



# National Energy Research Scientific Computing Center (NERSC)

## The Landscape of Computer Architecture

John Shalf  
NERSC Center Division, LBNL



ISC2007, Dresden  
June 25, 2007



# The Landscape of Parallel Computer Architecture

Session chair not satisfied with title



# The NEW Landscape of Parallel Computer Architecture

Session chair *still* not satisfied with title



# Overturning the Conventional Wisdom for the Multicore Era: Everything You Know is **WRONG!**

Andreas (session chair) happy now...



# Traditional Sources of Performance Improvement are Flat-Lining

- **New Constraints**
  - 15 years of *exponential* clock rate growth has ended
- **But Moore's Law continues!**
  - How do we use all of those transistors to keep performance increasing at historical rates?
  - Industry Response: #cores per chip doubles every 18 months *instead* of clock frequency!

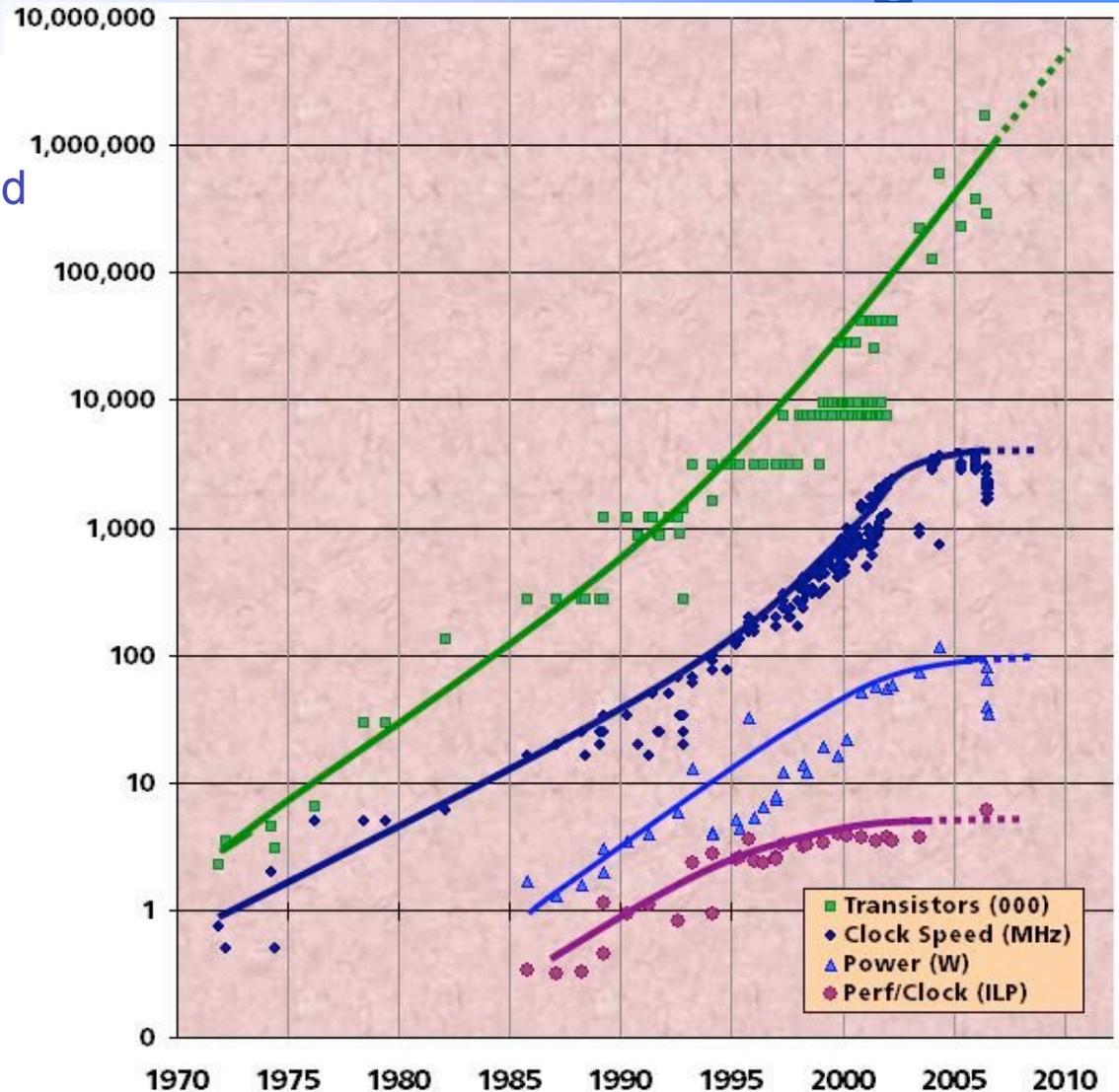


Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith



## Is Multicore the Correct Response to New Lithography Constraints?

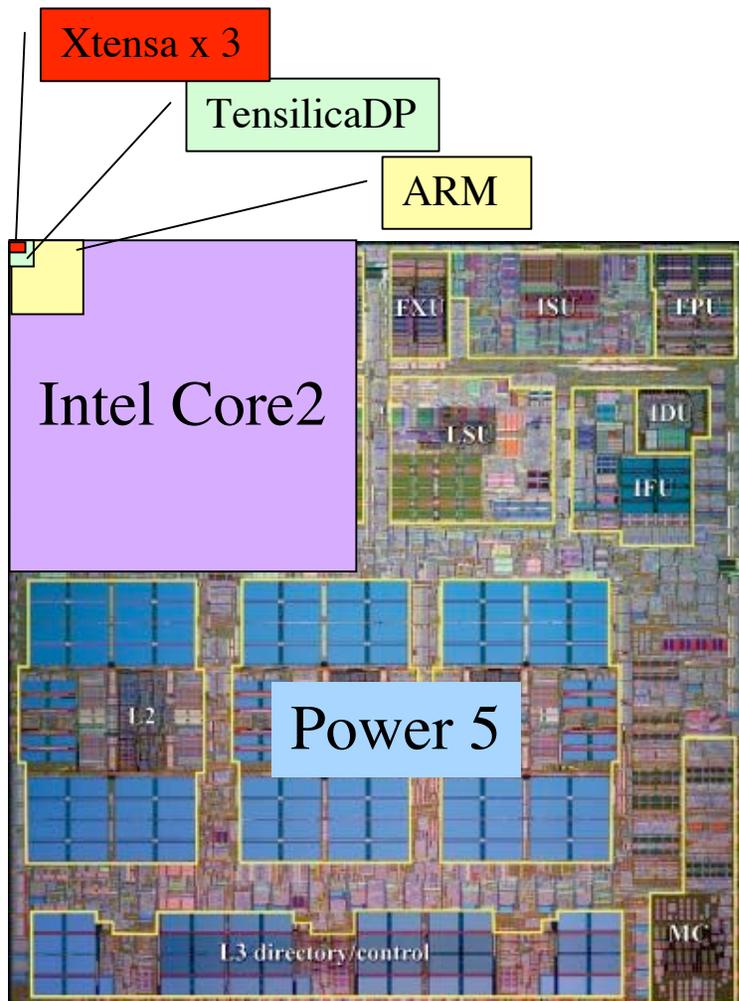
- Kurt Keutzer: “This shift toward increasing parallelism is not a triumphant stride forward based on breakthroughs in novel software and architectures for parallelism; instead, this plunge into parallelism is actually a retreat from even greater challenges that thwart efficient silicon implementation of traditional uniprocessor architectures.”
- David Patterson: “Industry has already thrown the hail-mary pass. . . But nobody is running yet.”

# Hardware: What are the problems?

- **Current Hardware/Lithography Constraints**
  - Power limits leading edge chip designs
    - Intel Tejas Pentium 4 cancelled due to power issues
  - Yield on leading edge processes dropping dramatically
    - IBM quotes yields of 10 – 20% on 8-processor Cell
  - Design/validation leading edge chip is becoming unmanageable
    - Verification teams > design teams on leading edge processors
- **Solution: Small Is Beautiful**
  - Expect modestly pipelined (5- to 9-stage) CPUs, FPUs, vector, SIMD PEs
    - Small cores not much slower than large cores
  - Parallel is energy efficient path to performance:  $CV^2F$ 
    - Lower threshold and supply voltages lowers energy per op
  - Redundant processors can improve chip yield
    - Cisco Metro 188 CPUs + 4 spares; Sun Niagara sells 6 or 8 CPUs
  - Small, regular processing elements easier to verify



# How Small is “Small”



- **Power5 (Server)**
  - 389mm<sup>2</sup>
  - 120W@1900MHz
- **Intel Core2 sc (Laptop)**
  - 130mm<sup>2</sup>
  - 15W@1000MHz
- **ARM Cortex A8 (BMW Auto)**
  - 5mm<sup>2</sup>
  - 0.8W@800MHz
- **Tensilica DP (cell phones)**
  - 0.8mm<sup>2</sup>
  - 0.09W@600MHz
- **Tensilica Xtensa (Cisco Router)**
  - 0.32mm<sup>2</sup> for 3!
  - 0.05W@600MHz

## Consider the comparison

	Traditional Core	Throughput Core
uArch	Out of Order	In Order
Size	50	10
Power	37.5	6.25
Freq	4	4
Threads	2	4
Single Thread	1	0.3
Vector	4 (128-bit)	16 (512-bit)
Peak Throughput	32	128
Area Capacity	0.6	13
Power Capacity	0.9	20



*Potential for 20x power efficiency in*



# Power Efficiency Motivates Manycore

*(why should HPC “users” care?)*

- **Power Efficiency concerns drive industry to Manycore**
- **Power is leading factor limiting future system growth**
  - Cost of power > cost of hardware
  - \$33M/year projected power+cooling costs at ORNL 2010
  - 130MW projected power for Exascale based on today's technology (>\$130M/year for cheap power!)
  - Increasing fraction of budget will go to power (which means less capability for YOU)
- **Misplaced Concerns**
  - **Old CW: Computational Efficiency is “sustained-to-peak” (KmPH/HP)**
  - **New CW: Computational Efficiency is performance/watt (MPG)**
  - *Optimizing performance-per-watt necessarily includes consideration of programmability!*
  - *This means hardware architects MUST understand application requirements to move forward with next-generation architectures! (a considerable departure from status quo)*

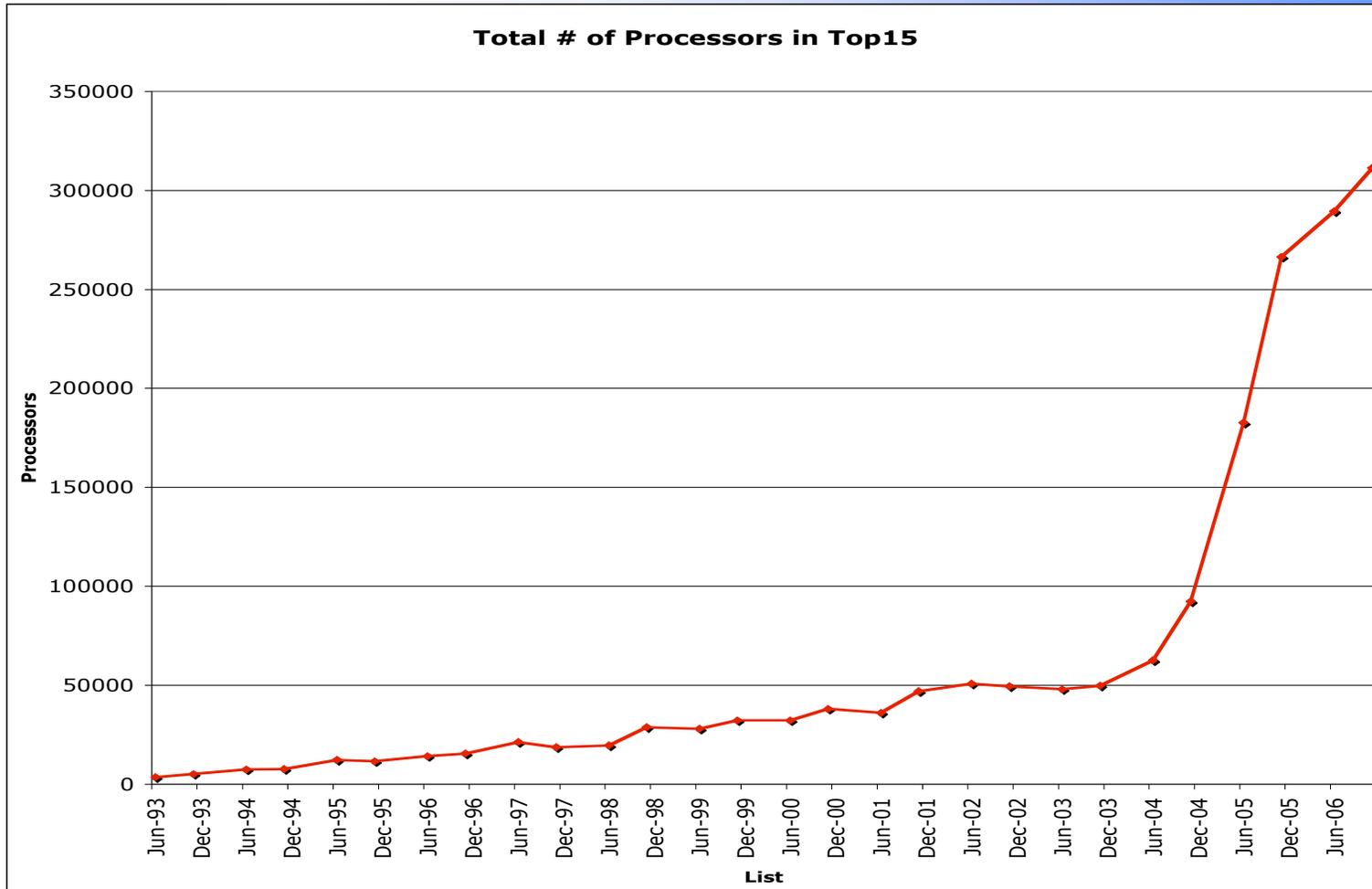


# Multicore vs. Manycore

- **Multicore: current trajectory**
  - Stay with current fastest core design
  - Replicate every 18 months (2, 4, 8 . . . Etc...)
  - Advantage: Do not alienate serial workload
  - Example: AMD X2 (2 core), Intel Core2 Duo (2 cores), Madison (2 cores), AMD Barcelona (4 cores)
- **Manycore: converging in this direction**
  - Simplify cores (shorter pipelines, lower clock frequencies, in-order processing)
  - Start at 100s of cores and replicate every 18 months
  - Advantage: highest compute/surface-area, best power efficiency, easier verification, defect tolerance, lower design costs.
  - Examples: Cell SPE (8 cores), Nvidia G80 (128 cores), Intel Polaris (80 cores), Cisco/Tensilica Metro (188 cores)
- **Convergence: Ultimately toward Manycore**
  - Manycore *if we can figure out how to program it!*
  - Hedge: Heterogenous Multicore



# The Future of HPC System Concurrency



**Must ride exponential wave of increasing concurrency for foreseeable future!**

**You will hit 1M cores sooner than you think!**



# The *Entire* Computing Industry is Betting Its future on Parallelism

- **Old CW:** Innovation trickles down from High End Computing to your PC and consumer-electronics
- **New CW:** The flow is reversing! Innovation is trickling UP from consumer applications to HPC! (*petaflop cell phone?*)
  - They know more about computational/power efficiency than we do!
  - They know more about cost-effectiveness than we do!
- **This transition is NOT just about HPC!**
  - You are NOT in the driver's seat of this revolution (get over it!)
  - Your Motorola Razor Cell Phone already has 8 CPU cores in it (and will grow geometrically from there)
  - Cisco CRS-1 router has 188 tensilica CPU cores/socket (Metro) and scales to 400,000 cores! (more than current HPC... runs an OS too!)
  - Your *toaster oven* is going be running parallel applications on manycore processors
- **Industry has already moved forward with parallelism without having a software solution in place (or even agreed upon)**
  - They are as scared as we are!



# Failure Modes for the Multicore Revolution

- **System Balance:** Concern that memory and interconnect performance will ultimately cap multicore performance
- **Reliability:** More “moving parts” means more opportunity for failures
- **Programmability:** How can I possibly program 1M+ cores in an effective manner?



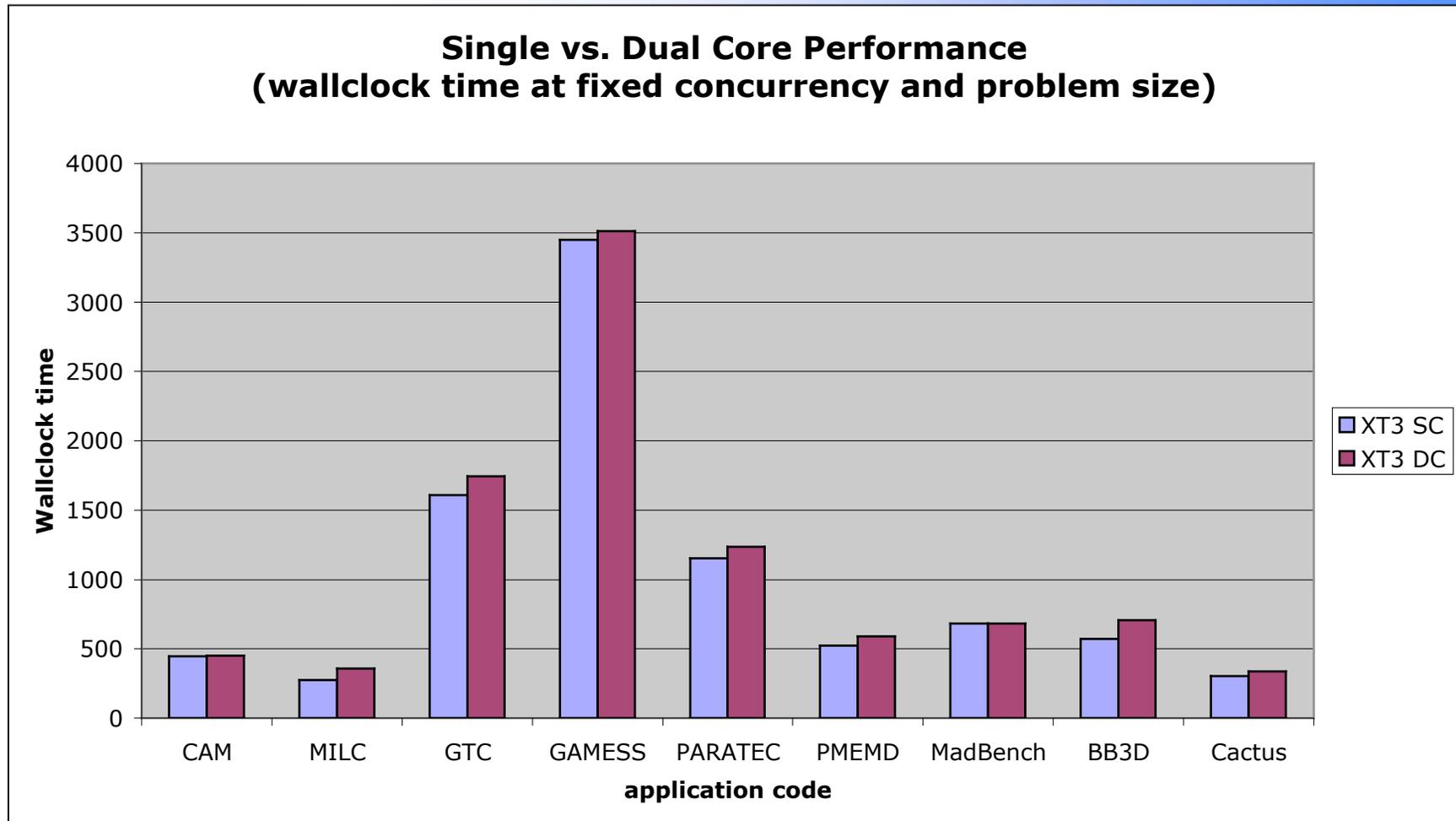
# System Balance

*Memory Bandwidth is the Primary Limiting Factor for Future Multicore Performance*



# System Balance

*(its all about the memory bandwidth)*

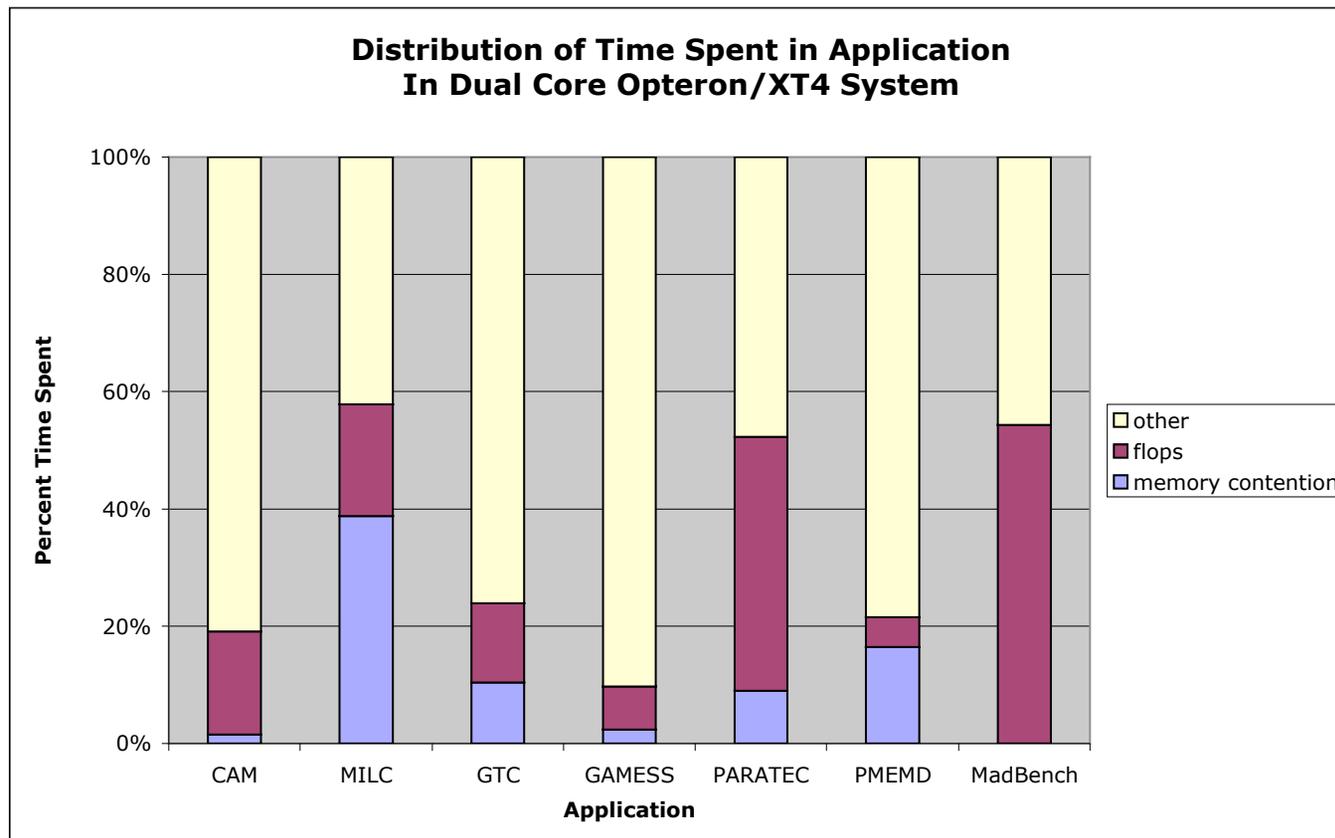


**Halving the memory bandwidth has virtually no effect on performance for a broad variety of applications!!!**



# System Balance

*(its all about memory bandwidth?)*



- Even infinite improvement in FLOP rate or memory bandwidth nets us 15% average performance benefit
- But biggest benefit from improving “other” which is dominated by latency stalls!
- not application limited... it is limited by the CPU architecture
- multicore is not making this situation WORSE!



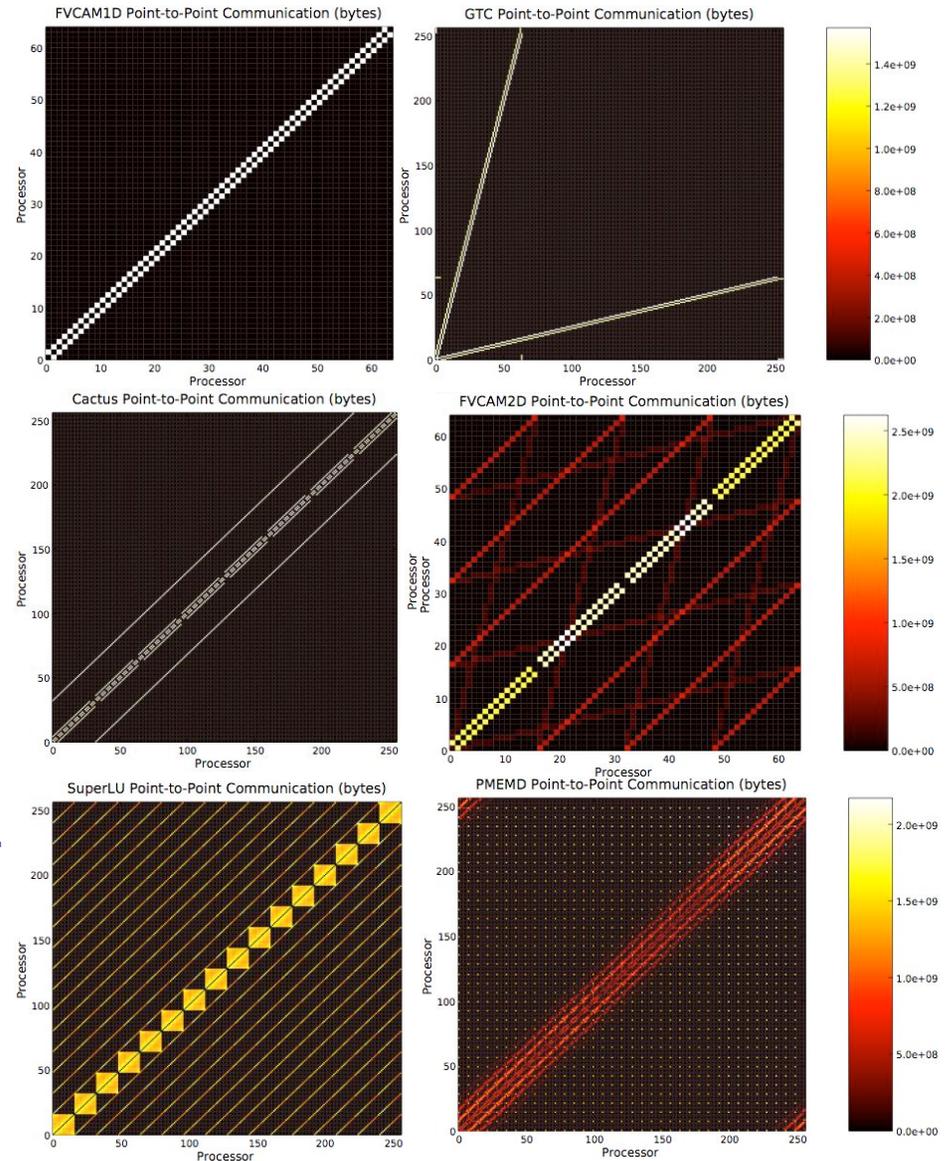
## System Balance

***Bisection Bandwidth is the primary limiting factor for future performance of MPPs (therefore, must use Crossbar or CLOS)***



# Interconnect Design Considerations for Massive Concurrency

- **Application studies provide insight to requirements for Interconnects (both on-chip and off-chip)**
  - On-chip interconnect is 2D planar (crossbar won't scale!)
  - Sparse connectivity for dwarfs; crossbar is overkill
  - No single best topology
- **A Bandwidth-oriented network for data**
  - Most point-to-point message exhibit sparse topology & bandwidth bound
- **Separate Latency-oriented network for collectives**
  - E.g., Thinking Machines CM-5, Cray T3D, IBM BlueGene/L&P
- **Ultimately, need to be aware of the on-chip interconnect topology in addition to the off-chip topology**
  - Adaptive topology interconnects (HFAST)
  - Intelligent task migration?

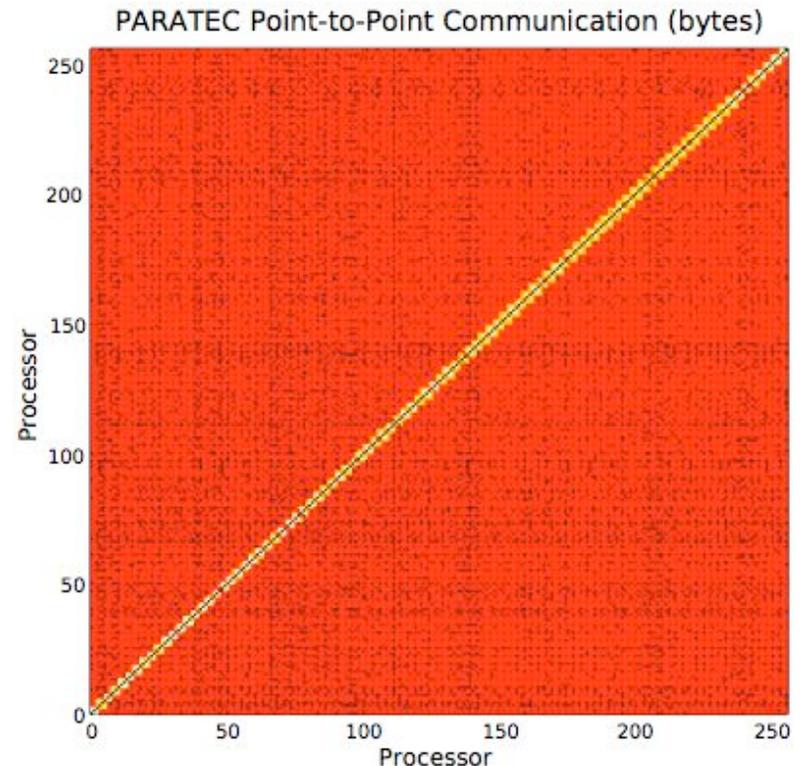




# Interconnects

## Need For High Bisection Bandwidth

- **3D FFT easy-to-identify as needing high bisection**
  - Each processor must send messages to all PE's! (all-to-all) for 1D decomposition
  - However, most implementations are currently limited by overhead of sending small messages!
  - 2D domain decomposition (required for high concurrency) actually requires  $\sqrt{N}$  communicating partners! (*some-to-some*)
- **Same Deal for AMR**
  - AMR communication is sparse, but by no means is it bisection bandwidth limited





# Reliability

- Do more cores mean less reliability?



# Reliable System Design

- **The future is unreliable**
  - Silicon Lithography pushes towards the atomic scale, the opportunity for spurious hardware errors will increase dramatically
- **Reliability of a system is not necessarily proportional to the number of cores in the system**
  - Reliability is proportional to # of sockets in system (not #cores/chip)
  - At LLNL, BG/L has longer MTBF than Purple despite having 12x more processor cores
  - Integrating more peripheral devices onto a single chip (e.g. caches, memory controller, interconnect) can further reduce chip count and increase reliability (System-on-Chip/SOC)
- **A key limiting factor is software infrastructure**
  - Software was designed assuming perfect data integrity (but that is not a multicore issue)
  - Software written with implicit assumption of smaller concurrency (1M cores not part of original design assumptions)
  - Requires fundamental re-thinking of OS and math library design assumptions



# Operating Systems for CMP

- **Even Cell Phones will need OS** (*and our idea of an OS is tooooo BIG!*)
  - Mediating resources for many cores, protection from viruses, and managing increasing code complexity
  - But it has to be very small and modular! (see *also* embedded Linux)
- **Old OS Assumptions are bogus for hundreds of cores!**
  - Assumes limited number of CPUs that must be shared
    - *Old OS: time-multiplexing (context switching and cache pollution!)*
    - *New OS: spatial partitioning*
  - Greedy allocation of finite I/O device interfaces (eg. 100 cores go after the network interface simultaneously)
    - *Old OS: First process to acquire lock gets device (resource/lock contention! Nondeterm delay!)*
    - *New OS: QoS management for symmetric device access*
  - Background task handling via threads and signals
    - *Old OS: Interrupts and threads (time-multiplexing) (inefficient!)*
    - *New OS: side-cores dedicated to DMA and async I/O*
  - Fault Isolation
    - *Old OS: CPU failure --> Kernel Panic (will happen with increasing frequency in future silicon!)*
    - *New OS: CPU failure --> Partition Restart (partitioned device drivers)*
  - Old OS invoked any interprocessor communication or scheduling vs. direct HW access
- **What will the new OS look like?**
  - Whatever it is, it will probably look like Linux (*or ISV's will make life painful*)
  - Linux too big, but microkernel not sufficiently robust
  - Modular kernels commonly used in embedded Linux applications! (e.g. vxWorks running under a hypervisor XEN, K42, D.K. Panda Side Cores)



# I/O For Massive Concurrency

- **Scalable I/O for massively concurrent systems!**
  - Many issues with coordinating access to disk within node (on chip or CMP)
  - OS will need to devote more attention to QoS for cores competing for finite resource (*mutex locks and greedy resource allocation policies will not do!*) (*its like rugby where device == the ball*)

nTasks	I/O Rate	I/O Rate
	16 Tasks/node	8 tasks per node
8	-	131 Mbytes/sec
16	7 Mbytes/sec	139 Mbytes/sec
32	11 Mbytes/sec	217 Mbytes/sec
64	11 Mbytes/sec	318 Mbytes/sec
128	25 Mbytes/sec	471 Mbytes/sec



## Programming Model

*High Productivity Computing is  
the Uber Design Goal for Parallel  
Computing Languages!*



# Programmability

- **Widespread panic over programming model that can ride the “Tsunami of concurrency”**
- **Inter-dependent requirements for programming environment**
  - Productivity
  - Performance
  - Correctness
- **Approaches**
  - Abstracting single-chip parallelism
    - Focus of the Broader Consumer Electronics/Computing Industry
    - Even in HPC, observe that # chips growing much slower than # cores
  - Hiding complexity of global parallelism
    - Frameworks, Advanced compilers and programming languages, Auto-tuning
  - Nightmare Scenario: Microsoft solves in-socket programming model and we are stuck writing MPI between sockets that run C# code!
- **Competing Goals**
  - Productivity Layer: *Simplify specification of program/problem to solve*
  - Performance Layer: *Expose all hardware Capabilities to programmer*



# Multicore is NOT a Familiar Programming Target

- **What about Message Passing on a chip?**
  - MPI buffers & datastructures growing  $O(N)$  or  $O(N^2)$  a problem for constrained memory
  - Redundant use of memory for shared variables and program image
  - *Flat view of parallelism doesn't make sense given hierarchical nature of multicore sys.*
- **What about SMP on a chip?**
  - Hybrid Model (MPI+OpenMP) : *Long and mostly unsuccessful history*
  - But it is NOT an SMP on a chip
    - 10-100x higher bandwidth on chip
    - 10-100x lower latency on chip
  - SMP model ignores potential for much tighter coupling of cores
  - *Failure to exploit hierarchical machine architecture will drastically inhibit ability to efficiently exploit concurrency! (requires code structure changes)*
- **Looking beyond SMP**
  - Cache Coherency: *necessary but not sufficient (and not efficient for manycore!)*
  - Fine-grained language elements difficult to build on top of CC protocol
  - Hardware Support for Fine-grained hardware synchronization
  - Message Queues: direct hardware support for messages
  - Transactions: Protect against incorrect reasoning about concurrency



# Hardware Transactional Memory (TM) Will Make Parallelism Easy

*the silver bullet is already in our hands*

- **What are Transactions**
  - Speculatively execute, but don't commit result to memory (stays resident in cache for HW-assisted transactions)
  - If another thread wrote to the same memory addresses read by my thread, then DO NOT commit results and re-execute (Rollback)
  - If no dataflow conflict occurred, then commit results to memory
- **Why transactions are good**
  - Can assume parallelization of loop iterations where every iteration is a transaction (Lay-Z-boy parallelization!)
  - If you reason incorrectly about dataflow hazards (read-after-write), then suffer slower performance, but still get the correct answer (very GOOD property)
  - Auto-parallelizing compilers can be more aggressive
- **Why transactions are bad (*a subset of leading issues*)**
  - What does it mean to have a nested transaction?
  - What does it mean to mix transactional regions with non-transactional regions? (*Kozyrakis*)
  - How large can a transaction be? (*finite hardware resources*)



# Threads and locks are a messy parallel programming model





# Transactional Memory Helps a Bit





# Programming Model

*Some Advanced Compiler Technology Will  
Save Us!*



# Advanced Compiler Technology Will Save Us!

- **Old Conventional Wisdom: Depend on the Compiler to hide architectural details**
  - We underestimate the amount of information compiler optimizers and back-ends have to work with
  - Many ambiguities at compile-time that are only resolved at runtime
  - Have to be conservative to ensure “correctness”
  - Don’t hold your breath waiting for “autoparallelization” (it WON’T happen!)
- **New approach: “Auto-tuners” 1st run variations of program on computer to heuristically search for best combinations of optimizations (blocking, padding, ...) and data structures, then produce C code to be compiled for *that* computer**
  - E.g., PHiPAC (BLAS), Atlas (BLAS), Spiral (DSP), FFT-W
  - Can achieve 10X over conventional compiler
  - Encode body of knowledge regarding tuning strategies
- **Can even have compiler assist with generating test cases (see *also* Mary Hall)**



# Community Codes & Frameworks

*(hiding complexity using good SW engineering)*

- **Frameworks (eg. Chombo, Cactus, SIERRA, UPIC, etc...)**
  - Clearly separate roles and responsibilities of your expert programmers from that of the domain experts/scientist/users (productivity layer vs. performance layer)
  - Define a *social* contract between the expert programmers and the domain scientists
  - Enforces and facilitates SW engineering style/discipline to ensure correctness
  - Hides complex domain-specific parallel abstractions from scientist/users to enable performance (hence, most effective when applied to community codes)
  - Allow scientists/users to code nominally serial plug-ins that are invoked by a parallel “driver” (either as DAG or constraint-based scheduler) to enable productivity
- **Properties of the “plug-ins” for successful frameworks (CSE07)**
  - Relinquish control of main(): invoke user module when framework thinks it is best
  - Module must be stateless
  - Module only operates on the data it is handed (no side-effects)
- **Frameworks can be thought of as driver for coarse-grained dataflow**
  - Very much like classic static dataflow, except coarse-grained objects written in declarative language (dataflow without the functional languages)
  - Broad flexibility to schedule Directed Graph of dataflow constraints
  - See *also* Jack Dongarra & Parry Husbands’ work on DAG-based scheduling



# Multicore Opportunities

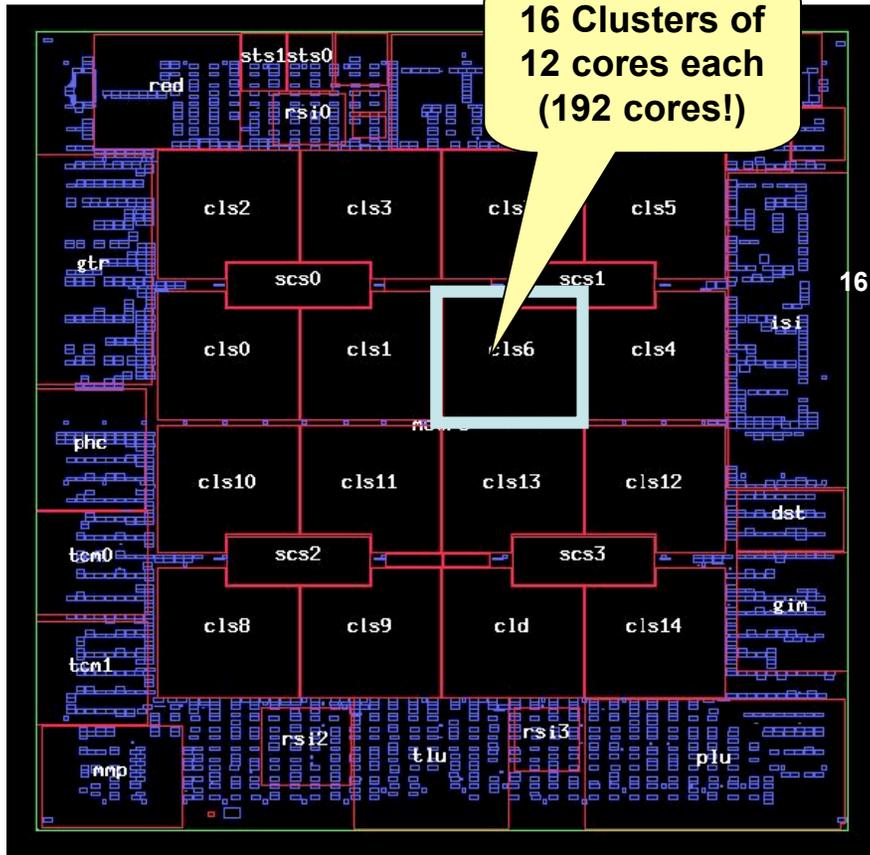
*(thinking about large numbers of cores)*

- **Operating Systems**
  - Spatially partition cores instead of time multiplexing
  - “Side-cores” for OS services and interrupts (D.K. Panda)
- **Offload engines for efficient one-sided communication**
- **Truly asynchronous/background I/O**
- **Load balancing calculation and data movement**
  - Load-imbalance is looming impediment to future scalability
  - Currently creates load-imbalance by attempting to compute balance
  - Run in tandem with computations (background balancing) on dedicated cores
- **Exploiting on-chip bandwidth (dataflow)**
  - Rather than decomposing for SPMD parallelism, decompose laterally (feed-forward pipelines) to reuse on-chip bandwidth
  - Good: More general than streaming. Better exploitation of on-chip bandwidth and data locality!
  - Bad: Requires strict control of side-effects
  - Would benefit greatly from rediscovering dataflow and functional programming languages
- **SoC: Perhaps we shouldn't use the area just for more cores!?!**
  - Integrated memory controllers etc... better reliability

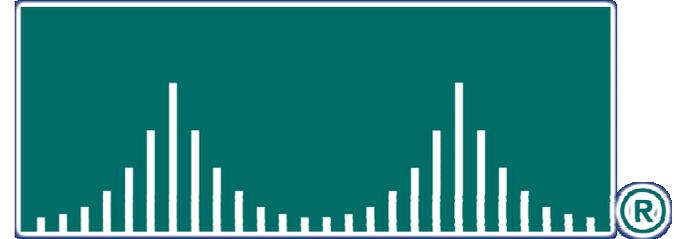


# Parallel Computing Everywhere

## Cisco CRS-1 Terabit Router



# CISCO SYSTEMS



- 188+4 Xtensa general purpose processor cores per Silicon Packet Processor
- Up to 400,000 processors per system
  - (this is not just about HPC!!!)



Replaces ASIC using 188 GP cores!

Emulates ASIC at nearly same power/performance

Better power/performance than FPGA!

New Definition for “Custom” in SoC



U.S. DEPARTMENT OF ENERGY



# Conclusions

- ***“The processor is the new transistor!” Chris Rowen***
- **Enormous transition is underway that affects all sectors of computing industry**
  - Motivated by power limits
  - Proceeding before emergence of the parallel programming model
- **Will lead to new era of architectural exploration given uncertainties about programming and execution model (*and we MUST explore!*)**
- **Need to get involved now**
  - 3-5 years for new hardware designs to emerge
  - 3-5 years lead for new software ideas necessary to support new hardware to emerge
  - 5+ MORE years to general adoption of new software



## More Info

- “The Landscape of Computing Architecture: A View from Berkeley”
  - <http://view.eecs.berkeley.edu>
- Discussion of Impact of Embedded Technology on HPC
  - [http://vis.lbl.gov/~jshalf/SIAM\\_CSE07](http://vis.lbl.gov/~jshalf/SIAM_CSE07)
- RAMP Architectural Simulator
  - <http://ramp.eecs.berkeley.edu/>



## Extra Material



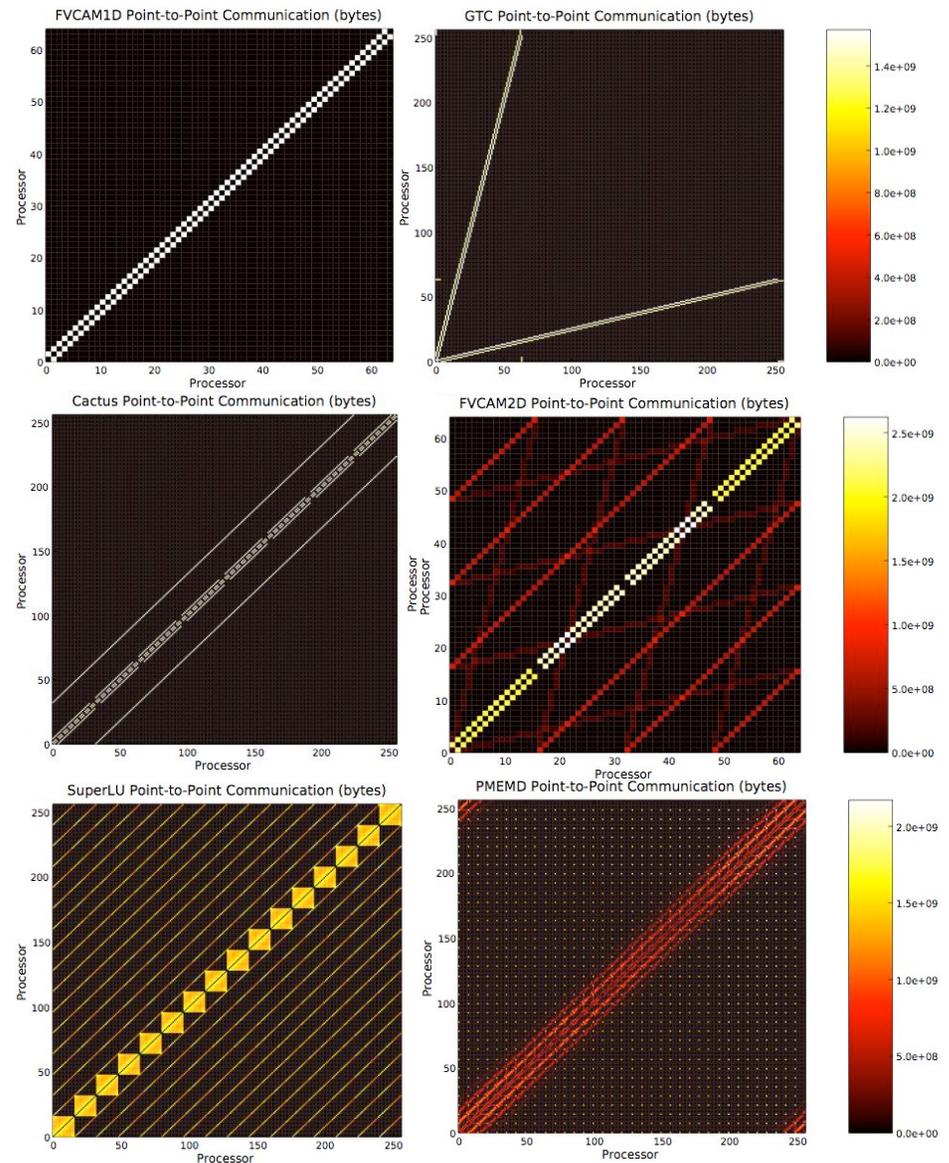
# Memory Bandwidth *is not the primary problem*

- **Old CW: Bandwidth is the primary problem**
  - Strict adherence to Amdahl byte/flop ratios!
  - Treat Latency and Bandwidth as independent issues!
- **New CW: Bandwidth and Latency are intimately related via Little's Law**
  - $\text{Concurrency} = \text{Bandwidth} * \text{Latency}$
  - Exacerbates problem if you throw memory bandwidth at it without addressing deficient latency tolerance in existing CPU cores!
  - Reduce problem if we have more cores to increase concurrent memory requests
  - Devote cores to hiding latency through DMA or other novel architectural features for latency hiding!
  - Please do not quote amdahl byte/flop ratios to me again!!
- **Is Memory Bandwidth the problem?**
  - We should be so lucky...



# Interconnect Design Considerations for Massive Concurrency

- **Bisection Bandwidth**
  - Old CW: scientific applications (especially AMR) require high bisection bandwidth (use a crossbar!)
  - New CW: No they don't
- **Topology**
  - Old CW: We can find an optimal topology to serve needs of applications
  - New CW: No you can't
- **Collectives**
  - Old CW: Carry collective operations over the same network used for point-to-point communication
  - New CW: Have specialized networks for point-to-point and collective/sync operations





# Frameworks

*(its not all about computer languages!)*

- **Frameworks (eg. Chombo, Cactus, SIERRA, UPIC, etc...)**
  - Clearly separate roles and responsibilities of your expert programmers from that of the domain experts/scientist/users (productivity layer vs. performance layer)
  - Define a social contract between the expert programmers and the domain scientists (importance constantly underestimated!)
  - Hides complex domain-specific parallel abstractions from scientist/users (hence, most effective when applied to community codes)
  - Allow scientists/users to code nominally serial plug-ins that are invoked by a parallel “driver” (either as DAG or constraint-based scheduler)
- **Properties of the “plug-ins” for successful frameworks (CSE07)**
  - Relinquish control of main(): invoke user module when framework thinks it is best
  - Module must be stateless
  - Module only operates on the data it is handed (no side-effects)
- **Frameworks can be thought of as driver for coarse-grained dataflow**
  - Very much like classic static dataflow, except coarse-grained objects written in declarative language (dataflow without the functional languages)
  - Broad flexibility to schedule Directed Graph of dataflow constraints
  - We are slowly rediscovering dataflow (but don’t realize it yet)

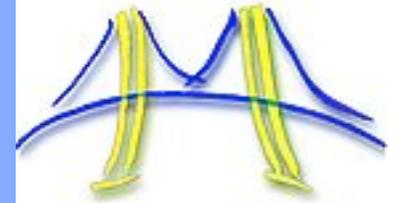


# Programmability

- **Widespread panic over programming model that can ride the “Tsunami of concurrency”**
- **Inter-dependent requirements for programming environment**
  - Productivity
  - Performance
  - Correctness
- **Approaches**
  - Abstracting single-chip parallelism
    - Industry focus
    - Observe # chips growing much slower than # cores
  - Hiding complexity of global parallelism
    - Frameworks, Advanced compilers and programming languages, Auto-tuning
  - Imagining infinite-parallelism



# 21<sup>st</sup> Century Code Generation

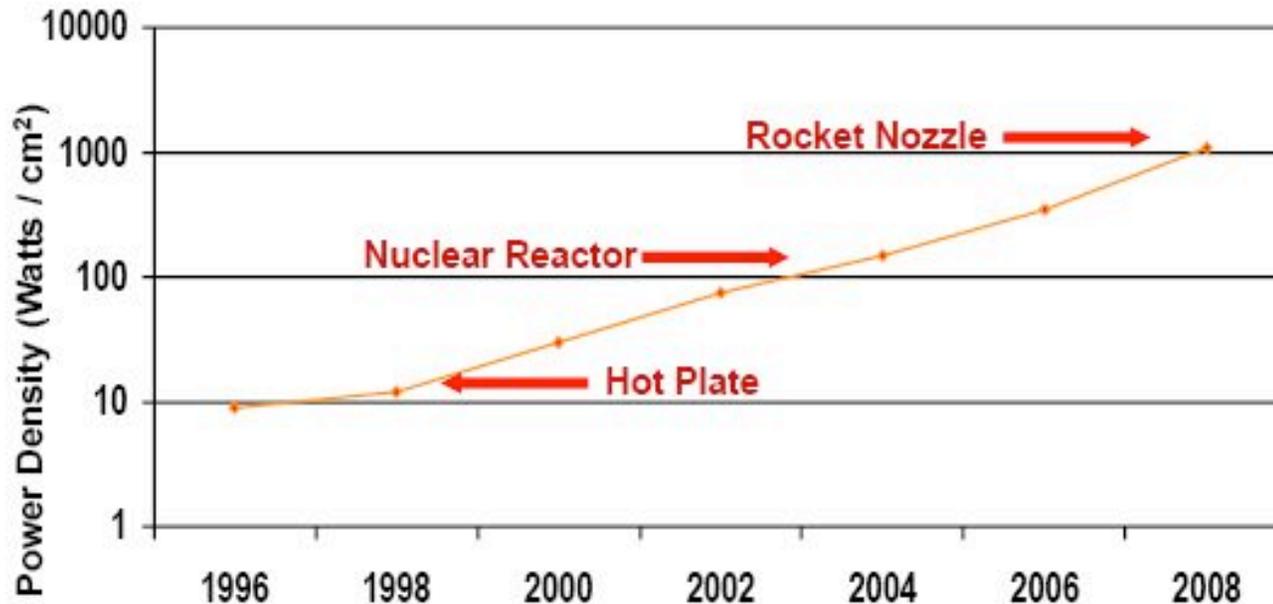


- New approach: “**Auto-tuners**” 1st run variations of program on computer to heuristically search for best combinations of optimizations (blocking, padding, ...) and data structures, then produce C code to be compiled for ***that*** computer
  - E.g., PHiPAC (BLAS), Atlas (BLAS), Spiral (DSP), FFT-W
  - Can achieve 10X over conventional compiler
- **Example: Sparse Matrix (SPMv) for 3 multicores**
  - Fastest SPMv: 2X OSKI/PETSc Clovertown, 4X Opteron
  - Optimization space: register blocking, cache blocking, TLB blocking, prefetching/DMA options, NUMA, BCOO v. BCSR data structures, 16b v. 32b indices, ...



# Power Now a Dominant Concern

## Moore's Law Extrapolation: Power Density for Leading Edge Microprocessors



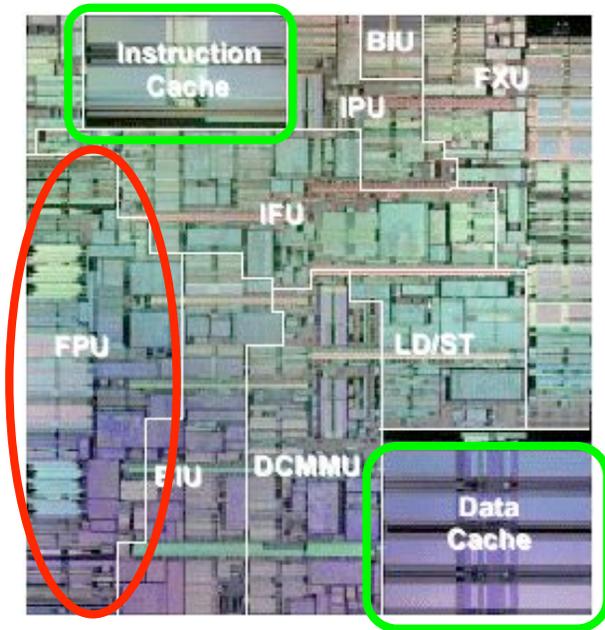
Power Density Becomes Too High to Cool Chips Inexpensively

Source: Shekhar Borkar, Intel Corp



# Chip Complexity and Design Costs and Verification also Concern

## Power3

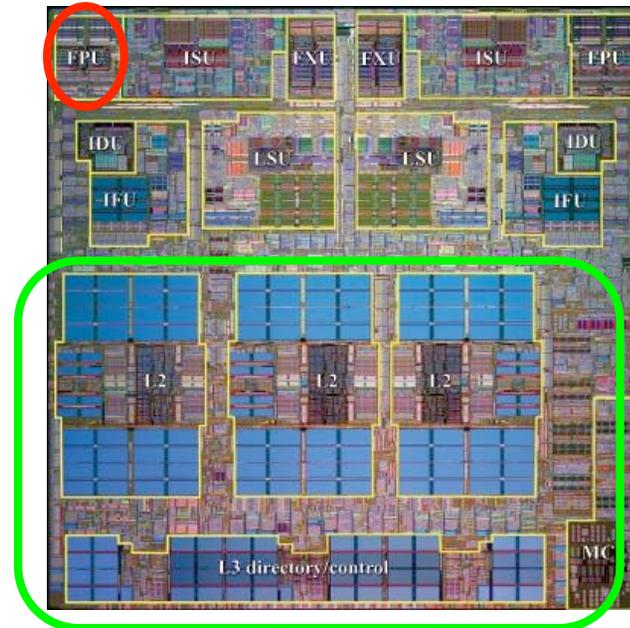


15 M Transistors

375 MHz

*Image From IBM Power3 Redbook*

## Power5



276 M Transistors **(18x more!)**

1900 MHz

*Image From IBM Power5 Redbook*

- Most chip area used to hide latency for high clock frequencies
- Cost of new high-end chip ~\$400M with 3 year lead time
- One defect and the entire core is useless
- Impossible to exhaustively verify 70M logic gates!

# Hardware: What are the problems?

- **Current Hardware/Lithography Constraints**
  - Power limits leading edge chip designs
    - Intel Tejas Pentium 4 cancelled due to power issues
  - Yield on leading edge processes dropping dramatically
    - IBM quotes yields of 10 – 20% on 8-processor Cell
  - Design/validation leading edge chip is becoming unmanageable
    - Verification teams > design teams on leading edge processors
- **Solution: Small Is Beautiful**
  - Expect modestly pipelined (5- to 9-stage) CPUs, FPUs, vector, SIMD PEs
    - Small cores not much slower than large cores
  - Parallel is energy efficient path to performance:  $CV^2F$ 
    - Lower threshold and supply voltages lowers energy per op
  - Redundant processors can improve chip yield
    - Cisco Metro 188 CPUs + 4 spares; Sun Niagara sells 6 or 8 CPUs
  - Small, regular processing elements easier to verify



## Basic Processor Efficiency

### The Usual List of Suspects

	IBM/ Sony/ Toshiba Cell	IBM BlueGene /L (PowerPC 440 ASIC)	AMD Opteron K8L	Intel Xeon 5100 Woodcrest	Intel Itanium2	Xtensa- based SIMD/LIW Scientific Engine
DP Operations per Cycle per Processor	0.6 per SPE	4	4	4	4	4
Cycles per second (GHz)	3.2	0.7	3.0	3.0	1.4	0.65
Processors per IC	8	2	2	2	1	32
Aggregate DP GLFOPS per IC	15	5.6	24	24	5.6	83
Approx IC Power (Watts)	30	10	80	65	130	12
IC GFLOPS/Watt	0.5	0.6	0.3	0.4	0.06	7

DP FP pipelines in FPGA: 15.9 GFLOPs @ 25W (Xilinx Virtex-4 LX200): 0.63 GFLOPS/W