

Performance Measurement and Visualization on the Cray XT

Luiz DeRose
Programming Environments Director
Cray Inc.

Outline

- Cray Performance Analysis Toolset Overview
- Using Hardware Performance Counters
- Visualization with Cray Apprentice²
- Load imbalance detection
- Documentation for the Cray Performance Toolset

Cray Toolset Design Goals



- **Assist** the user with application performance analysis and optimization
 - Help user identify important and meaningful information from potentially massive data sets
 - Help user identify problem areas instead of just reporting data
 - Bring optimization knowledge to a wider set of users

- Focus on **ease** of use and **intuitive** user interfaces
 - Automatic program instrumentation
 - Automatic analysis

- Target **scalability** issues in all areas of tool development
 - Data management
 - Storage, movement, presentation

Performance Analysis with Cray Tools



- Important performance statistics:
 - Top time consuming routines
 - Load balance across computing resources
 - Communication overhead
 - Cache utilization
 - FLOPS
 - Vectorization (SSE instructions)
 - Ratio of computation versus communication

The Cray Performance Analysis Framework



- Supports traditional post-mortem performance analysis
 - Automatic identification of performance problems
 - Indication of causes of problems
 - Suggestions of modifications for performance improvement
- **CrayPat**
 - **pat_build**: automatic instrumentation (no source code changes needed)
 - **run-time library** for measurements (transparent to the user)
 - **pat_report** for performance analysis reports
 - **pat_help**: online help utility
- **Cray Apprentice²**
 - Graphical performance analysis and visualization tool

Performance Data Collection



- Two dimensions
 - When performance collection is triggered
 - Externally (asynchronous)
 - Sampling
 - » Timer interrupt
 - » Hardware counters overflow
 - Internally (synchronous)
 - Code instrumentation
 - » Event based
 - » Automatic or manual instrumentation
 - How performance data is recorded
 - Profile ::= Summation of events over time
 - run time summarization (functions, call sites, loops, ...)
 - Trace file ::= Sequence of events over time

Application Instrumentation with pat_build



- **No** source code or makefile **modification** required
 - **Automatic instrumentation** at group (function) level
 - Groups: mpi, io, heap, math SW, ...

- Performs link-time instrumentation
 - **Requires object files**
 - Instruments optimized code
 - Generates stand-alone instrumented program
 - Preserves original binary
 - Supports **sample-based** and **event-based** instrumentation

CrayPat API - For Fine Grain Instrumentation



- Fortran

```
include "pat_apif.h"
...
call PAT_region_begin(id, "label", ierr)
do i = 1,n
...
enddo
call PAT_region_end(id, ierr)
```

- C & C++

```
include <pat_api.h>
...
ierr = PAT_region_begin(id, "label");
< code segment >
ierr = PAT_region_end(id);
```

Where to Run Instrumented Applications



- **MUST run on Lustre** (/work/... , /lus/..., /scratch/..., etc.)
- Number of files used to store raw data
 - 1 file created for program with 1 – 256 processes
 - \sqrt{n} files created for program with 257 – n processes
 - Ability to customize with **PAT_RT_EXPFIL_MAX**

pat_report



- Performs data conversion
 - Combines information from binary with raw performance data
- Performs analysis on data
- Generates text report of performance results
- Formats data for input into Cray Apprentice²

Automatic Profiling Analysis



- **Analyze** the performance data and **direct the user** to meaningful information
- **Simplifies** the procedure to instrument and collect performance data for novice users
- Based on a two phase mechanism
 1. **Automatically** detects the most time consuming functions in the application and feeds this information back to the tool for further (and focused) data collection
 2. Provides performance information on the most significant parts of the application

Steps to Collecting Performance Data



- Access performance tools software

```
% module load xt-craypat apprentice2
```
- Build application keeping .o files (CCE: -h keepfiles)

```
% make clean
% make
```
- Instrument application for automatic profiling analysis
 - You should get an instrumented program a.out+pat

```
% pat_build -O apa a.out
```
- Run application to get top time consuming routines
 - You should get a performance file ("**<sdatafile>.xf**") or multiple files in a directory **<sdatadir>**

```
% aprun ... a.out+pat (or qsub <pat script>)
```

Steps to Collecting Performance Data (2)

- Generate report and .apa instrumentation file

```
% pat_report -o my_sampling_report [<sdatafile>.xf |
<sdatadir>]
```

- Inspect .apa file and sampling report
- Verify if additional instrumentation is needed
 - Check the sampling report for possible regions to instrument with the CrayPat API

Sampling Output (Table 1)

Notes for table 1:

...

Table 1: Profile by Function

Samp %	Samp	Imb. Samp	Imb. Samp %	Function
100.0%	775	--	--	Total
94.2%	730	--	--	USER
43.4%	336	8.75	2.6%	mlwxyz_
16.1%	125	6.28	4.9%	half_
8.0%	62	6.25	9.5%	full_
6.8%	53	1.88	3.5%	artv_
4.9%	38	1.34	3.6%	bnd_
3.6%	28	2.00	6.9%	currentf_
2.2%	17	1.50	18.6%	bndsf_
1.7%	13	1.97	13.5%	model_
1.4%	11	1.53	7.2%	cfl_
1.3%	10	0.75	7.0%	currenth_
1.0%	8	5.28	41.9%	bndbc_
1.0%	8	8.28	53.4%	bndtc_
5.4%	42	--	--	MPI
1.9%	15	4.62	23.9%	mpi_sendrecv_
1.8%	14	16.53	55.0%	mpi_bcast_
1.7%	13	5.66	30.7%	mpi_barrier_

Samp % provides absolute percentages

Sampling Output (Table 2)

Table 2: Profile by Group, Function, and Line

Samp %	Samp	Imb. Samp	Imb. Samp %	Group Function Source Line PE='HIDE'
100.0%	777	--	--	Total
94.2%	732	--	--	USER
43.4%	337	--	--	mlwxyz ldr/mhd3d/src/mlwxyz.f
2.1%	16	1.47	8.9%	line.39
2.8%	22	2.25	9.7%	line.78
1.2%	9	1.09	11.3%	line.116
1.4%	11	1.22	10.5%	line.129
2.2%	17	2.12	11.5%	line.139
2.7%	21	0.84	4.0%	line.568
1.3%	10	1.72	14.8%	line.604
2.4%	19	0.72	3.7%	line.634
16.1%	125	--	--	half ldr/mhd3d/src/half.f
5.4%	42	6.41	13.8%	line.28
10.7%	83	5.41	13.8%	line.40
8.0%	62	--	--	full ldr/mhd3d/src/full.f
8.0%	62	6.31	9.6%	line.22
5.4%	42	--	--	MPI
1.9%	15	4.62	23.9%	mpi_sendrecv
1.8%	14	16.53	55.0%	mpi_bcast
1.7%	13	5.66	30.7%	mpi_barrier

APA File Example

```
# You can edit this file, if desired, and use it
# to reinstrument the program for tracing like this:
#
# pat_build -O mhd3d.Oapa.x+4125-401sdt.apa
#
# These suggested trace options are based on data from:
# /home/crayadm/ldr/mhd3d/run/mhd3d.Oapa.x+4125-401sdt.ap2,
# /home/crayadm/ldr/mhd3d/run/mhd3d.Oapa.x+4125-401sdt.xf
#
# .....
#
# HWPC group to collect by default.
#
# -Drtenv=PAT_RT_HWPC=1 # Summary with instructions metrics.
#
# .....
#
# Libraries to trace.
#
# -g mpi
#
# .....
#
# User-defined functions to trace, sorted by % of samples.
#
# Limited to top 200. A function is commented out if it has < 1%
# of samples, or if a cumulative threshold of 90% has been reached,
# or if it has size < 200 bytes.
#
# Note: -u should NOT be specified as an additional option.

# 43.37% 99659 bytes
# -T mlwxyz_

# 16.09% 17615 bytes
# -T half_

# 6.82% 6846 bytes
# -T artv_

# 1.29% 5352 bytes
# -T currenth_

# 1.03% 25294 bytes
# -T bndbo_

# Functions below this point account for less than 10% of samples.

# 1.03% 31240 bytes
# -T bndto_

...

# .....
#
# -o mhd3d.x+apa # New instrumented program.
#
# /work/crayadm/ldr/mhd3d/mhd3d.x # Original program.
```

-g tracegroup (subset)

- blas Basic Linear Algebra subprograms
- CAF Co-Array Fortran (Cray CCE compiler only)
- HDF5 manages extremely large and complex data collections
- heap dynamic heap
- io includes stdio and sysio groups
- lapack Linear Algebra Package
- math ANSI math
- mpi MPI
- omp OpenMP API
- omp-rtl OpenMP runtime library (not supported on Catamount)
- pthreads POSIX threads (not supported on Catamount)
- shmem SHMEM
- sysio I/O system calls
- system system calls
- upc Unified Parallel C (Cray CCE compiler only)

For a full list, please see `man pat_build`

Steps to Collecting Performance Data (2)

- Instrument application for further analysis (`a.out+apa`)

```
% pat_build -O <apafilename>.apa
```

- Run application

```
% aprun ... a.out+apa (or qsub <apa script>)
```

- Generate text report and visualization file (`.ap2`)

```
% pat_report -o my_text_report.txt [<datafile>.xf |  
<datadir>]
```

- View report in text and/or with Cray Apprentice²

```
% app2 <datafile>.ap2
```

pat_report: Flat Profile



Table 1: Profile by Function Group and Function

Time %	Time	Imb. Time	Imb. Time %	Calls	Group Function PE='HIDE'
100.0%	104.593634	--	--	22649	Total
71.0%	74.230520	--	--	10473	MPI
69.7%	72.905208	0.508369	0.7%	125	mpi_allreduce_
1.0%	1.050931	0.030042	2.8%	94	mpi_alltoall_
25.3%	26.514029	--	--	73	USER
16.7%	17.461110	0.329532	1.9%	23	selfgravity_
7.7%	8.078474	0.114913	1.4%	48	ffte4_
2.5%	2.659429	--	--	435	MPI_SYNC
2.1%	2.207467	0.768347	26.2%	172	mpi_barrier_(sync)
1.1%	1.188998	--	--	11608	HEAP
1.1%	1.166707	0.142473	11.1%	5235	free

pat_report: Message Stats by Caller



Table 4: MPI Message Stats by Caller

MPI Msg Bytes	MPI Msg Count	MsgSz <16B Count	4KB<= MsgSz <64KB Count	Function Caller PE[mmm]
15138076.0	4099.4	411.6	3687.8	Total
15138028.0	4093.4	405.6	3687.8	MPI_ISEND
8080500.0	2062.5	93.8	1968.8	calc2_ MAIN_
8216000.0	3000.0	1000.0	2000.0	pe.0
8208000.0	2000.0	--	2000.0	pe.9
6160000.0	2000.0	500.0	1500.0	pe.15
6285250.0	1656.2	125.0	1531.2	calc1_ MAIN_
8216000.0	3000.0	1000.0	2000.0	pe.0
6156000.0	1500.0	--	1500.0	pe.3
6156000.0	1500.0	--	1500.0	pe.5

Using Hardware Performance Counters

Luiz DeRose
Programming Environments Director
Cray Inc.

Hardware Performance Counters

■ AMD Opteron Hardware Performance Counters

- **Four** 48-bit performance counters.
 - Each counter can monitor a single event
 - Count specific processor events
 - » the processor increments the counter when it detects an occurrence of the event
 - » (e.g., cache misses)
 - Duration of events
 - » the processor counts the number of processor clocks it takes to complete an event
 - » (e.g., the number of clocks it takes to return data from memory after a cache miss)
- Time Stamp Counters (TSC)
 - Cycles (user time)

PAPI Predefined Events



- Common set of events deemed relevant and useful for application performance tuning
 - Accesses to the memory hierarchy, cycle and instruction counts, functional units, pipeline status, etc.
 - The “papi_avail” utility shows which predefined events are available on the system – execute on compute node

- PAPI also provides access to native events
 - The “papi_native_avail” utility lists all AMD native events available on the system – execute on compute node

- Information on PAPI and AMD native events
 - pat_help counters
 - man papi_counters
 - For more information on AMD counters:
 - http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/26049.PDF

Hardware Counters Selection



- PAT_RT_HWPC <set number> | <event list>
 - Specifies hardware counter events to be monitored
 - A set number can be used to select a group of predefined hardware counters events (recommended)
 - CrayPat provides 19 groups on the Cray XT systems

 - Alternatively a list of hardware performance counter event names can be used
 - Both formats can be specified at the same time, with later definitions overriding previous definitions
 - A statistical (**multiplexing**) approach is used when more than 4 events are provided

 - Hardware counter events are not collected by default

Hardware Counters Overhead Elimination

Hardware performance counter events:

```

PAPI_L1_DCM  Level 1 data cache misses
CYCLES_RTC   User Cycles (approx, from rtc)
PAPI_L1_DCA  Level 1 data cache accesses
PAPI_TLB_DM  Data translation lookaside buffer misses
PAPI_FP_OPS  Floating point operations
    
```

Estimated minimum overhead per call of a traced function, which was subtracted from the data shown in this report (for raw data, use the option: `-s overhead=include`):

```

PAPI_L1_DCM      8.040  misses
PAPI_TLB_DM      0.005  misses
PAPI_L1_DCA     474.080  refs
PAPI_FP_OPS      0.000  ops
CYCLES_RTC      1863.680  cycles
Time             0.693  microseconds
    
```

PAT_RT_HWPC=1 (Summary with TLB)

```

PAPI_TLB_DM  Data translation lookaside buffer misses
PAPI_L1_DCA  Level 1 data cache accesses
PAPI_FP_OPS  Floating point operations
DC_MISS      Data Cache Miss
User_Cycles  Virtual Cycles
    
```

=====

USER

```

-----
Time%                98.3%
Time                 4.434402 secs
Imb.Time             -- secs
Imb.Time%            --
Calls                0.001M/sec    4500.0 calls
PAPI_L1_DCM          14.820M/sec    65712197 misses
PAPI_TLB_DM          0.902M/sec    3998928 misses
PAPI_L1_DCA          333.331M/sec    1477996162 refs
PAPI_FP_OPS          445.571M/sec    1975672594 ops
User time (approx)   4.434 secs    11971868993 cycles 100.0%Time
Average Time per Call
CrayPat Overhead : Time      0.1%
HW FP Ops / User time      445.571M/sec    1975672594 ops  4.1%peak(DP)
HW FP Ops / WCT            445.533M/sec
Computational intensity     0.17 ops/cycle    1.34 ops/ref
MFLOPS (aggregate)         1782.28M/sec
TLB utilization             369.60 refs/miss    0.722 avg uses
D1 cache hit,miss ratios    95.6% hits    4.4% misses
D1 cache utilization (misses) 22.49 refs/miss    2.811 avg hits
=====
    
```

PAT_RT_HWPC=1

Flat profile data

Hard counts

Derived metrics

PAT_RT_HWPC=2 (L1 and L2 Metrics)



```

=====
USER
-----
Time%                               98.3%
Time                               4.436808 secs
Imb.Time                           -- secs
Imb.Time%                           --
Calls                               0.001M/sec      4500.0 calls
DATA_CACHE_REFILLS:
  L2_MODIFIED:L2_OWNED:
    L2_EXCLUSIVE:L2_SHARED      9.821M/sec      43567825 fills
DATA_CACHE_REFILLS_FROM_SYSTEM:
  ALL                            24.743M/sec     109771658 fills
PAPI_L1_DCM                      14.824M/sec     65765949 misses
PAPI_L1_DCA                      332.960M/sec    1477145402 refs
User time (approx)                4.436 secs     11978286133 cycles 100.0%Time
Average Time per Call              0.000986 sec
CrayPat Overhead : Time            0.1%
D1 cache hit,miss ratios           95.5% hits     4.5% misses
D1 cache utilization (misses)      22.46 refs/miss 2.808 avg hits
D1 cache utilization (refills)     9.63 refs/refresh 1.204 avg uses
D2 cache hit,miss ratio            28.4% hits     71.6% misses
D1+D2 cache hit,miss ratio         96.8% hits     3.2% misses
D1+D2 cache utilization            31.38 refs/miss 3.922 avg hits
System to D1 refill                24.743M/sec     109771658 lines
System to D1 bandwidth             1510.217MB/sec  7025386144 bytes
D2 to D1 bandwidth                 599.398MB/sec  2788340816 bytes
=====
  
```

PAT_RT_HWPC=5 (Floating point mix)



```

=====
USER
-----
Time%                               98.4%
Time                               4.426552 secs
Imb.Time                           -- secs
Imb.Time%                           --
Calls                               0.001M/sec      4500.0 calls
RETIRED_MMX_AND_FP_INSTRUCTIONS:
  PACKED_SSE_AND_SSE2          454.860M/sec    2013339518 instr
PAPI_FML_INS                    156.443M/sec    692459506 ops
PAPI_FAD_INS                     289.908M/sec    1283213088 ops
PAPI_FDV_INS                      7.418M/sec     32834786 ops
User time (approx)                4.426 secs     11950955381 cycles 100.0%Time
Average Time per Call              0.000984 sec
CrayPat Overhead : Time            0.1%
HW FP Ops / Cycles                 0.17 ops/cycle
HW FP Ops / User time              446.351M/sec    1975672594 ops  4.1%peak(DP)
HW FP Ops / WCT                    446.323M/sec
FP Multiply / FP Ops               35.0%
FP Add / FP Ops                    65.0%
MFLOPS (aggregate)                1785.40M/sec
=====
  
```

PAT_RT_HWPC=12 (QC Vectorization)



```

=====
USER
-----
Time%                98.3%
Time                4.434163 secs
Imb.Time            -- secs
Imb.Time%           --
Calls               0.001M/sec    4500.0 calls
RETIRED_SSE_OPERATIONS:
SINGLE_ADD_SUB_OPS:
SINGLE_MUL_OPS              0 ops
RETIRED_SSE_OPERATIONS:
DOUBLE_ADD_SUB_OPS:
DOUBLE_MUL_OPS           225.224M/sec    998097162 ops
RETIRED_SSE_OPERATIONS:
SINGLE_ADD_SUB_OPS:
SINGLE_MUL_OPS:OP_TYPE              0 ops
RETIRED_SSE_OPERATIONS:
DOUBLE_ADD_SUB_OPS:
DOUBLE_MUL_OPS:OP_TYPE  445.818M/sec    1975672594 ops
User time (approx)    4.432 secs  11965243964 cycles  99.9%Time
Average Time per Call              0.000985 sec
CrayPat Overhead : Time    0.1%
=====

```

Vectorization Example



```

=====
USER / calc2_
-----
Time%                28.2%
Time                0.600875 secs
Imb.Time            0.069872 secs
Imb.Time%           11.9%
Calls               864.9 /sec    500.0 calls
RETIRED_SSE_OPERATIONS:
SINGLE_ADD_SUB_OPS:
SINGLE_MUL_OPS              0 ops
RETIRED_SSE_OPERATIONS:
DOUBLE_ADD_SUB_OPS:
DOUBLE_MUL_OPS           369.139M/sec    213408500 ops
RETIRED_SSE_OPERATIONS:
SINGLE_ADD_SUB_OPS:
SINGLE_MUL_OPS:OP_TYPE              0 ops
RETIRED_SSE_OPERATIONS:
DOUBLE_ADD_SUB_OPS:
DOUBLE_MUL_OPS:OP_TYPE  369.139M/sec    213408500 ops
User time (approx)    0.578 secs  1271875000 cycles  96.2%Time

When compiled with fast:
=====
USER / calc2_
-----
Time%                24.3%
Time                0.485654 secs
Imb.Time            0.146551 secs
Imb.Time%           26.4%
Calls               0.001M/sec    500.0 calls
RETIRED_SSE_OPERATIONS:
SINGLE_ADD_SUB_OPS:
SINGLE_MUL_OPS              0 ops
RETIRED_SSE_OPERATIONS:
DOUBLE_ADD_SUB_OPS:
DOUBLE_MUL_OPS           208.641M/sec    103016531 ops
RETIRED_SSE_OPERATIONS:
SINGLE_ADD_SUB_OPS:
SINGLE_MUL_OPS:OP_TYPE              0 ops
RETIRED_SSE_OPERATIONS:
DOUBLE_ADD_SUB_OPS:
DOUBLE_MUL_OPS:OP_TYPE  415.628M/sec    205216531 ops
User time (approx)    0.494 secs  1135625000 cycles  100.0%Time
=====

```

How do I interpret these derived metrics?

- The following thresholds are guidelines to identify if optimization is needed:
 - **Computational Intensity: < 0.5 ops/ref**
 - This is the ratio of FLOPS by L&S
 - Measures how well the floating point unit is being used
 - **FP Multiply / FP Ops or FP Add / FP Ops: < 25%**
 - **Vectorization: < 1.5**

Memory Hierarchy Thresholds

- **TLB utilization: < 90.0%**
 - Measures how well the memory hierarchy is being utilized with regards to TLB
 - This metric depends on the computation being single precision or double precision
 - A page has 4 Kbytes. So, one page fits 512 double precision words or 1024 single precision words
 - TLB utilization < 1 indicates that not all entries on the page are being utilized between two TLB misses
- **Cache utilization: < 1 (D1 or D1+D2)**
 - A cache line has 64 bytes (8 double precision words or 16 single precision words)
 - **Cache utilization < 1** indicates that not all entries on the cache line are being utilized between two cache misses
- **D1 cache hit (or miss) ratios: < 90% (> 10%)**
- **D1 + D2 cache hit (or miss) ratios: < 92% (> 8%)**
 - D1 and D2 caches on the Opteron are complementary
 - This metric provides a view of the Total Cache hit (miss) ratio

Profile Visualization with Cray Apprentice²

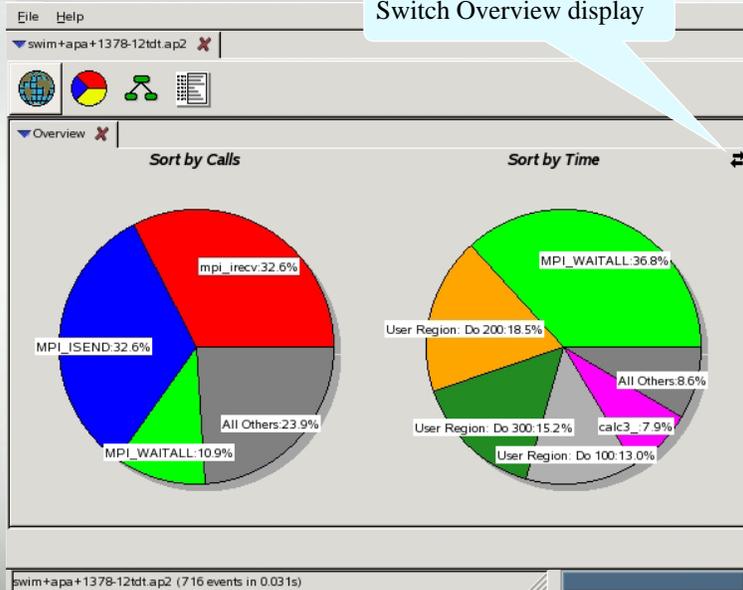
Luiz DeRose
Programming Environments Director
Cray Inc.

Cray Apprentice²

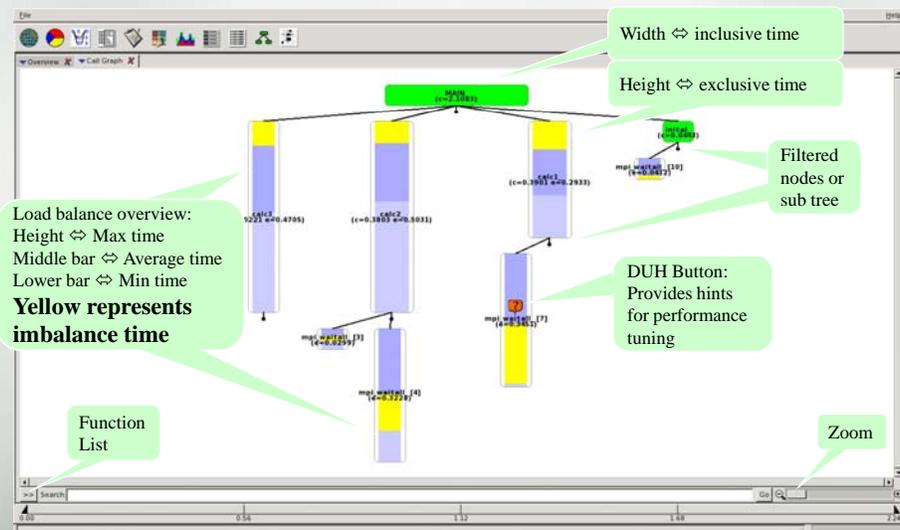
- Call graph profile
 - Communication statistics
 - Time-line view
 - Communication
 - I/O
 - Activity view
 - Pair-wise communication statistics
 - Text reports
 - Source code mapping
- Cray Apprentice²
 - is target to help and correct:
 - Load imbalance
 - Excessive communication
 - Network contention
 - Excessive serialization
 - I/O Problems



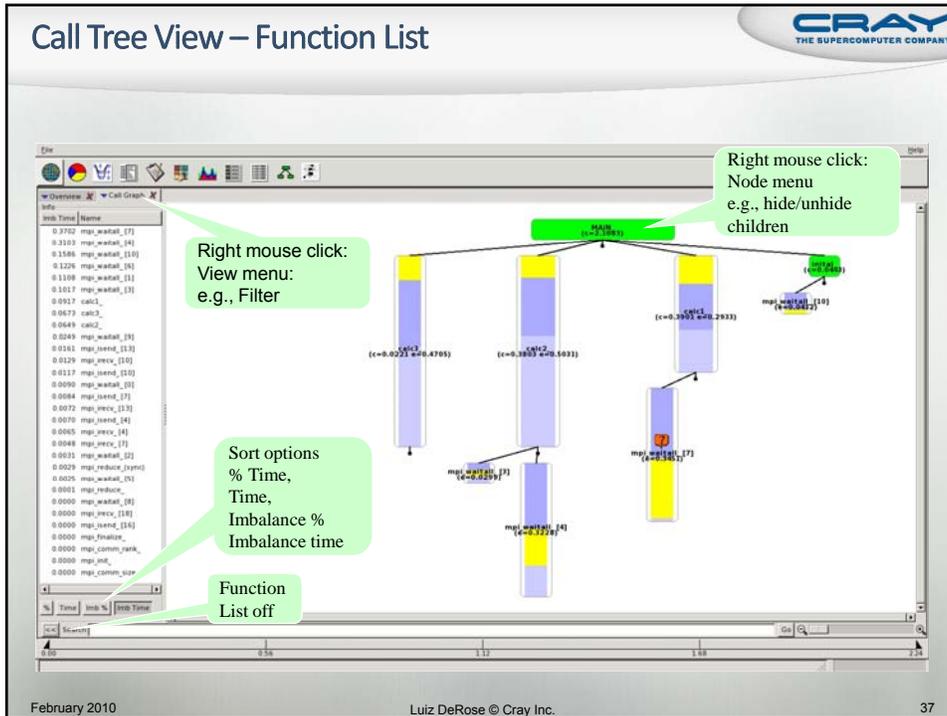
Statistics Overview



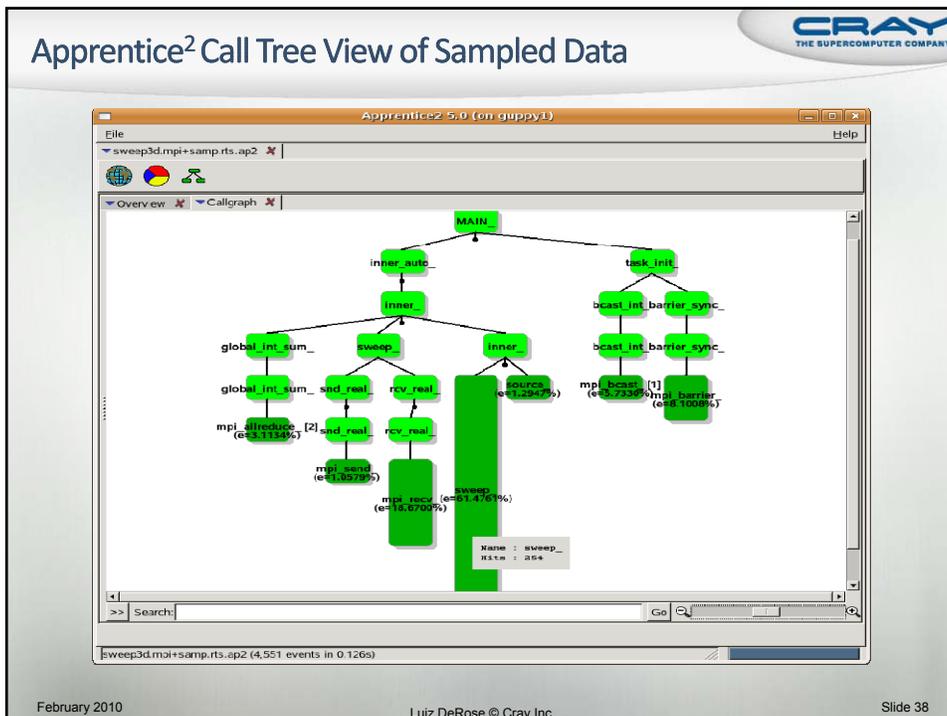
Call Tree View



Call Tree View – Function List



Apprentice² Call Tree View of Sampled Data



Source Mapping from Call Tree

```
Apprentice2.2.3
vresp3d+mp06p+tr.vms.gz
Overview | Traffic Report | Activity | Call Graph | vresp.f
165
166 c angle pipelining loop (batches of mmi angles)
167 c
168   DO m0 = 1, nmo
169     mio = (m0-1)*mmi
170
171 c K=Inflows (k=k0 boundary)
172 c
173   if (k2.lt.0 .or. kbc.eq.0) then
174     do mi = 1, mmi
175       do j = 1, jt
176         do i = 1, it
177           phikb(i,j,mi) = 0.0d+0
178         end do
179       end do
180     end do
181   else
182     if (.dca) then
183       leak = 0.0
184       k = k0 - k2
185       do mi = 1, mmi
186         m = mi + mio
187         do j = 1, jt
188           do i = 1, it
189             phikb(i,j,mi) = phikb(i,j,m)
190             leak = leak
191             + vts1(m)*phikb(i,j,mi)*di(i)*dj(j)
192             Face(i,j,k+k2,2) = Face(i,j,k+k2,2)
193             + vts1(m)*phikb(i,j,mi)
194           end do
195         end do
196       end do
197       leakage(S) = leakage(S) + leak
198     end if
199   end if
```

Detecting Load Imbalance on the Cray XT

Luiz DeRose
Programming Environments Director
Cray Inc.

Motivation for Load Imbalance Analysis



- Increasing system software and architecture complexity
 - Current trend in high end computing is to have systems with tens of thousands of processors
 - This is being accentuated with multi-core processors
- Applications have to be very well balanced in order to perform at scale on these MPP systems
 - Efficient application scaling includes a balanced use of requested computing resources
- Desire to minimize computing resource “waste”
 - Identify slower paths through code
 - Identify inefficient “stalls” within an application

Cray Tools Load Imbalance Support



- Very few performance tools focus on load imbalance
 - Need standard metrics
 - Need intuitive way of presentation
- CrayPat support:
 - MPI sync time
 - Imbalance time and %
 - MPI rank placement suggestions
 - OpenMP Performance Metrics
- Cray Apprentice² support:
 - Load imbalance visualization

MPI Sync Time

- Measure load imbalance in programs instrumented to trace MPI functions to determine if MPI ranks arrive at collectives together
- Separates potential load imbalance from data transfer
- Sync times reported by default if MPI functions traced
- If desired, PAT_RT_MPI_SYNC=0 deactivated this feature

Profile with Load Distribution by Groups

Table 1: Profile by Function Group and Function

Time %	Time	Imb. Time	Imb. Time %	Calls	Group Function PE='HIDE'
100.0%	513.581345	--	--	368418.8	Total
69.5%	356.710479	--	--	37064.0	USER
24.9%	127.809860	34.800347	21.5%	1.0	main
23.7%	121.692894	30.797216	20.3%	480.0	montum_
7.8%	40.231832	14.622935	26.8%	480.0	cnuity_
6.1%	31.135595	16.354488	34.6%	34174.0	mod_xc_xctilr_
5.9%	30.404372	14.887689	33.0%	482.0	hybgen_
1.1%	5.435825	2.256039	29.4%	1446.0	dpudpv_
24.7%	127.038044	--	--	325626.8	MPI
23.0%	117.877116	307.988571	72.6%	79473.6	mpi_waitall_
1.4%	7.203319	5.428131	43.1%	79470.8	mpi_startall_
5.8%	29.832822	--	--	5728.0	MPI_SYNC
4.9%	25.147203	30.818426	55.3%	2814.0	mpi_allreduce_(sync)

Imbalance Time

- Metric based on execution time
- It is dependent on the type of activity:
 - User functions
Imbalance time = Maximum time – Average time
 - Synchronization (Collective communication and barriers)
Imbalance time = Average time – Minimum time
- Identifies computational code regions and synchronization calls that could benefit most from load balance optimization
- Estimates how much overall program time could be saved if corresponding section of code had a perfect balance
 - Represents upper bound on “potential savings”
 - Assumes other processes are waiting, not doing useful work while slowest member finishes

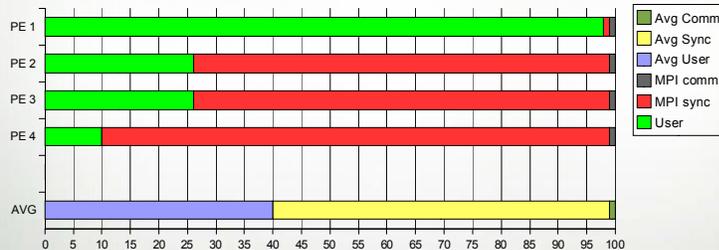
Load balance metric - rationale

Between two barriers

User: $\text{Imb} = \text{Max} - \text{Avg} = 99 - 40 = 59$

MPI Sync: $\text{Avg} = 59$

MPI Sync+Comm: $\text{Avg} - \text{Min} = 60 - 1 = 59$

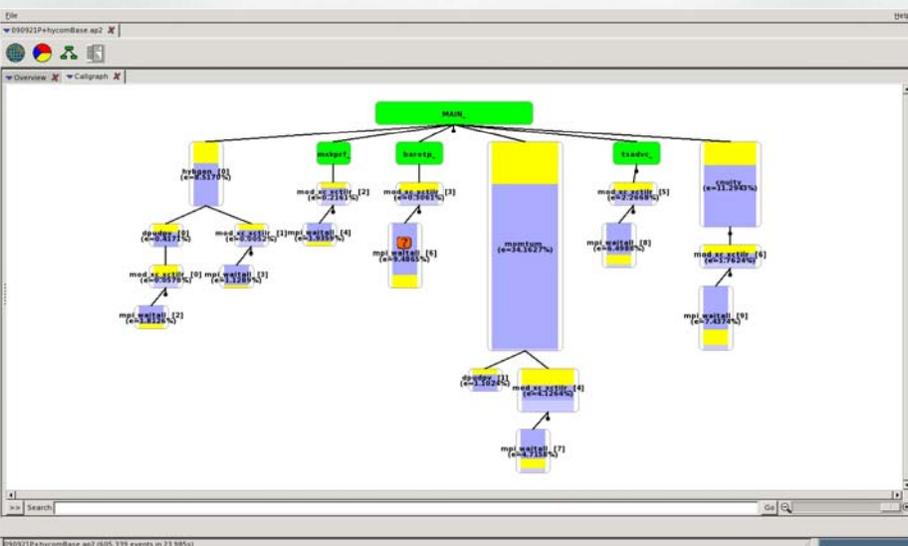


Imbalance %

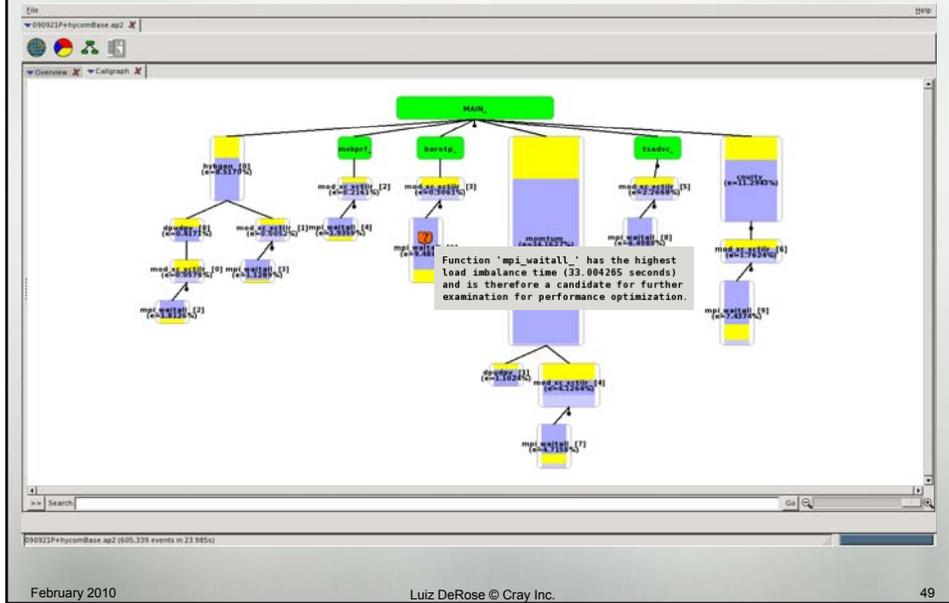
$$\text{Imbalance\%} = 100 \times \frac{\text{Imbalance time}}{\text{Max Time}} \times \frac{N}{N - 1}$$

- Represents % of resources available for parallelism that is “wasted”
- Corresponds to % of time that rest of team is not engaged in useful work on the given function
- Perfectly balanced code segment has imbalance of 0%
- Serial code segment has imbalance of 100%

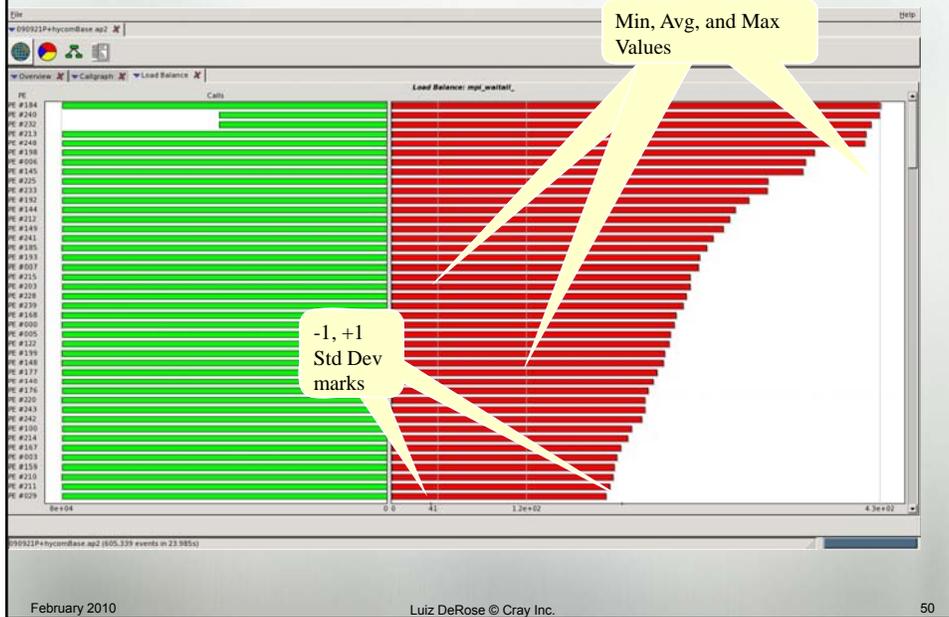
Call Tree Visualization (Hycom)



Call Tree Visualization (Hycom)



Load Distribution



MPI Rank Reorder

- MPI rank placement with environment variable



- Distributed placement
- SMP style placement
- Folded rank placement
- User provided rank file

MPI Rank Placement Suggestions

- When to use?
 - Point-to-point communication consumes significant fraction of the program time and have a significant imbalance
 - `pat_report -O mpi_sm_rank_order ...`
 - When there seems to be a load imbalance of another type
 - Can get a suggested rank order file based on user time
 - `pat_report -O mpi_rank_order ...`
 - Can have a different metric for load balance
 - `pat_report -O mpi_rank_order -s mro_metric=DATA_CACHE_MISSES ...`
- Information in resulting report
 - Available if MPI functions traced (`-g mpi`)
 - Custom placement files automatically generated
 - Report provides quad core and dual core suggestions
 - 2, 4, and 8 cores per node
 - See table notes in resulting report for instructions on how to use
- Set `MPICH_RANK_REORDER_METHOD` environment variable
 - Set to numerical value or `MPICH_RANK_ORDER` file from `pat_report`

Rank Reorder Example - hycom

pat_report -O load_balance

Table 2: Load Balance across PE's by FunctionGroup

Time %	Time	Calls	Group PE[mmm]
100.0%	513.581345	368418.8	Total

69.5%	356.710479	37064.0	USER

0.3%	441.604004	37064.0	pe.73
0.3%	395.835561	37064.0	pe.62
0.0%	23.942438	37064.0	pe.184
=====			
24.7%	127.038044	325626.8	MPI

0.3%	437.244595	239807.0	pe.232
0.1%	90.023179	317002.0	pe.12
0.0%	49.907519	317002.0	pe.73
=====			
5.8%	29.832822	5728.0	MPI_SYNC

0.0%	62.473245	5728.0	pe.184
0.0%	27.165827	5728.0	pe.25
0.0%	10.940857	5728.0	pe.56
=====			

Example: -O mpi_rank_order (hycom)

Notes for table 1:

To maximize the load balance across nodes, specify a Rank Order with small Max and Avg USER Time per node for the target number of cores per node.

To specify a Rank Order with a numerical value, set the environment variable MPICH_RANK_REORDER_METHOD to the given value.

To specify a Rank Order with a letter value 'x', set the environment variable MPICH_RANK_REORDER_METHOD to 3, and copy or link the file MPICH_RANK_ORDER.x to MPICH_RANK_ORDER.

Table 1: Suggested MPI Rank Order
USER Time per MPI rank

	Max USER Time	Avg USER Time	Max Rank		
1015754691532	820499583863	73			

Four cores per node: USER Time per node					
Rank Order	Max USER Time	Max/ SMP	Avg USER Time	Avg/ SMP	Max Node Ranks
d	3441386576933	85.0%	3281998335454	100.0%	113,227,115,197
0	3857929506520	95.3%	3281998335454	100.0%	49,112,174,236
2	3911647317171	96.7%	3281998335454	100.0%	57,67,182,191
1	4046815451585	100.0%	3281998335454	100.0%	72,73,74,75

Eight cores per node: USER Time per node					
Rank Order	Max USER Time	Max/ SMP	Avg USER Time	Avg/ SMP	Max Node Ranks
d	6657050297152	82.8%	6563996670908	100.0%	130,195,65,214,136,178,190,0
0	7315118136737	91.0%	6563996670908	100.0%	18,50,81,112,143,174,205,236
2	7499444177191	93.3%	6563996670908	100.0%	30,32,93,94,155,156,217,218
1	8036827543002	100.0%	6563996670908	100.0%	72,73,74,75,76,77,78,79

Example: File MPICH_RANK_ORDER.d (hycom)



```
# The custom rank placement in this file is the one labeled 'd'
# in the report from:
#
# pat_report -O mpi_rank_order \
# /home/users/ldr/ldr/COE_Workshop/hycom/090921P+hycomBase.ap2
#
# It targets multi-core processors, based on Time in USER group
# collected for:
#
# Program:      hycom.2009Sep10.x
# Number PEs:  249
# Cores/Node:  4
#
# To use this file, copy it to MPICH_RANK_ORDER and set the
# environment variable MPICH_RANK_REORDER_METHOD to 3 prior
# to executing the program.
#
46,126,32,176,109,224,48,243,39,142,154,220,137,21,174,140
36,151,155,242,15,219,133,177,110,108,134,100,25,118,132,148
130,195,65,214,136,178,190,0,61,141,54,167,162,161,189,122
111,183,35,3,163,129,50,199,93,84,88,29,246,26,153,168
38,196,182,210,156,30,51,5,81,231,64,159,24,179,49,239
104,95,175,143,206,120,152,215,80,217,135,4,13,31,34,228
37,169,66,211,157,83,172,203,92,226,53,234,201,86,188,7
180,238,52,139,138,229,33,193,45,194,94,166,207,205,173,185
63,204,18,11,14,164,97,241,131,123,99,22,102,101,98,149
40,107,223,85,103,202,76,212,82,247,67,70,124,208,79,144
105,235,89,160,119,87,78,192,181,10,55,127,20,221,77,225
191,9,44,209,23,106,96,233,60,187,17,146,170,244,57,145
112,150,43,117,236,216,56,6,47,8,19,121,116,12,58,198
16,27,42,186,158,230,218,248,41,1,245,165,62,200,59,213
114,2,91,28,128,125,75,232,222,147,90,71,171,69,72,240
113,227,115,197,237,68,74,184,73
```



Documentation for the Cray Performance Toolset

Luiz DeRose
Programming Environments Director
Cray Inc.

Accessing Software Versions

- Software package information
 - Use **avail**, **list** or **help** parameters to module command
 - With 5.0 release and later, '**module help xt-craypat**' shows release notes
- craypat version (same for pat_build, pat_report, pat_help)

% pat_build -V

CrayPat/X: Version 5.0 Revision 2786 08/31/09 12:18:23

- Cray Apprentice² version
 - Displayed in top menu bar when running GUI

Release Notes

```
(ldr@hawk) 112% module help xt-craypat
----- Module Specific Help for 'xt-craypat/5.0.2.2' -----
=====
CrayPat 5.0.1
=====
Release Date: November 19, 2009

Differences:
-----
Changes from CrayPat 5.0.0 release to 5.0.1 release
=====

CrayPat 5.0.0 release revision: 2786
CrayPat 5.0.1 release revision: 2901
-----

Purpose
-----
The following have been fixed/implemented
o PGI "-Mprof=func" compiler instrumentation is no longer supported by pat_build

o pat_build's -g math group has redefined all eligible entry points
  to more closely conform to the ISO/IEC standard

o the default for hardware performance counter multiplexing is now OFF (set
  PAT_RT_HWPC_MPX to non-zero to activate multiplexing)

o increase the number of supported CPU cores to 32

Bugs closed since 5.0.0 release
-----
. . .
Known Problem(s)
-----
. . .
Product and OS Dependencies:
-----
. . .
```

Online information



- User guide
 - <http://docs.cray.com>
 - Click on “Latest Docs” and choose “Performance Tools 5.0”
- Man pages
- To see list of reports that can be generated

```
% pat_report -O -h
```
- Notes sections in text performance reports provide information and suggest further options

Online Information (2)

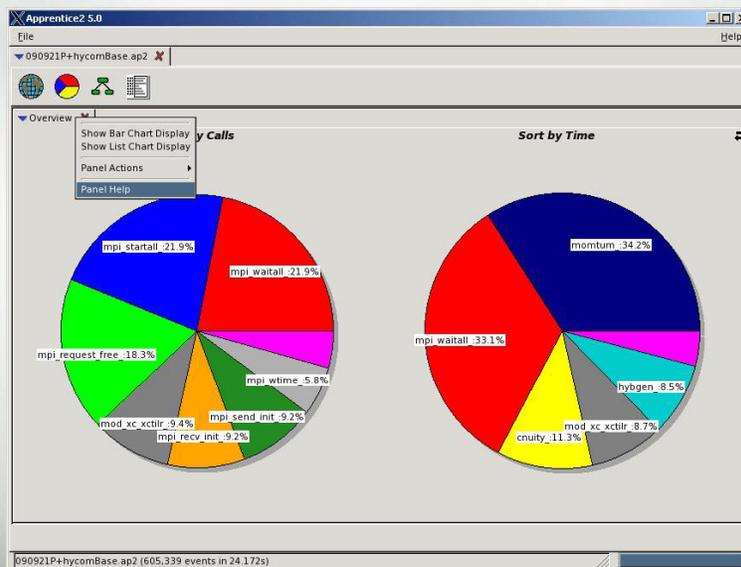


- Cray Apprentice² panel help
- `pat_help` – interactive help on the Cray Performance toolset
- FAQ available through `pat_help`

Man pages

- **intro_craypat(1)**
 - Introduces the craypat performance tool
- **pat_build**
 - Instrument a program for performance analysis
- **pat_help**
 - Interactive online help utility
- **pat_report**
 - Generate performance report in both text and for use with GUI
- **hwpc(3)**
 - describes predefined hardware performance counter groups
- **papi_counters(5)**
 - Lists PAPI event counters
 - Use papi_avail or papi_native_avail utilities to get list of events when running on a specific architecture

Cray Apprentice² Panel Help



Top of Default Report from APA Sampling

CrayPat/X: Version 5.0 Revision 2631 (xf 2571) 05/29/09 14:54:00

Number of PEs (MPI ranks): 48
 Number of Threads per PE: 1
 Number of Cores per Processor: 4

Execution start time: Fri May 29 15:31:49 2009
 System type and speed: x86_64 2200 MHz
 Current path to data file:
 /lus/nid00008/homer/sweep3d/sweep3d.mpi+samp.rts.ap2 (RTS)

Notes:

Sampling interval was 10000 microseconds (100.0/sec)
 BSD timer type was ITIMER_PROF

Trace option suggestions have been generated into a separate file from the data in the next table. You can examine the file, edit it if desired, and use it to reinstrument the program like this:

```
pat_build -O sweep3d.mpi+samp.rts.apa
```

pat_help

- Interactive by default, or use trailing '.' to just print a topic:
- New FAQ craypat 5.0.0.
- Has counter and counter group information

% pat_help counters amd_fam10h groups

pat_help Example

The top level CrayPat/X help topics are listed below.
A good place to start is:

overview

If a topic has subtopics, they are displayed under the heading "Additional topics", as below. To view a subtopic, you need only enter as many initial letters as required to distinguish it from other items in the list. To see a table of contents including subtopics of those subtopics, etc., enter:

toc

To produce the full text corresponding to the table of contents, specify "all", but preferably in a non-interactive invocation:

```
pat_help all . > all_pat_help
pat_help report all . > all_report_help
```

Additional topics:

API	execute
balance	experiment
build	first_example
counters	overview
demos	report
environment	run

```
pat_help (.=quit ,=back ^=up /=top ~=search)
=>
```

pat_help: FAQ

```
pat_help (.=quit ,=back ^=up /=top ~=search)
```

```
=> FAQ
Additional topics that may follow "FAQ":
```

Application Runtime	Miscellaneous
Availability and Module Environment	Processing Data with pat_report
Building Applications	Visualizing Data with Apprentice2
Instrumenting with pat_build	

```
pat_help FAQ (.=quit ,=back ^=up /=top ~=search)
```

```
=> I
Additional topics that may follow "Instrumenting with pat_build":
```

1. Can not access the file ...
2. ERROR: Missing required ELF section 'link information' from the program 'FILE'.
3. ERROR: Missing required ELF section 'string table' from the program '...'.
4. FATAL: The link information was not found in the .note section of ...
5. How can I find out the text size of functions?
6. How can I list trace points from my instrumented binary?
7. How can I lower the size of data files with pat_build?
8. How can I NOT instrument some of my object file(s)?
9. How do I get MPI rank order suggestions?
10. How do I specify a directory containing object files?
11. My error message is "xyz can not be traced because ... not writable"
12. Problems with instrumented programs using both MPI and OpenMP?
13. User sampling with compiler hooks present is not allowed
14. WARNING: Entry point 'FUNCTION' can not be traced because it is a locally defined function
15. WARNING: The function 'FUNCTION' can not be traced because a trace wrapper was not successfully created
16. What is APA?
17. Why am I getting an error with userTraceFunctions.c?
18. Why does my binary take longer to run when using 'pat_build -u'?

```
pat_help FAQ "Instrumenting with pat_build"
(.=quit ,=back ^=up /=top ~=search) =>
```

FAQ Example

```
pat_help FAQ "Instrumenting with pat_build"  
(.=quit ,=back ^=up /=top ~=search) => 4
```

```
FATAL: The link information was not found in the .note section of ...
```

If an executable is compiled and linked without the xt-craypat module loaded, then it will not contain link information needed by pat_build, which will issue an error message and exit.

To verify that an executable was built with the link information that pat_build requires, use

```
readelf -S $executable
```

It should show a .note section with a size of several kilobytes, say section 19, and the output from

```
readelf -x 19 $executable
```

should contain the string 'Cray Inc' and library paths.

```
pat_help FAQ "Instrumenting with pat_build"  
(.=quit ,=back ^=up /=top ~=search) =>
```

Short Quiz

- 1. Using pat_help, find the definition for computational intensity**
- 2. Using pat_help, find which hardware counters events are used in PAT_RT_HWPC 3**
- 3. Using pat_help, find how do you get MPI rank order suggestions**

Hands-on Tasks



1. Generate an “.apa” file and a sampling report from your application
2. Read the “.apa” file and add I/O instrumentation
3. Use the .apa file to generate a profile of the application
4. Look at the sampling report and identify areas where work is concentrated. Using the CrayPat API add instrumentation around the important loop(s)
5. Generate a second profile of the application with code regions
6. Obtain MFLOPS, TLB Utilization, Cache Hit/Miss ratios (L1 and L2), Cache utilization (L1, and L2), FP Mix, and Vectorization information for the main regions and functions of the application
7. Visualize the performance file (.ap2) with Cray Apprentice2 and identify the most imbalanced function or region of the application
8. Generate a trace file of the application (if the application is large, limit the size of the trace file)
9. Visualize the trace file with Cray Apprentice2
10. Optimize your application with the data that you collected



Performance Measurement and Visualization on the Cray XT

Questions / Comments
Thank You!