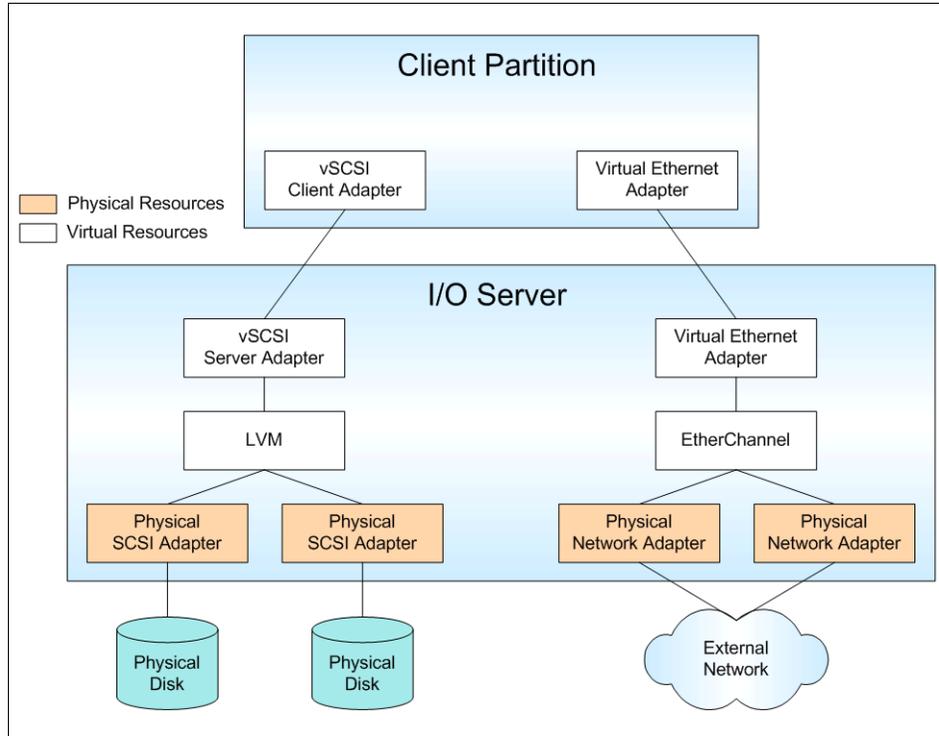


Figure 6-5 Single Virtual I/O Server configuration

Network interface backup

Figure 6-6 on page 158 shows a configuration using network interface backup.

The client partition has two virtual Ethernet adapters. Each adapter is assigned to a different VLAN (using the PVID). Each Virtual I/O Server is configured with a Shared Ethernet Adapter that bridges traffic between the virtual Ethernet and the external network. Both Shared Ethernet Adapters should be able to connect to the same set of hosts in the external network.

Each of the Shared Ethernet Adapters is assigned to a different VLAN (using PVID). By using two VLANs, network traffic is separated so that each virtual Ethernet adapter in the client partition seems to be connected to a different Virtual I/O Server.

The two virtual Ethernet adapters in the client partition are configured as an EtherChannel using Network Interface Backup. The Link Aggregation is configured with a primary adapter and a backup, and the operation mode is left as the default standard mode. Additionally, the EtherChannel is configured with an Internet Address to Ping. This address will be pinged periodically by the

EtherChannel to determine whether connectivity to the external network exists. Typically a router that should be always available is used as the ping target.

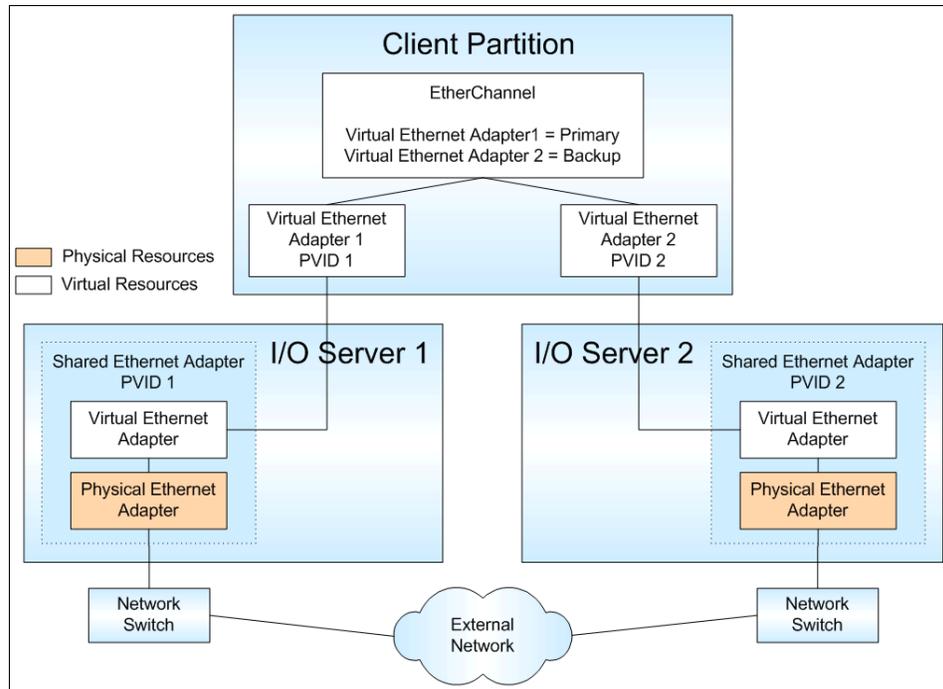


Figure 6-6 Virtual I/O Server configuration with network interface backup

Even though a Link Aggregation with more than one primary virtual Ethernet adapter is not supported, a single virtual Ethernet adapter Link Aggregation is possible because a single adapter configured as an EtherChannel in standard mode does not require switch support from the POWER Hypervisor.

The IP address of the client partition is configured on the network interface of the EtherChannel. If the primary adapter fails, the EtherChannel will automatically switch to the backup adapter. The IP address of the client server partition that is configured on the EtherChannel network interface will remain available.

Restriction: When using the EtherChannel with two adapters as in this example and configuring one adapter as backup, no aggregation resulting in higher bandwidth will be provided. No network traffic will go through the backup adapter unless there is failure of the primary adapter.

Also note that gratuitous ARP has to be supported by the network in order for adapter failover to work.

This configuration protects your network interface adapter against:

- ▶ Failure of one physical network adapter in one Virtual I/O Server partition
- ▶ Failure of one Virtual I/O Server partition
- ▶ Failure of one network switch (if adapters are connected to different switches as shown in this example)

The physical Ethernet adapters shown in Figure 6-6 on page 158 are connected to the network switches on untagged ports. The Virtual I/O Server partition strips VLAN tags from packets before delivering them to the switches. The network switches see the MAC addresses on the virtual Ethernet adapters in the client partition, but will not see the VLAN tags. The Virtual I/O Server partition propagates broadcast packets from the switches to the virtual Ethernet adapters in the client partition.

If a Virtual I/O Server (or some network component) fails, the Ethernet network will see the client partition's IP address suddenly hop from one switch and MAC address to another. Such behavior will be handled acceptably if both of the following are true:

- ▶ The network supports Gratuitous ARP.
- ▶ The network switches are configured such that both ports (one on each switch) can contact the same set of hosts in the rest of the network.

It is recommended that the client partition be configured to detect network unreachability by specifying in the Network Interface Backup configuration an IP address (or host name) of a router to which connectivity should always be available.

For more details about configuring Link Aggregation (EtherChannel) see *AIX System Management Guide: Communications and Networks*, which is available with the product documentation.

Multipath routing and dead gateway detection

Figure 6-7 on page 160 shows a configuration using multipath routing and dead gateway detection.

The client partition has two virtual Ethernet adapters. Each adapter is assigned to a different VLAN (using the PVID). Each Virtual I/O Server is configured with a Shared Ethernet Adapter that bridges traffic between the virtual Ethernet and the external network. Each of the Shared Ethernet Adapters is assigned to a different VLAN (using PVID).

By using two VLANs, network traffic is separated so that each virtual Ethernet adapter in the client partition seems to be connected to a different Virtual I/O Server.

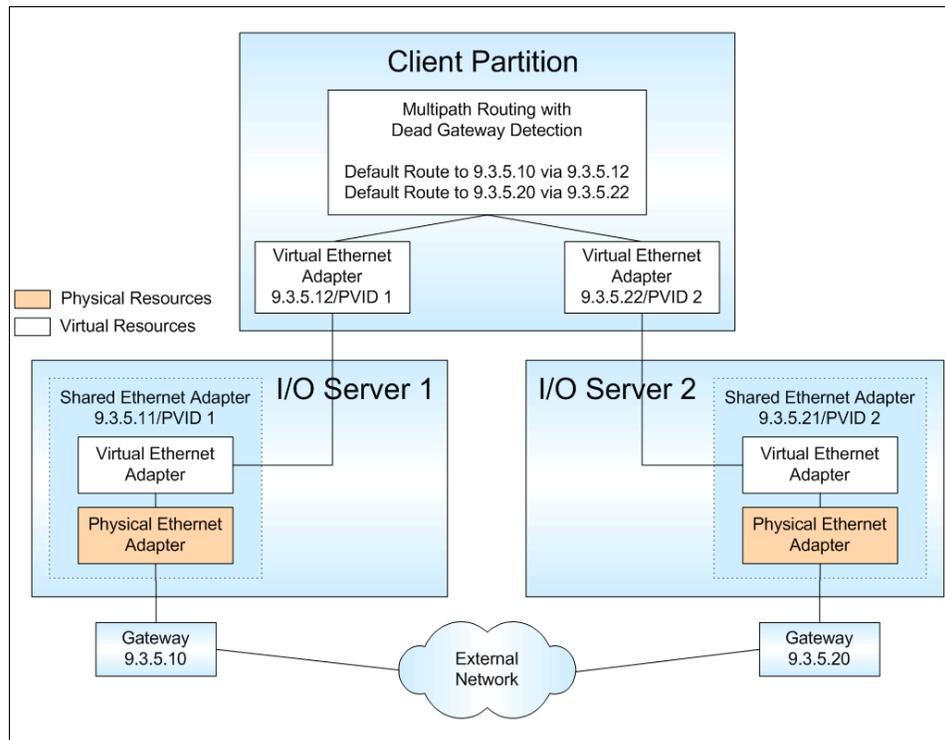


Figure 6-7 Configuration with multipath routing and dead gateway detection

In the client partition, two default routes with dead gateway detection are defined: One route is going to gateway 9.3.5.10 using virtual Ethernet adapter with address 9.3.5.12; the second default route is going to gateway 9.3.5.20 using the virtual Ethernet adapter with address 9.3.5.22.

In case of a failure of the primary route, access to the external network is provided through the second route. AIX 5L detects route failures and adjusts the cost of the route accordingly.

Restriction: It is important to note that multipath routing and dead gateway detection do not make an IP address highly available. In the case of failure of one path, dead gateway detection will route traffic through an alternate path. The network adapters and their IP addresses remain unchanged. Therefore, when using multipath routing and dead gateway detection, only your access to the network will become redundant, but *not* the IP addresses.

This configuration protects your access to the external network against failure of:

- ▶ One physical network adapter in one Virtual I/O Server partition
- ▶ One Virtual I/O Server partition
- ▶ One gateway

LVM mirroring

Figure 6-8 shows a Virtual I/O Server configuration using LVM mirroring on the client partition. The client partition is LVM mirroring its logical volumes using the two virtual SCSI client adapters. Each of these adapters is assigned to a separate Virtual I/O Server partition.

The two physical disks are each attached to a separate Virtual I/O Server partition and made available to the client partition through a virtual SCSI server adapter.

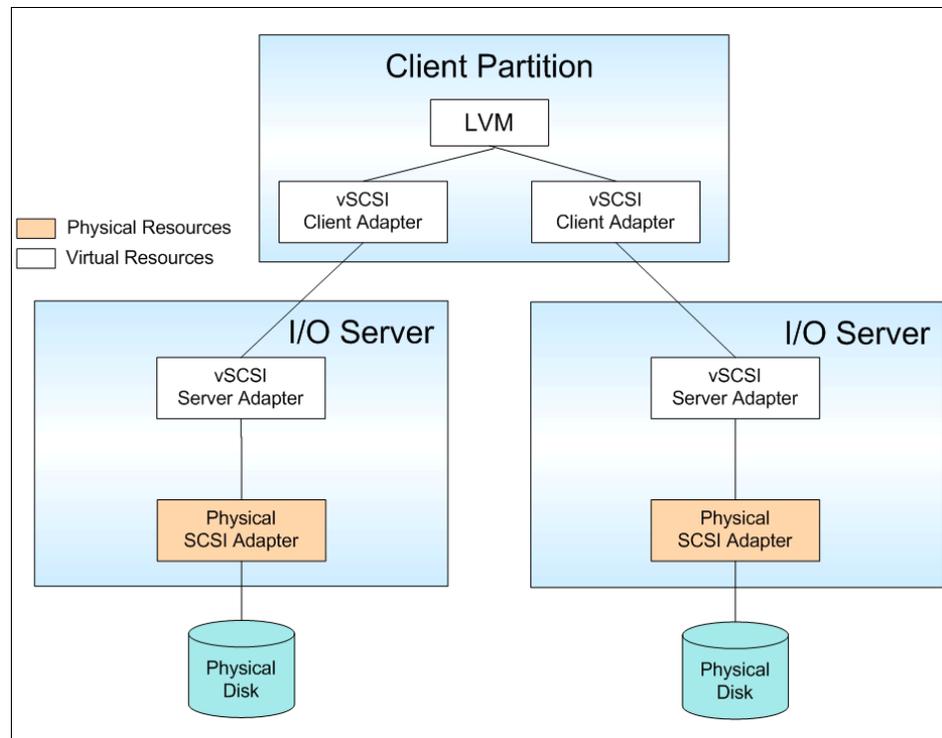


Figure 6-8 Virtual I/O Server configuration with LVM mirroring

Restriction: At we write this book, LVM mirroring using virtual SCSI works only when the logical volume on the Virtual I/O Server is configured with the following settings:

- ▶ Mirror Write Consistency turned off
- ▶ Bad Block Relocation turned off
- ▶ No striping
- ▶ Logical volume must not span several physical volumes

This configuration protects a virtual disk in a client partition against failure of:

- ▶ One physical disk
- ▶ One physical adapter
- ▶ One Virtual I/O Server partition

Multipath I/O

Figure 6-9 on page 163 shows a configuration using Multipath I/O to access an ESS disk. The client partition sees two paths to the physical disk through MPIO. Each path is using a different virtual SCSI adapter to access the disk. Each of these virtual SCSI adapters is backed by a separate Virtual I/O Server partition.

Note: This type of configuration works only when the physical disk is assigned as a whole to the client partition. You cannot split up the physical disk into logical volumes at the Virtual I/O Server level.

This configuration protects a virtual disk in a client partition against failure of:

- ▶ One physical FC adapter in one Virtual I/O Server partition
- ▶ One Virtual I/O Server partition

Depending on your SAN topology, each physical adapter could be connected to a separate SAN switch to provide redundancy. At the physical disk level, the ESS provides redundancy because it uses RAID technology internally.

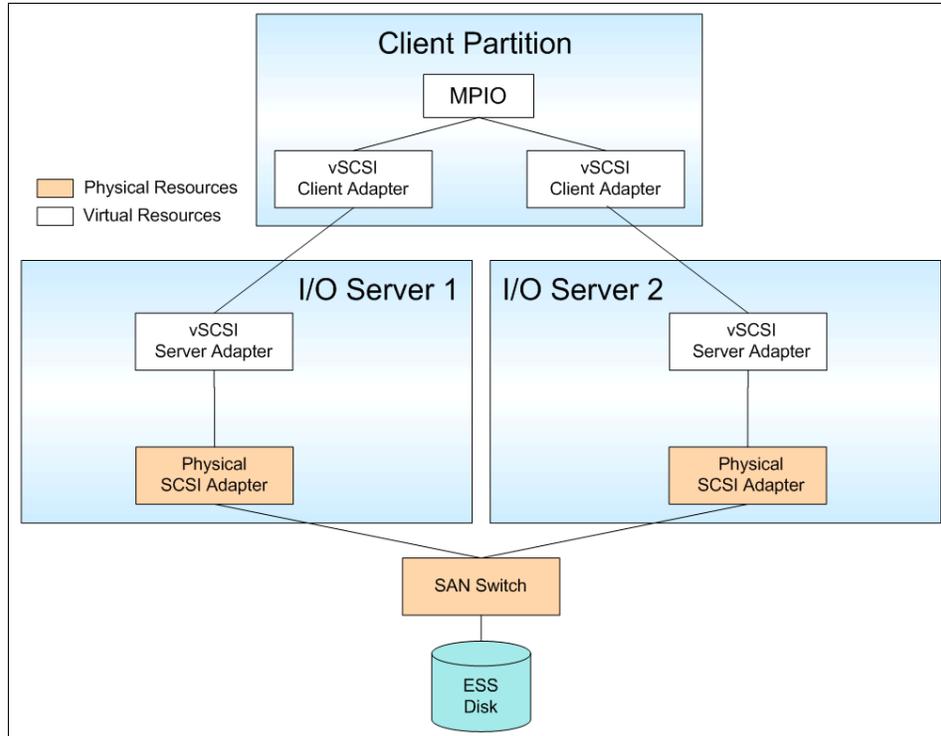


Figure 6-9 Virtual I/O Server configuration with MPIO

6.4 Virtual Serial Adapter (VSA)

The POWER Hypervisor supports three types of virtual I/O devices:

- ▶ Virtual LAN (VLAN; see 6.5, “Virtual Ethernet” on page 164)
- ▶ Virtual SCSI (VSCSI; see 6.8, “Virtual SCSI” on page 205)
- ▶ Virtual Serial Adapter (VSA).

The VSA can only be used for providing a virtual console to the partitions. This console is visible to the end user in the HMC display.

The virtual serial port cannot be used for any other purpose. For example, it cannot be used for HACMP heartbeat monitoring.

There are no specific performance considerations to address regarding the VSA.

6.5 Virtual Ethernet

Virtual Ethernet enables inter-partition communication without the need for physical network adapters assigned to each partition. Virtual Ethernet enables the administrator to define in-memory point-to-point connections between partitions. These connections exhibit characteristics similar to physical high-bandwidth Ethernet connections and support multiple protocols (IPv4, IPv6, ICMP). Virtual Ethernet requires an @server p5 system with either AIX 5L V5.3 or the appropriate level of Linux and an HMC to define the virtual Ethernet devices. Virtual Ethernet does not require the purchase of any additional features or software such as the Advanced POWER Virtualization feature.

6.5.1 Virtual LAN

This section discusses the concepts of Virtual LAN (VLAN) technology with specific reference to its implementation within AIX 5L V5.3.

Virtual LAN overview

Virtual LAN is a technology used for establishing virtual network segments on top of physical switch devices. If configured appropriately, a VLAN definition can straddle multiple switches.

In every partition, virtual and dedicated network devices can be used simultaneously for communication. Figure 6-10 shows adapters of a partition that has one virtual Ethernet adapter (ent0) and two real adapters (ent1 and ent2). Up to 256 adapters (sum of virtual and real) are supported per LPAR.

```
# lsdev -Cc adapter
ent0 Available Virtual I/O Ethernet Adapter (1-lan)
ent1 Available 01-08 2-Port 10/100/1000 Base-TX PCI-X Adapter (14108902)
ent2 Available 01-09 2-Port 10/100/1000 Base-TX PCI-X Adapter (14108902)
vsa0 Available LPAR Virtual Serial Adapter
vscsi0 Available Virtual SCSI Client Adapter
```

Figure 6-10 Virtual and local adapters on one partition

Typically, a VLAN is a broadcast domain that enables all nodes in the VLAN to communicate with each other without any L3 routing or inter-VLAN bridging. In Figure 6-11 on page 165, two VLANs (VLAN 1 and 2) are defined on three switches (Switch A, B, and C). Although nodes C-1 and C-2 are physically connected to the same switch C, traffic between two nodes can be blocked. To enable communication between VLAN 1 and 2, L3 routing or inter-VLAN bridging should be established between them; typically this is provided by an L3 device.

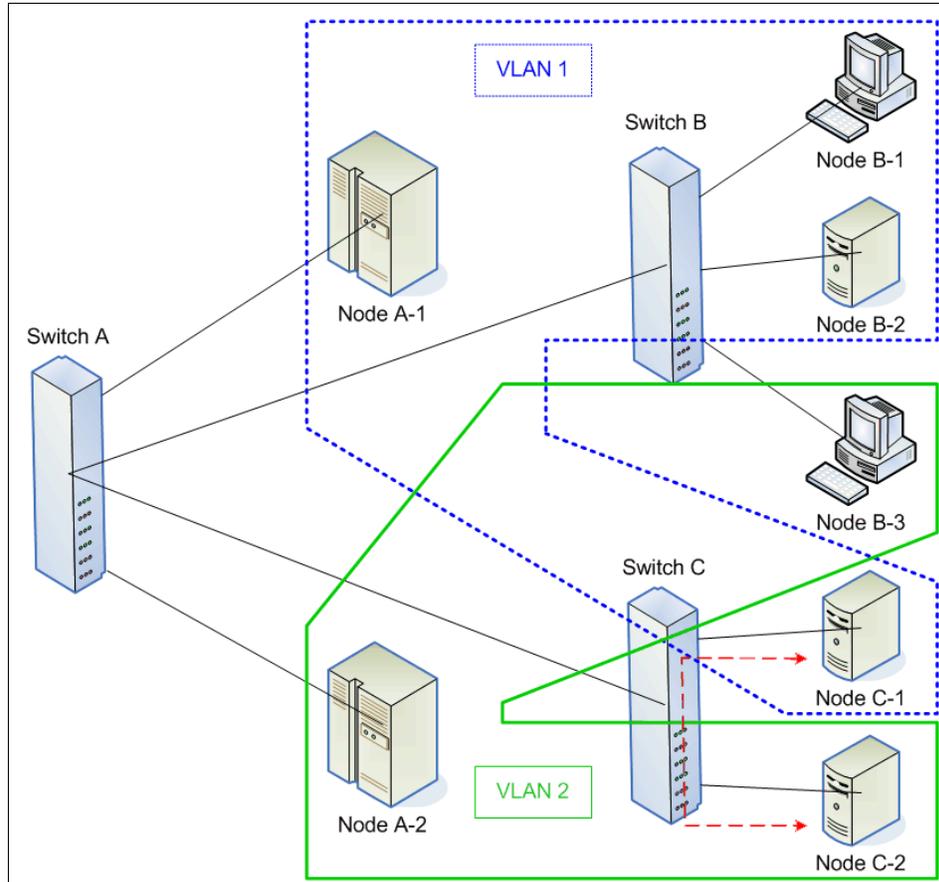


Figure 6-11 Example of a VLAN

The use of VLAN provides increased LAN security and flexible network deployment over traditional network devices.

AIX 5L V5.3 VLAN support

Some of the technologies for implementing VLANs include:

- ▶ Port-based VLAN
- ▶ Layer 2 VLAN
- ▶ Policy-based VLAN
- ▶ IEEE 802.1Q VLAN

VLAN support in AIX 5L V5.3 is based on the IEEE 802.1Q VLAN implementation. The IEEE 802.1Q VLAN is achieved by adding a VLAN ID tag to an Ethernet frame. The Ethernet switches restrict the frames to ports that are

authorized to receive frames with that VLAN ID. Switches also restrict broadcasts to the logical network by ensuring that a broadcast packet is delivered to all ports that are configured to receive frames with the VLAN ID that the broadcast frame was tagged with.

A port on a VLAN-capable switch has a default PVID (Port VLAN ID) that indicates the default VLAN the port belongs to. The switch adds the PVID tag to untagged packets that are received by that port. In addition to a PVID, a port may belong to additional VLANs and have those VLAN IDs assigned to it that indicate the additional VLANs that the port belongs to.

A port will only accept untagged packets or packets with a VLAN ID (PVID or additional VLANs) tag of the VLANs the port belongs to. A port configured in the untagged mode is only allowed to have a PVID and will receive untagged packets or packets tagged with the PVID. The untagged port feature helps systems that do not understand VLAN tagging communicate with other systems using standard Ethernet.

Each VLAN ID is associated with a separate Ethernet interface to the upper layers (for example, IP) and creates unique logical Ethernet adapter instances per VLAN (for example, ent1 or ent2).

You can configure multiple VLAN logical devices on a single system. Each VLAN logical device constitutes an additional Ethernet adapter instance. These logical devices can be used to configure the same Ethernet IP interfaces as are used with physical Ethernet adapters.

VLAN communication by example

This section discusses how VLAN communication between partitions and with external networks works in more detail, using the sample configuration in Figure 6-12 on page 167. The configuration uses four client partitions (Partition 1 through Partition 4) and one Virtual I/O Server partition. Each of the client partitions is defined with one virtual Ethernet adapter. The Virtual I/O Server partition has a Shared Ethernet Adapter that bridges traffic to the external network. The Shared Ethernet Adapter is discussed in more detail in 6.6, “Shared Ethernet Adapter” on page 186.

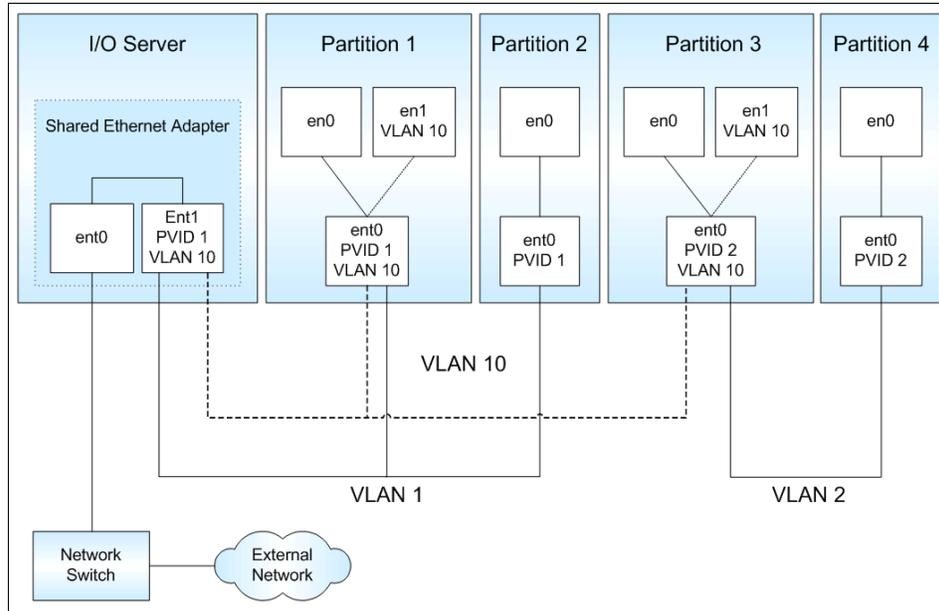


Figure 6-12 VLAN configuration

Interpartition communication

Partition 2 and Partition 4 are using only the PVID. This means that:

- ▶ Only packets for the VLAN specified as PVID are received.
- ▶ Packets that are sent have a VLAN tag added for the VLAN specified as PVID by the virtual Ethernet adapter.

In addition to the PVID, the virtual Ethernet adapters in Partition 1 and Partition 3 are also configured for VLAN 10 using specific network interface (en1) create through `smitty vlan`. This means that:

- ▶ Packets sent through network interfaces en1 are added a tag for VLAN 10 by the network interface in AIX 5L V5.3.
- ▶ Only packets for VLAN 10 are received by the network interfaces en1.
- ▶ Packets sent through en0 are automatically tagged for the VLAN specified as PVID.

Only packets for the VLAN specified as PVID are received by the network interfaces en0. Table 6-4 on page 168 lists which *client* partitions can communicate with each other through what network interfaces.

Table 6-4 Interpartition VLAN communication

VLAN	Partition / network interface
1	Partition 1 / en0 Partition 2 / en0
2	Partition 3 / en0 Partition 4 / en0
10	Partition 1 / en1 Partition 3 / en1

Communication with external networks

The Shared Ethernet Adapter is configured with PVID 1 and VLAN 10. This means that untagged packets that are received by the Shared Ethernet Adapter are tagged for VLAN 1. Handling of outgoing traffic depends on the VLAN tag of the outgoing packets.

- ▶ Packets tagged with the VLAN that matches the PVID of the Shared Ethernet Adapter are untagged before being sent out to the external network.
- ▶ Packets tagged with a VLAN *other than* the PVID of the Shared Ethernet Adapter are sent out with the VLAN tag unmodified.

In our example, Partition 1 and Partition 2 have access to the external network through network interface en0 using VLAN 1. Since these packets are using the PVID, the Shared Ethernet Adapter will remove the VLAN tags before sending the packets to the external network.

Partition 1 and Partition 3 have access to the external network using network interface en1 and VLAN 10. These packets are sent out by the Shared Ethernet Adapter with the VLAN tag. Therefore, only VLAN-capable destination devices will be able to receive the packets. Table 6-5 lists this relationship.

Table 6-5 VLAN communication to external network

VLAN	Partition / Network interface
1	Partition 1 / en0 Partition 2 / en0
10	Partition 1 / en1 Partition 3 / en1

6.5.2 Virtual Ethernet connections

Virtual Ethernet connections supported in POWER5 processor-based systems use VLAN technology to ensure that the partitions can access only data directed to them. The POWER Hypervisor provides a virtual Ethernet switch function based on the IEEE 802.1Q VLAN standard that enables partition communication within the same server. The connections are based on an implementation internal to the Hypervisor that moves data between partitions. This section describes the various elements of a virtual Ethernet and implications relevant to different types of workloads. Figure 6-13 is an example of an inter-partition VLAN.

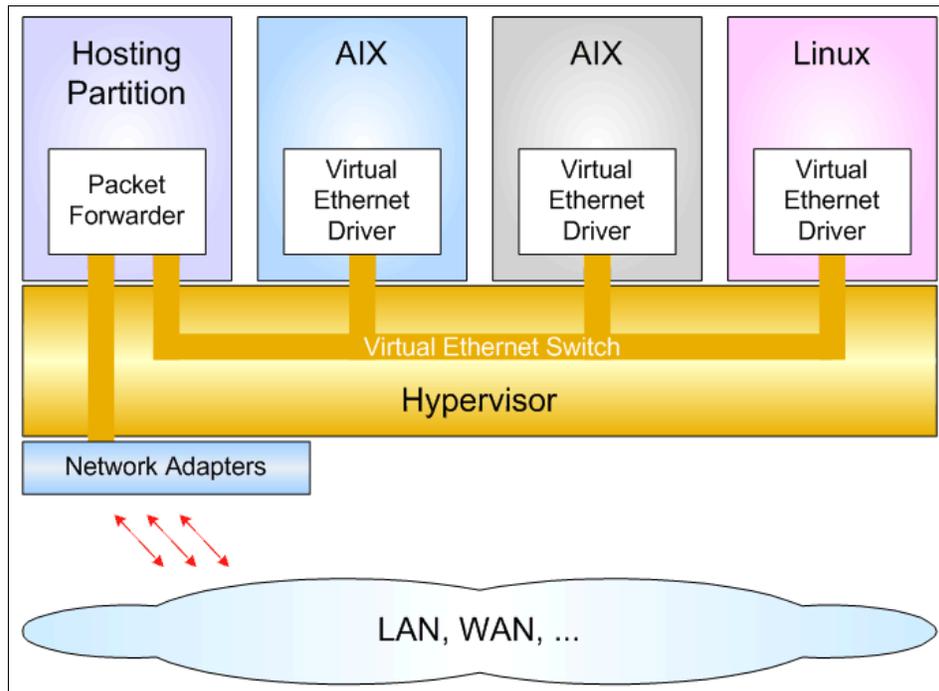


Figure 6-13 logical view of an inter-partition VLAN

Virtual Ethernet concepts

Partitions that communicate through a virtual Ethernet channel must have an additional in-memory channel. This requires the creation of an in-memory channel between partitions on the HMC. The kernel creates a virtual device for each memory channel indicated by the firmware. The AIX 5L V5.3 configuration manager creates the device special files. A unique MAC address is also generated when the virtual Ethernet device is created. A *prefix* value can be assigned for the system so that the generated MAC addresses in a system

consist of a common system prefix plus an algorithmically generated unique part per adapter.

The virtual Ethernet can also be used as a bootable device to enable such tasks as operating system installations to be performed using NIM.

MTU Sizes

The virtual Ethernet adapter supports, as Gigabit (Gb) Ethernet, Standard MTU-Sizes of 1500 bytes and Jumbo frames with 9000 bytes. Additionally to physical Ethernet, the *MTU-Size of 65280 bytes* is also supported in virtual Ethernet. So, the MTU of 65280 bytes can be only used inside a virtual Ethernet.

IPv6 Support

Virtual Ethernet supports multiple protocols, such as IPv4 and IPv6.

6.5.3 Benefits of virtual Ethernet

Due to the number of possible partitions on many systems being greater than the number of I/O slots, virtual Ethernet is a convenient and cost-saving option to enable partitions within a single system to communicate with one another through a VLAN. The VLAN creates logical Ethernet connections between one or more partitions and is designed to help prevent a failed or malfunctioning operating system from being able to affect the communication between two functioning operating systems. The virtual Ethernet connections may also be *bridged* to an external network to permit partitions without physical network adapters to communicate outside of the server.

The transmission speed of virtual Ethernet is in the range of 1 Gb to 3 Gb per second, depending on the transmission (MTU) size. A partition can support up to 256 virtual Ethernet adapters with each virtual Ethernet capable of being associated with up to 21 VLANs (20 VID and 1 PVID).

A virtual Ethernet adapter appears to the operating system in the same way as a physical adapter. It also can be configured in the same manner. While the MAC address of physical Ethernet is coded on the (hardware) adapter, the MAC address of the virtual adapter is generated by the HMC.

6.5.4 Limitations and considerations

Consider the following limitations when implementing a virtual Ethernet:

- ▶ A maximum of up to 256 virtual Ethernet adapters are permitted per partition.
- ▶ Virtual Ethernet can be used in both shared and dedicated processor partitions if the partition is running AIX 5L V5.3 or Linux with the 2.6 kernel or a kernel that supports virtualization.
- ▶ A mixture of virtual Ethernet connections, real network adapters, or both are permitted within a partition.
- ▶ Virtual Ethernet requires a POWER5 processor–based system and an HMC to define the virtual Ethernet adapters.
- ▶ Virtual Ethernet can connect only partitions within a single system.
- ▶ Virtual Ethernet connections from AIX 5L or Linux partitions to an i5/OS partition may work; however, when this book was being written these capabilities were unsupported.
- ▶ Virtual Ethernet uses the system processors for all communication functions instead of offloading the load to processors on network adapter cards, so an increase in system processor load is generated by the use of virtual Ethernet.

6.5.5 POWER Hypervisor switch implementation

The POWER Hypervisor switch is consistent with IEEE 802.1 Q. It works on OSI-Layer 2 and supports up to 4096 networks (4096 VLAN IDs).

When a message arrives at a Logical LAN switch port from a Logical LAN adapter, the POWER Hypervisor caches the message's source MAC address to use as a filter for future messages to the adapter. The POWER Hypervisor then processes the message differently depending on whether the port is configured for IEEE VLAN headers. If the port is configured for VLAN headers, the VLAN header is checked against the port's allowable VLAN list. If the message specified VLAN is not in the port's configuration, the message is dropped. After the message passes the VLAN header check, it passes onto destination MAC address processing.

If the port is *not* configured for VLAN headers, the POWER Hypervisor inserts a two-byte VLAN header (based on the port's configured VLAN number) into the message. Next, the destination MAC address is processed by searching the table of cached MAC addresses.

If a match for the MAC address is not found and if no *trunk adapter* is defined for the specified VLAN number, the message is dropped; otherwise, if a match for the MAC address is not found and if a trunk adapter is defined for the specified

VLAN number, the message is passed on to the trunk adapter. If a MAC address match is found, then the associated switch port's configured, allowable VLAN number table is scanned for a match to the VLAN number contained in the message's VLAN header. If a match is not found, the message is dropped.

Next, the VLAN header configuration of the destination switch port is checked. If the port is configured for VLAN headers, the message is delivered to the destination Logical LAN adapters, including any inserted VLAN header. If the port is configured for no VLAN headers, the VLAN header is removed before being delivered to the destination Logical LAN adapter.

Figure 6-14 on page 173 shows a graphical representation of the behavior of the virtual Ethernet when processing packets.

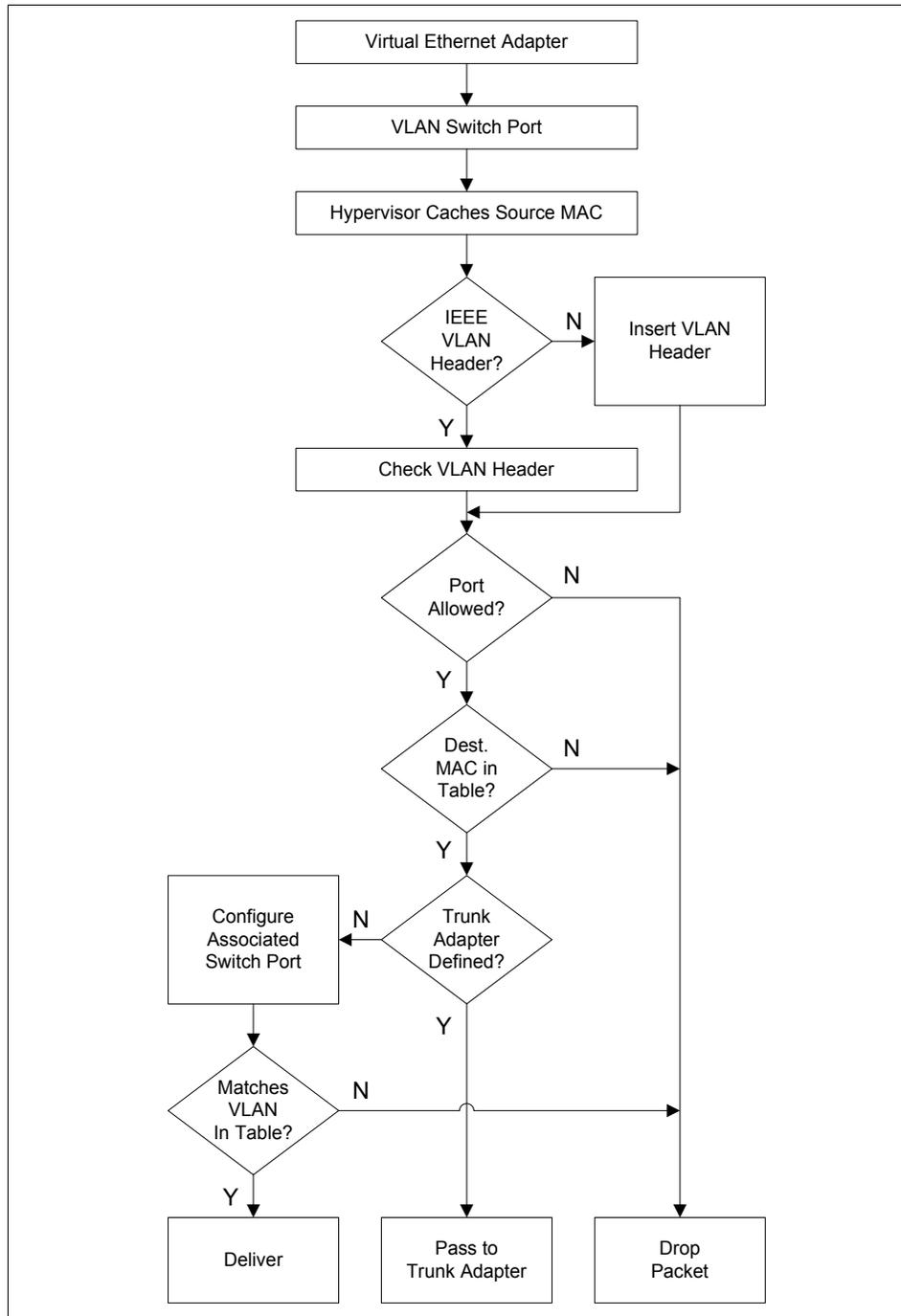


Figure 6-14 Flow chart of virtual Ethernet

6.5.6 Performance considerations

This section presents several experiments that were performed on an @server p5 server to measure the influence of some parameters that a system administrator can set.

General comments about measurements

The operating system that ran on all partitions was AIX 5L V5.3. The results of the measurements could vary if they would be repeated at a later time with updates to the operating system and firmware.

The platform that was used for these tests was a four-way 1.65 GHz IBM @server p5 570.

Unless otherwise mentioned, the VLAN connections were set up between two partitions, each configured with one dedicated processor. Simultaneous multithreading was enabled.

The virtual Ethernet and physical Ethernet adapters were tested with their default interface specific network options (as defined in the `no` command) and Object Data Manager (ODM) settings. Specifically, these were:

Virtual Ethernet For MTU 1500, `tcp_sendspace=131072`,
`tcp_recvspace=65536`

For MTU 9000, `tcp_sendspace=262144`,
`tcp_recvspace=131072`, `rfc1323=1`

For MTU 65394, `tcp_sendspace=262144`,
`tcp_recvspace=131072`, `rfc1323=1`

Physical Ethernet Gigabit Ethernet

For MTU 1500, `tcp_sendspace=131072`,
`tcp_recvspace=65536`

For MTU 9000, `tcp_sendspace=262144`,
`tcp_recvspace=131072`, `rfc1323=1`

The adapter defaults were used, which include `large_send` (also known as TCP segmentation off load), TCP checksum off load, and interrupt coalescing. The ODM attributes were: `large_send=1`, `chksum_offload=1` and `intr_rate=10000`.

Description of the performance tests and tools

To measure the VLAN performance, the benchmark used was `netperf`. This benchmark can be used to measure various aspects of networking performance. Currently, it focuses on bulk data transfer (streaming) and request/response

performance using either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP), with the Berkeley Sockets interface.

This benchmark is now part of the public domain and can be found at:

<http://www.netperf.org/netperf/NetperfPage.html>

IBM has developed a derivative version that is more tightly integrated with the capabilities of the AIX 5L V5.3 operating system. All measurements described in this book use the IBM-modified version of **netperf**.

The experiment results presented later use both operational modes of **netperf**: streaming mode, called TCP_STREAM, and transactional request/response mode, called TCP_RR.

TCP_STREAM This benchmark performs the data streaming test between the local system and the remote system. TCP_STREAM is used in simplex and duplex mode. In simplex mode, one side sends and the other end receives data; in duplex mode, both ends send and receive at the same time. So the amount of data that is transported via the media will increase. The TCP_STREAM benchmark can be performed with a different data chunk size. The results presented here are for an application that sends data chunks between 16 KB and 64 KB to the communication sockets (which then split them into IP packets depending on the MTU size).

TCP_RR **netperf** request/response performance is quoted as transactions per second for a given request and response site. A transaction is defined as the exchange of a single request and a single response. From a transaction rate, one can infer round-trip average latency. The TCP_RR benchmarks are done with one and 20 sessions. Unlike the one-session test, the 20-session test shows how the response time and latency is growing with more load.

In each mode, four programs called *sessions* are used. These sessions send traffic over the connection to simulate a real workload with multiple IP sessions flowing through the same adapter.

Overview of the following benchmark measurements

The first measurement shows how throughput is growing by adding more entitlements to a virtual processor, then a test compares parameters such as processor utilization, transaction rate, and latency in both physical and virtual networks. The last set of measurements shows the difference in performance of the VLAN using single-threaded and simultaneous multithreading modes.

6.5.7 VLAN throughput at different processor entitlements

This purpose of this test is to see what throughput might be expected in a VLAN. Because the throughput varies with processor entitlements and MTU size, these parameters are variable in the measurement.

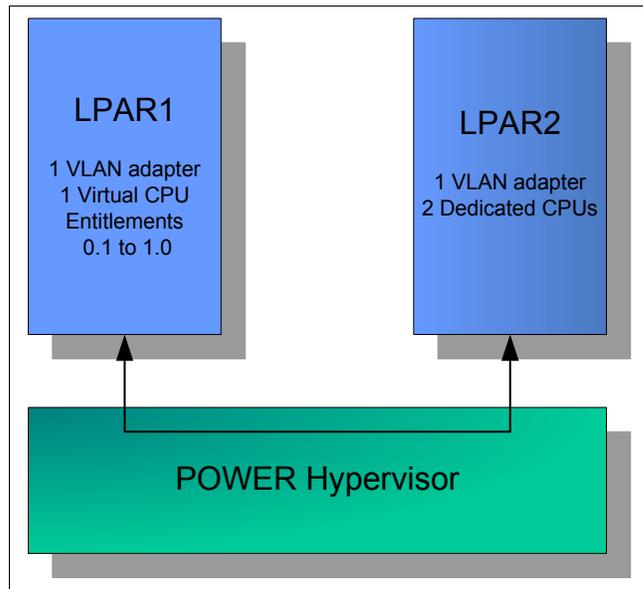


Figure 6-15 Processor entitlements and MTU sizes

Both LPARs have one VLAN adapter, and there are multiple sessions running between adapters. The benchmark used for this test is **netperf** TCP_STREAM.

LPAR1, with varied processor entitlements, is sending a simplex stream. LPAR2, with two dedicated processors, receives it.

The goal of the test was to measure the performance of LPAR1, so resources for LPAR 2 are oversized using two dedicated processors, so there is no bottleneck on the receiving side that would affect the measurement. This enables the throughput of the VLAN interface of LPAR1 to be effectively measured as a function of the CPU entitlement of LPAR1.

Figure 6-16 on page 177, Figure 6-17 on page 177, and Figure 6-18 on page 178 show the results of the performance measurements that were taken using varying processor entitlements and MTU sizes of 1500, 9000, and 65394 bytes.

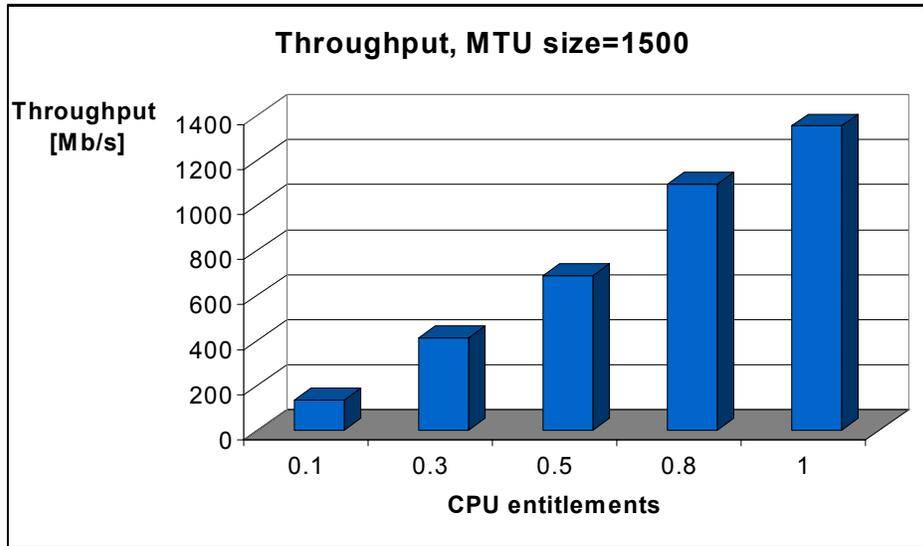


Figure 6-16 Throughput versus CPU entitlements, MTU size=1500

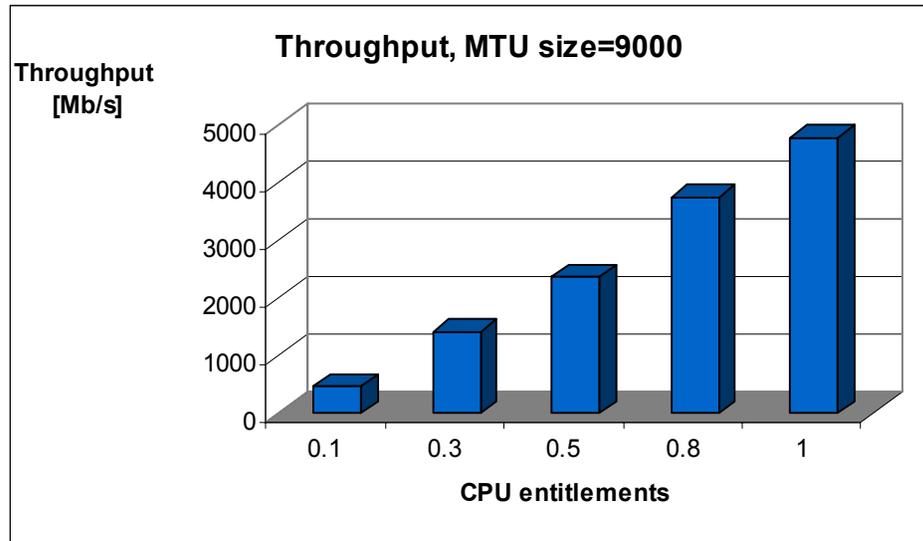


Figure 6-17 Throughput versus CPU entitlements, MTU size=9000

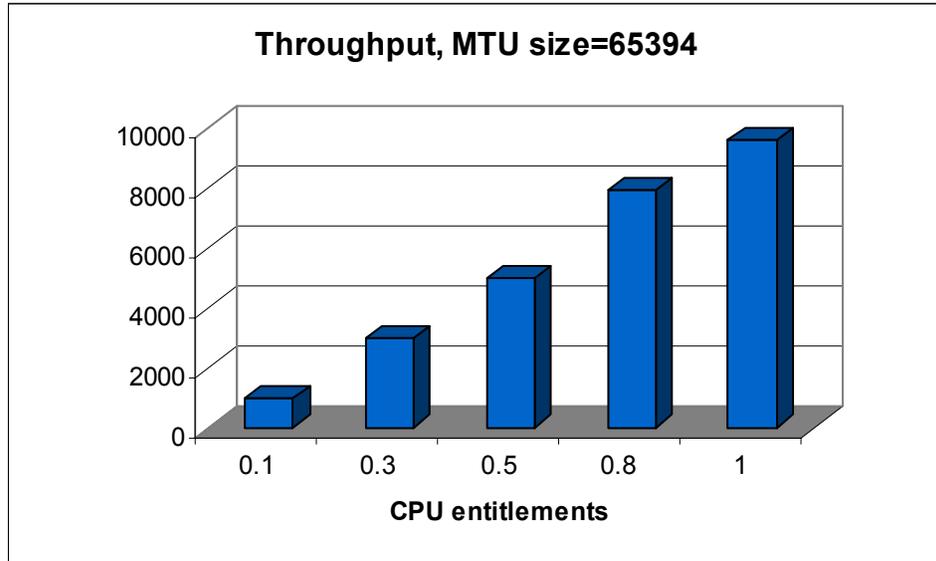


Figure 6-18 Throughput versus CPU entitlements, MTU size=65394

VLAN performance

The throughput of the VLAN scales nearly linear with the allocated processor entitlements. Throughput with MTU=9000 is more than three times the rate with MTU=1500, and the throughput with MTU=65394 is more than seven times the rate with MTU=1500. This is due to improved efficiency of sending larger packets with one call up or down the TCP/IP protocol stack.

6.5.8 Comparing throughput of VLAN to physical Ethernet

In the next set of tests, a performance comparison of the VLAN and the physical Ethernet adapter was made. Both LPARs are assigned one dedicated POWER5 processor, and ran in simultaneous multithreading mode.

Figure 6-19 on page 179 and Figure 6-20 on page 179 show the two different types of connections between the LPARs and VLAN through the POWER Hypervisor and physical Ethernet using a 1 Gb/s Ethernet switch.

The benchmark TCP_STREAM was running in simplex and duplex mode at different MTU sizes on both setups, measuring throughput and processor utilization.

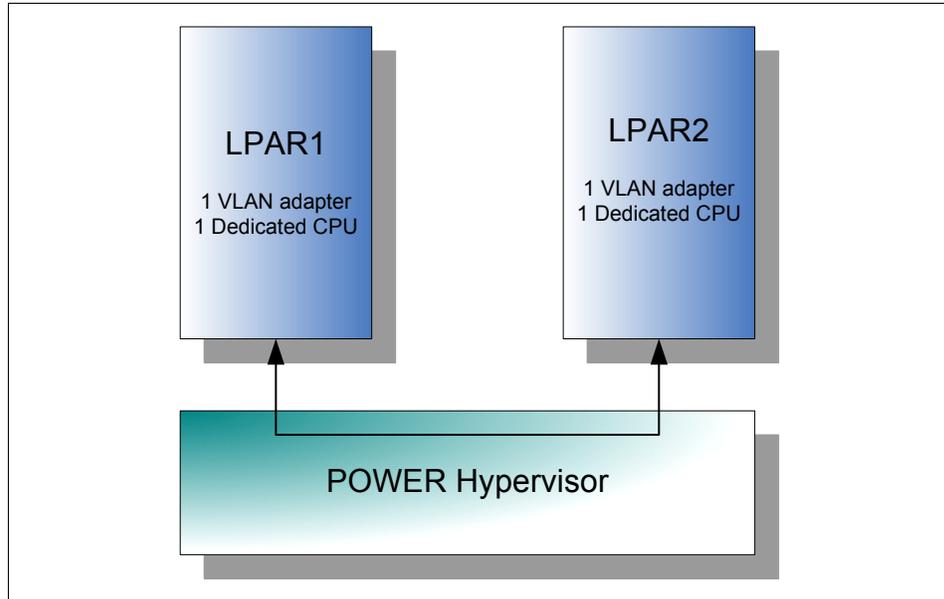


Figure 6-19 VLAN performance configuration

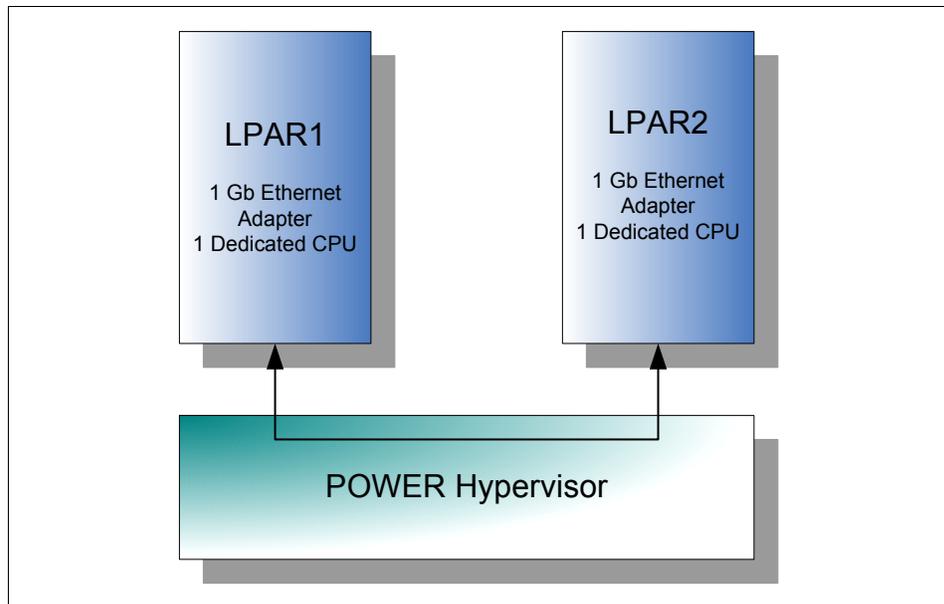


Figure 6-20 Physical Ethernet configuration

VLAN and physical Ethernet performance

Figure 6-21 shows how throughput varies with different values of MTU size in simplex and duplex modes. (The physical Ethernet adapter does not support an MTU size of 65394.)

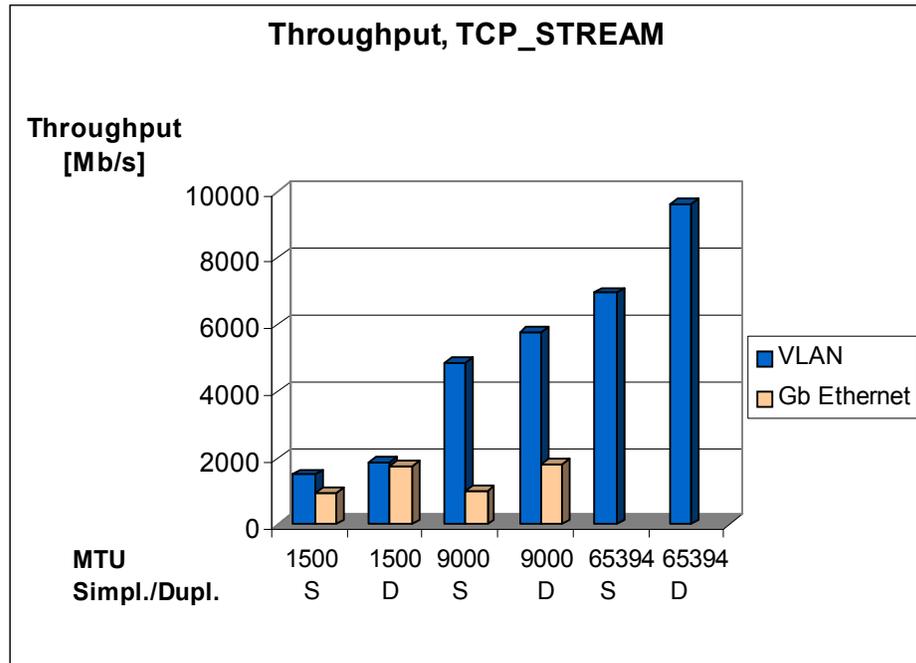


Figure 6-21 VLAN and physical Ethernet using TCP_STREAM

The VLAN adapter has a higher raw throughput at all MTU sizes. With an MTU size of 9000 bytes, the throughput difference is very large (four to five times) because the physical Ethernet adapter is running at wire speed (989 Mbit/s user payload), but the VLAN can run much faster because it is limited only by CPU and memory-to-memory transfer speeds.

6.5.9 Comparing CPU utilization

These measurements use the same configurations as shown in Figure 6-19 on page 179 and Figure 6-20 on page 179, and with the same TCP_STREAM workload. CPU utilization is shown for different MTU sizes, in both simplex and duplex mode, in Figure 6-22 on page 181 and Figure 6-23 on page 181.

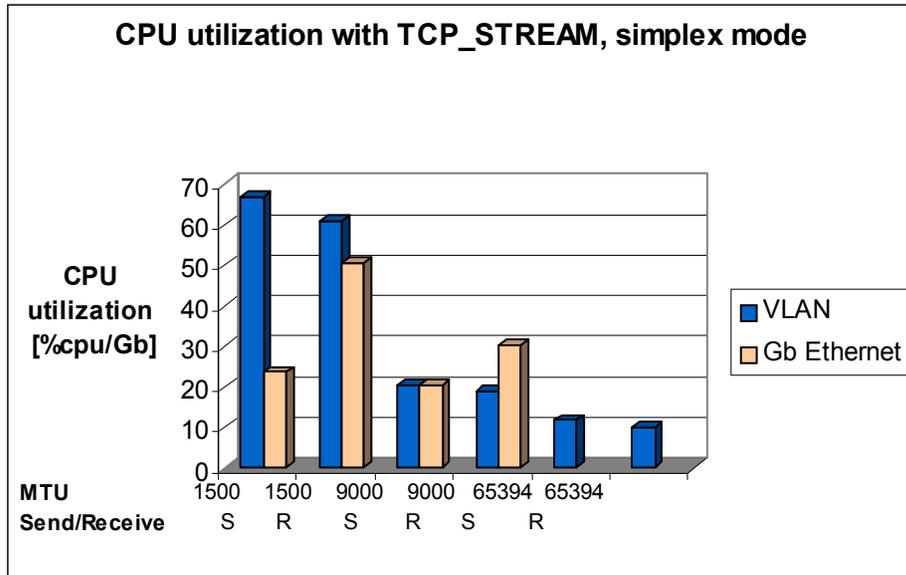


Figure 6-22 CPU utilization with TCP_STREAM, simplex mode

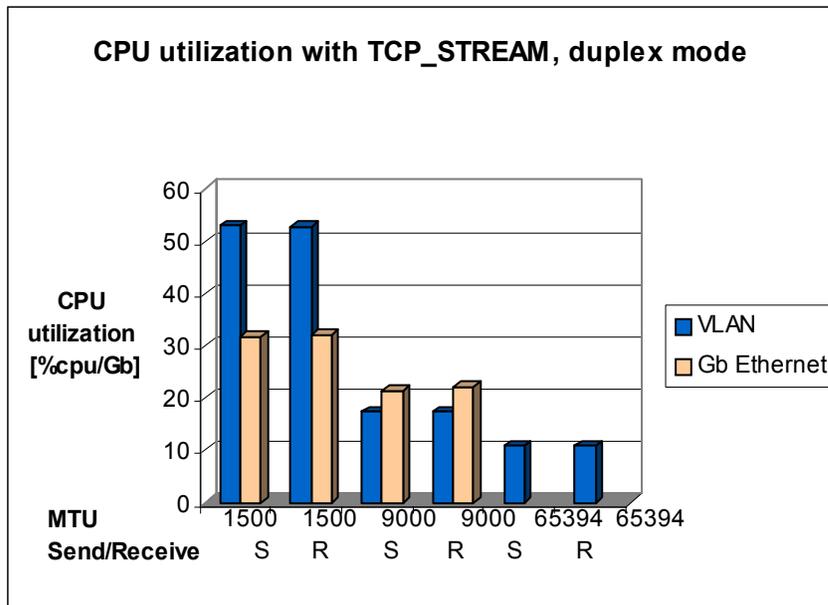


Figure 6-23 CPU utilization with TCP_STREAM, duplex mode

VLAN and physical Ethernet performance

As expected, the CPU utilization of the VLAN is higher than the throughput of physical Ethernet. As with most adapter cards, the physical Ethernet adapter has a processor on it to perform the memory transfers of the packets to and from the adapter card. The VLAN requires the POWER Hypervisor to do the memory transfers, resulting in higher CPU utilization. To compare CPU utilization, the results are normalized to 1 Gb throughput for both the VLAN and physical Ethernet. In addition, another difference in CPU utilization between the virtual Ethernet and the physical Ethernet adapter when using MTU 1500 is the effect of having the attributes `large_send` and `checksum_offload` enabled on the physical adapter. These two features reduce the CPU utilization for physical Ethernet, but they are not available on virtual Ethernet.

6.5.10 Comparing transaction rate and latency

These measurements were obtained using the configurations shown in Figure 6-19 on page 179 and Figure 6-20 on page 179. The TCP_RR workload was used to get a value for number of transactions and latency. TCP_RR is used with two different parameters for the number of sessions (1 and 20), which is a measure for different workloads.

The results are presented in two charts. Figure 6-24 shows the transaction rate for MTU size of 1500 and 9000 and for 1 and 20 sessions. Figure 6-25 on page 183 shows the latency for the same parameters.

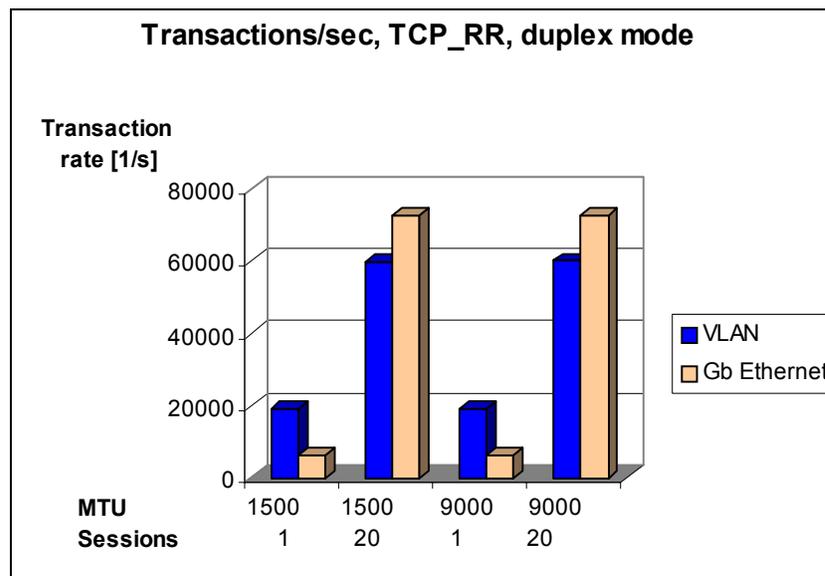


Figure 6-24 Transaction rate at different MTU sizes and 1 and 20 sessions

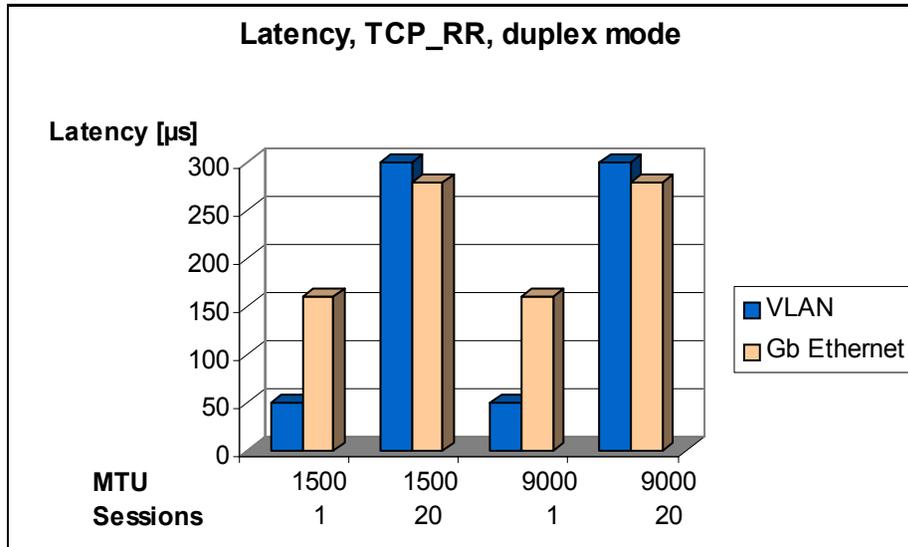


Figure 6-25 Latency at different MTU sizes and 1 and 20 sessions

VLAN and physical Ethernet performance

The virtual Ethernet has lower latency for light workloads than the physical Ethernet adapter. This is because the Ethernet adapter has interrupt coalescing enabled by default (ODM attribute `intr_rate=10000`). This adds latency to the adapter's single session test, but it helps reduce CPU utilization for higher transaction rate workloads (such as the 20-session test), which is why the throughput is similar at 20 sessions. The latency can be reduced by disabling interrupt coalescing (set the adapters `intr_rate=0`). The virtual Ethernet does not support any method of interrupt coalescing.

The physical Ethernet has lower latency in heavy workloads because interrupt coalescing is enabled by default on the adapter.

6.5.11 VLAN performance

This purpose of this test was to show the performance gain of running the processor in simultaneous multithreading mode. The configuration was the same as shown in Figure 6-19 on page 179. For this comparison, both TCP_STREAM and TCP_RR workloads are used.

Figure 6-21 on page 180 showed the results of VLAN throughput. The following charts show the percent gain in throughput when comparing simultaneous multithreading to single-threaded mode. Figure 6-26 on page 184 has results for TCP_STREAM, and Figure 6-27 on page 185 illustrates the results for TCP_RR.

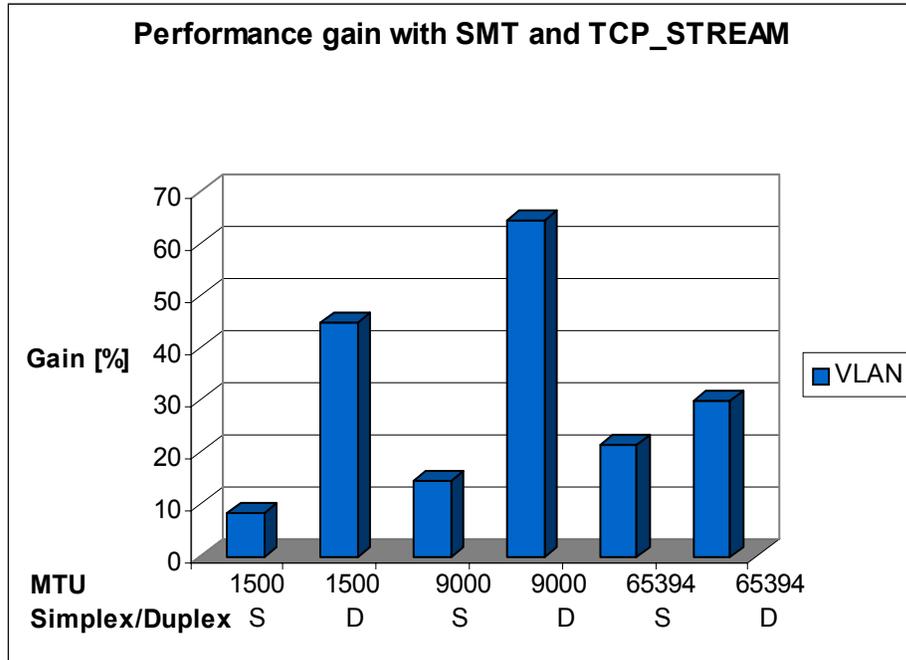


Figure 6-26 Performance gain with simultaneous multithreading, TCP_STREAM

The VLAN benefits from simultaneous multithreading because it is not limited by media speed and takes advantage of the extra available processor cycles.

The reason for negative scaling in Figure 6-27 on page 185 when simultaneous multithreading is enabled is that at very small workloads (which is the case when there is only one TCP_RR session), running in single-threaded mode is more efficient. With simultaneous multithreading enabled, the system disables the second thread when the load on the system is light but checks periodically to determine whether it needs to reactivate it. This checking, disabling, and enabling of the second thread tends to affect the latency of the TCP_RR transactions, thus reducing throughput.

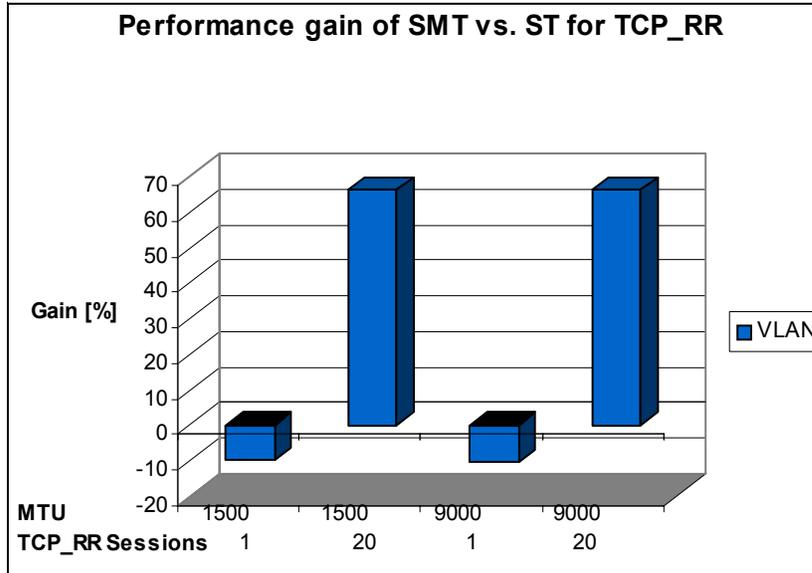


Figure 6-27 Performance gain with simultaneous multithreading, TCP_RR

6.5.12 VLAN implementation guidelines

Because there is little experience with VLANs before now, we offer some guidelines for designing VLANs.

Important: The following recommendations have no guarantee for enhancing performance; they are merely provided as suggestions.

1. Know your environment and the network traffic.
2. Choose the MTU size as high as it makes sense for network traffic in the VLAN.
3. Use an MTU size of 65394 if you expect a large amount of data to be copied in your VLAN.
4. Keep the `tcp_pmtu_discover` attribute set to its default value (active discovery).
5. If the VLAN is to be bridged to a Shared Ethernet Adapter for access to an external network, set the MTU size of the VLAN in the client partition to the value used for the definition of the Shared Ethernet Adapter on the Virtual I/O Server partition.
6. Do not turn off simultaneous multithreading unless the applications demand it.
7. The VLAN throughput scales linearly with processor entitlements, so there is no need to dedicate processors to partitions because of VLAN performance.

6.6 Shared Ethernet Adapter

A Virtual I/O Server partition is not required for implementing a VLAN. Virtual Ethernet adapters can communicate with each other via the POWER Hypervisor without the functionality of the Virtual I/O Server.

Bridging from the VLAN to the physical LAN can be accomplished in two ways:

- ▶ Routing
- ▶ Shared Ethernet Adapter

By enabling the AIX 5L V5.3 routing capabilities (ipforwarding network option), one partition with a physical Ethernet adapter connected to an external network can act as a *router*. Figure 6-28 shows a sample configuration. In this type of configuration the partition that routes the traffic to the external work does not necessarily have to be the Virtual I/O Server as in the pictured example. It could be any partition with a connection to the outside world. The client partitions would have their default route set to the partition that routes traffic to the external network.

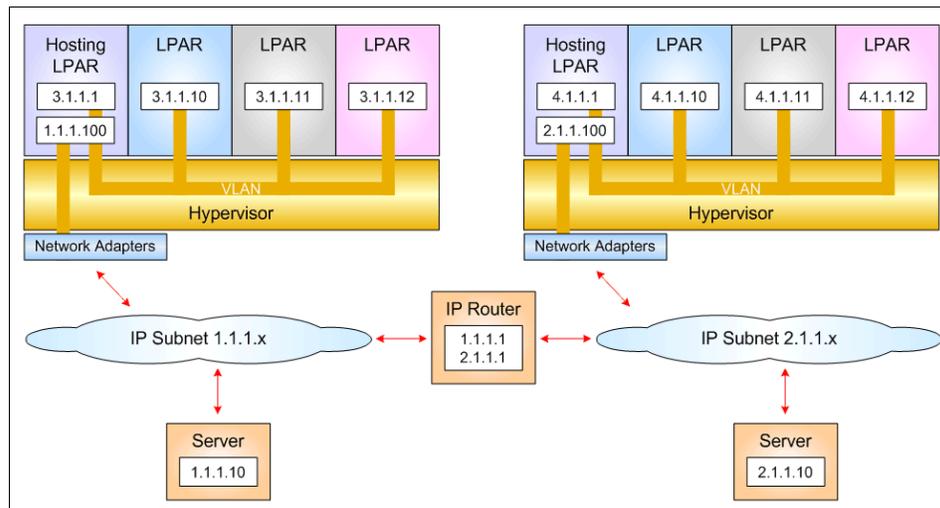


Figure 6-28 Connection to external network using AIX 5L V5.3 routing

Using a Shared Ethernet Adapter, you can connect internal and external VLANs using one physical adapter. The Shared Ethernet Adapter hosted in the Virtual I/O Server partition acts as an OSI Layer 2 switch between the internal and external network.

Figure 6-29 shows the Shared Ethernet Adapter being used as a bridge between the virtual Ethernet and physical Ethernet.

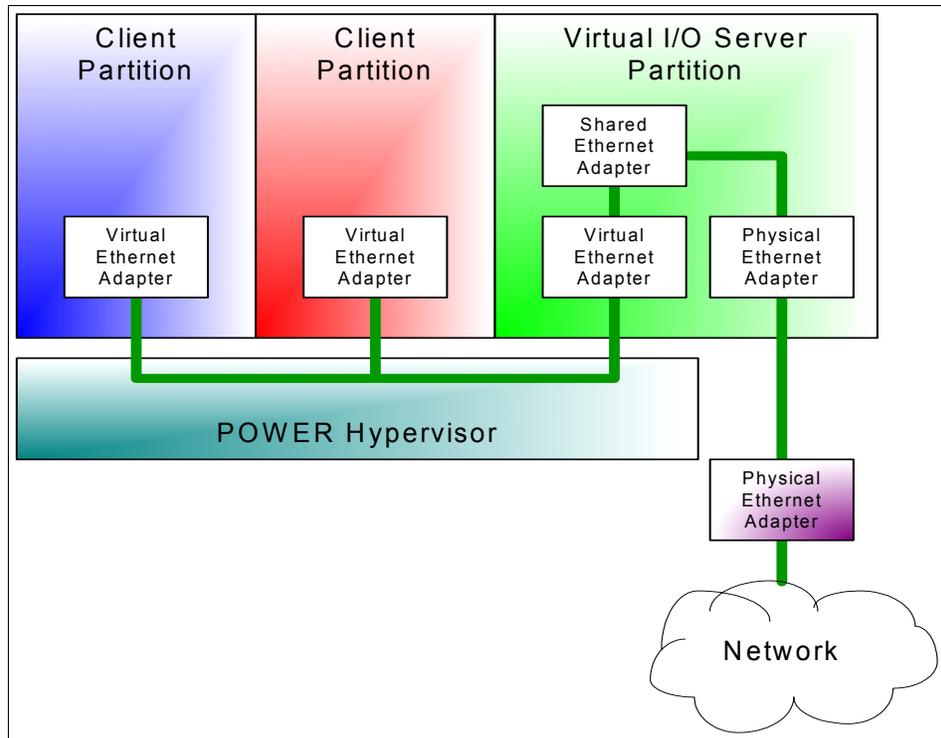


Figure 6-29 Shared Ethernet Adapter configuration

The bridge interconnects the logical and physical LAN segments at the network interface layer level and forwards frames between them. The bridge performs the function of a MAC relay (OSI Layer 2), and is independent of any higher layer protocol. Figure 6-30 on page 188 is a close-up view of the Virtual I/O Server partition.

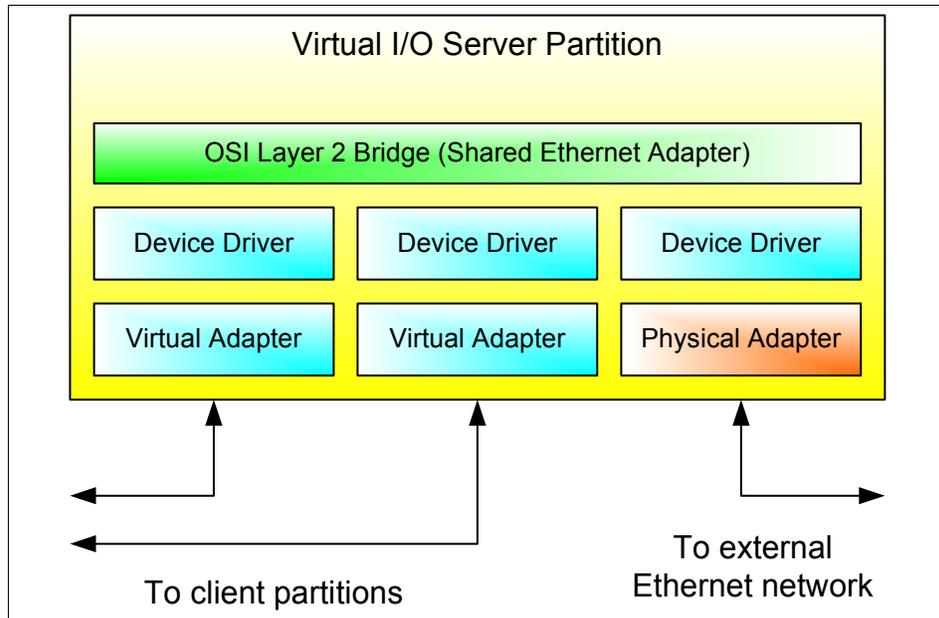


Figure 6-30 Sharing a (physical) Ethernet adapter on OSI layers

The bridge is transparent to the Internet Protocol (IP) layer. For example, when an IP host sends an IP datagram to another host on a network connected by a bridge, it sends the datagram directly to the host. The datagram “crosses” the bridge without the sending IP host being aware of it.

The Virtual I/O Server partition offers broadcast and multicast support. Address Resolution Protocol (ARP) and Neighbor Discovery Protocol (NDP) also work across the Shared Ethernet Adapter.

The Virtual I/O server does not reserve bandwidth on the physical adapter for any of the VLAN clients that send data to the external network. Therefore, if one client partition of the Virtual I/O Server sends data, it can take advantage of the full bandwidth of the adapter, assuming that the other client partitions do not send or receive data over the network adapter at the same time.

The following steps enable this connectivity:

1. Define the virtual Ethernet adapter on the I/O Server.

This is to be done on the HMC.

2. Define the virtual Ethernet adapters on the AIX 5L V5.3 or Linux partitions.

This definition is done on the HMC and is not a Virtual I/O Server function. It creates virtual Ethernet adapters that can be used like any other Ethernet

adapter. Different virtual networks can be separated using IEEE802.1Q-compatible VLAN features of the virtual Ethernet adapters.

3. Define the Shared Ethernet Adapter in the Virtual I/O Server partition.

The I/O Server acts as a bridge and forwards the IP packages using the virtual Ethernet connections to the AIX 5L V5.3 or Linux partitions.

The implementation of virtual Ethernet adapters on an IBM *@server* p5 system within Linux is assigned one IEEE VLAN-aware virtual Ethernet switch in the system. All partitions talking on the Ethernet are peers. Up to 4,096 separate IEEE VLANs can be defined. Each partition can have up to 65,533 virtual Ethernet adapters connected to the virtual switch. Each adapter can be connected to 21 IEEE VLANs (20 VID and 1 PVID).

The enablement and setup of a virtual Ethernet does not require any special hardware or software. After a specific virtual Ethernet is enabled for a partition, a network device named ethXX is created in the partition. The user can then set up TCP/IP configuration appropriately to communicate with other partitions. For information about network TCP/IP setup and configuration tools, see your AIX 5L V5.3 or Linux distribution documentation.

To define the Shared Ethernet Adapter (SEA) in the Virtual I/O Server partition, use the `mkvdev` command. The syntax is:

```
mkvdev -sea TargetDevice -vadapter VirtualEthernetAdapter ...
        -default DefaultVirtualEthernetAdapter
        -defaultid SEADefaultPVID [-attr Attributes=Value ...]
```

Using the example in Figure 6-31 on page 190, the target devices are the physical adapters (for example, ent0 and ent1). The virtual devices are ent2, ent3, and ent4, and the default ID is the default PVID associated with the default virtual Ethernet adapter.

Important: To set up the Shared Ethernet Adapter, all involved virtual and physical Ethernet interfaces have to be unconfigured (down or detached).

The following commands are required to set up the Shared Ethernet Adapter for this example:

```
$ mkvdev -sea ent0 -vadapter ent2 -default ent2 -defaultid 1
$ mkvdev -sea ent1 -vadapter ent3 ent4 -default ent3 -defaultid 2
```

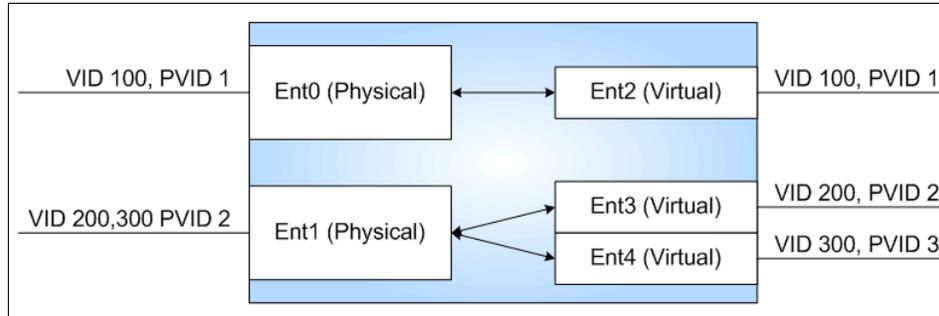


Figure 6-31 Example of Shared Ethernet Adapter bridging

In the second example, the physical Ethernet adapter is ent1. With the `mkvdev` command, we map the virtual Ethernet adapter ent3 and ent4 to the physical adapter. Additionally, ent3 is defined as a default adapter with the default VLAN ID of 2. This means that untagged packets received by the Shared Ethernet Adapter are tagged with the VLAN 2 ID and are sent to the virtual Ethernet adapter ent3.

After running the `mkvdev` command, the system will create the Shared Ethernet Adapter ent5. You now can configure the ent5 interface with an IP address using the `mktcpip` command.

6.6.1 Shared Ethernet Adapter performance

This test environment was conducted using the same conditions as described in “General comments about measurements” on page 174.

Figure 6-32 on page 191 shows the setup of the experiment. The communication path starts on a client partition that has a single dedicated processor, and is connected via a VLAN adapter, through the POWER Hypervisor to the VLAN adapter of the Virtual I/O Server partition, which bridges the virtual Ethernet adapter to a physical Ethernet adapter that is connected via a gigabit Ethernet network to a two-way POWER4+ processor-based server. The Virtual I/O Server runs in a partition with a single dedicated 1.65 GHz POWER5 processor.

The TCP_STREAM workload as described in “Description of the performance tests and tools” on page 174 is used to examine the throughput.

Note: The measurements are not done with a Gigabit Ethernet switch. Instead, a physical point-to-point connection (crossover cable) was used so there is no falsification of the measurement due to the internal behavior of a real switch.

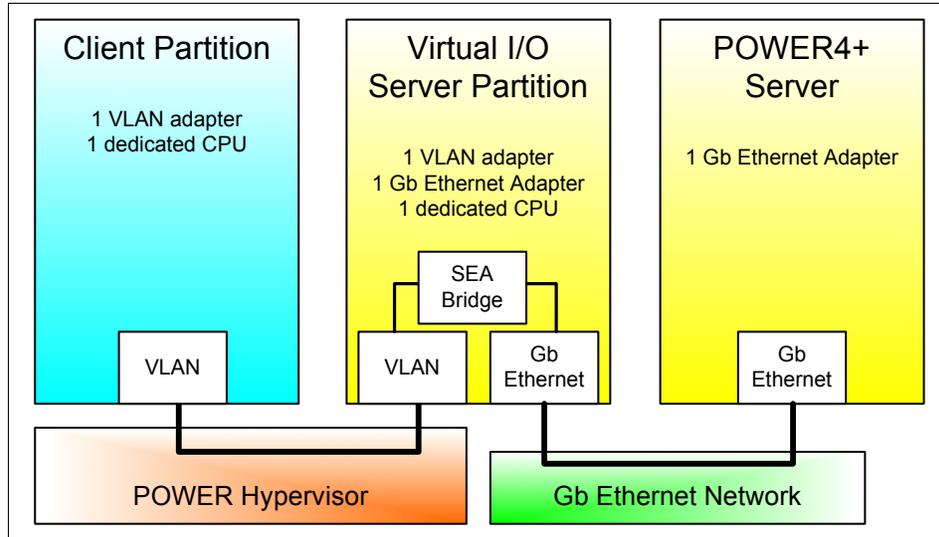


Figure 6-32 Configuration of test environment

Virtual I/O Server performance results

The next two figures show the results measured on the Virtual I/O Server. Figure 6-33 on page 192 shows the throughput of the Virtual I/O Server at MTU sizes of 1,500 and 9,000 in both modes, simplex and duplex. Note that this test maximized the line speed of the Gigabit Ethernet. Therefore, the limitation is the physical network media speed (1 Gb simplex or 2 Gb duplex).

Figure 6-34 on page 192 presents the utilization of the processor in the Virtual I/O Server partition. To provide a better comparison of processor utilization versus MTU size and simplex/duplex modes, the utilization is normalized to 1Gb data throughput.

The results show that the Shared Ethernet Adapter enables the adapters to stream data at media speed as long as it has enough processor entitlements.

Processor utilization per gigabit of throughput is higher with the Shared Ethernet Adapter because it has to use the POWER Hypervisor to move the packets of the VLAN between partitions, and because of the SEA's device driver code.

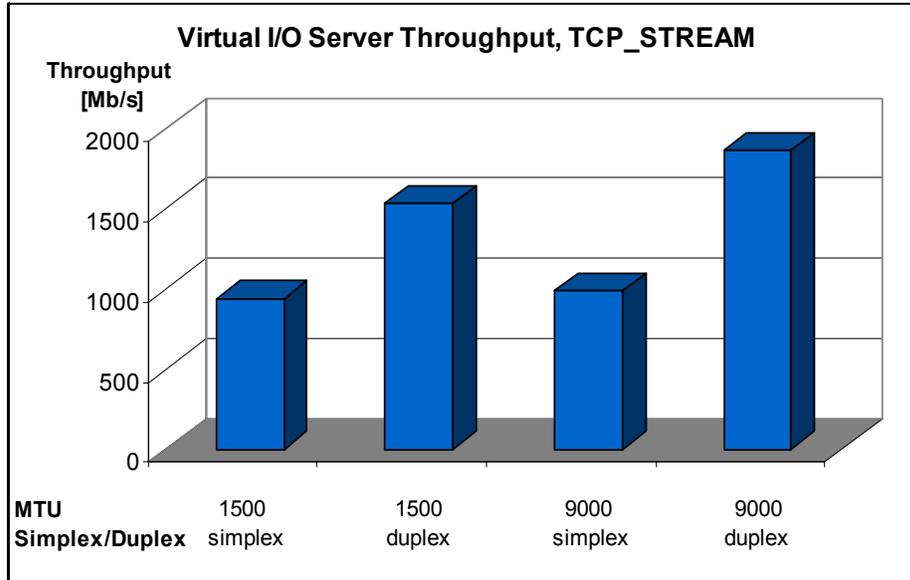


Figure 6-33 Throughput of the Virtual I/O Server

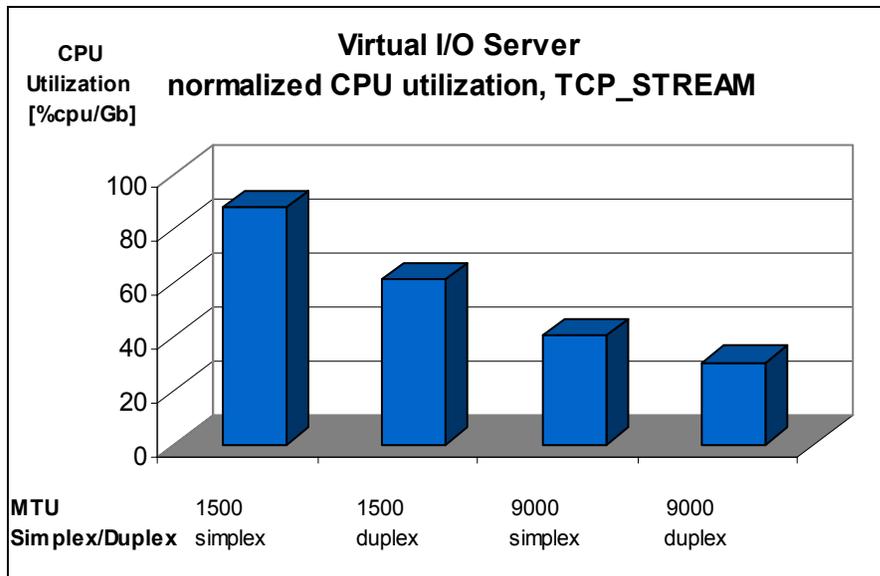


Figure 6-34 Processor utilization of the Virtual I/O Server

6.6.2 Request/response time and latency

In this test, the workload TCP_RR was used to determine the difference in transaction rate and the latency between the Shared Ethernet Adapter and a physical 1 Gb Ethernet adapter.

The measurements for the Shared Ethernet Adapter (SEA) were taken using the configuration that was shown in Figure 6-32 on page 191, with traffic exchanged between LPAR1 and the server.

Figure 6-35 shows the configuration of the physical Ethernet test. The Virtual I/O Server is bypassed and the traffic flows directly from the client partition to the POWER4+ server through the 1 Gbps physical network.

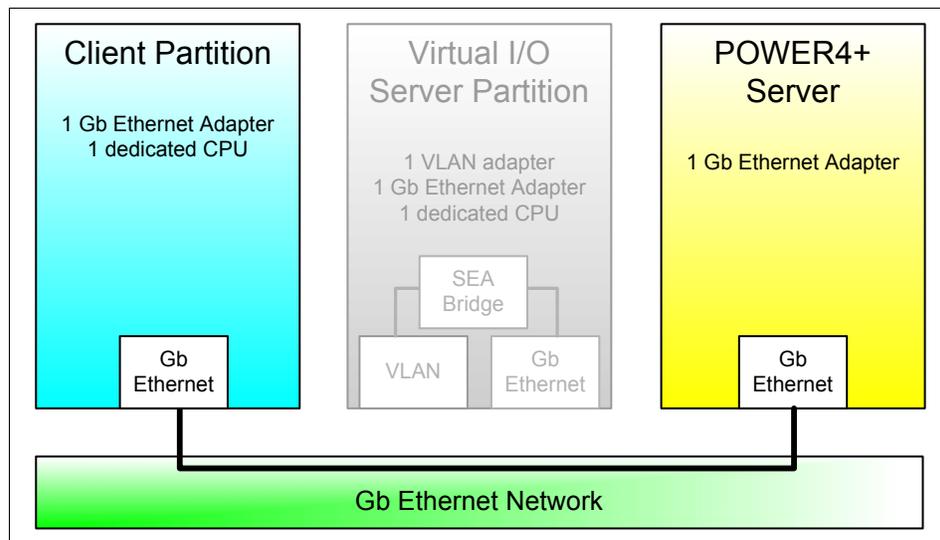


Figure 6-35 Dedicated connection between a partition and an external server

Results of request/response time and latency

The next two figures show the results of the TCP_RR benchmark for 1 and 20 sessions.

Note that the values shown for tests where there was just one session are limited by the default setting of the physical Ethernet adapter's interrupt coalescing value. The physical Ethernet adapter has interrupt coalescing enabled by default (`intr_rate=10000`) because this helps reduce CPU utilization at higher transaction rates. However, this adds latency when only a single transaction is running due to delaying the interrupt. Some workloads with small packets and light workload may benefit from disabling the interrupt coalescing on the physical adapter.

Tip: Some workloads with small packets and light workloads may benefit from disabling the interrupt coalescing on the physical adapter.

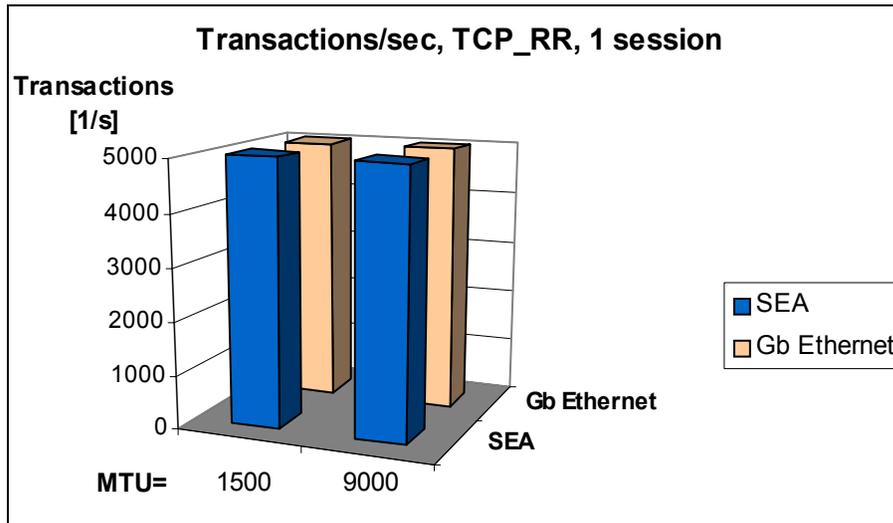


Figure 6-36 Transaction rates, TCP_RR, 1 session

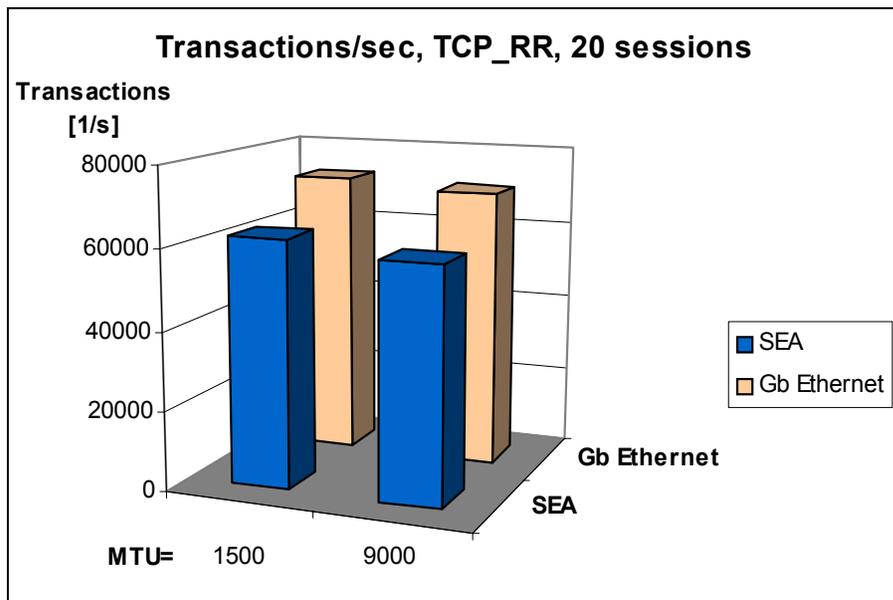


Figure 6-37 Transaction rates, TCP_RR, 20 sessions

Latency was measured with the same parameters as the transaction rate. Figure 6-38 and Figure 6-39 show the differences between the Shared Ethernet Adapter and physical Ethernet and the increasing latency if the load grows to 20 sessions.

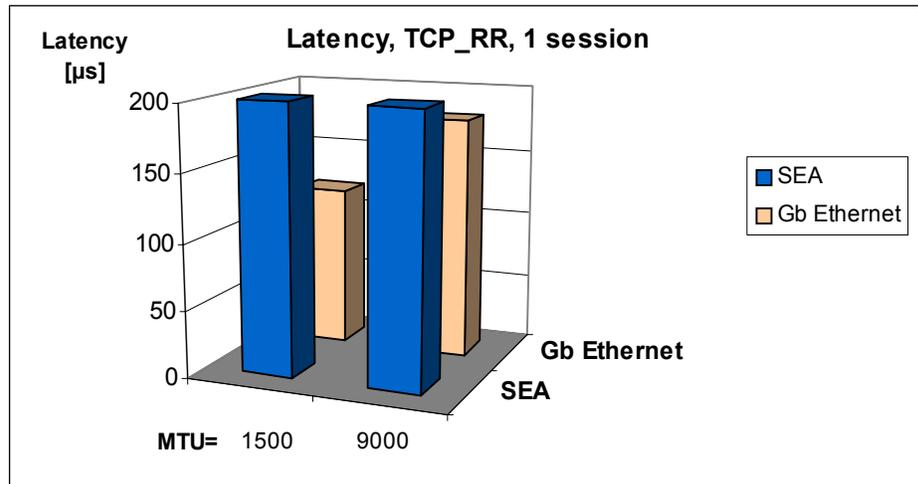


Figure 6-38 Latencies, TCP_RR, 1 session

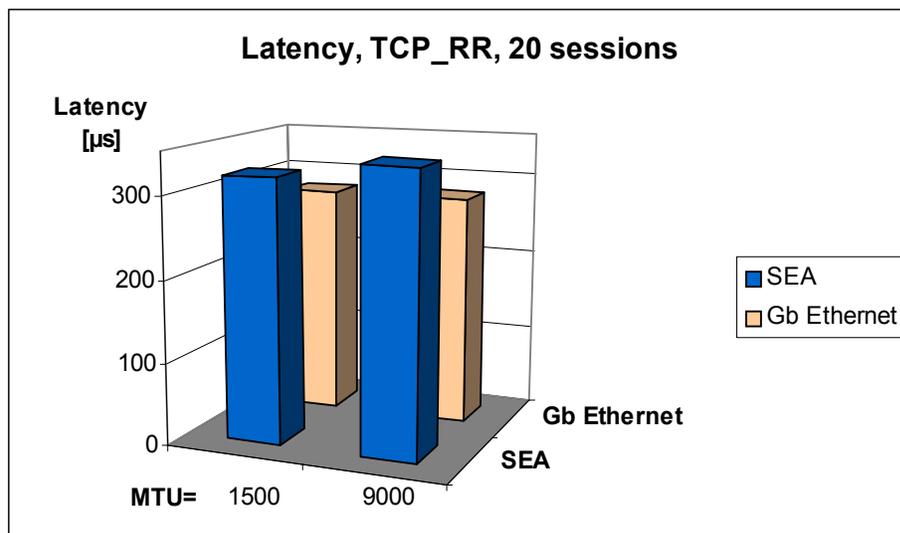


Figure 6-39 Latency, TCP_RR, 20 sessions

6.7 Implementation guidelines

Sizing a server can be somewhat complex and time-consuming. Furthermore, it can be performed with varying accuracy, depending on the amount of data you can collect about the resources requirements of your applications.

This section offers some guidelines for designing a Virtual I/O Server partition. The intent is to give some quick sizing guidelines that may be simple enough for initial sizing when very little data about the application requirements is available. Later on, the server could have its partition size increased or decreased to adjust for variations in the actual workload during peak times of the day. Because of the virtualization features of the hardware, the machine resources can be adjusted to meet the demands of the Virtual I/O server. See 6.7.1, “Guidelines for Shared Ethernet Adapter sizing” on page 197 for a more accurate method to adjust the Virtual I/O Server resources.

Important: The following recommendations provide a reasonable starting point for an initial configuration. Further tuning will be required to obtain optimal performance.

Guidelines for sizing and configuring the network

The following guidelines are given to assist you in properly sizing your network.

1. Know your environment and the network traffic.
2. For the most demanding network traffic between VLANs and local networks, use a dedicated network adapter.
3. For optimal performance, use dedicated processors for the Virtual I/O Server partition.
4. Choose 9000 for the MTU size or what makes sense for your network traffic.

Guidelines for optimizing network throughput

Table 6-6 lists guidelines for easy estimating of network throughput. The speed numbers are a bit conservative but rounded down for easy estimating. These numbers are for POWER5 processor-based systems with PCI-X slots.

Table 6-6 Network streaming rates

Adapter speed	Throughput (MB/second)	
	Simplex	Full Duplex
10 Mb Ethernet	1 MB/s	2 MB/s
100 Mb Ethernet	10 MB/s	20 MB/s

Adapter speed	Throughput (MB/second)	
	Simplex	Full Duplex
1000 Mb Ethernet (Gigabit Ethernet)	100 MB/s	150 MB/s (1.5X the simplex rate)

Guidelines for processor requirements

Because Ethernet running with an MTU size of 1500 bytes consumes more CPU cycles than Ethernet running with Jumbo frames (MTU 9000), the guidelines are different for each. In round numbers, the CPU utilization for large packet workloads on jumbo frames is about half of the CPU required for MTU 1500.

With configurations where MTU is 1500

A basic general rule is to provide 100% of one POWER5 processor (1.65 GHz) per Gigabit Ethernet adapter to drive it to maximum bandwidth. This would translate to ten 100-Mbit Ethernet adapters if attached to a 100 Mb LAN.

For example, if two Gigabit Ethernet adapters will be used, then up to two processors should be allocated to the partition.

With configurations where MTU is 9000 (jumbo frames)

The general rule is 50% of one POWER5 processor (1.65 GHz) per Gigabit Ethernet to drive to maximum bandwidth.

The processing power needed to transfer data over a network depends mainly on the number of packets to be handled. If your network traffic contains a lot of small transactions that do not take advantage of the jumbo frame payload but use small packets, then you should plan on one full CPU to drive each Gigabit Ethernet adapter. (Jumbo frames do not help the small packet workload case).

6.7.1 Guidelines for Shared Ethernet Adapter sizing

Sizing of the Virtual I/O Server for the Shared Ethernet Adapter component involves these steps:

1. Define the target bandwidth or transaction rate requirements.

The idea is to determine the target bandwidth on the physical Ethernet side of the Virtual I/O Server partition, as this will determine the rate that data can be transferred between the Virtual I/O Server partition and the client partitions. When the target rate is known, the proper type and number of network adapters can be selected. For example, various speed Ethernet adapters could be used, such as 10 Mb, 100 Mb, or Gigabit. One or more adapters could be used on individual networks or they could be aggregated using port aggregation.

2. Define the type of workload.

The type of workload can be streaming data for workloads such as file transfers or data backup, or small transaction workloads such as remote procedure calls (RPCs). The streaming workload is mainly dominated by large full-size network packets and associated small TCP Acknowledgement packets. Transaction workloads typically involve smaller packets or may involve small requests, such as a URL, and a larger response, such as a Web page. It is common for an I/O server to have to support streaming and small packet I/O during various periods of time. In such cases, the sizing should be approached from both models and the larger sizing used.

3. Identify the MTU size that will be used.

The standard Internet cell size is 1,500 bytes (1,518 bytes on the wire) and is the typical setting on adapter cards. Gigabit Ethernet can support MTU 9000-byte Jumbo frames and may be desirable for localized networks. The larger Jumbo frames can reduce the CPU cycles considerably for the streaming types of workloads. However for small workloads, the larger MTU size will not help reduce CPU cycles. In many cases, the MTU choice is driven by the existing network infrastructure and cannot be freely chosen according to the application requirements.

4. Define the Virtual I/O Server partition configuration.

This definition includes the number of processors and I/O adapters. Another issue that affects the CPU cycles used is whether the Shared Ethernet Adapter is configured to run in threaded or non-threaded mode. Threaded mode is used mainly when VSCSI will be configured on the same Virtual I/O Server partition. Threaded mode helps ensure that VSCSI and the Shared Ethernet Adapter share the CPU resource fairly. Threading adds more instruction path length, however, thus using more CPU cycles. If the Virtual I/O Server partition will be dedicated to running shared Ethernet and associated virtual Ethernet only, they should be configured with threading disabled in order to run in the most efficient mode. Enabling and disabling of threading is covered in 6.7.3, “Control of threading in the Shared Ethernet Adapter” on page 204.

Important: The threading concept discussed above is software threading. It is *not* the POWER5 hardware feature that enables running the virtual processors in single-threaded or simultaneous multithreading mode.

When the workload and type of adapters have been chosen, determine how many processors are required to move data through the network at the desired rate. The networking device drivers are CPU-intensive. Small packets can come in at a faster rate and use more CPU cycles than larger packet workloads. Larger packet workloads are normally limited by network wire bandwidth and come in at

a slower rate, thus requiring less CPU than small-packet workloads for the amount of data transferred.

CPU sizing

CPU sizing for the SEA is divided into two workload types, TCP streaming and TCP request/response (transaction), for both MTU 1500 and MTU 9000 networks. The sizing is provided in terms of number of *machine cycles needed per byte of throughput* or *machine cycles needed per transaction*.

The upcoming tables were derived with these formulas:

$$\text{cycles per byte} = (\# \text{ CPUs} * \text{CPU_Utilization} * \text{CPU clock frequency}) / \text{Throughput rate in bytes per second}$$

and

$$\text{cycles per transaction} = (\# \text{ CPUs} * \text{CPU_Utilization} * \text{CPU clock frequency}) / \text{Throughput rate in transactions per second}$$

When sizing, it is necessary to consider the impact of the threading option that is available for the device driver of the Shared Ethernet Adapter. The threading option increases processor utilization at lower workloads due to the threads being started for each packet. At higher workload rates, such as full duplex or the request/response workloads, the threads can run longer without waiting and being redispached. The thread option should be *disabled* if the Shared Ethernet Adapter is running in a partition by itself without VSCSI.

The numbers were measured on a single 1.65 GHz POWER5 processor, running with the default of simultaneous multithreading enabled. For other CPU frequencies, the numbers in these tables can be scaled by the ratio of the CPU frequencies for approximate values to be used for sizing.

For example, for a 1.5 GHz processor speed, use $1.65/1.5 * \text{cycles per byte}$ or transactions value from the table. This example would result in a value of 1.1 times the value in the table, thus requiring 10% more cycles to adjust for the 10% slower clock rate of the 1.5 GHz processor.

To use these values, multiply your required throughput rate (in bytes or transactions) by the number of cycles per byte or transactions in the tables that follow. This produces the required machine cycles for the workload for a 1.65 GHz speed. Then adjust this value by the ratio of the actual machine speed to this 1.65 GHz speed.

Then to find the number of CPUs, divide the result by 1,650,000,000 cycles. You would need that many CPUs to drive the workload. In these tables, one MB is 1,048,576 bytes.

For example, if the Virtual I/O Server must deliver 200 MB of streaming throughput, it would take $200 * 1,048,576 * 11.2 = 2,348,810,240 / 1,650,000,000$ cycles per CPU=1.42 CPUs. Thus, in round numbers, it would take 1.5 processors in the Virtual I/O Server partition to handle this workload.

This could be handled with either a two-CPU dedicated partition or a 1.5-CPU micro-partition.

Table 6-7 provides the figures to use for streaming workloads when the threading option is *enabled*.

Table 6-7 Streaming workload, machine cycles per byte - threading enabled

Streaming type	MTU 1500 rate and CPU utilization	MTU 1500, cycles per byte	MTU 9000 rate and CPU utilization	MTU 9000, cycles per byte
Simplex	112.8 MB at 80.6% CPU	11.2	117.8 MB at 37.7% CPU	5.0
Duplex	162.2 MB at 88.8% CPU	8.6	217.0 MB at 52.5% CPU	3.8

Table 6-8 provides the figures to use when the threading option is *disabled*.

Table 6-8 Streaming workload, machine cycles per byte - threading disabled

Streaming type	MTU 1500 rate and CPU utilization	MTU 1500, cycles per byte	MTU 9000 rate and CPU utilization	MTU 9000, cycles per byte
Simplex	112.8 MB at 66.4% CPU	9.3	117.8 MB at 26.7% CPU	3.6
Duplex	161.6 MB at 76.4% CPU	7.4	216.8 MB at 39.6% CPU	2.9

Table 6-9 on page 201 has figures to use for transaction workloads when the threading option is *enabled*. A transaction is a round-trip request and reply of size listed in the first column of the table. Table 6-10 on page 201 has figures for when threading is *disabled*.

Table 6-9 Transaction workload, transactions per second - threading enabled

Size of transactions	Transactions/second and I/O server utilization	MTU 1500 or 9000, cycles per transaction
Small packets (64 bytes)	59772 TPS at 83.4% CPU	23022
Large packets (1024 bytes)	51956 TPS at 80.0% CPU	25406

Table 6-10 Transaction workload, transactions per second - threading disabled

Size of transactions	Transactions/second and I/O server utilization	MTU 1500 or 9000, cycles per transaction
Small packets (64 bytes)	60249 TPS at 65.6% CPU	17965
Large packets (1024 bytes)	53104 TPS at 65.0% CPU	20196

Micro-Partitioning considerations

Creating the Virtual I/O Server partition with Micro-Partitioning can be used when interfacing to slower speed networks (for example, 10/100 Mb) since a full, dedicated processor is not needed. This probably should be done only if the workload is less than 50% CPU utilization or if the workload characteristics are burst-type transactions. Configuring the partition as *uncapped* can also enable it to use more processor cycles as needed to handle the bursts.

For example, if the network is used only at night when other processors may be idle, the partition may be able to use the unused machine cycles. It could be configured with minimal CPU to handle light traffic during the day, but the uncapped processor could use more machine cycles during idle periods.

When configuring Micro-Partitioning for the Virtual I/O Server partition, it is suggested that you increase the entitlement to accommodate the extra resources needed by the POWER Hypervisor.

Memory sizing

The memory requirements for a Virtual I/O Server partition that provides the Shared Ethernet Adapter functions only (no VSCSI) are minimal. In general, a 512 MB partition should work for most configurations.

Enough memory must be allocated for the I/O server data structures. For the Ethernet and virtual devices, there are dedicated receive buffers that each device will use. These buffers are used to store the incoming packets from the VLAN, before delivery to the physical Ethernet adapter, so they are very transient.

For physical Ethernet network, the system typically uses 4 MB for MTU 1500 or 16 MB for MTU 9000 for dedicated receive buffers. For virtual Ethernet, the system typically uses 6 MB for dedicated receive buffers; however, this number can vary based on load.

Each instance of a physical or virtual Ethernet would need memory for this many buffers.

In addition, the system has an *mbuf buffer pool* per CPU that is used if additional buffers are needed. These mbufs typically occupy 40 MB.

6.7.2 Guidelines for physical Ethernet sizing

This section provides information about bandwidth for various Ethernet adapters, CPU cycles required for the Virtual I/O Server to handle these packets, and the formulas used to compute the server sizings.

Table 6-11 has approximate throughput rates for the various Ethernet adapters and MTU sizes in *simplex* mode. Table 6-12 on page 203 provides approximate throughput rates for various Ethernet adapters and MTU sizes in *duplex* mode.

Table 6-11 TCP streaming rates, simplex mode

Network type	Raw bit rate (Mb/s)	Payload rate (Mb/s)	Payload rate (MB)
10 Mb Ethernet, Half Duplex	10	6	.7
10 Mb Ethernet, Full Duplex ^a	10 (20 Mbit full duplex)	9.48	1.13
100 Mb Ethernet, Half Duplex	100	62	7.3
100 Mb Ethernet, Full Duplex	100 (200 Mbit full duplex)	94.8	11.3
1000 Mb Ethernet, Full Duplex, MTU 1500	1000 (2000 Mbit full duplex)	948	113
1000 Mb Ethernet, Full Duplex, MTU 9000	1000 (2000 Mbit full duplex)	989	117.9

a. The peak numbers represent best case throughput with multiple TCP sessions running in duplex mode. Other rates are for single TCP sessions.

Table 6-12 TCP Streaming rates, duplex mode

Network type	Raw bit rate (Mb/s)	Payload rate (Mb/s)	Payload rate (MB)
10 Mbit Ethernet, half duplex	10	5.8	.7
10 Mbit Ethernet, full duplex	10 (20 Mbit full duplex)	18	2.2
100 Mbit Ethernet, half duplex	100	58	7
100 Mbit Ethernet, full duplex	100 (200 Mb full duplex)	177	21.1
1000 Mbit Ethernet, full duplex, MTU 1500 ^a	1000 (2000 Mbit full duplex)	1470 (1660 peak)	175 (198 peak)
1000 Mbit Ethernet, full duplex, MTU 9000 ^b	1000 (2000 Mbit full duplex)	1680 (1938 peak)	200 (231 peak)

a. 1000 Mbit Ethernet (Gigabit Ethernet) duplex rates are for the PCI-X adapter in PCI-X slots.

b. Data rates are for TCP/IP using IPV4 protocol. Adapters with MTU 9000 have RFC1323 enabled.

These tables provide the maximum network payload speeds. These are user payload data rates that can be obtained by sockets-based programs for applications that are streaming data (one program doing send() calls and the receiver doing recv() calls over a TCP connection). The rates are a function of the network bit rate, MTU size, physical level requirements such as Inter-frame gap and preamble bits, data link headers, and TCP/IP headers. These are best case numbers for a single LAN, and may be lower if going through routers or additional network hops or remote links.

Note that the *raw bit rate* is the physical media bit rate and does not reflect physical media data like Inter-frame gaps, preamble bits, cell information (for ATM), data link headers, and trailers. These all reduce the effective usable bit rate of the wire.

Uni-directional (simplex) TCP streaming rates are rates that can be seen by a workload such as File Transfer Protocol (FTP) operations sending data from machine A to machine B in a memory-to-memory test. Note that full duplex media performs slightly better than half duplex media because the TCP acknowledgement packets can flow back without contending for the same wire that the data packets are flowing on.

These are user payload data rates that can be obtained by sockets-based programs for applications that are streaming data (one program doing send() calls and the receiver doing recv() calls) over a TCP connection. The rates are a function of the network bit rate, MTU size, physical level requirements such as

Inter-frame gap and preamble bits, data link headers, and TCP/IP headers. These are best-case numbers for a single LAN, and may be lower if going through routers or additional network hops or remote links.

Bi-directional (duplex) TCP streaming workloads have streaming data in both directions (for example, an FTP from machine A to machine B and another FTP running from machine B to machine A, concurrently). Such workloads can take advantage of full duplex media that can send and receive concurrently. Some media (for example, Ethernet in half duplex mode), cannot send and receive concurrently, thus they will not perform any better (usually worse) when running duplex workloads. Duplex workloads do not provide twice the throughput as simplex workloads. This is because TCP acknowledge packets coming back from the receiver have to compete with data packets flowing in the same direction.

6.7.3 Control of threading in the Shared Ethernet Adapter

These steps are necessary to configure the threading mode for the Shared Ethernet Adapter:

1. Log on to the Virtual I/O Server partition as the user `padmin`.
2. Using the `lsdev` command, list the virtual adapters to find the Shared Ethernet Adapter, as shown in Example 6-2.

Example 6-2 Listing virtual devices with lsdev

```
$ lsdev -virtual
name          status      description
ent2          Available  Virtual I/O Ethernet Adapter (1-lan)
vsa0          Available  LPAR Virtual Serial Adapter
ent3          Available  Shared Ethernet Adapter
```

3. In this example, the Shared Ethernet Adapter is `ent3`. Use the `lsdev` command again to find the current settings of the adapter. Example 6-3 shows that the thread mode is currently disabled (0).

Example 6-3 Displaying attributes of the Shared Ethernet Adapter

```
$ lsdev -dev ent3 -attr
attribute      value description                                     user_settable
pvid           100   PVID to use for the SEA device                               True
pvid_adapter  ent2   Default virtual adapter to use for non-VLAN-tagged packets  True
real_adapter   ent0   Physical adapter associated with the SEA                     True
thread         0     Thread mode enabled (1) or disabled (0)                     True
virt_adapters ent2   List of virtual adapters associated with the SEA (comma      True
separated)
```

4. To enable threading, use the **chdev** command as shown in Example 6-4.

Example 6-4 Enabling threading with chdev

```
$ chdev -dev ent3 -attr thread=1
ent3 changed
```

5. You can confirm that the threading mode is enabled, shown in Example 6-5.

Example 6-5 Checking the new threading mode

```
$ lsdev -dev ent3 -attr
attribute      value description                                user_settable

pvid           100   PVID to use for the SEA device                True
pvid_adapter   ent2   Default virtual adapter to use for non-VLAN-tagged packets True
real_adapter   ent0   Physical adapter associated with the SEA        True
thread         1     Thread mode enabled (1) or disabled (0)        True
virt_adapters  ent2   List of virtual adapters associated with the SEA (comma separated) True
$
```

6.8 Virtual SCSI

Virtual SCSI is based on a client/server relationship. A Virtual I/O Server partition owns the physical resources, and logical client partitions access the virtual SCSI resources provided by the Virtual I/O Server partition. The Virtual I/O Server partition has physically attached I/O devices and exports one or more of these devices to other partitions. The client partition is a partition that has a virtual client adapter node defined in its device tree and relies on the Virtual I/O Server partition to provide access to one or more block interface devices. Virtual SCSI requires POWER5 hardware with the Advanced POWER Virtualization feature activated. It provides virtual SCSI support for AIX 5L V5.3 and Linux.

As we write this book, the virtualization features of the POWER5 platform support up to 254 partitions, but the server hardware only provides up to 160 I/O slots per machine. With each partition typically requiring one I/O slot for disk attachment and another one for network attachment, this puts a constraint on the number of partitions. To overcome these physical limitations, I/O resources must be shared. Virtual SCSI provides the means to do this for SCSI storage devices.

Furthermore, virtual I/O allows attachment of previously unsupported storage solutions. As long as the Virtual I/O Server partition supports the attachment of a storage resource, any client partition can access this storage by using virtual SCSI adapters.

For example, if there is no native support for EMC storage devices on Linux, running Linux in a logical partition of a POWER5 server makes this possible. A Linux client partition can access the EMC storage through a Virtual SCSI adapter. Requests from the virtual adapters are mapped to the physical resources in the Virtual I/O Server partition. Therefore, driver support for the physical resources is needed only in the Virtual I/O Server partition.

Note: You will see different terms in this publication that refer to the various components involved with virtual SCSI. Depending on the context, these terms may vary. With SCSI, usually the terms *initiator* and *target* are used, so you may see terms such as *virtual SCSI initiator* and *virtual SCSI target*. On the Hardware Management Console, the terms *virtual SCSI server adapter* and *virtual SCSI client adapter* are used. Basically they refer to the same thing. When describing the client/server relationship between the partitions involved in virtual SCSI, the terms *hosting partition* (meaning the Virtual I/O Server) and *hosted partition* (meaning the client partition) are used.

Virtual SCSI client and server architecture overview

Virtual SCSI is based on a client/server relationship. The Virtual I/O Server partition owns the physical resources and acts as *server* or, in SCSI terms, *target device*. The logical partitions access the virtual SCSI resources provided by the Virtual I/O Server partition as *clients*.

The virtual I/O adapters are configured using an HMC. The provisioning of virtual disk resources is provided by the Virtual I/O Server. The virtual SCSI adapter driver on the server partition is a dynamically loadable kernel extension and its entry points are contained in the device switch table. As a virtual SCSI *target device*, the primary function of the device driver is to convert SCSI Remote DMA Protocol (SRP) requests from the initiator driver (client side) into I/O requests that are forwarded to the device that is physically attached to the server. Data is then transferred directly to the client memory using LRDMA. LRDMA is covered in “Logical Remote Direct Memory Access (LRDMA)” on page 150, and SRP is covered in “SCSI Remote DMA Protocol” on page 214.

The virtual SCSI client adapter device driver (*vscsi_initdd*) is a dynamically loadable kernel extension and its entry points are contained in the device switch table. As a virtual SCSI *initiator*, the primary function of the initiator driver is to convert I/O requests from the peripheral or media device drivers to SRP Information Units (IUs), then forward the SRP IUs to the target device for LRDMA.

The virtual adapter on the client partition is in many ways similar to a physical SCSI adapter. While a typical SCSI adapter has a parallel bus or optical link

attached to it, the virtual adapter's link is the POWER Hypervisor's Reliable Command/Response Transport.

Physical disks owned by the Virtual I/O Server partition either can be exported and assigned to a client partition whole, or can be partitioned into several logical volumes. The logical volumes can then be assigned to different partitions. Therefore, Virtual SCSI enables sharing of adapters as well as disk devices.

For a physical or a logical volume to be available to a client partition, it is assigned to a virtual SCSI server adapter in the Virtual I/O Server. The basic command to map the Virtual SCSI with the logical volume or physical volume is:

```
mkvdev -vdev TargetDevice -vadapter VirtualSCSIServerAdapter
        [-dev DeviceName]
```

Run the **lsdev -virtual** command to make sure that your new virtual SCSI adapter is available, as shown in Example 6-6.

Example 6-6 Checking for virtual SCSI adapters

```
$ lsdev -virtual
name           status      description

ent2           Available  Virtual I/O Ethernet Adapter (1-lan)
vhost0        Available Virtual SCSI Server Adapter
vhost1         Available  Virtual SCSI Server Adapter
vsa0           Available  LPAR Virtual Serial Adapter
```

The next step is to create a virtual target device, which maps the Virtual SCSI server adapter vhost0 to the logical volume rootvg_dbsrv. When you do not use the -dev flag, the default name of the virtual target device adapter is vtscsix. Run the **mkvdev** command as shown in Example 6-7 to perform this task. If you want to map a physical volume to the virtual SCSI server adapter, use **hdiskx** instead of the logical volume devices for the -vdev flag.

Example 6-7 Using mkdev to create a virtual target device

```
$ mkvdev -vdev rootvg_dbsrv -vadapter vhost0 -dev vdbsrv
vdbsrv Available
```

The **lsdev** command (Example 6-8), shows the newly created virtual target device adapter.

Example 6-8 Using lsdev to show the virtual target device

```
$ lsdev -virtual
name           status      description
```

vhost0	Available	Virtual SCSI Server Adapter
vsa0	Available	LPAR Virtual Serial Adapter
vdbsrv	Available	Virtual Target Device - Logical Volume

The **lsmmap** command (Example 6-9), shows the logical connections between newly created devices.

Example 6-9 Using lsmmap to show logical connections

```
$ lsmmap -vadapter vhost0
SVSA          Physloc          Client PartitionID
-----
vhost0      U9111.520.10DDEEC-V1-C20    0x00000000

VTD          vdbsrv
LUN          0x8100000000000000
Backing device rootvg_dbsrv
Physloc
```

Here you also see the physical location being a combination of the slot number (in this case 20) and the logical partition ID. At this point the created virtual device can be attached from the client partition. You can now activate your partition into the SMS menu and install the AIX 5L V5.3 operating system on the virtual disk or add an additional virtual disk using the **cfgmgr** command. The Client PartitionID shows up as soon as the client partition is active.

The client partition accesses its assigned disks through a virtual SCSI client adapter, which sees standard SCSI devices and LUNs through this virtual adapter. Example 6-10 shows commands used to view the disks on an AIX 5L V5.3 client partition.

Example 6-10 Viewing virtual SCSI disks

```
# lsdev -Cc disk -s vscsi
hdisk2 Available Virtual SCSI Disk Drive
# lscfg -vpl hdisk2
hdisk2 111.520.10DDEEC-V3-C5-T1-L810000000000 Virtual SCSI Disk Drive
```

Figure 6-40 on page 209 shows an example in which one physical disk is partitioned into two logical volumes inside the Virtual I/O Server. Each of the two client partitions is assigned one logical volume, which it accesses through a virtual I/O adapter (virtual SCSI client adapter). Inside the partition, the disk is viewed as normal hdisk.

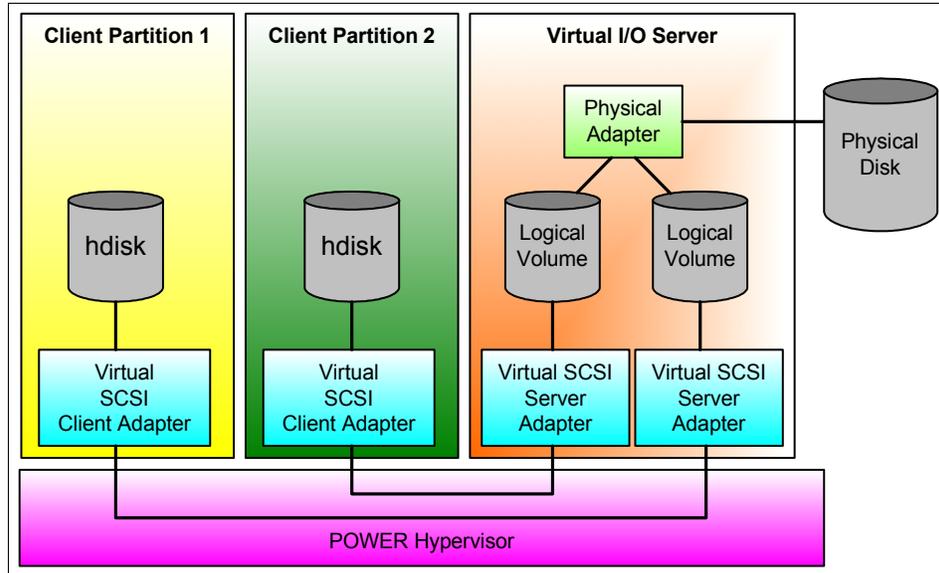


Figure 6-40 Virtual SCSI architecture overview

6.8.1 Client and server interaction

An example of a typical interaction between the target and initiator device drivers is a file read from a virtual disk device. The client stack considers the initiator driver a SCSI-3 device with access to the virtual disk.

A typical I/O read request involves the following steps:

1. (Client) The application program initiates a `read()` system call to the Journaled File System (JFS).
2. (Client) The file system requests a read from the Logical Volume Manager (LVM). The LVM forms a buffer structure (`struct buf`) with DMA buffer addressing information, as well as disk block information.
3. (Client) The buffer structure is passed to the disk device driver, which creates a `scsi_buf` and sends it to the `vscsi_initdd` driver.
4. (Client) The initiator driver takes the information in the `scsi_buf` and creates an SRP IU. If the I/O request includes data to be transferred, the initiator driver maps the client's data buffers for DMA.
5. (Client) The client builds a CRQ command element containing a pointer to the SRP IU and sends the CRQ command element (see "The Command/Response Queue" on page 149) through the POWER Hypervisor to the `vscsi_targetdd` device driver on the I/O server.

6. (Server) The target driver receives an interrupt indicating that an element has been queued on its command queue.
7. (Server) The target driver uses the pointer to the SRP IU in the CRQ command element and LRDMA services to copy the SRP IU from the client partition to the server partition's memory.
8. (Server) The target driver uses the information in the SRP IU to create a buffer structure.
9. (Server) The target driver passes the buffer structure to the LVM running in the server partition. The request ultimately makes its way to the physical adapter's device driver. This driver calls the usual DMA kernel services, which have been extended to map the client's buffers for DMA using LRDMA services.
10. (Server) When the transaction is complete, the target driver constructs an appropriate SRP response and uses LRDMA services to copy the response to the client's memory. It then builds a CRQ command element containing the tag (or correlator field) from the original SRP IU and sends the CRQ element through the POWER Hypervisor to the initiator.
11. (Client) The initiator driver receives an interrupt indicating that a CRQ element has been queued to its response queue.
12. (Client) The initiator driver uses the information in the SRP response to give status back to the `vscsi_initdd` driver. The driver passes the results back up to LVM and through to finish servicing the `read()` system call.

6.8.2 AIX 5L V5.3 device configuration for virtual SCSI

The virtual I/O adapters are connected to a virtual host bridge, which AIX 5L V5.3 treats much like a PCI host bridge. It is represented in the Object Data Manager as a bus device whose parent is `sysplanar0`. The virtual I/O adapters are represented as adapter devices with the virtual host bridge as their parent. On the Virtual I/O Server, each logical volume or physical volume that is exported to a client partition is represented by a virtual target device that is a child of a virtual SCSI server adapter.

On the client partition, the exported disks are visible as normal `hdisk`s, but they are defined in subclass `vscsi`. They have a virtual SCSI client adapter as parent. Example 6-41 on page 211 shows the relationship of the devices used by AIX 5L V5.3 for virtual SCSI and their physical counterparts.

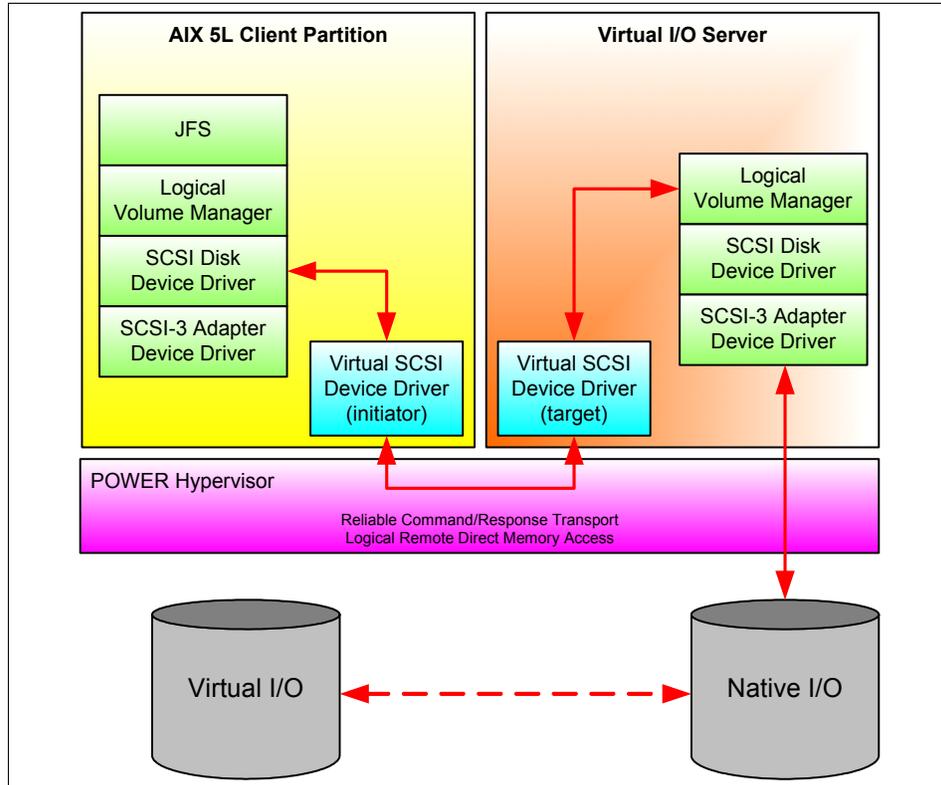


Figure 6-41 Virtual SCSI device relationship

The client and server adapters operate as a pair, in a point-to-point configuration, with the POWER Hypervisor providing the means of communication between the two. The SCSI requests and responses use the SCSI Remote DMA Protocol (SRP, detailed in “SCSI Remote DMA Protocol” on page 214). The client partition emulates a physical SCSI adapter that is presented to the client operating system for connecting to disks. The client driver accepts requests for storage services from the client operating system, converts those requests into *SRP information units (IUs)*, then uses copy RDMA to transmit those requests to the server driver. The server driver completes the requests using a combination of software emulation and services provided by the Virtual I/O Server operating system and its physical devices, then converts the results into SRP information units and returns the response back to the client driver.

In SCSI terminology, the client virtual SCSI adapter is called the *initiator* adapter, and the server virtual SCSI adapter is called the *target* adapter.

The target and initiator virtual adapters are created on the HMC during creation (or modification) of the partitions profiles. The target and initiator adapters are always connected in a point-to-point configuration. One initiator adapter can connect with at most one target adapter.

A target adapter can provide storage services to multiple initiators but not at the same time. When initiator virtual adapters of client partitions are created on the HMC, they may be assigned to a target adapter of a Virtual I/O Server partition that is already assigned to other partition profiles. When a client disconnects from an initiator driver, the adapter is put into the *defined* state, as opposed to the *available* state. At this point, another client can begin communication with the Virtual I/O Server.

The virtual SCSI architecture enables a partition to have instances of both client and server drivers. This could be the case when a server partition exports the disks it directly manages through its physical adapters, and boots from disks exported from another server partition. *We strongly recommend not using such configurations.* This could lead to deadlocks where two Virtual I/O Server partitions depend on each being activated before the operating system can boot. *Cascaded* devices (virtual devices that are backed by other virtual devices) are not supported.

6.8.3 Interpartition communication

The interpartition communication that took place in 6.8.1, “Client and server interaction” on page 209, involves the client device node in the Open Firmware device tree of one partition, the server device node in the Open Firmware device tree of another partition, the interpartition communication channel provided by the POWER Hypervisor and a communication protocol definition. The interpartition communication uses two primitive functions:

- ▶ Reliable Command/Response Transport
- ▶ SCSI Remote DMA Protocol

Reliable Command Response Transport

The Reliable Command Response Transport facility provides ordered delivery of messages between authorized partitions. In order to communicate, a client/server partition pair must establish a Command/Response Queue (CRQ). (See “The Command/Response Queue” on page 149.)

A CRQ is established during configuration by a virtual SCSI driver, given the presence in the Open Firmware device tree of a virtual SCSI device. The initiator driver registers a response queue and the target driver registers a command queue. Both use the `h_reg_crq` kernel service to call the POWER Hypervisor.

The POWER Hypervisor creates a connection between the two partitions through the queues.

When the queues are established, the virtual SCSI drivers can use the `h_send_crq` kernel service to put queue elements on each other's queues. The initiator driver attempts to queue an element to the target driver's command queue to initiate a transaction. If it is successful, the initiator driver returns, waiting for the interrupt indicating that a response has been posted by the target driver to the initiator driver's response queue.

The client partition uses only the Reliable Command/Response Transport. It does not use LRDMA. As the server partition's RTCE tables are not authorized for access by the client partition, any attempt by the client partition to modify server partition memory would be prevented by the POWER Hypervisor. RTCE table access is granted on a connection-by-connection basis (client/server virtual device pair).

The target driver is notified via an interrupt that it has received a message on its command queue. The target driver decodes the I/O request and routes it through the server partition's file subsystem for processing. When the request completes, the file subsystem calls the target driver and it packages a response into a queue element that is then queued to the initiator driver's response queue.

LRDMA defines an extended type of TCE table, the Remote DMA TCE Table. (See "Remote Translation Control Entry (RTCE)" on page 150.) An RTCE is used by the POWER Hypervisor to translate a server partition's Logical Remote DMA addresses. RTCE tables have extra data to help manage the use of its mappings by server partitions. Note that only the target driver uses the Logical Remote DMA primitives, not the initiator driver. The server partition's RTCE tables are not authorized for access by the client driver.

The use of redirected RDMA is completely invisible to the I/O client and has no impact on the virtual SCSI architecture defined in this document. It is left entirely to the discretion of the I/O server whether it first moves data from a physical device into its own memory before moving the data to the I/O client (using DMA), or whether the I/O server sets up the I/O request to the physical device in such a way that the physical device DMAs directly to the memory of the I/O client. The I/O server uses the RDMA mode that best suits its needs for a given virtual I/O operation.

The logical remote direct memory service enables the server driver to read and write to a well-defined part of the I/O client's memory. This service is unidirectional; that is, the client driver cannot use the service to write to, or read from, the I/O server's memory.

SCSI Remote DMA Protocol

The SCSI family of standards provides many different transport protocols that define the rules for exchanging information between SCSI initiators and targets. Virtual SCSI uses the SCSI Remote DMA Protocol (SRP), which defines the rules for exchanging SCSI information in an environment where the SCSI initiators and targets have the ability to directly transfer information between their respective address spaces.

SCSI requests and responses are sent using the virtual SCSI adapters that communicate through the POWER Hypervisor. However, the actual data transfer is done directly between a data buffer in the client partition and the physical adapter in the Virtual I/O Server by using the LRDMA protocol that was described in “Logical Remote Direct Memory Access (LRDMA)” on page 150.

SCSI Remote DMA Protocol defines a method of encapsulating SCSI command data blocks and is the protocol used for interpartition communication for virtual SCSI on IBM @server p5 logical partitions. Because virtual SCSI involves heterogeneous operating systems (AIX 5L and Linux) it is important to implement a common industry standard protocol for communicating I/O operations between partitions. SRP has defined the message format and protocol using an RDMA communication service. The SCSI RDMA Protocol defines the rules for exchanging SCSI information in an environment where SCSI initiators and targets have the ability to directly transfer information between their respective address spaces.

All SRP communication is accomplished via SRP Informational Units (IUs). An IU is an organized collection of data specified by the SRP to be transferred as login data, reject data, or a message on an RDMA channel. Thus all SCSI commands and their associated data and status are encapsulated in an SRP IU. Note that the protocol used for interpartition communication has no bearing on the makeup of the destination device. The SRP protocol works the same whether the target device is a physical device or a logical device (logical volume).

Memory descriptor mapping

The SRP architecture defines a *memory descriptor*, which is a 16-byte structure that identifies a memory segment on which DMA operations can be performed.

The virtual SCSI architecture is defined such that DMA operations are never initiated from the I/O client (from the initiator port). Because the I/O server's RTCE tables are not authorized for access by the I/O client, any attempt by the I/O client to modify the I/O server's memory would be prevented by the POWER Hypervisor. RTCE table access is granted on a connection-by-connection basis (client/server virtual device pair). If an I/O client happens to be serving some other logical device, then the partition is entitled to use Logical Remote DMA for the virtual devices that it is serving.

Memory descriptors sent in IUs that are defined in this architecture always reference memory in the initiator and are always used in DMA operations initiated by the target.

SRP initiator ports and SRP target ports shall be determined by both their role during LRDMA channel establishment and by the adapter types on which the messages are sent and received.

6.8.4 Disk considerations

A virtual disk device is exported by the I/O server to the client. It can be mapped by the server to either a logical volume, or defined on a slice of a physical volume or an entire physical disk.

It is viewed by the I/O client as a physical disk. There can be many virtual disk devices mapped onto a single physical disk. The system administrator creates a virtual disk device by choosing a logical volume and binding it to a virtual SCSI server adapter. The command adding virtual devices creates an ODM entry for the virtual disk device.

It is expected that most of the SCSI commands targeting a virtual disk device will be either reads or writes. Reads and writes are serviced by the LVM.

Figure 6-42 on page 216 shows the possible partitioning of a physical disk on the Virtual I/O Server where there are two logical volumes that support two virtual disk devices, hdx and hdy. In this example, hdx and hdy could be exported to two different partitions.

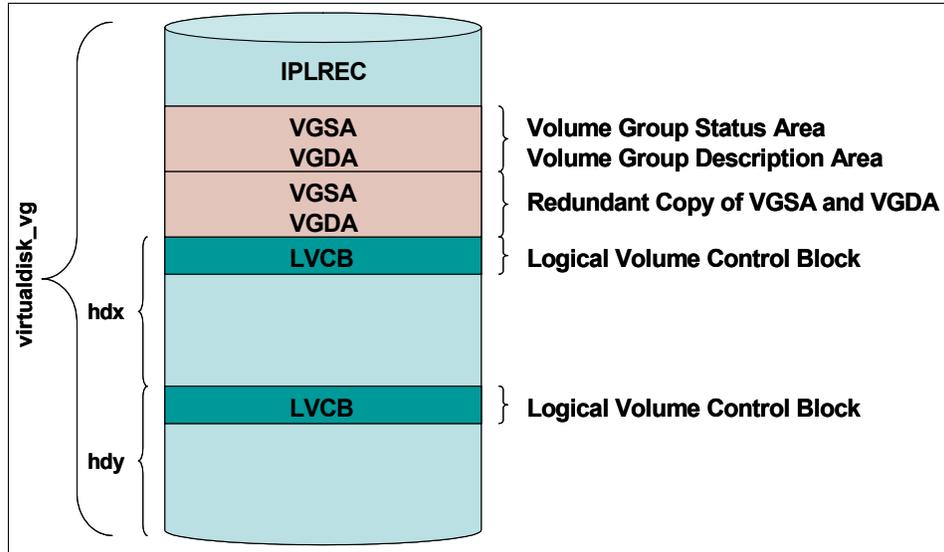


Figure 6-42 Volume group on Virtual I/O Server

SCSI RESERVE and RELEASE

The virtual SCSI virtual adapter driver emulates the SCSI RESERVE and RELEASE commands instead of passing them on to the device. That emulation is limited in scope to a single I/O server. When one I/O client wins a reservation on a logical volume, the virtual SCSI virtual adapter target driver has to refuse access by other I/O clients to the logical volume. When the I/O client holding a reservation fails, the virtual SCSI virtual adapter target driver has to break the reservation on that logical volume. This enables configurations where one I/O server provides storage services for multiple I/O clients.

However, this does not provide an adequate emulation of RESERVE and RELEASE for configurations in which the same physical storage can be accessed by multiple AIX 5L instances executing in different physical servers. This emulation does not prevent access by the native stack on that I/O server.

Command tag queuing

SCSI command tag queuing refers to queuing commands to a SCSI device. Command tag queuing requires the SCSI adapter, the SCSI device, the SCSI device driver, and the SCSI adapter driver to support this capability. The virtual SCSI architecture supports command tag queuing.

6.8.5 Configuring for redundancy

To minimize the adverse effect that would result from the loss of a Virtual I/O Server partition or physical adapter, a system administrator can use either of two ways to create redundant configurations. Each of these techniques enables a client partition to continue to function while maintenance is being done on the server partition.

Logical volume mirroring

AIX 5L Logical Volume Manager supports mirroring of virtual disks. This mirroring is configured on the client partition. For every write to a logical volume, the LVM generates a write request for every mirrored copy. The system administrator can define two virtual disk devices, either served by two distinct I/O servers or two devices on the same Virtual I/O Server, and mirror the client partition's data on the two devices. Mirroring makes no requirements on either the client or server drivers. It is cost-effective and the system configuration is readily understood. Figure 6-43 presents a configuration of mirrored virtual disks backed by physical disks.

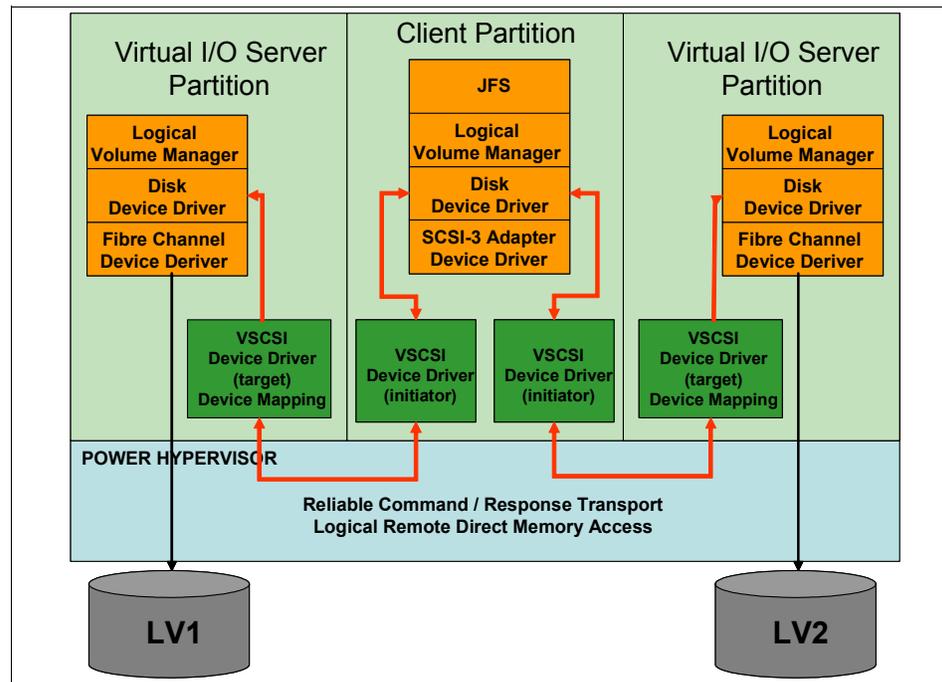


Figure 6-43 Using LVM mirroring for virtual SCSI

Note: The IBM Virtual I/O Server does not support mirroring. Each disk (either native or logical volume) exported from the Virtual I/O Server maps to only one physical disk. The disk mirroring must be defined in the client I/O partition. In the same way, the Virtual I/O Server does not support data striping over several disks. If striping is required, it must be defined in the client partition.

If mirroring is needed, set the scheduling policy to `parallel` and allocation policy to `strict`. The parallel scheduling policy enables reading from the disk that has the fewest outstanding requests, and strict allocation policy allocates each copy on separate physical volumes.

Physical mirroring

Physical mirroring is provided by the physical adapter rather than the operating system. If the physical adapter used in the I/O server to connect to the disk provides RAID support, it can be used along with virtual SCSI to provide a more reliable storage solution.

Multi-path I/O

Multi-path I/O (MPIO) offers another possible solution to the redundancy requirement. MPIO is a feature of AIX 5L V5.3 that permits a volume accessed by multiple physical paths to be seen as a single hdisk. It is therefore logically similar to IBM Subsystem Device Driver, which enables a volume on the TotalStorage® Enterprise Storage Subsystem that is accessed through multiple paths to be seen as a single path disk. However, the Subsystem Device Driver logical construct of a virtual path disk is above the level of the hdisk, whereas MPIO combines the paths underneath the level of the hdisk. MPIO is intended to support additional disk subsystems besides ESS. These disk subsystems are themselves capable of supporting multiple physical (parallel or Fibre Channel SCSI) attachments.

MPIO has numerous possible configuration parameters, but a detailed discussion of them is beyond the scope of this book. However, to gain the benefits of high availability and throughput that MPIO offers, it is recommended that it be configured with a round-robin algorithm, with health check enabled, and a reserve policy of `no_reserve`. This makes the best combination of throughput and reliability, because all paths are used for data transfer, and failed paths are detected and reacted to in a timely fashion.

Figure 6-44 shows a configuration using Multipath I/O to access an ESS disk.

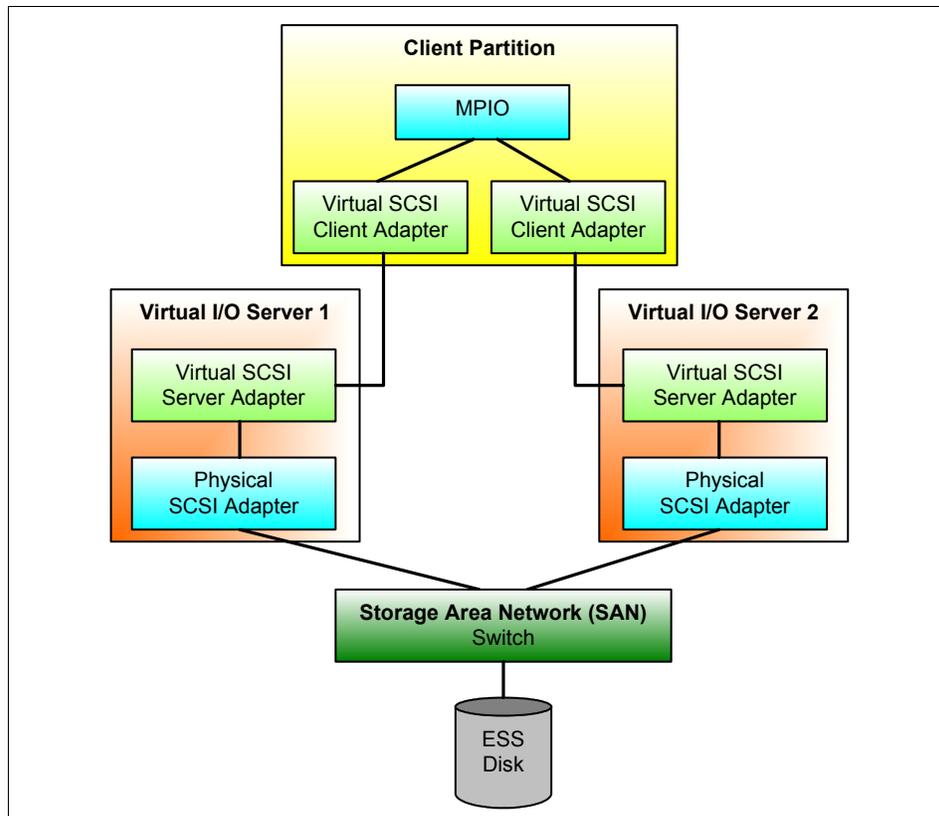


Figure 6-44 MPIO example configuration

Note: This type of configuration works only when the physical disk is assigned as a whole to the client partition. You cannot split the physical disk into logical volumes at the Virtual I/O Server level.

The client partition sees two paths to the physical disk through MPIO. Each path uses a different virtual SCSI adapter to access the disk. Each of these virtual SCSI adapters is backed by a separate Virtual I/O Server.

6.8.6 Performance considerations

The primary goal of virtualization is to lower the total cost of ownership of equipment by improving utilization of the overall system resources and reducing the labor requirements to operate and manage many servers.

With virtualization, the IBM @server p5 servers can now be used in similar to the way mainframes have been used for decades, sharing the hardware between many programs, services, applications, or users. Of course, for each of these individual users of the hardware, sharing resources may result in lower performance than having dedicated hardware, but the overall cost is usually far lower than when dedicating hardware to each user. The decision of using virtualization is therefore a trade-off between cost and performance.

The performance considerations that we detail in this section must be balanced against the savings made on the overall system cost. For example, the smallest physical disk that is available to the IBM @server p5 systems is 36 GB. A typical operating system requires 4 GB of disk. If one disk is dedicated to the operating system (for example, rootvg in AIX 5L), nearly 90% of this physical disk space is unused. Furthermore, the system disk I/O rate is often very low. With the help of virtual SCSI, it is possible to split the same disk into nine virtual disks of 4 GB each. If each of these disks is used for installation of the AIX 5L root volume group, you can support nine separate instances of the AIX 5L operating system, with nine times fewer disks and perhaps as many physical SCSI adapters. Compare these savings with the extra cost of processing power needed to handle the virtual disks.

Enabling virtual SCSI results in using extra processing power compared to directly attached disks, due to extra POWER Hypervisor activity. Depending on the configuration, this may or may not yield the same performance when comparing virtual SCSI devices to physically attached SCSI devices. If a partition has high performance and disk I/O requirements that justify the cost of dedicated hardware, then using virtual SCSI is not recommended. However, partitions with non-critical performance and low disk I/O requirements often can be configured to use virtual SCSI, which in turn lowers hardware and operating costs.

Using a logical volume for virtual storage means that the number of partitions is no longer limited by hardware. However, the trade-off is that some of the partitions may experience slightly less than optimal storage performance.

In the test results that follow, we see that the overhead of virtual SCSI in both SCSI and a FAStT implementation is small, and clients should assess the benefits of the virtual SCSI implementation for their environment. Simultaneous multithreading should be enabled in a virtual SCSI environment. With low I/O loads and a small number of partitions, micro-partitioning of the Virtual I/O Server partition has little effect on performance. For more efficient virtual SCSI

implementation with larger loads, it may be advantageous to keep the Virtual I/O Server partition as a dedicated processor.

Virtual storage can still be manipulated using the Logical Volume Manager the same as an ordinary physical disk. Some performance considerations from dedicated storage are still applicable when using virtual storage, such as spreading hot logical volumes across multiple volumes on multiple virtual SCSI so that parallel access is possible, the intra-disk policy (from the server's point of view, a virtual drive can be served using an entire drive, or a logical volume of a drive). If the entire drive is served to the client, then the rules and procedures apply on the client side as if the drive were local. If the server partition provides the client with a partition of a drive and a logical volume, then the server decides the area of the drive to serve to the client when the logical volume is created and sets the inter-policy to maximum. This spreads each logical volume across as many virtual storage devices as possible, allowing reads and writes to be shared among several physical volumes.

Consider the following general performance issues when using virtual SCSI:

- ▶ If not constrained by processor performance, virtual disk I/O throughput is comparable to local attached I/O.
- ▶ Virtual SCSI is a client/server model, so the combined CPU cycles required on the I/O client and the I/O server will always be higher than local I/O.
- ▶ If multiple partitions are competing for resources from a virtual SCSI server, care must be taken to ensure that enough server resources (processor, memory, and disk) are allocated to do the job.
- ▶ There is no data caching in memory on the Virtual I/O Server partition. Thus, all I/Os that it services are essentially synchronous disk I/Os. As there is no caching in memory on the server partition, its memory requirements should be modest.

In general, applications are functionally isolated from the exact nature of their storage subsystems by the operating system. An application does not have to be aware of whether its storage is contained on one type of disk or another when performing I/O. But different I/O subsystems have subtly different performance qualities, and virtual SCSI is no exception. What differences might an application observe using virtual SCSI versus directly attached storage? Broadly, we can categorize the possibilities into I/O latency and I/O bandwidth.

We define *I/O latency* as the time that passes between the initiation of I/O and completion as observed by the application. Latency is a very important attribute of disk I/O. Consider a program that performs 1000 random disk I/Os, one at a time. If the time to complete an average I/O is six milliseconds, the application will run no less than 6 seconds. However, if the average I/O response time is reduced to three milliseconds, the application's run time could be reduced by

three seconds. Applications that are multi-threaded or use asynchronous I/O may be less sensitive to latency, but under most circumstances, less latency is better for performance.

We define *I/O bandwidth* as the maximum data that can be read or written to storage in a unit of time. Bandwidth can be measured from a single thread or from a set of threads executing concurrently. Though many commercial codes are more sensitive to latency than bandwidth, bandwidth is crucial for many typical operations such as backup and restore of persistent data.

Because disks are mechanical devices, they tend to be rather slow when compared to high-performance microprocessors such as POWER5. As such, we will show that virtual SCSI performance is comparable to directly attached storage under most workload environments.

Virtual SCSI latency

Because virtual SCSI is implemented as a client/server model, naturally there is some extra latency that does not exist with direct attached storage. We define this extra latency as the additional amount of time necessary to complete an I/O operation when compared to the same operation on a locally attached device. Figure 6-45 on page 223 shows that this additional time varies from 0.03 to 0.06 ms per I/O operation depending primarily on the block size of the request using a dedicated Virtual I/O Server partition. It is comparable for both the physical disk and logical volume backed virtual drives. The latency experienced when using a Virtual I/O Server partition in a micro-partition may be higher and certainly more variable than using a dedicated I/O server partition.

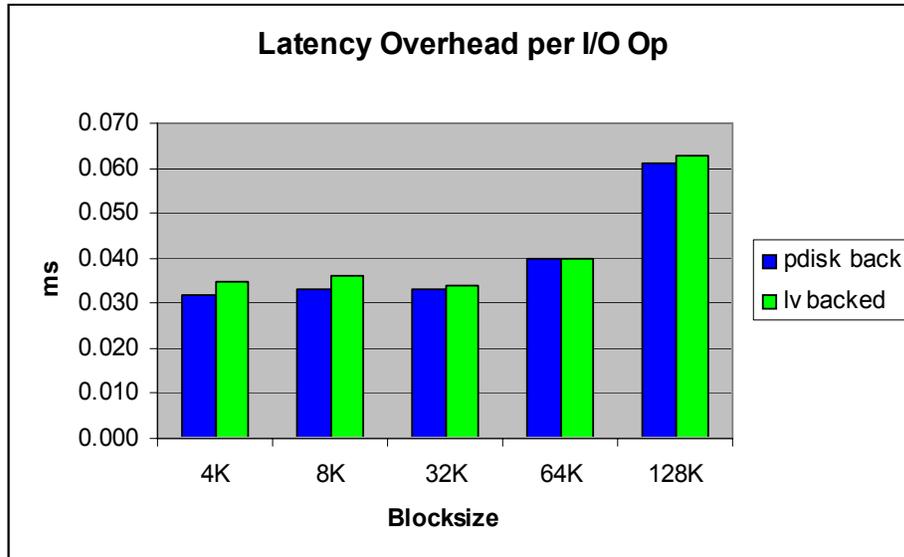


Figure 6-45 Latency for dedicated disk and logical volumes

For comparison purposes, Figure 6-46 on page 224 shows the average response times for locally attached I/O using one FASTT700 RAID0 LUN with five physical drives, caching enabled without write-cache mirroring. These measurements conduct sequential I/O, enabling the reads to be satisfied from the disk read cache and the writes to be cached in the FASTT700 controller. Because of caching, the physical I/Os in the test have much lower latency than in typical commercial environments, where random reads are not satisfied from cache so often. Nonetheless, the additional virtual SCSI latency for these low-latency caches is small compared to the actual disk latency. For I/Os with reads that are not cached by the controller, the virtual SCSI latency is small enough to be inconsequential.

Also observed the average disk response time increases with the block size. The latency increases in performing a virtual SCSI operation are relatively greater on smaller block sizes because of their shorter response time.

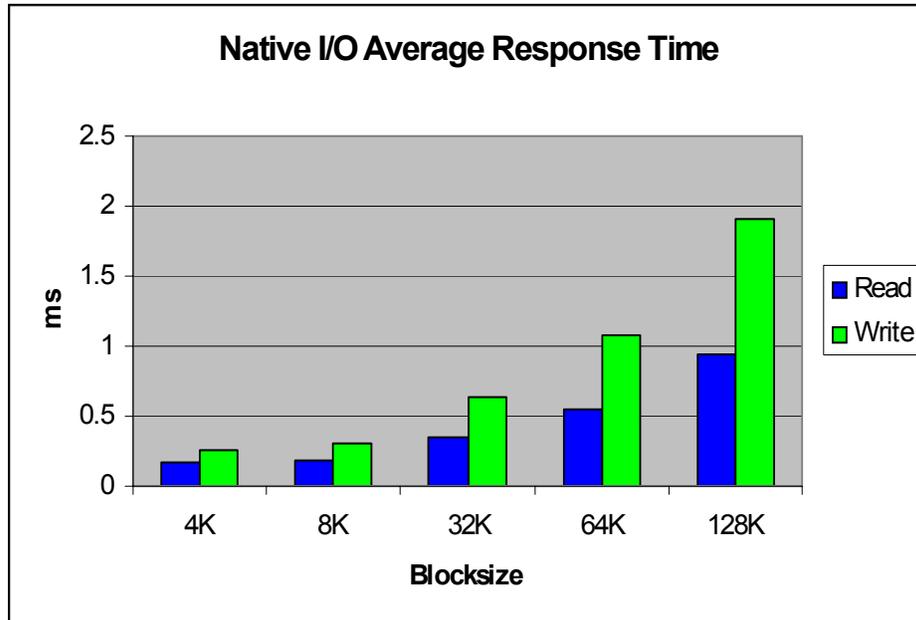


Figure 6-46 Response times for dedicated I/O

Virtual SCSI bandwidth

Figure 6-47 on page 225 compares virtual SCSI to native I/O performance on bandwidth tests. In these tests, a single thread operates sequentially on a constant file that is 256 MB in size, again with a dedicated Virtual I/O Server partition. More I/O operations are issued when reading or writing to the file using a small block size than with a larger block size. This figure shows a comparison of measured bandwidth using virtual SCSI and local attachment for reads with varying block sizes of operations. The difference between virtual I/O and native I/O in these tests is attributable to the increased latency using virtual I/O. Because of the larger number of operations, the bandwidth measured with small block sizes is much lower than with large block sizes.

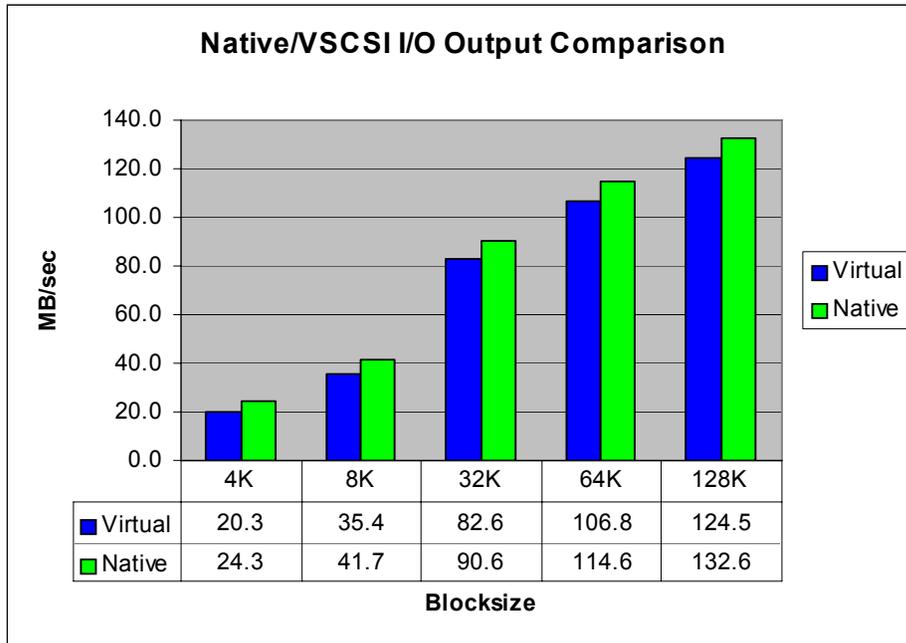


Figure 6-47 Native to virtual SCSI I/O comparison

Figure 6-48 on page 226 shows that virtual SCSI performance using a dedicated I/O server partition scales comparably to that of a similar native I/O-attached configuration to very high bandwidths. The experiment uses one FASTt disk and arrays of seven FASTt disks. Each array is attached to one Fiber Channel adapter. All I/Os use a blocksize of 128 KB. The difference in bandwidth between reads and writes is due to the cache in the FASTt controller. The experiment shows that the difference in bandwidth using virtual SCSI or native disks is not significant.

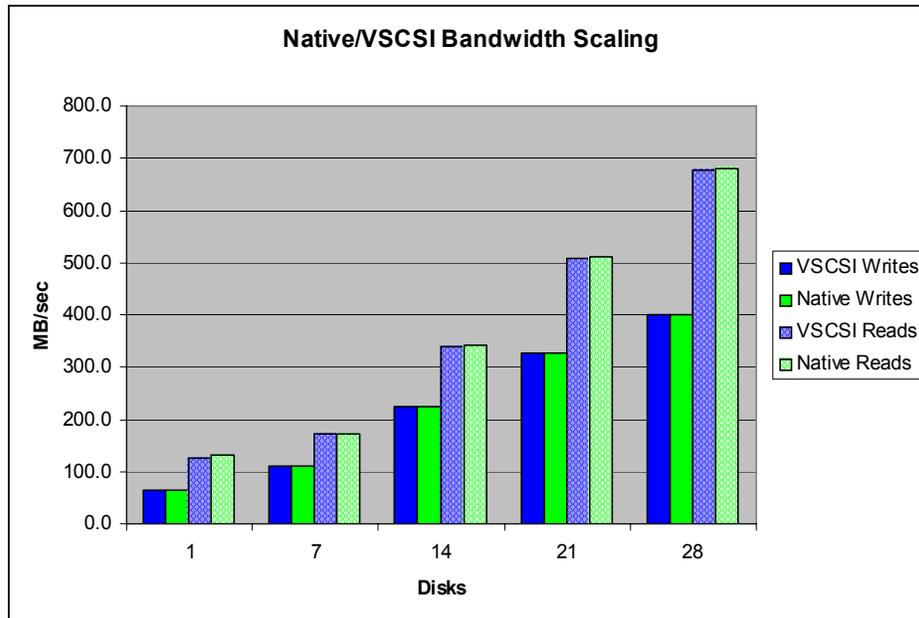


Figure 6-48 Native virtual SCSI bandwidth scaling

6.8.7 Sizing a virtual SCSI server

There are considerations to address when designing and implementing a virtual SCSI environment. The primary considerations are:

- ▶ Memory requirements
- ▶ Micro-Partitioning or dedicated processor partitions

One thing that does not have to be factored into sizing is the processor impact of using virtual I/O on the client. The processor cycles executed on the client to perform a virtual SCSI I/O are comparable to that of a locally attached I/O. Thus, there is no increase or decrease in sizing on the client partition for a known task. These sizing techniques do not address a Virtual I/O Server that uses both virtual SCSI and virtual Ethernet. If the two are combined, additional resources must be anticipated to support virtual Ethernet activity.

Memory requirements

The architecture of virtual SCSI simplifies memory sizing in that there is no caching of file data in the memory of the virtual SCSI server, so the memory requirements for the virtual SCSI server are fairly modest. With large I/O configurations and very high data rates, a 1 GB memory allocation for the virtual

SCSI server is more than sufficient. For low I/O rate cases with a small number of attached disks, 512 MB is a sufficient memory allocation.

Dedicated processor partitions

The amount of processor entitlement required for a virtual SCSI server is based on the maximum I/O rates required of it. Most virtual SCSI servers will not run at maximum I/O rates all the time, so the use of surplus processor time is potentially wasted by using dedicated processor partitions. In this section, we propose two sizing methodologies. For the first, you need a fair understanding of the I/O rates and I/O sizes required of the virtual SCSI server. In the second, we size the virtual SCSI server based more on the I/O configuration.

Sizing against expected I/O traffic

Our sizing methodology is based on the observation that processor time required to perform an I/O on the virtual SCSI server is fairly constant for a given I/O size. It is true that different devices (for example, SCSI and FASTT) have subtly varying efficiencies. But under most circumstances, the I/O devices supported by the virtual SCSI server are sufficiently similar to provide good recommendations. Table 6-13 shows the recommendations for both physical disk and LVM operations on a 1.65 GHz POWER5 processor with simultaneous multithreading enabled. These numbers are measured at the physical processor. For other I/O server CPU frequencies, you can adjust the cycles in Table 6-13 by multiplying the cycles per operation by the ratio of the frequencies. For example, to adjust for a 1.5 GHz CPU, $1.65 \text{ GHz} / 1.5 \text{ GHz} = 1.1$, so multiply the CPU cycles in the table by 1.1 to get the required cycles per operation.

Table 6-13 I/O CPU cycles required for various block sizes

	4K	8K	32K	64K	128K
Physical Disk	45,000	47,000	58,000	81,000	120,000
LVM	49,000	51,000	59,000	74,000	105,000

Figure 6-49 on page 228 shows a comparison of native I/O and virtual SCSI cycles per byte (CPB) using both logical volume–backed storage and physical disk–backed storage. The virtual SCSI measures are of only the Virtual I/O Server partition and do not include the client in the comparison. The processor efficiency of I/O improves with larger I/O size. Effectively, there is a fixed latency to start and complete an I/O, with some additional cycle time based on the size of the I/O.

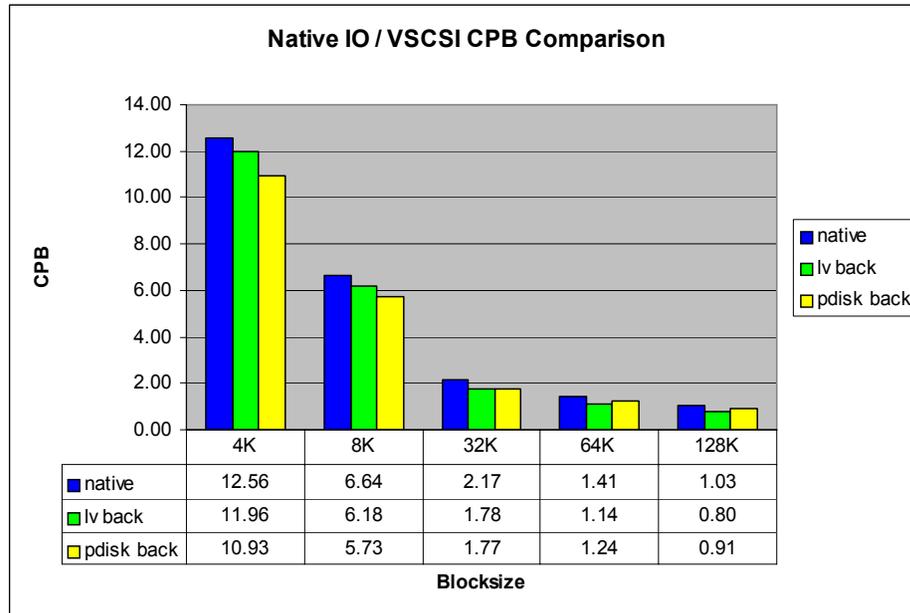


Figure 6-49 Comparison of native I/O to virtual SCSI

Configuration example

Consider a Virtual I/O Server partition that supports three client partitions on physical disk backed storage. The first client partition requires a maximum of 7,000 8-KB operations per second. The second client partition requires a maximum of 10,000 8-KB operations per second. The third client partition requires a maximum of 5,000 128-KB operations per second. The number of 1.65 GHz processors for this requirement is approximately:

$$(7,000*47,000+10,000*47,000+5,000*120,000)/1,650,000,000 = 0.85 \text{ processors}$$

We round up this total to one processor, as we are not using Micro-Partitioning.

Sizing against installed storage

An alternative approach, if you do not know the I/O rates of the client partitions, is to size the virtual SCSI server to the maximum I/O rate of the attached storage subsystem. The sizing could be biased to small I/Os or large I/Os. Sizing to maximum capacity for large I/Os balances the processor capacity of the virtual SCSI server to the potential I/O bandwidth of the attached I/O. The negative facet of this sizing methodology is that, in nearly every case, we will assign more processor entitlement to the virtual SCSI server than it typically consumes.

Consider a case where an I/O server manages 32 physical SCSI disks. We can arrive at an upper bound of processors required based on assumptions about the

I/O rates that the disks can achieve. If it is known that the workload is dominated by 8 KB operations that are random, we could assume that each disk is capable of approximately 200 disk I/Os per second (15 K rpm drives). At peak, the I/O server would have to service approximately 32 disks * 200 I/Os per second * 120,000 cycles per operation, resulting in a requirement for approximately 19% of one processor performance. Viewed another way, an I/O server running on a single processor should be capable of supporting more than 150 disks serving 8 KB random I/Os for other partitions' CPUs.

Alternatively, if the server is sized for maximum bandwidth, the calculation will result in a much higher processor requirement. The difference is that maximum bandwidth assumes sequential I/O. Because disks are much more efficient when performing large sequential I/Os than small random I/Os, we can drive a much higher number of I/Os per second. Assume that the disks are capable of 50 MB per second when doing 128 KB I/Os. That implies that each disk could average 390 disk I/Os per second. Thus, the entitlement necessary to support 32 disks, each doing 390 I/Os per second with an operation cost of 120,000 cycles ($32 * 390 * 120,000 / 1,650,000,000$), is approximately 0.91 processors. Simply put, an I/O server running on a single processor should be capable of driving approximately 32 fast disks to maximum throughput.

This sizing method can be very wasteful of processor entitlement when using dedicated processor partitions, but will guarantee peak performance. It is most effective if the average I/O size can be estimated so that peak bandwidth does not have to be assumed.

Sizing when using Micro-Partitioning

Defining virtual SCSI servers in micro-partitions enables much better granularity of processor resource sizing and potential recovery of unused processor time by uncapped partitions. Tempering those benefits, use of micro-partitions for virtual SCSI servers slightly increases I/O response time and creates somewhat more complex processor entitlement sizings.

The sizing methodology should be based on the same operation costs as for Virtual I/O Server partition. However, additional entitlement should be added for running in micro-partitions. We recommend that the Virtual I/O Server partition be configured as uncapped so that if it is undersized, it is possible to get more processor time to service I/O.

Because I/O latency with virtual SCSI varies with the machine utilization and Virtual I/O Server topology, consider the following:

1. For the most demanding I/O traffic (high bandwidth or very low latency), try to use native I/O.

2. If native I/O is not an option and the system contains enough processors, consider putting the Virtual I/O Server in a dedicated processor partition.
3. If using a Micro-Partitioning Virtual I/O Server, use as few virtual processors as possible.

6.9 Summary

Virtualization is an innovative technology that redefines the utilization and economics of managing an on demand operating environment. The POWER5 architecture provides new opportunities for clients to take advantage of virtualization capabilities. Virtual I/O provides the capability for a single physical I/O adapter to be used by multiple logical partitions of the same server, enabling consolidation of I/O resources.

Overhead of virtual SCSI in both a SCSI and a FASTT implementation is small, and clients should assess the benefits of the virtual SCSI implementation for their environment. Simultaneous multithreading should be enabled in a virtual SCSI environment. With low I/O loads and a small number of partitions, Micro-Partitioning of the Virtual I/O Server partition has little effect on performance. For a more efficient virtual SCSI implementation with larger loads, it may be advantageous to keep the I/O server as a dedicated processor. LVM mirroring should be used for redundancy.

Virtual SCSI implementation is an excellent solution for clients looking to consolidate I/O resources with a modest amount of processor overhead. The new POWER5 virtual SCSI capability creates new opportunities for consolidation, and demonstrates strong performance and manageability.



Part 2

Virtualization support and tuning

In this part we look at the changes to performance tools in the AIX 5L Version 5.3 operating system, POWER5 processor-based system performance, application tuning, and the Partition Load Manager.



AIX 5L Version 5.3 operating system support

In this chapter we discuss what is new in AIX 5L V5.3 from a performance and POWER5 point of view in the following sections:

- ▶ Physical and virtual processors
- ▶ Simultaneous multithreading
- ▶ Metrics problems
- ▶ Updated and new performance commands
- ▶ Logical Volume Manager
- ▶ Paging space
- ▶ Physical and virtual networks

7.1 Introduction

The appropriate version of AIX 5L for POWER5 is AIX 5L Version 5.3 for virtualization, which has modifications for the new functionalities of the POWER5 processor. AIX 5L Version 5.2 is also supported but cannot use the new features of POWER5. Versions prior to AIX 5L V5.2 are not supported.

The implementation of the logical processor abstraction is provided by the POWER5 architecture and the POWER Hypervisor firmware. From an operating system perspective, a logical processor is indistinguishable from a physical processor.

7.1.1 Processors

Before AIX 5L V5.3, physical processors were dedicated to partitions. With the virtualization capabilities of the POWER5 architecture, the concept of a logical processor can be used by the operating system.

Logical processors

A logical processor is seen by the operating system as being a single physical processor. In reality, it is just a single hardware thread on the processor. Changes to ODM have been made to reflect the new type of processors, Example 7-1 shows attributes of a POWER4 processor and Example 7-2 shows two new attributes for a POWER5 processor. The attributes of processors are exactly the same whether the operating system is running with a dedicated processor or in Micro-Partitioning.

Example 7-1 Attributes of POWER4 processor

```
lsattr -El proc3
frequency 1499960128      Processor Speed False
state      enable         Processor state False
type       PowerPC_POWER4 Processor type  False
```

Example 7-2 Attributes of POWER5 processor

```
# lsattr -El proc0
frequency 1656424000      Processor Speed      False
smt_enabled false        Processor SMT enabled False
smt_threads 2           Processor SMT threads False
state      enable         Processor state      False
type       PowerPC_POWER5 Processor type        False
```

Optimizations

For the most part, AIX 5L V5.3 should be able to run and function on a Micro-Partitioning system with no changes. However, to optimize OS performance as well as the collective performance of all shared partitions, it is important for the OS to add some specific Micro-Partitioning optimizations. These optimizations involve giving up the processor in the idle process so that another logical processor in our partition or so even another partition could use it.

We can call two POWER Hypervisor calls to control those optimizations:

H_CEDE Used to give processor cycles to the pool.
H_PROD Used to restore processor cycles to the processor that has ceded them.

Simultaneous multithreading

On AIX 5L V5.3 with simultaneous multithreading enabled, each hardware thread is supported as a separate logical processor. A dedicated partition with one physical processor is seen under AIX 5L V5.3 as a partition with two logical processors, as shown in Figure 7-1. The same applies to Micro-Partitioning: A logical processor partition is configured by AIX 5L V5.3 as a logical six-way partition. The two hardware threads are also called sibling threads.

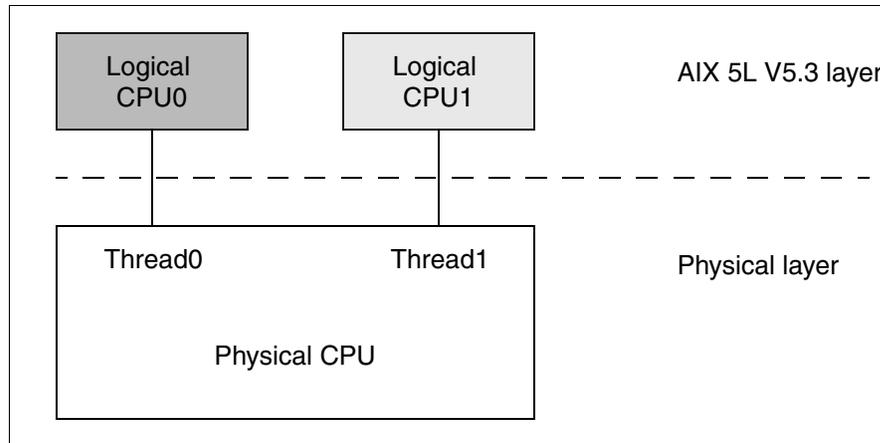


Figure 7-1 Logical versus physical processors

Simultaneous multithreading can be enabled or disabled dynamically with the **smtctl** command. The change can also be made at next boot and persists across system boots. By default, simultaneous multithreading is enabled.

The syntax of the **smtctl** command is:

```
smtctl [ -m off | on [ -w boot | now]]
```

When you enter the command without any flags, it returns information about the status of simultaneous multithreading on your system as shown in Example 7-3.

Example 7-3 smtctl command example

```
# smtctl

This system is SMT capable.

SMT is currently enabled.

SMT boot mode is not set.

Processor 0 has 2 SMT threads
SMT thread 0 is bound with processor 0
SMT thread 1 is bound with processor 0
```

The boot image includes an option for simultaneous multithreading. If the simultaneous multithreading mode is changed, the boot image must be recreated; otherwise at the next reboot the simultaneous multithreading mode will be the same as the previous boot.

Normally, AIX 5L V5.3 maintains sibling threads at the same priority but will boost or lower thread priorities in a few key places to optimize performance. AIX 5L V5.3 lowers thread priorities when the thread is doing non-productive work spinning in the idle loop or on a kernel lock. When a thread is holding a critical kernel lock, AIX 5L V5.3 boosts the thread priorities. These priority adjustments do not persist into user mode. AIX 5L V5.3 does not consider a software thread a dispatching priority when choosing its hardware thread priority.

Several scheduling enhancements were made to exploit simultaneous multithreading. For example, work will be distributed across all primary threads before being dispatched to secondary threads, because the performance of a thread is best when its sibling thread is idle. AIX 5L also considers thread affinity in idle stealing and periodic run queue load balancing.

For detailed information about simultaneous multithreading, refer to Chapter 3, “Simultaneous multithreading” on page 41.

Metrics problems

A dedicated partition that is created with one real processor is configured by AIX 5L V5.3 as a logical two-way by default. This is independent of the partition type, so a shared partition with two logical processors is configured by AIX 5L V5.3 as a logical four-way by default. Logically, the only supported kernel in a simultaneous multithreading environment is the multiprocessor.

In traditional processor utilization, data collection is sample-based. There are 100 samples per second sorted into four categories:

user	Interrupted code outside AIX 5L V5.3 kernel.
sys	Interrupted code inside AIX 5L V5.3 kernel and current running thread is not waitproc.
iowait	Current running thread is waitproc and there is an I/O pending.
idle	Current running thread is waitproc and there is no I/O pending.

Each sample corresponds to a 10 ms (1/100 sec.) clock tick. These are recorded in the `sysinfo` (system-wide) and `cpuinfo` (per-processor) kernel data structures. To preserve binary compatibility, this stayed unchanged with AIX 5L V5.3.

Note: Performance tools such as `vmstat`, `iostat`, and `sar` convert tick counts from the `sysinfo` structure into utilization percentages for the machine or partition. For other tools, such as `sar -P ALL` and the `topas` “hot cpu” section, there is a conversion of tick counts from `cpuinfo` into utilization percentages for a processor or thread.

Of course, this greatly affects the metrics. Traditional utilization metrics are misleading because the tools believe that there are two physical processors when in fact we only have one. As an example, one thread 100% busy and one thread idle would result in 50% utilization, but the physical processor is really 100% busy. This is similar to what happened with hardware multithreading, and the same problem exists with hyperthreading.

New metrics

The displayed `%user`, `%sys`, `%idle`, `%wait` are now calculated using the PURR-based metrics. Using the example in which one thread is 100% busy and the other is idle, reported utilization would no longer be 50% but the correct 100%. This is because one thread would receive (almost) all of the PURR increments and the other (practically) none, meaning 100% of PURR increments would go into the `%user` and `%sys` buckets. This is a more reasonable indicator of the division of work between the two threads. Unfortunately, this hides the simultaneous multithreading gain.

We now show the new metrics on AIX 5L V5.3 with simultaneous multithreading. We measure two different times: the thread’s processor time and the elapsed time. For the first, we use the thread’s PURRs, which are now virtualized. To measure the elapsed time we use the Timebase register (TB).

For the physical resource utilization metric for a logical processor, we use $(\text{delta PURR}/\text{delta TB})$, which represents the fraction of the physical processor consumed by a logical processor, and $((\text{delta PURR}/\text{delta TB}) * 100)$ over an interval to represent the percentage of dispatch cycles given to a logical processor.

Using PURR-based samples and entitlement, we calculate the “physical” processor utilization metrics. As an example we have:

$$\%sys = (\text{delta PURR in system mode}/\text{entitled PURR}) * 100$$

entitled PURR equals $(\text{ENT} * \text{delta TB})$ and ENT is entitlement in number of processors (entitlement/100).

When we need to know how much physical processor is being consumed (PPC) we use $\text{sum}(\text{delta PURR}/\text{delta TB})$ for each logical processor in a partition. The result is in decimal number of processors.

We also may need the percentage of entitlement consumed: $(\text{PPC}/\text{ENT}) * 100$.

Another useful metric is the available pool of processors. Taking the *pool idle count* (PIC), which represents clock ticks where the POWER Hypervisor was idle (that is, all partition entitlements are satisfied and there is no partition to dispatch), then we have $(\text{delta PIC}/\text{delta TB})$.

This also results in decimal number of processors.

Logical processor utilization is useful for figuring out whether to add more logical processors to a partition. We calculate it by summing the old 10 ms tick-based %sys and %user.

There are two other usages for the PURR. The first is the measurement of relative simultaneous multithreading split between threads and is simply the ratio $\text{purr0}/\text{purr1}$. To know the fraction of time partition1 ran on a physical processor (the relative amount of processing units consumed), use $(\text{purr0} + \text{purr1}) / \text{timebase0}$.

Binary compatibility

As with every release of AIX 5L, the maintenance of the binary compatibility is a requirement. In a Micro-Partitioning LPAR, commands such as the **bindprocessor** continue to work, albeit binding to the logical processor and not a physical processor. This aspect could possibly cause problems for an application or kernel extension, which is dependent on executing on a specific physical processor. For example, the AIX 5L V5.3 Floating-Point Diagnostic Test unit relied on the ability to bind itself to and execute the FP test unit to completion on each physical processor in the system.

Another example is the **bindintcpu** command, which enables an administrator to bind bus interrupt levels to specific processors. In Micro-Partitioning, AIX 5L V5.3 supports it, and will bind interrupts to logical processors. However, it will have no effect on the original intent of this command, which was to control the physical distribution of interrupts. The impact will be no absolute control over the routing of interrupts to physical processors when running in Micro-Partitioning mode. We do not expect this to be a significant risk because that type of physical resource management does not make sense in a Micro-Partitioning environment, and workloads that require specific distribution of interrupts probably would not be candidates for running in a Micro-Partitioning environment.

There could also be an impact on third-party performance tools due to resulting inconsistent or erroneous statistics, unless those tools become Micro-Partitioning aware.

7.1.2 Dynamic re-configuration

Dynamic operations enable the addition or removal of resources from a logical partition without rebooting.

A dynamic remove operation on a CPU may fail for various reasons; the most common reason for a removal failure is because a process is bound to a processor. To provide more information to the user, the **cpupstat** command was added. It helps to identify processes bound to logical processors. Example 7-4 shows that the **cpupstat** command first checks WLM classes, then rset attachments, and finally the logical processor number 2. If a process is bound to a processor, it can be unbound with the **bindprocessor** command. The highest bind ID is removed if the processor dynamic removal operation succeeds.

Example 7-4 cpupstat output

```
# cpupstat -i 2
0 WLM classes have single CPU rsets with CPU ID 2.
0 processes have single CPU rset attachments with CPU ID 2.
0 processes are bound to bind ID 2.
```

7.1.3 Existing performance commands enhancement

Due to simultaneous multithreading, Micro-Partitioning, and the ability to dynamically change some parameters, it was necessary to make some changes to the old tools. (See 8.1, “Performance commands” on page 258.)

If simultaneous multithreading is enabled or in a Micro-Partitioning environment, the **vmstat**, **iostat**, and **sar** commands automatically use the new PURR-based data and formula for %user, %sys, %wait, and %idle.

In Micro-Partitioning mode, new metrics are displayed, Example 7-5 shows the traditional output of **vmstat** command run on a dedicated partition.

Example 7-5 vmstat on a dedicated partition

```
# vmstat 2

System configuration: lcpu=2 mem=1024MB

kthr  memory                page                faults                cpu
-----
 r b  avm  fre  re pi po fr  sr cy in  sy cs us sy id wa
 0 0 56057 196194  0  0  0  0  0  0  3  68 191  0  0 99  0
 0 0 56058 196192  0  0  0  0  0  0  2   6 183  0  0 99  0
```

Example 7-6 shows the same **vmstat** command run on a micro-partition, adding the columns **pc** (physical processor consumed) and **ec** (entitled capacity consumed).

Example 7-6 vmstat on micro-partition

```
# vmstat 2

System configuration: lcpu=6 mem=512MB ent=0.3

kthr  memory                page                faults                cpu
-----
 r b  avm  fre  re pi po fr  sr cy in  sy cs us sy id wa  pc  ec
 0 0 46569 73370  0  0  0  0  0  0  1  73 149  0  1 99  0  0.00  1.5
 0 0 46577 73360  0  0  0  0  0  0  0   7 147  0  0 99  1  0.00  1.1
```

The following list gives the new metrics for each command:

- ▶ **vmstat**
 - Number of physical processors consumed.
 - Percentage of entitled capacity consumed.
- ▶ **iostat**
 - Percentage of entitled capacity consumed.
 - Percentage of physical processor consumed.
- ▶ **sar**
 - Number of physical processors consumed.
 - Percentage of entitled capacity consumed.
- ▶ **topas**
 - Number of physical processors consumed.
 - Percentage entitlement consumed.
 - New display dedicated to logical processors.

All of these tools have a new feature called dynamic configuration support. They need it because we no longer work in a static environment with a fixed number of processors and memory. This way the tools start by a new pre-header with the configuration but if the configuration changes, there is a warning. The tool then prints the current iteration line, followed by the summary line (in a **sar** case). The tool shows a new configuration pre-header and the regular header for the tool and continues. Obviously, each tool is monitoring a different set of configuration parameters, but when running in a shared partition, they all monitor the entitlement. In Example 7-7, while **vmstat** is running on a one logical processor partition, a configuration change occurred. The warning message is displayed, then the new configuration shows that a processor has been added.

Example 7-7 vmstat pre-header

```
# vmstat 2

System configuration: 1cpu=1 mem=1024MB

kthr  memory                page                faults                cpu
-----
r  b  avm  fre re pi po fr  sr cy in  sy cs us sy id wa
0  0 59481 193172  0  0  0  0  0  0  3  67 95  0  0 99  0
0  0 59481 193172  0  0  0  0  0  0  4  12 94  0  0 99  0
1  0 59481 193172  0  0  0  0  0  0  3  17 97  0  0 99  0
System configuration changed. The current iteration values may be inaccurate.
8  0 59741 192881  0  0  0  0  0  0  2  443 83  0 30 69  0

System configuration: 1cpu=2 mem=1024MB

kthr  memory                page                faults                cpu
-----
r  b  avm  fre re pi po fr  sr cy in  sy cs us sy id wa
0  0 59773 192849  0  0  0  0  0  0  1  16 87  0  0 99  0
0  0 59773 192849  0  0  0  0  0  0  0  7 90  0  0 99  0
```

The trace base tools **filemon**, **netpmon**, **curt**, and **splat** commands have been updated to give accurate information about processor usage.

Another tool that needed modification was **trace/trcrpt**. In a simultaneous multithreading environment, **trace** can optionally collect PURR register values at each trace hook, and **trcrpt** can display elapsed PURR. The **trace** tool added new trace hooks that enable the tracing of phantom interrupts. All trace-based tools will adjust processor times using a preemption hook. In addition, most POWER Hypervisor calls are traceable, so they will appear in **trcrpt** output.

Reporting tools **curt** and **splat** can optionally use the PURR values to calculate processor times in a simultaneous multithreading environment. For **splat** the -p

option specifies the use of the PURR register. **curt** shows physical affinity and phantom interrupt statistics when in a Micro-Partitioning environment. It also shows the *POWER Hypervisor call* summary reports similar to system calls reports, the number of preemptions, and the number of H_CEDE and H_CONFER POWER Hypervisor calls for each individual logical processor, as shown in Example 7-8.

Example 7-8 curt output - preemptions, H_CEDE and H_CONFER

```

Processor Summary processor number 0
-----
processing      percent      percent
total time      total time    busy time
(msec) (incl. idle) (excl. idle) processing category
=====
0.02           1.59          1.64 APPLICATION
0.08           5.26          5.43 SYSCALL
7.03          471.38        486.24 HCALL
0.19           12.43         12.82 KPROC (excluding IDLE and NFS)
0.00           0.00          0.00 NFS
1.10           73.50         75.81 FLIH
0.04           2.77          2.86 SLIH
0.02           1.39          1.44 DISPATCH (all procs. incl. IDLE)
0.01           0.44          0.45 IDLE DISPATCH (only IDLE proc.)
-----
1.45           96.94         100.00 CPU(s) busy time
0.05           3.06
-----
1.49                                     TOTAL

```

Avg. Thread Affinity = 1.00

Total number of process dispatches = 5
 Total number of idle dispatches = 5

Total Physical CPU time (msec) = 8.64
 Physical CPU percentage = 0.60
 Physical processor affinity = 0.997590
 Dispatch Histogram for processor (PHYSICAL CPUid : times_dispatched).
 PHYSICAL CPU 0 : 415

Total number of preemptions = 415
 Total number of **H_CEDE** = 415 with preemption = 414
 Total number of **H_CONFER** = 0 with preemption = 0

Processor Summary processor number 1

```

-----
processing      percent      percent
total time      total time    busy time

```

(msec)	(incl. idle)	(excl. idle)	processing category
=====	=====	=====	=====
243.43	98.65	98.65	APPLICATION
2.26	0.91	0.91	SYSCALL
0.03	0.01	0.01	HCALL
0.00	0.00	0.00	KPROC (excluding IDLE and NFS)
0.00	0.00	0.00	NFS
1.07	0.43	0.43	FLIH
0.00	0.00	0.00	SLIH
0.01	0.00	0.00	DISPATCH (all procs. incl. IDLE)
0.01	0.00	0.00	IDLE DISPATCH (only IDLE proc.)
-----	-----	-----	
246.77	100.00	100.00	CPU(s) busy time
0.00	0.00		IDLE
-----	-----		
246.77			TOTAL

Avg. Thread Affinity = 1.00

Total number of process dispatches = 5
Total number of idle dispatches = 2

Total Physical CPU time (msec) = 246.80
Physical CPU percentage = 19.74
Physical processor affinity = 0.997126
Dispatch Histogram for processor (PHYSICAL CPUid : times_dispatched).
PHYSICAL CPU 1 : 348

Total number of preemptions = 348
Total number of **H_CEDE** = 2 with preemption = 2
Total number of **H_CONFER** = 0 with preemption = 0

MAPI

With the new POWER5 processors, it was necessary to update the PMAPI.

There is a new API for POWER5 processor called `pm_initialize`, which you must use instead of the old `pm_init` API. With the updated PMAPI, there is a new way to return event status and characteristics: by bit array instead of char. There is a new *shared* characteristic for processors supporting simultaneous multithreading. A shared event is controlled by a signal not specific to a particular thread's activity and sent simultaneously to both sets (one for each thread) of hardware counters. There should be an average of counts across sibling threads. The added processor features bit array in the `pm_initialize` has two bits currently defined: the POWER Hypervisor mode and runlatch mode. Moreover, `pm_initialize` can also retrieve the event table for another processor instead of the old way in which we could only retrieve the tables for the current processor.

The new PMAPI now supports the M:N threading model, as opposed to the previous 1:1 model. This new model enables mapping M user threads to N kernel threads, and M is much bigger than N. There is a new set of APIs for third-party calls (debugger) generically called `pm_*_thread`, which differs from the old `pm_*_thread` interfaces in an additional argument to specify `ptid`. In 1:1 mode, there is no need to specify the `ptid`, but if you specify it, the library will verify that the specified `pthread` runs on the specified kernel thread. On the other hand, to use the M:N mode, the `ptid` must always be specified. If `ptid` is not specified, then there is the assumption that the `pthread` is currently undispached. Regarding all other APIs, they are unchanged but now work in M:N mode.

With this new API come some new commands, including `pm1ist` and `pmcycles`. `pm1ist` is a utility to dump and search a processor's event and group tables. It currently supports text and spreadsheet output formats. The `pmcycles` command uses the Performance Monitor cycle counter and the processor real-time clock to measure the actual processor clock speed in MHz, as shown in Example 7-9.

Example 7-9 pmcycles output

```
# pmcycles -m
Cpu 0 runs at 1656 MHz
Cpu 1 runs at 1656 MHz
Cpu 2 runs at 1656 MHz
Cpu 3 runs at 1656 MHz
Cpu 4 runs at 1656 MHz
Cpu 5 runs at 1656 MHz
```

GPROF

The new environment variable GPROF controls the **gprof** new mode that supports multi-threaded applications.

```
GPROF=[profile:{process|thread}] [,][scale:<scaling_factor>] [,][file:{one|multi|multithread}]
```

Here:

profile	Indicates whether it will do thread-level or process-level profiling.
scaling_factor	Represents the granularity of the collected profiling data.
file	Indicates whether it will generate a single or multiple <code>gmon.out</code> files.
multi	Creates a file for each process (for each fork or exec) <code>gmon.out.<progname>.<pid></code> .
multithread	Creates a file for each <code>pthread</code> <code>gmon.out.<progname>.<pid>.Pthread<ptid></code> that can be

used to look at one pthread at a time with **gprof** or **xprofiler**.

The default values for **gprof** are process for the profile option, a scaling factor of 2 for process level and 8 for thread level (the thread level profiling consumes considerably more memory), and one file for the output. Several flags enable optional separate output into multiple files:

- g filename** Writes the call graph information to the specified output filename. It suppresses the profile information unless -p is used.
- p filename** Writes flat profile information to the specified output filename. It suppresses the call graph information unless -g is used.
- i filename** Writes the routine index table to the specified output filename. If this flag is not used, the index table goes either at the end of the standard output or at the bottom of the filename (or filenames) specified with -p and -g.

The format of data itself is unchanged but now it can be presented in multiple sets in which the first set has cumulative data and the following sets have the data per thread.

Graphical tools

As with text-based tools, the processor accounting for graphical tools has to be changed to use the new metrics regarding the shared mode environment and simultaneous multithreading. Graphics tools such as PTX® 3dmon, PTX xmpperf, and PTX jtopas have also been updated.

PTX 3dmon and xmpperf

The most complete graphical tool, PTX, now uses PURR-based utilization metrics and entitlement utilization. An example of a **3dmon** display is shown in Figure 7-2 on page 246, and **xmpperf** Mini Monitor is shown in Figure 7-3 on page 246.

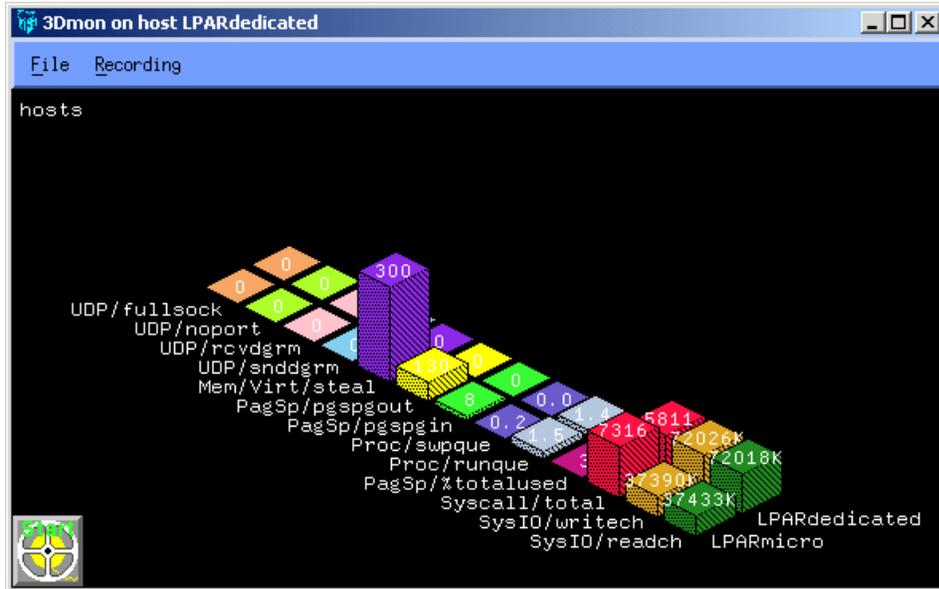


Figure 7-2 3dmon monitoring two LPARs

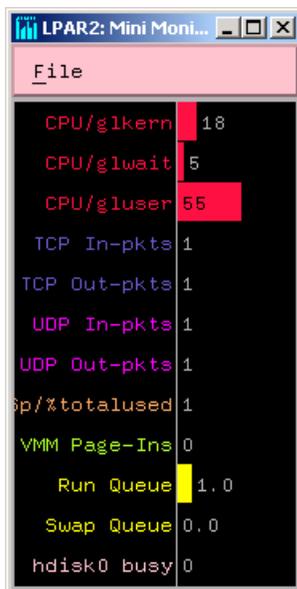


Figure 7-3 xmp perf Mini Monitor

PTX jtopas

Shipping with PTX since May 2003, the graphical tool **jtopas** is a hot-resource monitoring tool and a sibling of **topas**. **jtopas** starts with a predefined (no setup needed) Swing GUI. In the main screen, it shows a set of system metrics and hot-resource summaries similar to **topas**, with access to more detailed information for each area. This is a generalization of the P, W, and L commands of **topas**, which provide process, partition, and WLM detail reports. **jtopas** works locally or remotely, and it can generate dynamic reports with up to seven days of playback. It keeps data automatically for a week in seven rotating daily files, enabling **jtopas** to generate reports by hour or by day. You can save these reports in HTML format or in spreadsheet format. You can have a week-by-days report and a day-by-hours report. **jtopas** is a Swing GUI-enabled application, which means that you can minimize or move each window and all resources are always available using the scroll bar. **jtopas** uses the **xmtrend** daemon.

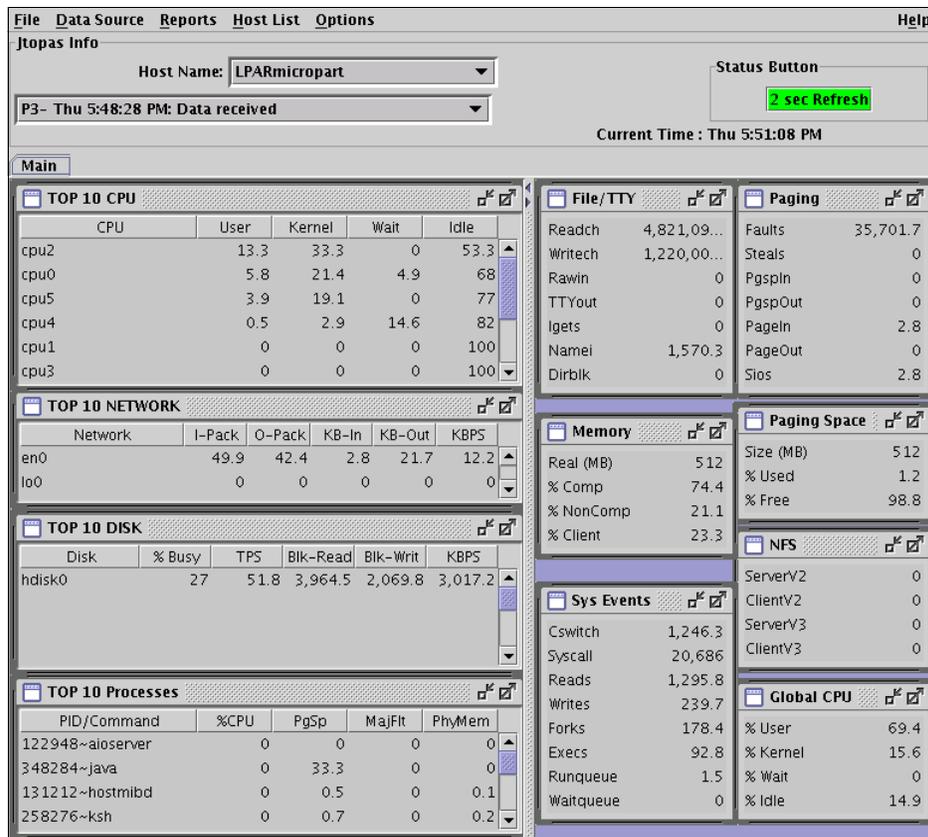


Figure 7-4 jtopas default display

7.1.4 New performance commands

Some new tools have been added for performance tuning in simultaneous multithreading or Micro-Partitioning environments.

The following list gives the main features of the **lparstat**, **mpstat**, and **perfwb** commands:

► **lparstat**

- Information and statistics about the partition.
- Details about the configuration of the partition.
- Summary and detailed POWER Hypervisor statistics and information.
- **lparstat** command output changes depending on the partition mode, as shown in Example 7-10 and Example 7-11.

In Micro-Partitioning, the real resource consumed by the user is the percentage of CPU user (%user) times percentage of entitlement consumed (entc%) times the entitlement (ent). In this case, 18.2% of 42.4% of 0.30 gives 2.3% of CPU consumed by the user.

The physical processor consumed (physc) is equal to the percentage of %entc times ent.

Example 7-10 lparstat output in a dedicated partition

```
# lparstat
```

```
System configuration: type=Dedicated mode=Capped smt=0n lcpu=2 mem=1024
```

```
%user  %sys  %wait  %idle
-----
 67.6   31.8   0.0    0.6
```

Example 7-11 lparstat output in Micro-Partitioning

```
# lparstat
```

```
System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 ent=0.30
```

```
%user  %sys  %wait  %idle  physc  %entc  lbusy  app  vcsw  phint
-----
 18.2   12.5   0.9    68.4   0.13  42.4   4.3    -   2747  3
```

- ▶ **mpstat**
 - Basic utilization metrics.
 - Logical and physical processor metrics (in simultaneous multithreading mode).
 - Interrupt metrics.
 - Logical processor affinity metrics.
 - The **mpstat** command output changes depending on the partition mode.
- ▶ **perfwb**
 - Dynamic process monitoring.
 - Partition-wide metrics about processor and memory activity (Figure 7-5).

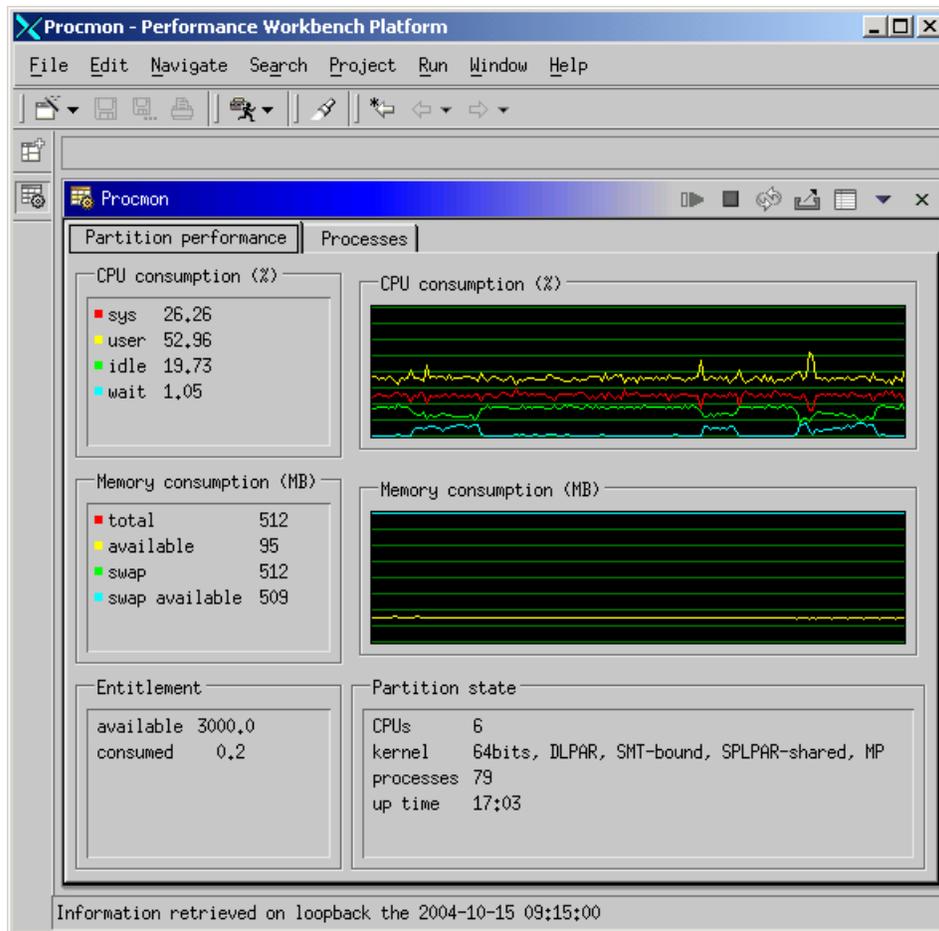


Figure 7-5 *procmon partition-wide metrics*

- Sorted list of processes, as shown in Figure 7-6.

PID	% CPU	↓ %...	User time	Command	Eff. login
262384	0	9	0:24.33	java	root
274568	0	2	0:00.10	IBM.ERmtd	root
372794	0	1	0:00.00	Imigratepp	root
241804	0	1	0:35.58	IBM.CSMAgentRmtd	root
20490	0	0	0:00.00	wait	root
24588	0	0	0:00.00	wait	root
28686	0	0	0:00.00	wait	root
32784	0	0	0:00.00	wait	root
36882	0	0	0:00.00	wait	root
40980	0	0	0:00.00	wait	root
45078	0	0	0:00.00	wait	root
49176	0	0	0:00.00	reaper	root
53274	0	0	0:00.00	lrud	root
57372	0	0	0:00.00	vmptact	root
61470	0	0	0:00.00	xmfreed	root
65568	0	0	0:00.00	memgrdd	root
69666	0	0	0:00.00	psgc	root
...
//	0	13	//	//	//

Information retrieved on loopback the 2004-10-15 09:19:14

Figure 7-6 *procmon: sorted list of processes*

- Columns can be added or removed, and sorted in ascending or descending order. Actions can be performed on listed processes, such as **kill**, **renice**, run performance commands, and obtain information.
- Part of the bos.perf.gtools fileset; start Performance Workbench with the **perfwb** command to launch the procmon tool.

For more about performance commands, see 8.1, “Performance commands” on page 258.

7.1.5 Paging space

AIX 5L V5.3 introduces enhanced paging space management algorithms to collect paging space as needed. They apply only to deferred page space allocation policy. This management may be useful when paging space is almost full, for example, due to dynamic memory removal. In that case, if the memory is added back to the partition, the paging space will not be freed. This adds some constraints on paging space although there is free real memory in the partition.

- ▶ Garbage collect paging space on re-pagein

This mechanism applies only to deferred page space allocation policy. A new mechanism determines whether to free a disk block after a pagein operation, depending on the free space remaining in the paging space.

- ▶ Garbage collect paging space scrubbing for in-memory frames

This mechanism tries to reclaim paging space disk blocks for pages that are already in memory, if the free space that is available in the paging space decreases under a tunable limit.

Tuning parameters for paging space garbage collection

Tuning on paging space parameters is performed with the `vm0` command. New parameters include:

npsrpgmin	Low paging space threshold for re-pagein garbage collector to start.
npsrpgmax	High paging space threshold for re-pagein garbage collector to stop.
rpgclean	Configures re-pagein garbage collector to be active when a page is read or when a page is read or write.
rpgcontrol	Enables or disables re-pagein garbage collector.
npssubmin	Low paging space threshold for garbage collector scrubbing to start.
npsscrubnax	High paging space threshold for garbage collector scrubbing to stop.
scrubclean	Configures garbage collector scrubbing to be active when a page is read or when a page is read or write.
scrub	Enables or disables garbage collector scrubbing.

7.1.6 Logical Volume Manager (LVM)

Several improvements have been made to the AIX 5L V5.3 Logical Volume Manager that concern performance.

Scalable volume group

The new scalable volume group supports up to 1024 disks. This expands the capacity of the volume groups but needs a substantially larger volume group descriptor area (VGDA) and volume group status area (VGSA). Increasing maximum logical volumes or maximum physical partitions per volume group from the defaults toward the limits increases the amount of metadata (VGDA or VGSA) that must be read or written during LVM operations. Every VGDA update operation (creating a logical volume, changing a logical volume, adding a physical volume, and so on) might take longer to run, as LVM keeps a copy of metadata on each physical volume. In previous AIX 5L releases, the maximum number of PPs was defined per disk; it is now defined per volume group. The limits for each type of volume group are listed in Table 7-1.

Table 7-1 Maximum values for volume groups

VG type	PVs	LVs	PPs	PP size
Normal VG	32	256	1016 per disk	1 GB
Big VG	128	512	1016 per disk	1 GB
Scalable VG	1024	4096	2097152 per VG	128 GB

Variable logical track group (LTG)

The LVM device driver breaks I/O into LTG-size chunks before passing the I/O to the device driver of the underlying disks. LTG size is an attribute of the volume group. In the previous release, LTG size was defined at volume group creation or update; now it is determined at vary on time and will be dynamically updated if a physical volume is added or removed in the volume group. AIX 5L V5.3 enables the stripe size of a logical volume to be larger than the LTG size of the volume group, which was not allowed previously. Also, AIX 5L V5.3 now supports larger LTG sizes and stripe sizes. Valid LTG and stripe sizes are listed in Table 7-2.

Table 7-2 LTG and stripe sizes

AIX release	Valid LTG sizes	Valid stripe sizes
AIX 5L V5.2 and previous	128 KB, 256 KB, 512 KB, 1 MB	4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1 MB
AIX 5L V5.3	adds support for 2 MB, 4 MB, 8 MB, 16 MB	adds support for 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB

Performance improvements

LVM metadata (VGDA and VGSA) must be stored in every single disk in a volume group. To improve performance, AIX 5L V5.3 writes all metadata in parallel. There is one thread for each disk in the volume group. Previously, some commands that would read data, utilize a small piece, then read again, and utilize a small piece. They now read the metadata once and keep it accessible throughout the life of the command. Special focus was put on these commands: **extendvg**, **importvg**, **mkvg**, **varyonvg**, **chlvcopy**, **mklvcopy**, **lslv**, and **lspv**.

Striped column

Prior to AIX 5L V5.3, there was no good way to extend a striped logical volume if one of the disks was full. The workaround was to back up the data, delete the striped logical volume, remake the logical volume with a larger stripe width, then restore the data. Now, we can extend a striped logical volume even if one of the disks is full. We do this by modifying the maximum number of physical volumes for the new allocation, the upper bound. Prior to AIX 5L V5.3, the stripe width and upper bound were required to be equal. In AIX 5L V5.3, the upper bound can be a multiple of stripe width, where you can think of each stripe as a “column.” You can use the **extendlv** command to extend a striped logical volume into the next column. You can use **extendlv -u** to raise the upper bound and extend the logical volume all in one operation (like a combined **extendlv** and **chlv -u**).

Volume group pbuf pools

The LVM uses a structure called pbuf to handle disk I/O. In previous versions, pbuf pool was a system-wide resource; now each volume group gets its own pbuf pool. To manage pbuf, we use the **lvmo** command, which displays and tunes several volume group specific items:

pv_pbuf_count	Number of pbufs added when a physical volume is added to the volume group. It is tunable with the lvmo command, and it takes effect immediately.
total_vg_pbufs	Number of pbufs currently available for the volume group. It is tunable with the lvmo command, and it takes effect at varyonvg time.
max_vg_pbuf_count	Maximum number of pbufs for this volume group. It is tunable with lvmo command, takes effect at varyonvg time.
pervg_blocked_io_count	Number of I/Os that were blocked due to lack of free pbufs for this volume group. Can only be displayed; not tunable.

The `lvmo` command also displays the following system-wide items:

global_pbuf_count Minimum number of pbufs that are added when a physical volume is added to any volume group. It is tunable with the `ioo` command. It takes effect at `varyonvg` time.

global_blocked_io_count
System-wide number I/Os that were blocked due to lack of free pbufs.

For more information about LVM, refer to the redbook *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463

7.1.7 Virtual local area network (VLAN)

VLAN is a method to logically segment a physical network, which means that only adapters belonging to a same VLAN can communicate. AIX 5L V5.3 supports *virtual Ethernet* technology, which enables communications between logical partitions on the same system using a VLAN.

Shared Ethernet Adapter technology enables the logical partitions to communicate with machines that are outside the system without any *physical Ethernet* slots assign to the logical partition. The Shared Ethernet Adapter creates a relationship between virtual Ethernet adapters and a real network adapter. The Shared Ethernet Adapter is part of the optional Virtual I/O Server.

In a dedicated partition or micro-partition we can configure physical and virtual adapters at the same time. Example 7-12 shows two types of adapters, the physical `ent0` and the virtual `ent1`. Each has a network address. The Device Specific.(YL) field contains in one case (`ent0`) a real physical location code and in the other case (`ent1`) a logical location code given by the Virtual I/O Server.

Example 7-12 Ethernet adapters

```
# lsdev -Ccadapter
ent0 Available 02-08 10/100 Mbps Ethernet PCI Adapter II (1410ff01)
ent1 Available Virtual I/O Ethernet Adapter (1-lan)

# lscfg -vl ent0
ent0 U787A.001.DNZ00XY-P1-C2-T1 10/100 Mbps Ethernet PCI Adapter II (1410ff01)

10/100 Mbps Ethernet PCI Adapter II:
Part Number.....09P5023
FRU Number.....09P5023
EC Level.....H10971A
Manufacture ID.....YL1021
Network Address.....000D600A58A4
ROM Level.(alterable).....SCU015
```

```

Product Specific.(Z0).....A5204209
Device Specific.(YL).....U787A.001.DNZ00XY-P1-C2-T1

# lscfg -vl ent1
ent1 U9111.520.10DDEDC-V2-C10-T1  Virtual I/O Ethernet Adapter (1-lan)

Network Address.....C6BB3000200A
Displayable Message.....Virtual I/O Ethernet Adapter (1-lan)
Device Specific.(YL).....U9111.520.10DDEDC-V2-C10-T1

```

For more about VLAN, refer to 6.5, “Virtual Ethernet” on page 164.

7.1.8 EtherChannel

The EtherChannel technology is based on port aggregation, which means that Ethernet adapters are aggregated together and belong to the same network. They share the same IP address and the same hardware address. The bandwidth of the EtherChannel adapter is increased due to the aggregation of physical Ethernet adapters.

Prior to AIX 5L V5.3, addition or removal operations of a physical adapter member of an EtherChannel was possible only if the interface was detached or not configured. The interface also must be detached in order to modify EtherChannel attributes.

With AIX 5L V5.3, the Dynamic Adapter Membership enables addition, removal, and update operations at runtime. A failed Ethernet adapter can be replaced without IP disruption.

A failover can be manually forced on the condition that the EtherChannel has a working backup adapter. This is useful for recovering from a failover caused by a failure. The recovery time to primary has been improved.

7.1.9 Partition Load Manager

Partition Load Manager for AIX 5L is a load manager that balances resources (processor and memory) between partitions executing within the same physical server.

To benefit from Partition Load Manager, the managed partitions must be running AIX 5L V5.2 or AIX 5L V5.3. (Linux and i5/OS are not supported.) Partition Load Manager works with both dedicated partitions and micro-partitions.

Partition Load Manager allocates resources to partitions according to rules defined by the system administrators. In Partition Load Manager terminology,

these rules are called policies. The policies define how Partition Load Manager assigns unused resources or resources from partitions with low usage to partitions with a higher demand, improving the overall resource utilization of the system.

Partition Load Manager is implemented using a client/server model. The server part of Partition Load Manager is packaged as part of the Advanced Power Virtualization feature of @server p5 servers. There is no special code to install on client partitions that are managed by Partition Load Manager.

The Partition Load Manager client/server model is event-based, not polling-based. The PLM server receives events each time one of its managed partitions needs extra resources.

When the PLM server starts, it registers several events on each managed partition. In order for Partition Load Manager to get system information and dynamically reconfigure resources, it requires an SSH network connection from the PLM server to the HMC. The Resource Management and Control (RMC) services are responsible for gathering all of the status information on the managed nodes. The RMC daemon exports system status attributes and processes the reconfiguration requests from HMC. With this data and in conjunction with the user-defined resource management policy, the PLM server decides what to do each time a partition exceeds one of the thresholds defined in the Partition Load Manager policies.

Partition Load Manager is presented in more detail in Chapter 10, “Partition Load Manager” on page 373.



POWER5 system performance

This chapter provides information about system performance and how to diagnose a problem with a processor, memory, or I/O. We mainly focus on new components introduced by the @server p5 architecture and virtualization.

The following topics are discussed:

- ▶ AIX 5L commands for performance analysis
- ▶ Performance and tuning on a system

8.1 Performance commands

Table 8-1 summarizes all of the AIX 5L commands described in this section. These commands are used to show typical performance issues in this chapter.

Table 8-1 Commands summary

Command	Function	Main measurement	Page number
lparstat	Logical partition information and statistics	CPU, Hypervisor	258
mpstat	Physical and logical processors statistics	CPU	264
vmstat	CPU and virtual memory monitoring	CPU, memory	268
iostat	System input/output device monitoring	Disk I/O	270
sar	Physical, logical processor, and I/O monitoring	CPU	272
topas	Displays system statistics dynamically.	CPU, memory, I/O	275
xmperf	Displays a great amount of system statistics	CPU, memory, I/O	278

8.1.1 lparstat command

The **lparstat** command reports logical partition information and statistics as well as POWER Hypervisor statistics. It displays on its first line a summary of the partition configuration. Table 8-2 gives a summary for **lparstat** command (a + sign means the command covers this topic).

Table 8-2 lparstat command summary

Command name	lparstat
Interface type	CLI
Updated or new command	new
AIX 5L package	bos.acct
Measurement CPU memory disk I/O network POWER Hypervisor	+ +

Command name	lparstat
Environment	
Dedicated partition	+
Micro-partition	+
Simultaneous multithreading	+
Virtual I/O Server	

Usage

```
lparstat { -i | [-H | -h] [Interval [Count]] }
```

Most important flags

- i** Displays information about the configuration of the logical partition.
- h** Adds summary POWER Hypervisor information to the default output.
- H** Displays detailed POWER Hypervisor information, including statistics for each of the POWER Hypervisor calls.

Output examples

In the default mode and on a dedicated partition, the **lparstat** command shows processor utilization in the usual manner (%user, %sys, %idle, %wait), as shown in Example 8-1.

Example 8-1 lparstat default mode on a dedicated partition

```
# lparstat 2

System configuration: type=Dedicated mode=Capped smt=On lcpu=2 mem=1024

%user  %sys  %wait  %idle
-----  ----  -----  -----
 99.2   0.7   0.0    0.0
 99.7   0.3   0.0    0.0
 99.8   0.2   0.0    0.0
```

In the default mode and on a micro-partition, **lparstat** adds the following information to the output, as shown in Example 8-2 on page 260:

- psize** Number of online physical processors in the shared pool.
- physc** Shows the number of physical processor consumed.
- %entc** Shows the percentage of the entitled capacity consumed.
- lbusy** Shows the percentage of logical processors utilization that occurred while executing at the user and system level.
- app** Shows the available processing capacity in the shared pool.

- vcsw** Number of virtual context switches that are the virtual processor hardware preemptions.
- phint** Shows the number of phantom (targeted to another shared partition in this pool) interruptions received.

Example 8-2 lparstat default mode on a micro-partition

```
# lparstat
```

```
System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 psize=1
ent=0.30
```

%user	%sys	%wait	%idle	physc	%entc	lbusy	app	vcsw	phint
8.2	3.0	0.2	88.6	0.00	0.2	3.3	0.89	168708	148

An *interval* and a *count* can be added to the command to display statistics every *interval* seconds for *count* iterations. In Example 8-3, the interval is 2 seconds and the count is 5.

Example 8-3 lparstat monitoring

```
# lparstat 2 5
```

```
System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 psize=1
ent=0.30
```

%user	%sys	%wait	%idle	physc	%entc	lbusy	app	vcsw	phint
70.2	14.1	0.0	15.6	0.53	175.9	23.4	0.00	2453	31
62.6	19.8	0.0	17.6	0.49	161.7	18.9	0.08	2611	24
67.6	16.8	0.0	15.6	0.35	117.9	14.8	0.23	2409	15
52.6	20.5	0.0	26.8	0.27	88.6	11.4	0.27	2486	8
61.5	21.2	0.0	17.4	0.32	106.3	11.9	0.22	2829	13

If the partition does not have shared processor pool utilization authority, the app column will not be displayed (Example 8-4 on page 261). This option enables the logical partition to collect information statistics from the managed system about shared processing pool utilization. Shared processors are processors that are shared between two or more logical partitions. The processors are held in the shared processor pool and are shared among the logical partitions.

Example 8-4 Logical partition without pool utilization authority

```
# lparstat 2 5
```

```
System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 psize=1  
ent=0.30
```

%user	%sys	%wait	%idle	physc	%entc	lbusy	vcsw	phint
67.8	16.7	0.0	15.5	0.66	219.6	16.2	3343	3
60.1	21.6	0.1	18.2	0.47	157.1	7.3	2402	2
65.6	18.7	0.0	15.6	0.86	287.9	17.3	3714	2
67.2	16.8	0.0	16.0	0.71	235.6	16.4	3411	2
67.4	16.5	0.0	16.1	0.66	220.7	17.3	3299	2

To choose this option, connect to the Hardware Management Console (HMC), edit the partition properties, click the **Hardware** tab and the **Processor and Memory** tab, then select the **Allow shared processor utilization authority** check box as in Figure 8-1.

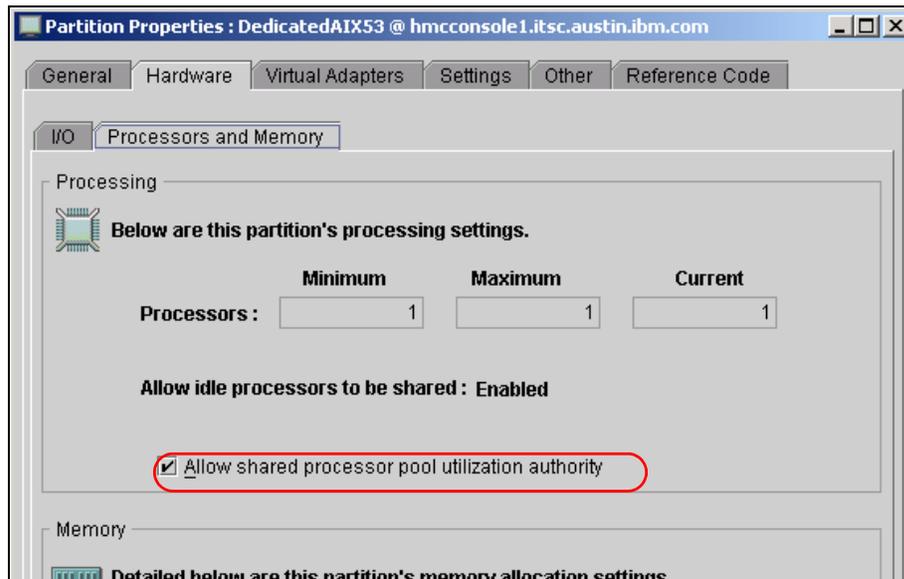


Figure 8-1 Shared processor utilization authority activation

The -h flag adds the percentage of time spent in POWER Hypervisor (%hypv) and the number of POWER Hypervisor calls executed to the default output, as shown in Example 8-5.

Example 8-5 lparstat -h output

```
# lparstat -h 2
```

System configuration: type=Shared mode=Capped smt=Off lcpu=2 mem=512 psize=2 ent=0.30

%user	%sys	%wait	%idle	physc	%entc	lbusy	app	vcsw	phint	%hypv	hcall
----	----	----	----	----	----	----	----	----	----	----	----
98.4	1.1	0.0	0.5	0.30	99.7	58.7	0.69	309	3	0.6	153
98.0	1.1	0.0	0.9	0.30	99.3	55.5	0.69	303	1	0.5	146
97.9	1.1	0.0	1.0	0.30	99.2	56.9	0.69	304	0	0.6	147
93.7	1.1	0.0	5.2	0.28	95.0	58.4	0.70	292	0	0.6	143
72.2	0.9	0.0	26.9	0.22	73.3	59.5	0.77	208	0	0.5	91
95.3	1.0	0.0	3.7	0.29	96.4	53.2	0.70	184	0	0.2	36
86.9	1.0	0.0	12.2	0.26	88.0	55.9	0.72	258	0	0.4	100
95.9	1.1	0.0	3.0	0.29	97.2	52.8	0.70	298	0	0.6	146
98.1	1.0	0.0	0.9	0.30	99.4	56.4	0.69	312	1	0.5	157
98.4	1.1	0.0	0.5	0.30	99.7	59.8	0.69	314	0	0.6	162
97.4	1.1	0.0	1.5	0.30	98.7	55.6	0.69	303	0	0.6	157
98.2	1.1	0.0	0.7	0.30	99.5	56.3	0.69	311	0	0.5	158

For information about the partition, such as minimum and maximum of CPU and memory, partition type, and mode use **lparstat -i** as shown in Example 8-6 on page 263.

The partition type can be one of the following:

- Dedicated** Processors are dedicated to the partition; simultaneous multithreading is disabled.
- Dedicated simultaneous multithreading**
Processors are dedicated to the partition; simultaneous multithreading is enabled.
- Shared** Partition is configured for Micro-Partitioning; simultaneous multithreading is disabled.
- Shared simultaneous multithreading**
Partition is configured for Micro-Partitioning; simultaneous multithreading is enabled.

For more information, see Chapter 3, “Simultaneous multithreading” on page 41.

The partition mode can be:

Capped	Partition is not allowed to consume idle cycles from the shared pool. Dedicated LPAR is implicitly capped.
Uncapped	Partition may use idle cycles from the shared pool if needed.

For more about Micro-Partitioning mode, refer to Chapter 5, “Micro-Partitioning” on page 93.

Example 8-6 lparstat -i output

```
# lparstat -i
Node Name                : LPARmicro
Partition Name           : MicroPartitionAIX53
Partition Number         : 4
Type                     : Shared-SMT
Mode                     : Uncapped
Entitled Capacity        : 0.30
Partition Group-ID       : 32772
Shared Pool ID           : 0
Online Virtual CPUs      : 3
Maximum Virtual CPUs     : 5
Minimum Virtual CPUs     : 2
Online Memory            : 512 MB
Maximum Memory           : 1024 MB
Minimum Memory           : 128 MB
Variable Capacity Weight : 128
Minimum Capacity         : 0.20
Maximum Capacity         : 0.50
Capacity Increment       : 0.01
Maximum Dispatch Latency : 17995218
Maximum Physical CPUs in system : 2
Active Physical CPUs in system : 2
Active CPUs in Pool      : 1
Unallocated Capacity     : 0.00
Physical CPU Percentage   : 10.00%
Unallocated Weight       : 0
```

Detailed information about the POWER Hypervisor calls are displayed with the **lparstat -H** command (Example 8-7 on page 264) especially the `cede` and `confer` values used by the operating system to return processor resources to the hardware when it no longer has demand for it or when it is waiting on an event to complete. In this example, time spent for `cede` is 99.2% of the total time spend in the POWER Hypervisor.

Example 8-7 lparstat -H output

```
# lparstat -H 2 1
```

```
System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 psize=1  
ent=0.30
```

Detailed information on Hypervisor Calls

Hypervisor Call	Number of Calls	%Total Time Spent	%Hypervisor Time Spent	Avg Call Time(ns)	Max Call Time(ns)
remove	6	0.0	0.0	268	647
read	2	0.0	0.0	50	449
nclear_mod	0	0.0	0.0	1	0
page_init	3	0.0	0.0	405	1989
clear_ref	0	0.0	0.0	1	0
protect	0	0.0	0.0	1	0
put_tce	141	0.0	0.2	689	2100
xirr	81	0.0	0.2	790	2791
eoi	80	0.0	0.1	449	927
ipi	0	0.0	0.0	1	0
cppr	80	0.0	0.0	221	434
asr	0	0.0	0.0	1	0
others	0	0.0	0.0	1	0
enter	11	0.0	0.0	250	874
cede	211	6.9	99.2	51344	70485
migrate_dma	0	0.0	0.0	1	0
put_rtce	0	0.0	0.0	1	0
confer	0	0.0	0.0	1	0
prod	151	0.0	0.1	338	1197
get_ppp	1	0.0	0.0	850	2583
set_ppp	0	0.0	0.0	1	0
purr	0	0.0	0.0	1	0
pic	1	0.0	0.0	125	758
bulk_remove	0	0.0	0.0	1	0
send_crq	70	0.0	0.1	777	2863
copy_rdma	0	0.0	0.0	1	0
get_tce	0	0.0	0.0	1	0
send_logical_lan	3	0.0	0.0	2178	4308
add_logical_lan_buf	9	0.0	0.0	638	1279

8.1.2 mpstat command

The **mpstat** command collects and displays performance statistics for all logical CPUs in the system. It can show up to 29 new metrics (when using -a option). Table 8-3 on page 265 gives a summary of the **mpstat** command.

Table 8-3 *mpstat* command summary

Command name	mpstat
Interface type	CLI
Updated or new command	new
AIX 5L package	bos.acct
Measurement Processor Memory Disk I/O Network POWER Hypervisor	+
Environment Dedicated partition Micro-partition Simultaneous multithreading Virtual I/O Server	+ ++ ++

Usage

```
mpstat [ { -a | -d | -i | -s } ] [ -w ] [ interval [ count ] ]
```

Most important flags

- d** Displays detailed affinity and migration statistics for AIX 5L V5.3 threads.
- i** Displays detailed interrupt statistics.
- s** Displays simultaneous multithreading utilization report if simultaneous multithreading is enabled.

The default mode shows:

- ▶ Utilization metrics (%user, %sys, %idle, %wait).
- ▶ Major and minor page faults (with and without disk I/O).
- ▶ Number of syscalls and interrupts.
- ▶ Dispatcher metrics, namely the number of migrations, voluntary and involuntary context switches, logical processor affinity (percentage of redispatches inside MCM), and run queue size.
- ▶ Fraction of processor consumed (simultaneous multithreading or Micro-Partitioning only).
- ▶ Percentage of entitlement consumed (Micro-Partitioning mode only).
- ▶ Number of logical context switches (Micro-Partitioning mode only), meaning the hardware preemptions.

Output examples

The default mode of the **mpstat** command shown in Example 8-8 displays the following information to show activity for each logical processor:

- mpc** Total number of interprocessor calls.
- cs** Total number of logical processor context switches.
- ics** Total number of involuntary context switches.
- mig** Total number of thread migrations to another logical processor.
- lpa** Logical processor affinity. The percentage of logical processor redispatches within the scheduling affinity domain 3 (same Multi-chip Module).

At the end of the output, the U line displays the unused capacity, and the ALL line is the sum of all virtual processors.

Example 8-8 mpstat default output

```
# mpstat 2 2

System configuration: lcpu=6 ent=0.3

cpu min maj mpc int cs ics rq mig lpa sysc us sy wa id pc %ec lcs
 0 135 0 0 688 358 179 0 0 100 3343 38 57 0 4 0.04 12.0 295
 1 0 0 0 46 0 0 0 0 - 0 0 11 0 89 0.01 3.1 290
 2 0 0 0 188 100 50 0 0 100 0 1 51 0 47 0.00 0.9 179
 3 0 0 0 43 0 0 0 0 - 0 0 59 0 41 0.00 0.9 179
 4 0 0 0 56 308 157 0 1 100 9 0 69 0 31 0.00 1.3 215
 5 0 0 0 37 0 0 0 1 100 0 0 30 0 70 0.00 0.7 195
 U - - - - - - - - - - - - 0 81 0.24 81.3 -
ALL 135 0 0 1058 766 386 0 2 100 3352 5 9 0 86 0.06 18.9 676
```

```
-
 0 29 0 0 261 34 17 0 0 100 725 43 52 0 4 0.01 3.6 69
 1 0 0 0 14 0 0 0 0 - 0 0 12 0 88 0.00 0.9 69
 2 15 0 0 166 108 53 0 0 100 5 3 59 0 37 0.00 0.8 129
 3 0 0 0 15 0 0 0 0 - 0 0 29 0 71 0.00 0.5 129
 4 0 0 0 23 60 35 0 0 100 0 0 66 0 34 0.00 0.4 61
 5 0 0 0 13 0 0 0 0 - 0 0 20 0 80 0.00 0.2 59
 U - - - - - - - - - - - - 0 94 0.28 93.7 -
ALL 44 0 0 492 202 105 0 0 100 730 2 3 0 96 0.02 6.3 258
```

In the **mpstat -d** output shown in Example 8-9 on page 267, the **rq** column shows the run queue size for each logical processor. The columns from **S0rd** to **S5rd** show the percentage of thread redispaches within a scheduling affinity domain. See the definition of “processor affinity” on page 105.

Example 8-9 mpstat -d output

```
# mpstat -d 2 1
```

```
System configuration: lcpu=6 ent=0.3
```

cpu	cs	ics	bound	rq	push	S3pull	S3grd	S0rd	S1rd	S2rd	S3rd	S4rd	S5rd	ilcs	vlcs
0	202	150	0	0	0	0	0	99.1	0.0	0.0	0.9	0.0	0.0	134	285
1	5	1	0	0	0	0	0	0.0	100.0	0.0	0.0	0.0	0.0	0	411
2	222	158	0	0	0	0	0	97.1	2.2	0.0	0.7	0.0	0.0	182	330
3	28	15	0	0	0	0	0	93.3	6.7	0.0	0.0	0.0	0.0	0	520
4	320	223	0	0	0	0	0	98.9	0.7	0.0	0.4	0.0	0.0	83	370
5	11	9	0	0	0	0	0	86.7	13.3	0.0	0.0	0.0	0.0	2	452
ALL	788	556	0	0	0	0	0	97.1	2.4	0.0	0.5	0.0	0.0	200	1184

A logical partition receives different kinds of interrupts. Example 8-10 shows for each logical processor the following interrupt metrics:

- mpcs, mpcr** Interrupts used to communicate between processors.
- dev** Number of hardware interrupts (external interrupts).
- soft** Number of software interrupts. (When a hardware interrupt takes too much time to complete, a software interrupt is created to finish the processing.)
- dec** Number of decremter interrupts. The decremter is the register used to generate time-based interrupts. AIX 5L loads a value in it, the processor decrements the register, and when it reaches zero, an interrupt is sent.
- ph** Number of phantom interrupts (the number of device interrupts received by the partition but targeted to another partition in the pool). The OS simply returns those to POWER Hypervisor.

Example 8-10 mpstat -i output

```
# mpstat -i 2 1
```

```
System configuration: lcpu=6 ent=0.3
```

cpu	mpcs	mpcr	dev	soft	dec	ph
0	0	0	20	5	105	0
1	0	0	23	0	10	0
2	0	0	25	48	195	1
3	0	0	24	0	11	0
4	0	0	23	0	101	0
5	0	0	21	0	11	0
ALL	0	0	136	53	433	1

If simultaneous multithreading is enabled, the **mpstat -s** command displays physical as well as logical processors usage, as shown in Example 8-11. Physical processor Proc0 is busy at 17.80%, which is dispatched on logical processor cpu0 (14.75%) and on logical processor cpu1 (3.05%). In this case, cpu0 and cpu1 are hardware threads for proc0.

Example 8-11 mpstat -s output

```
# mpstat -s 2 1
```

System configuration: lcpu=6 ent=0.3

	Proc0		Proc2		Proc4
	17.80%		16.24%		13.67%
cpu0	cpu1	cpu2	cpu3	cpu4	cpu5
14.75%	3.05%	13.51%	2.73%	11.18%	2.49%

8.1.3 vmstat command

The **vmstat** command reports statistics about kernel threads, virtual memory, disks, traps and processor activity. Table 8-4 gives a summary for **vmstat**.

Table 8-4 vmstat command summary

Command name	vmstat
Interface type	CLI
Updated or new command	update
AIX 5L package	bos.acct
Measurement	
Processor	++
Memory	++
Disk I/O	+
Network	
POWER Hypervisor	
Environment	
Dedicated partition	+
Micro-partition	+
Simultaneous multithreading	+
I/O server	

Usage

```
vmstat [ -fsviItlw ] [Drives] [ Interval [Count] ]
```

Most important flags

- t Prints the time stamp next to each line of output.
- v Writes to standard output various statistics maintained by the Virtual Memory Manager.

Output examples

In Example 8-12, we are running **vmstat** on an uncapped partition with 0.3 processing unit. At the beginning the partition is idle, the processor consumed (pc) is 0, and the percentage of entitlement consumed (ec) is 1.4%.

As activity begins on the partition, the percentage of CPU usage increases to 93% because the partition is uncapped and the processor pool is not fully utilized. The percentage of entitlement consumed increases to 330% and the processor consumed is nearly 1. This means that the partition is running on almost a full processor although it has been given only 0.3 processing unit.

Example 8-12 Activity on uncapped partition shown by vmstat command

```
# lparstat

System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 psize=2
ent=0.30

%user  %sys  %wait  %idle  physc  %entc  lbusy  app  vcsw  pinct
-----
  0.0   0.0   0.2   99.7   0.00   0.1    0.4   1.17 1063520 333

# vmstat 5

System configuration: lcpu=6 mem=512MB ent=0.30

kthr    memory                page                faults                cpu
-----
 r  b  avm  fre  re  pi  po  fr  sr  cy  in  sy  cs  us  sy  id  wa  pc  ec
0  0  46863 1408  0  0  0  0  0  0  0  32 139  0  0  99  0  0.00  1.4
0  0  46865 1406  0  0  0  0  0  0  3  26 177  0  1  99  0  0.00  1.4
1  0  46903 1368  0  0  0  0  0  0  1 1566 143 93  1  6  0  0.35 118.3
1  0  46903 1368  0  0  0  0  0  0  1 4323 137 93  0  6  0  0.99 330.8
1  0  46903 1368  0  0  0  0  0  0  0 4333 137 93  0  6  0  0.99 330.8
1  0  46903 1368  0  0  0  0  0  0  0 4330 139 93  0  6  0  0.99 330.7
1  0  46903 1368  0  0  0  0  0  0  0 4259 130 93  0  6  0  0.99 330.5
1  0  46903 1368  0  0  0  0  0  0  1 4010 130 93  1  6  0  0.91 301.9
1  0  46903 1368  0  0  0  0  0  0  0 2934 136 93  1  6  0  0.68 225.5
1  0  46903 1368  0  0  0  0  0  0  0 3578 140 93  1  6  0  0.82 272.6
1  0  46903 1368  0  0  0  0  0  0  1 2694 134 93  1  7  0  0.63 208.4
```

```

1 0 46865 1406 0 0 0 0 0 0 0 2311 136 93 1 6 0 0.53 177.6
0 0 46865 1406 0 0 0 0 0 0 0 8 136 0 0 99 0 0.00 1.1
0 0 46865 1406 0 0 0 0 0 0 0 10 142 0 0 99 0 0.00 1.2

```

8.1.4 iostat command

The **iostat** command is used for monitoring system input/output device loading. Table 8-5 gives a summary of the **iostat** command.

Table 8-5 *iostat* command summary

Command name	iostat
Interface type	CLI
Updated or new command	update
AIX 5L package	bos.acct
Measurement Processor Memory Disk I/O Network POWER Hypervisor	+ ++
Environment Dedicated partition Micro-partition Simultaneous multithreading Virtual I/O Server	+ + + +

Usage

```
iostat [-astTdmAPqQ1] [Drives] [Interval [Count]]
```

Most important flags

- d** Displays drive report only.
- t** Displays tty/cpu report only.
- T** Prints the time stamp next to each line of output.

Output examples

The **iostat** output in Example 8-13 on page 271 shows the two new columns:

- %physc** The percentage of physical processor consumed.
- %entc** The percentage of entitled capacity consumed.

Example 8-13 *iostat* command

```
# iostat 2 1

System configuration: lcpu=6 drives=2 ent=0.30

tty:  tin  tout  avg-cpu:  % user  % sys  % idle  % iowait  % physc  % entc
      0.0  25.0           58.4  22.6   19.1    0.0    0.5    182.4

Disks:      % tm_act   Kbps    tps    Kb_read  Kb_wrtn
hdisk0      40.0      440.4   110.6     0        880
cd0         0.0       0.0     0.0     0         0
```

The **iostat** command provides a new way to look at asynchronous I/O. You can check the statistics of either legacy asynchronous I/O or POSIX asynchronous I/O. You can use several flags:

- A** Shows processor utilization and asynchronous I/O statistics.
- q** Shows adapter individual queues and their request counts.
- Q** Shows mounted file systems and their associated adapter queue and request counts.
- P** Is similar to -A option, but for the POSIX adapter extension data.

When using -A or -P, new columns replace the tty information (Example 8-14):

- avgc** Average global non-fastpath adapter request count per second for the specified interval.
- avfc** Average fastpath request count per second for the specified interval.
- maxg** Maximum global non-fastpath adapter request count since the last time it fetched this value.
- maxf** Maximum fastpath request count since the last time it fetched this value.
- maxr** Maximum adapter I/O requests allowed on queue.

Example 8-14 *iostat* legacy adapter I/O

```
# iostat -A 3 3

System configuration: lcpu=3 drives=3 ent=2.00

aio:  avgc avfc maxg maxf maxr avg-cpu:  %user  %sys  %idle  %iow  physc  %entc
      0    0    0    0  4096           42.9  23.4  13.4  20.3   0.0  80.4

Disks:      % tm_act   Kbps    tps    Kb_read  Kb_wrtn
hdisk0      0.0       0.0     0.0     0         0
hdisk2      24.3     2293.4   573.4   6972     0
```

```

hdisk1          98.0    32000.0    125.0    48640    48640

aio: avgc avfc maxg maxf maxr avg-cpu: %user %sys %idle %iow physc %entc
      0    0    0    0 4096          43.5 23.3 12.6 20.6  0.0 81.3

Disks:      % tm_act    Kbps    tps    Kb_read    Kb_wrtn
hdisk0      0.0      0.0      0.0      0          0
hdisk2      21.3     2328.0    582.0    6984      0
hdisk1      99.3     28832.3   127.3    42041     44456

aio: avgc avfc maxg maxf maxr avg-cpu: %user %sys %idle %iow physc %entc
      0    0    0    0 4096          43.0 23.5 13.2 20.3  0.0 80.9

Disks:      % tm_act    Kbps    tps    Kb_read    Kb_wrtn
hdisk0      0.0      0.0      0.0      0          0
hdisk2      24.0     2265.3    566.3    6796      0
hdisk1      98.3     28576.3   126.3    41785     43944

```

Important: Some system resources are consumed in maintaining disk I/O history for **iostat**. Use the **sysconfig** subroutine or the System Management Interface Tool (SMIT) to stop history accounting if it is not needed.

8.1.5 sar command

The **sar** command writes to standard output the contents of selected cumulative activity counters in the operating system.

Table 8-6 gives a summary for the **sar** command.

Table 8-6 *sar command summary*

Command name	sar
Interface type	CLI
Updated or new command	update
AIX 5L package	bos.acct
Measurement	
Processor	+
Memory	
Disk I/O	+
Network	
POWER Hypervisor	

Command name	sar
Environment	
Dedicated partition	+
Micro-partition	+
Simultaneous multithreading	+
Virtual I/O Server	

Usage

```
sar [ -Aabcdkmqruvw ] [ Interval [ Number ]
```

Most important flags

- t** Prints the time stamp next to each line of output.
- v** Writes to standard output various statistics maintained by the Virtual Memory Manager.

Output examples

The default output of **sar** in Example 8-15 shows the two new columns:

physc The number of physical processors consumed.

%entc The percentage of entitled capacity consumed.

Example 8-15 Default sar output

```
# sar 2 5

AIX LPARmicro 3 5 00CDEDC4C00 10/20/04

System configuration: lcpu=6 ent=0.30

11:28:00 %usr %sys %wio %idle physc %entc
11:28:02 70 16 0 15 0.67 224.1
11:28:04 62 20 0 18 0.47 156.0
11:28:06 68 17 0 15 0.78 258.6
11:28:08 66 18 0 16 0.78 258.9
11:28:10 68 16 0 16 0.65 216.9

Average 67 17 0 16 0.67 222.9
```

The **-P ALL** output for all logical processors view option of the **sar** command with simultaneous multithreading enabled, or in Micro-Partitioning (Example 8-16 on page 274) shows the physical processor consumed **physc** (delta PURR/delta TB). This column shows the relative simultaneous multithreading split between processors (the measurement of the fraction of time a logical processor was getting physical processor cycles). When running in shared mode, **sar** displays

the percentage of entitlement consumed (%entc), which is ((PPFC/ENT)*100). This gives relative entitlement utilization for each logical processor and enables system average utilization calculation from logical processor utilization.

Example 8-16 Logical processor usage

```
# sar -P ALL 2 2
```

```
AIX LPARmicro 3 5 00CDEDC4C00 10/20/04
```

```
System configuration: lcpu=6 ent=0.30
```

11:30:25	cpu	%usr	%sys	%wio	%idle	physc	%entc
11:30:27	0	60	38	0	2	0.09	30.6
	1	0	2	0	98	0.02	7.2
	2	75	24	0	1	0.17	55.3
	3	0	1	0	99	0.03	11.4
	4	78	21	0	1	0.15	49.9
	5	0	2	0	98	0.03	9.5
	-	60	22	0	18	0.49	163.9
11:30:29	0	78	22	0	0	0.23	77.8
	1	0	1	0	99	0.04	14.9
	2	74	25	0	1	0.19	63.6
	3	0	1	0	99	0.03	9.8
	4	84	15	0	1	0.19	63.9
	5	0	1	0	99	0.03	8.9
	-	68	18	0	14	0.72	238.8
Average	0	73	26	0	1	0.16	54.3
	1	0	1	0	99	0.03	11.0
	2	74	25	0	1	0.18	59.5
	3	0	1	0	99	0.03	10.6
	4	82	17	0	1	0.17	56.9
	5	0	1	0	99	0.03	9.2
	-	65	19	0	16	0.60	201.4

Whenever the percentage of entitled capacity consumed is less than 100%, a line beginning with U is added to represent the unused capacity (Example 8-17 on page 275).

Example 8-17 Unused capacity displayed by sar command

```
# sar -P ALL 2 1

AIX LPARmicro 3 5 00CDEDC4C00    10/20/04

System configuration: lcpu=6 ent=0.30

11:31:22 cpu    %usr    %sys    %wio    %idle    physc    %entc
11:31:24  0        21      66      0        13      0.00    0.5
           1         0       11      0        89      0.00    0.1
           2         0       37      0        63      0.00    0.4
           3         0        5       0        95      0.00    0.2
           4         9       48      0        43      0.00    0.3
           5         0        4       0        96      0.00    0.2
           U         -        -       0      98    0.29  98.3
           -         0        1       0        99      0.01    1.7
```

8.1.6 topas command

The **topas** command reports selected statistics about the activity on the local system. The command displays its output in a format suitable for viewing on an 80x25 character-based display.

Table 8-3 on page 265 gives a summary for the **topas** command.

Table 8-7 topas command summary

Command name	topas
Interface type	CLI
Updated or new command	update
AIX 5L package	bos.perf.tools
Measurement Processor Memory Disk I/O Network POWER Hypervisor	+ + + +
Environment Dedicated partition Micro-partition Simultaneous multithreading Virtual I/O Server	+ + + +

Usage

topas [-dhimpwcPLUDW]

Most important flags

- i** Sets the monitoring interval in seconds. The default is 2 seconds.
- L** Displays the logical partition display. This display reports data similar to what is provided to **mpstat** and **lparstat**.

Output examples

The **topas** output as shown in Example 8-18 has been modified. In addition to changes on the main screen, a new one dedicated to virtual processors has been added. The new metrics have been applied, so processor utilization is calculated using the new PURR-based register and formula when running in simultaneous multithreading or Micro-Partitioning mode. When running in Micro-Partitioning mode, **topas** automatically adds new information:

Physc The fractional number of processors consumed.

%Entc The percentage of entitled capacity consumed.

Example 8-18 topas output

```
-----
Topas Monitor for host:  LPARmicro          EVENTS/QUEUES  FILE/TTY
Wed Oct 20 14:20:07 2004  Interval: 2          Cswitch  4080 Readch  46.9M
                               Syscall  15352 Writech  38.2M
Kernel  28.4 |#####| Reads    3386 Rawin    0
User    56.1 |#####| Writes   2383 Ttyout   302
Wait    0.4 |#| Forks    68 Igets    0
Idle    15.1 |####| Execs    56 Namei   1145
Physc = 0.77                               %Entc= 257.2 Runqueue  1.0 Dirblk  0
                               Waitqueue  0.0

Network KBPS  I-Pack  O-Pack  KB-In  KB-Out
en0     39830.3  54316.0  3313.0  78470.1  194.7  PAGING
lo0      0.0    0.0    0.0    0.0    0.0    Faults  15912 Real,MB  511
                               Steals  0 % Comp  51.1
Disk    Busy%   KBPS    TPS  KB-Read  KB-Writ  PgpsIn  0 % Noncomp  10.4
hdisk0  46.5    6583.3  144.3  12630.0  372.0  PgpsOut  0 % Client  12.0
                               PageIn  9
Name     PID  CPU%  PgSp  Owner  PageOut  0 PAGING SPACE
ftpd    200756  8.3  0.9  root  Sios     9 Size,MB  512
ksh     323618  0.0  0.6  root  % Used   0.9
topas   290946  0.0  1.1  root  NFS (calls/sec) % Free  99.0
gil     69666  0.0  0.1  root  ServerV2  0
getty   270468  0.0  0.4  root  ClientV2  0 Press:
rpc.lockd 237690  0.0  0.2  root  ServerV3  0 "h" for help
syncd   94344  0.0  0.5  root  ClientV3  0 "q" to quit
netm    65568  0.0  0.0  root
```

IBM.CSMAg	274568	0.0	2.1	root
rmcd	262274	0.0	1.4	root
wlmsched	73764	0.0	0.1	root
sendmail	82016	0.0	0.9	root

The new LPAR screen (Example 8-19) is accessible with the -L command line flag or by typing L while topas is running. It splits the screen in an upper section, which shows a subset of **lparstat** metrics, and a lower section that shows a sorted list of logical processors with **mpstat** columns. The **%hypv** and **hcalls** give the percentage of time spent in POWER Hypervisor and the number of calls made. The **pc** value is the fraction of physical processor consumed by a logical processor. When in Micro-Partitioning there are additional metrics:

Psize	Number of online physical processors in the pool.
physc	Number of physical processors consumed.
%entc	Percentage of entitlement consumed.
%lbusy	Logical processor utilization.
app	Available pool processors (the number of physical processor available in the shared pool).
lcsw and vcsw	Logical and virtual context switches per second over the monitoring interval.
phint	Number of phantom interrupts.
%hypv	Shows the percentage of time spent in POWER Hypervisor.

Example 8-19 Logical processors view using topas

```
Interval: 2      Logical Partition: MicroPartitionAIAvWed Oct 20 14:23:51
2004
Psize:      1      Shared SMT ON      Online Memory: 512.0
Ent: 0.30      Mode: UnCapped      Online Logical CPUs: 6
Partition CPU Utilization      Online Virtual CPUs: 3
%usr %sys %wait %idle physc %entc %lbusy app vcsw phint %hypv hcalls
  59  27   0   13  1.0 324.30 47.54  0.01 11807 9  0.0 0
=====
LCPU minpf majpf intr  csw icsw runq lpa scalls usr sys wt idl pc  lcsw
Cpu0  0  0  824  224 220  1 100 25698 73 25 0  2 0.23 2043
Cpu1  0  0  785 2265 1137  1 100  3270 54 21 0 25 0.17 2089
Cpu2  0  0  985  509 337  0 100  4392 56 30 0 14 0.16 2099
Cpu3  0  0  728  943 568  1 100  3325 56 33 0 11 0.18 2064
Cpu4  0  0  777  230 223  0 100 10586 69 29 0  2 0.19 1725
Cpu5  0  0  635  41  26  1 100  43  0 29 0 70 0.05 1787
```

8.1.7 xmpperf command

Table 8-3 on page 265 shows a summary for the **xmpperf** command.

Table 8-8 *xmpperf* command summary

Command name	xmpperf
Interface type	GUI
Updated or new command	update
AIX package	perfmgr.network, which is part of the Performance Tool Box
Measurement Processor Memory Disk I/O Network POWER Hypervisor	 + + + +
Environment Dedicated partition Micro-partition Simultaneous multithreading Virtual I/O Server	 + + +

Usage

```
xmpperf [-vauxz] [-h hostname]
```

Most important flags

- h** Local host name (host to be the local host).
- o** Configuration file, default is /\$HOME/xmpperf.cf.

Output examples

Figure 8-2 on page 279 shows the standard output of **xmpperf** command, called Local System Monitor. This console has eight monitors showing information about CPU usage, disk access, network traffic, paging space occupancy, memory, and process activity.

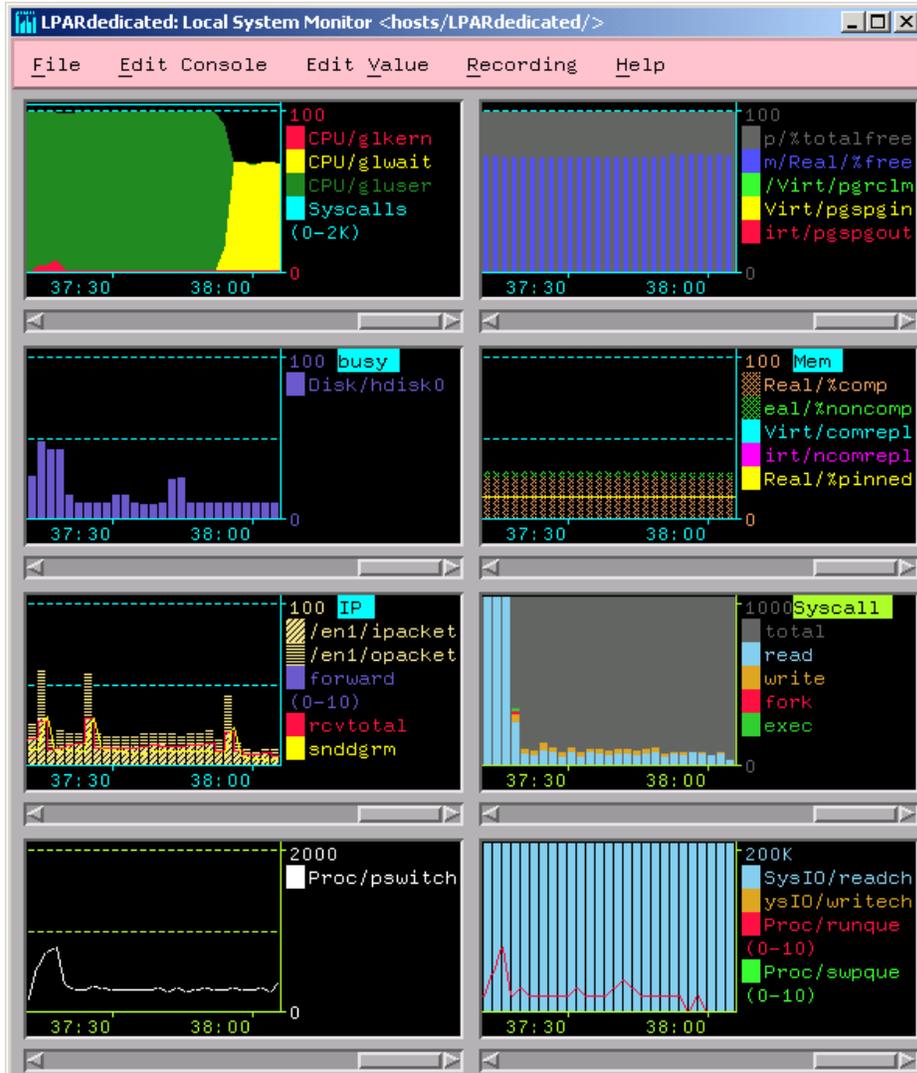


Figure 8-2 *xmperf* - Local System Monitor

System activity and processor utilization

In the following example, *xmperf* is used to show system activity and processor usage. One logical uncapped micro-partition is created with three logical processors, and the system has a pool size of two processors.

Figure 8-3 on page 281 shows system activity and CPU usage with several CPU consuming tasks started in the partition. The first three diagrams show activity on

processors `cpu0`, `cpu1`, and `cpu2`. The last part of the figure shows the whole system activity (`user`) and the number of physical processor consumed (`physc`).

Three tasks are launched one after the other and then a partition is started:

1. The first task starts at 00:30 on `cpu1`. At this moment the system activity increases to 100% and because only one mono thread task is running, only one physical processor is consumed.
2. The second task starts at 01:00 on `cpu0`, the system activity remains at 100% (it is obvious that the system cannot consume more), and almost a second physical processor is consumed.
3. The third task starts at 01:30 on `cpu2`, the system activity remains at 100%, and physical processor consumed stays around two because only two processors were available in the shared pool.
4. At 02:00, a logical partition with one dedicated processor is started in the same system. This removes processing units equivalent to one processor from the micro-partition. At this point, three tasks are running in the partition, each one on a distinct logical processor, and only one physical processor is consumed. This means that each task is consuming one-third of a physical processor.

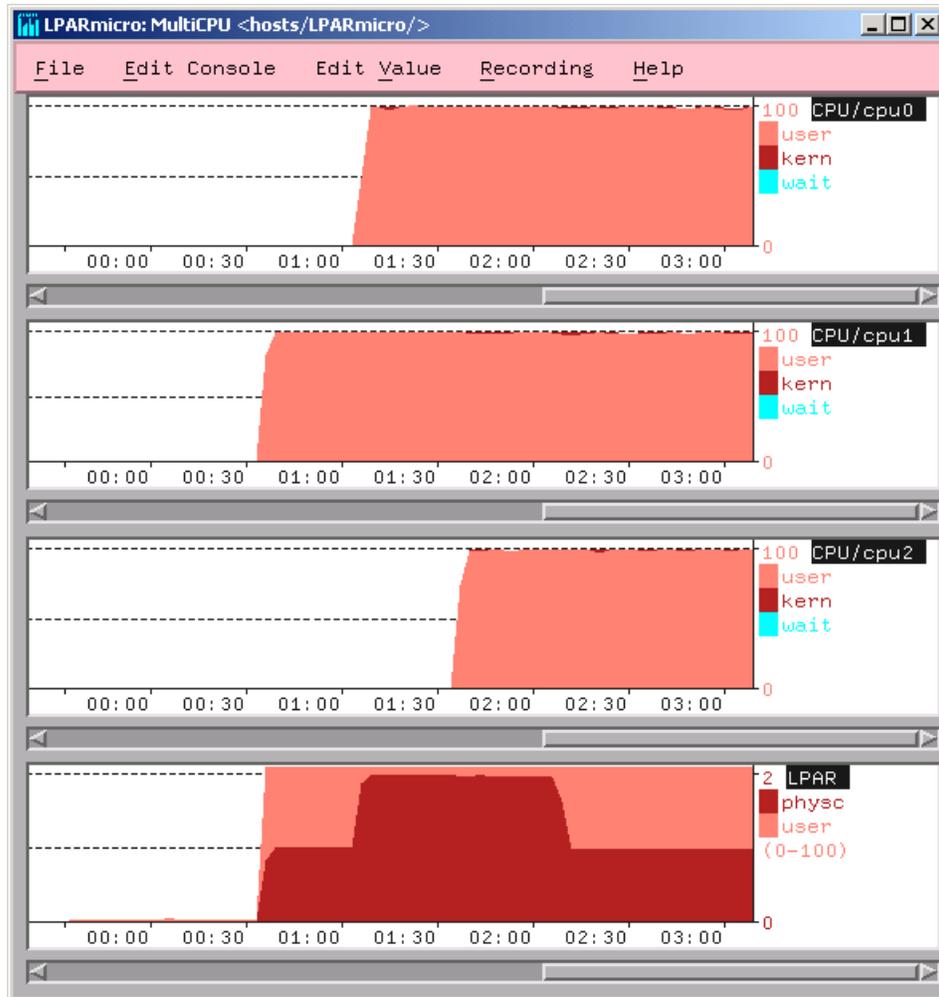


Figure 8-3 Physical and logical processors usage

Redispatch activity

When a process is running on a server, by default it is not bound to a processor, which means that it can run on any available processor. The POWER Hypervisor dispatches virtual processors on physical processors, so a process may not stay on the same processor all its life.

To illustrate this, the following case uses an uncapped micro-partition with three virtual processors and one processor available in the shared pool.

A CPU-intensive job is started around 06:32 as shown in Figure 8-4 on page 282. It is dispatched on cpu0, the partition activity increases to 100% (user)

and it consumes one physical processor (physc). The job is then redispached on cpu1, cpu2, cpu0, cpu1, and cpu0. During all this run, the system is always 100% busy and only one physical processor is used although a processor is available in the pool, and the job is not running on the same processor all the time.

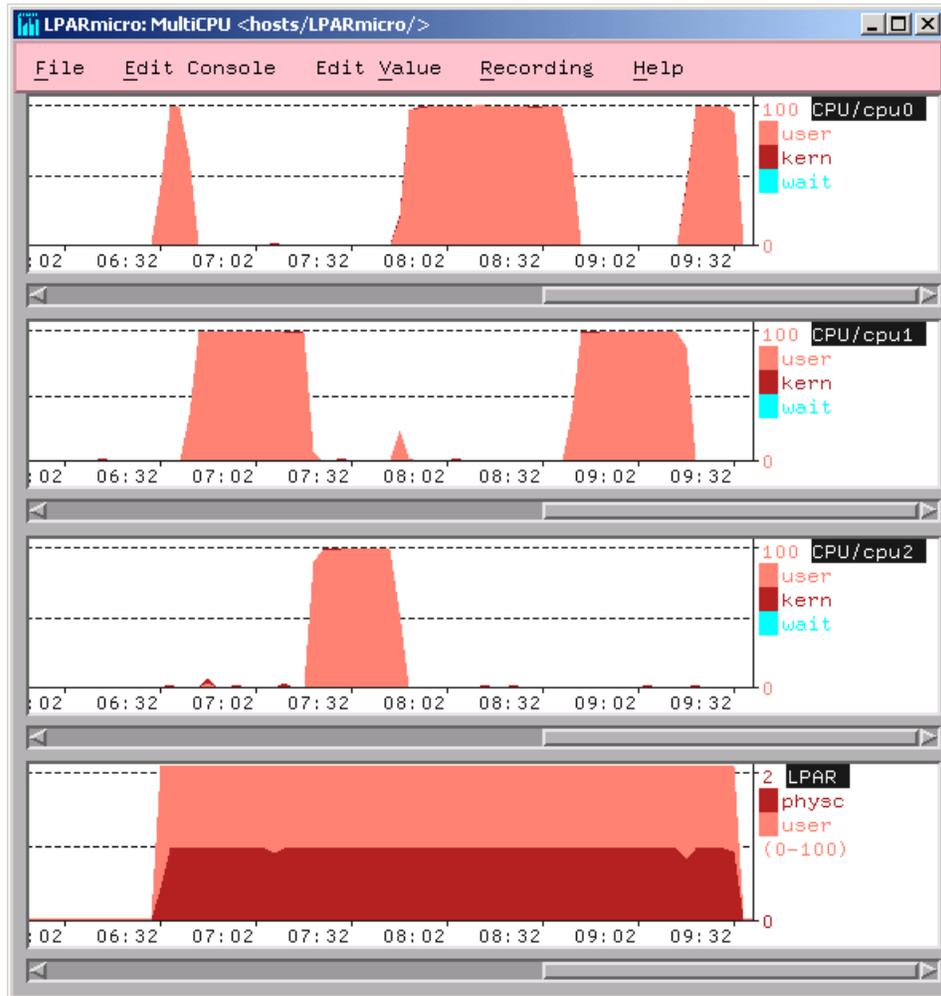


Figure 8-4 xmp perf - process redispach

8.2 Performance tuning approach

A system may experience performance problems for many reasons, including hardware problems, software problems, or human expectations.

In this section we mainly focus on hardware problems linked to the new POWER5 architecture.

Performance analysis and tuning demands great skills, knowledge, experience, and methodology. To determine which of the monitored values are high in a particular environment, it is a good idea to gather data on the system during an optimal performance state. This baseline information is useful for comparison during a performance problem. The `xmperf` command can be used to collect data. Screen shots of the `topas` command also provide a brief overview of all major performance information.

8.2.1 Global performance analysis

To solve performance problems, the investigation to find the root cause will be performed for the following categories:

- ▶ CPU-bound system on page 289
- ▶ Memory-bound system on page 294
- ▶ DISK I/O-bound system on page 296
- ▶ Network I/O-bound system on page 304

Figure 8-5 on page 284 gives the chronological order to follow when trying to identify performance issues: First check the CPU, then the memory, the disk, and finally the network.

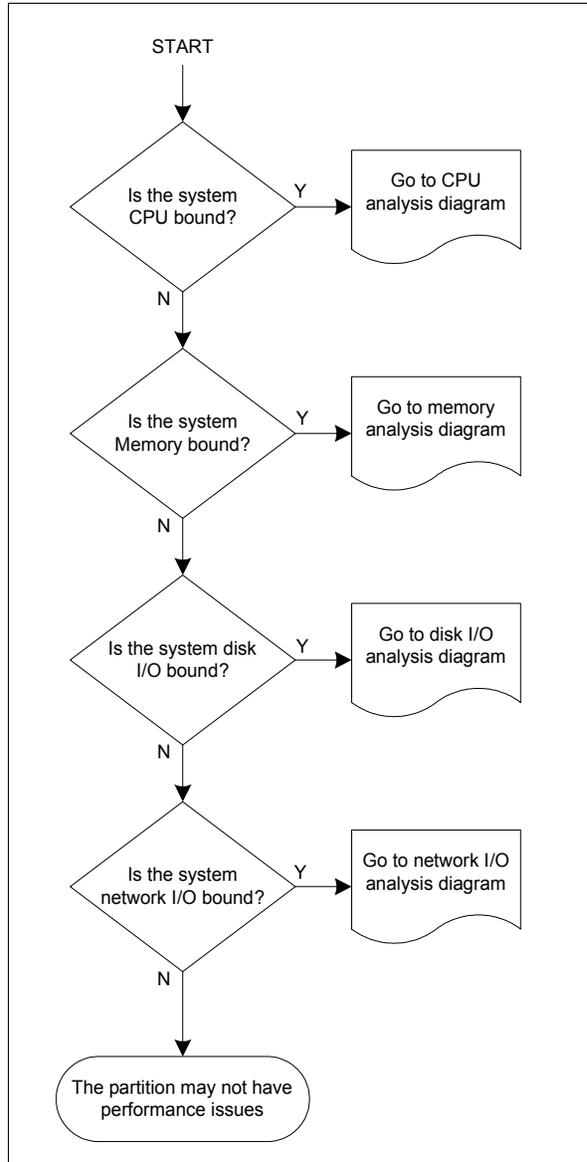


Figure 8-5 Global performance diagram

CPU-bound system

In a system that is CPU-bound, all the processors are 100% busy and some jobs are waiting for CPU in the run queue. A system with 100% busy CPU with a large run queue compared to the number of CPUs and more context switches than usual has a good chance of becoming CPU-bound.

In Example 8-20, the system has two dedicated processors that are 99.8% busy, a run queue of 4 (twice the number of processors), and 5984 context switches (for everyday work, this system usually has around 500). This system is CPU-bound.

Example 8-20 CPU-bound system

Topas Monitor for host: LPARdedicated						EVENTS/QUEUES		FILE/TTY	
Wed Oct 27 15:24:31 2004						Interval: 2			
						Cswitch	5984	Readch	808
						Syscall	12132	Writech	8078.3K
Kernel	0.2	#				Reads	1	Rawin	0
User	99.8	#####				Writes	2035	Ttyout	309
Wait	0.0					Forks	0	Igets	0
Idle	0.0					Execs	0	Namei	5
						Runqueue	4.0	Dirblk	0
						Waitqueue	0.0		
Network	KBPS	I-Pack	O-Pack	KB-In	KB-Out	PAGING			
en1	0.9	13.0	11.0	0.6	1.2	Faults	0	Real,MB	1023
lo0	0.0	0.0	0.0	0.0	0.0	Steals	0	% Comp	26.0
						PgspIn	0	% Noncomp	4.0
Disk	Busy%	KBPS	TPS	KB-Read	KB-Writ	PgspOut	0	% Client	4.8
hdisk0	0.0	0.0	0.0	0.0	0.0	PageIn	0		
						PageOut	0	PAGING SPACE	
Name	PID	CPU%	PgSp	Owner	Sios				
vpross	278618	25.0	15.4	root	0		Size,MB	512	
yes	291018	25.0	0.1	root	0		% Used	0.8	
ksh	286946	25.0	0.5	root	NFS (calls/sec)				
ksh	241918	25.0	0.5	root	0		% Free	99.1	
topas	266278	0.0	1.1	root	ServerV2				
snmpibd6	155772	0.0	0.7	root	0		Press:		
xmhc	45078	0.0	0.0	root	0		"h" for help		
netm	49176	0.0	0.0	root	0		"q" to quit		
IBM.CSMAg	262280	0.0	2.2	root	ClientV3				
getty	258174	0.0	0.4	root					
gil	53274	0.0	0.1	root					
aixmibd	139414	0.0	0.6	root					
syncd	65710	0.0	0.5	root					
rpc.lockd	209122	0.0	0.2	root					
nfds	196776	0.0	0.2	root					
lvmbb	86076	0.0	0.0	root					
dog	90184	0.0	0.1	root					

Memory-bound system

System memory includes real memory and paging space. AIX 5L uses the Virtual Memory Manager (VMM) to control real memory and paging space on the system. The VMM maintains a list of free memory pages, and a page replacement algorithm is used to determine which pages.

A memory-bound system has high memory occupancy and high paging space activity. The activity of the paging space is given by the number of pages read from disk to memory (page in) and number of pages written to disk (page out).

The amount of used memory and paging space activity can be obtained with the **topas** command. In Example 8-21, the memory is 100% consumed (Comp, Noncomp), paging space is 61% consumed (% Used), a lot of pages are written to disk (PgspOut), and the system needs real memory, which is why VMM steals pages (Steals). Because the system is using all of the memory and asking for more, this partition is memory-bound.

Example 8-21 Memory-bound system

```

Topas Monitor for host:  LPARdedicated      EVENTS/QUEUES  FILE/TTY
Wed Oct 27 18:19:37 2004  Interval: 2      Cswitch      998 Readch      12122
                               Syscall      406 Writch      290
Kernel  4.1  |##                               | Reads       17 Rawin       0
User    64.1 |#####                               | Writes      0 Ttyout      290
Wait    4.0  |##                               | Forks       0 Igets       0
Idle    27.8 |#####                               | Execs       0 Namei      12
                               Runqueue    1.0 Dirblk      0
Network KBPS  I-Pack  O-Pack  KB-In  KB-Out  Waitqueue  1.0
en1     0.4    3.0    1.0    0.1    0.6
lo0     0.0    0.0    0.0    0.0    0.0
Disk    Busy%    KBPS    TPS  KB-Read  KB-Writ  Steals  4963 % Comp  100.3
hdisk0  19.8  19863.0  419.9  444.0  39580.0 PgspIn   32 % Noncomp  0.5
                               PgspOut  4946 % Client  0.6
Name     PID  CPU%  PgSp  Owner  PageIn   56
perl     307230  25.0  1102.5  root  PageOut  4946
lrud     20490   0.0   0.1  root  Sios     5016
telnetd  245904   0.0   0.2  root  NFS (calls/sec) % Used  61.5
topas    266340  0.0   1.1  root  ServerV2  0
rgsr     69756   0.0   0.0  root  ClientV2  0  Press:
rmcd     229496   0.0   1.4  root  ServerV3  0  "h" for help
init     1     0.0   0.6  root  ClientV3  0  "q" to quit
netm     49176   0.0   0.0  root
IBM.CSMAg  262280  0.0   2.2  root
getty    258174   0.0   0.4  root
gil      53274   0.0   0.1  root
aixmibd  139414   0.0   0.6  root

```

Disk I/O-bound system

This system has at least one busy disk, it cannot fulfill other requests, and processes are blocked and waiting for the I/O operation to complete. Limitation can be either physical or logical. Physical limitation involves hardware, such as bandwidth of disks, adapters, and system bus. Logical limitations involve the organization of the logical volumes on disks and LVM tunings and settings, such as striping or mirroring. Example 8-22 shows a system with high-wait I/O at 86.6% (Wait), percentage of time that hdisk0 was active at 98.7% (Busy%) and over five processes waiting for paging space operations to complete (Waitqueue). This system is waiting for write operations on hdisk0, so it is disk I/O-bound.

Example 8-22 Disk I/O-bound system

Topas Monitor for host:		LPARdedicated		EVENTS/QUEUES		FILE/TTY	
Thu Oct 28 11:03:43 2004		Interval: 2		Cswitch	678	Readch	0
				Syscall	97	Writech	317
Kernel	1.1	#		Reads	0	Rawin	0
User	12.4	####		Writes	0	Ttyout	317
Wait	86.6	#####		Forks	0	Igets	0
Idle	0.0			Execs	0	Namei	0
				Runqueue	2.5	Dirblk	0
Network	KBPS	I-Pack	O-Pack	KB-In	KB-Out	Waitqueue	5.6
en1	0.5	3.0	1.0	0.3	0.7		
lo0	0.0	0.0	0.0	0.0	0.0	PAGING	MEMORY
Disk	Busy%	KBPS	TPS	KB-Read	KB-Writ	Faults	826 Real,MB 1023
hdisk0	98.7	2645.1	185.3	4.0	5220.0	Steals	0 % Comp 99.7
						PgspIn	0 % Noncomp 1.2
						PgspOut	652 % Client 0.5
Name	PID	CPU%	PgSp	Owner		PageIn	0
perl	291012	0.0	983.2	root		PageOut	652 PAGING SPACE
topas	266340	0.0	1.1	root		Sios	588 Size,MB 512
getty	258174	0.0	0.4	root			% Used 47.0
lrud	20490	0.0	0.1	root		NFS (calls/sec)	% Free 52.9
netm	49176	0.0	0.0	root		ServerV2	0
vpross	303230	0.0	15.4	root		ClientV2	0 Press:
IBM.CSMAG	262280	0.0	2.2	root		ServerV3	0 "h" for help
gil	53274	0.0	0.1	root		ClientV3	0 "q" to quit
syncd	65710	0.0	0.5	root			
aixmibd	139414	0.0	0.6	root			
rpc.lockd	209122	0.0	0.2	root			
nfstd	196776	0.0	0.2	root			
rgsr	69756	0.0	0.0	root			
errdemon	73858	0.0	0.5	root			
j2pg	77868	0.0	0.2	root			
lvmbb	86076	0.0	0.0	root			
dog	90184	0.0	0.1	root			
hostmibd	94238	0.0	0.4	root			

Network I/O-bound system

In a system that is network I/O-bound, the bandwidth of at least one network adapter is totally (or almost totally) used. Processes that need to send or receive data must wait for other processes' I/O to complete.

Example 8-23 shows a system using all of the bandwidth of its network adapter and having some wait I/O. The maximum bandwidth of the network adapter depends on its type.

Example 8-23 Network I/O-bound system

Topas Monitor for host:		LPARdedicated		EVENTS/QUEUES		FILE/TTY	
Fri Oct 29 14:36:34 2004		Interval: 2		Cswitch	1636	Readch	11.6M
				Syscall	422	Writech	11.6M
Kernel	11.0	####		Reads	185	Rawin	0
User	0.1	#		Writes	186	Ttyout	267
Wait	13.8	#####		Forks	0	Igets	0
Idle	75.1	#####		Execs	0	Namei	0
				Runqueue	1.0	Dirblk	0
Network	KBPS	I-Pack	O-Pack	KB-In	KB-Out	Waitqueue	0.0
en1	12285.3	10513.0	16175.0	472.3	23914.1		
lo0	0.0	0.0	0.0	0.0	0.0		
				PAGING		MEMORY	
Disk	Busy%	KBPS	TPS	KB-Read	KB-Writ	Faults	0 Real,MB 1023
hdisk0	0.0	0.0	0.0	0.0	0.0	Steals	0 % Comp 23.5
						PgspIn	0 % Noncomp 33.8
						PgspOut	0 % Client 34.4
Name	PID	CPU%	PgSp	Owner	PageIn	PageOut	0
ftpd	266380	0.0	0.8	root	PageOut	0	PAGING SPACE
topas	278692	0.0	1.1	root	Sios	0	Size,MB 512
getty	245882	0.0	0.4	root			% Used 0.8
aixmibd	204912	0.0	0.6	root	NFS (calls/sec)	% Free	99.1
rpc.lockd	159988	0.0	0.2	root	ServerV2	0	
nfsd	188590	0.0	0.2	root	ClientV2	0	Press:
xmgc	45078	0.0	0.0	root	ServerV3	0	"h" for help
netm	49176	0.0	0.0	root	ClientV3	0	"q" to quit
IBM.CSMAG	241790	0.0	2.0	root			
gil	53274	0.0	0.1	root			
ftpd	250008	0.0	0.8	root			
syncd	65710	0.0	0.5	root			
rgsr	69758	0.0	0.0	root			
errdemon	73858	0.0	0.5	root			
j2pg	77866	0.0	0.2	root			

8.2.2 CPU analysis

Now that we know how to recognize a CPU-bound system, the flow chart in Figure 8-6 helps determine the root cause for this activity.

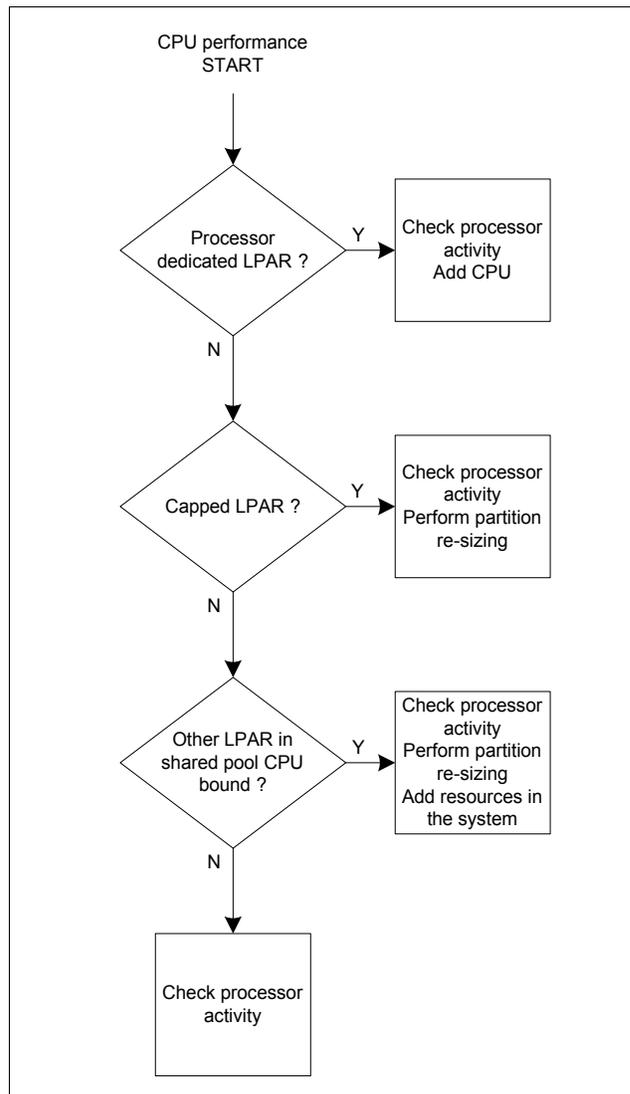


Figure 8-6 CPU analysis diagram

Identify CPU consumers

To understand why the partition is CPU-bound, we have to find the processes that consume the most CPU using the `ps` command, as shown in Example 8-24.

The `%CPU` column gives the percentage of time the process has used the CPU since the process started. In this example, three processes are using the CPU: `./vpross`, `yes`, and `./loop`. You must verify whether those processes are running correctly and if they are using the usual amount of CPU.

Example 8-24 ps - most CPU consumers

```
# ps aux|more
USER          PID %CPU %MEM    SZ   RSS   TTY STAT      STIME   TIME  COMMAND
root         295058 49.7   5.0 15952 15980 pts/1 A    08:34:25 66:04 ./vpross
root         139278 20.0   0.0   152   156 pts/0 A    09:40:46  0:02 yes
root         262232 14.3   0.0   676   712 pts/0 A    10:12:03  0:04 ./loop
root         241790  0.2   1.0  2856  2548    - A    11:26:34  4:08 /usr/sbin/rsct/b
root         258174  0.0   0.0   464   488    - A    11:26:29  0:16 /usr/sbin/getty
root          53274  0.0   0.0   116   116    - A    11:24:25  0:12 gil
root         155888  0.0   0.0   960   960    - A    11:26:15  0:04 /usr/sbin/aixmib
root          65710  0.0   0.0   500   508    - A    11:25:41  0:04 /usr/sbin/syncd
root         172270  0.0   0.0   200   200    - A    11:26:22  0:02 rpc.lockd
root         209028  0.0   0.0   200   200    - A    11:26:21  0:02 nfsd
root         196726  0.0   0.0   444   464    - A    11:26:03  0:00 /usr/sbin/inetd
root         200844  0.0   0.0  1692  1296    - A    11:25:57  0:00 sendmail: accept
...
```

Dedicated LPAR

If applications really need processing power, then you may dynamically add a processor (if any are available in the system) to the partition. To determine the available processors in the system, connect to the HMC, edit the system properties, and click the **Processor** tab. For example, in Figure 8-7 on page 291, only 0.3 processors are available. A list of each partition with its amount of processors used is given in the bottom of the figure.

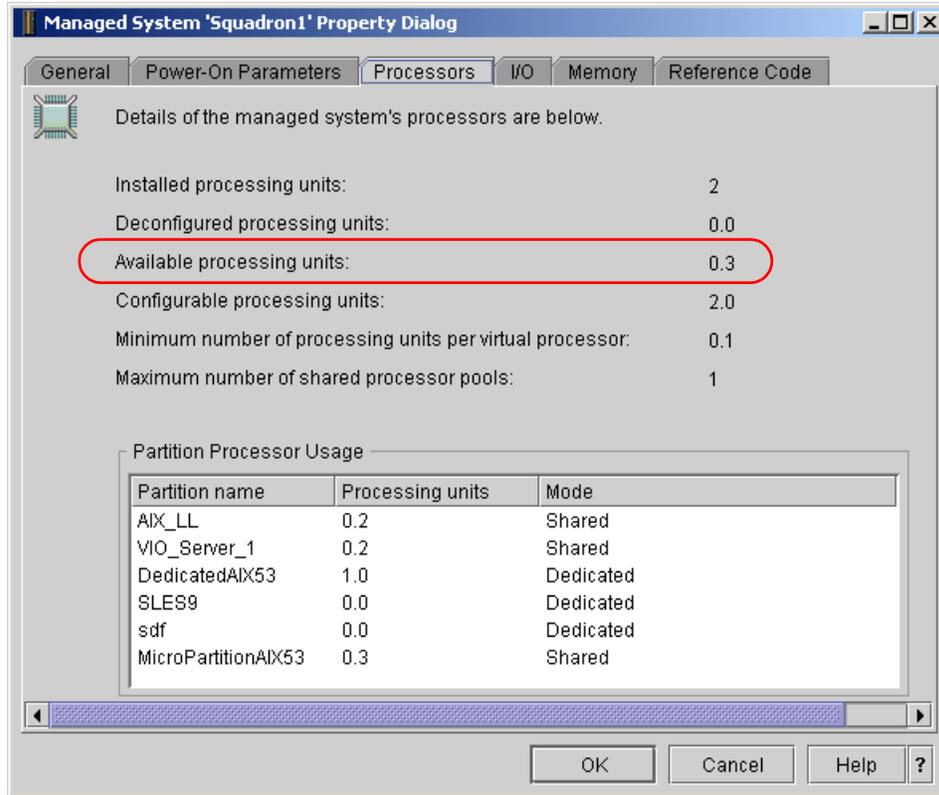


Figure 8-7 System properties: Processors tab

If no processors are available, check the CPU usage of all other partitions in the system to see whether you can free an unused processor. Then if all of the processor resources are used, upgrade the system with new processors.

Capped LPAR

If an application really needs processing power and there is an available processing unit in the system, you can increase the entitlement of the partition. Example 8-25 on page 292 shows a capped partition with one logical processor, an entitlement of 0.5 (ent) and 1.5 available processor pool size (app). Only one process (testp) is requesting processing power and it consumes half a processor (pc). In this case we can increase the entitlement up to 1.0, and the processes will run faster.

Example 8-25 Entitlement limited partition

```
# lparstat 2 2
```

```
System configuration: type=Shared mode=Capped smt=0ff lcpu=1 mem=512 psize=2
ent=0.50
```

%user	%sys	%wait	%idle	physc	%entc	lbusy	app	vcswh	phint
99.3	0.7	0.0	0.0	0.50	100.0	100.0	1.49	100	0
98.6	0.9	0.0	0.5	0.50	99.5	100.0	1.49	100	0

```
# vmstat 2 3
```

```
System configuration: lcpu=1 mem=512MB ent=0.5
```

kthr		memory				page				faults				cpu				
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa	pc	ec
2	0	49165	72012	0	0	0	0	0	0	2	3079	142	99	1	0	0	0.50	100.2
2	0	49167	72010	0	0	0	0	0	0	3	3031	142	99	1	0	0	0.50	99.8
2	0	49167	72010	0	0	0	0	0	0	3	3030	142	99	1	0	0	0.50	100.0

```
# ps aux|more
```

USER	PID	%CPU	%MEM	SZ	RSS	TTY	STAT	STIME	TIME	COMMAND
root	254108	43.8	0.0	152	156	pts/0	A	14:36:12	1:18	testp
root	172124	0.0	0.0	940	940	-	A	14:29:38	0:00	/usr/sbin/aixmib
root	168030	0.0	0.0	532	504	-	A	14:29:32	0:00	/usr/sbin/hostmi
root	163920	0.0	1.0	1156	1192	-	A	14:29:29	0:00	/usr/sbin/snmpd
root	184422	0.0	0.0	676	700	-	A	14:34:17	0:00	telnetd -a
root	180312	0.0	0.0	664	688	-	A	14:29:41	0:00	/usr/sbin/muxatm
root	176218	0.0	0.0	1048	916	-	A	14:29:35	0:00	/usr/sbin/snmpmi
...										

If many processes request processing power, you can increase the number of virtual processors (and the entitlement). Example 8-26 on page 293 shows a capped partition with one logical processor, an entitlement of 1.0 (ent) and 0.99 available processor pool size (app). Two processes are requesting processing power: testp and loop. They consume one physical processor (physc). Because two processes are running on one physical processor, we can increase the number of virtual processors in order to run each process on a distinct processor.

Example 8-26 Logical processor limited partition

```
# lparstat 2 2
```

```
System configuration: type=Shared mode=Capped smt=Off lcpu=1 mem=512 psize=2
ent=1.00
```

%user	%sys	%wait	%idle	physc	%entc	lbusy	app	vcswh	phint
99.4	0.6	0.0	0.0	1.00	100.1	100.0	0.99	0	1
99.5	0.5	0.0	0.0	1.00	100.0	100.0	0.99	0	0

```
# ps aux|more
```

USER	PID	%CPU	%MEM	SZ	RSS	TTY	STAT	STIME	TIME	COMMAND
root	229502	48.9	0.0	152	156	pts/0	A	16:21:00	0:23	testp
root	241812	37.4	0.0	152	156	pts/0	A	16:00:07	8:06	loop
root	237690	0.2	1.0	1852	1828	-	A	15:55:10	0:01	/usr/sbin/rsct/b
root	172126	0.0	0.0	1048	916	-	A	15:54:50	0:00	/usr/sbin/snmpmi
root	188572	0.0	0.0	200	200	-	A	15:55:01	0:00	rpc.lockd
root	176216	0.0	0.0	940	940	-	A	15:54:54	0:00	/usr/sbin/aixmib
root	180312	0.0	0.0	664	688	-	A	15:54:57	0:00	/usr/sbin/muxatm
root	184422	0.0	0.0	676	700	-	A	15:59:46	0:00	telnetd -a
root	168030	0.0	0.0	532	504	-	A	15:54:47	0:00	/usr/sbin/hostmi
root	147588	0.0	0.0	720	756	pts/0	A	15:59:46	0:00	-ksh
...										

Micro-partition

Verify whether other partitions can give some processing units or processors back to the shared pool. For example, check whether idle processors are allowed to be shared.

To select this option, connect to the HMC, edit the profile properties, click the **Processor** tab, then select the **Allow idle processors to be shared** check box as in Figure 8-8 on page 294. If this option is *not* selected, the unused processors will not be available for other partitions when the partition is stopped, but this also means that your partition is guaranteed to have its processor if needed.

Check profile properties for capped and dedicated partitions to be sure that no CPU resources are allocated but unused.

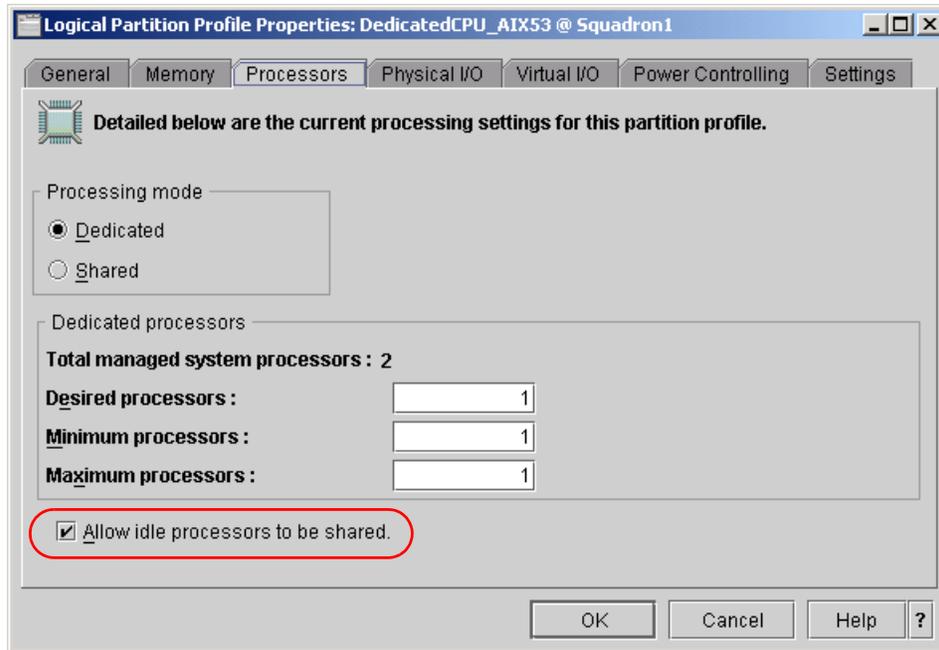


Figure 8-8 Allow idle processors to be shared

In all of the previous cases, if many processes need processors, you may enable simultaneous multithreading. For more information, refer to Chapter 3, “Simultaneous multithreading” on page 41.

8.2.3 Memory analysis

The goal of memory analysis is to determine which processes are making the system memory-bound. Use `ps` or `svmon` to look for processes that are consuming a lot of memory. In Example 8-27 on page 295, `per1` is the largest memory consumer, with a total number of pages in real memory of 218933 (around 875 MB) and total number of pages reserved or used on paging space of 97963 (nearly 400 MB). The second application, `vpross`, uses only 48 MB of memory and less than 7 MB of paging space; therefore the `per1` application is the root cause of this memory problem.

Example 8-27 svmon - process report

svmon -P|more

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
332008	perl	218933	4293	97963	318471	N	N	N
Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
7380	6	work	working storage	-	65536	0	0	65536
1383	3	work	working storage	-	47302	0	18215	65515
15389	7	work	working storage	-	44717	0	0	44717
17388	4	work	working storage	-	38021	0	27528	65536
21393	5	work	working storage	-	15031	0	50528	65536
0	0	work	kernel segment	-	6843	4290	1621	8454
3f8bd	d	work	loader segment	-	1352	0	71	3062
29397	f	work	shared library data	-	83	0	0	83
d385	2	work	process private	-	32	3	0	32
3362	1	clnt	code,/dev/hd2:12435	-	16	0	-	-
2f374	a	work	working storage	-	0	0	0	0
3f37c	9	work	working storage	-	0	0	0	0
3d37d	8	work	working storage	-	0	0	0	0
Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
303122	vpross	12193	4293	1696	15465	N	N	N
Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel segment	-	6843	4290	1621	8454
17368	2	work	process private	-	3913	3	4	3917
3f8bd	d	work	loader segment	-	1352	0	71	3062
1936f	1	pers	code,/dev/lv00:83977	-	53	0	-	-
1136b	f	work	shared library data	-	32	0	0	32
1336a	-	pers	/dev/lv00:83969	-	0	0	-	-
Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
299170	IBM.CSMAgentR	8572	4306	1965	12130	N	Y	N
Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel segment	-	6843	4290	1621	8454
3f8bd	d	work	loader segment	-	1352	0	71	3062
...								

On the Memory tab, determine whether the process is running correctly and actually needs this amount of memory.

- ▶ If so, increase memory. This can be done by adding physical memory, or with a dynamic operation if some memory is available in the system. To determine available memory, connect to the HMC, edit the system properties, and click the **Memory** tab. In Figure 8-9, 688 MB of memory is available in the system.
- ▶ If not, then check, debug, or tune the application.

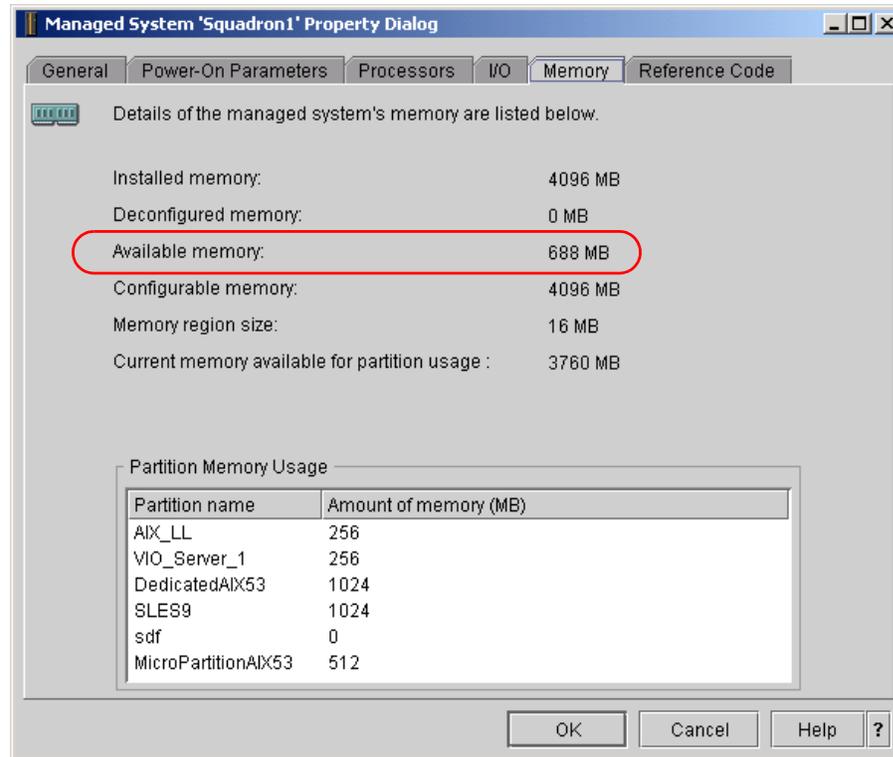


Figure 8-9 System properties: Memory tab

8.2.4 Disk I/O analysis

When a system has been identified as having disk I/O performance problems, you can find the source of the problem as shown in Figure 8-10 on page 297:

- ▶ For a dedicated device:
 - a. Check the dedicated adapter.
 - b. Check the dedicated disk.
- ▶ For a virtual device:
 - a. Check CPU on the Virtual I/O Server.

- b. Check adapter on Virtual I/O Server.
- c. Check disk on Virtual I/O Server.

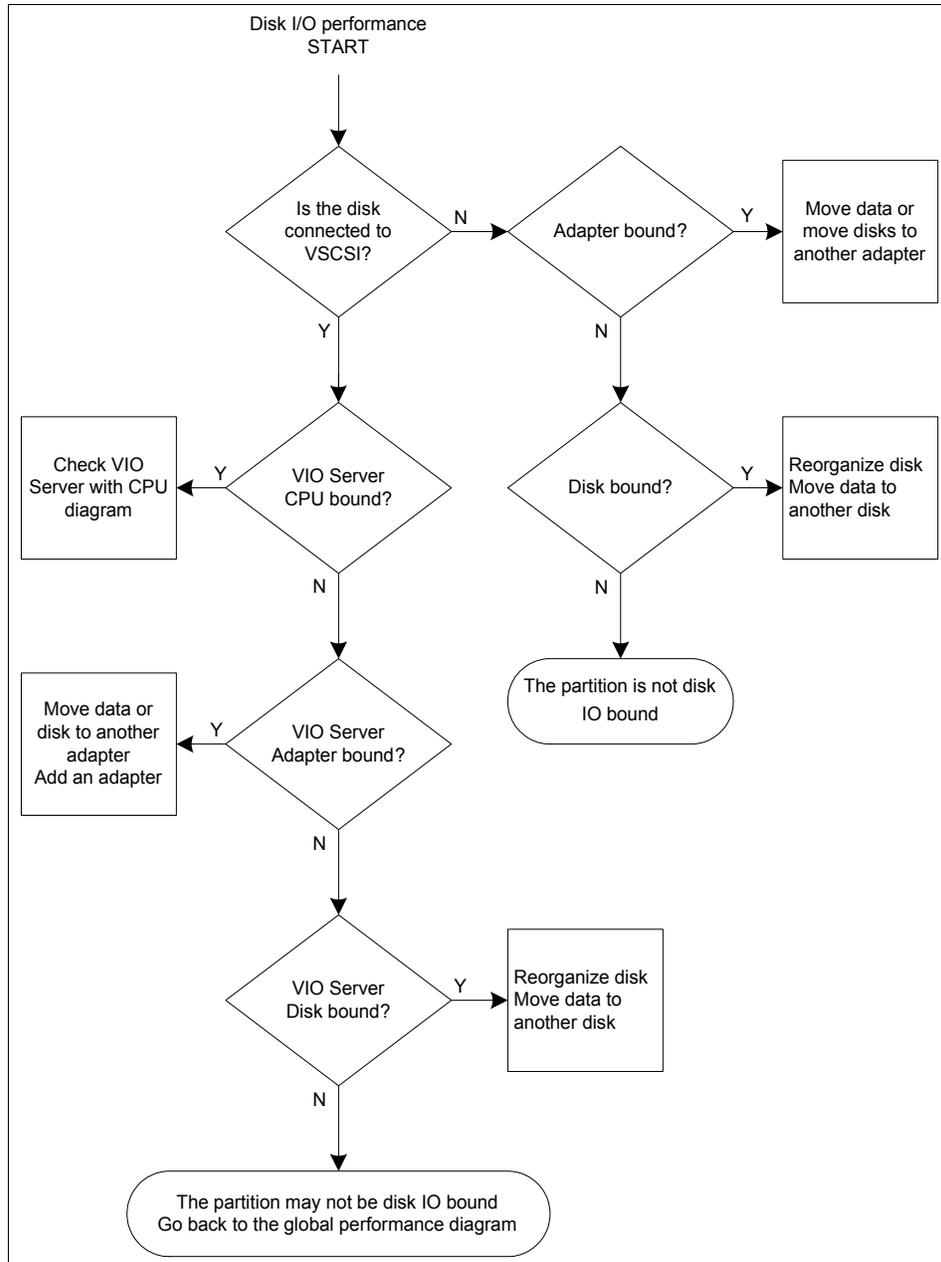


Figure 8-10 Disk I/O analysis diagram

Dedicated or virtual device

First check whether the disk is a physical disk belonging to the partition or a virtual SCSI disk using the **lscfg** command. As shown in Example 8-28, hdisk0, hdisk1, and hdisk2 are physical drives, and hdisk3 is a virtual SCSI drive. The **lsdev** command shows the adapter to which the disk is connected.

Example 8-28 Physical and virtual disks

```
# lsdev -Ccdisk
hdisk0 Available 02-08-00-3,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 02-08-00-4,0 16 Bit LVD SCSI Disk Drive
hdisk2 Available 02-08-00-5,0 16 Bit LVD SCSI Disk Drive
hdisk3 Available Virtual SCSI Disk Drive

# lsdev -Cl hdisk0 -F parent
vscsi0
```

Dedicated device

If a dedicated device (a physical adapter allocated to the partition) is bound, first check the adapter, then the disk.

Dedicated adapter

Use the **iostat -a** command to show the activity of a disk adapter. In Example 8-29, the adapter `siscsia0` has a throughput of 28537 kilobytes per second. Because the maximum bandwidth of an adapter depends on its type and technology, compare the statistics given by **iostat** to the theoretical value to find out the load percentage of the adapter. If the adapter is overloaded, try to move some data to another disk on a distinct adapter, move a physical disk to another adapter, or add a disk adapter.

Example 8-29 iostat - adapter statistics

```
# iostat -a 5

System configuration: lcpu=1 drives=3

tty:      tin      tout  avg-cpu:  % user   % sys    % idle   % iowait
          0.0      36.4          16.5    35.2     26.2     22.1

Adapter:          Kbps    tps    Kb_read  Kb_wrtn
siscsia0          28537.3  132.1    72619    70424

Paths/Disk:      % tm_act  Kbps    tps    Kb_read  Kb_wrtn
hdisk2_Path0     9.6      28537.3  132.1    72619    70424
hdisk0_Path0     0.0      0.0      0.0      0         0
hdisk1_Path0     0.0      0.0      0.0      0         0
```

Dedicated disk

The disk may be bound simply because the data is not well-organized. Verify the placement of logical volumes on the disk with the `lspv` command. If logical volumes are fragmented across the disk as in Example 8-30, reorganize them with `reorgvg` or `migratepv`.

Example 8-30 Fragmented logical volumes

```
# lspv -p hdisk0
hdisk0:
PP RANGE STATE REGION LV NAME TYPE MOUNT POINT
  1-3 used outer edge hd5 boot N/A
  4-13 used outer edge hd6 paging N/A
 14-49 free outer edge
 50-124 used outer edge fslv01 jfs2 /test2
125-125 used outer edge hd10opt jfs2 /opt
126-135 free outer edge
136-154 used outer edge hd10opt jfs2 /opt
155-165 free outer middle
166-166 used outer middle hd6 paging N/A
167-176 free outer middle
177-189 used outer middle hd6 paging N/A
190-199 used outer middle hd2 jfs2 /usr
200-210 used outer middle hd6 paging N/A
211-220 used outer middle hd2 jfs2 /usr
221-282 used outer middle hd6 paging N/A
283-283 used outer middle loglv00 jfslog N/A
284-287 free outer middle
288-291 used outer middle hd1 jfs2 /home
292-301 used outer middle hd3 jfs2 /tmp
302-307 used outer middle hd9var jfs2 /var
308-308 used center hd8 jfs2log N/A
309-316 used center hd4 jfs2 /
317-320 used center hd2 jfs2 /usr
321-330 used center hd10opt jfs2 /opt
331-380 used center hd2 jfs2 /usr
381-390 free center
391-460 used center hd2 jfs2 /usr
461-590 used inner middle hd2 jfs2 /usr
591-591 free inner middle
592-601 used inner middle hd6 paging N/A
602-613 free inner middle
614-688 used inner edge fslv02 jfs2 /test1
689-689 free inner edge
690-700 used inner edge hd6 paging N/A
701-712 free inner edge
713-722 used inner edge hd6 paging N/A
723-767 free inner edge
```

If logical volumes are well-organized in the disks, the problem may come from the file distribution in the file system. The **fileplace** command displays the file organization as shown in Example 8-31. In this case, space efficiency is near 100%, which means that the file has few fragments and they are contiguous. To increase a file system's contiguous free space by reorganizing allocations to be contiguous rather than scattered across the disk, use the **defragfs** command.

Example 8-31 fileplace output

```
# fileplace -lv testFile

File: testFile Size: 304998050 bytes Vol: /dev/fslv00
Blk Size: 4096 Frag Size: 4096 Nfrags: 74463
Inode: 4 Mode: -rw-r--r-- Owner: root Group: system

Logical Extent
-----
00000064-00000511          448 frags      1835008 Bytes,   0.6%
00003328-00028511      25184 frags     103153664 Bytes, 33.8%
00054176-00077759     23584 frags     96600064 Bytes, 31.7%
00000048-00000063         16 frags         65536 Bytes,   0.0%
00003312-00003327         16 frags         65536 Bytes,   0.0%
00032760-00032767          8 frags          32768 Bytes,   0.0%
00077760-00077767          8 frags          32768 Bytes,   0.0%
00000044-00000047          4 frags          16384 Bytes,   0.0%
00003296-00003308         13 frags          53248 Bytes,   0.0%
00028512-00032735      4224 frags     17301504 Bytes,  5.7%
00032768-00053725     20958 frags     85843968 Bytes, 28.1%

74463 frags over space of 77724 frags:   space efficiency = 95.8%
11 extents out of 74463 possible:   sequentiality = 100.0%
```

For more about disk performance, refer to the redbook *AIX 5L Performance Tools Handbook*, SG24-6039.

Virtual device

If the bound device is a virtual SCSI disk, refer to 8.2.2, “CPU analysis” on page 289 to check CPU activity on the Virtual I/O Server.

The following steps describe how to find a physical volume hosting a virtual disk allocated to a partition:

1. Get the slot number of the virtual SCSI adapter for the partition as shown in Example 8-32 on page 301.

Example 8-32 Virtual SCSI adapter slot number

```
# lsdev -Cl hdisk0 -F parent
vscsi0

# lscfg -vI vscsi0
vscsi0          U9111.520.10DDEDC-V4-C20-T1  Virtual SCSI Client Adapter

Device Specific.(YL).....U9111.520.10DDEDC-V4-C20-T1
```

2. Check the partition profile on the HMC and collect the slot number of the Virtual I/O Server that is associated with the slot number found previously in the partition. To do this, connect to the HMC, edit the profile properties, click the **Virtual I/O** tab, select the Client SCSI line, and click **(Properties...)** as shown in Figure 8-11.

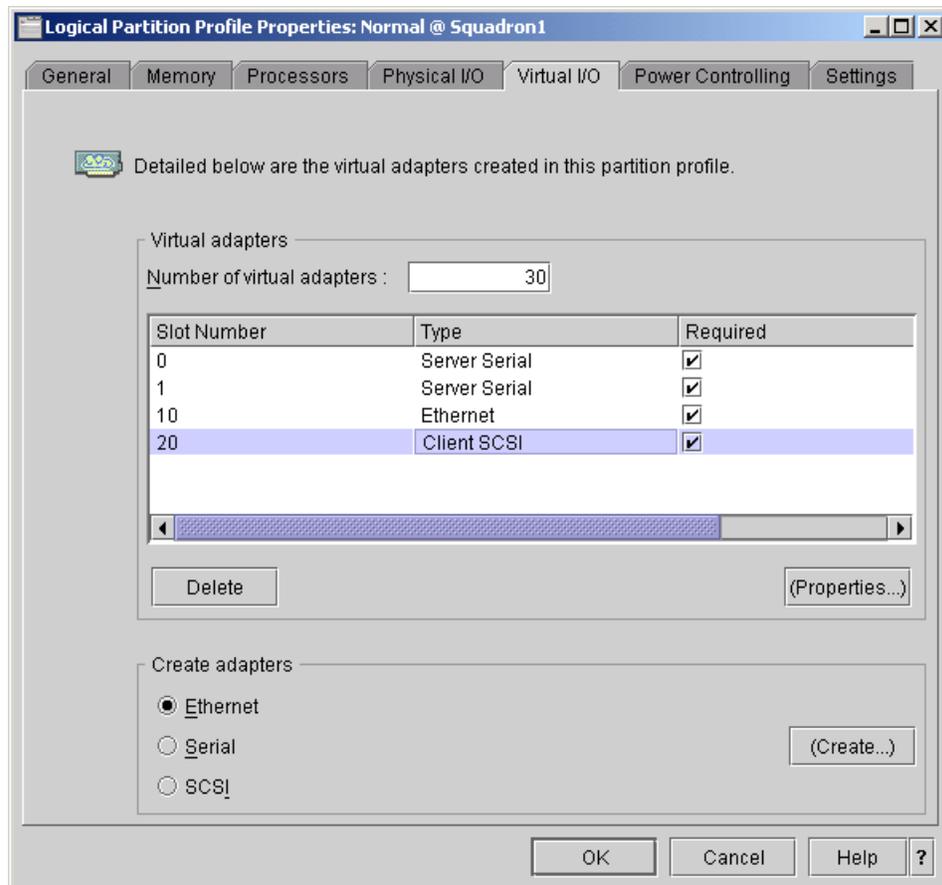


Figure 8-11 Virtual I/O adapters

A new window displays the virtual SCSI adapter properties with slot numbers (Figure 8-12).

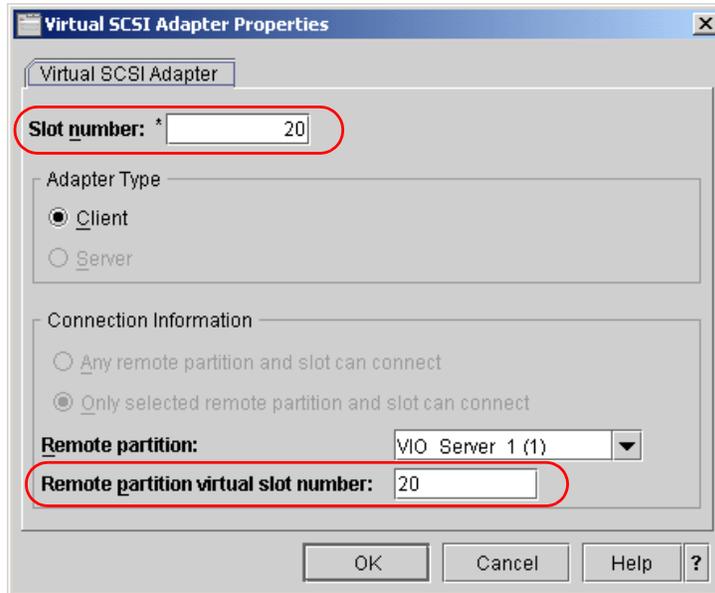


Figure 8-12 Virtual SCSI Adapter Properties

3. Find the name of the disk that contains the partition's data as shown in Example 8-33 on page 303 by following these steps:
 - a. Find the virtual SCSI server adapter with the **1sdev** command.
 - b. Find the logical volume name with the **1smap** command.
 - c. Find the volume group name with the **1s1v** command.
 - d. Find the disk name with the **1svg** command.

Example 8-33 Virtual I/O Server commands to find a disk

```
$ lsdev -vpd|grep vhost.*C20
vhost2          U9111.520.10DDEDC-V1-C20          Virtual SCSI Server Adapter

$ lsmmap -vadapter vhost2
SVSA          Physloc          Client Partition
ID
-----
vhost2        U9111.520.10DDEDC-V1-C20          0x00000004

VTD          vMicroPartAIX53
LUN          0x8100000000000000
Backing device MicroPartAIX53
Physloc

$ lslv MicroPartAIX53
LOGICAL VOLUME:  MicroPartAIX53          VOLUME GROUP:  rootvg_clients
LV IDENTIFIER:   00cddedc00004c000000000102a1f53e.4  PERMISSION:
read/write
VG STATE:        active/complete          LV STATE:       opened/syncd
TYPE:            jfs                      WRITE VERIFY:   off
MAX LPs:         32512                   PP SIZE:        32 megabyte(s)
COPIES:          1                       SCHED POLICY:   parallel
LPs:             96                       PPs:            96
STALE PPs:       0                       BB POLICY:      non-relocatable
INTER-POLICY:    minimum                   RELOCATABLE:    yes
INTRA-POLICY:    middle                   UPPER BOUND:    1024
MOUNT POINT:     N/A                      LABEL:          None
MIRROR WRITE CONSISTENCY: on/ACTIVE
EACH LP COPY ON A SEPARATE PV ?: yes
Serialize IO ?:  NO
DEVICESUBTYPE : DS_LVZ

$ lsvg -pv rootvg_clients
rootvg_clients:
PV_NAME      PV STATE      TOTAL PPs   FREE PPs   FREE DISTRIBUTION
hdisk1      active        1082        26         00..00..00..00..26
```

Virtual adapter

If the Virtual I/O Server is not CPU-bound, check the adapter activity. If many disks experience performance problems on the same adapter, it may be overloaded. In that case, move some of the data to another disk on a different adapter (if any are available) or add a physical adapter.

Virtual disk

If only one disk has a performance problem, verify the placement of logical volumes on the disk on both sides on the Virtual I/O Server as shown in Example 8-34 and on the partition using this virtual SCSI disk. If logical volumes are fragmented across the disk, reorganize them with **reorgvg** or **migratepv**. For more about virtual disks, refer to 6.8, “Virtual SCSI” on page 205.

Example 8-34 lspv - logical volume placement

```
$ lspv -pv hdisk1
hdisk1:
PP RANGE STATE REGION LV NAME TYPE MOUNT POINT
 1-25 used outer edge rootvg_11 jfs N/A
 26-121 used outer edge MicroPartAIX53 jfs N/A
122-217 used outer edge rootvg_11 jfs N/A
218-433 used outer middle rootvg_aix53 jfs N/A
434-537 used center rootvg_aix53 jfs N/A
538-649 used center rootvg_sles9 jfs N/A
650-857 used inner middle rootvg_sles9 jfs N/A
858-865 used inner middle fs1_11 jfs N/A
866-985 used inner edge fs1_11 jfs N/A
986-1056 used inner edge rootvg_11 jfs N/A
1057-1082 free inner edge
```

For more about disk performance refer to the redbook *AIX 5L Performance Tools Handbook*, SG24-6039.

8.2.5 Network I/O analysis

When a system has been identified as having network I/O performance problems, the next point is to find where the problem comes from. Figure 8-13 on page 305 shows the steps to follow on a disk I/O-bound system:

1. For a dedicated adapter, check the dedicated adapter statistics.
2. For a virtual adapter:
 - a. Virtual Ethernet adapter:
 - i. Check CPU utilization.
 - ii. Check physical adapter.
 - b. For a Shared Ethernet Adapter, check the adapter statistics.

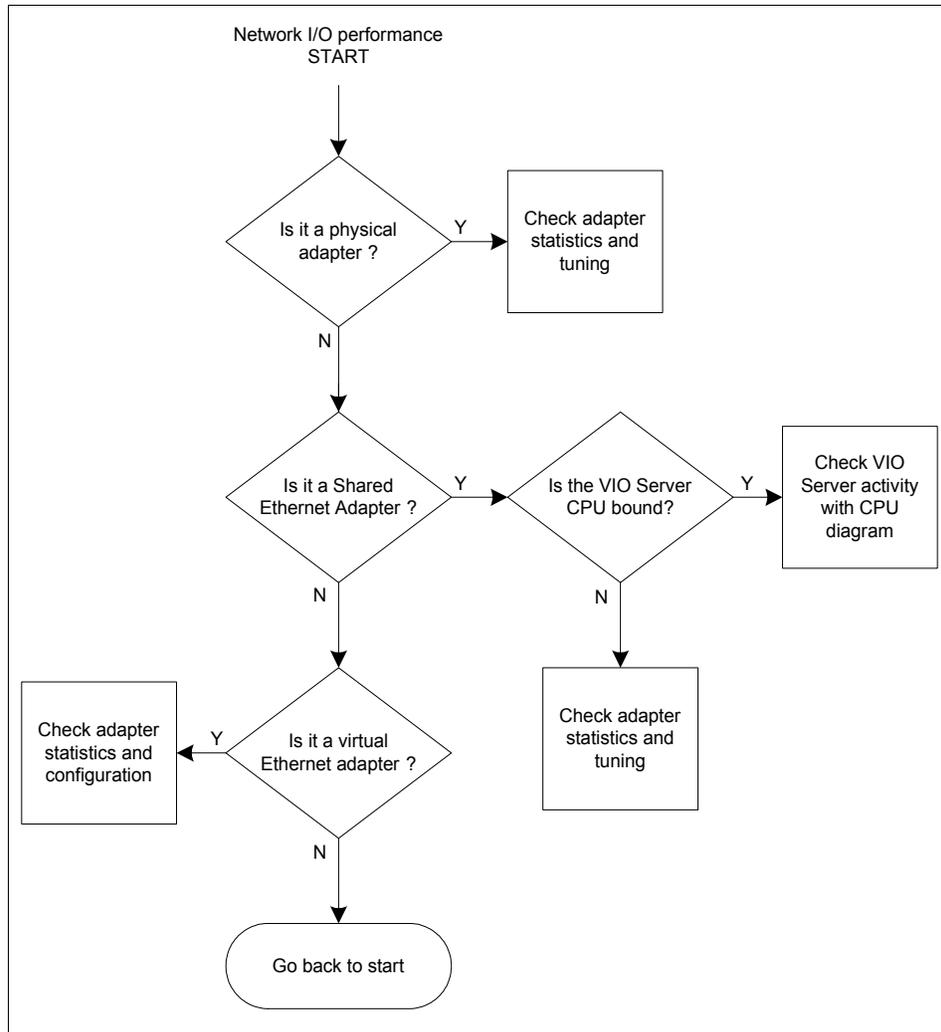


Figure 8-13 Network I/O analysis diagram

Dedicated adapter

If the system is network I/O-bound because of a dedicated adapter, check it with the `netstat` and `entstat` commands, and modify the configuration with `no` and `chdev`. If the partition is using NFS, check the statistics with `nfsstat`.

For more details about network performance, refer to the redbook *AIX 5L Performance Tools Handbook*, SG24-6039.

Virtual adapter

A virtual adapter is provided by the Virtual I/O Server. It can be a virtual Ethernet adapter or a Shared Ethernet Adapter. For more about the Virtual I/O Server, refer to Chapter 6, “Virtual I/O” on page 143.

Virtual Ethernet adapter

If a virtual Ethernet adapter is bound, check the adapter statistics with **entstat**. Check adapter memory use with **netstat** to validate that there is enough buffer allocated to this adapter and confirm that the system is not memory-bound.

Verify the adapter configuration with **lsattr** as in Example 8-35. If the adapter is only for communication between partitions using the same VLAN (no traffic going outside the system), then the mtu can be increased to 64000.

Example 8-35 Network adapter parameters

```
# lsattr -El en1
alias4                IPv4 Alias including Subnet Mask          True
alias6                IPv6 Alias including Prefix Length        True
arp                   on    Address Resolution Protocol (ARP)         True
authority             Authorized Users                          True
broadcast             Broadcast Address                         True
mtu                   1500  Maximum IP Packet Size for This Device    True
netaddr               9.3.5.150 Internet Address                          True
netaddr6              IPv6 Internet Address                    True
netmask               255.255.0.0 Subnet Mask                              True
prefixlen             Prefix Length for IPv6 Internet Address   True
remmtu                576   Maximum IP Packet Size for REMOTE Networks True
rfc1323               Enable/Disable TCP RFC 1323 Window Scaling True
security              none   Security Level                            True
state                 up    Current Interface Status                 True
tcp_mssdf1t           Set TCP Maximum Segment Size             True
tcp_nodelay           Enable/Disable TCP_NODELAY Option         True
tcp_recvspace         Set Socket Buffer Space for Receiving     True
tcp_sendspace         Set Socket Buffer Space for Sending       True
```

For more about VLAN, refer to 6.5, “Virtual Ethernet” on page 164.

Shared Ethernet Adapter

If the system is network I/O-bound because of a Shared Ethernet Adapter, check it on the Virtual I/O Server with **netstat**, **entstat**, and modify the configuration with **chdev**. Example 8-36 on page 307 shows Ethernet adapter statistics with many errors and collisions reported.

Example 8-36 entstat - Virtual I/O Server

```
$ entstat en3
```

```
-----  
ETHERNET STATISTICS (en3) :  
Device Type: Shared Ethernet Adapter  
Hardware Address: 00:09:6b:6b:05:b1  
Elapsed Time: 0 days 0 hours 0 minutes 0 seconds
```

```
Transmit Statistics:
```

```
-----  
Packets: 5656592  
Bytes: 7666680307  
Interrupts: 0  
Transmit Errors: 359712  
Packets Dropped: 0
```

```
Receive Statistics:
```

```
-----  
Packets: 4189578  
Bytes: 365071314  
Interrupts: 3841545  
Receive Errors: 0  
Packets Dropped: 0  
Bad Packets: 0
```

```
Max Packets on S/W Transmit Queue: 98  
S/W Transmit Queue Overflow: 0  
Current S/W+H/W Transmit Queue Length: 2
```

```
Broadcast Packets: 1615551  
Multicast Packets: 622570  
No Carrier Sense: 0  
DMA Underrun: 0  
Lost CTS Errors: 0  
Max Collision Errors: 0  
Late Collision Errors: 359712  
Deferred: 207086  
SQE Test: 0  
Timeout Errors: 0  
Single Collision Count: 191677  
Multiple Collision Count: 48  
Current HW Transmit Queue Length: 2
```

```
Broadcast Packets: 1610367  
Multicast Packets: 622568  
CRC Errors: 0  
DMA Overrun: 0  
Alignment Errors: 0  
No Resource Errors: 0  
Receive Collision Errors: 0  
Packet Too Short Errors: 0  
Packet Too Long Errors: 0  
Packets Discarded by Adapter: 0  
Receiver Start Count: 0
```

```
General Statistics:
```

```
-----  
No mbuf Errors: 0  
Adapter Reset Count: 0  
Driver Flags: Up Broadcast Running  
              Simplex 64BitSupport
```

Network parameters such as `thewall`, `tcp_sendspace`, and `tcp_recvspace` can be tuned with the **optimizenet** command. Information about available parameters such as `default`, `current`, and `range` is shown in Example 8-37 on page 308.

Example 8-37 Network parameters list

```
$ optimizenet -list
```

NAME	CUR	DEF	BOOT	MIN	MAX	UNIT	TYPE
DEPENDENCIES							
arptab_bsz	7	7	7	1	32K-1	bucket_size	R
DEPENDENCIES							
arptab_nb	73	73	73	1	32K-1	buckets	R
DEPENDENCIES							
clean_partial_conns	0	0	0	0	1	boolean	D
DEPENDENCIES							
net_malloc_police	0	0	0	0	8E-1	numeric	D
DEPENDENCIES							
rfc1323	0	0	0	0	1	boolean	C
DEPENDENCIES							
route_expire	1	1	1	0	1	boolean	D
DEPENDENCIES							
tcp_pmtu_discover	1	1	1	0	1	boolean	D
DEPENDENCIES							

tcp_recvspace sb_max	16K	16K	16K	4K	8E-1	byte	C

NAME	CUR	DEF	BOOT	MIN	MAX	UNIT	TYPE
DEPENDENCIES							

tcp_sendspace sb_max	4K	16K	16K	4K	8E-1	byte	C

NAME	CUR	DEF	BOOT	MIN	MAX	UNIT	TYPE
DEPENDENCIES							

thewall	128K	128K	128K	0	1M	kbyte	S

NAME	CUR	DEF	BOOT	MIN	MAX	UNIT	TYPE
DEPENDENCIES							

udp_recvspace sb_max	42080	42080	42080	4K	8E-1	byte	C

NAME	CUR	DEF	BOOT	MIN	MAX	UNIT	TYPE
DEPENDENCIES							

udp_sendspace sb_max	9K	9K	9K	4K	8E-1	byte	C

For more about Shared Ethernet Adapters, refer to 6.8, “Virtual SCSI” on page 205.



Application tuning

This chapter provides an introduction for Fortran and C/C++ programmers or users who are interested in tuning their applications for POWER5. Although this chapter assumes that the reader has a working knowledge of these concepts, some sections are very well suited for those who are beginning to measure applications performance and would like to start improving the performance of a particular application. Also, it is important to point out that for POWER3 and POWER4 an entire book was dedicated to providing a tuning guide. Here, much of that work has been condensed into a single chapter. Therefore, we make reference to those books as much as possible to preserve this chapter's issues that are related to POWER5.

In this chapter we cover the following major topics:

- ▶ Identification of the type of bottleneck and its location within the code
- ▶ Tuning using compiler flags
- ▶ Profiling the code to uncover performance bottlenecks
- ▶ General tuning for single processor performance
- ▶ Making use of highly optimized libraries
- ▶ General tuning for parallel performance

9.1 Performance bottlenecks identification

This section gives an overview of the basic steps that are required to localize performance bottlenecks. In this chapter we assume that performance limitations are not related to hardware but are a function of how an application was coded. Our definitions of performance bottlenecks and code optimization terms follow:

Performance bottleneck

Sections of a code that tend to consume most of the user time, elapsed time, or both (also referred to as real time or wall-clock time) that, after careful analysis, require code optimization.

Code optimization

A series of steps that are required to modify a section or sections of the code, manually or via the compiler, to improve performance.

Elapsed time

The time that it took the program to run from beginning to end. This is the sum of all factors that can delay the program, plus the program's own attributed costs.

User time

This is the time used by itself and any library routine that it calls while it is attached to a processor.

System time

The time used by system calls invoked by the program, directly or indirectly.

In this definition, optimization falls into two categories:

▶ Code optimization using advanced compiler flags

In this type of optimization we simply rely on how much performance we can gain by selecting the best combination of compiler flags for a particular application.

▶ Hand-tuning

This type of optimization requires manually modifying the code to improve performance. In an extreme case different methodology might be required.

Figure 9-1 on page 313 illustrates the basic steps that are required to localize performance bottlenecks.

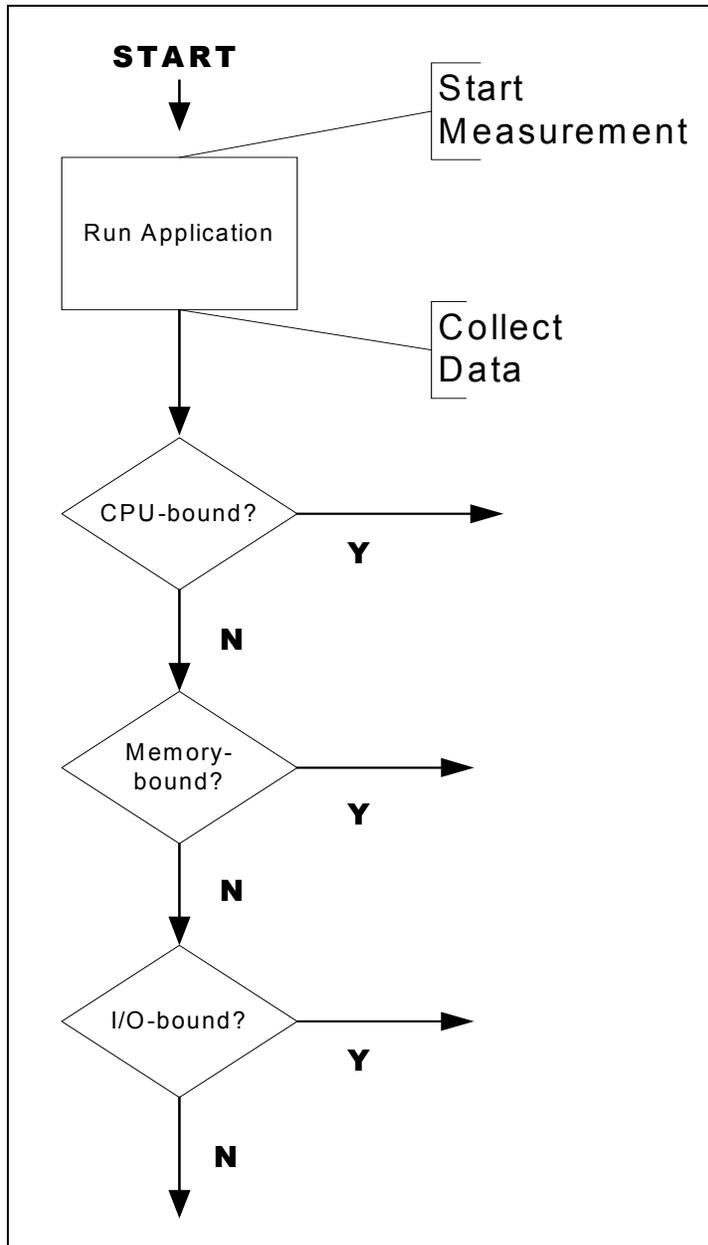


Figure 9-1 This flowchart illustrates the first step in applications tuning

The first step requires running the application and collecting data from some sort of measurement. The simplest way to start analyzing an application is by looking at timers. It does not require recompiling or modifying the code. The most

common timing routines can be invoked at run time. Of course, carrying out timing of certain routines inside the code requires recompiling and building the program. These two techniques are covered in the next section.

9.1.1 Time commands, time utilities, and time routines

In this section we describe commands, utilities, and routines that may be used to identify bottlenecks in the application. The set described here is by no means the only set of tools available for performing a coarse analysis but we have found them to be fairly common and easy to use. The commands described here are:

- ▶ **time**
- ▶ **timex**
- ▶ **vmstat**
- ▶ **irtc**
- ▶ **rtc**

The first and simplest set of commands are **time** and **timex**. They both print elapsed time, user time, and system time of a command during execution in seconds. In their simplest form they can be invoked as follows:

```
$/usr/bin/timex a.out
```

or

```
$/usr/bin/time a.out
```

A version of **time** that produces additional information corresponds to the C shell built-in command. The following example illustrates the additional information obtained with this built-in version:

```
%time a.out
%5.1u 0.1s 0:05 98% 137+91548k 0+0io 27pf+0w
```

Table 9-1 shows the meaning of the fields from the **time** C shell built-in version.

Table 9-1 *time* description fields

Field	Description
5.1 u	Number of seconds of user time
0.1 s	Number of seconds of user time consumed by system calls invoked by the program
0:05	Elapsed time
98%	Total user time plus system time, as a percentage of elapsed time
137+9154 8 k	Average amount of shared memory used, plus average amount of unshared data space used, in kilobytes

Field	Description
0+0 io	Number of blocks input and output operations
27pf+0w	Page faults plus number of swaps

Prior to providing empirical rules to classify the performance of a particular application, we define CPU-bound, memory-bound, and I/O-bound applications:

- CPU-bound** When sections of the code dominate most of the run time by performing processor calculations
- Memory-bound** When sections of the code dominate most of the run time by memory issues or memory limitations
- I/O-bound** When sections of the code dominate most of the run by performing I/O

The output provided by these simple commands can give an initial indication of the type of bottleneck in our particular application. *The following empirical rules should be viewed as guidance for this classification:*

- ▶ Excessively large user time can be an indication of a CPU-bound application that might not be running optimally and requires tuning.
- ▶ The ratio between elapsed time and the user time ($r_{W/O} = \text{elapsed time} / \text{user time}$) may provide an indication of an I/O-bound application. A ratio larger than 1 represents an imbalance between elapsed time and user time. For certain cases this may be interpreted as a large I/O wait time. A ratio larger than 2.5 is for us an empirical threshold in the I/O performance that must be considered. In addition, the C shell built-in time function provides information about the number of blocks of input and output, which can be an indication of the amount of I/O that an application is performing for a particular run.
- ▶ In general, user time tends to be larger, by at least an order of magnitude, than the system time. A large system time could be attributed to a memory-bound program. If page faults plus number of swaps and the average amount of shared memory used is large, this may help confirm suspicion of memory-bound code.

In this chapter we briefly mention **vmstat** as an alternative way to get similar information as in the case of **time** or **timex**. (See 8.1.3, “vmstat command” on page 268 for more details.) On a system, **vmstat** can provide statistics about kernel threads, virtual storage, disks, and CPU capacity. These systemwide statistics are calculated as averages for values reported as percentages or sums.

Attention: **vmstat** provides systemwide statistics.

Example 9-1 illustrates an example in which a small program that performs a matrix multiplication is monitored using **vmstat**.

Example 9-1 Example invoking vmstat

```
$ matmul.F2000

$ vmstat 1 10
kthr      memory          page          faults          cpu
-----
 r   b   avm   fre re  pi  po  fr  sr  cy  in  sy  cs  us  sy  id  wa
 1   1 2533495 1516953 0 0 0 0 0 0 0 445 1249 336 0 0 99 0
 0   0 2533499 1516949 0 0 0 0 0 0 0 447 3397 336 0 0 99 0
 0   0 2533499 1516949 0 0 0 0 0 0 0 445 3234 329 0 0 99 0
 0   0 2533499 1516949 0 0 0 0 0 0 0 444 3246 335 0 0 99 0
 1   0 2556989 1493459 0 0 0 0 0 0 0 449 3301 351 16 3 81 0
 1   0 2556989 1493459 0 0 0 0 0 0 0 444 3198 338 25 0 75 0
 1   0 2556989 1493459 0 0 0 0 0 0 0 447 3267 356 25 0 75 0
 7   0 2556989 1493459 0 0 0 0 0 0 0 446 3227 340 25 0 75 0
 2   0 2556993 1493455 0 0 0 0 0 0 0 448 3297 347 25 0 75 0
 2   0 2556993 1493455 0 0 0 0 0 0 0 446 3210 338 25 0 75 0
```

If fine-grain timing is required, see Example 9-2 and Example 9-3 on page 317 for templates for the use of `irtc()` and `rtc()`. These two functions require code modification and recompilation.

IRTC This function returns the number of nanoseconds since the initial value of the machine's real-time clock.

Example 9-2 irtc() template

```
integer(8) T1, T2, IRTC

T1 = IRTC()

[your section of the code]

T2 = IRTC()
write(*,*)'Untuned loop took', (T2-T1)/1000000, 'msec'

end
```

RTC The rtc function returns the number of seconds since the initial value of the machine's real-time clock.

Example 9-3 rtc() template

```
real*8 T1, T2, RTC
```

```
T1 = RTC()
```

```
[Your section of the code]
```

```
T2 = RTC()
```

```
write(*,*)'Untuned loop took', (T2-T1), 'sec'
```

```
end
```

9.2 Tuning applications using only the compiler

In this section we provide an overview of the compiler, selected features of the XL Fortran, and XL C and C++ compiler that relates to the optimization of applications running on POWER5 processors, and we examine some of the compiler flags that are relevant for scientific and engineering applications.

The emphasis in this section is on illustrating the compiler capabilities to carry out code optimization. *This is what we consider the first step in the process of code optimization.* However, for many programmers who are not interested in optimizing their applications further, this step usually turns out to be the last step. In general, this is an important section for all programmers.

9.2.1 Compiler brief overview

Compiler technology represents a formidable challenge, in particular the development of an optimizer that can tune any code.

The Toronto Portable Optimizer (TPO) is designed to operate on many source languages for many target platforms. Figure 9-2 shows that TPO is a key component in the overall compiler architecture.

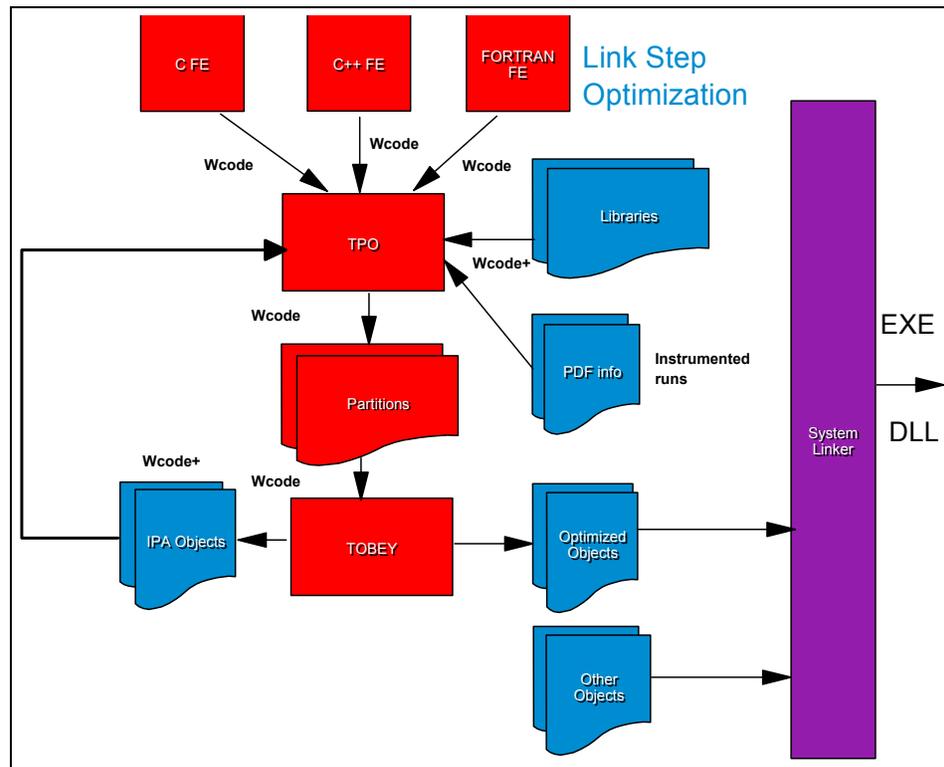


Figure 9-2 IBM compiler architecture

Figure 9-3 delves deeper into the compiler architecture and shows the steps that TPO goes through to optimize codes. This compiler is designed to optimize at several levels:

- ▶ Expression
- ▶ Basic block
- ▶ Procedure
- ▶ Whole program

Some of the TPO strengths are:

- ▶ Deep analysis
- ▶ High-level program restructuring
- ▶ Portability
- ▶ Seamless product integration

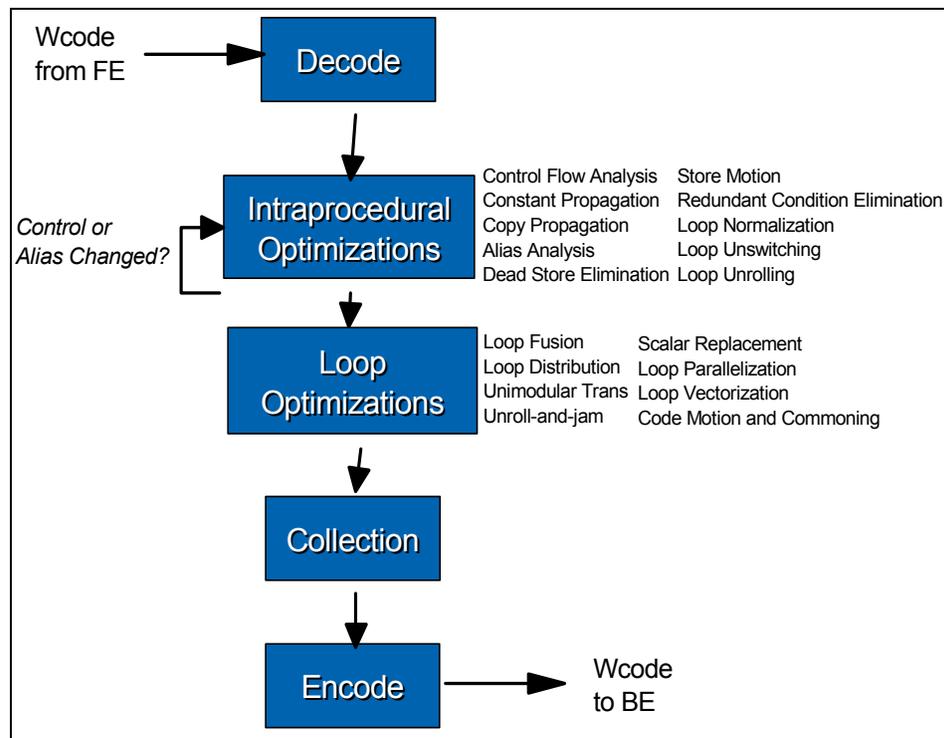


Figure 9-3 Inside TPO compile time

Figure 9-4 summarizes the details of loop optimization. Some of the techniques that are illustrated are presented in this chapter, in particular, loop fusion and loop unrolling.

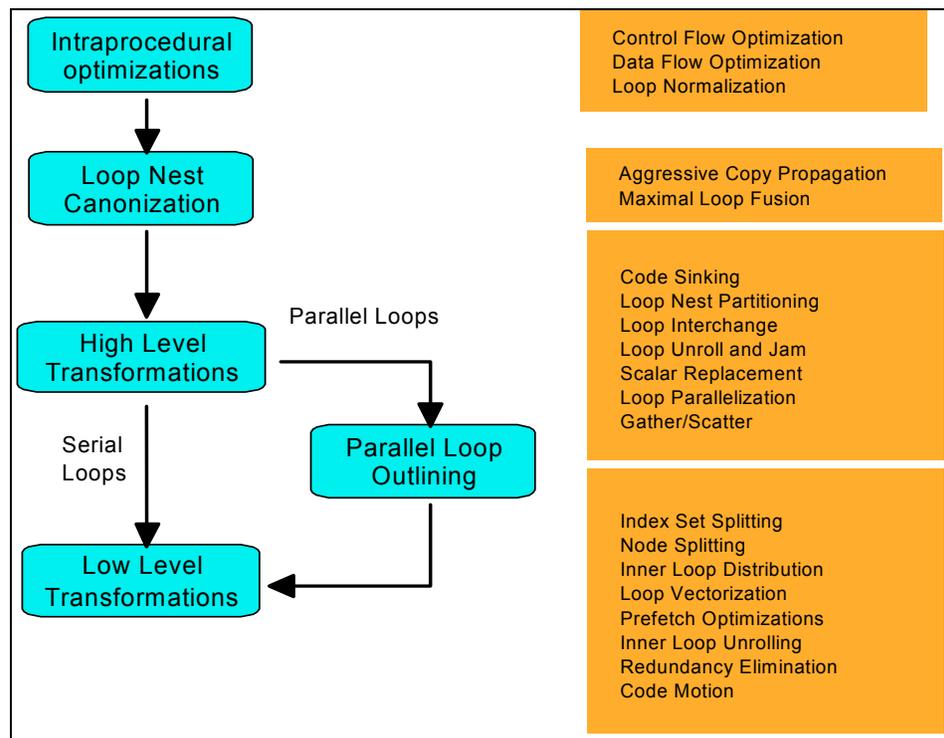


Figure 9-4 Loop optimization overview in TPO

In general, compiler optimization strategies are mainly based on *cost reduction* and exploitation of parallel facilities. More specifically:

- ▶ Reduce expected dynamic path length
- ▶ Reduce cost of accessing data memory
- ▶ Reduce cost of accessing instruction memory
- ▶ Exploit parallel facilities on the target platform to run instructions concurrently

It is important to understand that they will change the original code and the programmer might need to be aware of the consequences and effects of code re-ordering. Some of these techniques will be provided later as general rules for hand-tuning code.

9.2.2 Most commonly used flags

In this section we look at the compiler flags that affect the performance of an application. For a comprehensive list, visit the IBM AIX compiler information center at:

<http://publib.boulder.ibm.com/infocenter/comphelp/index.jsp>

It is important to always check answers as you increase the level of compiler optimization or aggressivity. This is due to the fact that the compiler makes certain assumptions about some of the statements in the code that can potentially be optimized by rewriting that section of the code.

A typical example is the property of associativity in a product. At low levels of compiler optimization (for example, -O2), XL Fortran will always evaluate $a*b*c$ starting from a , even if $b*c$ has already been computed. Although more time will be consumed, it is safer because the answer might be dependent on the order of execution. As the level of optimization increases, some of these restrictions might be eliminated.

Optimization level

A few basic rules to remember when using the compiler for optimization:

- ▶ Optimization requires additional compilation time.
- ▶ Optimization produces faster code; but always check answers, especially when using aggressive levels of compiler optimization.
- ▶ By default, the compiler chooses -O0 or -qnoopt.
- ▶ Enable compiler optimization with -ON; where N is 0, 2, 3, 4, or 5
Example: `$xlf -O3`

Next, we discuss the different levels of compiler optimization regarding the effects of performance flags on scientific and engineering applications.

Level 0: -O0

This option is recommended for debugging. It is the fastest way to compile the program. *It preserves program semantics.* This is also useful to see the effect of hand-tuning small kernels or certain loops.

Level 2: -O2

This is the same as -O. At this level, the compiler performs conservative optimization. The optimization techniques used at this level are:

- ▶ Global assignment of user variables to registers, also known as graph coloring register allocation

- ▶ Strength reduction and effective use of addressing modes
- ▶ Elimination of redundant instructions, also known as common subexpression elimination
- ▶ Elimination of instructions whose results are unused or that cannot be reached by a specified control flow, also known as dead code elimination
- ▶ Value numbering (algebraic simplification)
- ▶ Movement of invariant code out of loops
- ▶ Compile-time evaluation of constant expressions, also known as constant propagation
- ▶ Control flow simplification
- ▶ Instruction scheduling (reordering) for the target machine
- ▶ Loop unrolling and software pipelining

Level 3: -O3

At this level the compiler performs more extensive optimization. This includes:

- ▶ Deeper inner-loop unrolling
- ▶ Better loop scheduling
- ▶ Increased optimization scope, typically to encompass a whole procedure
- ▶ Specialized optimizations (those that might not help all programs)
- ▶ Optimizations that require large amounts of compile time or space
- ▶ Elimination of implicit memory usage limits (equivalent to compiling with `qmaxmem=-1`)
- ▶ Implies `-qnostrict`, which allows some reordering of floating-point computations and potential exceptions

Important: At this level `-qnostrict` is invoked by default. This implies:

- ▶ Reordering of floating-point computations
- ▶ Reordering or elimination of possible exceptions (such as division by zero, overflow)

Level 4: -O4

At this level the compiler introduces more aggressive optimization and increases the optimization scope to the entire program. This option includes:

- ▶ Includes `-O3`
- ▶ `-qhot`

- ▶ -qipa
- ▶ -qarch=auto
- ▶ -qtune=auto
- ▶ -qcache=auto

Level 5: -O5

At this level the compiler introduces more aggressive optimization. This option includes:

- ▶ Includes -O4
- ▶ -qipa=level=2

Important: If -O5 is specified on the compile step, then it should be specified on the link step.

Machine-specific flags

This set of flags is specific to a family architecture. The idea is to provide code that is optimized for a particular architecture.

Table 9-2 Machine-specific flags

Option	Description
-q32	For 32-bit execution mode.
-q64	For 64-bit execution mode.
-qarch	Selects specific architecture for which instruction is generated
-qtune	Biases optimization toward execution on a given processor, without implying anything about the instruction set architecture to use as a target
-qcache	Defines a specific cache or memory

Important: By default, the compiler generates code that runs on all supported systems, but it might not be optimized for a particular system. This default is true only for the low level of compiler optimization. As mentioned above, -O4 implies -qarch=auto, which generates code compatible with the machine used for compilation (and not necessarily every supported architecture).

High-order transformations

-qhot optimization is targeted to improve the performance of loops and array language. This may include:

- ▶ Loop optimization
 - Loop-nest canonization
 - Aggressive copy propagation to create more perfect loop nests
 - Aggressive loop fusion to create larger loops and loop nests
 - Code sinking to create more perfect loop nests
 - High-level transformations (outer loops)
 - Loop distribution to create more perfect loop nests
 - Loop interchange for data locality and outermost parallelization
 - Loop unroll-and-jam for data reuse
 - Gather/scatter to create more perfect loop nests
 - Peeling to eliminate loop-carried dependencies
 - Identification and outlining of parallel loops
 - Low-level transformations (Inner loops)
 - Node splitting, scalar replacement, and automatic vectorization
 - Inner loop distribution (sensitive to number of hardware streams)
 - Gather/scatter and index set splitting to eliminate branches in inner loops

The goals of high-order transformation are:

- ▶ Reducing the costs of memory access through the effective use of caches and translation look-aside buffers
- ▶ Overlapping computation and memory access through effective utilization of data-prefetching capabilities provided by the hardware
- ▶ Improving the utilization of processor resources through reordering and balancing the usage instructions with complementary resource requirements

Interprocedural analysis

-qipa enables the compiler to perform optimization across different files. In other words, it provides analysis for the entire program. The interprocedural analysis has the suboptions shown in Table 9-3.

Table 9-3 -qipa suboptions

Suboption	Description
level=0	Automatic recognition of standard libraries.
	Localization of statistically bound variables and procedures.
	Partitioning and layout of procedures according to their calling relationships, which is also referred to as their call affinity.
	Expansion of scope for some optimizations, especially register allocation.
level=1	Procedure inlining.
	Partitioning and layout of static data according to reference affinity.
level=2	Whole-program alias analysis. This level includes the disambiguation of pointer dereferences and indirect function calls, and the refinement of information about the side effects of a function call.
	Intensive interprocedural optimizations. This can take the form of value numbering, code propagation and simplification, code motion into conditions or out of loops, elimination of redundancy.
	Interprocedural constant propagation, dead code elimination, pointer analysis.
	Procedure specialization.

XL Fortran new and changed functionality

Some features have been added or improved in the XL Fortran compiler. We provide a brief overview of this new functionality, and Table 9-4 shows some of the new options.

Table 9-4 *New options and suboptions*

Options and suboptions	Comments
-qftrap=nanq	Detects all NaN values handled or generated by floating-point instructions, including those not created by invalid operations.
-qport=nullarg	Treats an empty argument, which is delimited by a left parenthesis and a comma, two commas, or a comma and a right parenthesis, as a null argument.
-qmodule=mangle81	Provides compatibility with the V8.1 module naming conventions for non-intrinsic modules.
-qsaveopt	Saves the command-line options used for compiling a source file in the corresponding object file.
-qversion	Provides the version and release for the invoking compiler.

In addition, new options and suboptions that affect performance have been added with the XL Fortran 9.1 compiler. Table 9-5 summarizes these newly added options and suboptions. Some of the options presented in this table are discussed in more detail in other sections.

Table 9-5 *Changed options and suboptions*

Option/Suboption	Description
-qarch and -qtune	These two options now provide support for POWER5 and PowerPC 970 architectures (pwr5 and ppc970).
-qshowpdf and -qpdf1	Provide additional call and block count profiling information to an executable.
showpdf and mergepdf utilities	Provide enhanced information about PDF-directed compilation; mergepdf merges two or more PDF files.
-qdirecstorage	Informs the compiler that a given compilation unit may reference write-through-enabled or cache-inhibited storage.

Option/Suboption	Description
SWDIV and SWDIV_NOCHK intrinsics	Provide software floating-point division algorithms.
FRE and FRSQRTES intrinsic	Floating-point reciprocal estimate and floating-point square root reciprocal.
POPCNT and POPCNTB intrinsics	Provide set bit counts in registers for data objects.
POPPAR intrinsic	Determines the parity for a data object.

Note: The compiler will use either `fddiv` or `FRE`, if computing with `-qarch=pwr5`, and depending on which one it deems better. In particular, single block loops will sometimes use `fddiv` rather than `FRE` and the expansion, because overall latency is sometimes more important than parallelization.

9.2.3 Compiler directives for performance

After the compiler flags have been optimized, the programmer can still use highly optimized libraries, compiler directives, or both to improve performance without major changes to the code. (We cover highly optimized libraries in 9.6, “Optimized libraries” on page 360.) This section concerns compiler directives, in particular looking at directives for code tuning and hardware-specific directives that potentially can help improve performance.

To identify a sequence of characters called trigger constants, XL Fortran uses the `-qdirective` option:

```
-qdirective [=directive_list] | -qnodirective [=directive_list]
```

The compiler recognizes the default trigger constant `IBM*`. Table 9-6 on page 328 provides a list of assertive, loop optimization, and hardware-specific directives.

Table 9-6 Assertive, loop optimization, and hardware-specific directives

Directive	Type	Description
ASSERT	Assertive	Provides characteristics of do loops for further optimization; requires -qsmp or -qhot
CNCALL		Declares that no loop-carried dependencies exist within any procedure called from the loop; requires -qsmp or -qhot
INDEPENDENT		Must precede a loop, FORALL statement; it specifies that the loop can be executed and iterations in any order without affecting semantics; requires -qsmp or -qhot
PERMUTATION		Specifies that the elements of each array listed in the integer_array_name_list have no repeated values; requires -qsmp or -qhot
BLOCK_LOOP	Loop optimization	Allows blocking inside nested loops; requires -qhot or -qsmp
LOOPID		Allows the assignment of a unique identifier to loop within a scoping unit
STREAM_UNROLL		Allows for a combination of software prefetch and loop unrolling; requires -qhot, -qipa=level=2, or -qsmp, and -O4
UNROLL		Allows loop unrolling where applicable
UNROLL_AND_FUSE		Allows loop unrolling and fuse where applicable

Directive	Type	Description
CACHE_ZERO	Hardware-specific	Invokes machine instruction dcbz; sets the data cache block corresponding to the variable specified to zero
ISYNC		Enables discarding of any prefetched instructions after all preceding instructions complete
LIGHT_SYNC		Ensures that all stores prior to LIGHT_SYNC complete before any new instructions can be executed on the processor that executed the LIGHT_SYNC directive
PREFETCH_BY_STREAM		Uses the prefetch engine to recognize sequential access to adjacent cache lines and then requests anticipated lines from deeper levels of memory hierarchy
PREFETCH_FOR_LOAD		Prefetches data into the cache for reading by way of a cache prefetch instruction
PREFETCH_FOR_STORE		Prefetches data into the cache for writing by way of a cache prefetch instruction
PROTECTED_UNLIMITED_STREAM_SET_GO_FORWARD		Establishes an unlimited-length protected stream that begins with the cache line at the specified prefetch variable and fetches from increasing memory addresses
PROTECTED_UNLIMITED_STREAM_SET_GO_BACKWARD		Fetches from decreasing memory addresses
PROTECTED_STREAM_SET_GO_FORWARD		Establishes a limited-length protected stream that begins with the cache line at the specified prefetch variable and fetches from increasing memory
PROTECTED_STREAM_SET_GO_BACKWARD		Fetches from decreasing memory addresses

Directive	Type	Description
PROTECTED_STREAM_COUNT		Sets the number of cache lines for the specified limited-length stream
PROTECTED_STREAM_GO		Starts to prefetch all limited-length streams
PROTECTED_STREAM_STOP		Stops prefetching the specified protected stream
PROTECTED_STREAM_STOP_ALL		Stops prefetching all protected streams

Directives usage

In this section we provide a series of examples that illustrate how to apply some of these compiler directives. Some of them are not difficult to implement in the code, and others are more involved.

As we previously described, the ASSERT directive provides a way to specify that a particular DO loop does not have dependencies. The assertion can take the following forms:

- ▶ ITERCNT(n); where n specifies the number of iterations for a given DO loop. n must be positive, scalar, and an integer initialization expression.
- ▶ NODEPS specifies that no loop dependencies exist within a particular DO loop.

Important: The ASSERT directive applies only to the DO loop following the directive. It does not apply to nested DO loops.

Example 9-4 ASSERT directive

```
c  ASSERT Directive
   program dir1
   implicit none
   integer i,n, fun
   parameter (n = 100000)
   real*8 a(n)
   integer(8)  t0, tfin, irtc
   do i = 1,n
     a(i) = rand()
   end do
c ...  start timer
      t0 = irtc()
!IBM*  ASSERT (NODEPS)
```

```

do i = 1, n
  a(i) = a(i) * fun(i)
end do
c ... time
tfin = (irtc() - t0)/1000000
write(6,*)'Time: ',tfin, 'msec.'
stop
end
C
function fun(i)
fun = i * i
return
end

```

In this example we used the idea of loop-carried dependencies (or data dependency), because this concept is commonly used throughout this chapter.

Dependencies Current iteration requires data computed in some previous iteration, or computes data for some subsequent iteration.

An example in a loop with $a(i) = a(i-1)*2$ is that computing $a(5)$ requires $a(4)$.

The loop optimization directive is `BLOCK_LOOP`. This directive relies on a well-known optimization technique called *blocking*. This directive separates large iterations into smaller groups of iterations. The basic idea is to increase the utilization of the submemory hierarchy. Note that in Example 9-5, `L2_cache_size` and `L3_cache_size` must be assigned values corresponding to the cache of the particular system where this example will be executed.

Example 9-5 BLOCK_LOOP directive

```

c BLOCK_LOOP Directive
program dir4
implicit none
integer i,j,k,n
integer L3_cache_size, L3_cache_block
integer L2_cache_size, L2_cache_block
parameter (n = 100)
integer a(n,n), b(n,n), c(n,n)
integer(8) t0, tfin, irtc
do j = 1,n
  do i = 1,n
    a(i,j) = rand()
    b(i,j) = rand()
  enddo
enddo
do j = 1, n

```

```

        do i = 1, n
            c(i,j)=0.0
        enddo
    enddo
c ... start timer
    t0 = irtc()
!IBM* BLOCK_LOOP(L3_cache_size, L3_cache_block)
    do i = 1, n

!IBM* LOOPID(L3_cache_block)
!IBM* BLOCK_LOOP(L2_cache_size, L2_cache_block)
        do j = 1, n

!IBM* LOOPID(L2_cache_block)
            do k = 1, n
                c(i,j) = c(i,j) + a(i,k) * b(k,j)
            enddo
        enddo
    enddo
c ... time
    tfin = (irtc() - t0)/1000000
    write(6,*)'Time: ',tfin, 'msec.'
    call dummy (c,n)
    stop
end
c

```

9.2.4 POWER5 compiler features

Some of the options and suboptions that perform specific optimization for the POWER5 processor microarchitecture are:

- ▶ -qarch=pwr5
- ▶ -qtune=pwr5
- ▶ -qcache=auto

Also, the following intrinsics are included:

SWDIV	Provides an algorithm to carry out division on POWER5.
SWDIV_NOCHK	Similar to SWDIV except that checking for invalid arguments is not performed.
FRE(S)	Provides an algorithm to estimate the reciprocal of a floating-point on POWER5; used for single-precision.
FRSQRTE(S)	Provides an algorithm to eliminate the reciprocal of a square root operation on POWER5; used for single-precision floating point.

POPCNT	Used to count the number of set bits in a data object. The resulting value is the number of bits set to ON or 1.
POPCNTB	Used to count the number of set bits of each byte in a register. The result is an INTEGER(4) in 32-bit mode or iNTEGER(8) in 64-bit mode.
POPPAR	Used to determine the parity for a data object. The result is 1 if there is an odd number of bits set, or 0 if there is an even number of bits set.

Compiler directives

XL Fortran 9.1 introduces a few new directives for POWER5. These streams are protected from being replaced by any hardware-detected streams. The directives correspond to:

- ▶ Valid for PowerPC 970 and POWER5:
PROTECTED_UNLIMITED_STREAM_SET_GO_BACKWARD
- ▶ Valid for PowerPC 970 and POWER5:
PROTECTED_STREAM_SET_GO_FORWARD
- ▶ Valid for POWER5:
PROTECTED_STREAM_SET_GO_BACKWARD
PROTECTED_STREAM_COUNT
PROTECTED_STREAM_GO
PROTECTED_STREAM_STOP
PROTECTED_STREAM_STOP_ALL

Example 9-6 illustrates the use of some of these directives and their performance benefits. The performance improvement for n between 10 and 2000 oscillates from a value of 2% to as large as 11%. In this four-stream case the prefetch directives improve performance of short vector lengths.

Example 9-6 New POWER5 prefetch directives

```

c  PROTECTED_STREAM_SET_FORWARD Directive
c  PROTECTED_STREAM_COUNT Directive
  program dir8
  implicit none
  integer i,j,k,n,m,nopt2,ndim2,lcount
  parameter (n=2000)
  parameter (m=1000)
  parameter (ndim2 = 1000)
  real*8 x(n,ndim2),a(n,ndim2),b(n,ndim2),c(n,ndim2)
  integer(8) irtc, t0, tfin
  do j = 1, ndim2
    do i = 1, n
      x(i,j) = rand()
    
```

```

        a(i,j) = rand()
        b(i,j) = rand()
        c(i,j) = rand()
    end do
end do
c ... start timer
t0 = irtc()
do k = 1, m
    lcount = 1 + n/16
    do j = ndim2, 1, -1
!IBM* PROTECTED_STREAM_SET_FORWARD(x(1,j),0)
!IBM* PROTECTED_STREAM_COUNT(lcount,0)
!IBM* PROTECTED_STREAM_SET_FORWARD(a(1,j),1)
!IBM* PROTECTED_STREAM_COUNT(lcount,1)
!IBM* PROTECTED_STREAM_SET_FORWARD(b(1,j),2)
!IBM* PROTECTED_STREAM_COUNT(lcount,2)
!IBM* PROTECTED_STREAM_SET_FORWARD(c(1,j),3)
!IBM* PROTECTED_STREAM_COUNT(lcount,3)
!IBM* EIEIO
!IBM* PROTECTED_STREAM_GO
        do i = 1, n
            x(i,j) = x(i,j) + a(i,j) * b(i,j) + c(i,j)
        enddo
    enddo
    call dummy(x,n)
enddo
c ... time
tfin = (irtc() - t0)/1000000
write(6,*) 'Time: ',tfin, 'msec.'
stop
end

c
subroutine dummy(x,n)
c
dimension x(n)
c
return
end

```

Recommended compiler flags

In the POWER3 tuning guide, the compiler flags recommended for POWER3 are:

```

-03 -qarch=pwr3 -qtune=pwr3 [-qcache=auto] or
-03 -qstrict -qarch==pwr3 -qtune=pwr3 [-qcache=auto]

```

For POWER4 the recommendation for starting compiler options is:

```

-03 -qarch=pwr4 -qtune=pwr4

```

In this POWER5 version, the recommendation is:

```
-O3 -qarch=auto -qtune=auto -qcache=auto
```

- O level** The -O3 option provides conservative optimization and it is currently used on most major scientific and engineering applications. In some cases -O2 might be preferable over -O3, particularly for very large applications where after extensive testing the -O3 does not show better performance. Also, it is a good starting point for somebody who is planning to optimize the application using higher compiler levels of optimization such as -O4 and -O5. When using -O3, the next step should be to improve the level of compiler optimization by introducing -qhot (also called higher-order transformations). The objective of this option is to optimize loops. For very large scientific and engineering applications, the use of -O4 or -O5 will most likely have to be restricted to a few routines.
- qarch** This option and suboptions present several possibilities to choose from. The selection of the suboption depends on how the binaries will be used. In our recommendation we assume that the application will be built on the system where the executables will be used. With -qarch=auto, the particular architecture where the binaries are built will be recognized. If only one architecture will be used, then -qarch=[pwr5] or [pwr4], and so on is recommended. On the other hand, if only one set of binaries will be used across all platforms, then you might consider -qarch=com (or compile for ALL PowerPC platforms by specifying -qarch=pwr).
- qtune** Instruction selection, scheduling, and other implementation-dependent performance enhancements for a specific implementation of hardware architecture. We also recommend this option as auto. Similar to the previous case, this option is specific to the application (or where the executable will run).

Note: When running an application on more than one architecture, but the desire is to tune it for a particular architecture, the combination of -qarch and -qtune can be used.

Important: -qtune can improve performance but only has an effect when used in combination with options that enable optimization.

-qcache We recommend this option as auto.

Finally, there are other options used by scientific and engineering applications that might work for your particular application as well. These correspond to:

- ▶ -qstrict
- ▶ -qmaxmem
- ▶ -qfixed
- ▶ -q64

9.3 Profiling applications

In the beginning of this chapter when we introduced the flowchart to identify application bottlenecks (Figure 9-1 on page 313), we learned as a first approximation how to classify applications as CPU-bound, memory-bound, and I/O bound. We now introduce a series of tools to help localize crucial sections in the code. These crucial sections correspond to a section or sections of the code that tend to dominate CPU utilization, and this is reflected in the user time.

9.3.1 Hardware performance monitor

A key feature of this utility is its ability to provide *hardware performance counters information*. It can provide very fine-grain information about how the application that it being monitored takes advantage (or not) of the Power Architecture. This utility is part of the IBM High Performance Computing Toolkit. This package includes the following new software (and a new installation method):

- ▶ Watson Sparse Matrix Library (WSMP)
- ▶ Modular I/O Performance Tool (MIO)
- ▶ MPI Tracer™
- ▶ SHMEM Profiler
- ▶ OpenMP Profiler (PompProf)
- ▶ Graphical Interface tool with source code traceback (PeekPerf)
- ▶ New installation method via RPM modules (Linux)

This set of tools can be classified based on the type of performance that the programmer is interested in analyzing. Based on the functionality of the different utilities, the following components can be analyzed with this package:

- ▶ Hardware performance (HPM Toolkit)
 - catch
 - hpmcount
 - libhpm
 - hpmstat

- ▶ Shared memory performance (DPOMP)
 - pomprof
- ▶ Message-passing performance
 - MP_profiler, MP_tracer
 - TurboSHMEM, TurboMP
- ▶ Memory performance
 - Sigma
- ▶ Performance visualization
 - PeekPerf
- ▶ I/O performance
 - MIO (modular I/O)
- ▶ Mathematics performance
 - Watson sparse matrix (WSMP)

This toolkit can be obtain in one of these ways:

- ▶ Acquired as part of new procurement
- ▶ Acquired as part of ACTC performance tuning workshop
- ▶ License purchased directly from IBM Research

In this section we discuss the version of hpmcount that works on the POWER4 processor. The hpmcount usage for POWER4 is:

```
$hpmcount [-o <file>] [-n][-g <group>] <a.out>
```

For help and additional information from hpmcount:

```
$hpmcount [-h] [-c] [-l]
```

Table 9-7 *hpmcount flags and description*

Flag	Description
-h	Display all available flags and a brief description.
-c	List all events from all counters.
-l	List all the groups available on POWER4.
-o <file>	Create an output file with all collected statistics called <file>.<pid>. For parallel runs this flag generates one file for each process.
-n	Force hpmcount <i>not</i> to send output to stdout. Only active in combination with -o <file>.
-g <group>	List selected groups

The groups that are available on POWER4 are extensive, and the valid groups go from 0 to 60. Selecting the groups to analyze depends on the type of information that is being searched, but the groups that have been recommended for applications tuning are listed in Table 9-8. A complete listing of the groups for POWER4 may be found in `/usr/pmapilib/POWER4.gps`.

Table 9-8 Selected set of groups for applications tuning

Group	Information provided
5	pm_lsource, information on data source, counts of loads from L2, L3, and memory
53	pm_pe_bench1, information for fp analysis, counts of cycles on instructions, fixed-point operations, and FP operations (includes divides, SQRT, FMA, and FMOV or FEST)
56	pm_pe_bench4, information for L1 and TLB analysis, counts of cycles on instructions, TLB misses, loads, stores, and L1 misses
58	pm_pe_bench6, information for L3 analysis, counts of cycles on instructions, loads from L3, and loads from memory
60	pm_hpmcount2, information for computation intensity analysis, counts of cycles on instructions on FP operations (including divides, FMA, loads, and stores)

The output obtained from `hpmcount` can be divided into several sections:

1. This section prints the `hpmcount` version and the total elapsed time taken to perform a particular run.
2. Prints the resource usage statistics.
3. Lists hardware counter information for the ones that `hpmcount` is following for a particular run.
4. Performance of miscellaneous hardware features. On POWER4 this report depends on the group that was selected at run time.

We do not describe each of the parameters printed by `hpmcount`, because most are self-explanatory and they are defined in the `hpmcount` manual. Instead, the following two examples illustrate how to use `hpmcount` for a simple performance tuning exercise. Example 9-7 on page 339 shows a simple program in which the double-nested loops have not been optimized (blocked) and therefore incurred misses in L2, L3, and TLB. In other words, no data can be reused.

Example 9-7 Double-nested loop without blocking

```
program reuse
c
    IMPLICIT NONE
    integer ARRAY_SIZE
    parameter(ARRAY_SIZE = 25000)
    integer I, J, II, JJ
    real A(ARRAY_SIZE, ARRAY_SIZE)
    real B(ARRAY_SIZE, ARRAY_SIZE)
    real S, SS
    integer(8) T1, T2, IRTC
C*****
C*                               Untuned Loop                               *
C*****
    T1 = IRTC()

    DO J=1, ARRAY_SIZE
        DO I=1, ARRAY_SIZE
            S = S + A(I,J)*B(J, I)
        ENDDO
    ENDDO

    T2 = IRTC()
    write(*,*)'Untuned loop took', (T2-T1)/1000000, 'msec'

C    Need to actually use the results of the calculations or else the
C    optimizing compiler may just skip doing them...so we'll just
C    print them. This will force the optimizer to actually do the work.
    print *, A(ARRAY_SIZE, ARRAY_SIZE), S

999  end
```

Example 9-8 shows a double-nested loop that has been optimized by blocking both loops. In this case there is data reuse because this example is keeping data in the different levels of cache. hpmcount helps show this.

Example 9-8 Tuned version of the reuse program

```

program reuse
c
    IMPLICIT NONE
    integer ARRAY_SIZE, NB
    parameter(ARRAY_SIZE = 25000, NB = 4)
    integer I, J, II, JJ
    real A(ARRAY_SIZE, ARRAY_SIZE), AA(ARRAY_SIZE, ARRAY_SIZE)
    real B(ARRAY_SIZE, ARRAY_SIZE), BB(ARRAY_SIZE, ARRAY_SIZE)
    real S, SS
    integer(8) T1, T2, IRTC
C*****
C*                               Tuned Loop                               *
C*****
    T1 = IRTC()

    DO JJ = 1, ARRAY_SIZE, NB
        DO II = 1, ARRAY_SIZE, NB
            DO J = J, JJ, MIN(ARRAY_SIZE, JJ+NB-1)
                DO I = II, MIN(ARRAY_SIZE, II+NB-1)
                    SS = SS + AA(I,J)*BB(J,I)
                ENDDO
            ENDDO
        ENDDO
    ENDDO

    T2 = IRTC()
    write(*,*)'Tuned loop took', (T2-T1)/1000000, 'msec'

C    Need to actually use the results of the calculations or else the
C    optimizing compiler may just skip doing them...so we'll just
C    print them. This will force the optimizer to actually do the work.
    print *, AA(ARRAY_SIZE, ARRAY_SIZE), SS

999  end

```

We have selected only part of the `hpmcount` output in both cases, mainly the section that illustrates the performance relating to L2, L3, and TLB using the two versions of the code. Example 9-9 shows the `hpmcount` command for groups 5 and 56.

Example 9-9 hpmcount output for groups 5 and 56

```

$ hpmcount -g 5 reuse_u
Execution time (wall clock time): 170.883827 seconds
PM_DATA_FROM_L3 (Data loaded from L3)           :          464363407
  PM_DATA_FROM_MEM (Data loaded from memory)     :          151160443
  PM_DATA_FROM_L35 (Data loaded from L3.5)      :           7981418
  PM_DATA_FROM_L2 (Data loaded from L2)         :          29723097
  PM_DATA_FROM_L25_SHR (Data loaded from L2.5 shared) :           0
  PM_DATA_FROM_L275_SHR (Data loaded from L2.75 shared) :           4
  PM_DATA_FROM_L275_MOD (Data loaded from L2.75 modified) :          16
  PM_DATA_FROM_L25_MOD (Data loaded from L2.5 modified) :           0

Total Loads from L2                               :          29.723 M
L2 load traffic                                   :        3628.310 MBytes
L2 load bandwidth per processor                   :          21.233 MBytes/sec
L2 Load miss rate                                 :          95.450 %
Total Loads from L3                               :          472.345 M
L3 load traffic                                   :        57659.280 MBytes
L3 load bandwidth per processor                   :          337.418 MBytes/sec
L3 Load miss rate                                 :          24.244 %
Memory load traffic                               :        18452.203 MBytes
Memory load bandwidth per processor               :          107.981 MBytes/sec

$ hpmcount -g 5 reuse_t
Execution time (wall clock time): 1.48904 seconds
PM_DATA_FROM_L3 (Data loaded from L3)           :           16
  PM_DATA_FROM_MEM (Data loaded from memory)     :           93
  PM_DATA_FROM_L35 (Data loaded from L3.5)      :           10
  PM_DATA_FROM_L2 (Data loaded from L2)         :          1167
  PM_DATA_FROM_L25_SHR (Data loaded from L2.5 shared) :           0
  PM_DATA_FROM_L275_SHR (Data loaded from L2.75 shared) :           10
  PM_DATA_FROM_L275_MOD (Data loaded from L2.75 modified) :           10
  PM_DATA_FROM_L25_MOD (Data loaded from L2.5 modified) :           0

Total Loads from L2                               :           0.001 M
L2 load traffic                                   :           0.145 MBytes
L2 load bandwidth per processor                   :           0.097 MBytes/sec
L2 Load miss rate                                 :           9.112 %
Total Loads from L3                               :           0.000 M
L3 load traffic                                   :           0.003 MBytes
L3 load bandwidth per processor                   :           0.002 MBytes/sec
L3 Load miss rate                                 :          78.151 %
Memory load traffic                               :           0.011 MBytes

```

Memory load bandwidth per processor : 0.008 MBytes/sec

\$ hpmcount -g 56 reuse_u

Execution time (wall clock time): 170.550834 seconds

PM_DTLB_MISS (Data TLB misses)	:	628854887
PM_ITLB_MISS (Instruction TLB misses)	:	149977
PM_LD_MISS_L1 (L1 D cache load misses)	:	930603848
PM_ST_MISS_L1 (L1 D cache store misses)	:	130067750
PM_CYC (Processor cycles)	:	231231424980
PM_INST_CMPL (Instructions completed)	:	20001760251
PM_ST_REF_L1 (L1 D cache store references)	:	2391568051
PM_LD_REF_L1 (L1 D cache load references)	:	7406257146
Utilization rate	:	93.272 %
%% TLB misses per cycle	:	0.272 %
number of loads per TLB miss	:	11.777
Total l2 data cache accesses	:	1060.672 M
%% accesses from L2 per cycle	:	0.459 %
L2 traffic	:	129476.513 MBytes
L2 bandwidth per processor	:	759.167 MBytes/sec
Total load and store operations	:	9797.825 M
number of loads per load miss	:	7.959
number of stores per store miss	:	18.387
number of load/stores per D1 miss	:	9.237
L1 cache hit rate	:	89.174 %
MIPS	:	117.277
Instructions per cycle	:	0.087

\$ hpmcount -g 56 reuse_t

Execution time (wall clock time): 1.488173 seconds

PM_DTLB_MISS (Data TLB misses)	:	452
PM_ITLB_MISS (Instruction TLB misses)	:	106
PM_LD_MISS_L1 (L1 D cache load misses)	:	1998
PM_ST_MISS_L1 (L1 D cache store misses)	:	4629
PM_CYC (Processor cycles)	:	2156932098
PM_INST_CMPL (Instructions completed)	:	1172053048
PM_ST_REF_L1 (L1 D cache store references)	:	237543426
PM_LD_REF_L1 (L1 D cache load references)	:	377625191
Utilization rate	:	99.710 %
%% TLB misses per cycle	:	0.000 %
number of loads per TLB miss	:	835453.962
Total l2 data cache accesses	:	0.007 M
%% accesses from L2 per cycle	:	0.000 %
L2 traffic	:	0.809 MBytes
L2 bandwidth per processor	:	0.544 MBytes/sec
Total load and store operations	:	615.169 M
number of loads per load miss	:	189001.597
number of stores per store miss	:	51316.359

number of load/stores per D1 miss	:	92827.617
L1 cache hit rate	:	99.999 %
MIPS	:	787.578
Instructions per cycle	:	0.543

The execution time reported for the program that has the double-nested loop that is not optimized (`reuse_u`) is approximately 180 seconds. The time for `reuse_t` (optimized version) is only approximately 2 seconds. For this trivial example, clearly this is an indication that there is a problem with that double-nested loop.

Further analysis of the `hpmcount` output reveals that the version that is not optimized is not taking advantage of the memory hierarchy. A simple inspection of some of the counters, such as: `PM_DATA_FROM_L3`, `PM_DATA_FROM_MEM`, `PM_DATA_FROM_L2`, `PM_LD_MISS_L1`, and `PM_ST_MISS_L1` indicates that the version that is not optimized is not reusing data and is incurring misses in a large number of caches.

9.3.2 Profiling utilities

This series of utilities is a set of tools that can help you find bottlenecks at a very fine-grained level. Scientific and engineering applications consist of both the actual code that performs simulations and system library calls and system-specific routines. Normally when an application has not been optimized, most of the CPU time is spent executing critical sections of the code, and these critical sections are normally localized to a subroutine or a few loops. These utilities are essential to finding these bottlenecks and filtering them from system or kernel-related software. Although there might be other utilities, this section focuses on `tprof`, `gprof`, and `xprofiler`.

tprof

In AIX 5L V3, **tprof** is part of the AIX Performance Toolbox. This utility reports CPU usage for both individual programs and the system as a whole. It is useful to identify sections of the code that are using CPU most heavily.

The raw data from **tprof** is obtained via the trace facility. In the following examples, **tprof** was used in conjunction with a scientific application, namely AMBER7. We previously introduced this life sciences application, so we recommend reviewing 3.5, “Simultaneous multithreading performance” on page 59 for an overview of AMBER7. In this example, we instrumented the `sander` module:

```
$ tprof -z -u -p PID -x read sander
```

Example 9-10 tprof output obtained with the sander module

Process	FREQ	Total	Kernel	User	Shared	Other
=====	====	=====	=====	=====	=====	=====
/scratch/amber7/exe/sander	1	6978	3	6975	0	0
wait	2	957	957	0	0	0
IBM.CSMAgentRMd	1	87	2	0	85	0
/home/db2inst1/sqllib/adm/db2set	2	2	1	0	1	0
/home/db2as/das/bin/db2fm	2	2	2	0	0	0
/home/db2inst1/sqllib/bin/db2fm	1	1	1	0	0	0
/usr/sbin/muxatmd	1	1	1	0	0	0
/usr/bin/sh	1	1	1	0	0	0
=====	====	=====	=====	=====	=====	=====
Total	11	8029	968	6975	86	0

Process	PID	TID	Total	Kernel	User	Shared	Other
=====	===	===	=====	=====	=====	=====	=====
ch/amber7/exe/sander	348388	774299	6978	3	6975	0	0
wait	8196	8197	838	838	0	0	0
wait	12294	12295	119	119	0	0	0
IBM.CSMAgentRMd	163856	544779	87	2	0	85	0
/db2as/das/bin/db2fm	274632	782505	1	1	0	0	0
/db2as/das/bin/db2fm	274620	606215	1	1	0	0	0
/usr/bin/sh	188626	786589	1	1	0	0	0
t1/sqllib/adm/db2set	188620	786583	1	0	0	1	0
/usr/sbin/muxatmd	225408	344283	1	1	0	0	0
t1/sqllib/adm/db2set	188614	733255	1	1	0	0	0
st1/sqllib/bin/db2fm	266376	639199	1	1	0	0	0
=====	===	===	=====	=====	=====	=====	=====
Total			8029	968	6975	86	0

Total Samples = 8029 Total Elapsed Time = 71.74s

Total Ticks For All Processes (USER) = 6975

User Process	Ticks	%	Address	Bytes
=====	=====	=====	=====	=====
/scratch/amber7/exe/sander	6975	86.87	10000150	116478

Profile: /scratch/amber7/exe/sander

Total Ticks For All Processes (/scratch/amber7/exe/sander) = 6975

Subroutine	Ticks	%	Source	Address	Bytes
=====	=====	=====	=====	=====	=====
._log	3816	47.53	ib/libm/POWER/logF.c	93490	380
._egb	2118	26.38	._egb_.f	d0ad0	4ea0
._exp	901	11.22	ib/libm/POWER/expF.c	16a70	2d0
._daxpy	89	1.11	daxpy.f	9c7c0	230

.ephi	18	0.22	_ene_.f	8bd30	21c0
._sin	13	0.16	ib/libm/POWER/sinF.c	ac90	200
._acos	6	0.07	b/libm/POWER/acosF.c	18470	2b0
.shake	5	0.06	_shake_.f	f42f0	9e0
._cos	4	0.05	ib/libm/POWER/cosF.c	aa70	220
.angl	4	0.05	_ene_.f	8a9b0	e00
.bond	1	0.01	_ene_.f	89330	e80

Profiling the sander module means that the trace facility is activated and instructed to collect from the trace hook that records the contents of the Instruction Address Register when a system-clock interrupt occurs. **tprof** reports the distribution of address occurrences as *ticks* across the programs involved in the workload.

Example 9-11 shows the output of **tprof** for the sander module. Note that one the three most time-consuming routines correspond to `logF.c`, `egb.f`, and `expF.c`. Furthermore, the two C routines are part of the `libm`, which can be easily replaced by the equivalent routine in the Mathematical Acceleration Subsystem (MASS) library.

Example 9-11 tprof output obtained with the sander module using MASS libraries

Process	FREQ	Total	Kernel	User	Shared	Other
===== /scratch/amber7/exe/sander_mass	1	5956	1	5954	1	0
wait	2	764	764	0	0	0
/home/db2inst1/sqllib/adm/db2set	1	1	1	0	0	0
===== Total	4	6721	766	5954	1	0

Process	PID	TID	Total	Kernel	User	Shared	Other
===== ber7/exe/sander_mass	356544	753849	5956	1	5954	1	0
wait	8196	8197	669	669	0	0	0
wait	12294	12295	95	95	0	0	0
t1/sqllib/adm/db2set	188632	786595	1	1	0	0	0
===== Total	===	===	6721	766	5954	1	0

Total Samples = 6721 Total Elapsed Time = 64.13s

Total Ticks For All Processes (USER) = 5954

User Process	Ticks	%	Address	Bytes
===== /scratch/amber7/exe/sander_mass	5954	88.59	10000150	114238

Profile: /scratch/amber7/exe/sander_mass

Total Ticks For All Processes (/scratch/amber7/exe/sander_mass) = 5954

Subroutine	Ticks	%	Source	Address	Bytes
=====	=====	=====	=====	=====	=====
._log	4066	60.50	ib/libm/POWER/logF.c	90590	380
.egb	1555	23.14	_egb_.f	cdbd0	4e60
vrsqrt	148	2.20	vrsqrt_p4.32s	7ef50	7a0
.daxpy	67	1.00	daxpy.f	998c0	230
.vexp	59	0.88	vexp_p4.32s	d2a30	8e8
.ephi	16	0.24	_ene_.f	88e30	21c0
._cos	13	0.19	ib/libm/POWER/cosF.c	aa70	220
._acos	11	0.16	b/libm/POWER/acosF.c	18470	2b0
.angl	9	0.13	_ene_.f	87ab0	e00
._sin	7	0.10	ib/libm/POWER/sinF.c	ac90	200
.runmd	3	0.04	_runmd_.f	e9a30	63c0

Here we see that one of the bottlenecks (the one coming from the expF.c routine) has totally disappeared. We now show similar information with **gprof** and **xprofiler**.

gprof

This utility enables you to look at code to identify its critical sections. **gprof** was developed by GNU. The following steps are required to use **gprof**:

- ▶ Compile and link the application with profiling enabled.
- ▶ Run the application to generate a profile data file.
- ▶ Run **gprof** to analyze the data.

Rather than illustrating the use of **gprof** or **tprof** with a trivial example, we apply it to the life sciences application AMBER7. The first step involves modifying the script that compiles and builds the sander module. This is seen where we included the -pg option.

Tip: Ensure that -pg is also included when the loader is invoked.

This corresponds to Machine.ibm_aix modified to invoke either **gprof** or the **xprofiler**:

```
##### LOADER/LINKER:
# Use Standard options
setenv LOAD "xlf90 -bmaxdata:0x80000000 -pg "
# Load with the IBM MASS & ESSL libraries
setenv LOADLIB " "
if ( $HAS_MASSLIB == "yes" ) setenv LOADLIB "-L$MASSLIBDIR -lmassvp4 "
if ( $VENDOR_BLAS == "yes" ) setenv LOADLIB "$LOADLIB -lblas "
```

```

if ( $VENDOR_LAPACK == "yes" ) setenv LOADLIB "$LOADLIB -lessl "

# little or no optimization:
setenv L0 "xlf90 -qfixed -c -pg"

# modest optimization (local scalar):
setenv L1 "xlf90 -qfixed -O2 -c -pg"

# high scalar optimization (but not vectorization):
setenv L2 "xlf90 -qfixed -O3 -pg -qmaxmem=-1 -qarch=auto -qtune=auto -c"

# high optimization (may be vectorization, not parallelization):
setenv L3 "xlf90 -qfixed -O3 -pg -qmaxmem=-1 -qarch=auto -qtune=auto -c"

```

Example 9-11 on page 345 shows the output of **tprof** after replacing `expF.c` with the vectorized version in the MASS library.

Example 9-12 Results of replacing `expF.c` with vectorized version

ngranularity: Each sample hit covers 4 bytes. Time: 90.27 seconds

index	%time	self	descendents	called/total called+self called/total	parents name children	index
[1]	89.6	0.00	80.92	1/1	._start [2]	
		0.00	80.92	1	.main [1]	
		0.00	80.92	1/1	.runmd [3]	
		0.00	0.00	1/1	.rdparm2 [26]	

6.6s					<spontaneous>	
[2]	89.6	0.00	80.92	1/1	._start [2]	
		0.00	80.92	1/1	.main [1]	

[3]	89.6	0.00	80.92	1/1	.main [1]	
		0.00	80.92	1	.runmd [3]	
		0.01	80.89	100/100	.force [4]	
		0.02	0.00	100/100	.shake [20]	
		0.00	0.00	1/366954097	._log [6]	
		0.00	0.00	300/1429	.timer_start [37]	

[4]	89.6	0.01	80.89	100/100	.runmd [3]	
		0.01	80.89	100	.force [4]	
		26.21	54.04	100/100	.egb [5]	
		0.20	0.29	200/200	.ephi [11]	
		0.04	0.09	200/200	.ang1 [15]	
		0.02	0.00	100/100	.bond [19]	
		0.00	0.00	1300/1300	.get_stack [40]	

```

                26.21    54.04    100/100    .force [4]
[5]    88.9    26.21    54.04    100    .egb [5]
                43.93    0.00    366954096/366954097    ._log [6]
                10.11    0.00    99723844/99723844    ._exp [7]
                0.00    0.00    225/1429    .timer_start [37]
                0.00    0.00    225/1429    .timer_stop [38]
-----
                0.00    0.00    1/366954097    .runmd [3]
                43.93    0.00    366954096/366954097    .egb [5]
[6]    48.7    43.93    0.00    366954097    ._log [6]
-----

                10.11    0.00    99723844/99723844    .egb [5]
[7]    11.2    10.11    0.00    99723844    ._exp [7]
-----

6.6s
[8]    8.2    7.42    0.00    <spontaneous>
                ._mcount [8]
-----

6.6s
[9]    0.8    0.70    0.00    <spontaneous>
                .daxpy [9]
-----

6.6s
[10]    0.7    0.62    0.00    <spontaneous>
                .qincrement [10]
-----

[11]    0.5    0.20    0.29    200/200    .force [4]
                0.20    0.29    200    .ephi [11]
                0.19    0.00    1618200/2075531    ._sin [14]
                0.06    0.00    809100/1266400    ._acos [17]
                0.05    0.00    1618200/1618231    ._cos [18]
-----

6.6s
[12]    0.3    0.25    0.00    <spontaneous>
                .__stack_pointer [12]
-----

6.6s
[13]    0.3    0.25    0.00    <spontaneous>
                .qincrement1 [13]
-----

                0.00    0.00    31/2075531    .dihpar [25]
                0.05    0.00    457300/2075531    .angl [15]
                0.19    0.00    1618200/2075531    .ephi [11]
[14]    0.3    0.24    0.00    2075531    ._sin [14]
-----
.....
-----
ngranularity: Each sample hit covers 4 bytes. Time: 90.27 seconds

```

```

%    cumulative    self            self     total

```

time	seconds	seconds	calls	ms/call	ms/call	name
48.7	43.93	43.93	366954097	0.00	0.00	._log [6]
29.0	70.14	26.21	100	262.10	802.50	.egb [5]
11.2	80.25	10.11	99723844	0.00	0.00	._exp [7]
8.2	87.67	7.42				._mcount [8]
0.8	88.37	0.70				.daxpy [9]
0.7	88.99	0.62				.qincrement [10]
0.3	89.24	0.25				._stack_pointer [12]
0.3	89.49	0.25				.qincrement1 [13]
0.3	89.73	0.24	2075531	0.00	0.00	._sin [14]
0.2	89.93	0.20	200	1.00	2.47	.ephi [11]
0.1	90.03	0.10				.EndIORWFmt [16]
0.1	90.12	0.09	1266400	0.00	0.00	._acos [17]
0.1	90.17	0.05	1618231	0.00	0.00	._cos [18]
0.0	90.21	0.04	200	0.20	0.63	.angl [15]
0.0	90.23	0.02	100	0.20	0.20	.bond [19]
0.0	90.25	0.02	100	0.20	0.20	.shake [20]
0.0	90.26	0.01	15149	0.00	0.00	.cvtloop [22]

xprofiler

This utility is a graphical tool that enables you to perform profiling of your code. The output is similar to the other two tools, except that **xprofiler** is more flexible and more powerful for analyzing large applications. The procedure to build and run the application with **xprofiler** enabled is similar to **gprof**. After this is done, just invoke **xprofiler**:

```
$ xprofiler
```

When **xprofiler** has started, all you have to do to start displaying information related to your application is to select your executable and the gmon.out file.

xprofiler can also produce a flat profile as part of its functionality. Figure 9-5 on page 350 shows the flat profile for the AMBER7 run.

<u>File</u>	<u>Code Display</u>	<u>Utility</u>					<u>Help</u>
%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name	
48.7	43.93	43.93	366954097	0.00	0.00	._log [6]	
29.0	70.14	26.21	100	262.10	802.50	.egb [5]	
11.2	80.25	10.11	99723844	0.00	0.00	._exp [7]	
8.2	87.67	7.42				.__mcount [8]	
0.8	88.37	0.70				.daxpy [9]	
0.7	88.99	0.62				.qincrement [10]	
0.3	89.24	0.25				.__stack_pointer [12]	
0.3	89.49	0.25				.qincrement1 [13]	
0.3	89.73	0.24	2075531	0.00	0.00	._sin [14]	

Search Engine: (regular expressions supported)

Figure 9-5 Flat profile produced using xprofiler

Of the tools that can be utilized to analyze the performance of scientific and engineering applications, **xprofiler** provides a powerful utility with a friendly interface that can help expedite the localization of critical sections in the code. If there is no access to the source code, **tprof** provides a good alternative for obtaining information about a running application by attaching to the process ID.

9.4 Memory management

In this section explore the importance of data locality to code optimization. Figure 9-6 on page 351 shows how memory hierarchy is organized on POWER5. Previous chapters have provided detailed descriptions of each of the components.

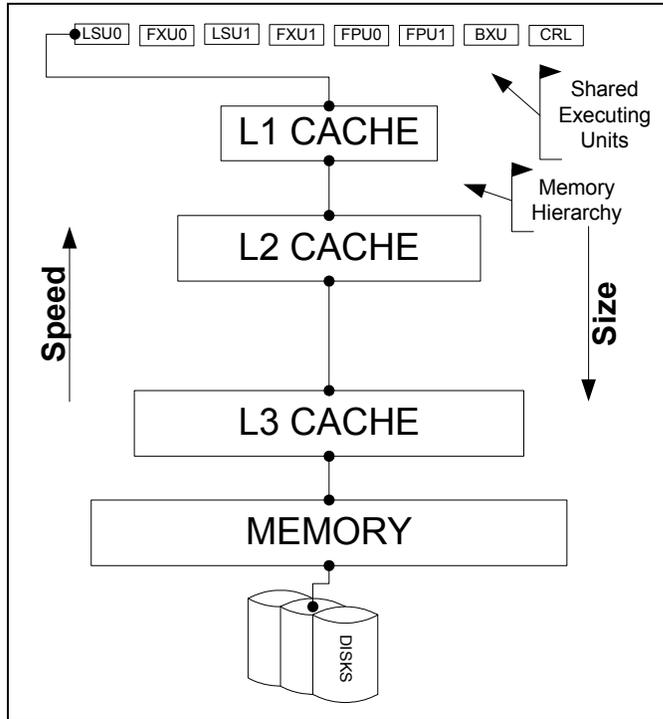


Figure 9-6 Memory hierarchy

The POWER architecture has several layers where your application can access data. The important message from Figure 9-6 that we demonstrate throughout this chapter is that the closer the data is to the registers, the more efficiently your application will perform.

Scientific and engineering applications are floating-point intensive programs. The architectural features that are of the highest direct relevance from the figure are:

- ▶ All three levels of cache
- ▶ Translation lookaside buffer (not shown in Figure 9-6)

Data prefetch streaming and superscalar floating-point units also affect memory performance.

9.5 Optimization of critical sections in the code

This is a very extensive topic and we do not attempt to cover all of the possible techniques to optimize scientific and engineering applications. Instead we focus our attention on certain techniques that tend to benefit the POWER architecture.

A good example of this is matrix multiplication. In the past, we managed to achieve approximately 60% of peak performance via matrix multiplication routines. Now with the advent of POWER5, which has more rename registers, it is possible to achieve near-peak performance with a well-optimized routine.

In this section we try to illustrate some simple techniques using standard examples and an example of matrix multiplication. We look at matrix multiplication from an over-simplistic point of view, which does not show good performance, but from this example we provide a Fortran version that compares well with a highly optimized version of matrix multiplication routines.

Before doing that, it is important to recall a series of general optimization rules that can improve the performance of critical sections of the code within scientific and engineering applications. To make better use of memory, keep in mind that whenever possible data should be accessed sequentially. In a loop, this is called accessing memory with unity stride or stride of 1. In large applications, loops usually tend to grow in size as operations are carried out within the loop. If possible, try to keep the loop small and manageable—the smaller the loop the better. It is important to avoid expensive operations, such as divide, square-root, and exponential. If this type of transcendental function is required for the code, consider using the MASS library, which is explained in 9.6.1, “MASS Library” on page 361. In a loop, *if* statements and calls to subroutines tend to introduce data dependency and usually inhibit optimization. The following are less common problems that we only mention here:

- ▶ Avoid using EQUIVALENCE for critical variables.
- ▶ Avoid implicit type conversions.
- ▶ Try to reduce the number of arguments that are passed from the caller to the callee.
- ▶ If multiple “if” statements are required, evaluate the most likely if statement first. In other words, try to reduce the number of if statements that have to be evaluated.

In general, the key to performance is to be able to map the application as close as possible to the POWER Series architecture. The use of the POWER5 memory hierarchy can be cleverly manipulated in an algorithm to gain efficiency. This may include prefetching to facilitate accessing memory that is currently not in the cache. Prefetching provides a mechanism for hiding memory latency due to cache misses. We shall see that loop unrolling is very important to proper use of the memory hierarchy. Simulation of higher precision arithmetic helps performance, especially when this process is highly repetitive.

9.5.1 General rules for optimization strategies

Code simplification:

- ▶ Eliminate unused or redundant computations.
- ▶ Use algebraic identities, when possible, to simplify expressions.
- ▶ Eliminate unnecessary branching.
- ▶ Move code to less-frequently executed points.
- ▶ Eliminate unnecessary procedure calls and pointer indirections.

Data memory cost:

- ▶ Eliminate redundant memory loads and stores.
- ▶ Data that is constantly used together should be stored in memory as close together as possible.
- ▶ Reorganize loop structures to exploit data reuse and locality.
- ▶ Perform careful mapping of data to avoid cache and TLB interference.
- ▶ Overlap memory access and computation through software or hardware prefetch.

Instruction memory cost:

- ▶ Reduce code size when possible.
- ▶ Limit inlining and loop unrolling to avoid excessive code growth.
- ▶ Maintain code for loops together and move non-loop code out.
- ▶ Move branches and their targets closer together.

Multiple instructions:

- ▶ Identify loops whose iterations can profitably run in parallel, and execute them concurrently using a run-time schedule.
- ▶ Find primitive operations in loops that can be vectorized profitably, such as divide and square root, and compute the vectors in a pipeline.
- ▶ Balance loop computations through loop unrolling to enable effective software pipelining.

9.5.2 Array optimization

To take advantage of the memory hierarchy on POWER architected systems, the first step requires understanding the multiple arrays used in the code and how their elements will be used. In general, this is particularly true for numerically

intensive applications that spend a large amount of CPU performing repetitive tasks, such as loading and storing data in arrays.

As in any popular Fortran or C textbook, we start by describing the difference in the way elements in a matrix are stored between these two languages. For C codes, matrix rows are stored contiguously. For Fortran codes, matrix columns are stored contiguously.

4x4 Matrix with one-dimensional index in square brackets:

$$\begin{bmatrix} a(1, 1)[1] & a(1, 2)[2] & a(1, 3)[3] & a(1, 4)[4] \\ a(2, 1)[5] & a(2, 2)[6] & a(2, 3)[7] & a(2, 4)[8] \\ a(3, 1)[9] & a(3, 2)[10] & a(3, 3)[11] & a(3, 4)[12] \\ a(4, 1)[13] & a(4, 2)[14] & a(3, 4)[15] & a(4, 4)[16] \end{bmatrix}$$

C: row-major order; matrix rows are stored contiguously

$$[1 | 2| 3 |4| 5 |6| 7 |8| 9 |10| 11 |12| 13 |14| 15| 16]$$

Fortran: column-major order; matrix columns are stored contiguously

$$[1 | 5| 9 |13| 2 |6| 10 |14| 3 |7| 11 |15| 4 |8| 12| 16]$$

Important: Keep this ordering in mind to help the efficiency of your code.

Stride is relevant when addressing the elements of a matrix.

Stride The distance between successively accessed matrix elements in successive loop iterations

Example 9-13 Do loop with two different strides

```
do i = 1, 200
  do j = 1, 500
    a(i,j) = 1.d0    !stride 500
    b(j,i) = 1.d0    !stride 1
  enddo
enddo
```

Tip: Stride of 1 ensures sequential access to memory and provides best performance.

Stride of 1 is beneficial because it ensures that after a line of cache has been loaded, the next set of elements needed will be available in the same cache line. This decreases the number of times that is required to go to memory hierarchy to load data. On POWER5, the size of a cache line is 128 bytes. The implication of this size is that for 32-bit words, strides larger than 32 will reduce performance (and 16 for 64-bit words) since only one element can be fetched per cache line. In addition, stride 1 will make use of the prefetch engine.

9.5.3 Loop optimization

The technique of loop optimization is basic for taking advantage of memory hierarchy. A good example that is commonly used to illustrate multiple optimization techniques is matrix multiplication. Optimizing the matrix multiplication triple-nested loop has been discussed extensively, so we simply summarize how this is done and show results with respect to POWER5.

Example 9-14 shows the typical triple-nested loop of the main kernel in a matrix multiplication routine. The triple-nested loop comes from the following expression:

$$[C] = [C] + [A]^T [B]$$

Example 9-14 Matrix multiplication triple-nested loop

```
do i = 1, n
  do j = 1, n
    do k = 1, n
      c(i,j) = c(i,j) + a(j,k)*b(k,i)
    enddo
  enddo
enddo
```

This loop has been written without any particular attention to an optimal arrangement of the loop indices. In fact, optimizing all three loops for stride 1 may not be possible. This makes matrix multiplication a good candidate for loop unrolling.

Example 9-15 illustrates an implementation of a 2x2 loop unrolling.

Example 9-15 Matrix 2x2 loop unrolling

```
do i = 1, 1, 2
  do j = 1, m, 2
    s00 = zero
    s21 = zero
    s12 = zero
    s22 = zero
c
    do k = 1, n
      s11 = s11 + a(k,i)*b(k,j)
      s21 = s21 + a(k,i+1)*b(k,j)
      s12 = s12 + a(k,i)*b(k,j+1)
      s22 = s22 + a(k,i+1)*b(k,j+1)
    enddo
    c(i,j) = c(i,j) + s11
    c(i+1,j) = c(i+1,j) + s21
    c(i,j+1) = c(i,j+1) + s12
    c(i+1,j+1) = c(i+1,j+1) + s22
  enddo
enddo
```

Table 9-9 shows the level of unrolling that has been reported as optimal for different POWER series, including our findings that seem to work well for POWER5.

Table 9-9 Optimal unrolling levels for some IBM POWER Series machines

System	Unrolling level
POWER1	2x2
POWER2™	4x4
POWER3	4x4
POWER4	5x4
POWER5	4x4

The POWER1 had only a single floating-point unit, the 2x2 unrolling, which achieved very good performance. The POWER2 and POWER3 have dual floating-point units; for these systems, 4x4 appeared to be favored. The unrolling reported for POWER4 is 4x5.

In addition to unrolling, block matrices must be able to make use of registers as much as possible.

In order to carry out our performance measurements, we used a hand-tuned version of DGEMM. This version is not publicly available. Table 9-10 on page 357 illustrates the performance of the different level of unrolling. Column 1, which is used as reference, corresponds to a well-coded DGEMM version without unrolling. The next set of columns illustrate the impact of the various unrolling levels on POWER5. Note that in all of these cases, the blocking size was kept constant at 256. The only variable was unrolling. The main difference between 4x4 unrolling and @server p5-4x4 unrolling is in the use of two extra compiler directives in @server p5-4x4. One is to further unroll the innermost loop of the matrix multiplication and the second one is to enhance prefetching.

A simple inspection of Table 9-10 shows that 4x4 is the technique that is best suited for POWER5. @server p5-4x4 shows the additional benefit of using compiler directives. The recommended version for square matrices is @server p5-4x4.

Table 9-10 Performance of matrices with different unrolling levels

Dimension	No unrolling	2X2	5X4	4X4	4X4 new
100	1,580	3,080	3,938	4,763	4,902
200	1,786	3,583	4,141	5,340	5,486
500	1,797	3,732	3,906	5,106	5,691
1000	1,848	3,835	3,522	5,166	5,944
2000	1,812	3,754	3,731	5,224	5,837
5000	1,355	3,622	3,809	5,320	6,028

Figure 9-7 shows the different techniques that were used to unroll the matrix. The reference is the peak performance for this POWER5 running at 1.65 GHz. Again, it is easy to see that @server p5-4x4 shows the best performance and is remarkably close to peak performance. This is because POWER5 introduces more rename registers.

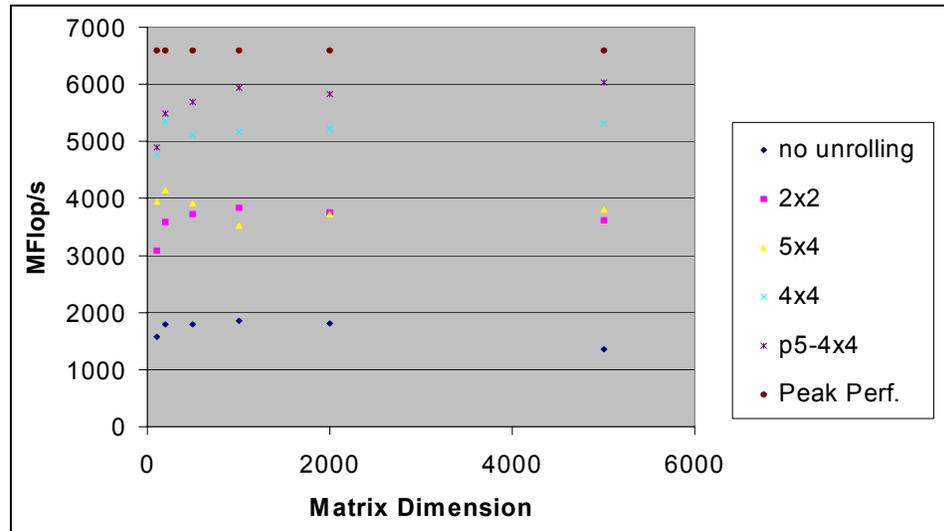


Figure 9-7 Different techniques against peak performance

Next we examine the effect of blocking and large pages on matrix multiplication. We looked at block sizes of 32, 64, 96, 128, 160, 192, 224, and 256. Table 9-11 summarizes the performance of a matrix multiplication for a subset of these sizes.

Table 9-11 Blocking and large pages' effect on matrix multiplication

Dimension	32	64	128	256
100				
Small pages	4,560	4,596	3,920	3,795
Large pages	4,588	4,880	4,917	4,902
500				
Small pages	4,850	5,080	5,375	5,363
Large pages	5,030	5,232	5,679	5,691
1000				

Dimension	32	64	128	256
Small pages	4,050	4,751	5,396	5,609
Large pages	4,391	5,103	5,843	5,944
2000				
Small pages	3,629	4,810	5,249	5,483
Large pages	4,190	5,365	5,702	5,837
5000				
Small pages	3,401	4,498	5,143	5,350
Large pages	4,068	5,082	5,906	6,028
10000				
Small pages	3,448	4,490	4,923	5,098
Large pages	4,207	5,372	5,546	5,900

Figure 9-8 shows the effect of different block sizes for three matrices with three different dimensions. The dimensions that we have selected for the three square matrices correspond to: 100x100, 1000x1000, and 10000x10000. From this plot we have identified that for all of the matrices tested here, block sizes between 224 and 256 give the best results.

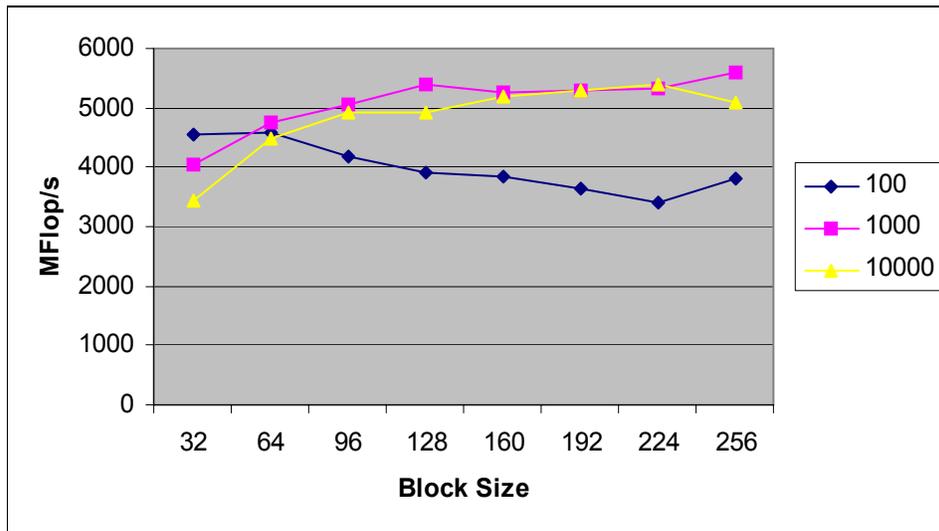


Figure 9-8 Matrix multiplication as a function of block size

The last figure in this section illustrates the effect of large pages on matrix multiplication. Figure 9-9 shows that independent of the block size and dimensions of the matrix, the large pages effect is proportional to the dimensions of the matrixes. Although not shown in the table nor in the figure, we found that the improvement measured in percentage difference (in going from small to large pages) for matrixes was: 100x100, 200x200, 500x500, 1000x1000, 2000x200, 5000x5000, and 10000x10000 are 1%, 2%, 4%, 8%, 15%, 20%, and 22%, respectively.

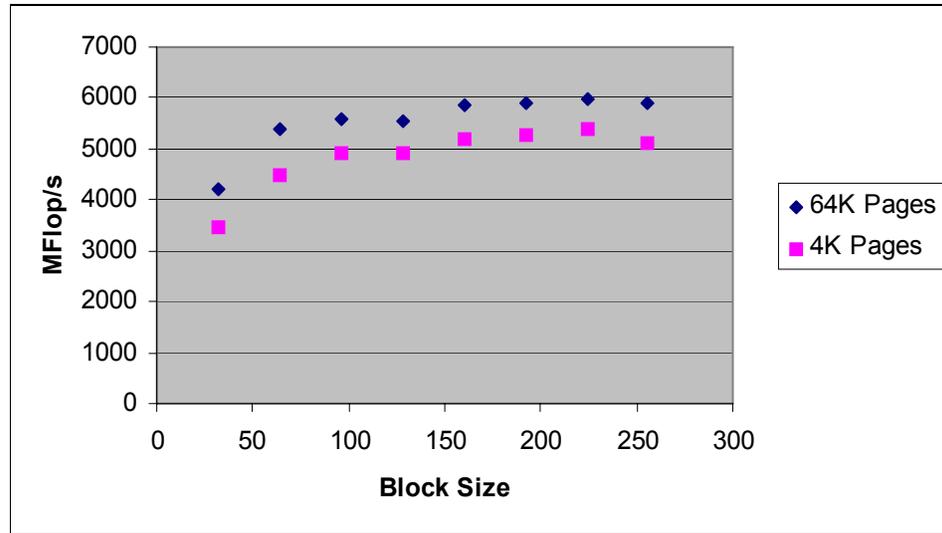


Figure 9-9 Large pages effect on performance for matrix multiplication

9.6 Optimized libraries

IBM provides collections of routines that have been fully optimized to a particular architecture, in this case POWER5. As previously mentioned, the advantage of using highly tuned libraries is that in many cases the code requires few changes to take full advantage of these library functions. In this section, we show how scientific applications can be customized to fully utilize these libraries. In this section we cover two of them:

- ▶ MASS library:

<http://www.ibm.com/support/docview.wss?uid=swg24007650>

- ▶ ESSL library:

http://publib.boulder.ibm.com/c/resctr/windows/public/esslbooks.html#essl_42

9.6.1 MASS Library

Figure 9-1 on page 313 presented a flowchart that illustrates how to analyze applications performance. (A section is reproduced in Figure 9-10.) As part of the steps in this flowchart, we mentioned that prior to attempting any hand tuning, the programmer should rely on compiler flags, compiler directives and *highly optimized libraries*. This chapter starts with the MASS libraries.

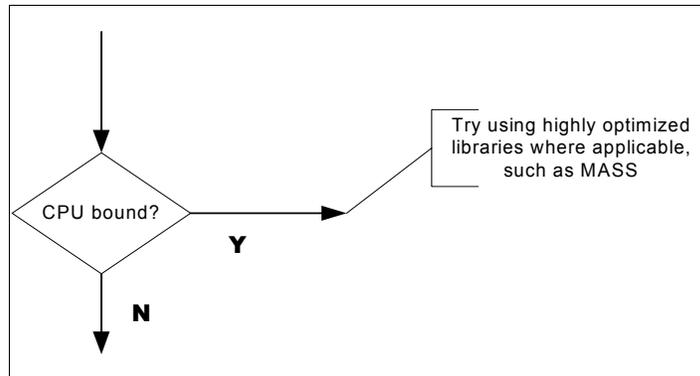


Figure 9-10 CPU-bound code might benefit from optimized libraries

MASS is a set of libraries of highly tuned mathematical intrinsic functions for C, C++, and Fortran applications that are optimized for specific POWER architectures. The MASS library consists of the scalar and vector libraries. The MASS scalar library, libmass.a, contains an accelerated set of the most frequently used math intrinsic functions in the AIX 5L system library libxlf90.a. This library can be used under AIX 5L and Linux on the POWER5 family.

Important: In some cases MASS is not as accurate as the system library and it might handle certain cases differently. We recommend always checking answers when using any kind of optimized libraries for the first time.

The second set of libraries correspond to the MASS vector library. The general vector libraries, libmassv.a, libmassvp3.a (for POWER3), and libmassvp4.a (for POWER4) contain functions that have been optimized. Table 9-12 on page 362 reproduces the vector table presented in:

<http://www.ibm.com/support/docview.wss?rs=2021&context=SSVKBV&uid=swg27005374>

It includes POWER5 performance.

The vector libraries libmassv.a, libmassvp3.a, and libmassvp4.a can be used with either FORTRAN or C applications. When calling the library functions from C, only call by reference is supported, even for scalar arguments. As with the

scalar functions, the vector functions must be called with the IEEE rounding mode set to round-to-nearest and with exceptions masked off. The accuracy of the vector functions is comparable to that of the corresponding scalar functions in libmass.a, though results may not be bit-wise identical.

The MASS vector Fortran source library enables application developers to write portable vector codes. The source library, libmassv.f, includes Fortran versions of all vector functions in the MASS vector libraries. The following performance table lists estimates of the number of processor cycles per evaluation of a vector element for the various MASS vector libraries. The estimates used vectors of length 1000 so that the caches contain all the vectors. The columns labeled libm give the results from using the functions in the MASS Fortran source code library libmassv.f to call the functions in libxlf90.a. The Fortran source code was compiled with IBM XLF compiler using the -O option. The columns labeled mass show results of the same process, except they use libmass.a instead of libxlf90.a. The columns labeled massv and vp4 list the results obtained with the libraries libmassv.a and libmassvp4.a, respectively. Times are not given for functions in the libmassvp4.a libraries that are identical to the functions in the libmassv library. The results were measured on both POWER4 and POWER5 systems.

Results will vary with vector length. Entries in the table where the library function does not exist or where the measurement was not done are blank.

Table 9-12 Vector library performance (cycles per evaluation, length 1000 loop)

Function	Range	libm	POWER4			POWER5			
			mass	massv	vp4	libm	mass	massv	vp4
vrec	D	29*		11	5.1	29*		11	5.1
vsrec	D	28*		7.6	3.9	23*		6.4	3.8
vdiv	D	28*		14	5.8	29*		13	5.4
vsdiv	D	30*		9.4	4.9	23*		8.8	4.8
vsqrt	C	36*		17	7.1	36*		17	7.8
vssqrt	C	36*		11	5.7	23*		10	5.4
vrsqrt	C	64*		18	7.0	36*		17	8.0
vsrsqrt	C	64*		11	5.6	23*		9.4	6.1
vexp	D	91	42	14	11	168	47	12	11
vsexp	D	105	45	10	8.9	192	50	8.9	8.6
vlog	C	153	84	9.9		207	86	9.5	

			POWER4			POWER5			
vslog	C	157	90	6.7		212	92	6.9	
vlog10	C	158	90	10		207	93	9.8	
vslog10	C	162	97	6.9		206	95	6.9	
vsin	B	56	24	7.2	14	66	24	6.8	13
vsin	D	75	69	20	15	83	64	19	15
vssin	B	57	26	5.5	11	67	25	5.2	10
vssin	D	80	72	16	13	86	66	16	12
vcos	B	54	23	6.6	15	66	22	6.3	13
vcos	D	80	69	20	17	82	64	20	15
vscos	B	57	27	5.4	11	66	24	4.9	10
vscos	D	81	73	16	12	82	66	16	12
vsincos	B	103	51	13	10	122	46	12	9.3
vsincos	D	157	135	22	18	166	122	22	18
vssincos	B	110	54	9.8	8.3	124	147	9.5	7.2
vssincos	D	162	138	18	15	180	124	17	15
vcosisin	B	105	50	12	10	123	45	12	9.4
vcosisin	D	160	134	21	18	165	119	20	18
vscosisin	B	107	51	9.9	8.3	124	47	9.7	7.2
vscosisin	D	158	136	18	15	172	125	18	15
vtan	D	172	73	19		185	67	18	
vstan	D	192	76	15		187	68	15	
vatan2	D	722	139	59	24	64	136	48	25
vsatan2	D	738	146	48	15	772	142	47	15
vcosh	D	195	53	14		242	54	13	
vscosh	E	176	55	13		244	56	12	
vsinh	D	275	68	15		372	67	13	
vssinh	E	293	73	14		359	74	12	

			POWER4			POWER5			
vtanh	F	307	80	19		329	78	18	
vstanh	E	282	84	17		355	85	15	
vpow	C	399	185	29		462	189	30	
vspow	G	396	190	17		462	193	17	
vasin	B	97		24		108		23	
vsasin	B	103		14		112		14	
vacos	B	104		24		108		23	
vsacos	B	107		14		114		14	
vexpm1	D	169		12		197		12	
vsexpm1	E	140		10		214		10	
vlog1p	H	202		13		221		12	
vslog1p	H	202		9.2		219		8.9	
vdint	D	40		6.6		46		6.4	
vdnint	D	39		8.9		43		7.8	
* hardware instructions (in simple loop)									
Range key: A = 0,1 B = -1,1 C = 0,100 D = -100,100 E = -10,10 F = -20,20 G = 0,10 H = -1,100									
1500 MHz POWER4 (GQ)									
1650 MHz POWER5 (GR)									

Accuracy data for the scalar and vector libraries

In some cases, MASS is not as accurate as the system library libm.a, and it may handle edge cases differently (sqrt(Inf)), for example). Table 9-13 provides sample accuracy data for the libm, libmass, libmassv, and libmassvp4 libraries. The numbers are based on the results for 10,000 random arguments chosen in the specified ranges (except for some of the libmassvp4 functions that were tested more extensively; see range F in the table). Real*16 functions were used to compute the errors. There may be portions of the valid input argument range for which accuracy is not as good as illustrated in the table. Also, accuracies may vary from values in the table when argument values are used that are not represented in the table.

The entries in the percent correctly rounded (PCR) column were obtained by counting the number of correctly rounded results out of 10,000 samples with random argument. A result is correctly rounded when the function returns the IEEE 64-bit value that is closest to the exact (infinite-precision) result.

Table 9-13 MASS library accuracy (MASS 41)

function	range	libxlf90		libmass		libmassv		libmassvp3		libmassvp4	
		PCR	MaxE	PCR	MaxE	PCR	MaxE	PCR	MaxE	PCR	MaxE
rec	D	100.00*	.50*			100.00	.50	100.00	.50	99.95	.51
srec	D	100.00*	.50*			92.47	.66	99.97	.50	99.92	.50
div	D	100.00*	.50*			74.78	1.32	74.78	1.32	74.77	1.32
sdiv	D	100.00*	.50*			100.00	.50	.74.49	1.35	74.47	1.35
sqrt	A	100.00	.50	96.59	.58	96.42	.60	86.86	.96	86.86	.96
ssqrt	A	100.00	.50	100.00	.50	87.64	.79	83.80	1.01	83.80	1.01
rsqrt	A	88.52	.98	98.60	.54	97.32	.62	97.84	.55	97.84	.55
srsqrt	A	100.00	.50	100.00	.50	86.39	.82	89.66	.86	89.66	.86
exp	D	99.95	.50	96.55	.63	96.58	.63	96.58	.63	96.58	.63
sexp	D	100.00	.50	100.00	.50	98.87	.52	98.87	.52	98.87	.52
log	C	99.99	.50	99.69	.53	99.45	.89	99.45	.89	99.45	.89
slog	C	100.00	.50	100.00	.50	99.99	.50	99.99	.50	99.90	.50
log10	C	64.63	1.56	64.58	1.56	99.29	1.03	99.29	1.03	99.29	1.03
slog10	C	100.00	.50	100.00	.50	99.99	.50	99.99	.50	99.99	.50
sin	B	81.31	.91	96.88	.80	97.28	.72	97.28	.72	92.87	1.35

function	range	libxl90		libmass		libmassv		libmassvp3		libmassvp4	
sin	D	86.03	.94	83.88	1.36	83.85	1.27	83.85	1.27	83.07	1.33
ssin	B	100.00	.50	100.00	.50	99.95	.50	99.95	.50	99.85	.51
ssin	D	100.00	.50	100.00	.50	99.73	.51	99.73	.51	99.73	.51
cos	B	92.95	1.02	92.20	1.00	93.19	.88	91.19	.88	86.15	1.11
cos	D	86.86	.93	84.19	1.33	84.37	1.33	84.37	1.33	83.17	1.33
scos	B	100.00	.50	100.00	.50	99.35	.51	99.35	.51	99.57	.51
scos	D	100.00	.50	100.00	.50	99.82	.51	99.82	.51	99.82	.51
tan	D	99.58	.53	64.51	2.35	54.31	2.71	54.31	2.71	54.31	2.71
stan	D	100.00	.50	100.00	.50	98.11	.68	98.11	.68	98.11	.68
atan2	D	74.66	1.59	86.02	1.69	84.01	1.67	84.01	1.67	84.00	1.67
satan2	D	100.00	.50	100.00	.50	100.00	.50	100.00	.50	98.76	.62
cosh	D	95.64	.97	92.73	1.04	57.56	2.09	57.56	2.09	57.56	2.09
scosh	E	100.00	.50	100.00	.50	99.08	.52	99.08	.52	99.08	.52
sinh	D	94.78	1.47	98.54	1.45	82.53	1.58	82.53	1.58	82.53	1.58
ssinh	E	100.00	.50	100.00	.50	98.75	.53	98.75	.53	98.75	.53
tanh	F	96.97	2.53	91.30	1.85	58.57	2.98	58.57	2.98	58.57	2.98
stanh	E	100.00	.50	100.00	.50	89.44	.74	89.44	.74	89.44	.74
pow	C	99.95	.50	96.58	.63	97.04	.58	97.04	.58	97.04	.58
spow	G	100.00	.50	100.00	.50	99.16	.52	99.16	.52	99.16	.52
acos	B	99.44	.59			84.72	1.85	84.72	1.85	84.72	1.85
sacos	B	100.00	.50			99.06	.55	99.06	.55	99.06	.55
asin	B	98.82	.61			68.52	1.95	68.52	1.95	68.52	1.95
sasin	B	100.00	.50			97.66	.56	97.66	.56	97.66	.56
expm1	D	95.58	.98			98.58	.98	98.58	.98	98.58	.98
sexpm1	E	100.00	.50			100.00	.50	100.00	.50	100.00	.50
log1p	H	99.91	.97			99.56	1.29	99.56	1.29	99.56	1.29
slog1p	H	100.00	.50			100.00	.50	100.00	.50	100.00	.50
dint	D	100.00	.00			100.00	.00	100.00	.00	100.00	.00
dnint	D	100.00	.00			100.00	.00	100.00	.00	100.00	.00

function	range	libxl90		libmass		libmassv		libmassvp3		libmassvp4	
atan	B	99.82	.51	92.58	1.78						
atan	D	99.98	.50	98.86	1.72						
* Indicates hardware instruction was used. PCR = percentage correctly rounded MaxE = Maximum observed error in ulps											
Range key:		D = -100, 100 A = 0, 1 B = -1, 1 C = 0, 100									
		E = -10, 10 F = -20, 20 G = 0, 10									

For information about performance for each of the MASS library functions, visit the MASS library page at:

<http://www.ibm.com/support/docview.wss?uid=swg24007650>

Example 9-16 shows how the vector MASS library can be implemented for a scientific application such as AMBER, replacing `1/sqrt` with `vrsqrt`.

Example 9-16 MASS vector library example

```

icount = 0
  do 25 j=i+1,natom
c
    xij = xi - x(3*j-2)
    yij = yi - x(3*j-1)
    zij = zi - x(3*j )
    r2 = xij*xij + yij*yij + zij*zij
    if( r2.gt.cut ) go to 25
c
    icount = icount + 1
    jj(icount) = j
    r2x(icount) = r2
c
  25 continue
c
c
c
#ifdef MASSLIB
  call vrsqrt( vectmp1, r2x, icount )
#else
  do j=1,icount
    vectmp1(j) = 1.d0/sqrt(r2x(j))
  end do
#endif

```

Table 9-14 shows the performance improvement from using the MASS libraries. The case presented in this table corresponds to the Generalized Born myoglobin simulation. This protein has 2,492 atoms and is run with a 20Å cutoff and a salt concentration of 0.2 M, with nrespa=4 (long-range forces computed every 4 steps.)

Table 9-14 AMBER7 performance with the sqrt vector MASS routine

Elapsed time in seconds	
Without vector MASS	With vector MASS
68	79

In this particular example, the MASS libraries are used in only three locations in the routine that is using most of the CPU time, one time for exp() and two for sqrt(). The table illustrates that by performing these simple substitutions there is an almost 15% improvement in single processor performance.

9.6.2 ESSL library

In this section we look at performance improvements using highly optimized library routines, in this case the IBM Engineering and Scientific Subroutine Library (ESSL). Find more information at the ESSL Web site at:

<http://publib.boulder.ibm.com/c/resctr/windows/public/esslbooks.html>

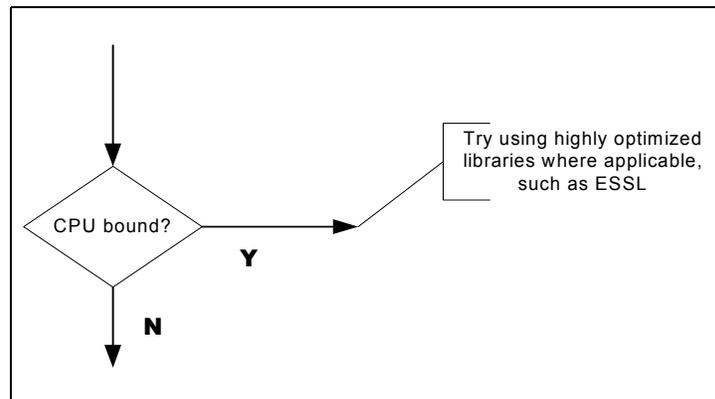


Figure 9-11 CPU-bound code might benefit from optimized libraries

This is a state-of-the-art collection of mathematical routines. The ESSL family of subroutines for AIX 5L and Linux contains:

- ▶ Basic Linear Algebra Subprograms (BLAS)
- ▶ Linear Algebraic Equations
- ▶ Eigensystem Analysis
- ▶ Fourier Transforms

To illustrate that ESSL provides the best performance we used ESSL Version 4.2, which is available for AIX 5L 5.3 for POWER5. This version requires XL Fortran Enterprise Edition Version 9.1 for AIX 5L and the XL Fortran Enterprise Edition Run-Time Environment Version 9.1 for AIX 5L. For this test we use DGEMM, which has been extensively optimized for L1 and L2 caches. The results from ESSL are compared against the hand-tuned Fortran version of DGEMM presented in previous sections. Table 9-15 summarizes the results for DGEMM ESSL and DGEMM Fortran.

Table 9-15 DGEMM routine optimized in the ESSL library

Dimension	Fortran	ESSL
100	4902	4403
200	5486	5494
500	5691	5893
1000	5944	6126
2000	5837	6140
5000	6028	6124
10000	5900	6105

These results are also summarized in Figure 9-12. All benchmarks were carried out on an IBM *e*server p5 system with a clock speed of 1.65 GHz. The only case in which the Fortran version has a slight advantage over ESSL is for small-square matrixes, in this case 100x100. In all other cases, the version of DGEMM in ESSL outperforms the Fortran version by as much as 5%.

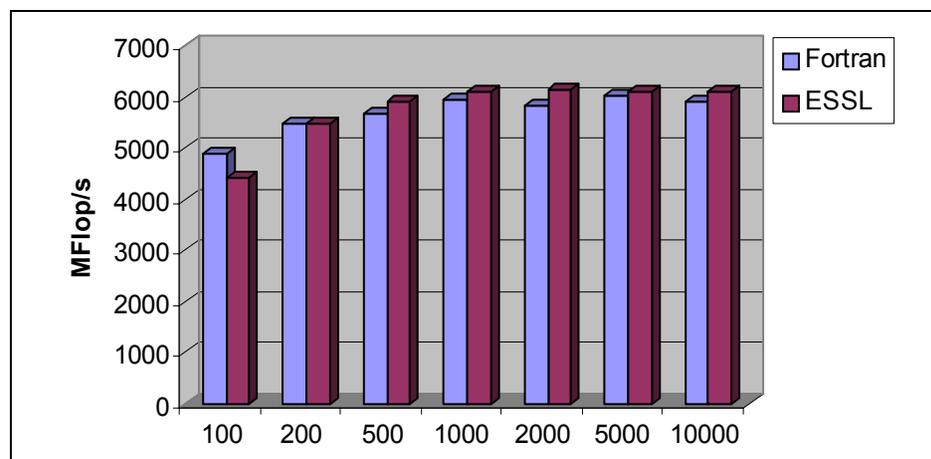


Figure 9-12 DGEMM optimized routine in ESSL versus Fortran version

The ESSL DGEMM routine is consistently higher than 90% of peak performance for matrixes with rank higher than 500. Peak performance for this particular POWER5 is 6.6 Gigaflops per second.

9.7 Parallel programming general concepts

Parallel computing involves dividing a task into smaller and more manageable blocks and distributing these blocks, in our case, among processors (physical or logical processors). In parallel computing, scientific and engineering applications are very important. They can take full advantage of a system with multiple processors, such as the IBM POWER5 server. In order to take full advantage of all of the power of a system with many or few processors, it is necessary to consider the following issues:

- ▶ Parallel algorithms

To be able to make use of all of the processors that are available on POWER5, we need algorithms that can be efficiently parallelized.

- ▶ Parallel languages

To implement a parallel algorithm, a parallel language is required. AIX 5L and Linux on POWER5 support the most common parallel paradigms.

- ▶ Parallel programming tools

This may involve evaluating performance of a particular application. As in the previous sections, this can answer questions such as how efficiently applications are taking advantage of the POWER5 architecture. In addition, parallel programming tools may involve interfaces that assist programmers debugging and shielding them from any low-level machine characterization.

- ▶ Parallel compiler programming

As compilers become more sophisticated and more information regarding the behavior of applications is put into the compiler, chances to provide programmers with automatic parallelization become greater.

Metrics

Generally for parallel applications, performance is a function of more parameters than for serial applications. Performance depends on the characteristics of the input, and basic considerations such as number of processors, memory, and

processor communication are very important. The optimal combination of all of these variables defines good scalability. In this book we define scalability as:

Scalability A measurement of how close an application performs proportional to the number of processors. It is expressed as parallel speedup.

To measure parallel speedup, we rely on elapsed time. (See 9.1, “Performance bottlenecks identification” on page 312.) The parallel time is not just the cumulative time for the parallel regions; it is the elapsed time from beginning to end of the simulation. In effect, this definition includes sequential and parallel regions. In scientific and engineering applications this is the most practical way to define it. The parallel speedup is defined as follows:

Parallel speedup A measurement that reflects scalability or the ability of an application to reduce the time to solution proportional to the number of processors:

$$S = \frac{T_{\text{sequential}}}{T_{\text{parallel}}}$$

Efficiency is simply $e = \frac{S}{p}$ where p is the number of processors.



Partition Load Manager

A general presentation of Partition Load Manager is provided in chapters 3 and 6 of *Advanced POWER Virtualization on IBM eServer p5 Servers Introduction and Basic Configuration*, SG24-7940. Chapter 3 contains a description of the general behavior of Partition Load Manager, and Chapter 6 has a detailed explanation of Partition Load Manager installation and configuration.

10.1 When and how should I use Partition Load Manager?

Partition Load Manager for AIX 5L is a load manager that provides automated processor and memory resource management across dynamic LPAR-capable logical partitions running AIX 5L V5.2 or AIX 5L V5.3. Partition Load Manager allocates resources to partitions as requested, within the constraints of a user-defined policy. It assigns resources from partitions with low usage to partitions with a higher demand, improving the overall resource utilization of the system. Partition Load Manager works with both dedicated partitions and micro-partition environments.

Partition Load Manager is an optional feature of most IBM @server p5 servers. (It is standard in p-590 and p-595 models.) In many cases, you can run your server satisfactorily without Partition Load Manager. First, this section shows how Partition Load Manager relates to other workload management tools available. This section then describes the cases where you would want to take advantage of Partition Load Manager. Finally, it provides best practices for deploying Partition Load Manager.

Note: Partition Load Manager can be used to manage AIX 5L V5.2 and V5.3 and Virtual I/O Server partitions. Partition Load Manager does not currently manage i5/OS or Linux partitions.

Although it is possible to manage the resources of a Virtual I/O Server with Partition Load Manager, this requires manual setup of Partition Load Manager so that the restricted shell (**rsh**) can be used on this partition. We therefore do not recommend that you perform this kind of resource management.

10.1.1 Partition Load Manager and other load-balancing tools

The POWER Hypervisor provides some built-in features to automatically allocate physical CPU resources. AIX 5L also provides load-balancing possibilities using Workload Manager (WLM). In this section, we compare these tools with Partition Load Manager.

Partition Load Manager versus POWER Hypervisor resource allocation

For dedicated partitions and capped micro-partitions, the POWER Hypervisor deals with resource allocation only at boot time and during dynamic LPAR operations generated from the HMC. At boot time, the POWER Hypervisor

checks for available resources and presents the booting partition with either its desired amount of resources, if available, or a smaller amount if not. After the partition is booted, the POWER Hypervisor will not modify the resources that are allocated to the partition unless a system administrator issues commands from the (Hardware Management Console) HMC.

Furthermore, if dedicated partitions (that are defined on the servers) are not booted, their unused processors will be moved by the POWER Hypervisor to the free processor pool for use by micro-partitions, and given back to the dedicated partitions when they are booted.

For uncapped micro-partitions, AIX 5L V5.3 provides an additional resource load balancing: As shown in Chapter 5, “Micro-Partitioning” on page 93, a partition’s unused processor time is given to other partitions when the kernel calls the H-cede or H-confer POWER Hypervisor calls. This freed processor time is then shared between the uncapped micro-partitions. This resource management is limited to processors and does not balance memory.

After partitions are booted, the POWER Hypervisor makes sure that each partition gets its share of resources as defined on the HMC:

- ▶ Memory and number of physical processors for dedicated partitions
- ▶ Memory, processing units, and number of virtual processors for micro-partitions
- ▶ Possibly additional processing power for uncapped micro-partitions

But the POWER Hypervisor does not dynamically modify these resource shares by itself.

Partition Load Manager adds a level of automation in the allocation of resources. It dynamically changes the amount of resources given to each partition.

Partition Load Manager versus Workload Manager

Partition Load Manager and WLM can be used concurrently. There is no overlap in their scope:

- ▶ The scope of Partition Load Manager is the set of hardware resources (processors and memory) that are configured in a physical server. Partition Load Manager dynamically allocates these physical resources to partitions.
- ▶ The scope of WLM is the set of resources (processor, memory, and disk I/O) within one partition. WLM dynamically allocates these resources among the processes running within one partition.

Because WLM allocates resources according to configuration files that contain relative values (percentage or shares) and not absolute values, WLM can work

independently of the allocation of resources by Partition Load Manager to a partition.

Consider this (very simplified) example. A partition contains two processes, P1 and P2, that are managed by WLM, so that P1 gets three shares of CPU and P2 gets two shares. Initially, the partition is allocated two physical processors, so P1 is using 60% of two processors (1.2 processor) and P2 is using 40% of two processors (0.8 processor). Later on, if processors become unused in the server and the partition is given three extra processors by Partition Load Manager, P1 will get 60% of five processors (three processors), and P2 will get two processors. The WLM priorities defined by shares are kept unchanged, and each process gets more physical resources thanks to the Partition Load Manager action, without any need to modify the WLM settings. In other words, WLM manages percentages of available resources given by Partition Load Manager.

10.1.2 When to use Partition Load Manager

Partition Load Manager provides an automated way to move processing power and memory between dynamic LPAR-capable logical partitions. The @server p5 servers support two types of partitions (dedicated partitions and micro-partitions), and Partition Load Manager provides slightly different benefits in each case. The following sections presents situations in which Partition Load Manager can be used, first for managing dedicated partitions, then to manage micro-partitions.

Partition Load Manager for dedicated partitions

In dedicated partitions, Partition Load Manager is a replacement for the manual dynamic LPAR reallocation of resources that a system administrator performs using the HMC. Rather than having to monitor partitions for lack or excess of computing resources, the system administrator can define, in a configuration file, thresholds for the use of these resources by the partitions. Partition Load Manager monitors the actual resources utilization against these thresholds, and automatically moves some resources from the partition with a low demand for these resources to the partition with a high demand.

If your system contains partitions with a fairly constant workload and equal relative priority, Partition Load Manager may offer little benefit. However, if you manually reconfigure your partitions using the HMC DPLAR feature, or if your partitions have a changing resource demand over time, Partition Load Manager can help you. Here are a few examples.

Partitions with unpredictable workload peaks

You may decide to consolidate onto a large @server p5 server several applications that are running on independent servers. Each independent server

is sized with enough processors and memory to satisfy the application peak loads, but these resources are unused a significant part of the time. Because the peaks of all applications are statistically spread over time, you can install a new server with fewer overall computing resources. Each application now runs in its own partition, for which you define a minimum, a desired, and a maximum amount of processing power and memory. When all partitions are booted, they are each given their desired amount of resources (assuming the sum of desired values is less than the overall server capacity). The amount of resources that are allocated to each partition will not change when one partition reaches a peak of activity. By activating Partition Load Manager with one configuration file using the same minimum/desired/maximum thresholds as those defined in the HMC, the partition with a peak of activity automatically takes advantage of the unused resources.

Partitions with time-based priorities

Assume that a server has two partitions that must be given different priority over time:

- ▶ For example, one partition is providing interactive service to end users, and we want to give priority to this partition only when the end users are awake. The other partition processes batch-type jobs and we want to give it priority at night, even though some end users may access the system.
- ▶ Another example of such a scenario is with one application that processes weekly reports and will get most of the server resources on Saturdays and Sundays, and another application that produces the data during the week.

In these cases, we can define two Partition Load Manager configurations that will be activated at different times. Here is an example that uses an eight-way server with 32 GB of memory:

1. On the HMC, we defined partitions P1 and P2 with the same amount of resources, so they can each use up to seven processors and 28 GB of memory, as shown in Table 10-1.

Table 10-1 Partitions resources definition on the HMC

	Minimum	Desired	Maximum
Processors	1	2	7
Memory	4	8	28

2. We defined two Partition Load Manager configurations for each of the time periods (night/day and weekdays/weekends). Table 10-2 on page 378 shows the settings of the resources in Partition Load Manager for the time period when partition P2 will have priority over P1.

Table 10-2 Resource allocations for time period 1

		Minimum	Desired	Maximum
P1	Processors	1	2	3
	Memory	4	8	12
P2	Processors	5	6	7
	Memory	20	24	28

The table for the other time period gives the opposite values to each partition. With these values, we guarantee that the partition with the highest priority gets at least five processors (and as many as seven). The partition with the low priority could go to as little as one processor if the other partition has a very high resource demand. Furthermore, compared to not using Partition Load Manager, this configuration adds some flexibility to the configuration within each time period, because the memory and processing resource of each partition can still fluctuate. For example, if the high-priority partition does not use all of its processing resource, the low-priority partition will be able to use up to three processors automatically.

Partition Load Manager does not provide a time-based reconfiguration feature. The change of configuration at the boundary between the two time periods is implemented using (for example) the `crontab` command to load a new configuration file in the PLM manager.

When you plan to activate several Partition Load Manager policies over time using low-minimum and high-maximum values in the partition profiles on the HMC, allow for more flexibility with the Partition Load Manager policies because these (HMC profile) values bound the values that can be used in the Partition Load Manager policy.

Partitions subsets: multiple clients

In this scenario, a large server is used to host applications belonging to several clients. Each client has paid for a fixed amount of resources, and each client needs to run several partitions.

In this case, you can take advantage of the Partition Load Manager concept of *partition group*.

A group is allocated a number of processors and a chunk of memory. Partition Load Manager reallocates these resources among the partitions belonging to this group. At least one group must be defined, and the simplest way to use Partition Load Manager is to gather all managed partitions within the same group.

When managing multiple clients, a better way is to define for each client a group that is allocated exactly the resources that the clients pays for. For example, the server contains 64 processors, and one client pays to use 16 processors. If the client needs five partitions, the client can then define the Partition Load Manager configuration file with one group owning these 16 processors. All of these client partitions belong to this group. The client can then configure the policies according to his preferences for allocation of free resources between his own partitions. This would have no impact on the allocation of the remaining 48 processors belonging to other clients.

Partition Load Manager with micro-partitions

The scenarios described previously for dedicated partitions are also valid for micro-partitions. But there are other cases specific to micro-partitioning in which you may want to use Partition Load Manager.

Splitting the shared pool

All processors that will be used by micro-partitions belong to only one shared pool. By using the Partition Load Manager groups, you can divide the shared pool into several subsets, so that processor entitlement and number of virtual processors resource balancing by Partition Load Manager is performed only within each group. This is similar to “Partitions subsets: multiple clients” on page 378.

To be more accurate about the behavior of the system when multiple groups are created, each with a maximum entitled capacity: Partition Load Manager distributes that capacity among the partitions within that group, but if any of the running partitions in the system is uncapped, then unused CPU cycles from any Partition Load Manager group can be given to the uncapped partition, whether it is within or outside the Partition Load Manager group. If a client creates a Partition Load Manager group and defines a maximum of five for the entitlement, if any of those partitions is uncapped, that group can (technically) run with more than an entitlement of five, because the POWER Hypervisor can assign unused cycles from other partitions outside of our group.

Partition Load Manager does not prevent the POWER Hypervisor from trying to optimize overall system throughput.

Partitions with concurrent peak loads

Several services may have their workload peaks at the same time, with a workload profile similar to those presented in Figure 5-17 on page 130 and Figure 5-18 on page 131. If these services are running in partitions of the same @server p5 server, they will compete for resources. The business case for sizing the server may have used the planned over-commit strategy described in 5.3.6, “Micro-Partitioning planning guidelines” on page 133. This is because some of

these services are not business-critical, and there is no justification for affording a system that can handle all applications workload peaks during small periods but be underutilized for long periods.

Partition Load Manager can help to allocate the systems resources to the partitions that have the highest business priority while only providing the other partitions with leftover resources. There are two parameters that the system administrator can use for this purpose: *cpu_guaranteed* and *cpu_shares* (These parameters are described in Section 10.2.3, “Configuration file and tunables” on page 386).

- ▶ The *cpu_guaranteed* parameter represents an absolute value of processing power that a partition is guaranteed to be allocated only if it needs it. If the partition runs below this guaranteed amount, the remaining capacity is available for the other partitions. By setting these parameters to a high value for all partitions that are considered critical, you ensure them that this processing power is available, whatever the workload of the other partitions.
- ▶ The *cpu-shares* parameter represents a relative value of processing power. It is used only to allocate the CPU resources in excess of the sum of the guaranteed CPU resources, between the partitions that need extra power. Using a higher value of *cpu_share* for high-priority partition than for lower-priority partitions enables prioritizing the distribution of the extra processing power that cannot be allocated otherwise.

Let us take an example of how to use a combination of these parameters. We assume that a system runs 10 micro-partitions with 10 CPUs in the free pool (total entitlement equal to 1000). Five partitions have high priority and are allocated a guaranteed CPU power of 180. The five partitions with low priority are assigned only 20 guaranteed CPU power units. If all partitions experience a workload peak except one of the high-priority partitions, which is idle, each partition is given its guaranteed power, and there are 180 units left to distribute among them. It is then the value of each partition's *cpu_share* that defines how to distribute this remaining power among the nine competing partitions.

Managing the number of virtual processors

The processing power of a micro-partition is defined by two parameters:

- ▶ Its entitlement (or percentage of one physical processor processing capacity)
- ▶ Its number of virtual processors

For an uncapped partition, the maximum processing power is reached when the POWER Hypervisor has allocated to this partition an entitlement equivalent to its allocated number of virtual processors running at 100%. If there are still some idle processors in the server, the micro-partition cannot use them and this unused processing power is wasted because the POWER Hypervisor will not

automatically add virtual processors to the partition. To take advantage of the available processing power, you must:

- ▶ Define the uncapped partition with a large maximum number of virtual processors.
- ▶ Define Partition Load Manager policies that add virtual processors to the partition when the entitlement exceeds a threshold (default is 80% of potential entitlement).

Assuming that the uncapped partition has been booted with its desired number of processors, when it reaches a peak of activity and if there is free processing capacity in the free processor pool, the POWER Hypervisor gives the partition extra CPU cycles to the limit of its current number of virtual processors. In addition to the POWER Hypervisor action, Partition Load Manager monitors several thresholds, and if one is crossed, Partition Load Manager receives a message (through RMC), and starts increasing the partition entitled capacity and number of virtual processors, up to the maximum number defined in the HMC for the partition. AIX 5L automatically takes into account the extra processors.

In a similar way, you can define the low-utilization threshold that lowers the number of virtual processors used by a partition when it has a low processing activity.

No rule is valid for all application profiles, but in general, for the same overall CPU entitlement, the performance of a micro-partition is better with a small number of virtual processors running with a high CPU utilization, than with a large number of virtual processors running at a low CPU utilization.

Many factors can influence this behavior. For example, an application that is not programmed to use parallelism or AIX 5L V5.3 software, simultaneous multithreading does not benefit from the availability of multiple processors. If this application were run on a physical SMP system, it would not take advantage of the physical processors. In the same way, when running on a micro-partition, it would not take advantage of the multiple virtual processors. On a micro-partition, you can improve the throughput by reducing the number of virtual processors: the POWER Hypervisor will spend less time dispatching these virtual processors and AIX 5L will spend less time trying to allocate processes to the processors.

If you create uncapped partitions on your system and your workload profile is such that processing resources could be moved from partition to partition, we recommend that you start deploying Partition Load Manager with the default values of the processor/entitlement ratios. Then, you can start changing the ratios to find the best fit for your applications.

Managing the memory

All of the scenarios that we just described were related to reallocation of processing power. Partition Load Manager can also reallocate physical memory between partitions, and the scenarios we presented can also apply to memory resource balancing.

10.1.3 How to deploy Partition Load Manager

When you have chosen to use Partition Load Manager, decide where to install the PLM manager. Here are a few considerations to take into account:

- ▶ Currently, the PLM server can run only on AIX 5L, so *you cannot use the HMC as the Partition Load Manager*. You need to find an AIX 5L server, and it can be either a dedicated server or a partition on a POWER4 processor-based or POWER5 processor-based system.
- ▶ Partition Load Manager does not require a dedicated AIX 5L instance. It can run on a system that is also running other applications.
- ▶ One Partition Load Manager instance manages only partitions within one physical server (one central processor complex, or CPC). However, you can run multiple Partition Load Manager instances on the same AIX 5L system. If you plan to manage partitions in many physical servers, you may want to centralize all Partition Load Manager management functions within the same AIX 5L instance to provide a single point of control for all Partition Load Manager operations in your computer environment.
- ▶ Partition Load Manager uses very few processing resources. It uses Resource Management and Control (RMC) to communicate between the Partition Load Manager and the managed partitions. When Partition Load Manager is activated, it sets up monitoring of threshold values on each managed partition. When a threshold is reached, the managed partition sends an event through RMC to the management server, which takes the appropriate action. The Partition Load Manager does not poll the managed partitions. You can find an example of resource requirements for a PLM server in 10.4.1, “Partition Load Manager resource requirements” on page 396.
- ▶ The Partition Load Manager can run in one of the partitions that it manages.

With these considerations, we propose a few recommended configurations:

- ▶ For a server farm or a large computing center, it is likely that some existing AIX 5L servers are dedicated to infrastructure support. These could be the control workstation of a PSSP¹ cluster, the Management Station of a CSM² cluster, a software repository such as a NIM³ server, or a monitoring server

¹ PSSP: Parallel System Support Program

² CSM: Cluster Systems Management

such as Tivoli® TEC or TMR. The Partition Load Manager management function requires very few processor cycles, memory, and disk space, so it can be implemented on one of these infrastructure servers as long as it has IP connectivity to all managed partitions. This solution has the advantage of providing a single point of control for all operations related to Partition Load Manager.

- ▶ For managing partitions in a single physical server, an inexpensive PLM server can be instantiated by dedicating a small micro-partition. The necessary disk space can be provided by a virtual disk exported from a Virtual I/O Server (logical volume). The IP connection with the managed partitions can be implemented through virtual Ethernet in memory VLAN, so that no hardware Ethernet adapters is required. One Physical Ethernet adapter would be needed for communication with the HMC, unless a put Shared Ethernet Adapter can provide this connectivity.
- ▶ An even less expensive configuration is to install the Partition Load Manager management function on one of the partitions running applications.

The choice then depends on the operations guidelines of each site.

10.2 More about Partition Load Manager installation and setup

Chapter 6 of *Advanced POWER Virtualization on IBM @server p5 Servers Introduction and Basic Configuration*, SG24-7940, includes a detailed explanation of basic Partition Load Manager installation and configuration.

In this redbook, we present more advanced installation and configuration options. First, we briefly describe how Partition Load Manager works before we investigate configuration details.

10.2.1 Overview of Partition Load Manager behavior

Partition Load Manager involves three (types of) entities:

- ▶ The PLM server that executes the Partition Load Manager code and decides resource reallocation actions.
- ▶ The managed partitions, which can request more or fewer resources.
- ▶ The HMC that drives the physical server that hosts the managed partitions. The HMC actually performs the resource reallocation actions decided by the PLM manager.

³ NIM: The Network Installation Manager feature of AIX

The Partition Load Manager resource manager is the server part of this client-server model and it runs on AIX 5L V5.2 and AIX 5L V5.3. When it starts, it uses RMC services to register several events on every client partition that will be managed by Partition Load Manager. For Partition Load Manager to obtain system information and dynamically reconfigure resources, it requires an SSH network connection from the PLM manager to the HMC, as well as IP connectivity between the PLM manager, the HMC, and the managed partitions. The RMC services are responsible for gathering all of the status information. The RMC daemon exports system status attributes and processes reconfiguration requests from the HMC. With this data and in conjunction with the user-defined resource management policy, Partition Load Manager decides what to do. Every time a partition exceeds a threshold, Partition Load Manager receives an RMC event. When a node requests additional resources, Partition Load Manager determines whether the node can accept additional resources. If so, Partition Load Manager conducts a search for available resources. It then checks the policy file in order to see if a partition is more or less deserving of the resources. Only then, Partition Load Manager allocates the requested resources.

Partition Load Manager uses a Micro-Partitioning entitlement model with a guaranteed or desired amount of resources, amount of shares, and (optional) minimum and maximum amounts. (The guaranteed amount of resources is the amount guaranteed to a partition when demanded.) It can get the resources from the free pool if they are available and the amount does not exceed its maximum, take underutilized resources from other partitions, or take utilized resources from partitions that are over their guaranteed resource. The allocated resource will vary between minimum and maximum values defined in the Partition Load Manager configuration file. For a partition to be allocated resources above the guaranteed amount, Partition Load Manager must know its share amount (relative priority versus other partitions priority). This amount is a factor between 1 and 255. The formula to calculate the ratio of resources allocated to each partition is $(\text{shares of the partition}) / (\text{sum of shares from competing partitions})$.

Partition Load Manager manages partitions within groups. Each partition must be a member of a group, and at least one group must be defined in the Partition Load Manager policy. One PLM server can manage independent groups of partitions but it cannot share resources across groups. It cannot take unused resources in one group in order to satisfy a demand for resources by another group. The partitions belonging to a group must be of the same type: either micro-partitions or dedicated partitions. One group may contain both capped and uncapped partitions. Partition Load Manager manages the entitled processor capacity, memory, and number of virtual processors for both capped and uncapped partitions.

System administrators must set up Partition Load Manager partition definitions in a way that is compatible with the HMC policy definition. The Partition Load

Manager cannot decrease a partition's minimum below the HMC's minimum, nor can it increase a partition's maximum over the HMC's maximum. Partition Load Manager will use the HMC partition definition minimum, desired, and maximum partition resource values as Partition Load Manager minimum, guaranteed, and maximum values if not specified in the Partition Load Manager policy. If the Partition Load Manager minimum and maximum values are not within the range defined on the HMC, Partition Load Manager will use an effective range defined by the intersection of the ranges defined on the HMC and in the Partition Load Manager configuration file.

10.2.2 Management versus monitoring modes

Partition Load Manager can execute in two modes: management and monitoring modes, which are analogous to the WLM active and passive modes.

- ▶ In monitoring mode, Partition Load Manager receives through RMC a request from partitions for resource reallocation when thresholds are reached. Partition Load Manager appends an entry in its log for each received RMC message but does not take action.
- ▶ In management mode, Partition Load Manager takes action for each RMC message, according to the policies defined in the configuration file.

If you do not have a test environment on which to define the best Partition Load Manager settings for your environment, if your production environment is critical, or if you do not have a thorough understanding of the partitions workload profile, we recommend that you start using Partition Load Manager in monitoring mode only, with the default configuration values. You can then run it for a significant time duration (one day, for example), and then analyze the log, looking for the frequency at which Partition Load Manager would take actions.

Note: Because Partition Load Manager does not take action when in monitoring mode, a request for additional resources will not be satisfied, and the requesting partition will repeat the request until it no longer needs extra resources. When analyzing the log, you should only take into account the resource need changes.

You can also use the `xplstat` command to recognize workload patterns. When you understand the workload profile of the managed partitions, you can decide which values to use in the Partition Load Manager configuration files.

Partition reconfiguration is not an instantaneous action, especially for dedicated processor or memory migration between partitions. You may not want to generate such an action to respond to a very short activity peak, when you know that the resources would no longer be needed in the following seconds.

10.2.3 Configuration file and tunables

The system administrator who decides about deploying Partition Load Manager has only two ways of defining Partition Load Manager behavior:

1. The values that are set in the Partition Load Manager configuration file.
2. The arguments that are given to the `xlplm` command (used to start Partition Load Manager).

The `xlplm` command is addressed in 10.3, “Managing and monitoring with Partition Load Manager” on page 390.

The Partition Load Manager configuration file is also called policy file in the Partition Load Manager documentation and on the Web System Management panels. You must create at least one policy file for each CPC on which you want to manage partitions. Policy files are ASCII files with a formal syntax. The policy file can be created (and modified) either through Web System Management panels or by using a text editor (`vi`, `emacs`).

Note: When you edit the file manually, take care to respect the syntax.

If you start Partition Load Manager using a policy file with an incorrect syntax, Partition Load Manager startup will fail. If Partition Load Manager is already running and you try to load a new policy with an incorrect syntax, Partition Load Manager will continue executing with the previously loaded policy.

The policy file contains different variables, called attributes, which are grouped in several sections, each with a different scope: global to one CPC, global for a group of partitions, or specific to one partition. *Tunables*, a subset of the attributes, define Partition Load Manager behavior. Tunables can be set at a system-wide level, and optionally overridden for some groups or partitions. Example 10-1 shows a policy for a physical server on which two partitions are managed by Partition Load Manager.

Example 10-1 Partition Load Manager policy file

```
#Example PLM policy file.
```

```
globals:
```

```
    hmc_host_name = p5hmc1
    hmc_user_name = hscroot
    hmc_cec_name = p5Server1
```

```
example:
```

```
    type = group
    cpu_type = shared
```

```
cpu_maximum = 2
mem_maximum = 0

p5_1test1:
  type = partition
  group = example
  cpu_guaranteed = 0.3
  cpu_maximum = 0.6
  cpu_minimum = 0.1
  cpu_shares = 2
  cpu_load_high = 0.3
  cpu_load_low = 0.2
  cpu_free_unused = yes

p5_1test3:
  type = partition
  group = example
  cpu_guaranteed = 0.3
  cpu_maximum = 0.5
  cpu_minimum = 0.1
  cpu_shares = 2
```

Partition Load Manager can read the definition of partition on the HMC using SSH. When starting managed partitions, Partition Load Manager reads the definition of all partitions to extract default values for these attributes. Then Partition Load Manager reads the Policy file to override the HMC-defined values.

Note: Partition Load Manager never overwrites a partition profile in the HMC.

We now discuss some of the attributes.

cpu_minimum, cpu_guaranteed, cpu_maximum, memory_minimum, memory_guaranteed, memory_maximum

The values are optional. If not present, Partition Load Manager accesses the definition of the partition in the HMC and extracts the values of minimum, desired, and maximum values for the CPU or memory to set these values.

You can set these attributes to values different from the HMC values.

For example, the HMC minimum value is defined as the minimum amount of resources needed to start a partition. However, you may know that if the partition is only given that amount of resources, its performance is degraded. In this case, you may want to define a Partition Load Manager `cpu_minimum` with a higher value than the HMC minimum, so that Partition Load Manager will never make

the partition work with bad performance. In other words, the HMC value is the bare minimum of resources needed to run the partition, while the Partition Load Manager minimum is the lowest reasonable value for acceptable performance.

Let us take another example. The HMC desired memory is the size of memory a partition will be allocated at boot time, when the system has enough memory resources for all partitions to be started. When running, the partition may run perfectly well with less memory. For example, an FTP server needs memory to load the files requested by FTP clients, but no longer needs this memory when the files are sent. Because there is no memory clean-up in AIX 5L, the memory used for these files will remain occupied. By setting the `memory_guaranteed` value to the same value as `memory_minimum`, you enable Partition Load Manager to request the partition to release the memory it no longer needs, to give it to other partitions with real needs for memory.

group

This tunable must appear in each partition stanza. One partition can belong to only one group.

Tip: There is a concept of group in the HMC that can be used when defining the partition. If you plan to use Partition Load Manager groups, we recommend that you do not use the HMC-defined groups.

cpu_shares

When Partition Load Manager is not managing an uncapped partition, the POWER Hypervisor allocates unused processor time to the uncapped partition according to the current active weight of the partition, defined by the value of the partition weight as defined on the HMC.

When the uncapped partition is managed by Partition Load Manager, its current active weight is overridden by the `cpu_shares` value defined on the Partition Load Manager policy file.

`cpu_shares` defines the relative priority of the partitions. Unused resources are allocated to partitions that have their guaranteed (desired) amount or more, in the ratio of their share value to the number of active shares.

The default value of the HMC-defined weight is 128, and the default of the `cpu_shares` value is 1, so it is important to make sure that all partitions within a Partition Load Manager group use a current active weight set from the same source: the HMC definition of Partition Load Manager policy file.

hmc_command_wait

As mentioned before, the reallocation of resources is not immediate. It takes some time for the HMC to ask a partition to release a CPU, move it from one partition to another, and then tell the target partition to activate the CPU. The `hmc_command_wait` attribute is the delay Partition Load Manager waits when asking the HMC to perform an operation, before determining that the HMC failed to process the request. If you have already used the dynamic LPAR feature, you can set up this attribute to the value of the dynamic reconfiguration timeout used for dynamic LPAR operations on your system.

cpu_load_low, cpu_load_high

These tunables are the threshold values beyond which Partition Load Manager decides that a partition has unneeded CPU or not enough CPU. The difference between these two values must be greater than 0.1 (entitlement measured as a fraction of one processor capacity). When reaching such a threshold, dedicated partitions give or receive one dedicated processor, while a micro-partition gives or receives an amount of entitled capacity defined by the `ec_delta` tunable. When adding capacity, `ec_delta` is a percentage of current capacity, while for removing capacity, `ec_delta` is a percentage of the resulting capacity.

The metric for these two tunables is the average number of runnable threads per processor. This is the value you would obtain when dividing the `r` column of the `vmstat` command (or the `runq_sz` column of the `sar -q` command, or the `load average` field of the `uptime` command) by the number of logical processors of the partitions. You can also find this value in the `rq` field of the new AIX 5L V5.3 `mpstat -d` command.

cpu_free_unused, mem_free_unused

These tunables define whether unneeded resources are taken from a partition when they are detected as unneeded (when set to `yes`), or only when another partition needs them (when set to `no`).

Immediately returning unused resources to the free pool improves the time to complete resource allocation to another partition. However, it decreases the performance of the partition that returns the resources if it needs extra resources later on.

Therefore, these values should be set to `yes` for partitions that seldom use extra resources and to `no` for partitions that have frequent peaks of resource utilization.

ec_per_vp_min, ec_per_vp_max

These are the two tunables to use to have Partition Load Manager automatically change the number of virtual processors of a micro-partition. Partition Load Manager does not directly handle this number because virtual processor addition

and removal is triggered by capacity changes. When entitlement must be added or removed, if the reallocation of entitled capacity results in crossing the `ec_per_vp_min` or `ec_per_vp_max` threshold, then virtual processors are also added or removed from the partition.

mem_util_low, mem_util_high, mem_pgstl_high

These tunables are the threshold values beyond which Partition Load Manager decides a partition has unneeded or insufficient memory.

To be considered as a memory requester, the partition must reach *both* `mem_util_high` and `mem_pgstl_high`. There is no lower limit for the page steal rate.

The unit used to measure `mem_util_low` and `mem_util_high` are is a percentage of memory. The memory utilization of the partition is defined as:

$$\text{pct} = ((\text{memory pages} - \text{free pages}) / \text{memory pages}) * 100$$

`memory pages` and `free pages` are the values returned by the `vmstat -v` command.

The unit that is used to measure `mem_pgstl_high` is a number of page steals per second. The value compared to the `mem_pgstl_high` threshold is the value returned in the `fr` field of the `vmstat` command.

10.3 Managing and monitoring with Partition Load Manager

There is a single point of control for configuring, managing, and operating Partition Load Manager: the PLM manager. The Partition Load Manager policy and log files are stored on the PLM server. Partition Load Manager commands can only be used on the PLM server. There are no Partition Load Manager commands that can be used from the HMC or the Partition Load Manager managed partitions.

Partition Load Manager can be operated in two ways:

1. The UNIX way, using the AIX 5L command line interface to edit the policy files, browse the logs, start and stop Partition Load Manager, and so on.
2. The GUI way using Web System Management to access the PLM server.

Note: There is no SMIT panel to configure and operate Partition Load Manager.

Operating Partition Load Manager is very simple, as it has only two commands:

x1plm Used to start, stop, modify, or query Partition Load Manager.
x1pstat Used to display statistics about the managed partitions.

Web System Management provides the equivalent functions in a graphical environment.

10.3.1 Managing multiple partitions

We have mentioned that one Partition Load Manager policy file controls partitions that must all run on the same CPC, and that one PLM manager can manage several partitions executing onto different physical servers. Let us be more accurate about how this works.

We will call:

PLM server The AIX 5L instance in which the Partition Load Manager code has been installed.
PLM server instance The server part of the Partition Load Manager client/server relationship: The clients are the managed partitions.

Note: Each PLM server instance executes under AIX 5L as an **x1plmd** daemon.

On the PLM server, you can start several PLM server instances by using the **x1plm** command several times, with different policy files.

One PLM server instance executes only one policy file at a time, and therefore manages only partitions within the same CPC.

The way to start a PLM server instance is to use the **x1plm** command with the **-S** (Start) argument, and the name of a policy file:

```
x1plm -S -p policy_file
```

This command start a server instance which will be named default. If you want to start multiple PLM server instances from the same PLM server, you need to give a name to each PLM server instance:

```
x1plm -S -p policy_file server_instance_name
```

Each policy file starts with a global stanza containing the host name of the HMC that manages a CPC and the name of the CPC as seen from the HMC. For each set of managed partitions, you may want to use a different set of Partition Load

Manager policies (for example, to change the Partition Load Manager behavior with the time of day). A good practice is therefore to create policy file names and PLM server names according to mnemonic naming conventions.

A simple example is:

```
<Server_Instance_Name> =: <HMC_shortcode><CEC_name>  
<Policy_Name> =: <Server_Instance><Suffix>
```

If you type two commands in a row, you know that you have applied the third policy defined for the second CPC managed by your first HMC, and the fourth policy defined for the first CPC managed by the second HMC.

```
xlplm -S -p hmc1cec2policy3 hmc1cec2  
xlplm -S -p hmc2cec1policy4 hmc2cec1
```

In the same way, you should use a meaningful name for the log files. There is one log file per PLM server instance.

You can start two PLM server instances to manage partitions that are hosted on the same CPC. This can be useful for examples when two different system administration teams manage different sets of partitions on one system. Each team can have full control of its own Partition Load Manager policy file.

Using the `xlplm` command with `-Q` (Query) displays the policy used for each PLM server controlled from one PLM manager:

`xlplm -Q` Shows all active instance names.

`xlplm -Q server_instance_name` Shows all details for one instance.

10.3.2 Extra tips about the `xlplm` command

The `xlplm` command has multiple arguments and flags. Refer to the official documentation for an extensive explanation of this command.

Here are a few:

-C The `-C` (Check) flag is used to verify the syntax of a policy file. It is useful when the file has been edited manually instead of through Web System Management. The `-C` flag does not check the policy file tunables consistency versus the values defined in the HMC.

-M The `-M` (Modify) flag enables the system administrator to dynamically load a new policy, to ask for writing the log in a different log file, or to switch from monitoring to managing mode. This flag can be used when the `xlplm`

command is called by a **crontab**, for example, when time-based policies must be used.

The Modify flag can be used to change resources groups. For example, if a system is intensively used during daytime and all partitions are competing for resources, it can be a good practice to define multiple groups and to gather partitions by workload type, so that each partition is sure to have a fair share of the resources. This improves the overall system throughput. However, if the system is underused at night, optimizing the throughput is no longer a critical issue. It can then be better to gather all partitions in one large group, so that if one partition has a burst of activity, it can use all available resources. In this case, the policy helps improve the response time of one partition rather than the overall throughput.

-Q -f By default, the **-Q** (query) option displays the intersection of the min and max range as defined in the HMC and in the Partition Load Manager policy file. When used with the **-f** flag, the command returns the values defined in the Partition Load Manager policy file.

-Q -v The **-v** (verbose) option, when used with the **-Q** option, provides extra information. In particular, this is the option to use to see whether the policy is used in management or monitoring mode, and to see the thresholds set for each partition and group.

10.3.3 Examples of Partition Load Manager commands output

This section shows some examples of Partition Load Manager command output. Example 10-2 displays the output of the **xlplm** query command when used in command line mode.

Example 10-2 xlplm query

```
# xlplm -v -Q benchtest
PLM Instance: benchtest
```

CEC Name	Server-9117-570-SN105428C
Mode	manage
Policy	/etc/plm/policies/testBenchmark
Log	/var/test.out
HMC Host	isvlab064.austin.ibm.com
HMC User	hscroot

```
GROUP: Benchmark1
```

	CUR	MAX	AVAIL	RESVD	MNGD
CPU:	2.77	4.00	1.23	0.00	Yes
MEM:	14336	0	0	0	No

CPU TYPE: shared

basil.austin.ibm.com

RESOURCES:

	CUR	MIN	GUAR	MAX	SHR
CPU:	1.72	0.10	1.00	4.00	128
MEM:	7168	1024	7168	15360	1

TUNABLES:

	INTVL	FRUNSD	LOADLO	LOADHI	DELTA	PGSTL
CPU:	3	1	0.75	1.00	20%	-
MEM:	6	0	50%	90%	16	0

	MIN	MAX
PER VP CAP:	0.74	0.90

sage.austin.ibm.com

RESOURCES:

	CUR	MIN	GUAR	MAX	SHR
CPU:	1.04	0.10	1.00	4.00	128
MEM:	7168	1024	7168	15360	1

TUNABLES:

	INTVL	FRUNSD	LOADLO	LOADHI	DELTA	PGSTL
CPU:	3	1	0.75	1.00	20%	-
MEM:	6	0	50%	90%	16	0

	MIN	MAX
PER VP CAP:	0.74	0.90

#

You can also use Web System Management tool to manage Partition Load Manager. Figure 10-1 on page 395 shows where to find Partition Load Manager in the Web System Management navigation area.

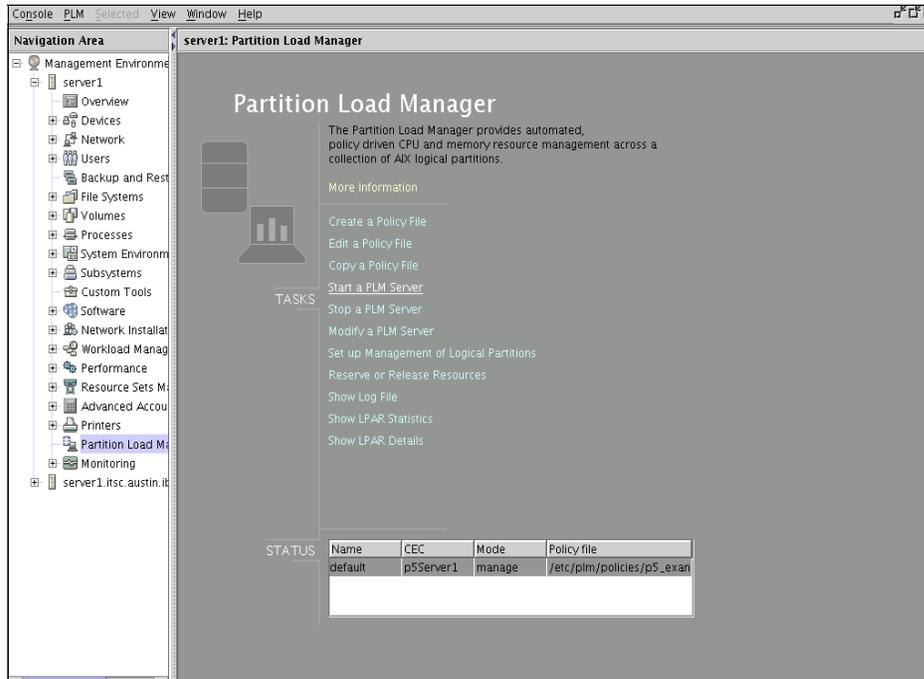


Figure 10-1 Partition Load Manager management using Web System Management

Figure 10-2 and Figure 10-3 on page 396 show the partitions processor statistics and memory statistics, respectively.

CPU Statistics		Memory Statistics						
Name	Minimum	Guaranteed	Maximum	Share	Current	Use %	Load average	
p5Server1			2.00					
--- example			2.00					
----- p5_1test1	0.00	0.00	0.00	2	0.00	0.00	0.00	
----- p5_1test3	0.10	0.30	0.50	2	0.50	60.80	0.09	

Buttons: Close, Refresh, Help

Figure 10-2 Partition Load Manager processor statistics using Web System Management

CPU Statistics		Memory Statistics					
Name	Minimum	Guaranteed	Maximum	Share	Current	Use %	Pagesteal
p5Server1			0				
--- example			0				
----- p5_1test1	0	0	0	1	0	0.00	0
----- p5_1test3	512	1024	2048	1	1024	32.46	0

Figure 10-3 Partition Load Manager memory statistics using Web System Management

10.4 Partition Load Manager performance impact

The performance aspect of Partition Load Manager is twofold: the resource requirement needed to run the PLM server, and the impact of Partition Load Manager actions on managed partitions performance.

10.4.1 Partition Load Manager resource requirements

The Partition Load Manager function is provided on the server by the `xlplmd` daemon. Here are some measurements of this daemon's resource requirements.

- ▶ We set up an environment in which Partition Load Manager manages two partitions, and we configure it so that Partition Load Manager receives three RMC events every minute. Example 10-3 shows how you can use the `tail` command to monitor the Partition Load Manager log and see these events.

Example 10-3 Checking the Partition Load Manager log

```
# tail -f test.out
<04/06/70 17:38:40> <PLM_TRC> Event notification of CPUZone low for Benchmark1 .
<04/06/70 17:39:05> <PLM_TRC> Event notification of CPUZone low for Benchmark2 .
<04/06/70 17:39:20> <PLM_TRC> Event notification of CPUZone low for Benchmark1 .
<04/06/70 17:39:34> <PLM_TRC> Event notification of CPUZone low for Benchmark2 .
<04/06/70 17:40:00> <PLM_TRC> Event notification of CPUZone low for Benchmark1 .
<04/06/70 17:40:14> <PLM_TRC> Event notification of CPUZone low for Benchmark2 .
<04/06/70 17:40:40> <PLM_TRC> Event notification of CPUZone low for Benchmark1 .
<04/06/70 17:40:54> <PLM_TRC> Event notification of CPUZone low for Benchmark2 .
```

In a stable environment, the frequency of these events would be much lower.

- ▶ Under these conditions, Partition Load Manager uses approximately 1.4 MB of memory, as shown in Example 10-4. Column RSS displays the size (in 1 KB units) of the real memory used by `x1plmd`.

Example 10-4 Memory utilization for the Partition Load Manager daemon

```
# ps xvg 344232
  PID   TTY STAT  TIME PGIN  SIZE  RSS  LIM  TSIZ  TRS %CPU %MEM COMMAND
344232  -  A    0:00  66  1288  1396  xx   78   120  0.0  0.0 x1plmd
b
#
```

- ▶ We also use the `ps` command to measure the amount of processing resources used by the Partition Load Manager daemon. Example 10-5 shows that after managing partitions for 1.5 hours, Partition Load Manager has only used 11 seconds of CPU time. It also shows that the percentage of memory and CPU used by the daemon is not measurable (percentage equal to 0).

Example 10-5 Processing resources needed by Partition Load Manager

```
# ps -o uid,pid,pmem,etime,time,pcpu,comm -p 405676
UID  PID  %MEM  ELAPSED      TIME  %CPU COMMAND
0 405676  0.0   01:29:36   00:00:11  0.0 x1plmd
#
```

The Partition Load Manager daemon does not generate any significant disk or network activity.

Conclusion

The resources that are required to run the Partition Load Manager daemon are not a criteria for deciding where to instantiate the PLM server. It can run on any existing AIX 5L system with no visible impact on this system performance.

10.4.2 Partition Load Manager impact on managed partitions

Besides the Partition Load Manager commands described in Section 10.3, “Managing and monitoring with Partition Load Manager” on page 390, one way of visualizing the effect of Partition Load Manager is to use the AIX 5L V5.3 Performance Toolbox (PTX).

Figure 10-4 on page 398 shows a PTX window while CPU-intensive programs are executing. A four-way p5-570 is split in two uncapped micro-partitions, called Basil and Sage. Sage is the partition where a benchmark program is run (the benchmark partition). This benchmark consists of many processes running in parallel, so that it can take advantage of as many logical processors as are given

to the partition. Basil is the monitoring partition. It runs the PLM server and the PTX management program (`xmperf`). It is also loaded with dummy programs to show the impact on the benchmark partition of varying workload in the other partitions.



Figure 10-4 Partition Load Manager impact on number of virtual processors

Note: To help readers whose copies of this book are printed in black and white, we have annotated the figure. The arrow with the circled A points to the blue line, and the arrow with the circled B points to the red line. The blue line presents many peaks, and the red line looks like stairs.

The screen has three parts (also called *instruments* in PTX terminology):

- ▶ The top part shows the workload of the monitoring partition. It has two lines:
 - The blue line displays the run queue.
 - The red line displays the number of virtual processors.
- ▶ The middle part shows the workload of the benchmark partition. It also displays the run queue size and the number of virtual processors using the same graphical conventions as the top part.
- ▶ The bottom part shows the CPU utilization of the benchmark partitions. It shows the percentage of CPU time spent in user, kernel, wait, or idle mode.

In each part, the X-axis represents time, with a scale using a 24h base time representation.

The figure demonstrates that Partition Load Manager will adjust the number of virtual processors allocated to a partition according to resources needed and the amount of available resources in the server. Partition Load Manager also adjusts the CPU entitlement, but for the sake of clarity of the diagram, we have chosen to plot only two values (run queue size and number of virtual processors):

- ▶ Initially, both partitions have only one processor, because this is the minimum as defined in the Partition Load Manager policy, and there is no activity.
- ▶ Shortly before 14h58, the benchmark is started on Sage, and Basil is nearly idle. Partition Load Manager notices the need for extra resources of Sage, and gradually increases its number of allocated virtual processors up to the maximum (4) to use all resources in the server.
- ▶ Around 15h00, the workload on Basil increases (due to starting some CPU-intensive programs).
- ▶ Around 15h01, Partition Load Manager notices that Basil's resource need is persistent, and it starts increasing its entitlement and number of virtual processors (set to 2).
- ▶ At the same time, because all resources in the system are used, Partition Load Manager decreases the entitlement of Sage. Because the ratio of entitlement per processor falls below the `ec_per_vp_min` threshold, Partition Load Manager removes one virtual processor from Sage.
- ▶ At 15h02, the system is stable. Both partitions are running CPU-intensive applications. Since they are defined with the same thresholds, tunables, and priority in the policy file, Partition Load Manager allocates two virtual processors to each of the partitions.

Related publications

The publications that are listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 403. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Advanced POWER Virtualization on IBM @server p5 Servers Introduction and Basic Configuration*, SG24-7940
- ▶ *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463
- ▶ *AIX 5L Performance Tools Handbook*, SG24-6039
- ▶ *AIX Logical Volume Manager, From A to Z: Introduction and Concepts*, SG24-5432
- ▶ *The Complete Partitioning Guide for IBM @server pSeries Servers*, SG24-7039
- ▶ *IBM @server pSeries Facts and Features*, G320-9878
- ▶ *IBM @server pSeries Sizing and Capacity Planning*, SG24-7071
- ▶ *Logical Partitions on IBM PowerPC: A Guide to Working with LPAR on POWER5*, SG24-8000
- ▶ *Managing AIX Server Farms*, SG24-6606
- ▶ *The POWER4 Processor Introduction and Tuning Guide*, SG24-7041
- ▶ *A Practical Guide for Resource Monitoring and Control (RMC)*, SG24-6606
- ▶ *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, SG24-5155
- ▶ *Understanding IBM @server pSeries Performance and Sizing*, SG24-4810

Other publications

These publications are also relevant as further information sources:

- ▶ “IBM POWER5 Chip: A Dual-Core Multithreaded Processor,” Ron Kalla, Balaram Sinharoy, Joel M. Tendler, IBM, *IEEE micro*, March/April 2004 (Vol. 24, No. 2) pp. 40-47.
- ▶ “Performance Workloads in a Hardware Multi-threaded Environment,” by Bret Olszewski and Octavian F. Herescu
<http://www.hpcaconf.org/hpca8/sites/caecw-02/s3p2.pdf>
- ▶ “IBM @server POWER4 System Microarchitecture,” J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, *IBM Journal of Research and Development*, Vol. 46, No. 1, 2002, pp 5-26.
- ▶ “Simultaneous Multi-threading: A Platform for Next-Generation Processors,” Eggers, Emer, Levy, Lo, Stamm, Tullsen, *IEEE micro*, September/October 1997 (Vol. 17, No. 5), pp. 12-19.
- ▶ *Electronic Service Agent for pSeries and RS/6000 User's Guide*
ftp://ftp.software.ibm.com/aix/service_agent_code/AIX/svcUG.pdf
- ▶ *Electronic Service Agent for pSeries Hardware Management Console User's Guide*
ftp://ftp.software.ibm.com/aix/service_agent_code/HMC/HMCSAUG.pdf
- ▶ “An Introduction to Virtualization,” Amit Singh
<http://www.kernelthread.com/publications/virtualization/index.html>
- ▶ “AIX 5L Support for Micro-Partitioning and SMT”, L. Browning
http://www.ibm.com/servers/aix/whitepapers/aix_support.pdf
- ▶ Sanger Institute Human Genome Server
http://www.ensembl.org/Homo_Sapiens/
- ▶ Fluent, Inc.
<http://www.fluent.com>
- ▶ The GNU Profiler by J. Fenlason and Richard. Stallman
http://www.gnu.org/software/binutils/manual/gprof-2.9.1/html_mono/gprof.html

Online resources

These Web sites are also relevant as further information sources:

- ▶ Simultaneous Multithreading Project

<http://www.cs.washington.edu/research/smt/index.html>

- ▶ IBM AIX 5L home page
<http://www.ibm.com/servers/aix/>
- ▶ AIX Toolbox for Linux applications
<http://www.ibm.com/servers/aix/products/aixos/linux/download.html>
- ▶ Software for AIX 5L
<http://www.ibm.com/servers/aix/products/>
- ▶ CUoD process, brief explanation
<http://www.ibm.com/support/docview.wss?uid=tss1fq102449>
- ▶ IBM Configurator for e-business (IBM internal site)
<http://w3.ibm.com/transform/crm/crmsite.nsf/public/config>
- ▶ IBM @server pSeries LPAR documentation and references Web site
<http://www.ibm.com/servers/eserver/pseries/lpar/resources.html>
- ▶ AIX and Linux training
<http://www.technonics.com>
- ▶ Linux on eServer p5 and pSeries information
<http://www.ibm.com/servers/eserver/pseries/linux/>
- ▶ Microcode Discovery Service information
<http://techsupport.services.ibm.com/server/aix.invsoutMDS>
- ▶ OpenSSH Web site
<http://www.openssh.com>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications, and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Numerics

3dmon command 245

A

Address Resolution Protocol (ARP) 188
address translation 30–31
 effective to real address translation (ERAT) 88
adjustable thread priorities 50
AIX 5L V5.2 234
AIX 5L V5.3 35, 96, 122, 127, 218, 384, 396
 3dmon command 245
 bindintcpu command 239
 chlvcopy command 253
 commands
 lparstat 83
 cpuinfo structure 237
 extendlv command 253
 extendvg command 253
 gprof command 244
 importvg command 253
 ioo command 254
 iostat command 237, 239–240, 270
 jtopas command 247
 Logical Volume Manager
 max_vg_pbuf_count 253
 pb_pbuf_count 253
 pervg_blocked_io_count 253
 total_vg_pbufs 253
 lparstat command 91, 248, 258
 lslv command 253
 lspv command 253
 lvmo command 253
 mklvcopy command 253
 mkvg command 253
 mpstat command 249, 264
 Object Data Manager (ODM) 174
 Partition Load Manager 255
 xlplm command 391
 xlpstat command 391
 perfwb command 249
 sar command 237, 239–240, 272
 sysconfig subroutine 272
 thread priorities 56

topas command 237, 240, 276
trace command 241
trcrpt command 241
varyonvg command 253
vmo command 251
vmstat command 237, 239–240, 268
xmperf command 245, 278

AIX commands

bosboot 49
cfgmgr 208
curt 124
lsdev 207
lsmmap 208
mkdev 207
mksysb 152
mpstat 108, 114
no 174
smtctl 48, 117
trace 112
vmstat 117, 125

AMBER 60

AMBER7 60

 benchmark tests 65

Application Binary Interface (ABI) 78

 application tuning 3, 6

B

benchmarks

 simultaneous multithreading 61–71

Berkeley sockets 175

bindprocessor command 238

BLAST 60

 benchmark tests 68

Branch History Tables (BHT) 16

C

cache affinity 108

cache hit 22–23

cache miss 22–23

caches

 L1 data cache 22

 L1 instruction cache 21

 replacement policy 22

Capacity on Demand 6
 definition 2
Capped mode *See* Micro-Partitioning
 capped mode
chlvcopy command 253
Command/Response Queue (CRQ) 149, 212
Condition Register (CR) 19
Count Register (CTR) 19
course grain threading 43
critical data forwarding (CDF) 23

D

dead gateway detection 159
decision support systems (DSS) 132
decrementer 85
dedicated processor partition 97
dedicated processor partitions 94
dingle-threaded execution mode
 null state 46
Direct Memory Access (DMA) 80
dispatch wheel 105
Dual Chip Modules (DCMs) 36
dynamic power management 33
dynamic resource balancing (DRB) 44, 49

E

effective to real address translation (ERAT) 88
entitled capacity 89
Error Correction Control (ECC) 25
EtherChannel 255
extendlv command 253
extendvg command 253

F

fine-grain threading 43
Firmware
 Open Firmware 75
firmware
 low-level 75
 system 75
Fixed-Point Exception Register (XER) 19
Floating-point Register (FPR) 19
Floating-point Status and Control Register (FPSCR)
19
FLUENT 60
 benchmark tests 70
FLUENT rating 70

four-way set associative 23

G

Gaussian03 59
 benchmark tests 61
General Purpose Register (GPR) 19
gprof command 244
gratuitous ARP 159
guaranteed capacity 134

H

Hardware Maintenance Console (HMC) 97
Hardware Management Console (HMC) 80
hardware multithreading 43
Hardware Page Table (HPT) 86
higher availability
 LVM mirroring 161
 multipath I/O 162
 multipath routing 159
high-performance computing 59
hosted partition 206
hosting partition 206
hyperthreading 237
Hypervisor calls
 H_CEDD 76
 H_CONFER 76
 H_PROD 76
Hypervisor decremter 76
Hypervisor mode 51, 76

I

i5/OS 75
IBM Subsystem Device Driver (SDD) 218
IEEE 754 19
IEEE 802.1Q VLAN 165
importvg command 253
initiator 206
instruction cache 45
instruction cracking in POWER5 17
instruction translation facility 45
Internet Protocol (IP) 188
 ipforwarding 186
involuntary context switches 124
ioo command 254
iostat command 239–240, 270
iSeries Partition Licensed Internal Code (PLIC) 75

J

jtopas command 247

K

kernel locks 122

L

L1 cache 21–23

cache line size 21

L1 data cache 22

L1 instruction cache 21

L2 cache 11, 25–26

cache size 25

line size 25

L3 cache 11, 27–28

cache line size 27

cache size 27

Least Recently Used (LRU) replacement policy 22

Level 2 (L2) cache

store gathering 25

Link Register (LR) 19

Linux 48, 58, 75

on pSeries 403

load miss queue (LMQ) 23

locking considerations 121

Logical I/O Bus Number (LIOBN) 80

Logical LAN Switch 171

logical processor 96

logical processors 107

Logical Remote Direct Memory Access (LRDMA)
150, 214

logical states, virtual processor 77

logical volume

limitations 155

low-level firmware 75

lparstat command 83, 91, 248, 258

LRDMA

See Logical Remote Direct Memory Access
(LRDMA)

lsiv command 253

lspv command 253

LTG 252

LVM 252

LVM mirroring 161

lvmo command 253

M

MAC address

See Media Access Control (MAC) address

Machine State Register (MSR) 88

Media Access Control (MAC) address 169

memory affinity considerations 126

Memory Mapped I/O (MMIO) 88

memory wall 41

Merged Logic DRAM (MLD) 21

Micro-Partitioning 93

application considerations 128

cache affinity 108

capped mode 101

capped partitions 89

dedicated processor partition 97

dedicated processor partitions 94

definition 1

dispatch wheel 105

entitled capacity 89

guaranteed capacity 134

idle partition considerations 127

implementation 96

logical processor 96

phantom interrupts 112

physical processor 97

server consolidation 95

server provisioning 96

shared processor partitions 95

slow clock ticks 128

uncapped mode 101

uncapped partition weighting 104

uncapped partitions 89

uncapped weight 101

variable capacity weighting 104

virtual processor 95, 97

maximum number 98

virtual server farms 96

mkivcopy command 253

mkvkg command 253

mpstat command 249, 264

Multi-Chip Modules (MCMs) 36

multipath I/O 162, 218

multipath routing 159

multiple operating system support 145

multithreading

course grain multithreading 43

multithreading technology 10

N

Neighbor Discovery Protocol (NDP) 188
netperf benchmark 174
 TCP_RR 175
 TCP_STREAM 175
NIM on Linux (NIMoL) 152
no-operation (nop) instruction 51
NVRAM 79

O

Object Data Manager (ODM) 174, 210
Open Firmware 75, 79, 147, 212
 device nodes 147
 device tree 104, 147
 properties of 148
 virtual host bridge 148
 virtual interrupt source controller 148
OSI Layer 2 network bridge 187
OSI-Layer 2 171

P

page frame table 79
Page Table Entry (PTE) 79
Partition Load Manager 7, 124, 255
 xlplm command 391
 xlpstat command 391
PCI adapters 75
performance bottleneck 3
performance considerations
 virtual Ethernet 174
 Virtual I/O Server 190
Performance Monitor Support 80
performance tools
 3dmon command 245
 gprof command 244
 ioo command 254
 iostat command 239–240, 270
 jtopas command 247
 lparstat command 248, 258
 lvmo command 253
 mpstat command 249, 264
 perfwb command 249
 sar command 239–240, 272
 topas command 240, 276
 trace command 241
 trcrpt command 241
 vmo command 251
 vmstat command 239–240, 268

 xmperf command 245, 278
performance tuning
 memory affinity considerations 126
 rPerf numbers 139
perfwb command 249
phantom interrupts 112
physical Ethernet
 sizing guidelines 202
physical mirroring 218
physical processor 97
POSIX 271
Power Architecture 17
POWER Hypervisor 44, 119, 212–213, 374
 debugger support 80
 decrementer 31
 definition 1
 device nodes 147
 dump support 80
 Hypervisor calls
 H_CEDE 76
 H_CONFER 76
 H_PROD 76
 Hypervisor decremter 76
 Hypervisor mode 76
 Machine Check Interrupt 77
 managed device classes 145
 memory considerations 85
 memory migration support 80
 page frame table 79
 partition managed class 146
 performance monitor support 80
 support 76
 System (Hypervisor) Call Interrupt 78
 System Reset Interrupt 77
 Translation Control Entry (TCE) 80
 virtual I/O support 85
 virtual terminal support 80
POWER4 10, 44
POWER4 architecture 73, 87, 89, 94
POWER5 9–40
 address translation 30–31
 architecture 10–31
 Branch History Tables (BHT) 16
 branch prediction 16
 cache comparison 30
 chip design 12
 CR 19
 CTR 19
 decrementer 85

- Dual Chip Modules (DCMs) 36
- dynamic power management 33
- exceptions 20
- FPRs 19
- FPSCR 19
- FPU 11
- FXU 11
- Global Completion Table (GCT) 18
- GPRs 19
- group dispatch 17
- IDU 11
- IFU 11
- instruction cracking 17
- instruction decoding and preprocessing 17
- instruction execution 19
- Instruction Fetch Address Register (IFAR) 15
- Instruction Fetch Buffers (IFBs) 17
- instruction fetching 15
- instruction group 17
- instruction pipelines 14
- instruction size 15
- issue queues 17, 20
- ISU 11
- L1 cache 21–23
 - cache line size 21
- L1 data cache 22
- L1 instruction cache 21
- L2 cache 11, 25–26
 - cache size 25
 - line size 25
- L3 cache 11, 27–28
 - cache line size 27
 - cache size 27
- link stack 17
- load reorder queue 17
- LR 19
- LSU 11
- Machine State Register (MSR) 88
- maximum outstanding branches 17
- MC 11
- Merged Logic DRAM (MLD) 21
- Multi-Chip Modules (MCMs) 36
- multithreading technology 10
- register renaming 18
- Rename Mapper 18
- store reorder queue 17
- superscalar 18
- Thread Status Register (TSR) 50
- Time Base register 85
- total execution units 20
- XER 19
- POWER5 architecture 111
 - decrementer interrupt 46
 - Hypervisor mode 51
 - instruction fetch address registers 45
 - no-operation (nop) instruction 51
 - Processor Utilization Resource Register (PURR) 34
 - rPerf numbers 139
 - snoozing 48
 - supervisor mode 51
 - user mode 51
- POWER5 description
 - address registers 45
 - dynamic feedback 44
 - dynamic thread switching capabilities 44
 - software controlled thread priority 44
 - thread switching capabilities 44
- PowerPC 88
- PowerPC Application Binary Interface (ABI) 78
- PowerPC Architecture 17
- PowerPC AS architecture (V2.02) 10
- processor affinity 108
- processor pools 96
- Processor Utilization Resource Register (PURR) 34

R

- RDMA
 - See Remote DMA (RDMA)
- Redbooks Web site 403
 - Contact us xv
- register renaming in POWER5 18
- Reliable Command/Response Transport 213
- Remote DMA (RDMA) 149
- Remote DMA TCE Table (RTCE) 213
- Resource Management and Control (RMC) 256, 382
 - daemon 256, 382
- rPerf numbers 139
- Run-Time Abstraction Services (RTAS) 75

S

- sar command 239–240, 272
- SCSI Command Data Blocks (CDBs) 214
- SCSI Remote DMA Protocol (SRP) 214
- segment lookaside buffer 88

- server consolidation 95
- server provisioning 96
- Shared Ethernet Adapter 7, 186, 254, 306
 - controlling threading 204
 - CPU sizing 199
 - sizing guidelines 197
- shared processor partitions 95
 - guidelines 133
- silicon-on-insulator (SOI) 10
- simultaneous multithreading 10, 12, 41–72
 - benchmarks 61–71
 - definition 2
 - dynamic feedback 44
 - dynamic thread switching 44
 - effect on processor usage 117
 - hardware thread prioritization 44
 - IC pipeline stage 45
 - IF pipeline stage 45
 - instruction cache 45
 - performance benefits 59
 - scheduling 55
 - snooze and snooze delay 47
 - software considerations 55
 - software controlled thread priority 44
 - software thread prioritization 44
 - threads 41
 - translation facility 45
 - two instruction streams 44
- single-threaded execution mode 46
 - dormant state 46
- slice 27
- slow clock ticks 128
- smtctl command 48
- snoozing 48
- software controlled thread priority 44
- SRP Informational Units (IUs) 214
- superscalar 18, 43
- supervisor mode 51
- SVG 252
- sysconfig subroutine 272
- system firmware 75
- System Licensed Internal Code (SLIC) 75
- System Reset Interrupt 77
- system tuning 3

T

- tagged packets
 - See virtual LAN (VLAN) tagged packets

- Technology Independent Machine Interface (TIMI) 75
- thread switching capabilities 44
- Time Base 85
- Time Function History Scheduling (TFHS) algorithm 89
- topas command 240, 276
- TotalStorage Enterprise Storage Subsystem (ESS) 218
- trace command 241
- Translation Control Entry (TCE) 80, 149
- translation lookaside buffer 88
- Transmission Control Protocol (TCP) 175
- trcrpt command 241
- Two-way set associative 21

U

- Uncapped mode
 - See Micro-Partitioning
 - uncapped mode
- untagged packets 166
- User Datagram Protocol (UDP) 175
- user mode 51

V

- variable capacity weight 104
- varyonvg command 253
- VGDA 252
- VGSA 252
- victim cache 27
- virtual adapters 145
- virtual Ethernet 7, 163
 - as a boot device 170
 - benefits 170
 - communication with external networks 168
 - Hypervisor switch implementation 171
 - implementation guidelines 185
 - inter-partition communication 164
 - interpartition communication 167
 - IPv6 support 170
 - limitations and considerations 171
 - MAC addresses 169
 - MTU sizes 170
 - netperf benchmark 174
 - performance considerations 174
 - Shared Ethernet Adapter functionality 186
 - transmission speed 170
 - trunk adapter 171

- virtual host bridge 148, 210
- virtual I/O
 - Command/Response Queue 149
 - definition 1
 - multi-path I/O 218
- Virtual I/O Server 152
 - chdev command 205
 - command line interface 152
 - dead gateway detection 159
 - help command 153
 - hosted partitions 206
 - hosting partition 206
 - implementation guidelines 196
 - installation 152
 - interactive mode 153
 - ioscli command 153
 - limitations and considerations 155
 - lsdev command 204
 - network interface backup 157
 - padmin account 153, 204
 - performance results 191
 - restricted Korn shell 153
 - traditional mode 153
- virtual input/output 85
 - infrastructure 147
 - LRDMA 150
 - Remote DMA (RDMA) 149
 - virtual Ethernet 163
 - virtual SCSI 205
- virtual interrupt source controller 148
- virtual LAN
 - AIX 5L support 165
 - definition 1
 - overview 164
- virtual LAN (VLAN) 164
 - communication with external networks 168
 - IEEE VLAN header 171
 - implementation guidelines 185
 - port virtual LAN ID 166
 - tagged packets 166
- virtual memory management (VMM) 90
- virtual processor 89, 95, 97
 - maximum number 98
 - reasonable settings 103
 - time slice 120
- virtual processor states
 - expired 77
 - not-runnable 77
 - runnable 77
 - running 77
- virtual processors 85
 - dispatch latency 90
 - locking considerations 121
 - states of 104
 - entitlement expired 104
 - not-runnable 104
 - runnable 104
 - running 104
- virtual SCSI 7, 205–206
 - AIX device configuration 210
 - bandwidth 224
 - client adapter 206
 - command tag queueing 216
 - configuration example 228
 - configuring for redundancy 217
 - hosted partitions 206
 - hosting partitions 206
 - LVM mirroring 218
 - memory descriptor mapping 214
 - multi-path I/O 218
 - performance considerations 220
 - physical mirroring 218
 - Reliable Command/Response Transport 212
 - SCSI Remote DMA Protocol (SRP) 214
 - target 206
 - virtual host bridge 210
 - virtual SCSI
 - server adapter 206
- virtual server farms 96
- virtualization
 - definition 1
 - importance of 2
 - memory affinity considerations 126
- vmo command 251
- vmstat command 239–240, 268
- Voluntary context switching 124

W

- Web System Management tool 394
- weight, uncapped 101
- Workload Manager 6, 374

X

- xlplm command 391
- xlpstat command 391
- xmperf command 245, 278



Advanced POWER Virtualization on IBM @server p5 Servers: Architecture and Performance Considerations

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Advanced POWER Virtualization on IBM @server p5 Servers: Architecture and Performance Considerations



Detailed description of the POWER5 architecture

This IBM Redbook addresses the issues surrounding performance of the IBM System p5 and IBM @server p5 systems. Many features that have been introduced in these systems are discussed in this book.

In-depth analysis of Advanced POWER Virtualization features

The new systems use the POWER5 processors, which consist of a dual processor core with each core supporting two hardware threads of execution. This technology is referred to as simultaneous multithreading. We provide an in-depth look into the POWER5 processor architecture.

Performance analysis and application tuning

The POWER Hypervisor has been enhanced, so detailed descriptions of POWER Hypervisor and Hypervisor system calls are given.

This book also covers the concepts of virtual processors, virtual storage, and virtual networking, which are also part of the new systems.

With all of these new technologies and features, system administrators must realize the performance impact they may have. We look at these performance issues by providing the results of performance tests that were conducted in the course of writing.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks