Programming Models for Parallel Computing

Katherine Yelick

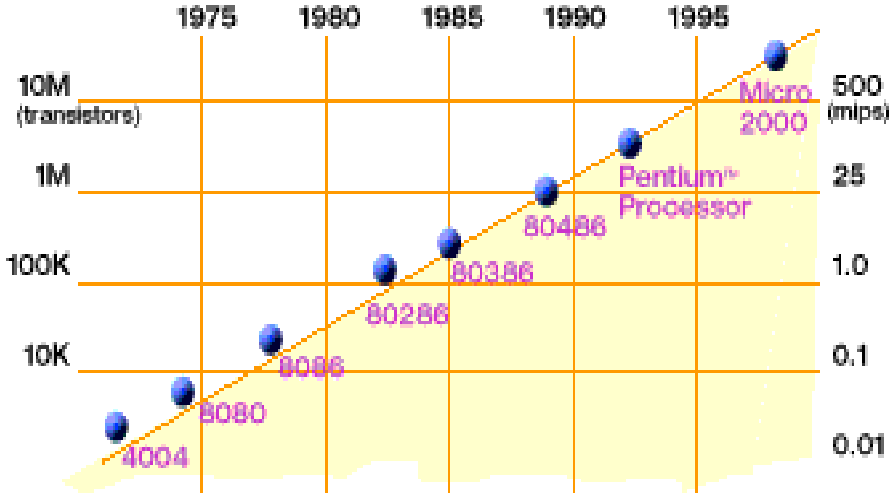
U.C. Berkeley and Lawrence Berkeley National Lab

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

Parallel Computing Past

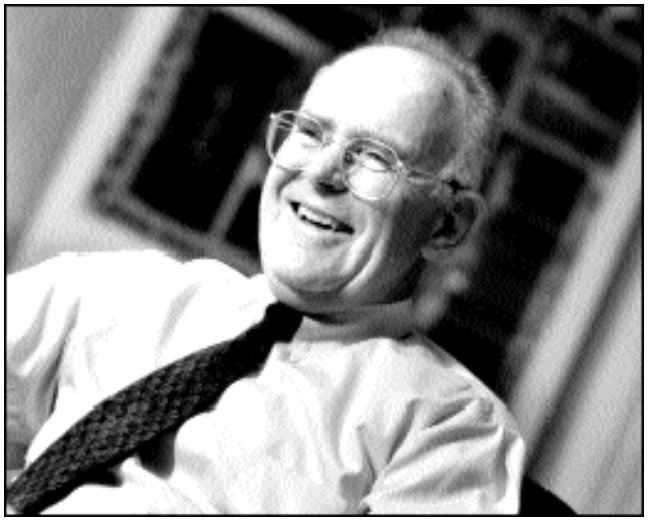
- **Not long ago, the viability of parallel computing was questioned:**
  - Several panels titled “Is parallel processing dead?”
  - “On several recent occasions, I have been asked whether parallel computing will soon be relegated to the trash heap reserved for promising technologies that never quite 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.

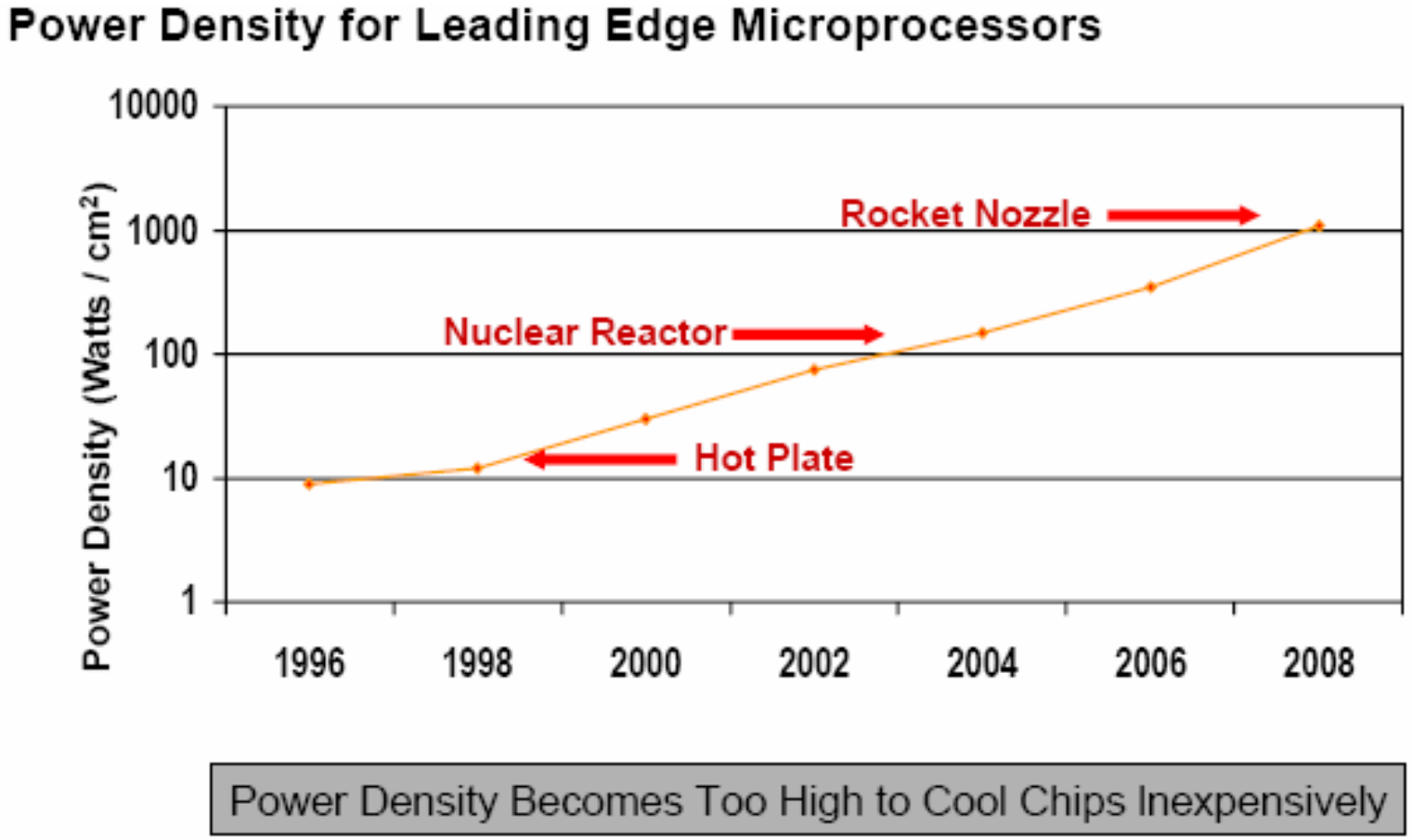
Slide source: Jack Dongarra

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

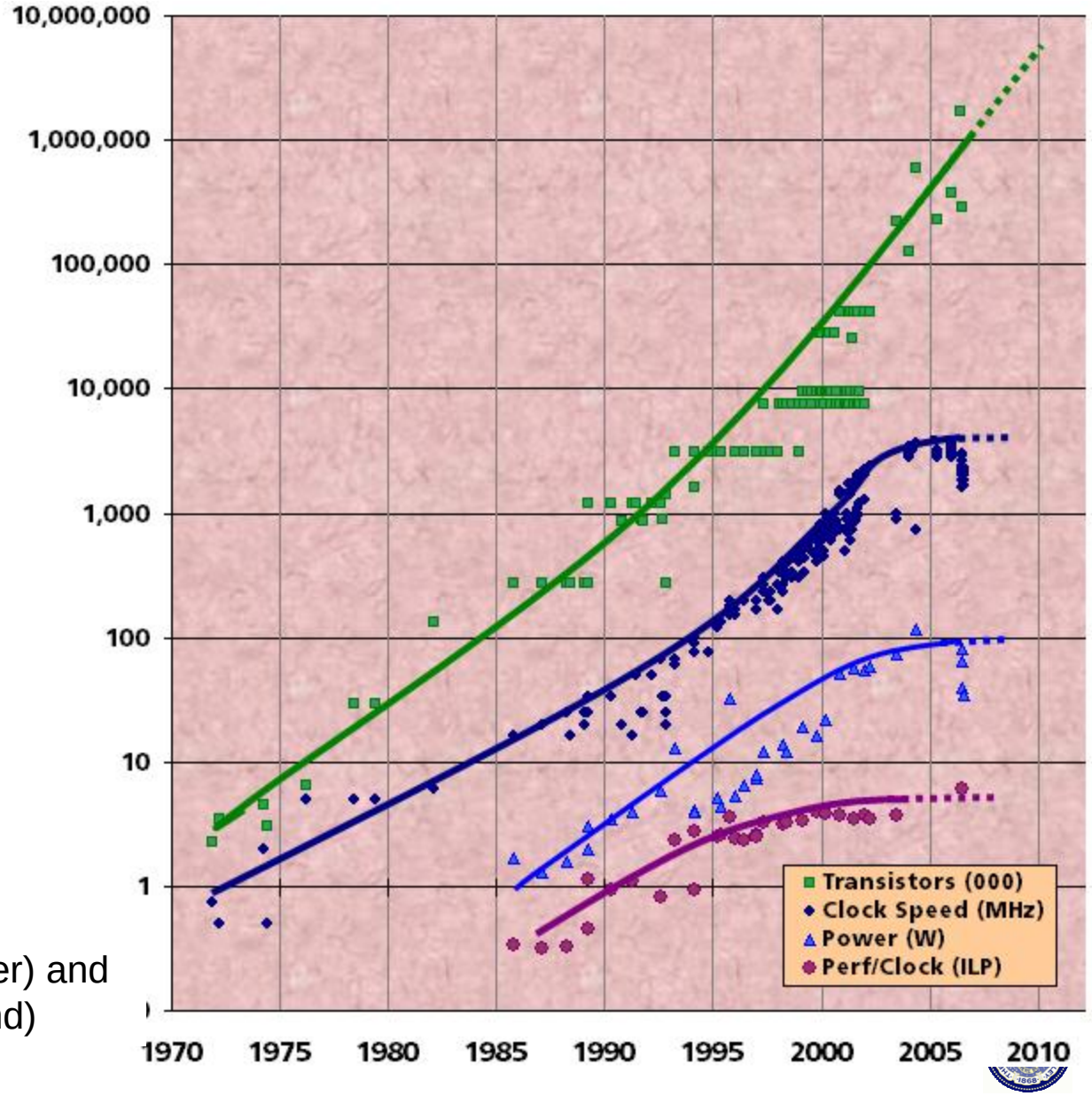
Clock Scaling Extrapolation:



Source: Shekhar Borkar, Intel Corp

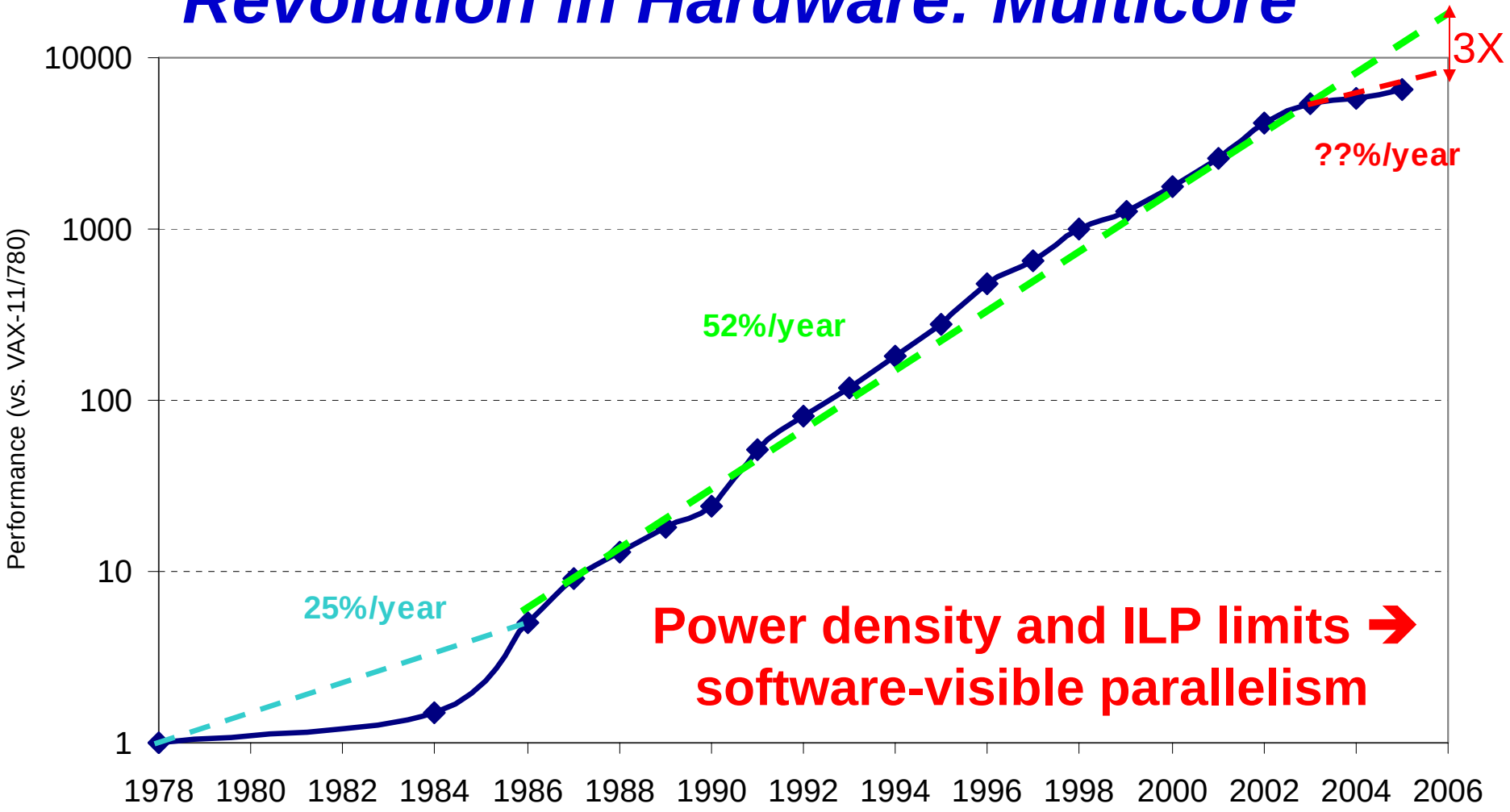
Revolution is Happening Now

- **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



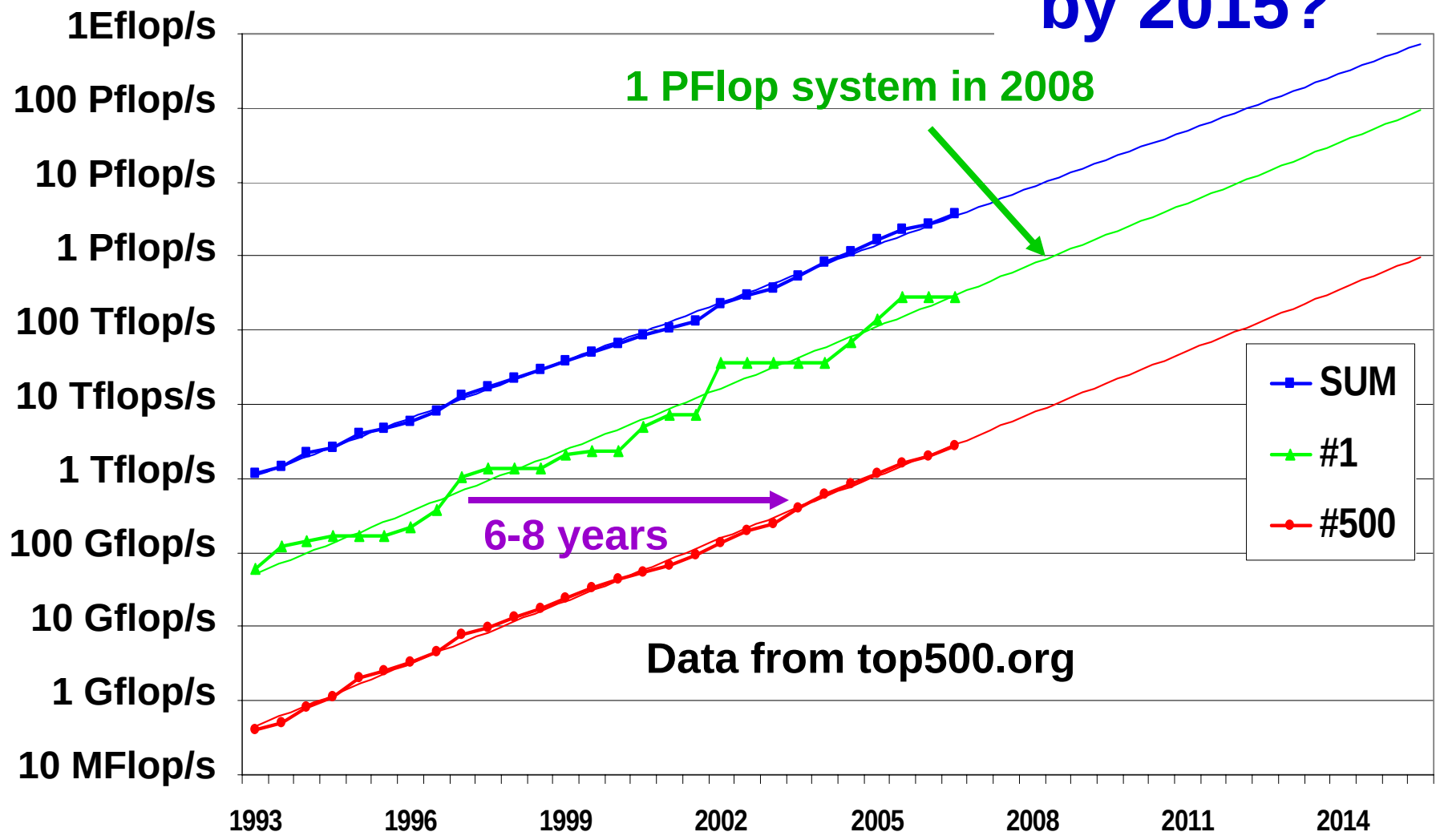
From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, Sept. 15, 2006

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?

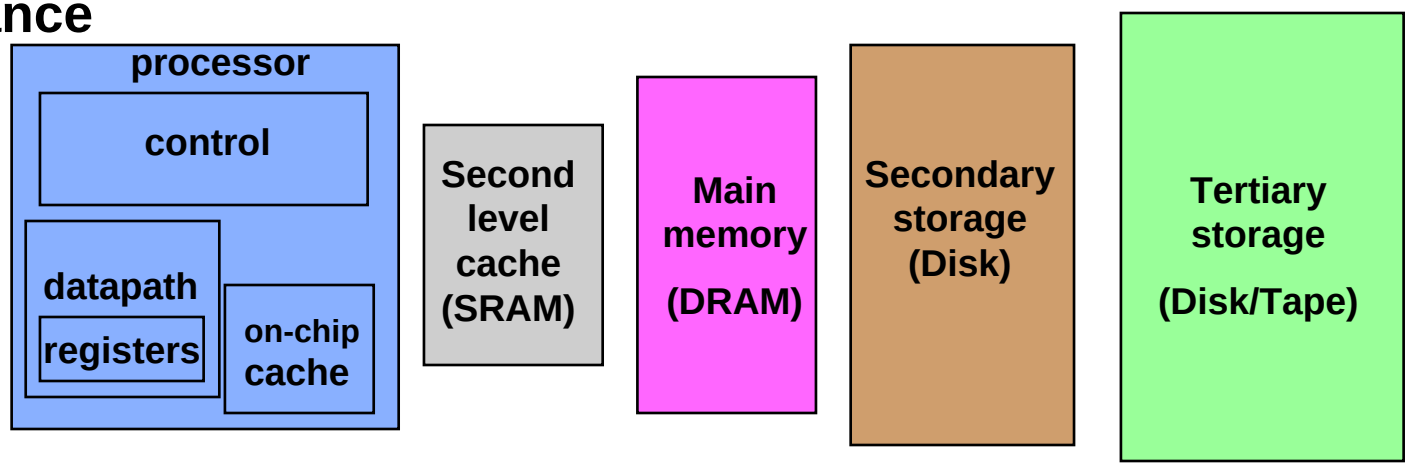
Petaflop with ~1M Cores

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

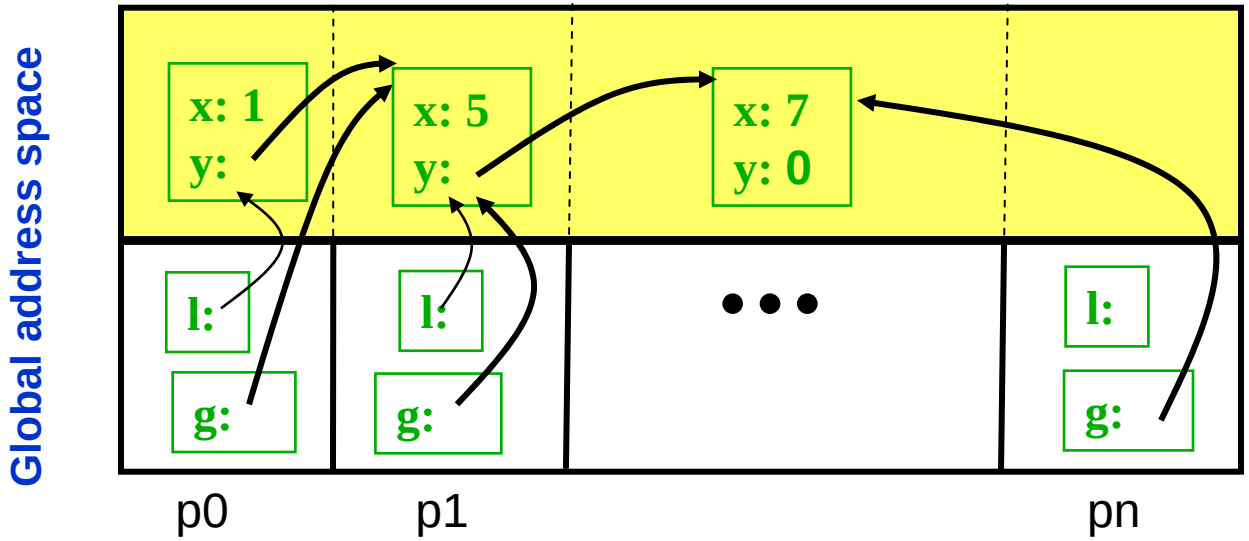
- Parallel software is still an unsolved problem !
- **Most parallel programs are written using either:**
  - Message passing with a SPMD model
    - for scientific applications; scales easily
  - Shared memory with threads in OpenMP, Threads, or Java
    - non-scientific applications; easier to program
- **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

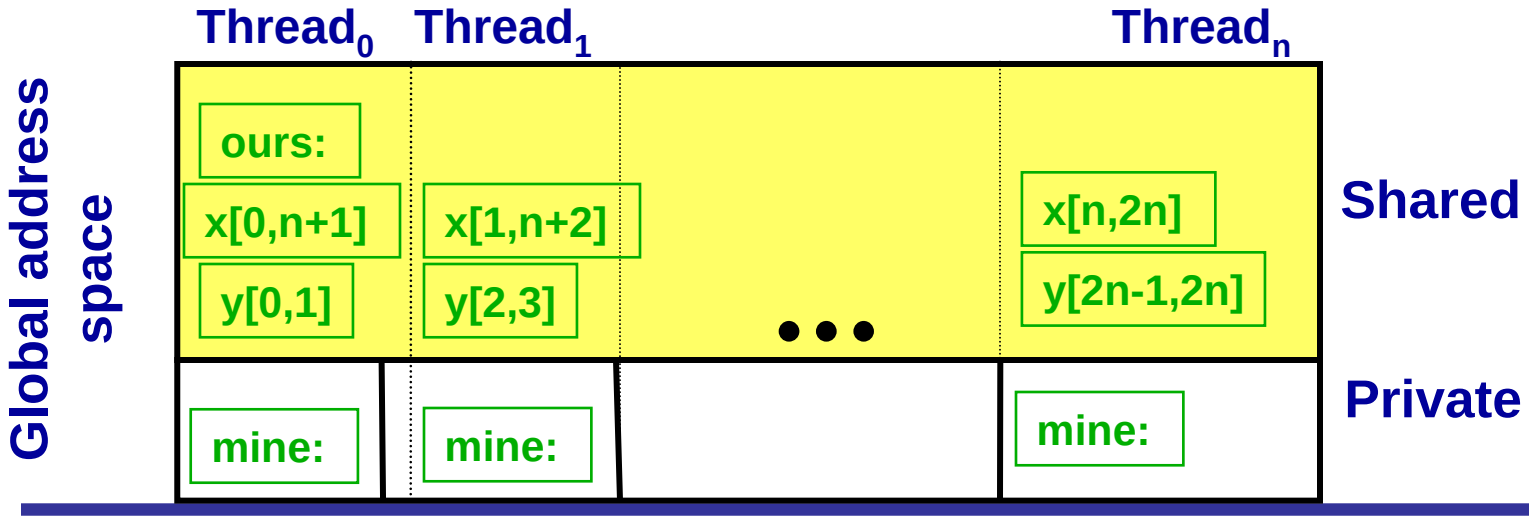
- C variables and objects are allocated in the private memory space
- Shared variables are allocated only once, in thread 0's space

```
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 */
```

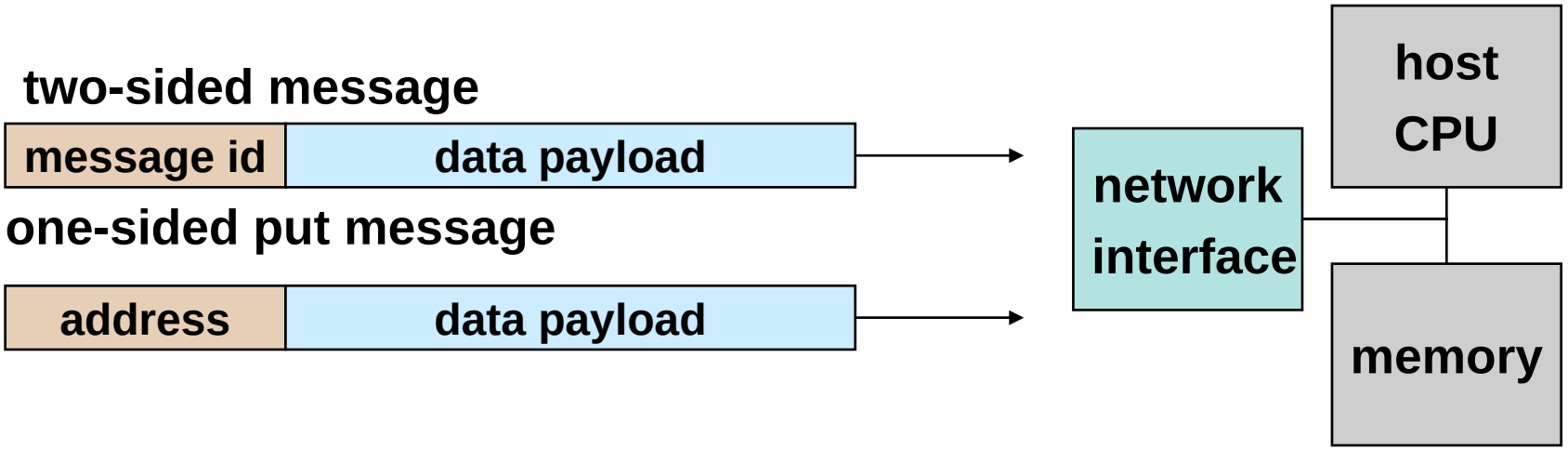
- Heap objects may be in either private or shared space



PGAS Language for Multicore

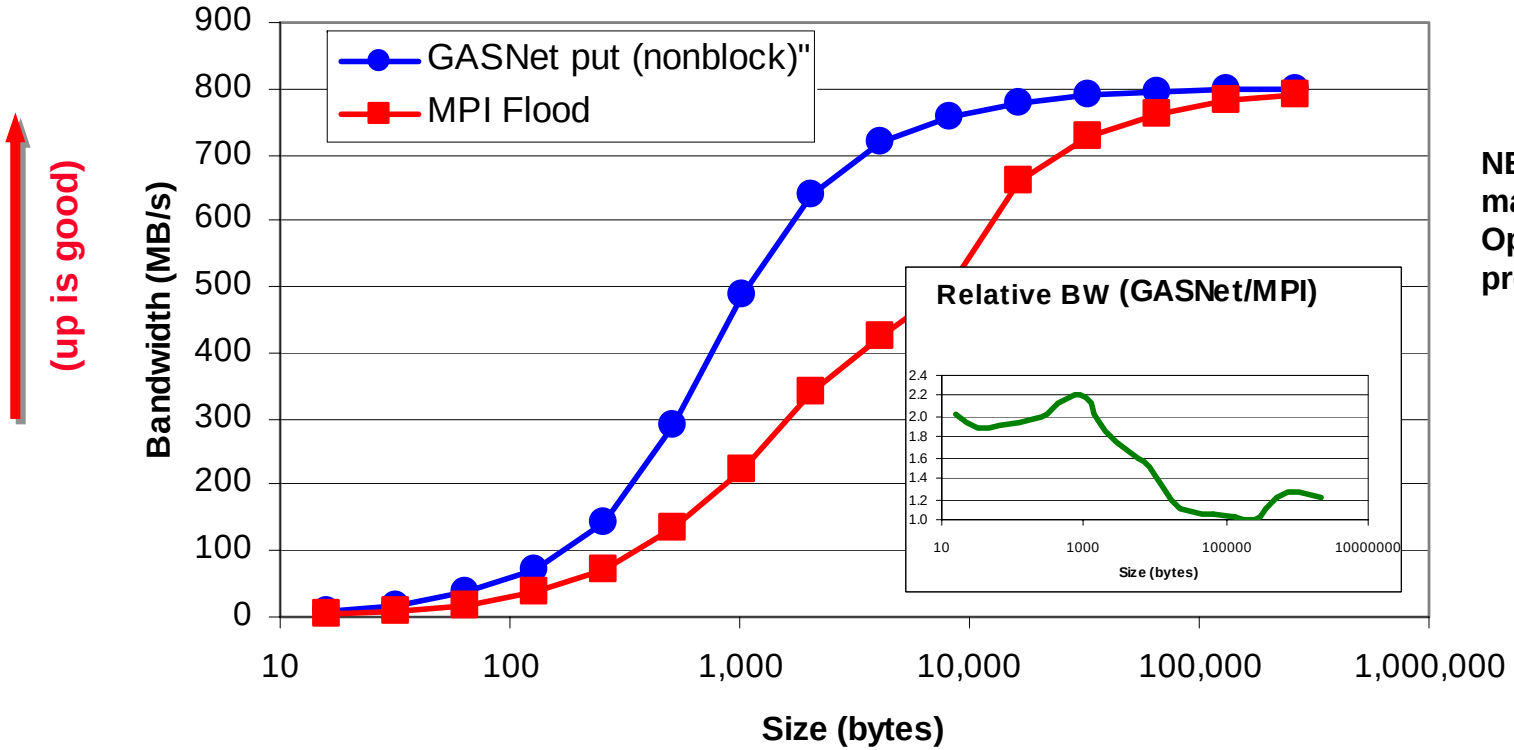
- **PGAS languages are a good fit to shared memory machines**
  - Global address space implemented as reads/writes
  - Current UPC and Titanium implementation uses threads
  - Working on System V shared memory for UPC
- **"Competition" on shared memory is OpenMP**
  - PGAS has locality information that 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)

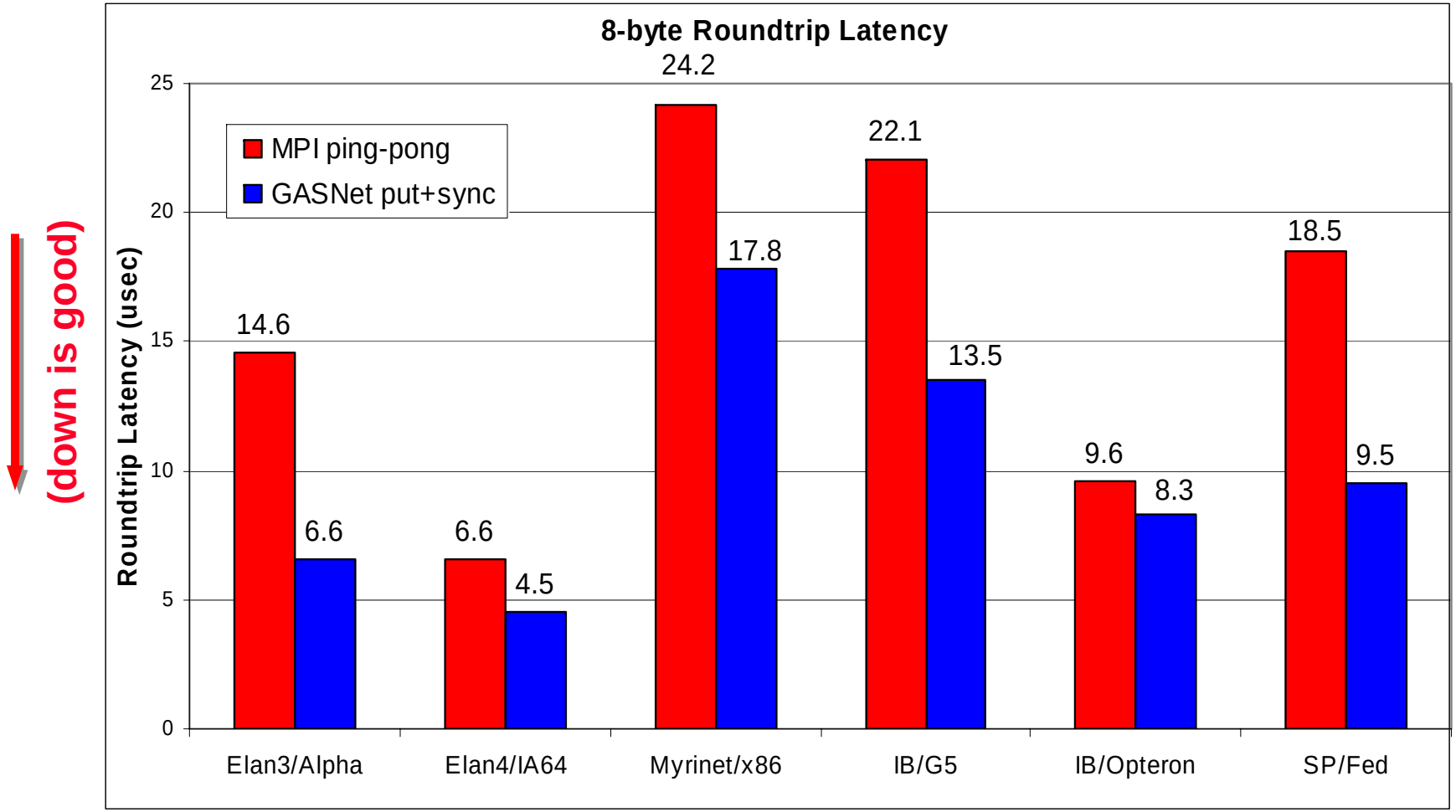
One-Sided vs. Two-Sided: Practice



NERSC Jacquard machine with Opteron processors

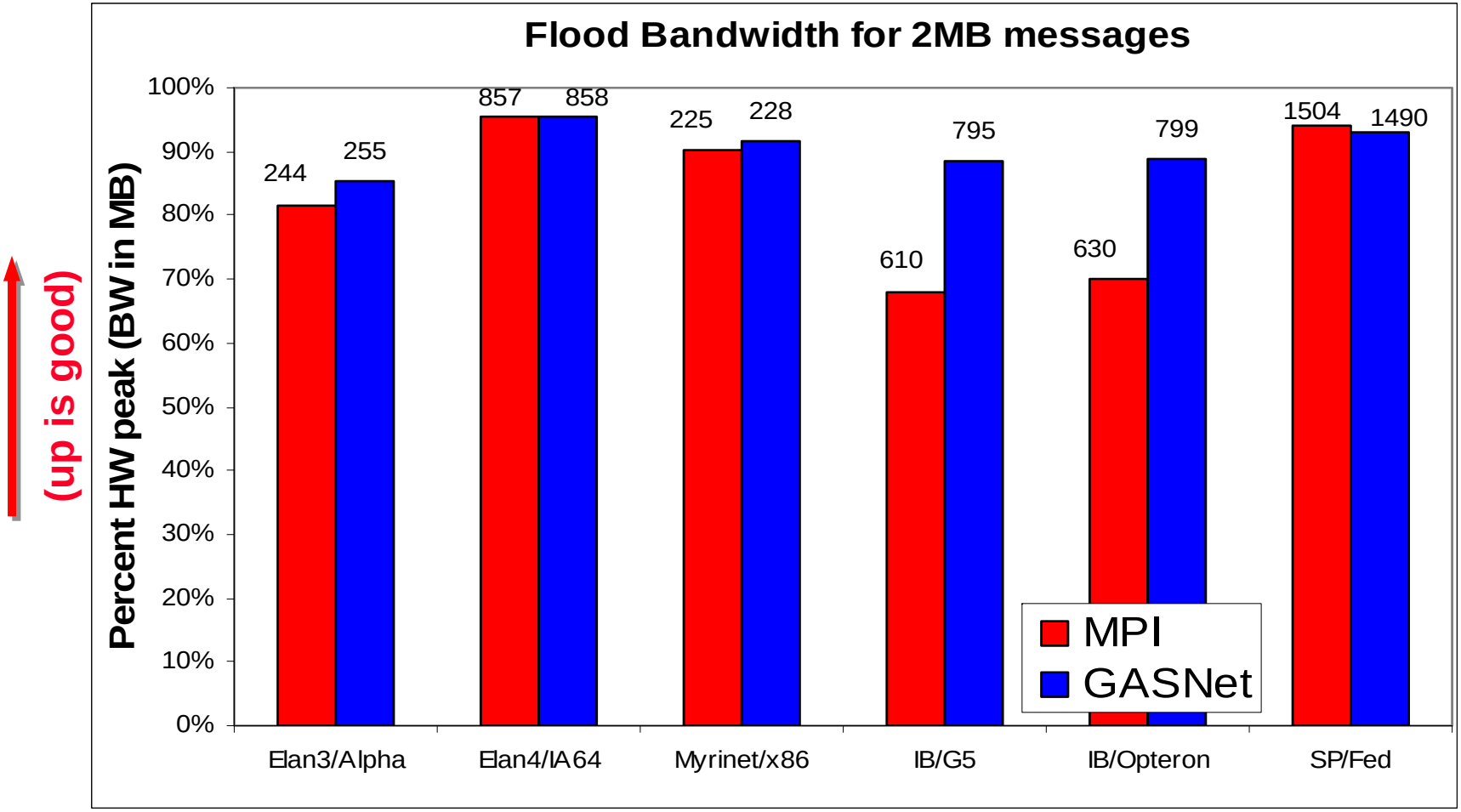
- InfiniBand: GASNet vapi-conduit and OSU MVAPICH 0.9.5
- Half power point (N ½) differs by *one order of magnitude*
- This is not a criticism of the implementation!

GASNet: Portability and *High-Performance*



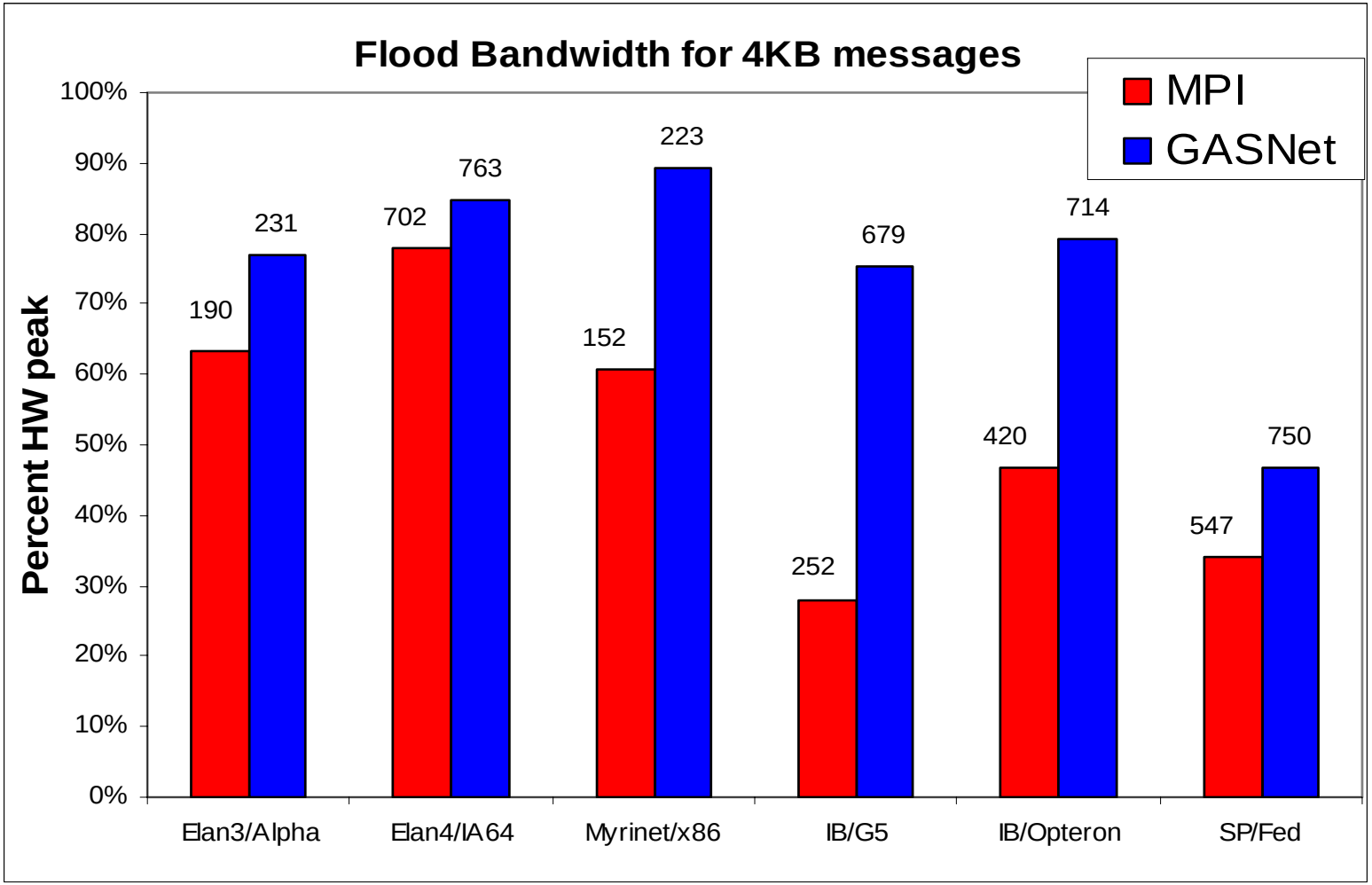
GASNet better for latency across machines

GASNet: Portability and *High-Performance*



GASNet at least as high (comparable) for large messages

GASNet: Portability and *High-Performance*

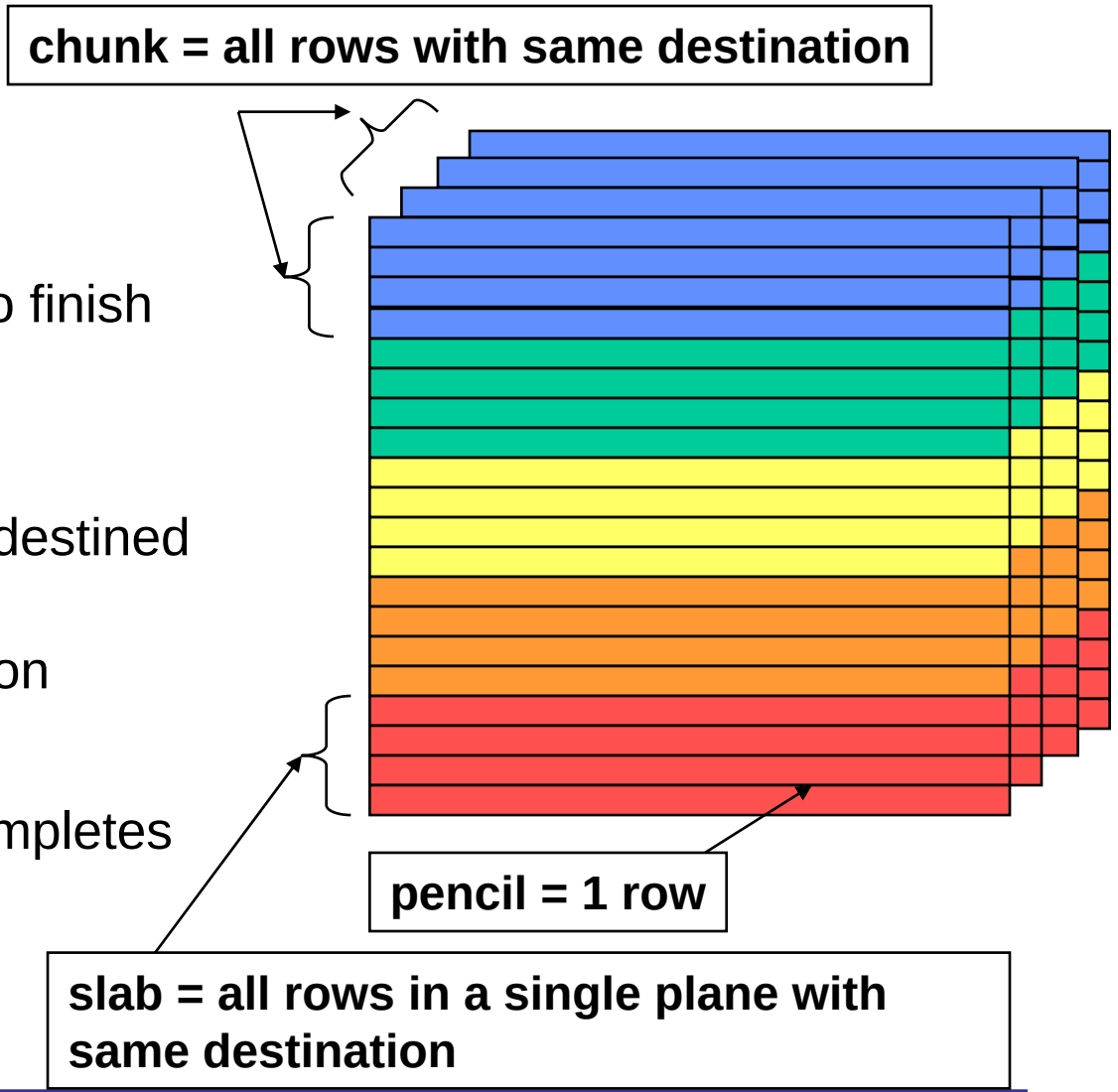


(up is good)

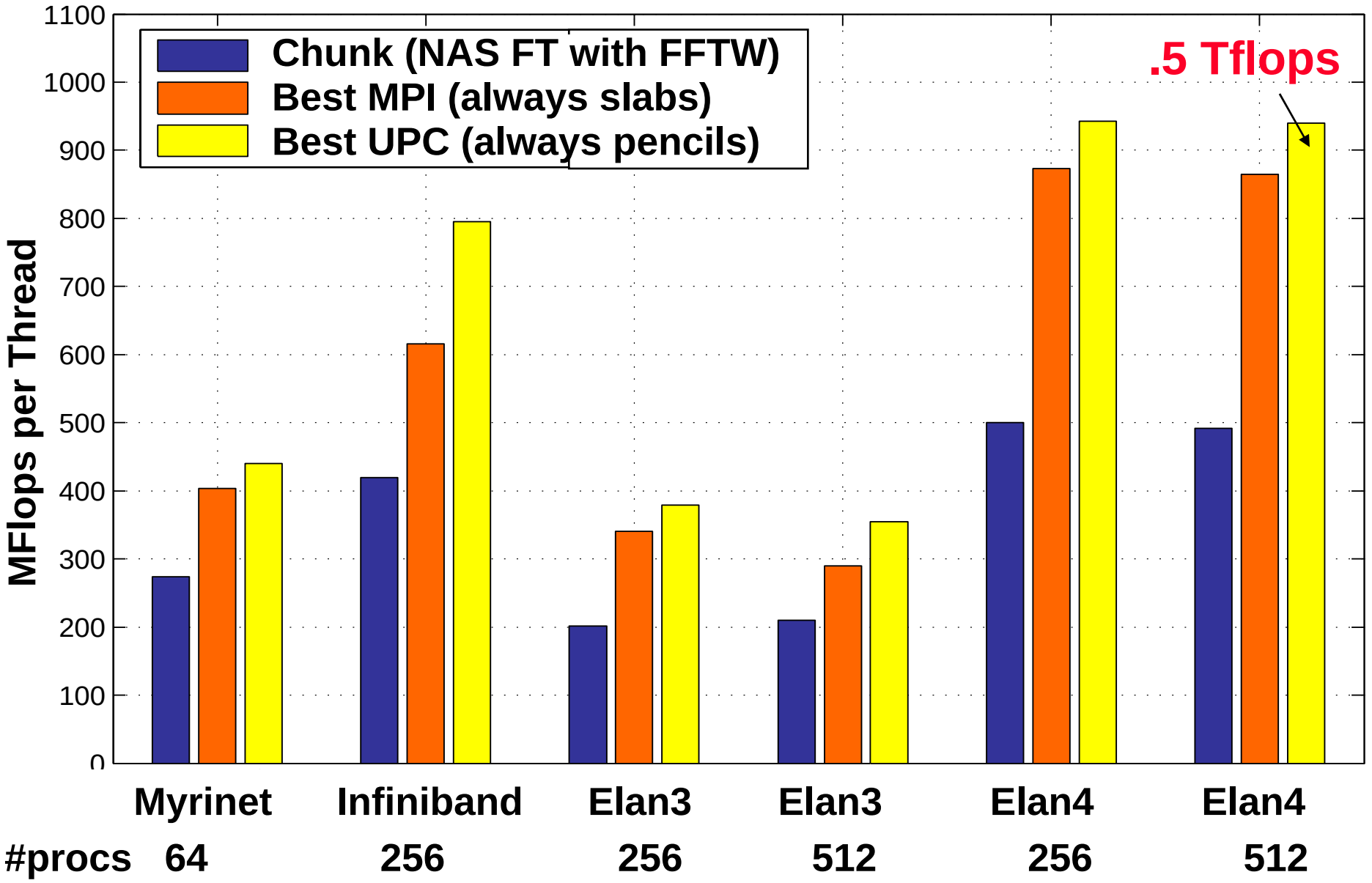
GASNet excels at mid-range sizes: important for overlap

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

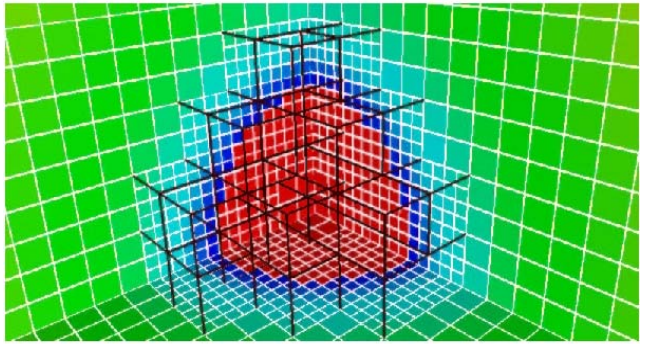
AMR in Titanium

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

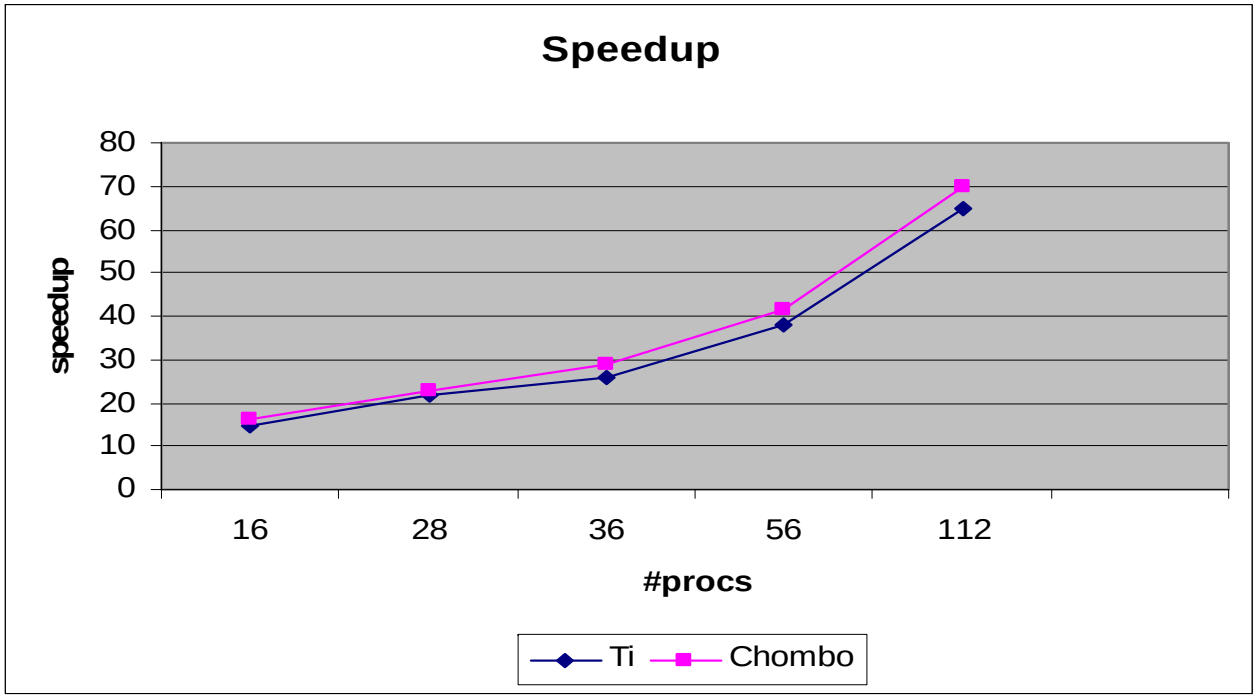


Code Size in Lines		
	C++/F/MPI	Titanium
AMR data Structures	35000	2000
AMR operations	6500	1200
Elliptic PDE solver	4200*	1500

* Somewhat more functionality in PDE part of Chombo code

10X reduction in lines of code!

Performance of Titanium AMR



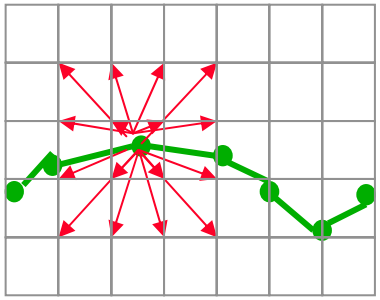
Comparable parallel performance

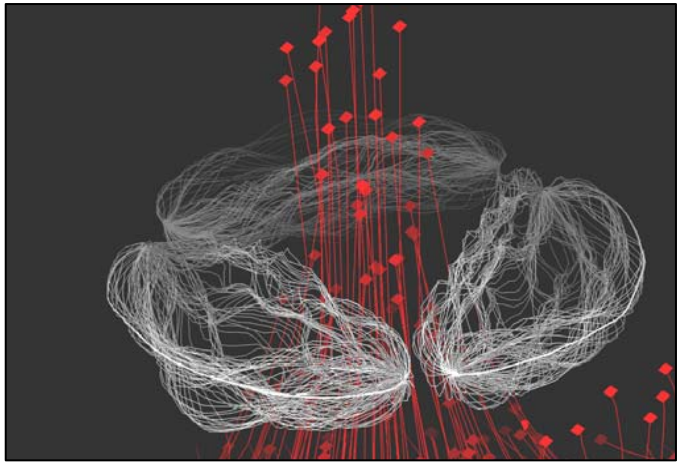
- **Serial: Titanium is within a few % of C++/F; sometimes faster!**
- **Parallel: Titanium scaling is comparable with generic optimizations**
  - **optimizations (SMP-aware) that are not in MPI code**
  - **additional optimizations (namely overlap) not yet implemented**

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

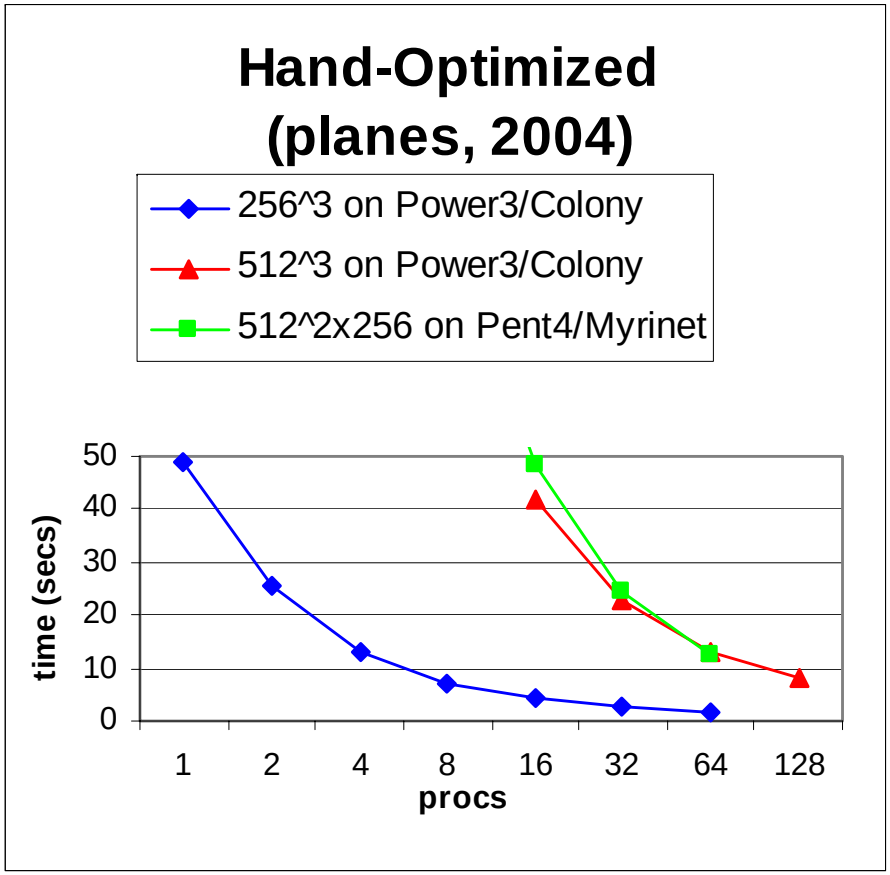


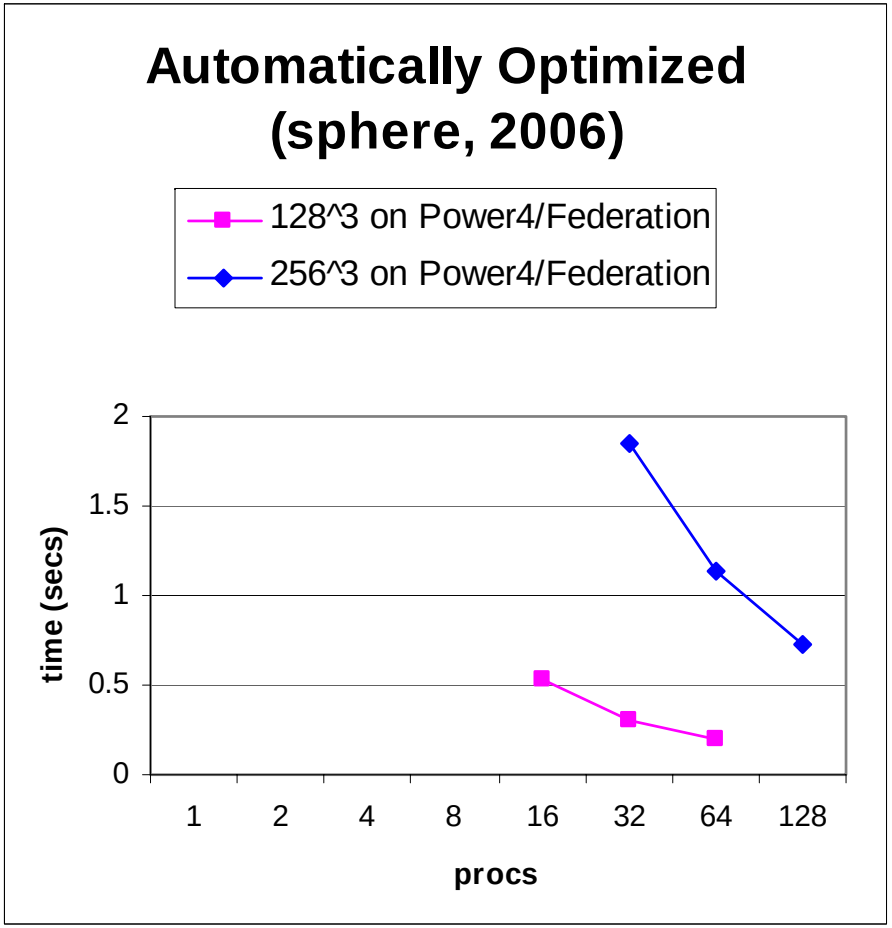


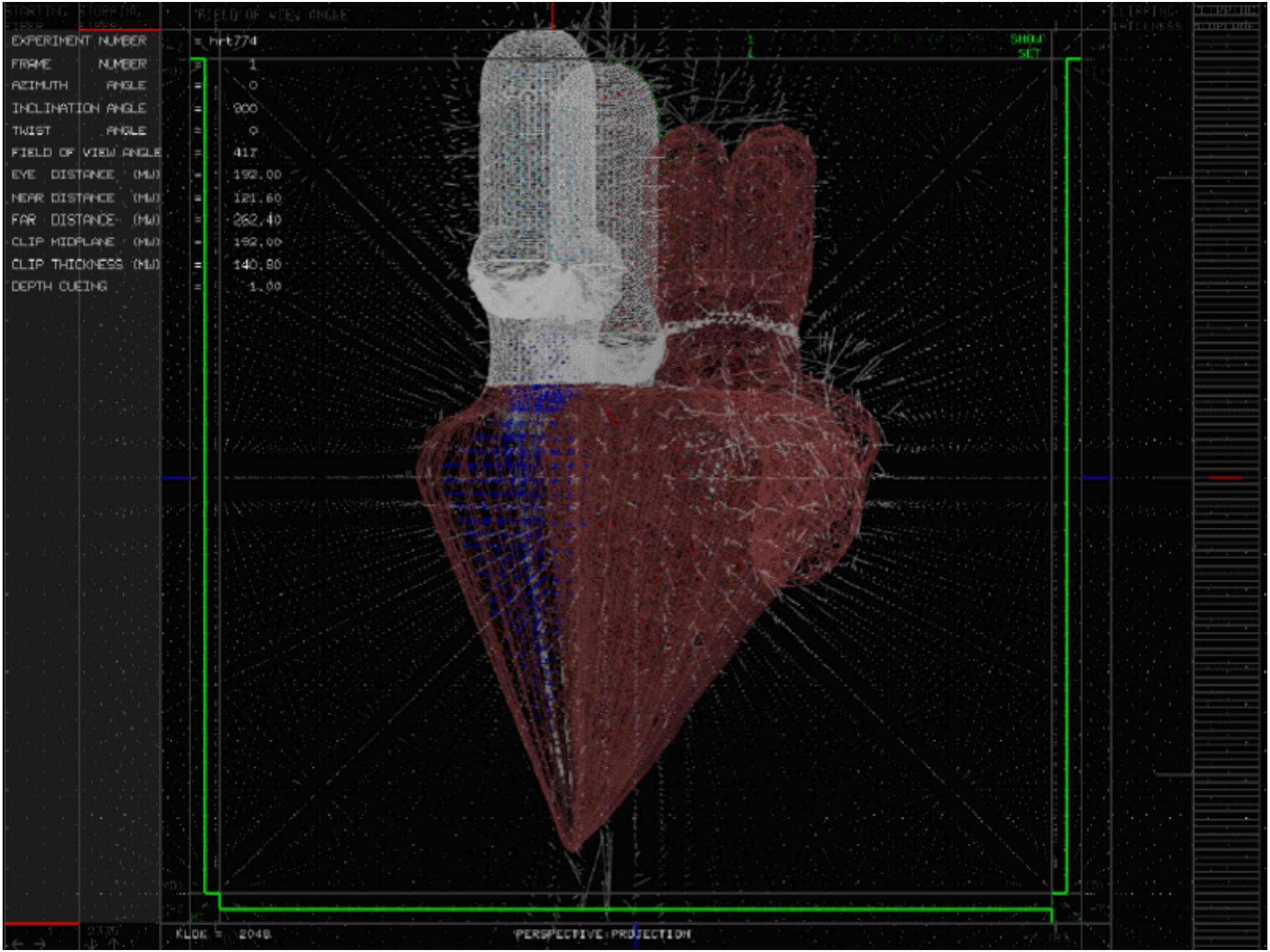
Code Size in Lines	
Fortran	Titanium
8000	4000

Note: Fortran code is not parallel

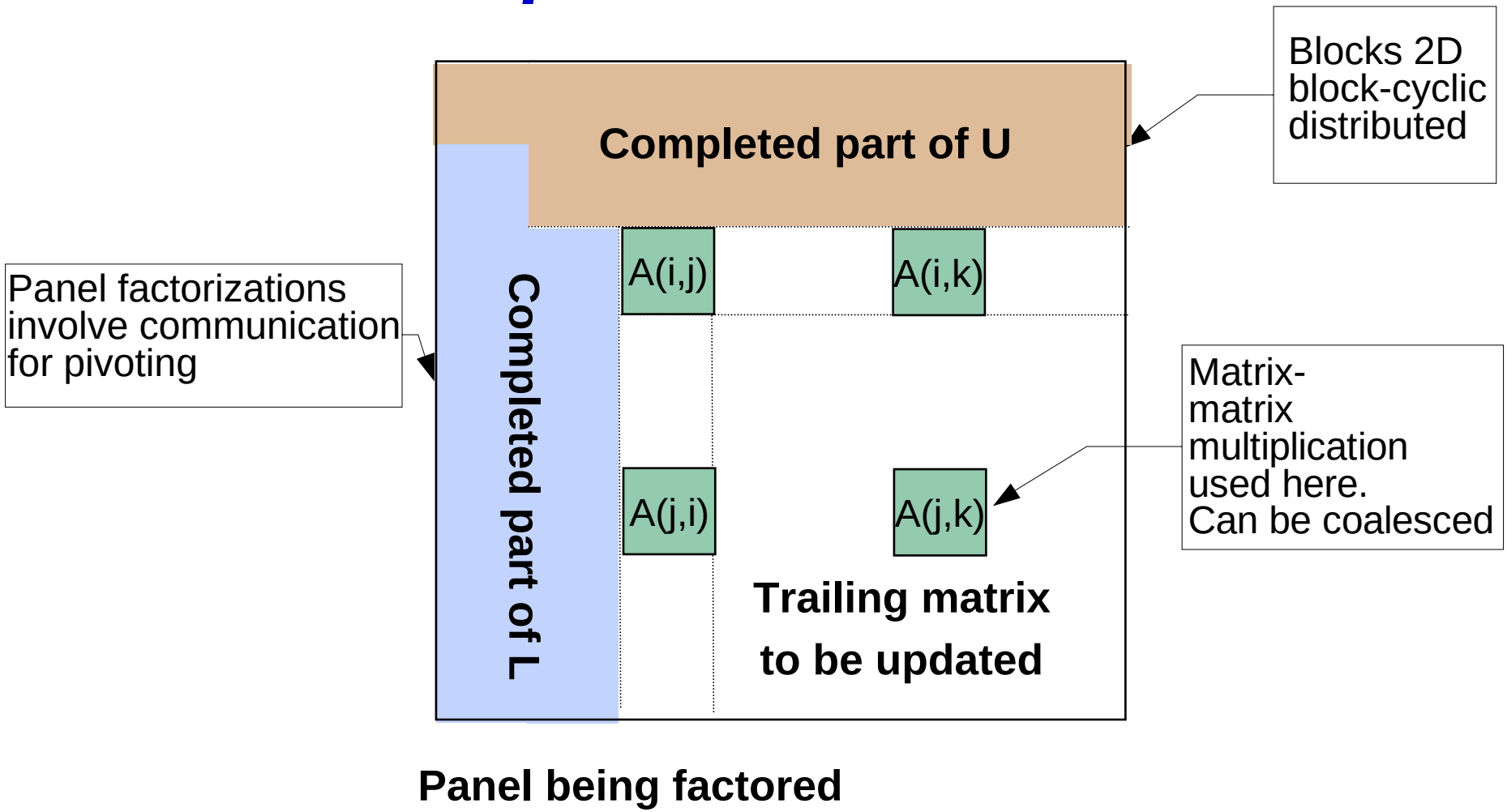
Immersed Boundary Method Performance






EXPERIMENT NUMBER = mrt774
FRAME NUMBER = 1
AZIMUTH ANGLE = 0
INCLINATION ANGLE = 900
TWIST ANGLE = 0
FIELD OF VIEW ANGLE = 417
EYE DISTANCE (MU) = 192.00
NEAR DISTANCE (MU) = 121.60
FAR DISTANCE (MU) = 262.40
CLIP MIDPLANE (MU) = 192.00
CLIP THICKNESS (MU) = 140.80
DEPTH CUEING = 1.00
SHOW SET
KLOK = 2048
PERSPECTIVE PROJECTION

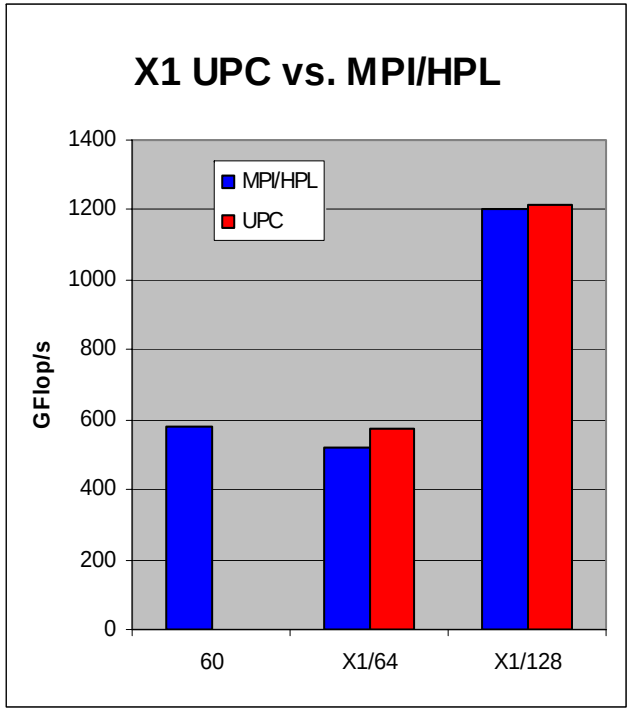
Dense and Sparse Matrix Factorization

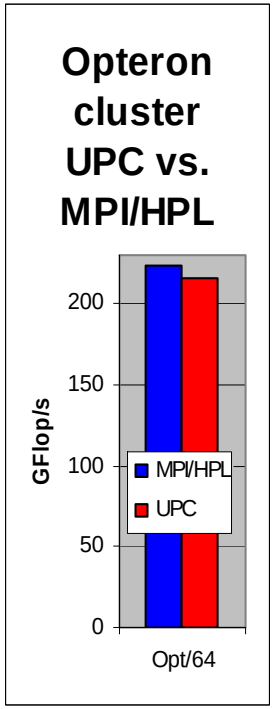


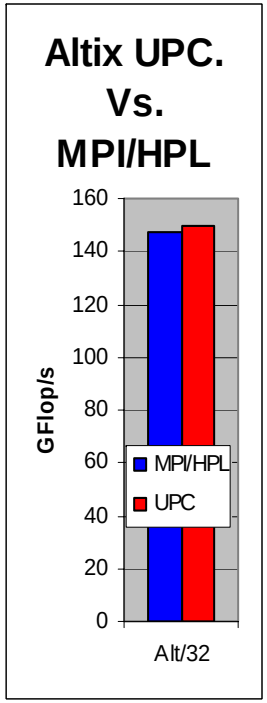
Matrix Factorization in UPC

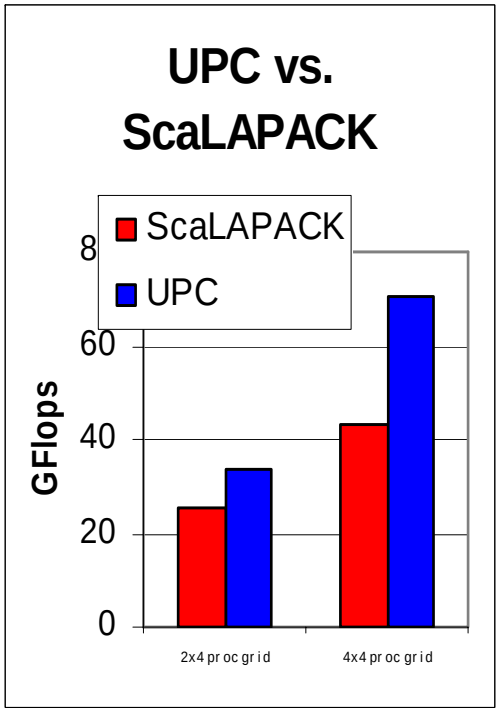
- **UPC factorization uses a highly multithreaded style**
  - Used to mask latency and to mask dependence delays
  - Three levels of threads:
    - UPC threads (data layout, each runs an event scheduling loop)
    - Multithreaded BLAS (boost efficiency)
    - User level (non-preemptive) threads with 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