Note Taker Checklist Form -*MSRI*

Name : Rob Stapleton

E-mail Address/ Phone #: jrstaple@ncsu.edu

Talk Title and Workshop assigned to:
Interactive Parallel Computing in Support of Research in Algebra, Geometry and Number Theory

Lecturer (Full name): Katherine Yelick
Date & Time of Event: 9:00 a.m Jan 30, 2007

Check List:

(✓) Introduce yourself to the lecturer prior to lecture. Tell them that you will be the note taker, and that you will need to make copies of their own notes, if any.

( ) Obtain all presentation materials from lecturer (i.e. Power Point files, etc). This can be done either before the lecture is to begin or after the lecture; please make arrangements with the lecturer as to when you can do this.
working on it!

(✓) Take down all notes from media provided (blackboard, overhead, etc.)

( ) Gather all other lecture materials (i.e. Handouts, etc.)

(✓) Scan all materials (notes) on PDF scanner in 2nd floor lab (assistance can be provided by Computing Staff) – Scan this sheet first, then materials. In the subject heading, enter the name of the speaker and date of their talk.

Please do **NOT** use **pencil** or colored pens other than black when taking notes as the scanner has a difficult time scanning pencil and other colors.

Please fill in the following after the lecture is done:

1. List 6-12 lecture keywords: parallel, interactive, programming models, multi-core, PGAS languages, Titanium

2. Please summarize the lecture in 5 or less sentences.
Moore's Law is alive and well when it comes to parallel computing. Clock speeds will not be expected to increase as much or as fast in the near future, so extra steps should be taken to harness the power of multicore machines.

Once the materials on check list above are gathered, please scan ALL materials and send to the Computing Department. Return this form to Larry Patague, Head of Computing (rm 214)

Jan 30: 9:00 a.m.
Programming Models for Parallel Computing
Katherine Yelick

http://titanium.cs.berkeley.edu/
http://upc.lbl.gov/

Not long ago, it looked as if the future of parallel computing was uncertain. Several panels titled "is parallel processing dead?"
No no, Moore's law is alive and well. Expect to continue for the next decade.
However, clock scaling bonanza has ended (no more speed increase there).
Must go to multicore processors.
-More cores with lower clock rates burn less power, lower temperatures.
-Instruction Level Parallelism (ILP) benefits declining
(We've had parallelism on single chips hidden from programmers)
The chips had the hardware (overengineered), but the software did not make use of it. Designers backed off.
-Yield Problems
IBM Cell processors have about a 10% yield. Blade system with all 8 is $20k, PS3 with 7 is $600.

Power Density Limits Serial Performance.
-Nowadays, same heat as a rocket nozzel.

The Revolution is Happening Now.
-Chip density continuing by Moore's Law, but clock speed is not. Little to no ILP to be found.

Why parallelism? (2007)
Multicore is all over the place. Not just theory any longer.
Will all programmers become performance programmers? Parallelism can "hide costs" by preprocessing / using parallelism
New features will have to be well-hidden, as clock speeds aren't increasing like they used to.
Big Open Questions:
1) What are the killer applications for multicore machines?
2) How should the chips be designed: multicore, manycore, heterogeneous?
3) How will they be programmed?
Intel announced they're looking at an 80 core processor.

We seem to be headed towards a PetaFLOp machine in 2008. (data from top500.org)
There's a 6-8 year lag between the #1 fastest computer and the #500 fastest, which is what many people are programming on. Will PetaFLOp machines be common by 2015?
Memory Hierarchy:
With explicit parallelism, performance becomes a software problem.
Off-chip latencies tend to go by only about 7ms/year

Predictions:
Parallelism will explode. Cores will double by about Moore's Law.
All top 500 machines will be PetaFLOp machines by 2015.
Performance will be placed on the software.
A parallel programming model will emerge to handle multicore programming and parallelism in general.

PGAS Languages

Parallel software is still an unsolved problem.
Most parallel programs are written using either Message passing with a SPMD model (scientific applications, scales easily) or shared memory with threads in OpenMP, Threads, or Java (non-scientific applications, easier to program, also lends itself very easily to user interfaces).

Partitioned Global Address Space (PGAS) Languages:
Maintains control over locality. Performance, programmability, flexability.

PGAS: Data is partitioned (designated) as global or local. Any thread/process may directly read/write data allocated by another.
3 current languages use an SPMD (Single Program, Multiple Data) execution model. 3 more are emerging: X10, Fortress, Chapel.
Remote references have a higher latency time, so should be used judiciously.

PGAS, commonalities:
Have both private and shared data
Support for distributed data structures
One-sided shared-memory communication
Synchronization (global barriers, locks, memory fences)
Collective Communication, IO Libraries, etc.
The 3 current languages are built to be close to the language off which they are based. UPC is based off of C, so it's low-level. Titanium is based off of Java, so is higher-level.

Private vs Shared Variables in UPC:
C variables and objects are allocated in the private memory space.
Shared variables are allocated only once, in thread 0's space.
Shared arrays are spread across the fields. (can be blocked or spread automatically)
Heap objects may be in either private or shared space
Titanium has a higher-level array abstraction.
These models all assume a static thread count. The HPCs languages are looking at dynamic thread counting.
The UPC compiler we're working on is an extention of an the existing Open64 compiler framework.
There's no JVM in Titanium. There's a lot of web programming built in to Java, which was not appropriate for high-performance parallel programming.

Are PGAS Languages good for multicore machines?
-They work very well on shared memory.
-Current UPC and Titanium implementations use threads.
-OpenMP gives substantial competition against PGAS on shared memory
-Unsure whether multicore processors will continue to have physical shared memory or move more towards something like a cell processor with explicit local storage.

PGAS Languages on Clusters: One-sided vs Two-sided Communication.
One-sided:
-Put/Get message can be handled directly by a network interface with RDMA support
-Some networks will now see a one-sided message and write it directly to memory without disturbing the CPU.
InfiniBand and similar networks have support for one-sided messages, ethernet does not currently.

Two-sided:
-Need to be matched with a receive to identify memory address to put data.
-Need to download match tables (from host) to interface.

One-Sided vs Two-: Practice
Half power point differs by one order of magnitude better (lower) for GASNet vs MPI.

GASNet has better latency across a network.
GASNet is at least as high (comparable) for large messages.
These numbers were calculated by comparing with MPI1, not MPI2, which does have one-sided message implementation. This implementation is not as efficient on many machines and most people use MPI1, which is why MPI1 was used for comparison purposes.
GASNet excels at mid-range message sizes, which is important for overlap, or asynchronous algorithms.

Making PGAS Real: Applications and Portability
AMR: Adaptive Mesh Refinement code.
Titanium AMR is entirely emplimented in Titanium and allows for finer-grained communication. Leads to 10x reduction in lines of code.

Beats Chombo code, the AMR package in LBNL.
The array abstraction in Titanium not only used in AMR--can be used across a wide array of applications.

In Serial, Titanium is comparable to (within a few % of) C++/Fortran.

Dense and Sparse Matrix Factorization:
As you break apart and factor a matrix, the dependant factors change on the fly. Especially with sparse matricies, dependencies change dramatically.
UPC factorization uses a highly multithreaded style, used to mask latency and dependence delays.
Three levels of threads: Static UPC threads, multithreaded BLAS, user-level threads with explicit yield.
No dynamic load balancing, but lots of remote invocation
Layout is fixed and tuned for block size
Many hard problem in here. Block size tuning for both locality and granularity. Task prioritization. et al.

UPC is significantly faster than ScaLAPACK due to the multithreadding to hide latency and dependence delays.

Most PGAS symbolic computing applications are numeric.
The applications all require:
Complex, irregular shared data structures
Ability to communicate and share data asynchronously
-Many current implementations built off of one-sided communication
Fast, low-overhead communication/sharing

Titanium and UPC are qiute portable.
-Beneath the implementation (which goes to C) is a common communication layer (GASNet), also used by gcc/upc.
-Both run on most PCs, SMPs, clusters, and supercomputers.
-Several compilers for Titanium and UPC.

PGAS Languages can easily go between shared and distributed memory machines.

Many persons are currently working on dynamic parallel environments (non-static thread counts).
Provide control over locallity and SPMD.

Languages with exceptions are still a major headache. Analysis is ongoing.
GASNet is the common framework between the PGAS languages. It can be used as a common framework for parallel implementation of your own work.
Both UPC and Titanium are working on being able to better support the ability to distinguish within a node and between nodes.

Programming Models for Parallel Computing

Katherine Yelick

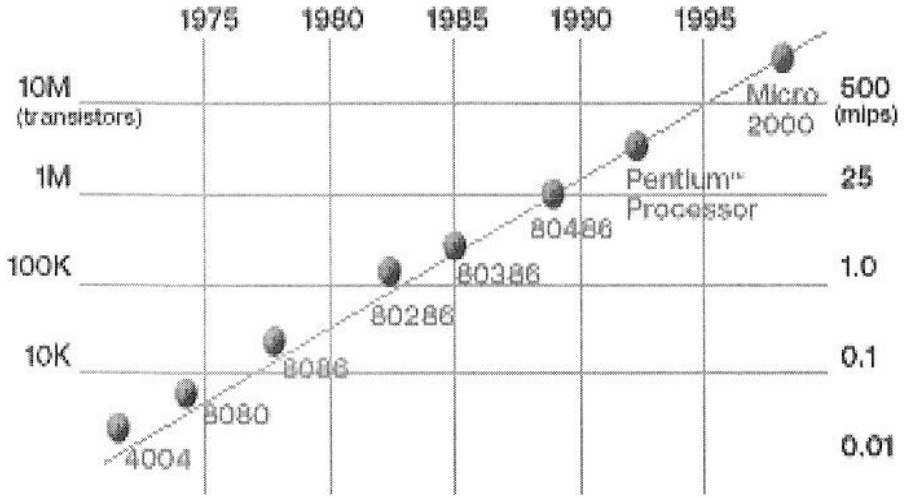
U.C. Berkeley and Lawrence Berkeley National Lab

http://titanium.cs.berkeley.edu
http://upc.lbl.gov

Parallel Computing Past

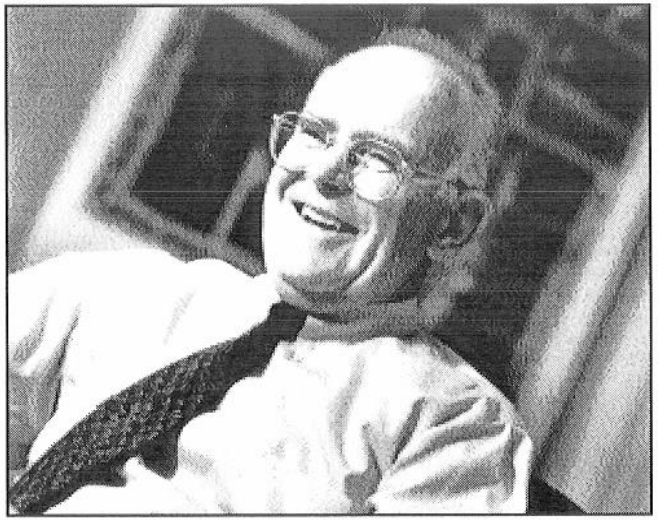
- **Not long ago, the viability of parallel computing was questioned:**
  - Several panels titled "Is parallel processing dead?"
  - "On several recent occasions, I have been asked whether parallel computing will soon be relegated to the trash heap reserved for promising technologies that never quite make it."
    - Ken Kennedy, CRPC Directory, 1994
- **But then again, there's a history of tunnel vision**
  - "I think there is a world market for maybe five computers."
    - Thomas Watson, chairman of IBM, 1943.
  - "There is no reason for any individual to have a computer in their home"
    - Ken Olson, president and founder of Digital Equipment Corporation, 1977.
  - "640K [of memory] ought to be enough for anybody."
    - Bill Gates, chairman of Microsoft, 1981.

Moore's Law is Alive and Well



2X transistors/Chip Every 1.5 years
Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.



Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

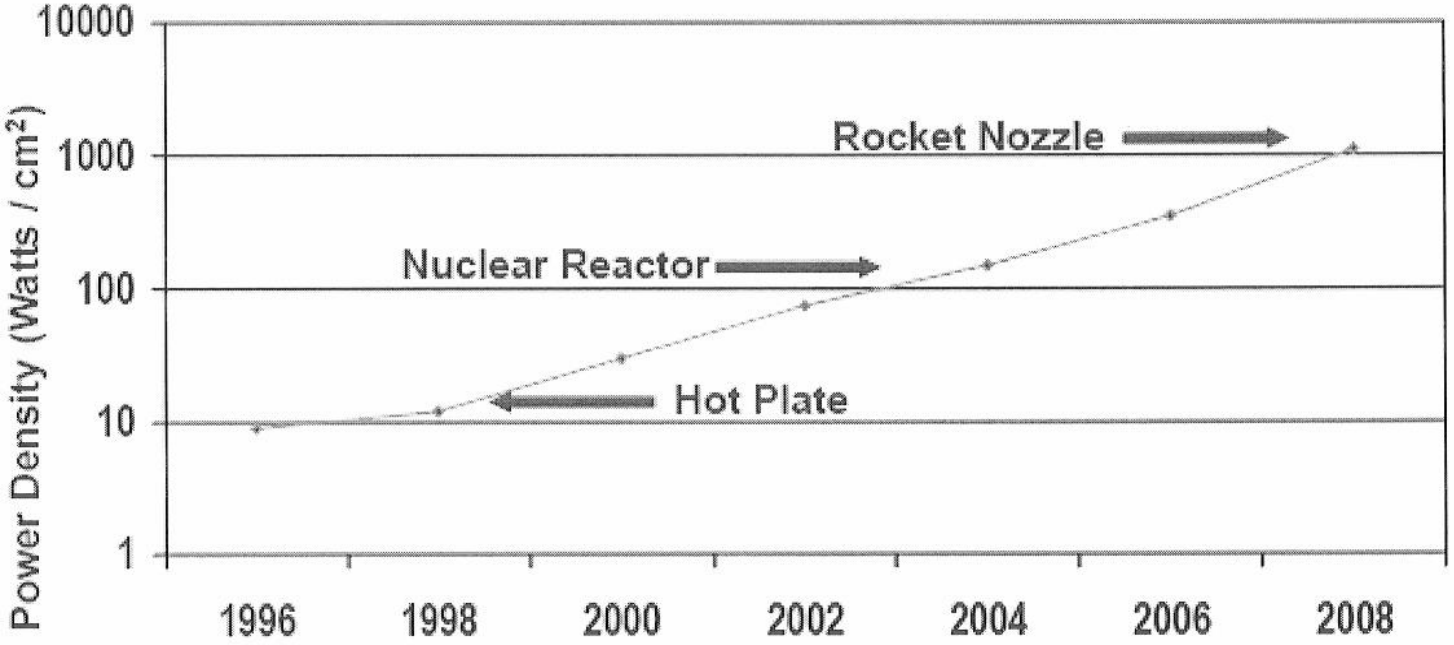
But Clock Scaling Bonanza Has Ended

- **Processor designers are forced to go "multicore" due to**
  - Heat density: faster clock means hotter chips
    - more cores with lower clock rates burn less power
  - Declining benefits of "hidden" Instruction Level Parallelism (ILP)
    - Last generation of single core chips probably over-engineered
    - Lots of logic/power to find ILP parallelism, but it wasn't in the apps
  - Yield problems
    - Parallelism can also be used for redundancy
    - IBM Cell processor has 8 small cores; a blade system with all 8 sells for $20K, whereas a PS3 is about $600 and only uses 7

Power Density Limits Serial Performance

Clock Scaling Extrapolation:

Power Density for Leading Edge Microprocessors



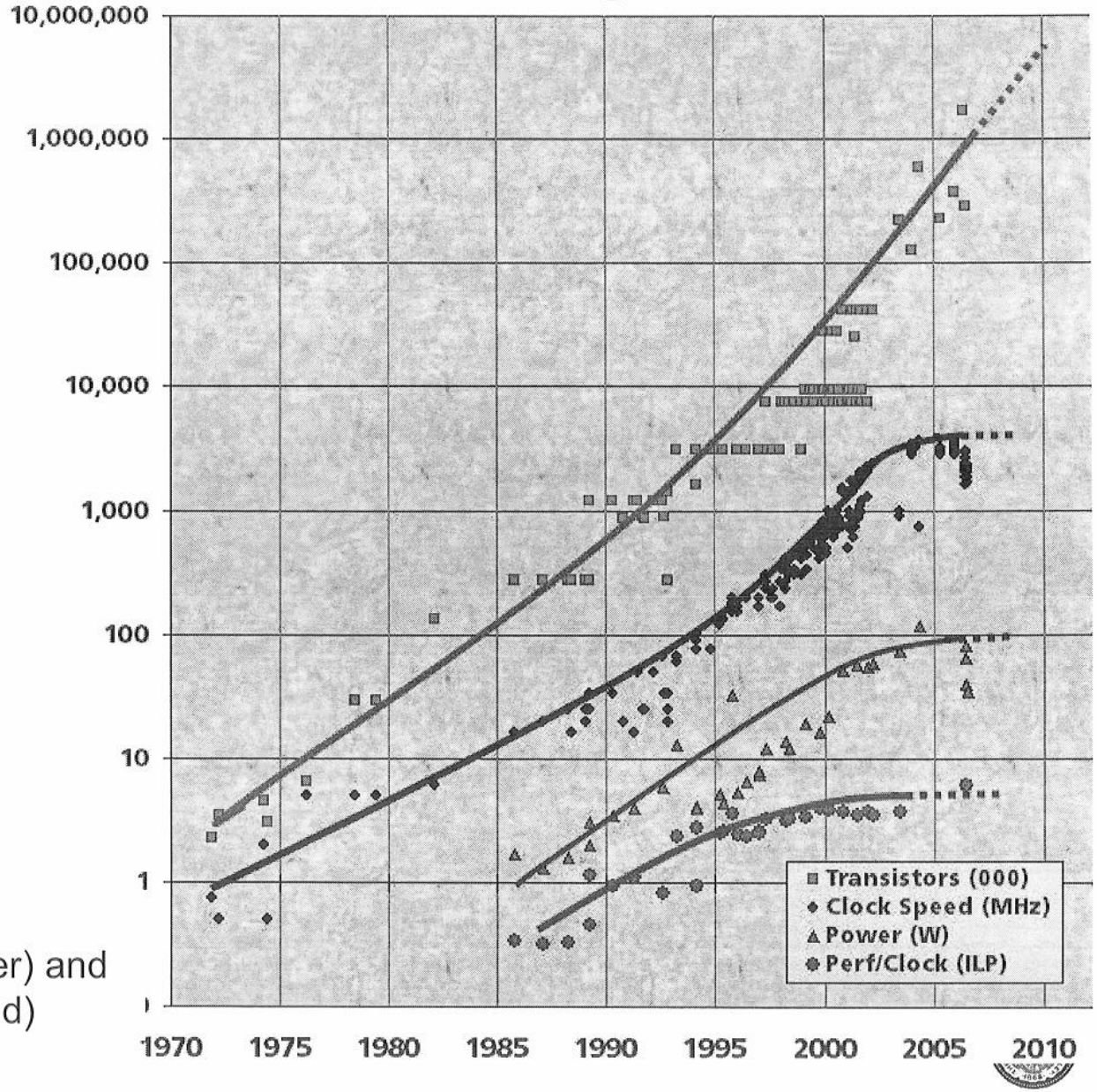
Power Density Becomes Too High to Cool Chips Inexpensively

Source: Shekhar Borkar, Intel Corp

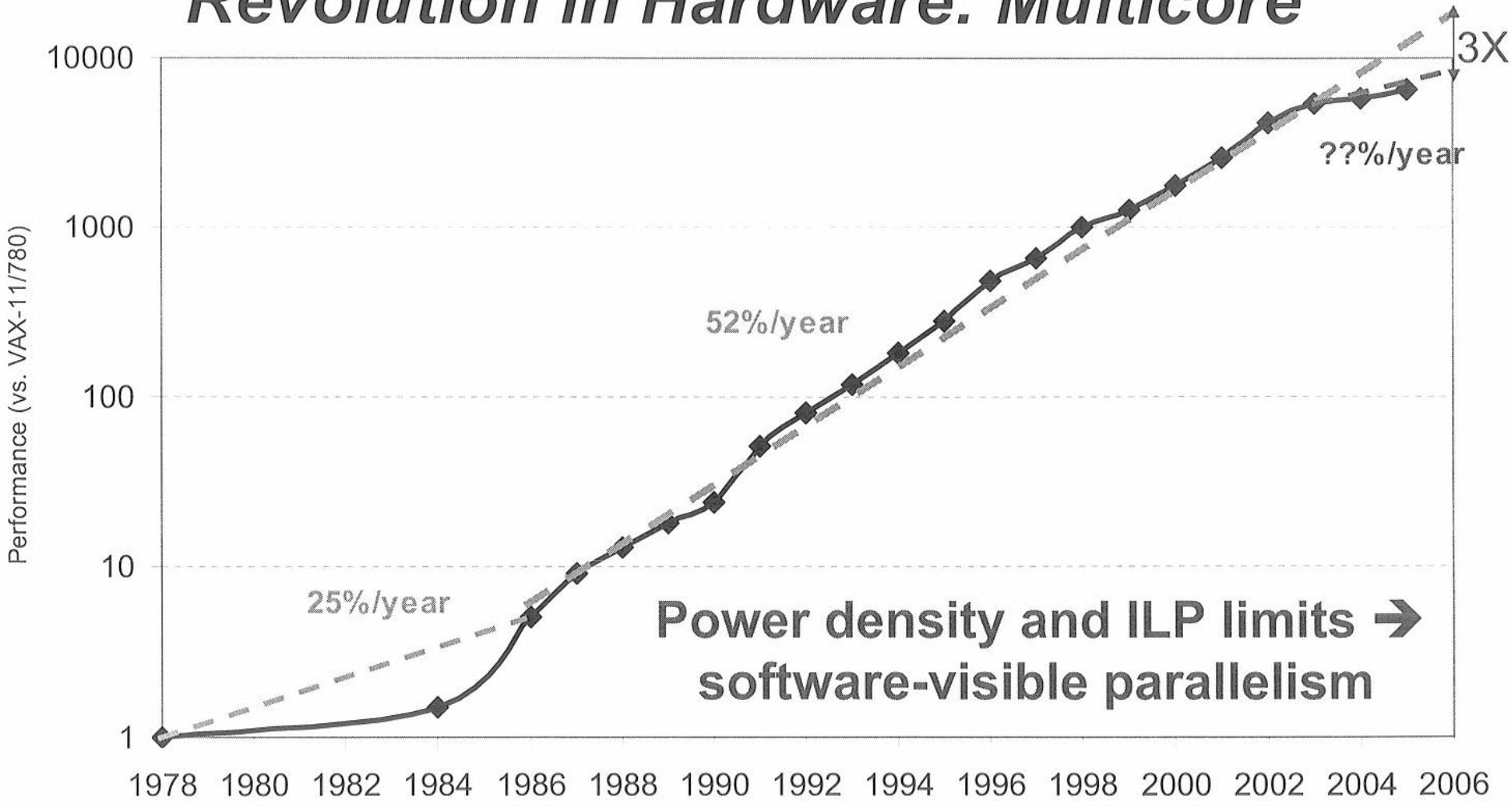
Revolution is Happening Now

- **Chip density is continuing increase ~2x every 2 years**
  - Clock speed is not
  - Number of processor cores may double instead
- **There is little or no hidden parallelism (ILP) to be found**
- **Parallelism must be exposed to and managed by software**

Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)



Revolution in Hardware: Multicore



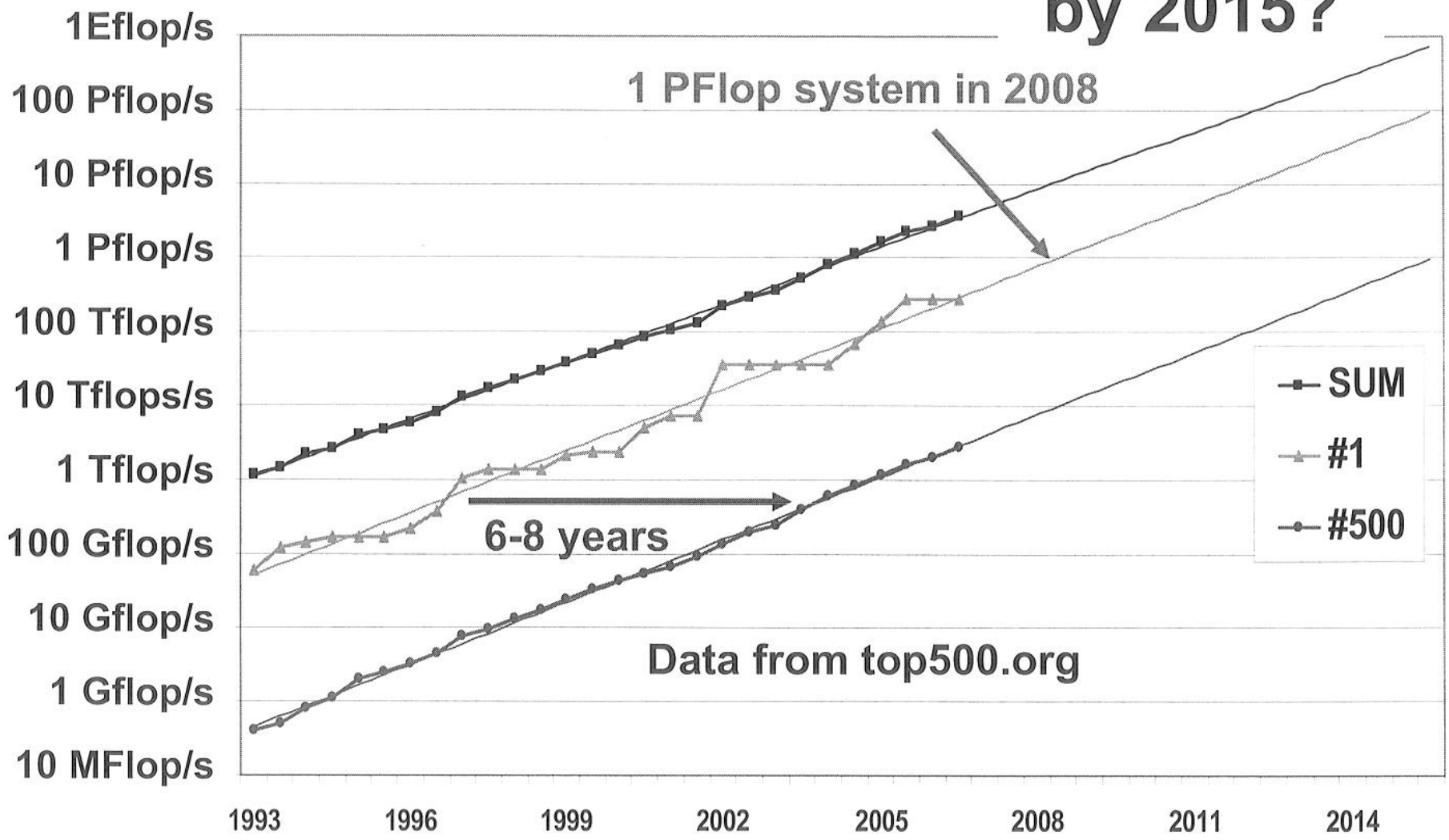
From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, Sept. 15, 2006

Why Parallelism (2007)?

- **These arguments are no longer theoretical**
- **All major processor vendors are producing multicore chips**
  - Every machine will soon be a parallel machine
  - All programmers will be parallel programmers???
- **New software model**
  - Want a new feature? Hide the "cost" by speeding up the code first
  - All programmers will be performance programmers???
- **Some may eventually be hidden in libraries, compilers, and high level languages**
  - But a lot of work is needed to get there
- **Big open questions:**
  - What will be the killer apps for multicore machines?
  - How should the chips be designed: multicore, manycore, heterogenous?
  - How will they be programmed?

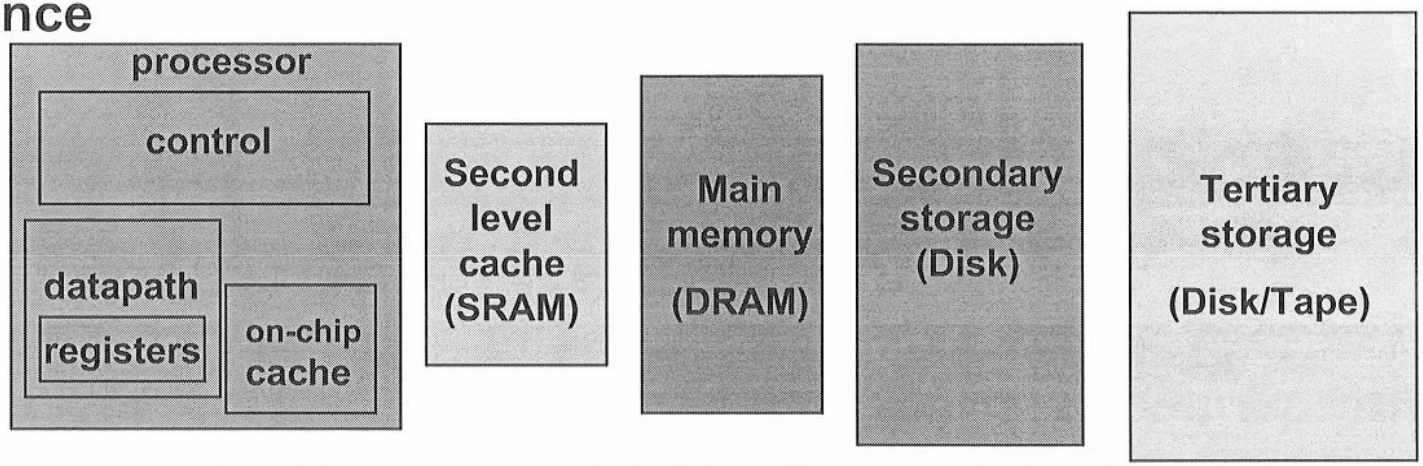
Petaflop with ~1M Cores

Common by 2015?



Memory Hierarchy

- With explicit parallelism, performance becomes a software problem
- Parallelism is not the only way to get performance; locality is at least as important
- And this problem is growing, as off-chip latencies are relatively flat (about 7% improvement per year) compared to processor performance



Speed	1ns	10ns	100ns	10ms	10sec
Size	B	KB	MB	GB	TB

Predictions

- **Parallelism will explode**
  - Number of cores will double every 12-24 months
  - Petaflop (million processor) machines will be common in HPC by 2015 (all top 500 machines will have this)
- **Performance will become a software problem**
  - Parallelism and locality are key will be concerns for many programmers – not just an HPC problem
- **A new programming model will emerge for multicore programming**
  - Can one programming model (not necessarily one language) cover games, laptops, and top500 space?

PGAS Languages: What, Why, and How

Parallel Programming Models

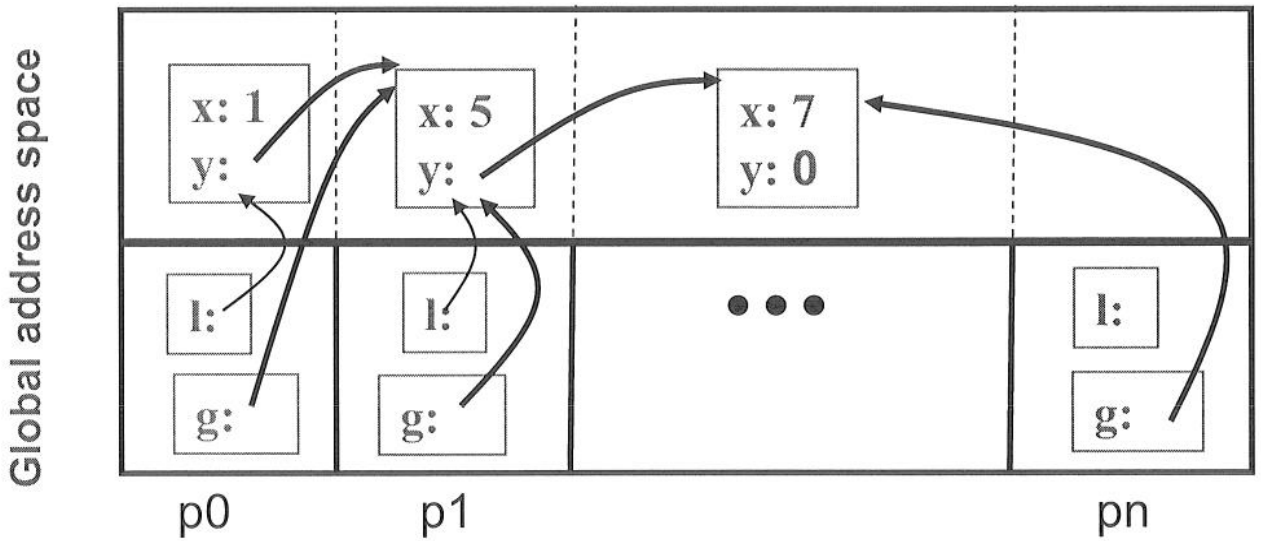
- Parallel software is still an unsolved problem !
- **Most parallel programs are written using either:**
  - Message passing with a SPMD model
    - for scientific applications; scales easily
  - Shared memory with threads in OpenMP, Threads, or Java
    - non-scientific applications; easier to program
- **Partitioned Global Address Space (PGAS) Languages**
  - global address space like threads (programmability)
  - SPMD parallelism like MPI (performance)
  - local/global distinction, i.e., layout matters (performance)

Partitioned Global Address Space Languages

- **Explicitly-parallel programming model with SPMD parallelism**
  - Fixed at program start-up, typically 1 thread per processor
- **Global address space model of memory**
  - Allows programmer to directly represent distributed data structures
- **Address space is logically partitioned**
  - Local vs. remote memory (two-level hierarchy)
- **Programmer control over performance critical decisions**
  - Data layout and communication
- **Performance transparency and tunability are goals**
  - Initial implementation can use fine-grained shared memory
- **Base languages UPC (C), CAF (Fortran), Titanium (Java)**
- **New HPCS languages have similar data model, but dynamic multithreading**

Partitioned Global Address Space

- *Global address space:* any thread/process may directly read/write data allocated by another
- *Partitioned:* data is designated as local or global



By default:

- Object heaps are shared
- Program stacks are private

- *3 Current languages:* UPC, CAF, and Titanium
  - All three use an SPMD execution model
  - Emphasis in this talk on UPC and Titanium (based on Java)
- *3 Emerging languages:* X10, Fortress, and Chapel

PGAS Language Overview

- **Many common concepts, although specifics differ**
  - Consistent with base language, e.g., Titanium is strongly typed
- **Both private and shared data**
  - **int x[10];** and **shared int y[10];**
- **Support for distributed data structures**
  - Distributed arrays; local and global pointers/references
- **One-sided shared-memory communication**
  - Simple assignment statements: **x[i] = y[i];** or **t = *p;**
  - Bulk operations: memcpy in UPC, array ops in Titanium and CAF
- **Synchronization**
  - Global barriers, locks, memory fences
- **Collective Communication, IO libraries, etc.**

Private vs. Shared Variables in UPC

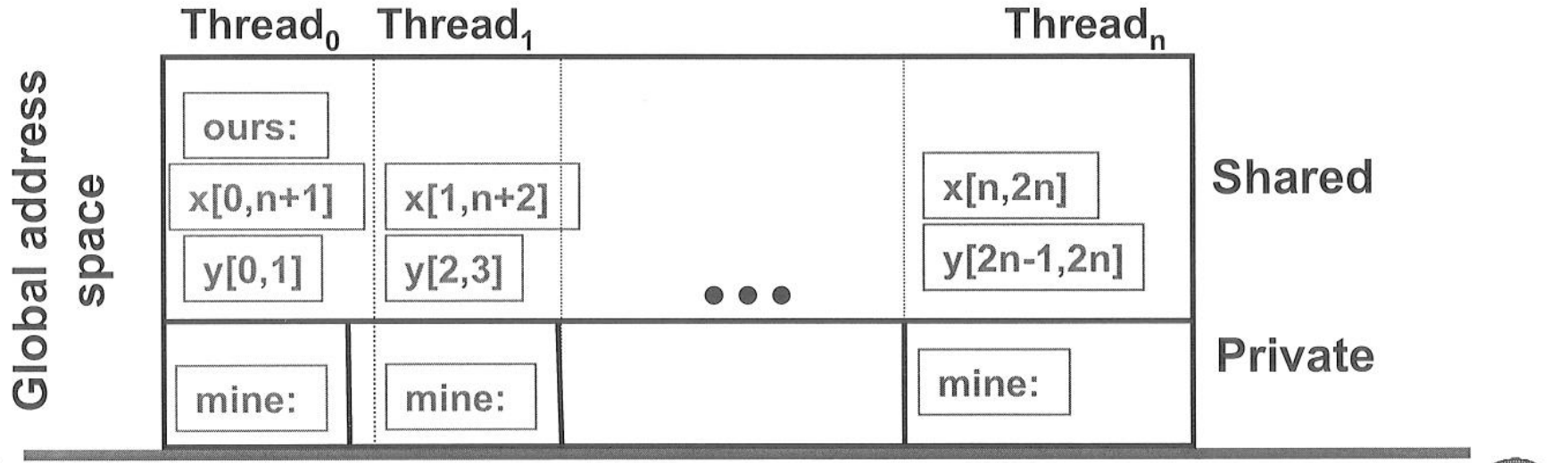
- C variables and objects are allocated in the private memory space
- Shared variables are allocated only once, in thread 0's space

```
shared int ours;
int mine;
```

- Shared arrays are spread across the threads

```
shared int x[2*THREADS]     /* cyclic, 1 element each, wrapped */
shared int [2] y [2*THREADS] /* blocked, with block size 2 */
```

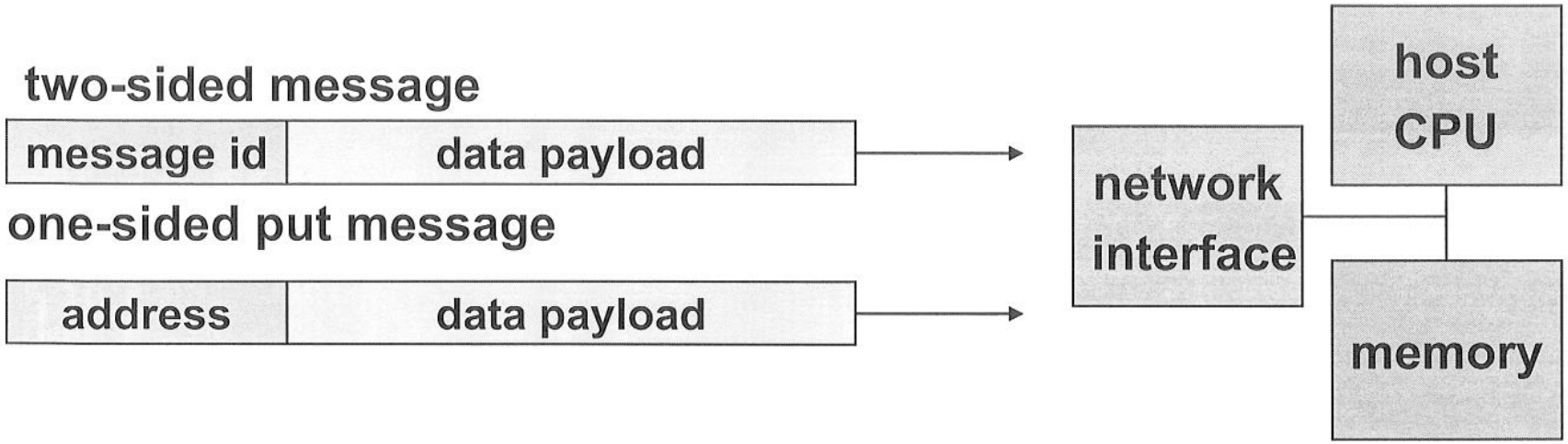
- Heap objects may be in either private or shared space



PGAS Language for Multicore

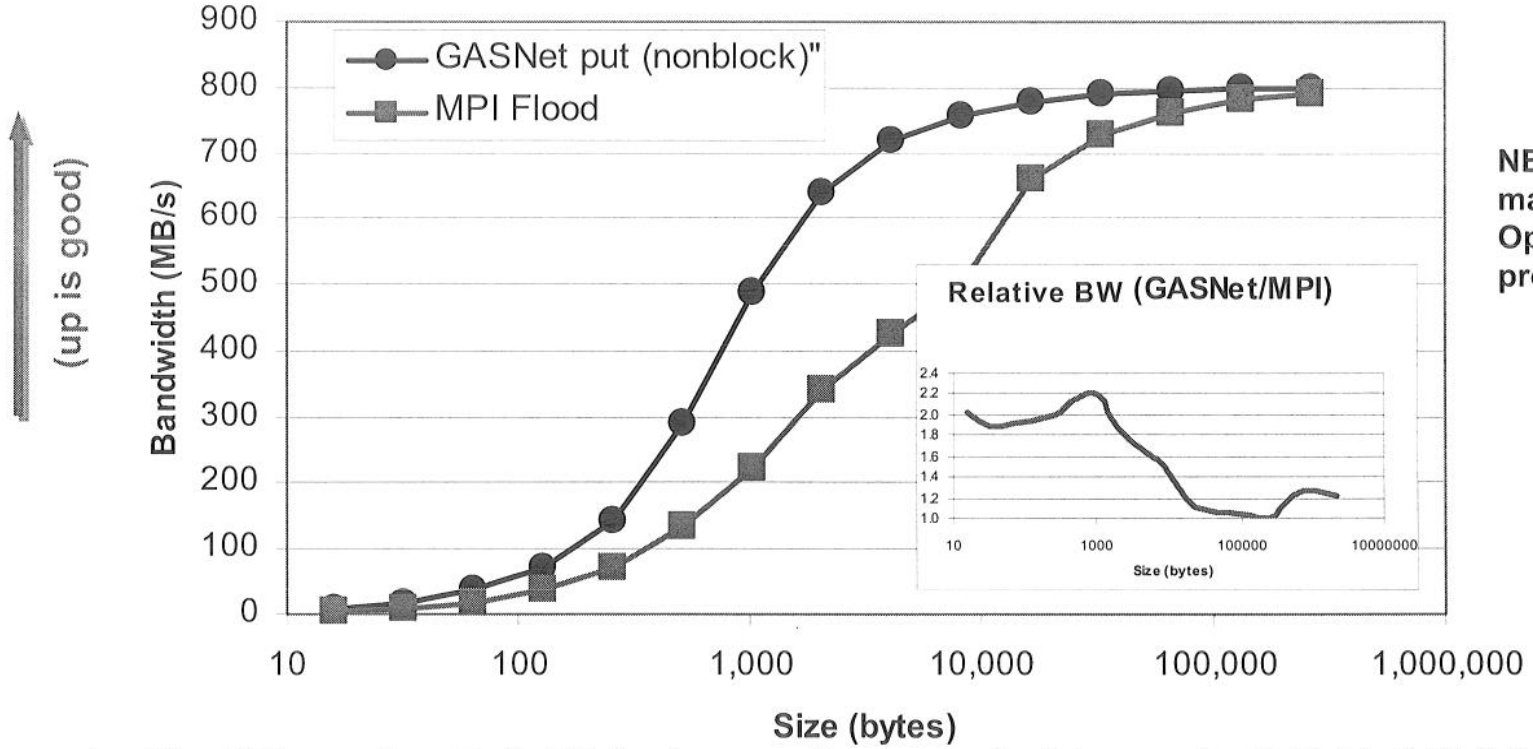
- **PGAS languages are a good fit to shared memory machines**
  - Global address space implemented as reads/writes
  - Current UPC and Titanium implementation uses threads
  - Working on System V shared memory for UPC
- **"Competition" on shared memory is OpenMP**
  - PGAS has locality information that may be important when we get to >100 cores per chip
  - Also may be exploited for processor with explicit local store rather than cache, e.g., Cell processor
  - SPMD model in current PGAS languages is both an advantage (for performance) and constraining

PGAS Languages on Clusters: One-Sided vs Two-Sided Communication



- **A one-sided put/get message can be handled directly by a network interface with RDMA support**
  - Avoid interrupting the CPU or storing data from CPU (preposts)
- **A two-sided messages needs to be matched with a receive to identify memory address to put data**
  - Offloaded to Network Interface in networks like Quadrics
  - Need to download match tables to interface (from host)

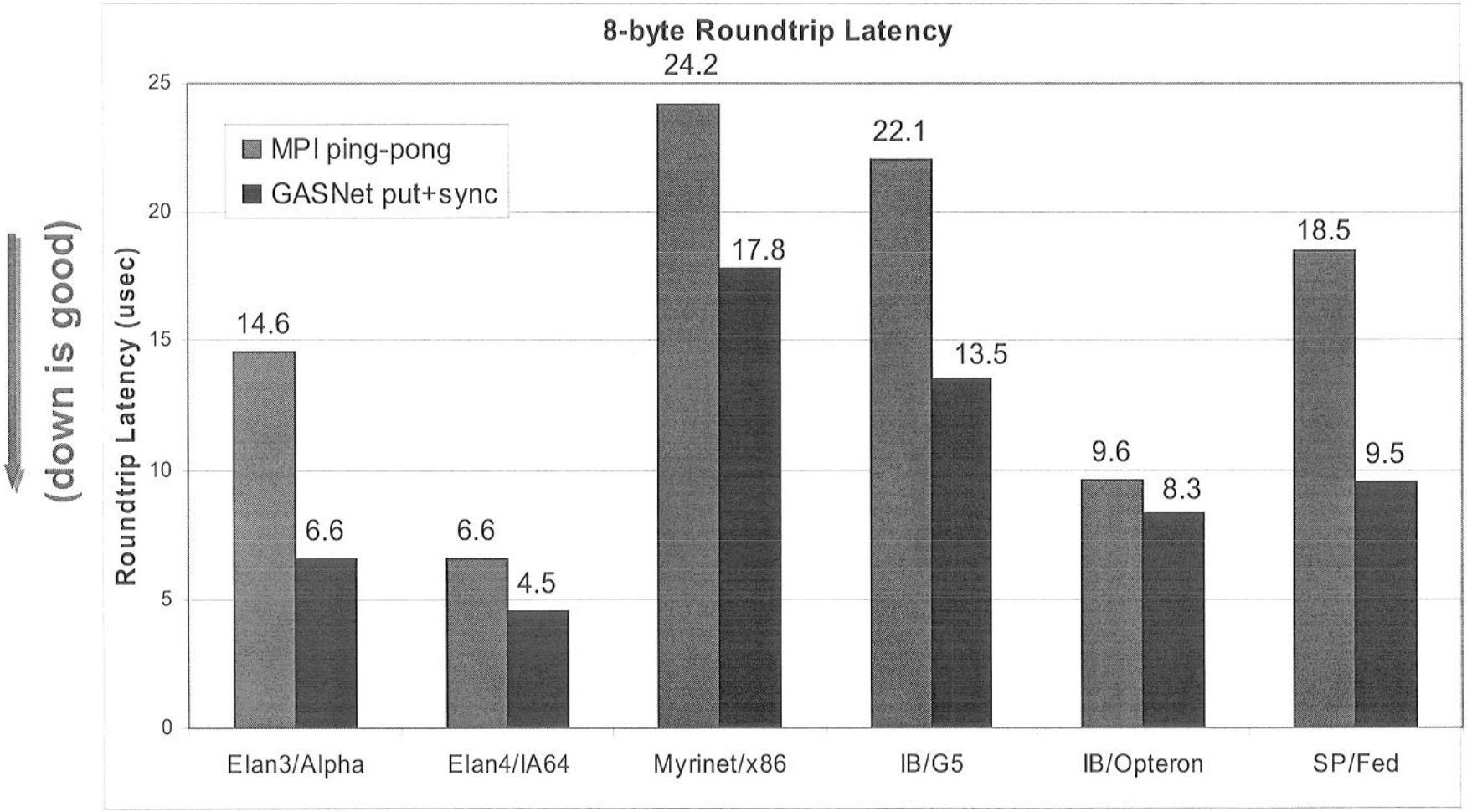
One-Sided vs. Two-Sided: Practice



NERSC Jacquard machine with Opteron processors

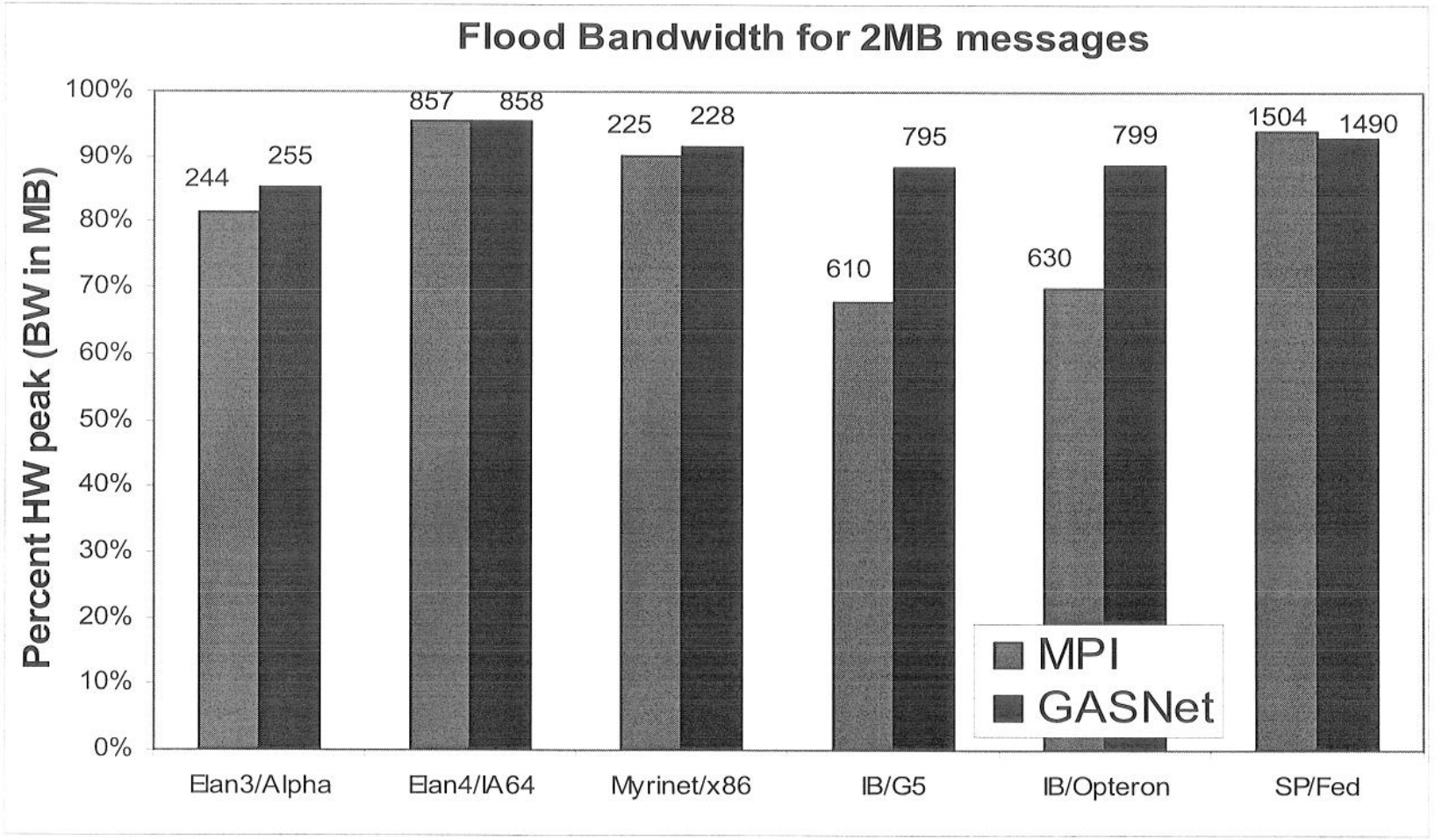
- InfiniBand: GASNet vapi-conduit and OSU MVAPICH 0.9.5
- Half power point (N ½) differs by *one order of magnitude*
- This is not a criticism of the implementation!

GASNet: Portability and *High-Performance*



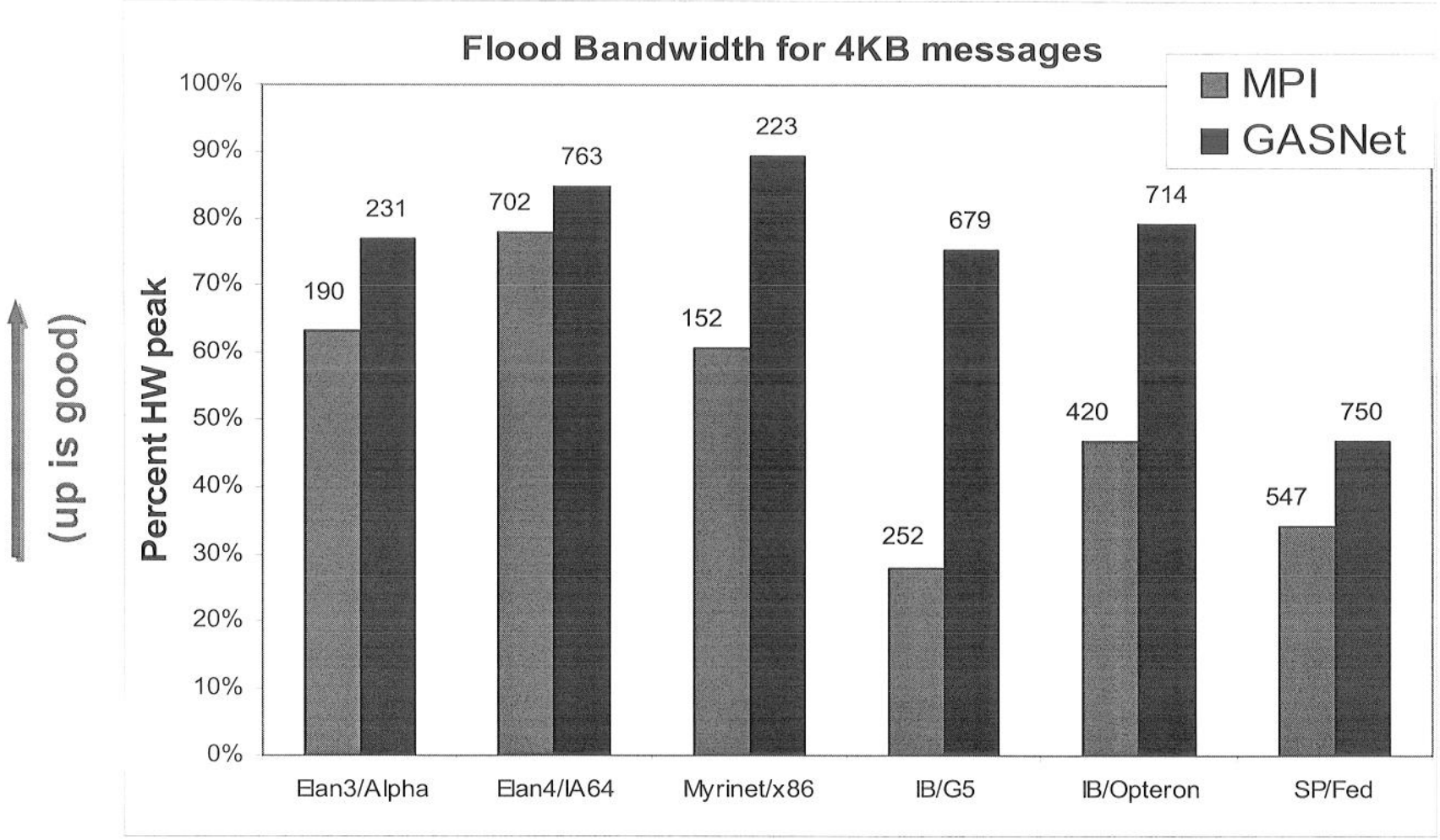
GASNet better for latency across machines

GASNet: Portability and *High-Performance*



GASNet at least as high (comparable) for large messages

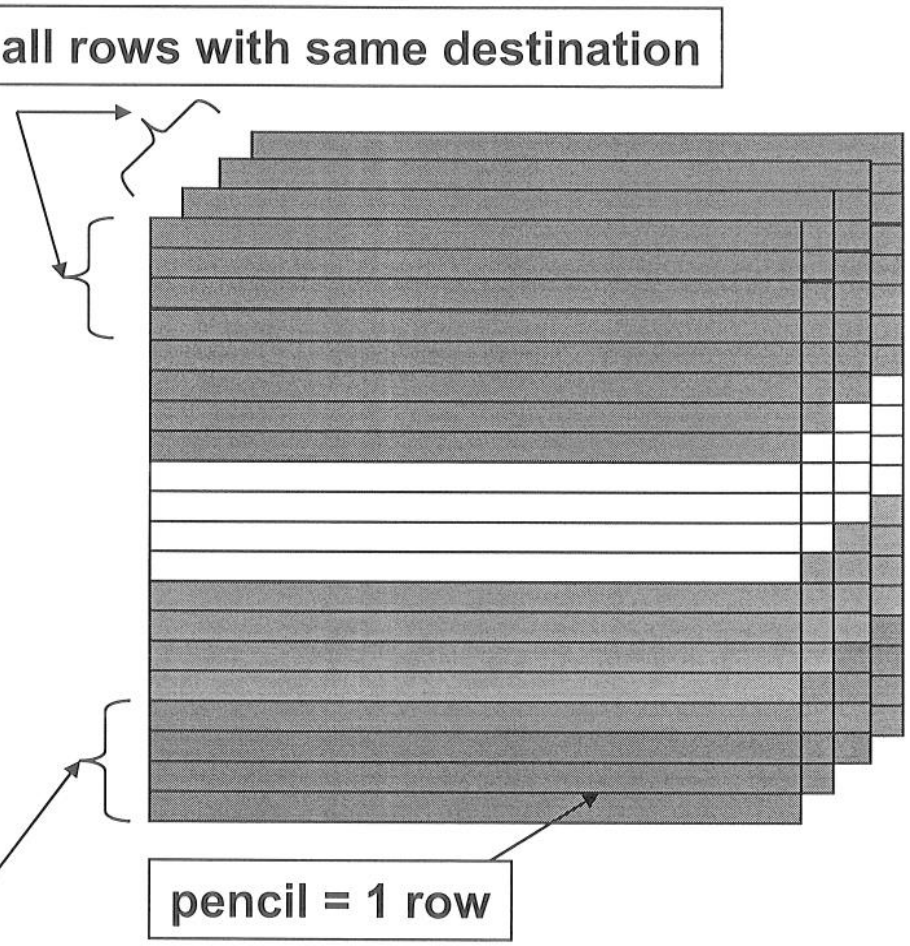
GASNet: Portability and *High-Performance*



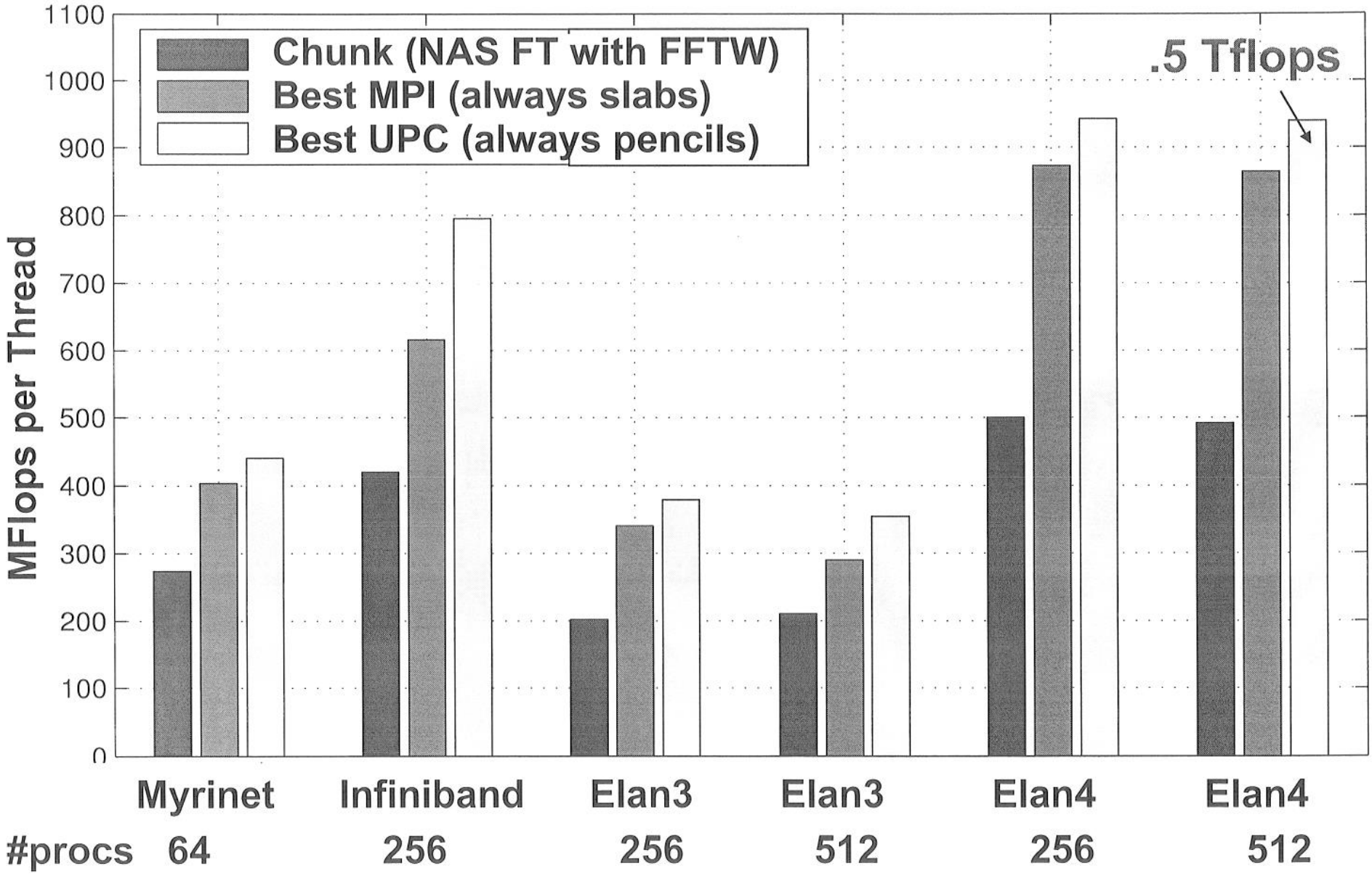
GASNet excels at mid-range sizes: important for overlap

Communication Strategies for 3D FFT

- **Three approaches:**
  - **Chunk:**
    - Wait for 2nd dim FFTs to finish
    - Minimize # messages
  - **Slab:**
    - Wait for chunk of rows destined for 1 proc to finish
    - Overlap with computation
  - **Pencil:**
    - Send each row as it completes
    - Maximize overlap and
    - Match natural layout



NAS FT Variants Performance Summary



Making PGAS Real: Applications and Portability

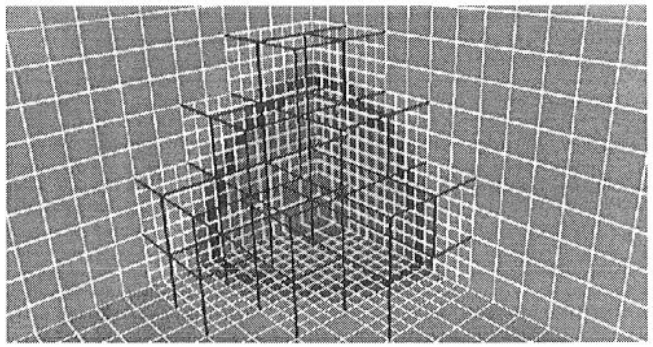
AMR in Titanium

C++/Fortran/MPI AMR

- Chombo package from LBNL
- Bulk-synchronous comm:
  - Pack boundary data between procs

Titanium AMR

- Entirely in Titanium
- Finer-grained communication
  - No explicit pack/unpack code
  - Automated in runtime system

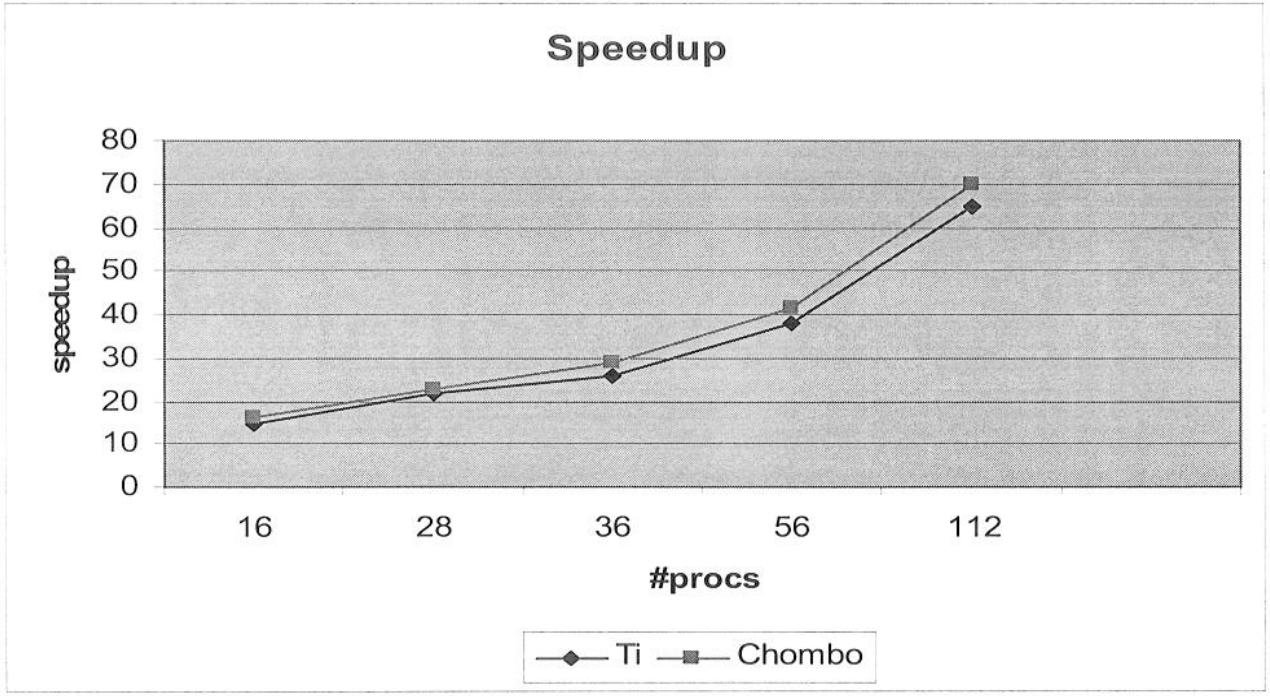


Code Size in Lines		
	C++/F/MPI	Titanium
AMR data Structures	35000	2000
AMR operations	6500	1200
Elliptic PDE solver	4200*	1500

* Somewhat more functionality in PDE part of Chombo code

10X reduction in lines of code!

Performance of Titanium AMR



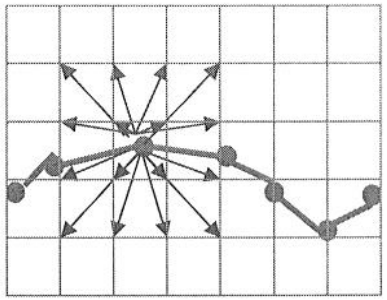
Comparable parallel performance

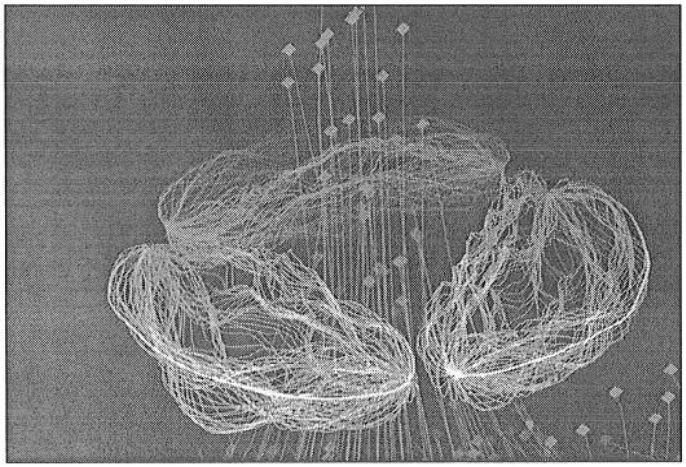
- Serial: Titanium is within a few % of C++/F; sometimes faster!
- Parallel: Titanium scaling is comparable with generic optimizations
  - optimizations (SMP-aware) that are not in MPI code
  - additional optimizations (namely overlap) not yet implemented

Particle/Mesh Method: Heart Simulation

- **Elastic structures in an incompressible fluid.**
  - Blood flow, clotting, inner ear, embryo growth, …
- **Complicated parallelization**
  - Particle/Mesh method, but "Particles" connected into materials (1D or 2D structures)
  - Communication patterns irregular between particles (structures) and mesh (fluid)

2D Dirac Delta Function

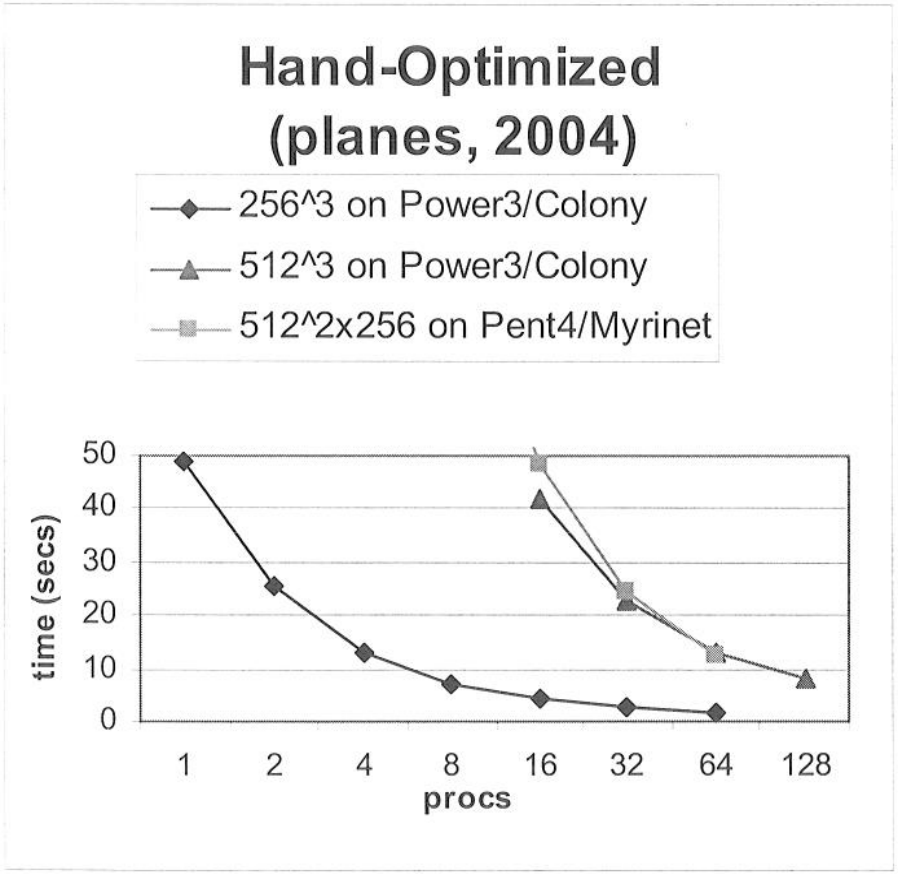


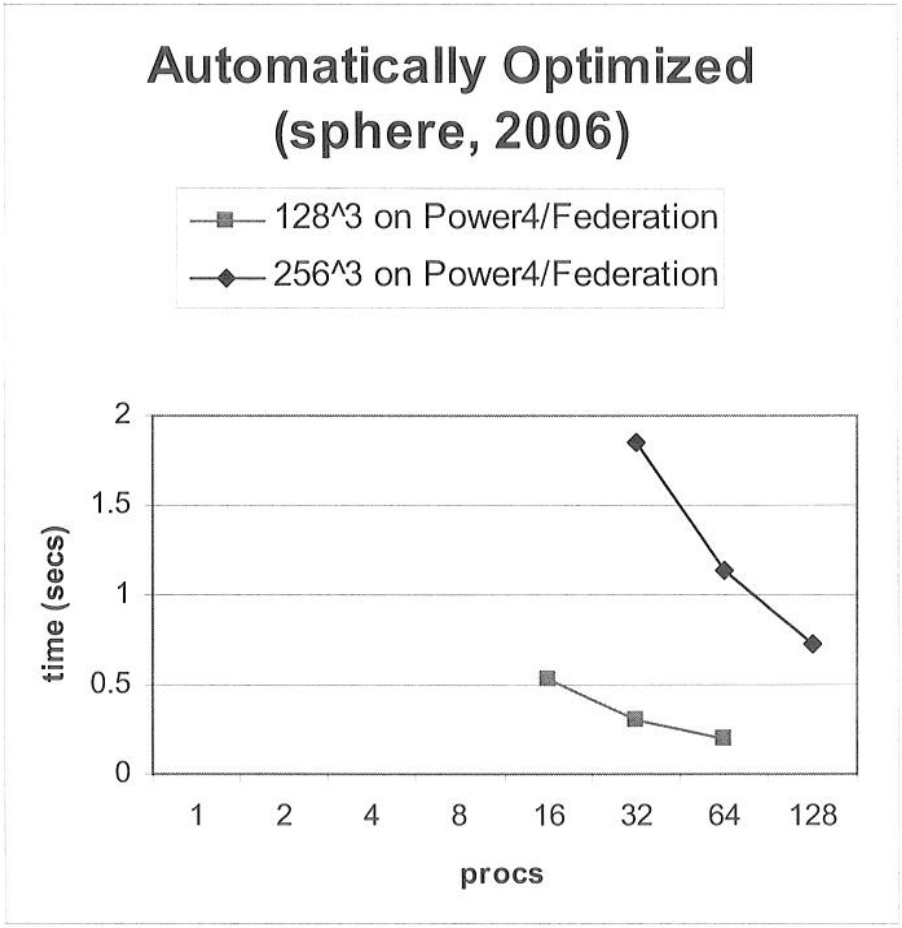


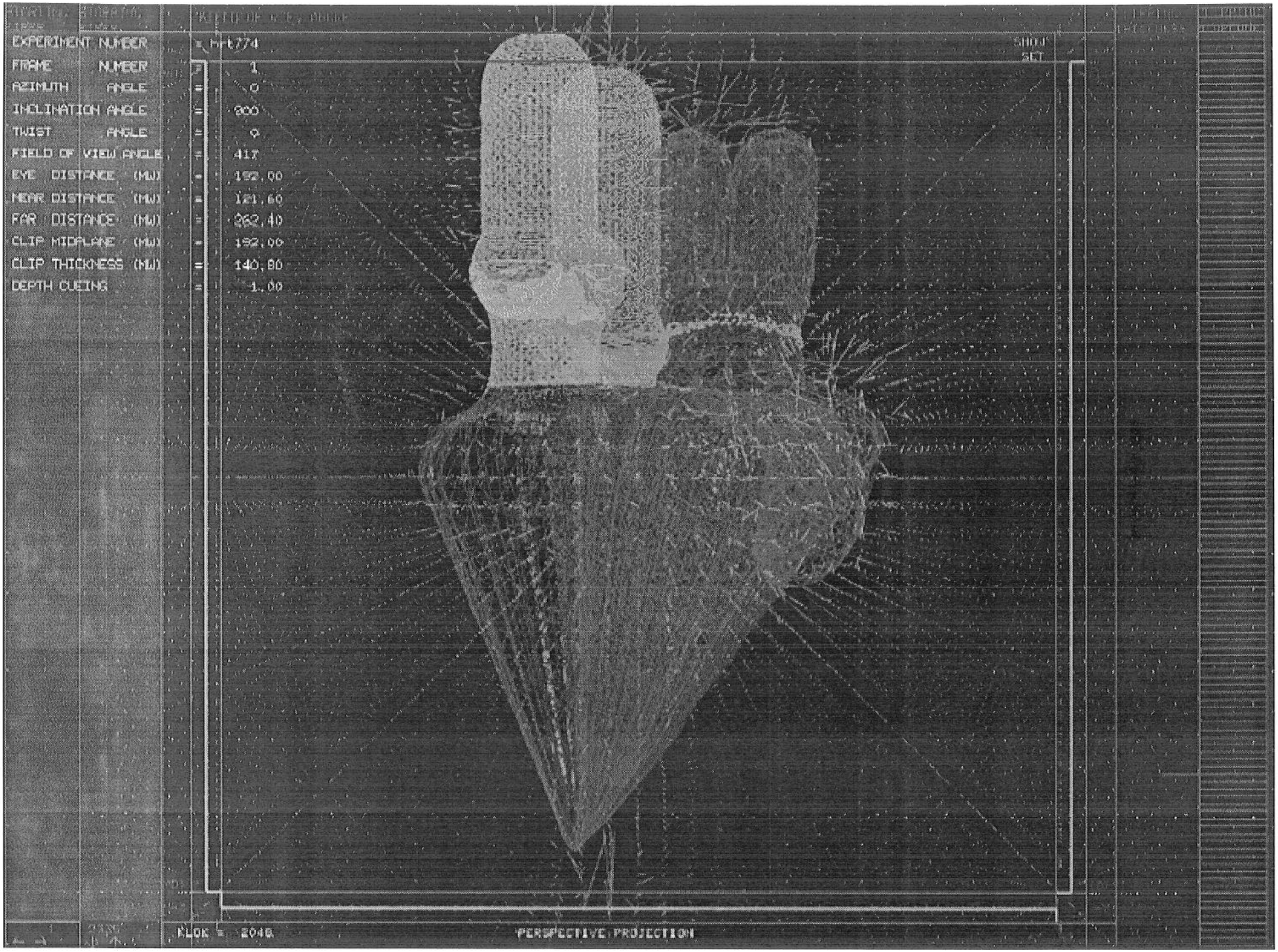
Code Size in Lines	
Fortran	Titanium
8000	4000

Note: Fortran code is not parallel

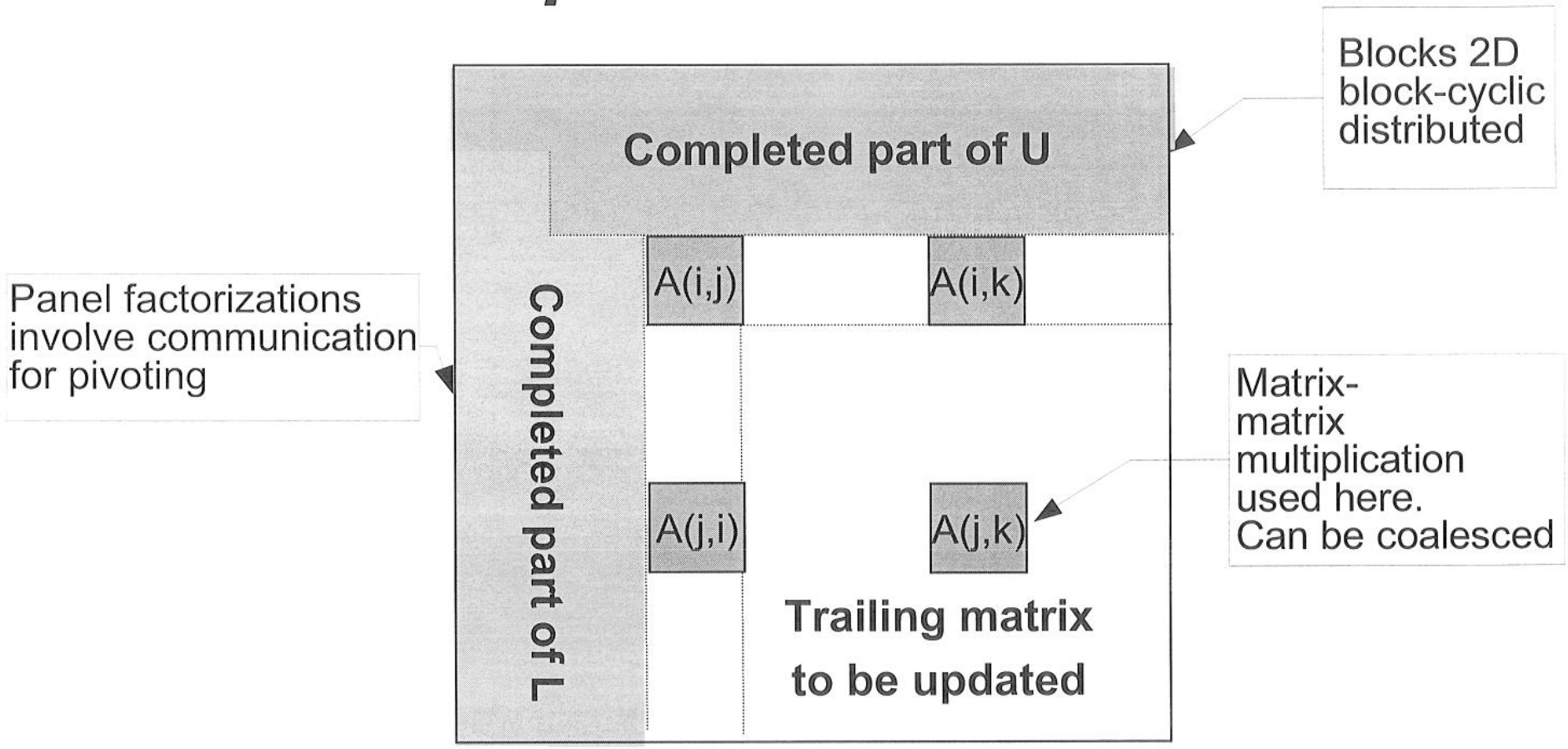
Immersed Boundary Method Performance






EXPERIMENT NUMBER = Brb774
FRAME NUMBER = 1
AZIMUTH ANGLE = 0
INCLINATION ANGLE = 900
TWIST ANGLE = 0
FIELD OF VIEW ANGLE = 417
EYE DISTANCE (MW) = 192.00
NEAR DISTANCE (MW) = 121.60
FAR DISTANCE (MW) = 262.40
CLIP MIDPLANE (MW) = 192.00
CLIP THICKNESS (MW) = 140.80
DEPTH CUEING = 1.00
SHOW SET
BLOK = 2048
PERSPECTIVE PROJECTION

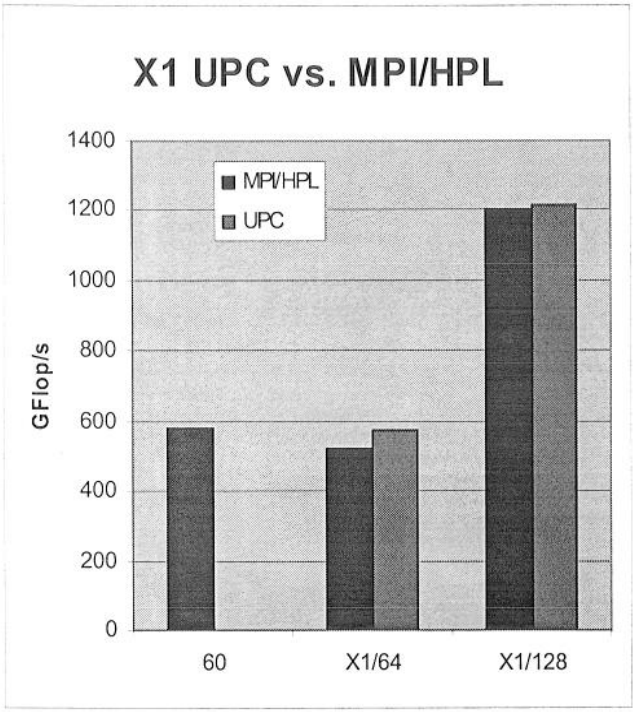
Dense and Sparse Matrix Factorization

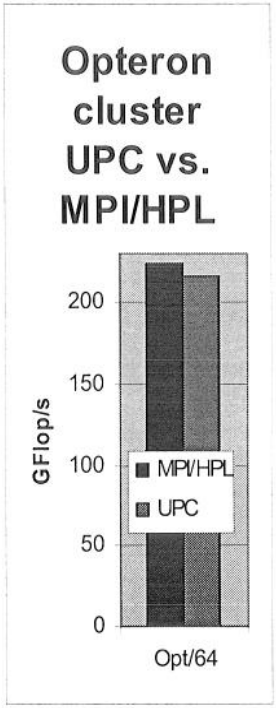


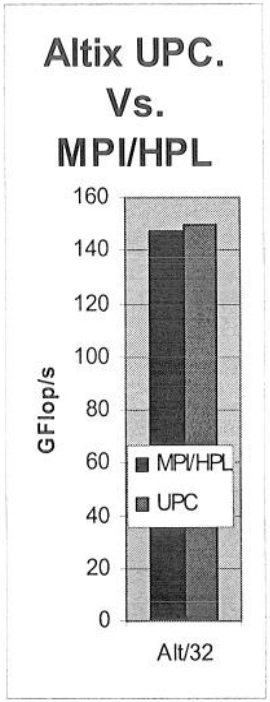
Matrix Factorization in UPC

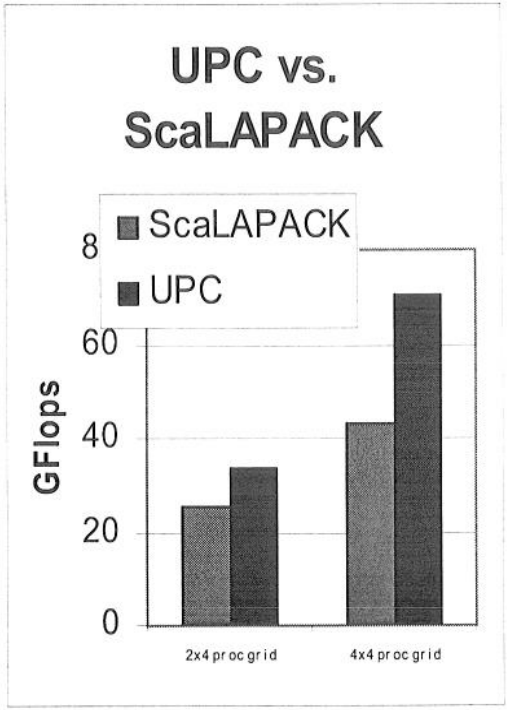
- **UPC factorization uses a highly multithreaded style**
  - Used to mask latency and to mask dependence delays
  - Three levels of threads:
    - UPC threads (data layout, each runs an event scheduling loop)
    - Multithreaded BLAS (boost efficiency)
    - User level (non-preemptive) threads with explicit yield
  - No dynamic load balancing, but lots of remote invocation
  - Layout is fixed (blocked/cyclic) and tuned for block size
- **Same framework being used for sparse Cholesky**
- **Hard problems**
  - Block size tuning (tedious) for both locality and granularity
  - Task prioritization (ensure critical path performance)
  - Resource management can deadlock memory allocator if not careful

UPC HP Linpack Performance









- Comparable to MPI HPL (numbers from HPCC database)
- Faster than ScaLAPACK due to less synchronization
- Large scaling of UPC code on Itanium/Quadrics (Thunder)
  - 2.2 TFlops on 512p and 4.4 TFlops on 1024p

PGAS Languages and Symbolic Computing

- **Most of these applications are numeric**
- **Experience in parallel symbolic computing**
  - Grobner basis completion procedure [CAD 92, PPoPP 93, RTA 93]
  - Compiling Verilog [IVC 95]
  - The Perfect Phylogeny Problem [Supercomputing 95]
  - Connected components
  - Mesh generation
- **What do these applications require?**
  - Complex, irregular shared data structures
    - Not just distributed arrays
  - Ability to communicate/share data asynchronously
    - Not bulk-synchronous; not two-sided messaging
  - Fast low-overhead communication/sharing
    - Shared memory is ideal, remote procedure invocation useful

Portability of Titanium and UPC

- **Titanium and the Berkeley UPC translator use a similar model**
  - Source-to-source translator (generate ISO C)
  - Runtime layer implements global pointers, etc
  - Common communication layer (GASNet)

  } Also used by gcc/upc
- **Both run on most PCs, SMPs, clusters & supercomputers**
  - Operating Systems:
    - Linux, FreeBSD, Tru64, AIX, IRIX, HPUX, Solaris, Cygwin, MacOSX, Unicos, SuperUX
  - Supported CPUs:
    - x86, Itanium, Alpha, Sparc, PowerPC, PA-RISC, Opteron
  - GASNet communication:
    - Myrinet, Quadrics, Infiniband, IBM LAPI, Cray X1, SGI Altix, SHMEM, MPI and UDP
  - Specific platforms:
    - HP AlphaServer, Cray X1, IBM SP, NEC SX-6, Cluster X (Big Mac), SGI Altix 3000
    - Underway: Cray XT3, BG/L (both run over MPI)
- **Can be mixed with MPI, C/C++, Fortran**
- **Several other compilers for UPC: HP, Cray, MTU, Intrepid, IBM**

Conclusions

- **Parallel computing is the future**
  - Time to think about parallelization strategies; think in long term towards machine trends
  - Best time ever for a new parallel language
- **PGAS Languages**
  - Good fit for shared and distributed memory
  - Control over locality and (for better or worse) SPMD
  - Support needs of symbolic and numeric communities
  - Offer incremental parallelism
- **Available for download**
  - Berkeley UPC compiler: http://upc.lbl.gov
  - Titanium compiler: http://titanium.cs.berkeley.edu