



Software for High Performance Computing

Requirements & Research Directions

Marc Snir

May 2006

Outline

- Petascale hardware
- Petascale operating system
- Programming models

Petascale Systems are Coming

- NSF plan to fund one petascale system to be available in 2010; DOE will also install petascale systems in 2010.
 - Petascale = 10^{15} operations per second; or
 - 10^{15} floating point operations per second (petaflops) peak; or
 - 10^{15} floating point operations per second (petaflops) sustained

Hardware Trends

- Chip density continues to increase; but computer architects do not know how to use the extra transistors in order to increase single processor performance
 - ➔ Vendors are moving to chip multiprocessors
- Clock speed does not increase
 - ➔ Performance growth toward Petascale will come almost uniquely from increased parallelism

Problems

1. Very large number of concurrent threads
(500,000 – 1,000,000)
2. Frequent failures (?)
 - might be resolved with hardware redundancy
3. Very hierarchical & possibly heterogeneous system

Hierarchy and Heterogeneity

- Hierarchy (due to package boundaries and distance)
 - Multi core chip with 8—16 core
 - 1—3 cycles
 - Multi chip SMP node [board] with 4-16 chips (shared L3/L4)
 - 10—50 cycles
 - Global network
 - 100 – 500 cycles
- Possible heterogeneous systems
 - chip with distinct cores: e.g., IBM Cell processor
 - distinct chips in node: e.g., node with attached FPGA accelerator
 - system with distinct nodes

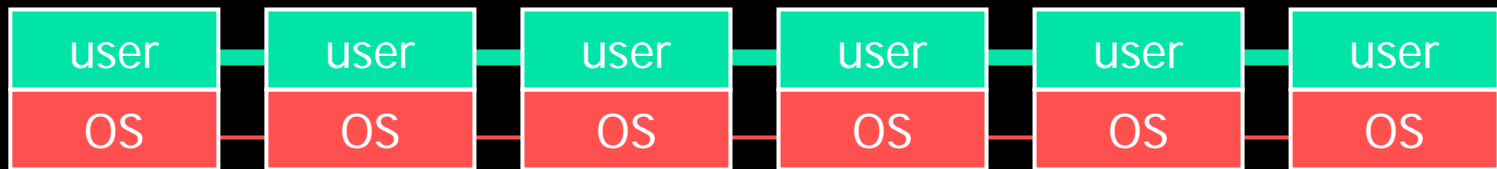
Communication

- Communication time to remote memory \gg communication time to local memory
 - Time of flight, switching time, queuing delays
- ➔ Cannot access remote memory efficiently via load/store operations; need bulk transfer operations
 - hw does not support enough pending operations; compiler does not issue loads early enough
- Message passing (2-sided communication): send, receive...
- Remote memory access (1-sided communication): put, get, accumulate
 - vector (location, length)
 - Scatter/gather (list of locations)
- 2-sided communication easier to support in software atop simple channel protocol; but
- 1-sided communication easier to support directly in hardware
- ➔ Network interface supports in hardware/firmware 1-sided communication, with 2-sided implemented in software (Infiniband, Myrinet, Quadrix...)

Petascale OS

■ Cluster OS

- Tightly coupled parallel application
- Distributed OS + (few) cluster services



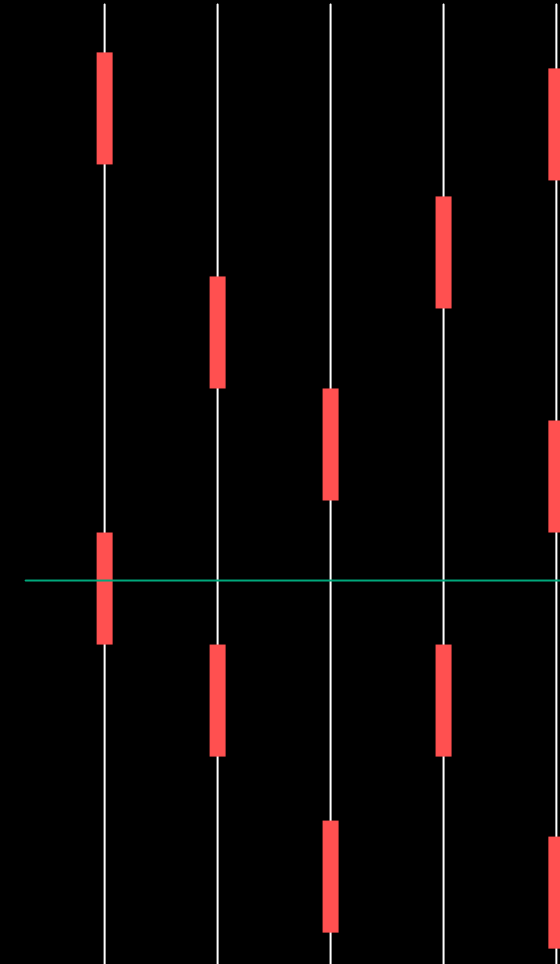
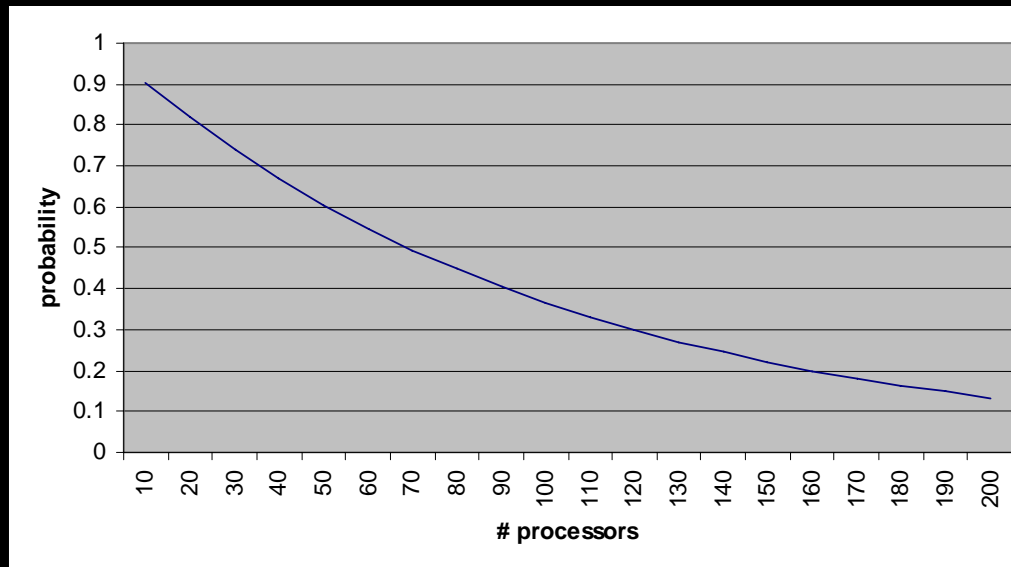
■ Problems

- OS Noise
- OS reliability (frequent crashes)
- Abstraction mismatch (no OS abstractions for parallel jobs)

(SPDP 95)

OS Noise

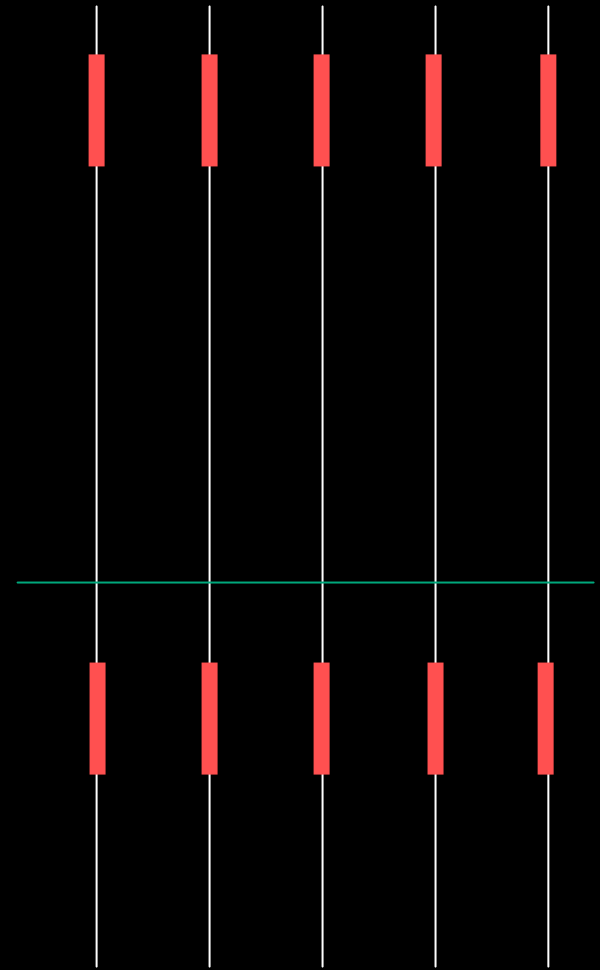
- Regular OS has many background activities (daemons, heartbeats...) that happen at random times
- Parallel programs often use barrier synchronization



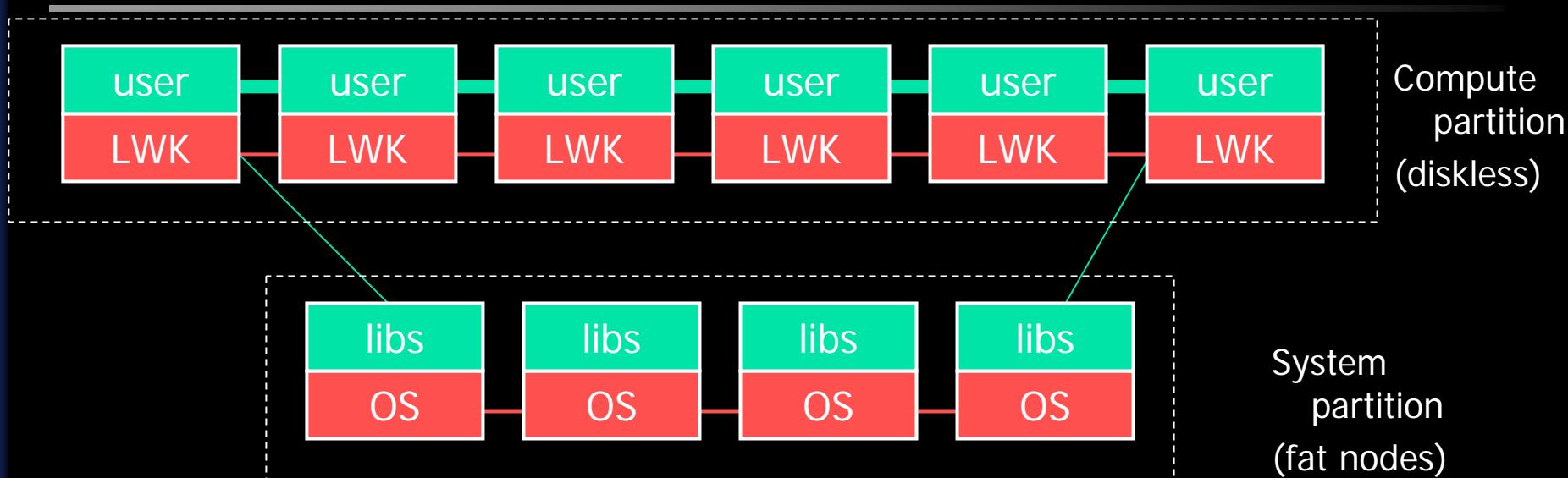
Probability of completing barrier without being interrupted, assuming 1% background activity

OS Noise (continued)

- Sequential processor with 1% background system activity “looks like” dedicated system running at 99% speed; same not true for parallel system.
- Solutions:
 1. avoid background noise (reduce OS functionality)
 2. synchronize background noise (provide right abstraction)

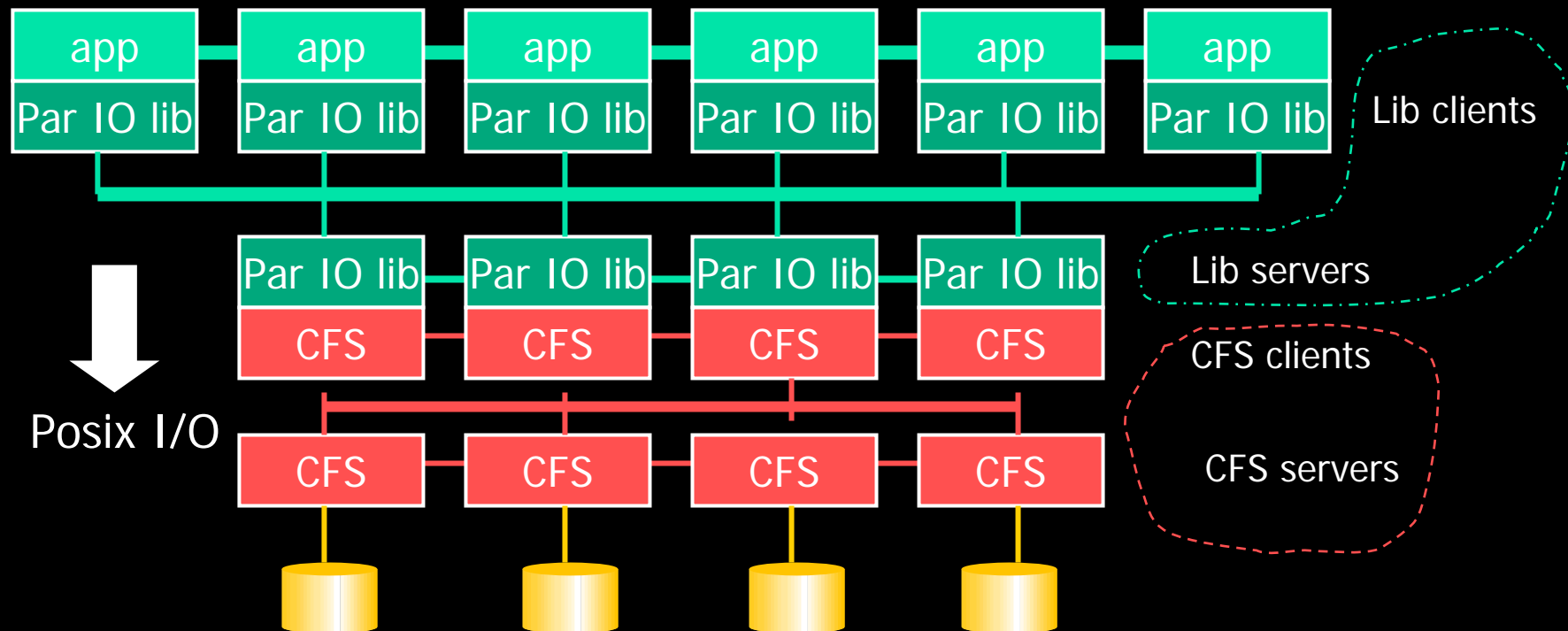


Solution (1)



- Light-weight kernel
 - no daemons, no server threads, no demand paging...
 - supports directly some Posix calls
 - offloads to proxy on OS node other calls
- Example: Cray XD1 & IBM Blue Gene/L
- Helps with noise and reliability
 - Less so, for data-intensive (I/O intensive) computing...
- Does not help with abstraction mismatch

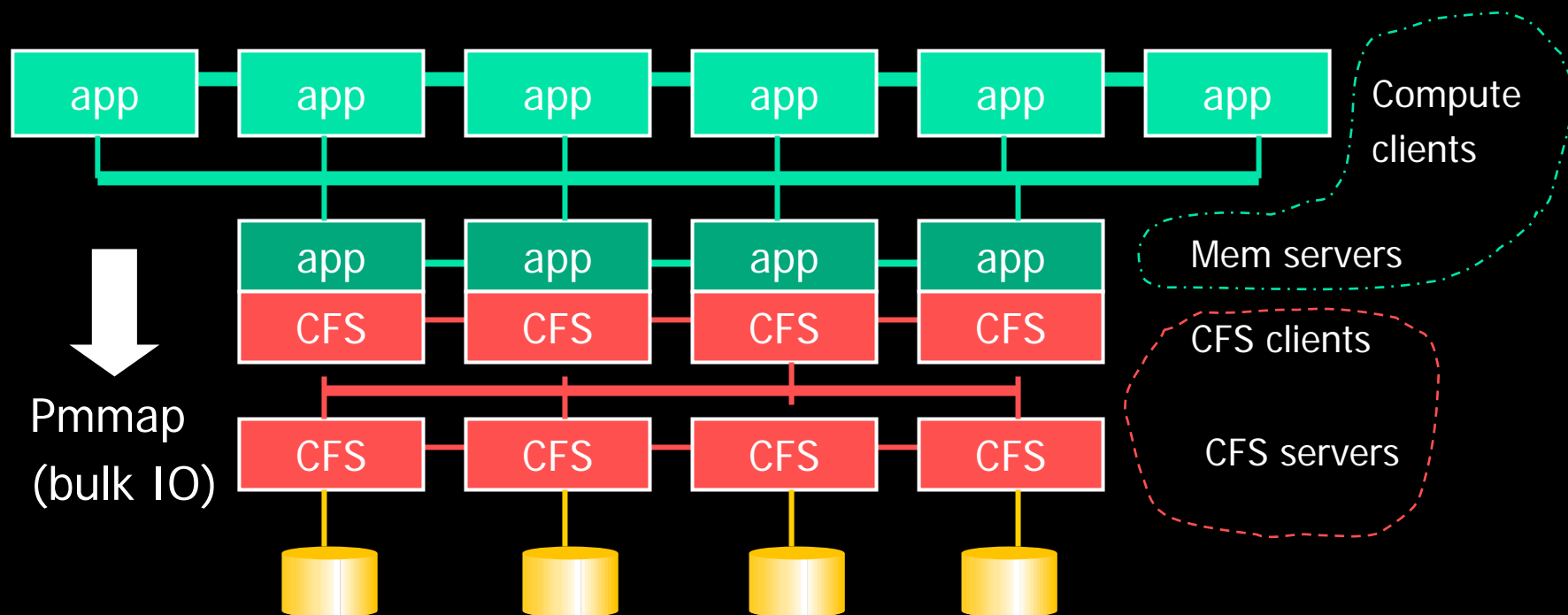
Example: Cluster File System



■ Abstraction mismatch:

- Posix semantics (file pointer)
- Performance requirements (30K fopen/sec!)
- Protection, sharing & coherence mechanisms (**distinct processes of same parallel application \neq distinct jobs**)

Alternative – distributed mmap



- Collective memory map call $pa = \text{mmap}(\text{addr}, \text{len}, \text{prot}, \text{flags}, \text{fildes}, \text{off}, \text{dist})$
 - Buffers between (temporal and spatial) fine grain app IO and coarse grain file IO
 - Issued by user (prefetch) or by library (collective miss)
 - Does not require coherence protocol!
 - Natural extension to global array libraries and languages
 - Possible gang scheduling point

Parallel OS

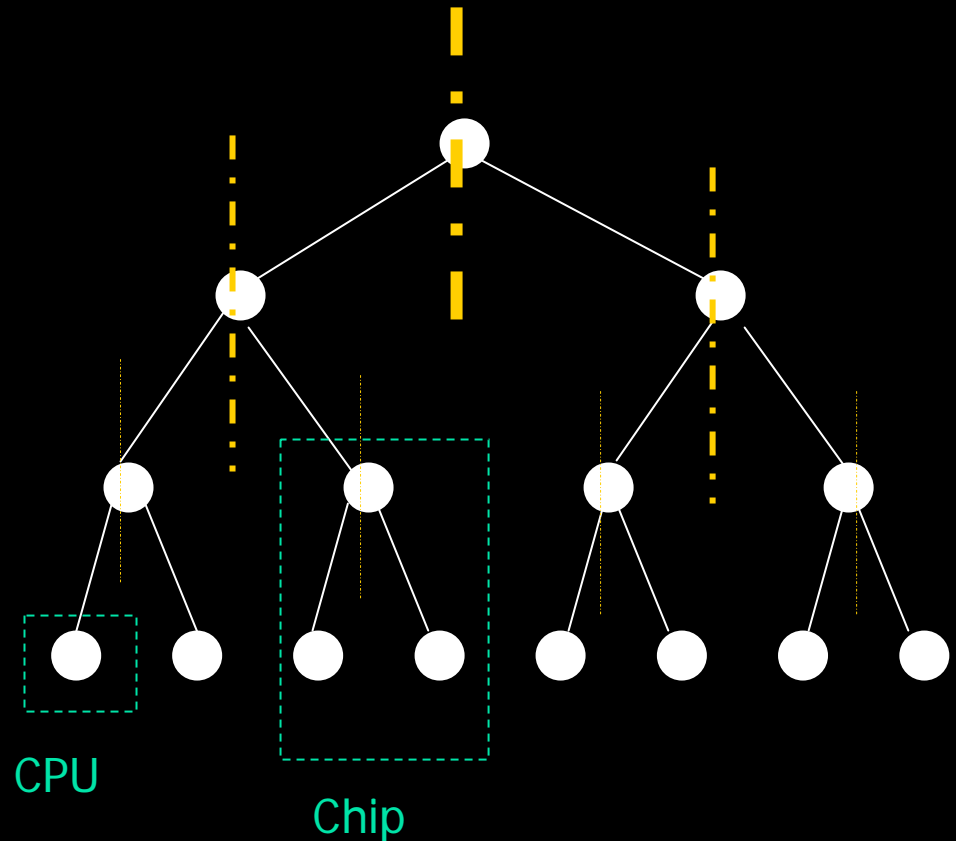
- **Parallel OS \neq Scalable OS**
 - support for parallel applications, not scaling of distributed OS
- Add collective resources
 - files, distributed memory buffers, distributed process cohorts...
 - resources (distributed or replicated) are owned by cohorts, not by individual processes
- Add collective calls
 - pfdopen, pmmmap, pshmget, pfork,...
- Virtualize nodes
 - cohortid.index, rather than (node, pid)
 - support migration for fault tolerance & ease of management
- Posix interfaces used for legacy; parallel interfaces used for performance

Programming for Deep Hierarchies

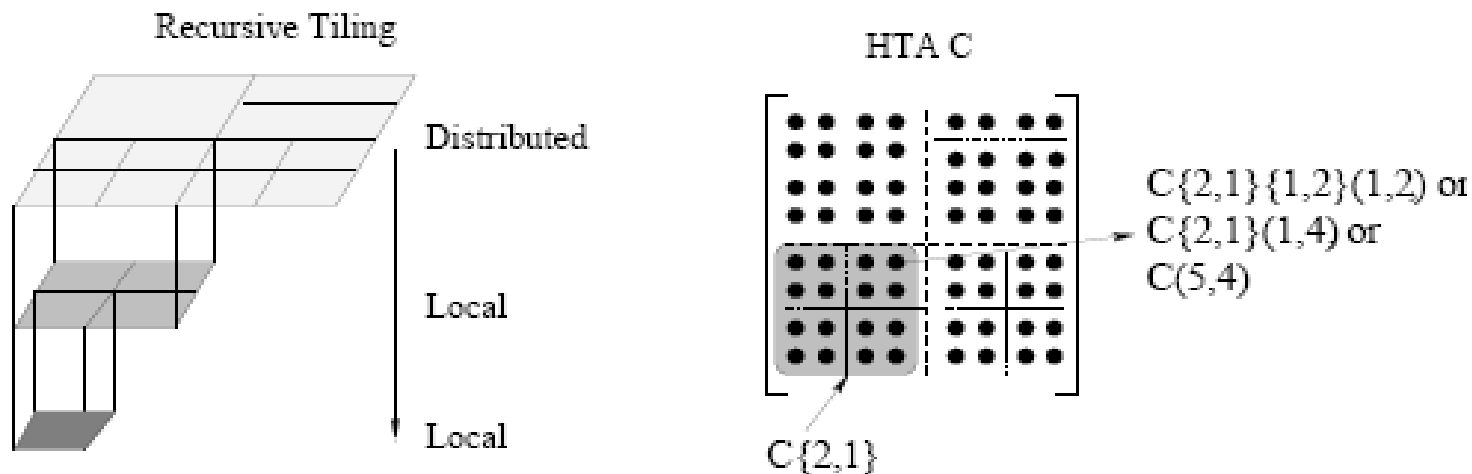
- “Wrong problem”: different communication models at different levels (MPI+OpenMP)
- “Right problem”: communication structure of algorithm should match architecture hierarchy
 - Assume algorithm using alternating computation/communication phases:
 - All communications during a phase should all have “same locality” (on chip, on node, global)
 - Local communication phases should be more frequent than global communication phases.
- Hard if need to explicitly map problem to machine topology
- Possible solution: recursion (nested dissections)

Recursive Algorithms

- Many parallel algorithms can be expressed recursively
 - Matrix product, FFT, Multigrid...
- Use recursive structure to map to machine
- Will get efficient mapping, no matter what the hierarchy is (Snir, Leicerson)



Example: Hierarchically Tiled Arrays (HTA)



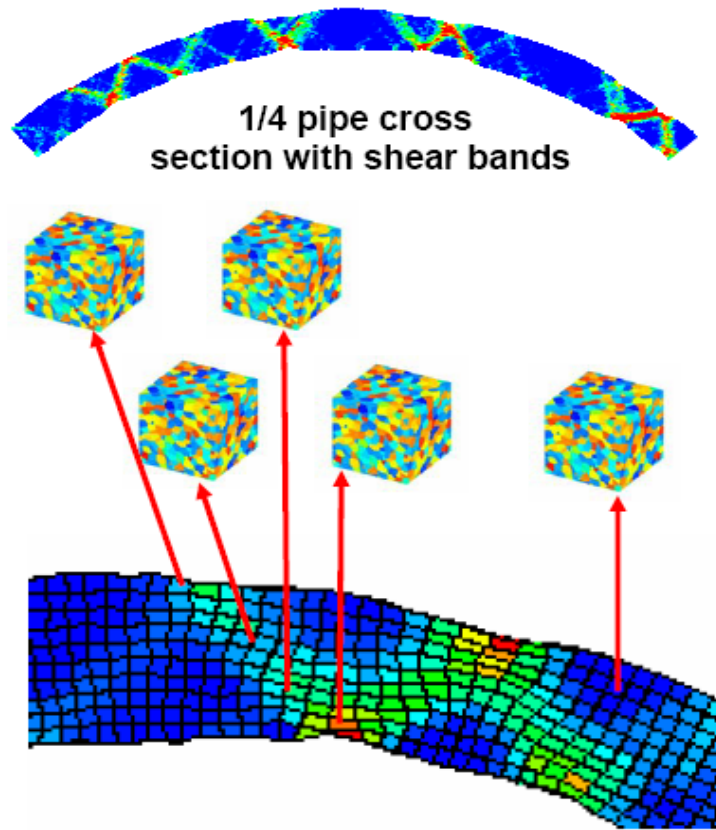
- Padua et. al.
- Added HTA's to Matlab
 - extended (data parallel) array operations to work with HTA's

Programming for Noisy Systems

- “Crystalline” model:
 - Fixed number of processes, all moving at same speed.
 - A computation consists of a sequence of global phases; all processes do “same amount of work” during a phase.
- Problems:
 1. Increased variance in processor speed
 - Dynamic power management
 - Dynamic error recovery
 - Asynchronous software (e.g., monitoring for debugging and performance tuning, concurrent checkpointing, etc.)
 2. Increased variance in compute work per phase

Example: Adaptive Sampling

Expanding cylinder model with 50 grains at each FE integration point



- Apply *fine scale model* to elements where continuum model is invalid...
- ...but to *just a sample* of the elements – denser sample where necessary
- Elsewhere, *interpolate* response function in state space from fine scale results calculated for similar elements
- *10X or better performance improvement achievable over 100% sample*
- **Requires load balancing**

(Steve Ashby)

Load Balancing & Dynamic Resource Allocation

- Done internally, as part of application logic
 - E.g., Zoltan Load Balancing Library, Sandia
 - Requires that entire application be managed by library; does not work when independently developed codes are integrated into one multi-physics application
- Done externally, by runtime
 - may be steered by application
 - **requires processor virtualization**
 - replace “processor” by run-time managed thread

On the Many Advantages of Processor Virtualization

- Facilitates composition of multiple paradigms
 - E.g., Charm++ and AMPI (Kale)
- Supports adaptive overlapping of computation and communication (message-driven scheduling)
- Supports run-time load balancing
- Supports migration and checkpointing
- Supports run-time communication optimization
- Improves cache performance
- **Does not worsen performance!** (Kale, NAMD)
- Need to work on interaction with rDMA (no message driven scheduling)

Programming Models

- Communication model
 - 2-sided (send-receive)
 - 1-sided (put, get)
 - 0-sided (load, store)
- Computation location model
 - static
 - dynamic
 - automatic
- Data location model
 - static
 - dynamic
 - automatic

Current Libraries and Languages

■ MPI

- 2-sided, 1-sided; static data and control allocation

■ OpenMP

- 0-sided; dynamic allocation of control and automatic allocation of data

■ Global Shared Array Languages (UPC, CAF)

- 0-sided
- static allocation of control
- static allocation of data (partitioned arrays)

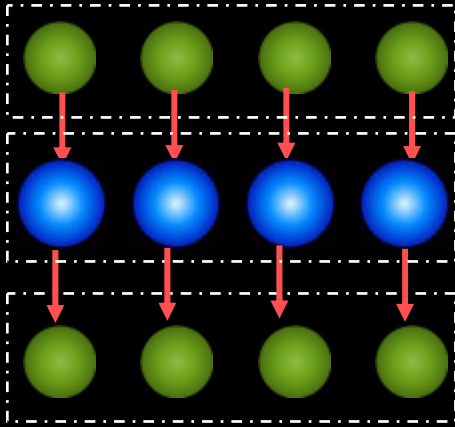
Problems with Current Languages

- 0-sided communication convenient, but compilers do not do a good job at optimizing communication when latency is high; need user control on data movement => 1-sided model
- Static allocation of data and control is too limited; need to support data and control migration
- All programming models associate variable name with location: if data is explicitly moved then it is renamed; this complicates programming and prevents compatibility between shared memory and distributed memory systems

Possible Solution (PPL1, Snir)

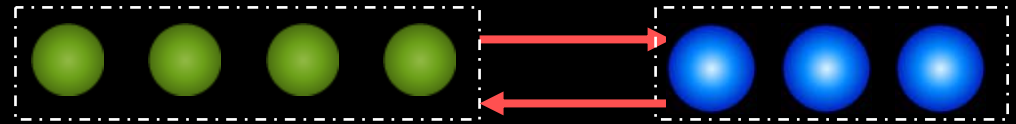
- Virtual locales
- Execution blocks (e.g., iterates in parallel loop) are explicitly associated to locales (**dynamic allocation of computation**)
- Arrays are partitioned over locales; partitions can be changed dynamically (**dynamic allocation of data**)
- **Locale is an abstraction for expressing locality, not a physical resource**
 - run-time dynamically maps locales to processors or nodes
- **PPL1 provides global name space with user control of locality**
 - data is moved to thread that needs it by repartitioning array; name does not change
 - compiler can optimize away data movement on shared memory system
- Always provide to compiler as much information as possible as early as possible on data access pattern
- Control locality with data remapping operation, not with explicit data copying

Parallel Component Models



Time slicing

- Collective method invocation
- Idle time if have load imbalance
 - Processor virtualization may avoid both pitfalls!
- Need more general model: *asynchronous parallel invocation*
 - Different levels of parallelism in two components
 - Possibly different data distribution (redistribution is part of invocation)
 - Blocking or nonblocking invocation (parallel future?)



Space slicing

- Communication between spatially disjoint component
- Loss of locality

Summary

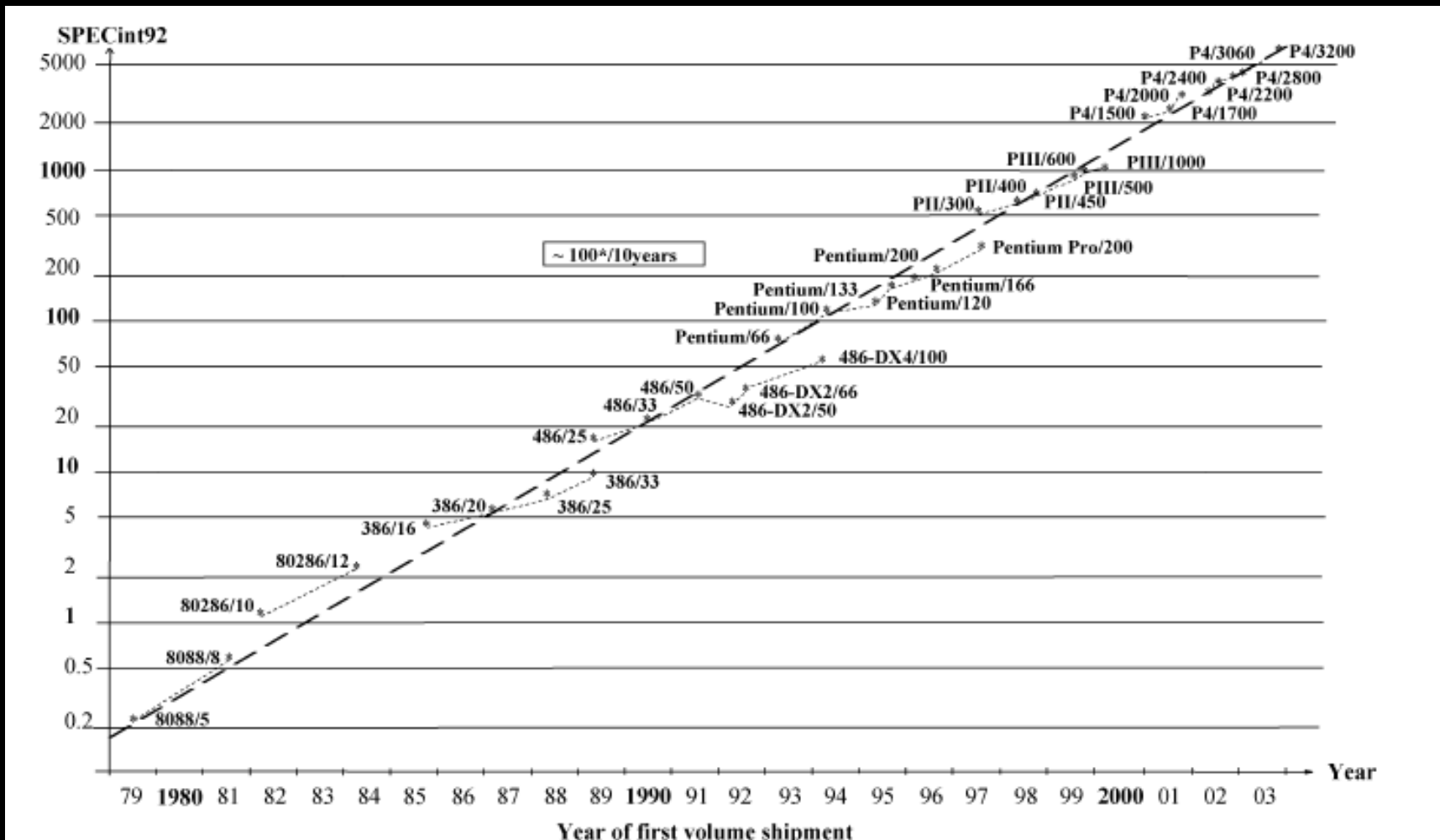
- Petascale can be reached with current programming models – but it's becoming increasingly hard to program [sound wall vs. heat wall]
- Parallel software has not evolved in last decade [applications have evolved]
- There is important research to do on High Performance Computing Software
- It will be hard to have community adopt new software paradigm [the boiling frog]

Questions?

Dennard's Scaling Principle

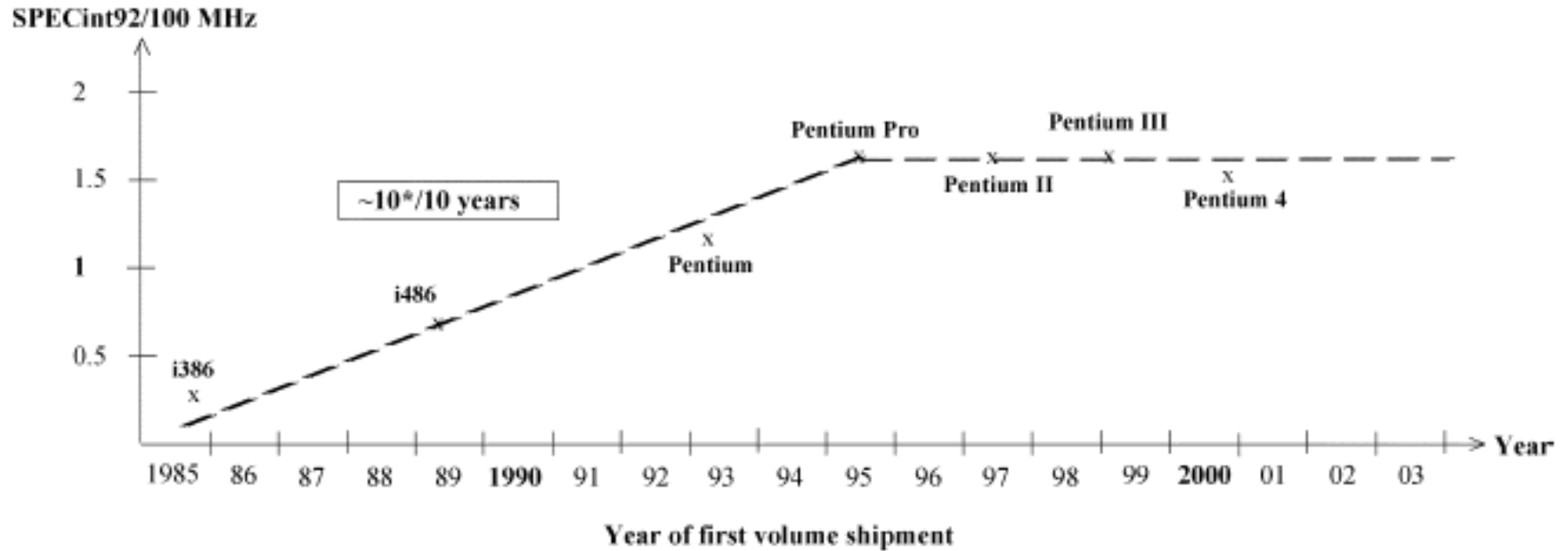
- If feature size, and voltage are scaled down by λ then
 - transistor count increases by λ^2
 - clock frequency increases by λ ; and
 - power stays constant.

“Moore’s Law”



(D. Sima)

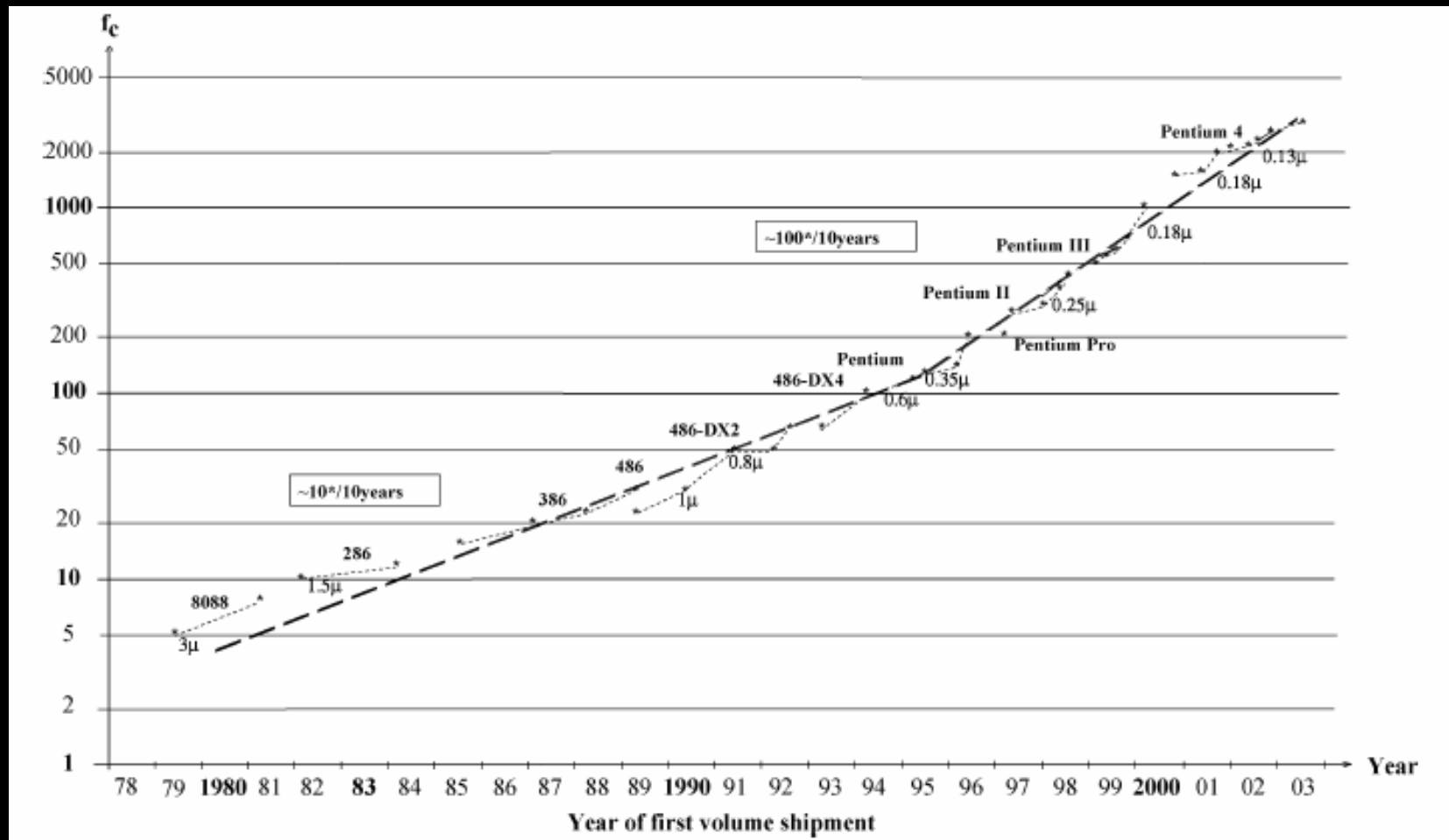
Instructions per Cycles (IPC)



- Computer architects have run out of tricks!

(D. Sima)

Frequency Evolution



(D. Sima)

Microprocessor Evolution

- Technologists have increased clock frequency faster than have decreased feature size (to compensate for the failure of computer architecture)
 - This increases power consumption
- Technologists cannot scale down voltage anymore, because of leakage current
 - This also increases power consumption
- ➡ So chips have become much warmer
- Chips have reached the limit of power density
- ➡ Single processor performance is barely increasing now; instead, multiple processors (cores) are put on one chip