Stanford Pervasive Parallelism Lab (PPL)

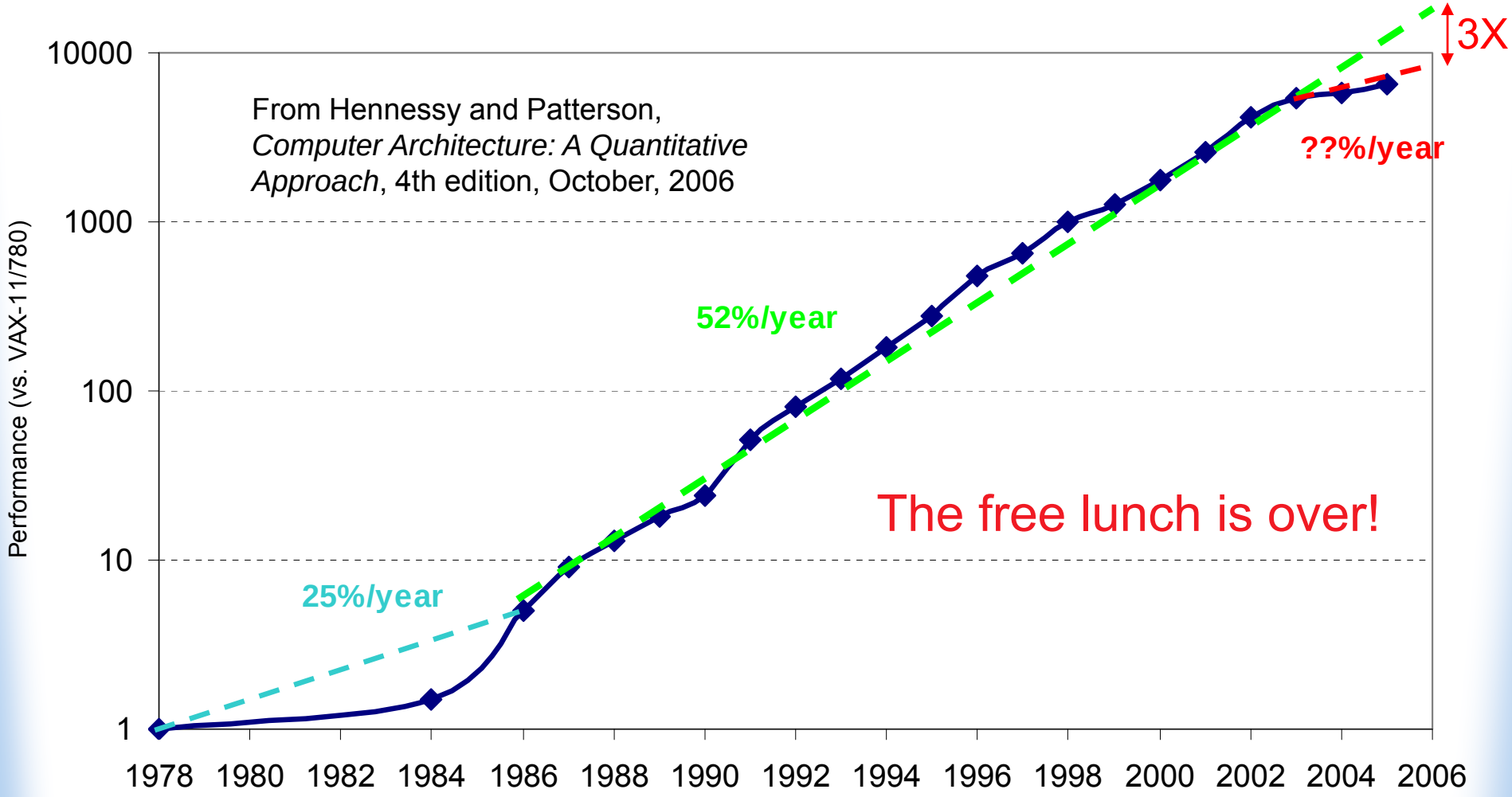
Alex Aiken, Bill Dally, Pat Hanrahan,
John Hennessy, Mark Horowitz,
Christos Kozyrakis, **Kunle Olukotun**,
Mendel Rosenblum

March 2007

Pervasive Parallelism Lab

- Goal: Parallel Programming environment for 2010
  - Parallelism for the masses: make parallel programming accessible to the average programmer
  - Parallel algorithms, development environments, and runtime systems that scale to 1000s of hardware threads
- PPL is a combination of
  - Leading Stanford researchers in applications, languages, systems software and computer architecture
  - Leading companies in computer systems, software and applications
  - Compelling vision for creating and managing pervasive parallelism

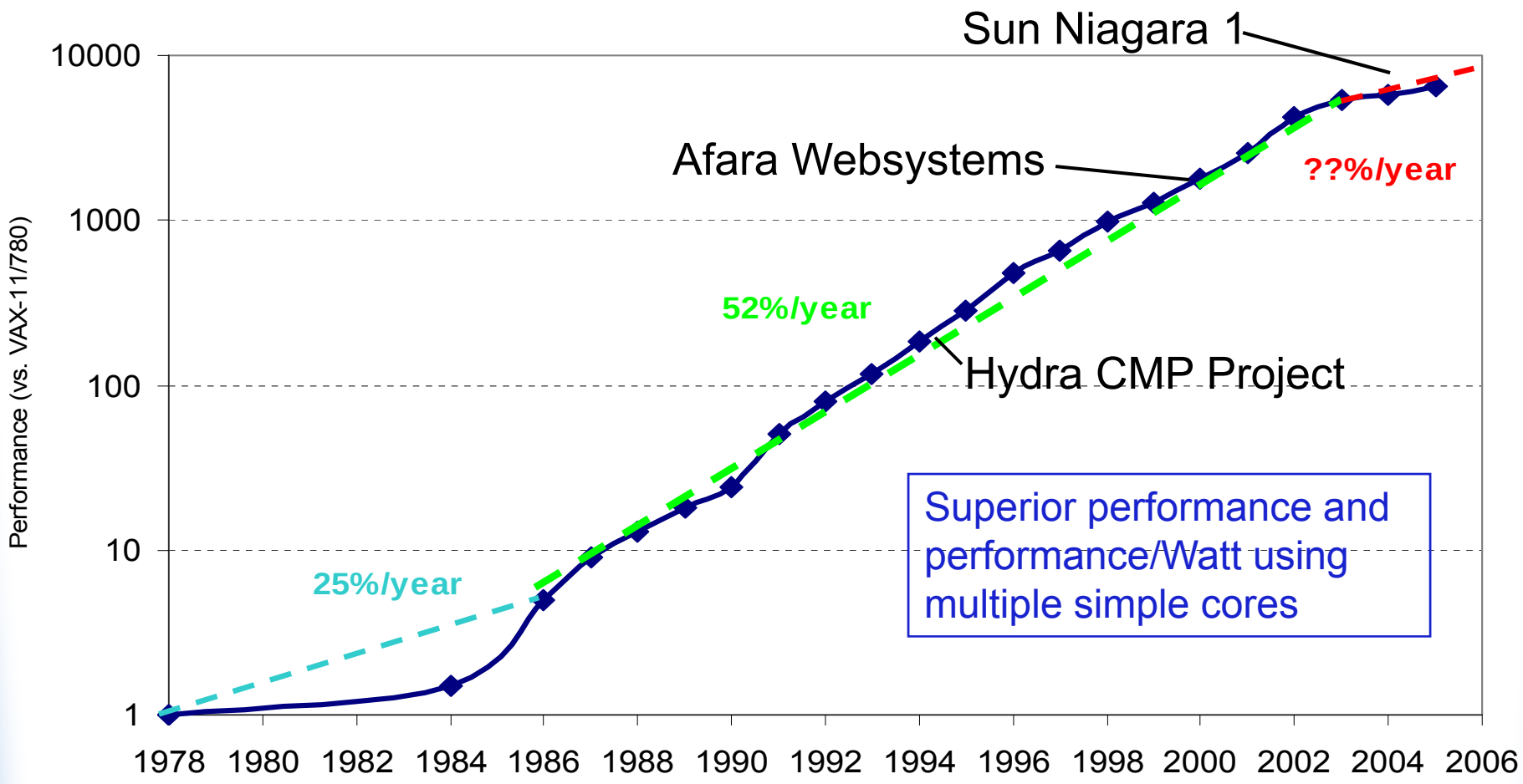
End of Uniprocessor Performance



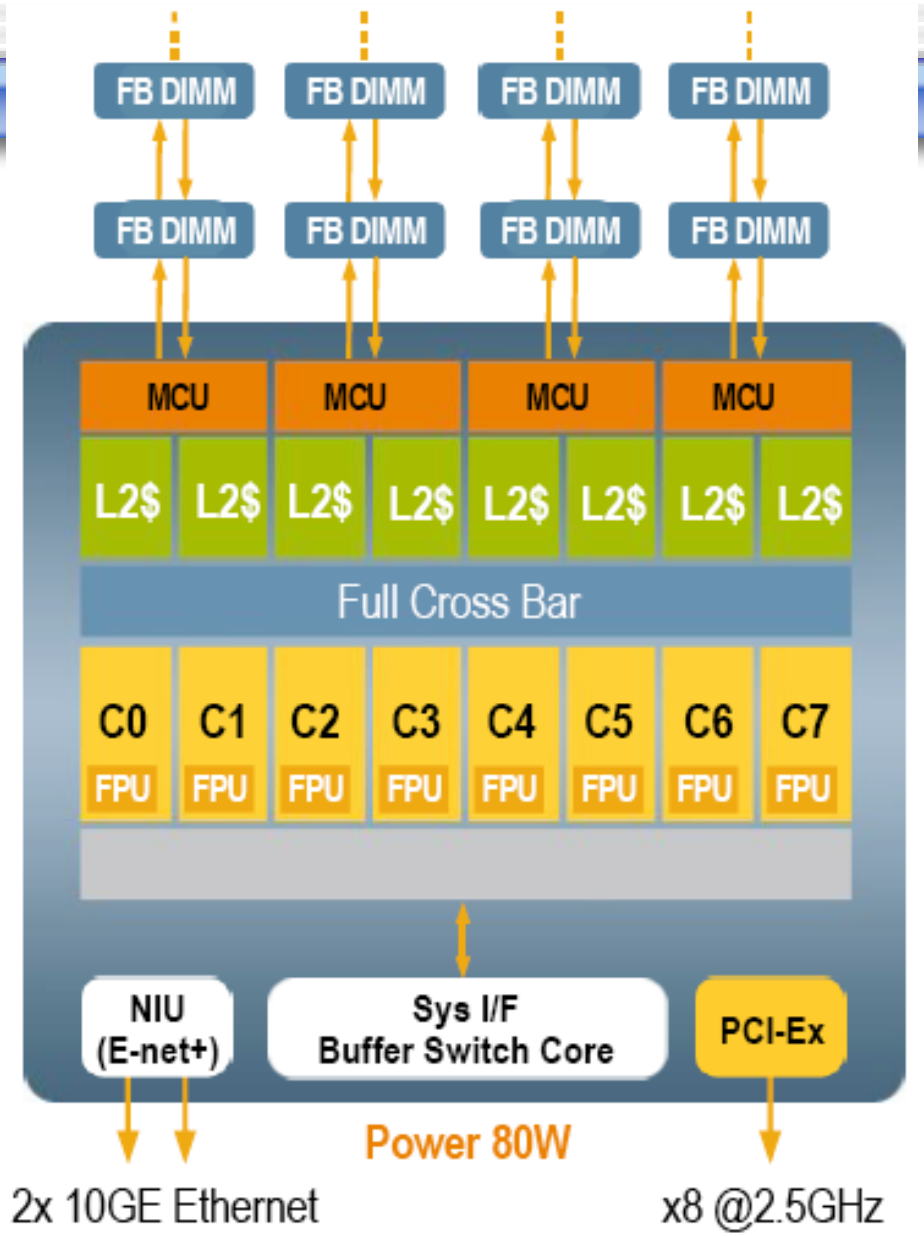
The Era of Chip Multiprocessors

- ILP and deep pipelining have run out of steam:
  - ILP parallelism in applications has been mined out
  - The power and complexity of microarchitectures taxes our ability to cool and verify
  - Frequency scaling is now driven by technology
- Single-chip multiprocessors systems are here
  - In embedded, server, and even desktop systems
  - Cores are the new GHz
- Now need parallel applications to take full advantage of microprocessors!
  - Problem: need a new approach to HW and SW systems based on parallelism
  - Concurrency revolution

Predicting The End of Uniprocessor Performance



Sun Niagara 2 at a Glance



- 8 cores x 8 threads = 64 threads
- Dual single issue pipelines
- 1 FPU per core
- 4MB L2, 8-banks, 16-way S.A
- 4 x dual-channel FBDIMM ports (60+ GB/s)
- > 2x Niagara 1 throughput and throughput/watt
- 1.4 x Niagara 1 int
- > 10x Niagara 1 FP
- Available H2 2007

The Looming Crisis

- By 2010, software developers will face…
- CPU's with:
  - 20+ cores (e.g. Sun Niagara and Intel Manycore)
  - 100+ hardware threads
  - Heterogeneous cores and application specific accelerators
  - Deep memory hierarchies
  - >1 TFLOP of computing power
- GPU's with general computing capabilities

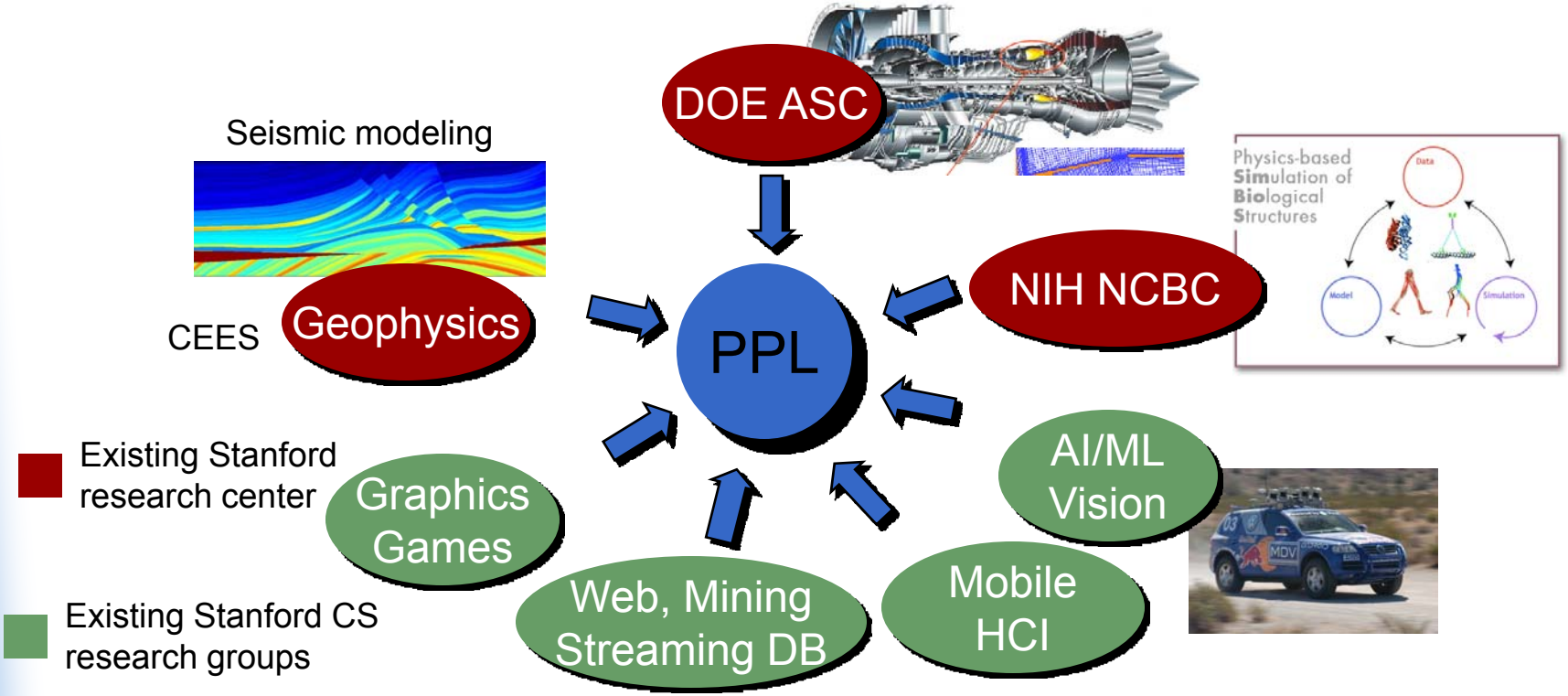
- **Parallel programming gap**: Yawning divide between the capabilities of today's programmers, programming languages, models, and tools and the challenges of future parallel architectures and applications

Components for Bridging the Parallel Programming Gap

Education	◆ New courses in parallel programming ◆ Move parallelism into mainstream CS curriculum
Parallel Applications	◆ Business, scientific, gaming, desktop, embedded ◆ Strong links to domain experts inside and outside of Stanford
Programming paradigms	◆ High-level concurrency abstractions ◆ Domain specific languages
Architecture	◆ Hardware support for new paradigms ◆ Real system prototypes for complex application development

Demanding Applications

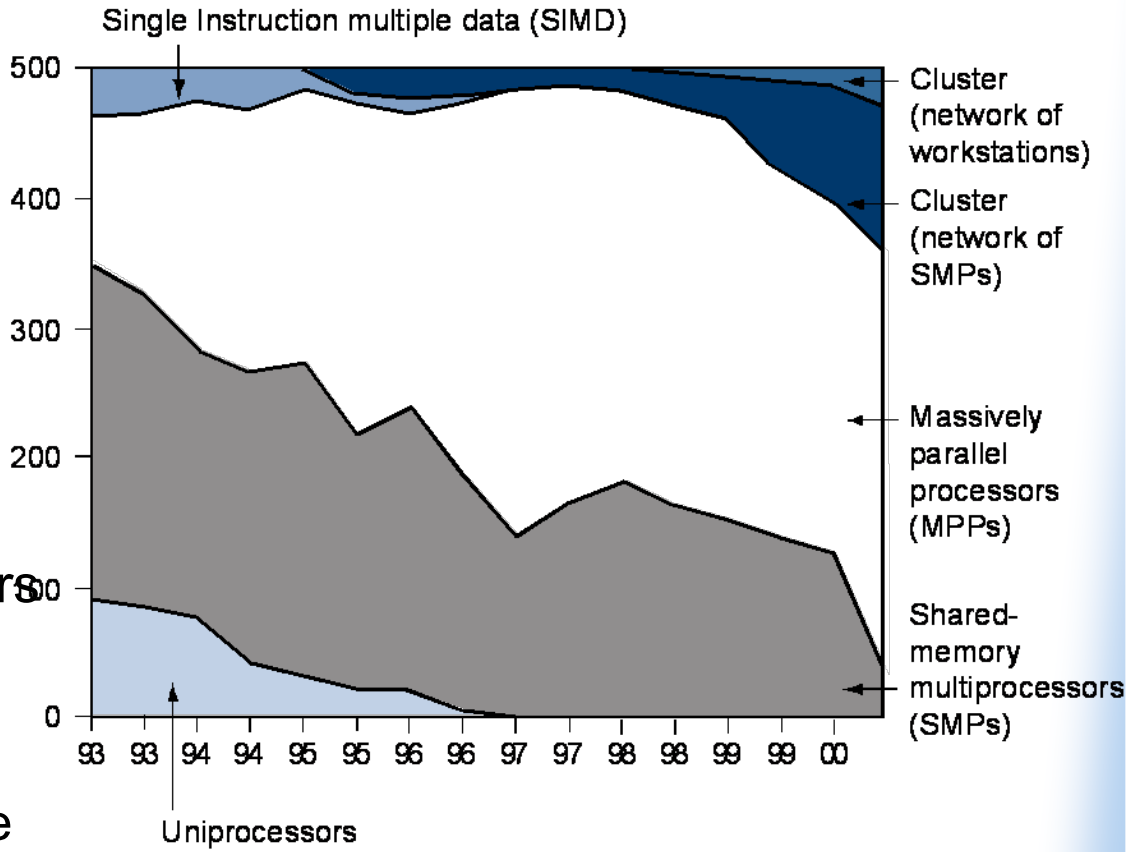
- Driving applications that leverage domain expertise at Stanford
- CS research groups & national centers for scientific computing
- New and emerging application areas (e.g. Neuroscience)



Shared Memory vs. Message Passing

- Lots of discussion in 90's with MPPs
  - SM much easier programming model
  - Performance similar, but MP much better for some apps
  - MP hardware is simpler
- Message passing won
  - Most machines > 100 processors use message passing
  - MPI the defacto standard
- Programmer productivity suffers
  - It takes too long to do "computational science"
  - Architectural knowledge required to tune performance

Plot of top 500 supercomputer sites over a decade



High-Level Programming Paradigms

- Need new parallel programming paradigms
  - Raise level of abstraction
  - Domain specific programming languages
  - Declarative vs. imperative
- Map-Reduce
  - Data parallelism (data mining, machine learning)
  - CMPs or clusters
- SQL
  - Information data management
- Synchronous Data Flow
  - Streaming computation
  - Telecom, DSP and Networking
- Matlab
  - Matrix based computation
  - Scientific computing
- What have we missed?

Parallelism Under the Covers

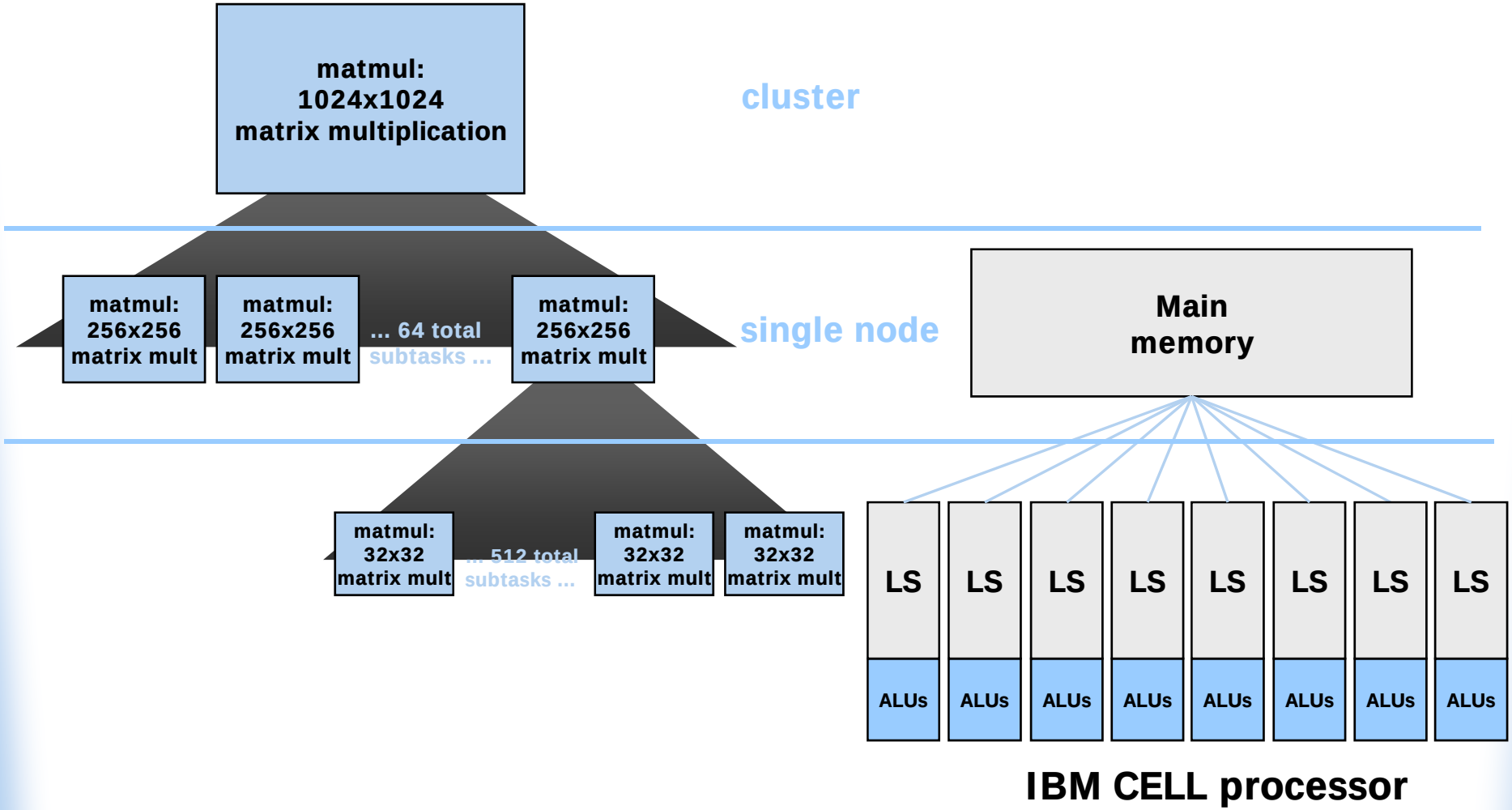
- Streams
  - Beyond message passing
  - Explicitly managed data transfers
  - Maximize use of memory and network bandwidth
- Transactions
  - Beyond shared memory
  - Eliminate locking problems and manual synchronization
  - Structured parallel programming
- Other Paradigms
  - Combining transactions and streams with mix of explicit and implicit data management

Streams

Alex Aiken, Bill Dally & Pat Hanrahan

- Goal
  - Efficient, portable, memory hierarchy aware, parallel, data-intensive programs
  - Required to achieve high performance with explicitly-managed deep memory hierarchies (e.g. clusters, IBM Cell)
- Approach: Sequoia programming
  - C++-based streaming language
  - Model parallel machines as trees of memories
  - Structure program as tree of self-contained computations called tasks
    - Data movement is explicit by parameter passing between tasks
    - Task hierarchies are machine independent
  - Map task hierarchies to memory hierarchies
    - Tasks are varied and parameterized to map efficiently to different levels of the memory hierarchy

Sequoia Matrix Multiplication



Transactional Memory

Christos Kozyrakis & Kunle Olukotun

- Programmer specifies large, atomic tasks
  - atomic { some_work; }
  - Multiple objects, unstructured control-flow, …
  - Declarative: user simply specifies, system implements details
- Transactional memory provides
  - Atomicity: all or nothing
  - Isolation: writes not visible until transaction commits
  - Consistency: serializable transaction commit order
- Performance through optimistic concurrency
  - Execute in parallel assuming independent transactions
  - If conflicts are detected, abort & re-execute one transaction
    - Conflict = one transaction reads and another writes same data

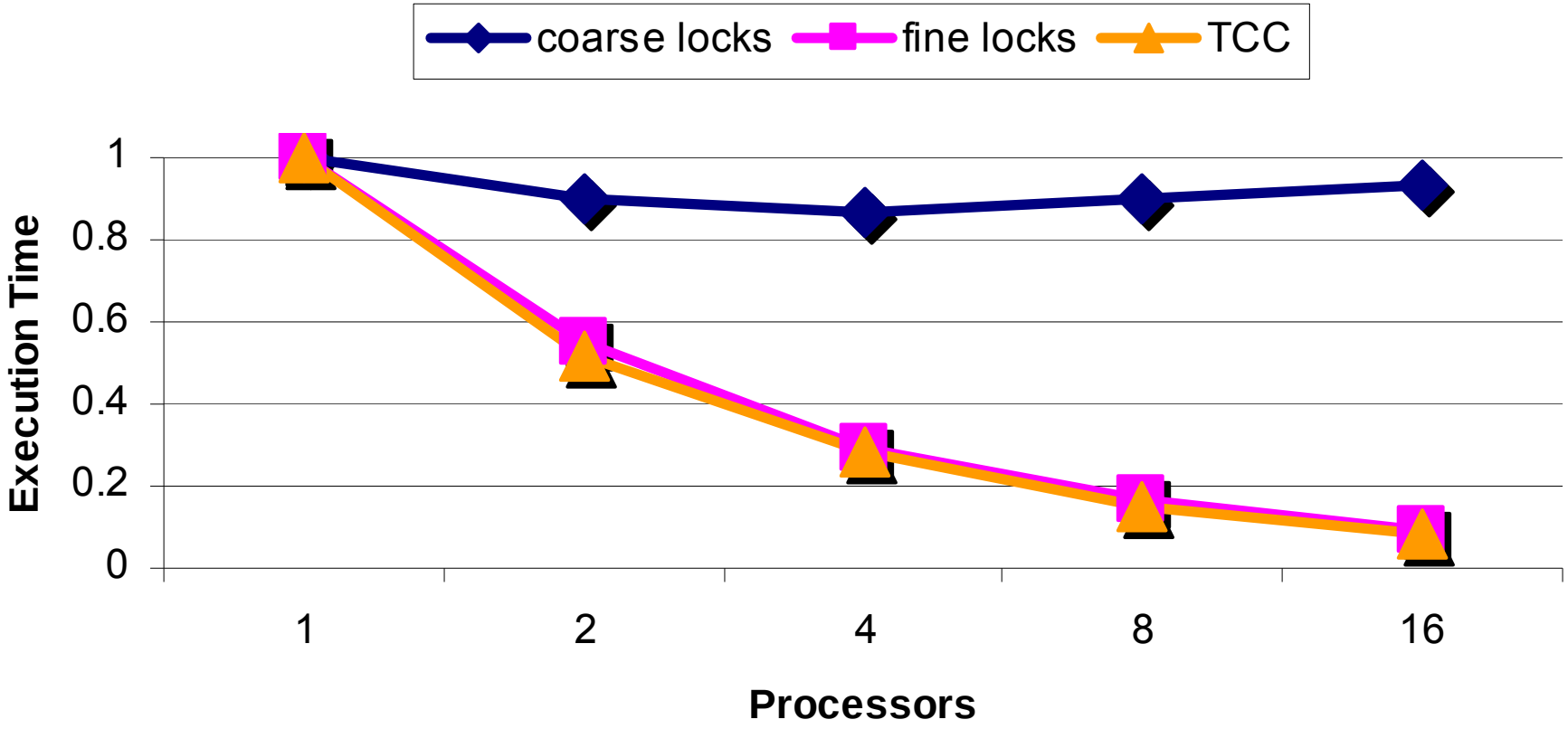
Why Transactional Memory?

- Locks are broken
  - Performance – correctness tradeoff
    - Coarse-grain locks: serialization
    - Fine-grain locks: deadlocks, livelocks, races, …
  - Cannot easily compose lock-based code
- TM simplifies parallel programming
  - Parallel algorithms: non-blocking sync with coarse-grain code
  - Sequential algorithms: speculative parallelization
- TM improves scalability and performance debugging
  - Facilitates monitoring & dynamic tuning for parallelism & locality
- TM enhances reliability
  - Atomic recovery from faults and bugs at the micro level
  - Clear boundaries for checkpoint schemes at the macro level

The Stanford Transactional Coherence and Consistency (TCC) Project

- A hardware-assisted TM implementation
  - Avoids ≥2x overhead of software-only implementation
  - Does not require recompilation of base libraries
- A system that uses TM for coherence & consistency
  - All transactions, all the time
  - Use TM to replace MESI coherence
    - Other proposals build TM on top of MESI
  - Sequential consistency at the transaction level
    - Address the memory model challenge as well
- Research on applications, TM languages,TM system issues, TM implementations, TM prototypes,…
  - See posters

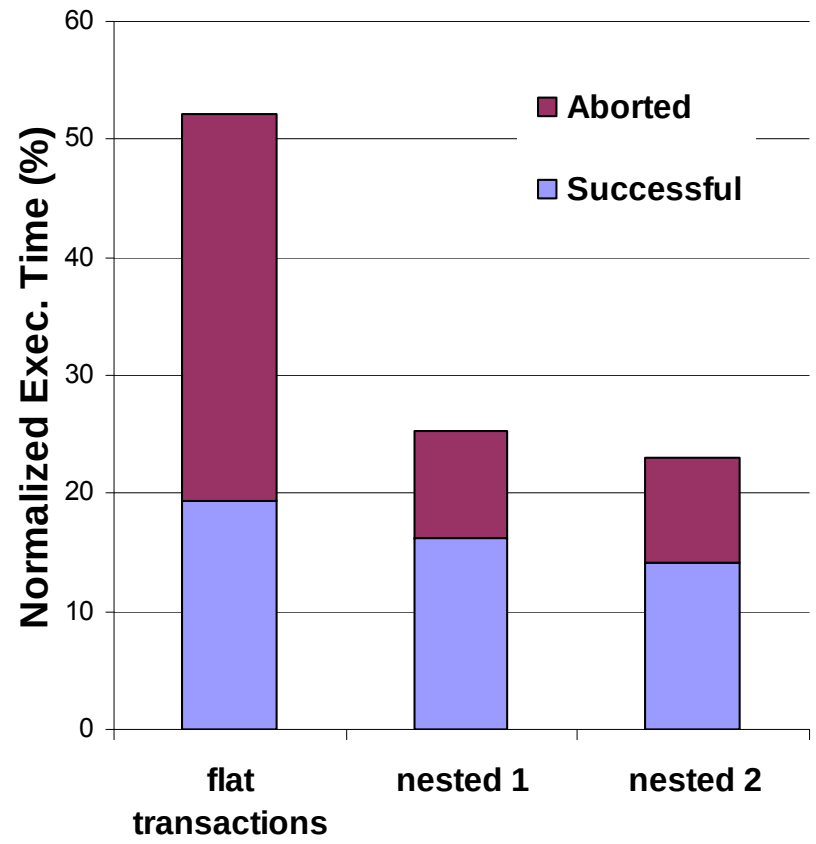
Java HashMap performance



Transactions scale as well as fine-grained locks

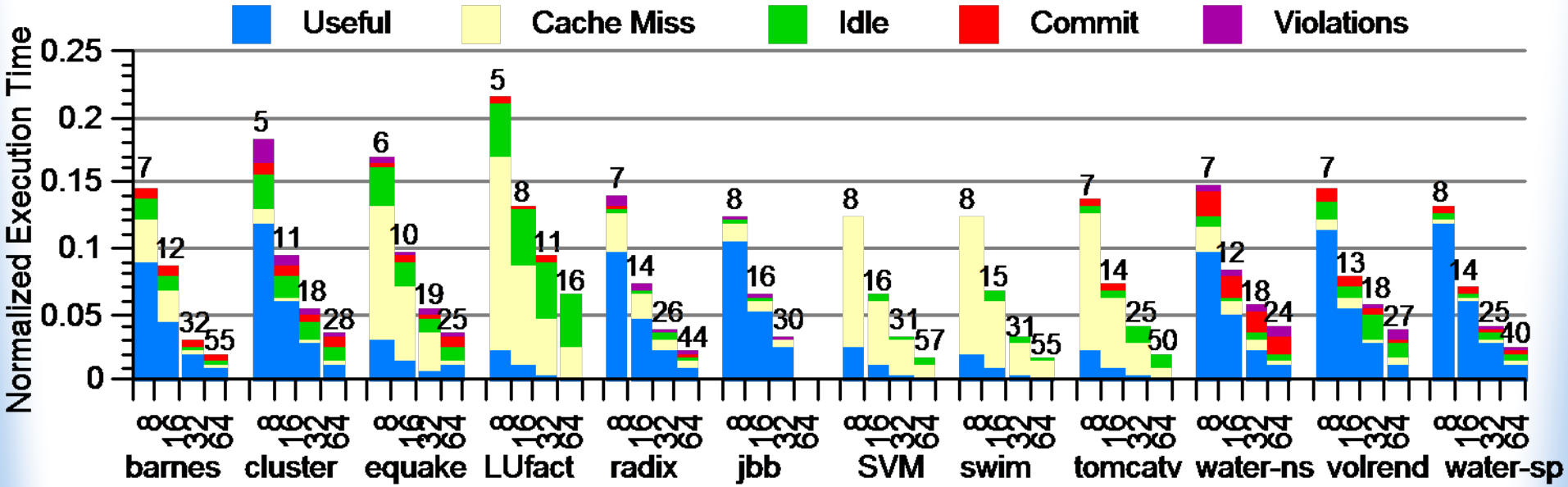
TCC Performance On SPEC JBB

- Simulated 8-way CMP with TCC support
- Speedup over sequential
  - Flat transactions: 1.9x
    - Code similar to coarse-grain locks
    - Frequent aborted transactions due to dependencies
  - Nested transactions: 3.9x to 4.2x
    - Reduced abort cost OR
    - Reduced abort frequency
- See paper in [WTW'06] for details
  - http://tcc.stanford.edu



Scalable TCC Performance
64 proc DSM

- Improve commit
  - 2-phase commit parallel commit
- Excellent scalability
  - Commit is not a bottleneck



PPL: A Real Systems Approach

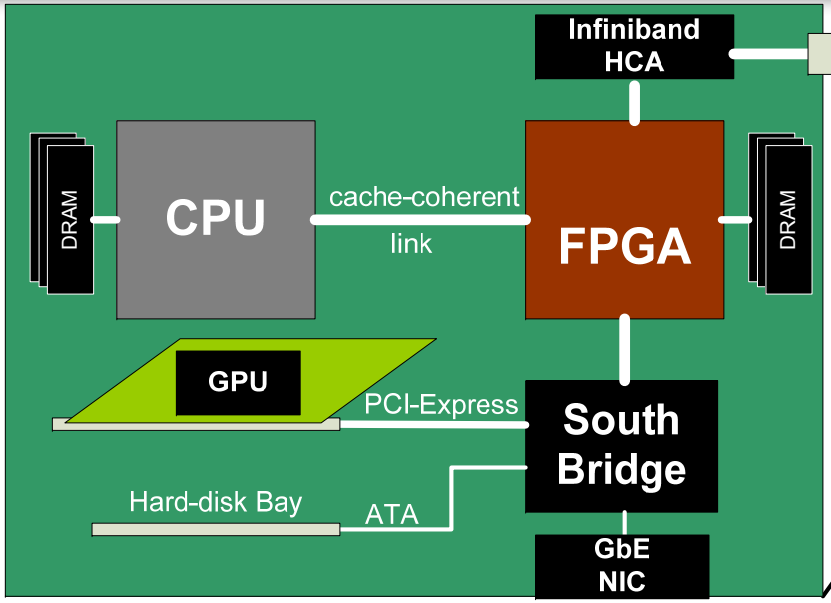
- Experiment with full-system prototypes
  - Research at full scale
  - Involves research in HW, system SW, and PL
  - Gets students excited
  - Forces the development of robust technology
  - Shows industry the way to commercialization
  - Many examples at Stanford: SUN, MIPS, DASH, Imagine, Hydra

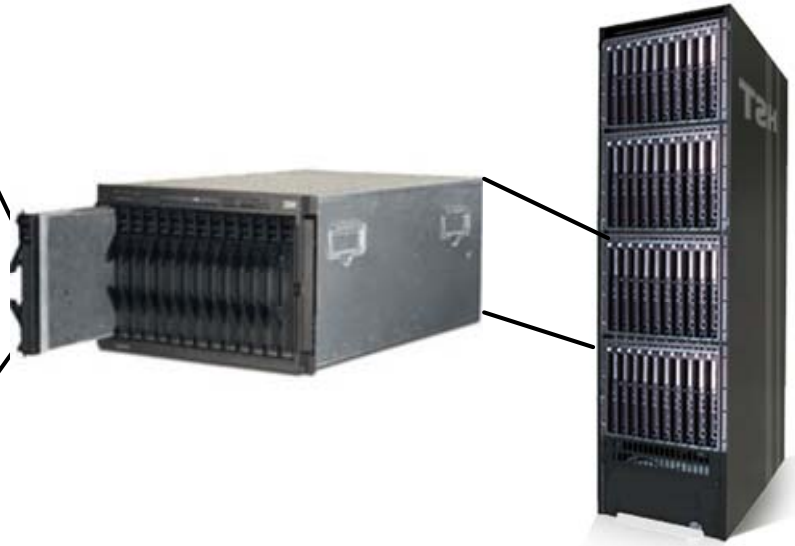
- Flexible Architecture Research Machine (FARM)
  - High-performance cluster technology augmented with FPGAs in the memory system
  - Platform enables significant parallel software and hardware co-development

Stanford FARM in a Nutshell

- An industrial-strength scalable machine
  - 0.1- 0.25 TFLOPS per node
  - Infiniband interconnect
  - Can scale to 10s or 100s nodes
  - Run real software
- A research machine
  - Can personalize computing
    - Threads, vectors, data-flow, reconfigurable
  - Can personalize memory system
    - Shared-memory, messages, transactions, streams
  - Can personalize IO and system support
    - Off-load engines, reliability support, …
  - Using FPGA technology

FARM Organization





Blade Server Chassis Blade Server Rack

- Industrial strength parallel system
  - High-end commodity CPU (AMD, Intel, Sun) and GPU
  - Infiniband interconnect
- Flexible protocol in FPGA
  - Connect to CPU through coherent link

Example Uses of FARM

- Transaction-based Shared-memory architecture
  - Implement TM hardware in FPGA
    - Directory, RAC, proactive research engine
  - Research
    - Explore scalability limits of transactions
    - Explore locality optimization potential of transactions
- Streaming architectures
  - Extended, chip-to-chip streaming engine in the FPGA
- Other
  - New hybrid architecture models
  - Accelerate networking tasks
  - Provide new functionality for reliability, debugging, and profiling

Summary

- Harnessing high degrees of thread-level parallelism is the next big challenge in computer systems
- "parallelism is the biggest challenge since high-level programming languages. It's the biggest thing in 50 years because industry is betting its future that parallel programming will be useful." David Patterson
- Pervasive Parallelism Lab
  - Figure out how to build/program/manage large scale parallel systems (1000s–10,000s of CPUs) for average programmers
  - Opportunity to collaborate on hardware and software research
    - Languages, programming models, operating systems, applications and architecture
- Kunle@stanford.edu