Partitioned Global Address Space Languages for Multilevel Parallelism

Katherine Yelick

U.C. Berkeley and Lawrence Berkeley National Lab

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

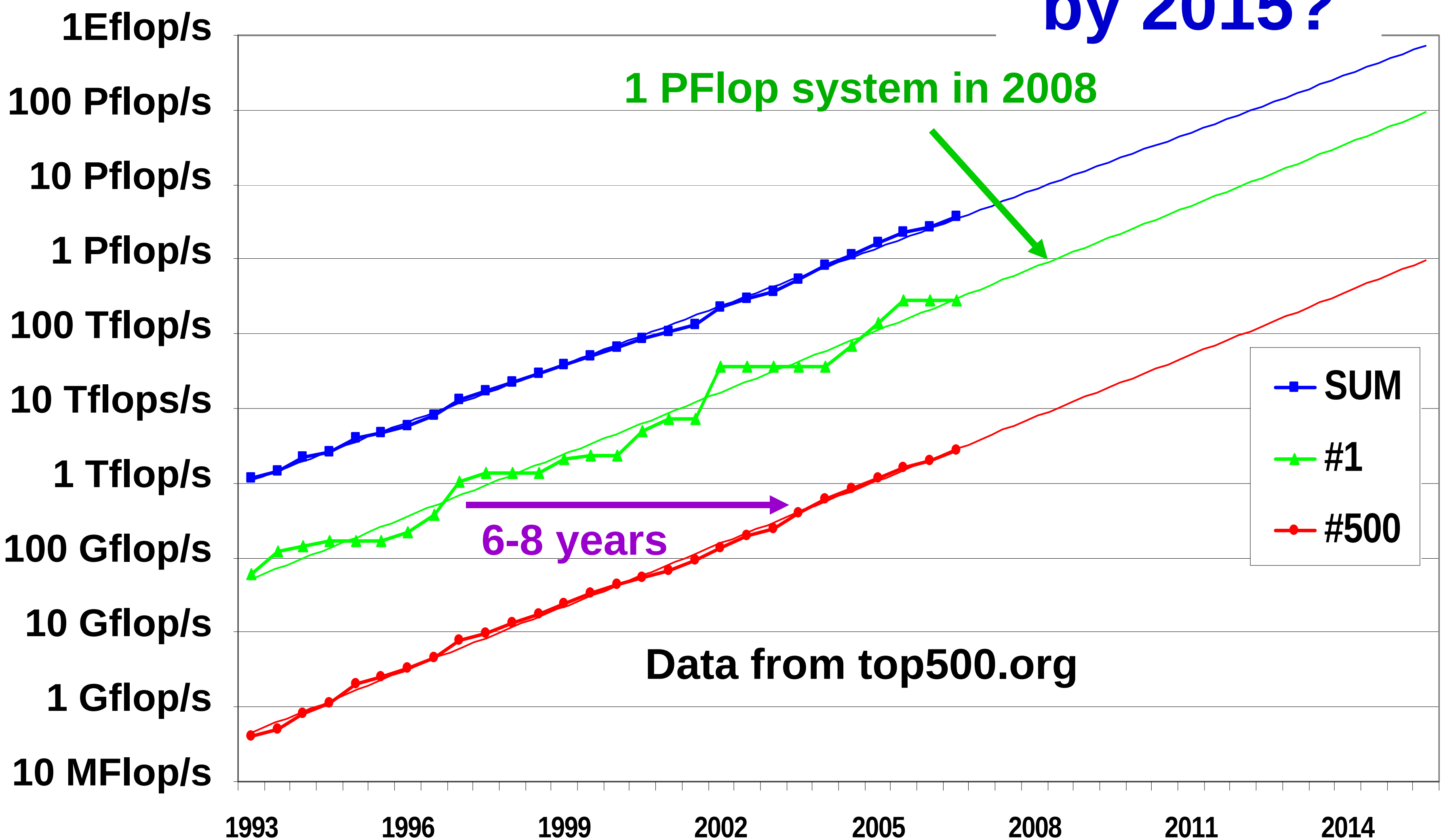
HPC Programming: Where are We?

- IBM SP at NERSC/LBNL has as 6K processors
  - There were 6K transistors in the Intel 8080a implementation
- BG/L at LLNL has 64K processor cores
  - There were 68K transistors in the MC68000
- A BG/Q system with 1.5M processors may have more processors than there are logic gates per processor

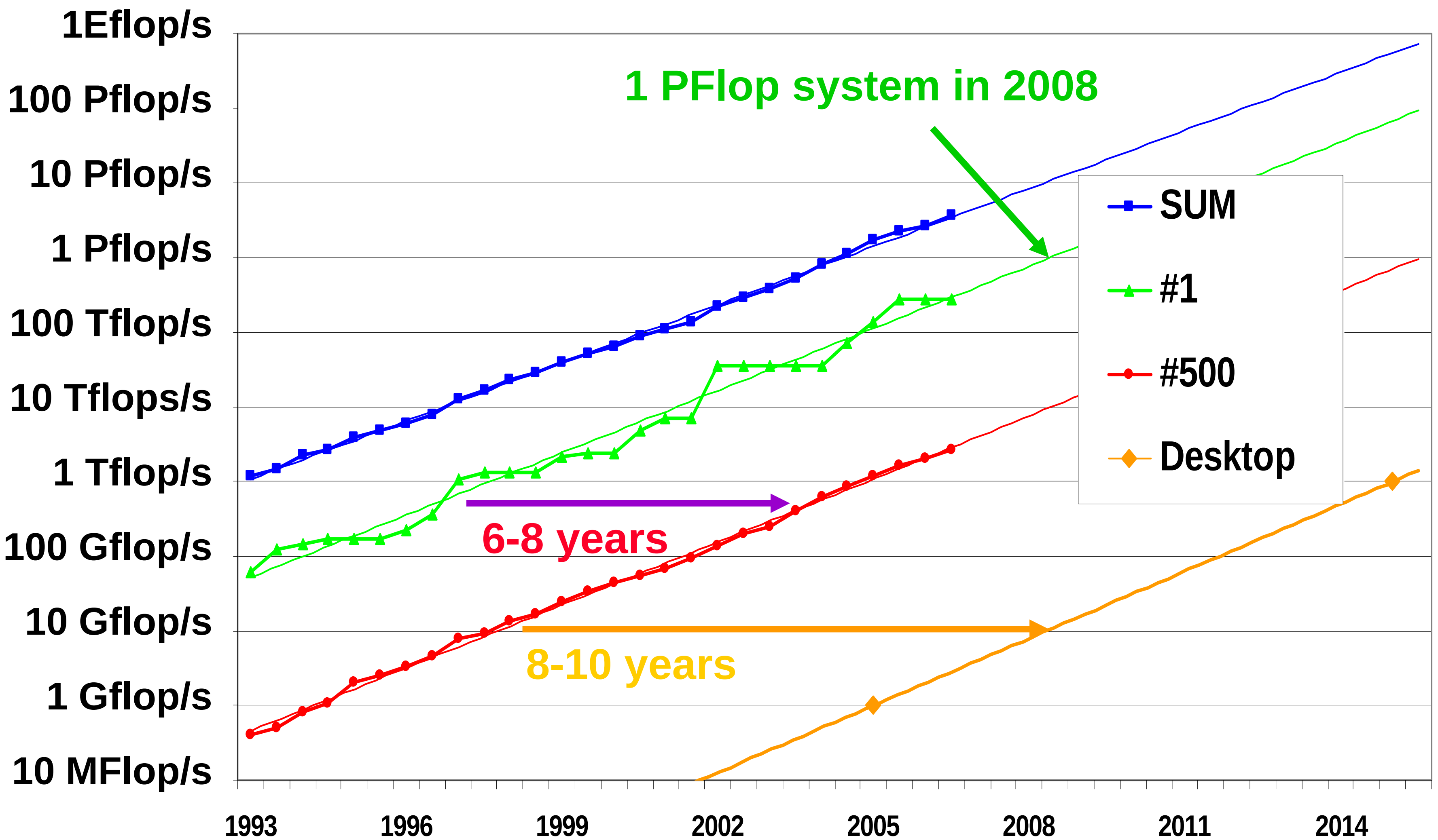
- HPC Applications developers today write programs that are as complex as describing where every single bit must move between the 6,000 transistors of the 8080a
- We need to *at least* get to "assembly language" level

Slide source: Horst Simon and John Shalf, LBNL/NERSC

Petaflop with ~1M Cores Common by 2015?



Petaflop Desktop By 2036



Predictions

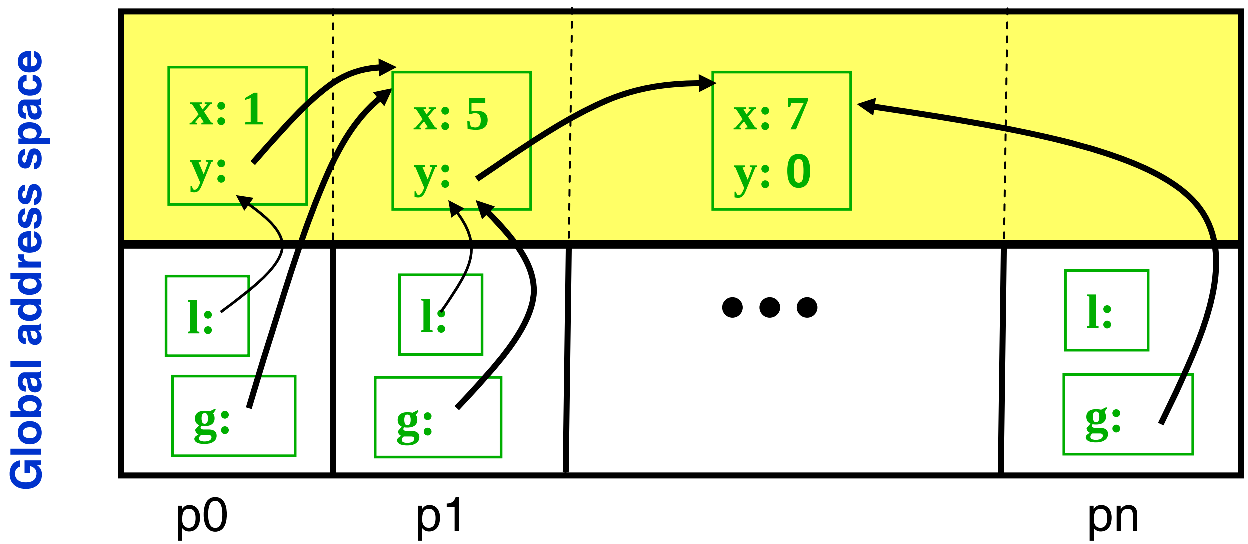
- **Parallelism will explode**
  - Number of cores will double every 12-24 months
  - Petaflop (million processor) machines will be common in HPC by 2015 (all top 500 machines will have this)
- **Performance will become a software problem**
  - Parallelism and locality are key will be concerns for many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

Partitioned Global Address Space (PGAS) Languages:

What, Why, and How

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**

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

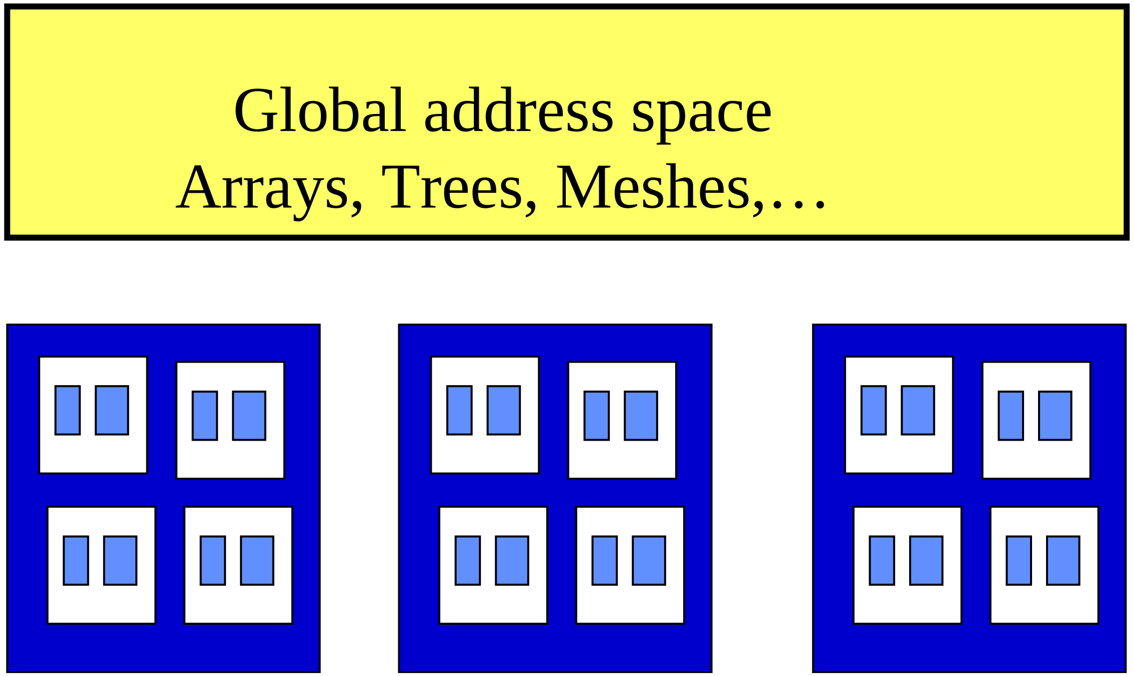
PGAS Language Overview

- **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.**

PGAS Language for Multicore

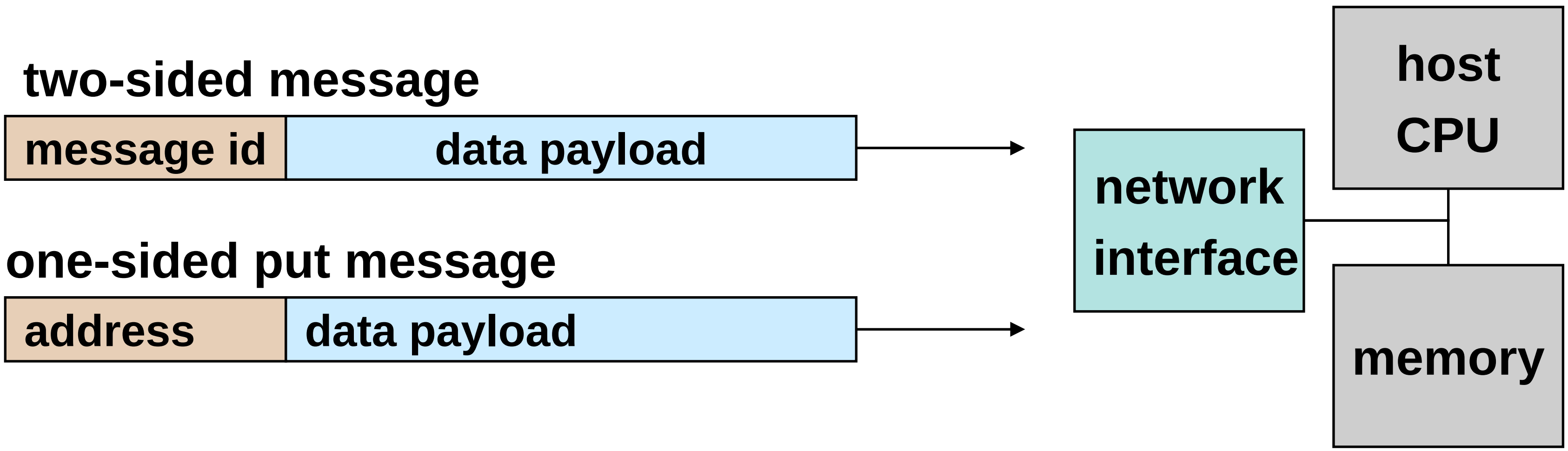
- **PGAS languages are a good fit to shared memory machines**
  - Global address space implemented as reads/writes
  - Current UPC and Titanium implementation uses threads
  - Working on System V shared memory for UPC
- **"Competition" on shared memory is OpenMP**
  - PGAS has locality information that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



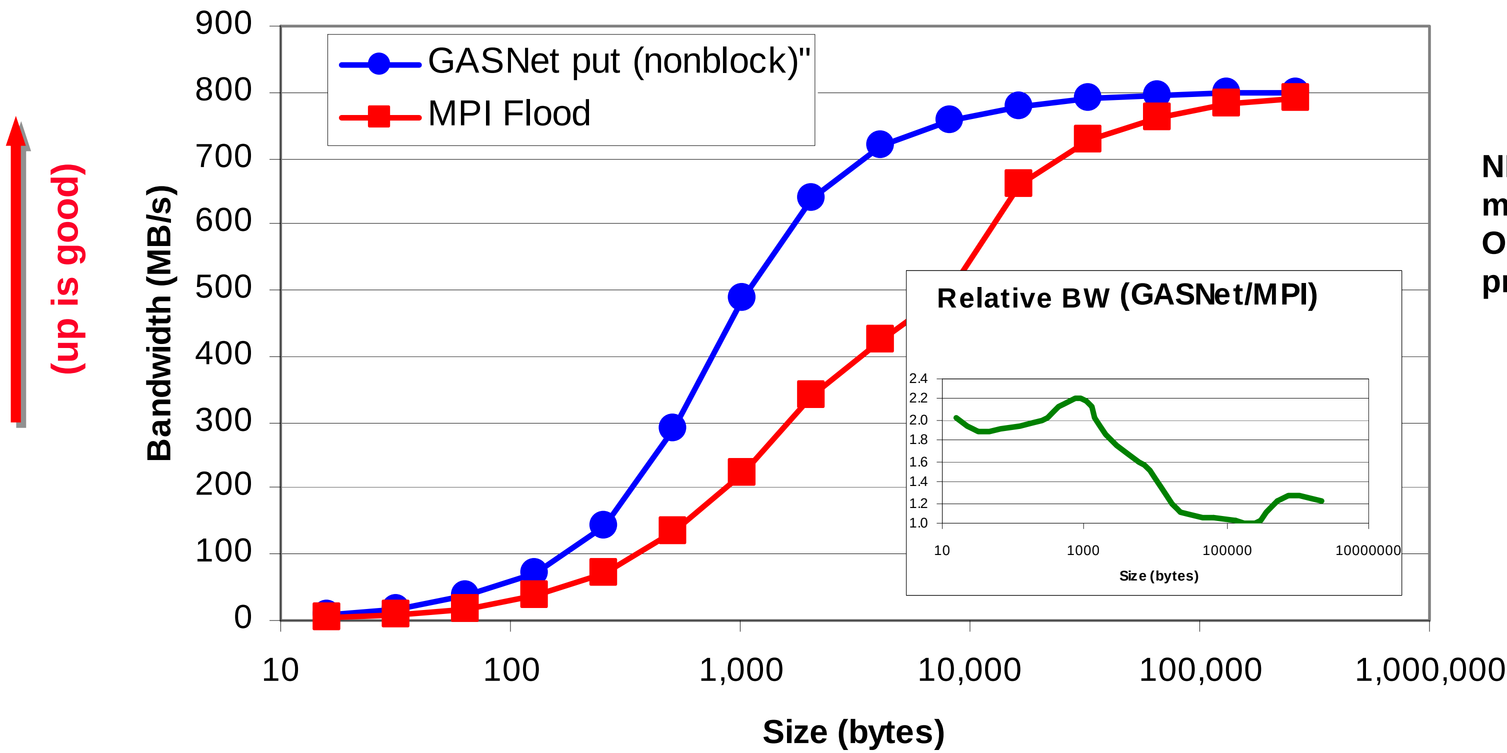
- Single global address space used across cores, SMPs, cluster/MPP networks
- Within an SMP or multicore, threads with direct load/store instructions are used
- Between nodes, one-sided communication (GASNet) is used

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



- **Two-sided requires information from remote host application**
  - Messages that arrive before receive create performance/memory problems
  - Message ordering preserved for semantics; limits bandwidth
  - Matching send to receives adds latency on many networks
- **A one-sided put/get encodes all information needed for delivery**
  - No tag/message matching or ordering
  - Message can be handled directly by a network interface with RDMA support
  - Avoid interrupting the CPU or recording data from

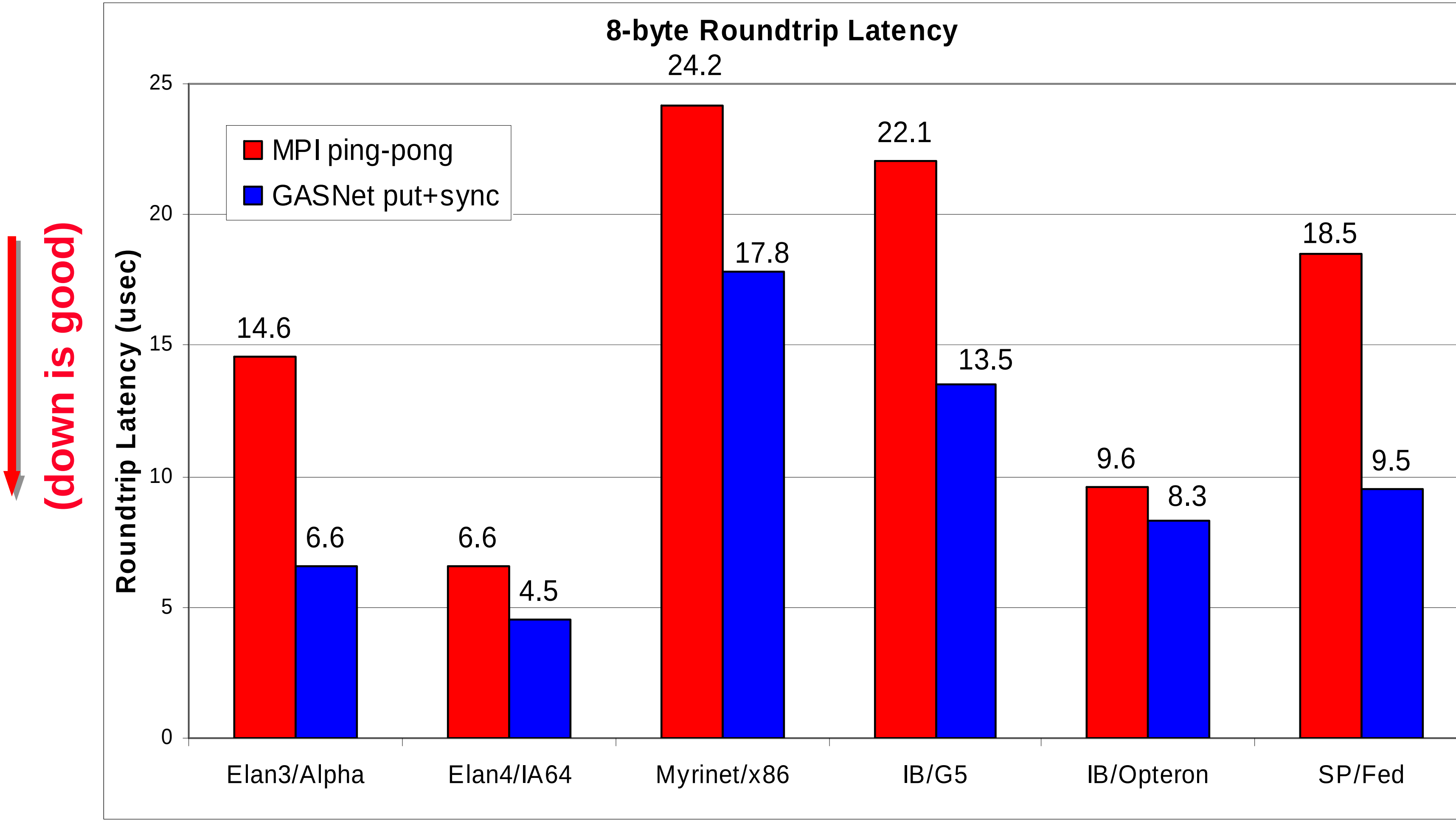
One-Sided vs. Two-Sided: Practice



NERSC Jacquard machine with Opteron processors

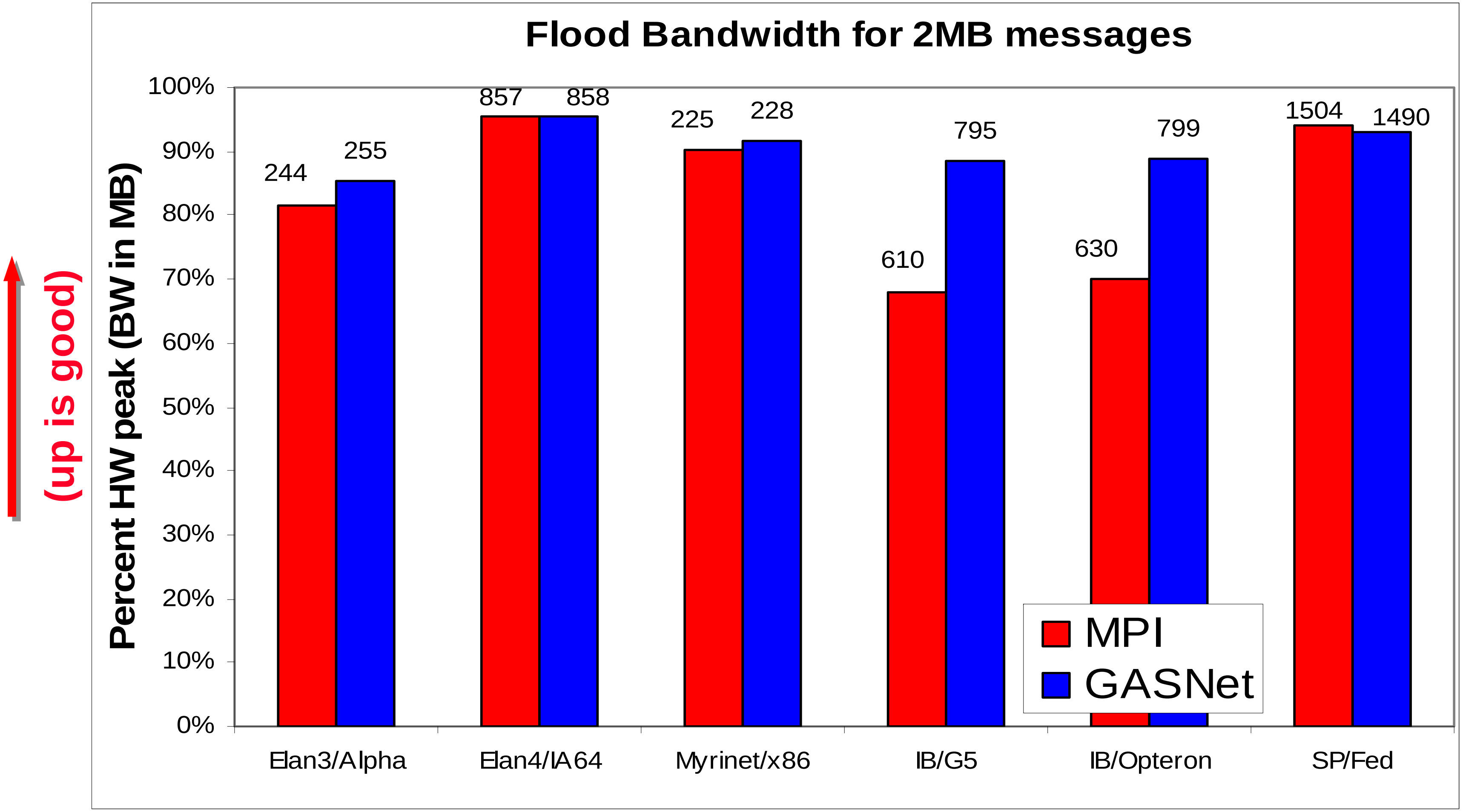
- InfiniBand: GASNet vapi-conduit and OSU MVAPICH 0.9.5
- Half power point (N ½) differs by *one order of magnitude*
- This is not a criticism of the implementation!

GASNet: *Portability* and *High-Performance*



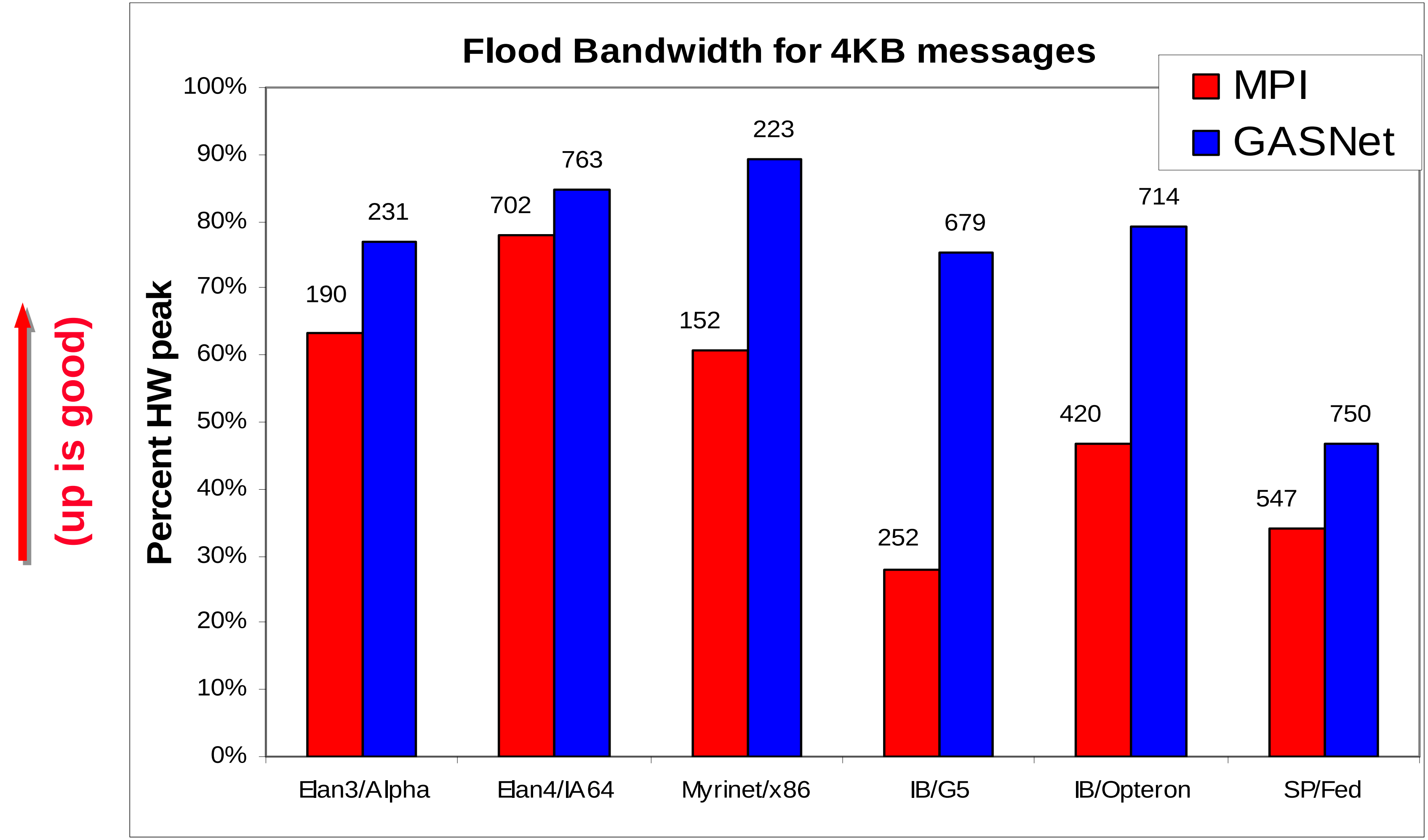
GASNet better for latency across machines

GASNet: Portability and *High-Performance*



GASNet at least as high (comparable) for large messages

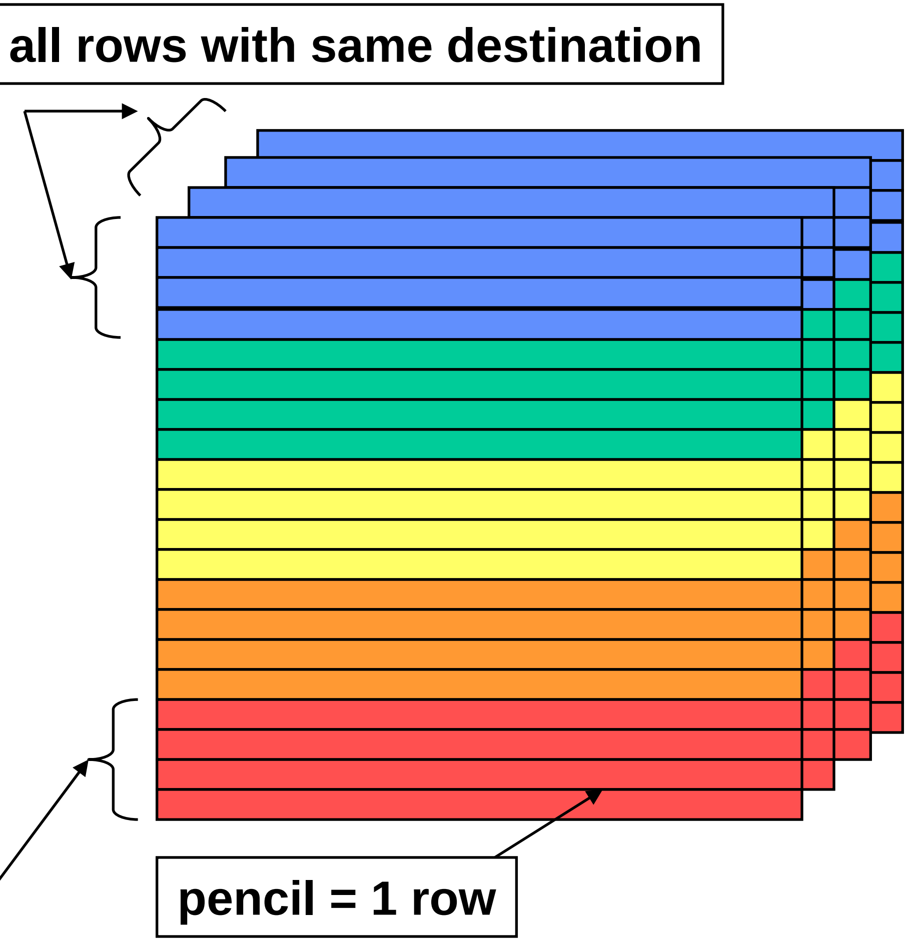
GASNet: Portability and *High-Performance*



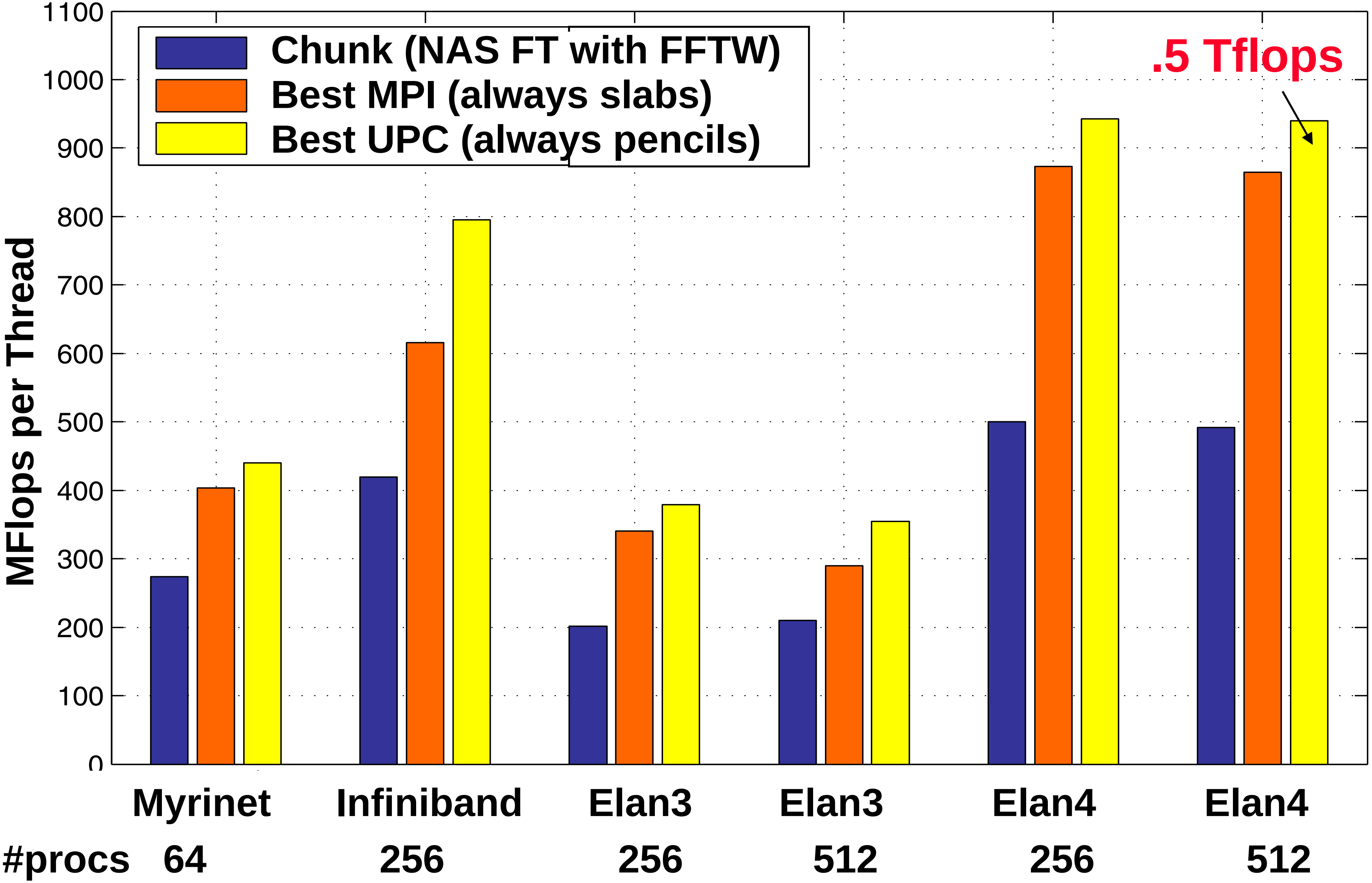
GASNet excels at mid-range sizes: important for overlap

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



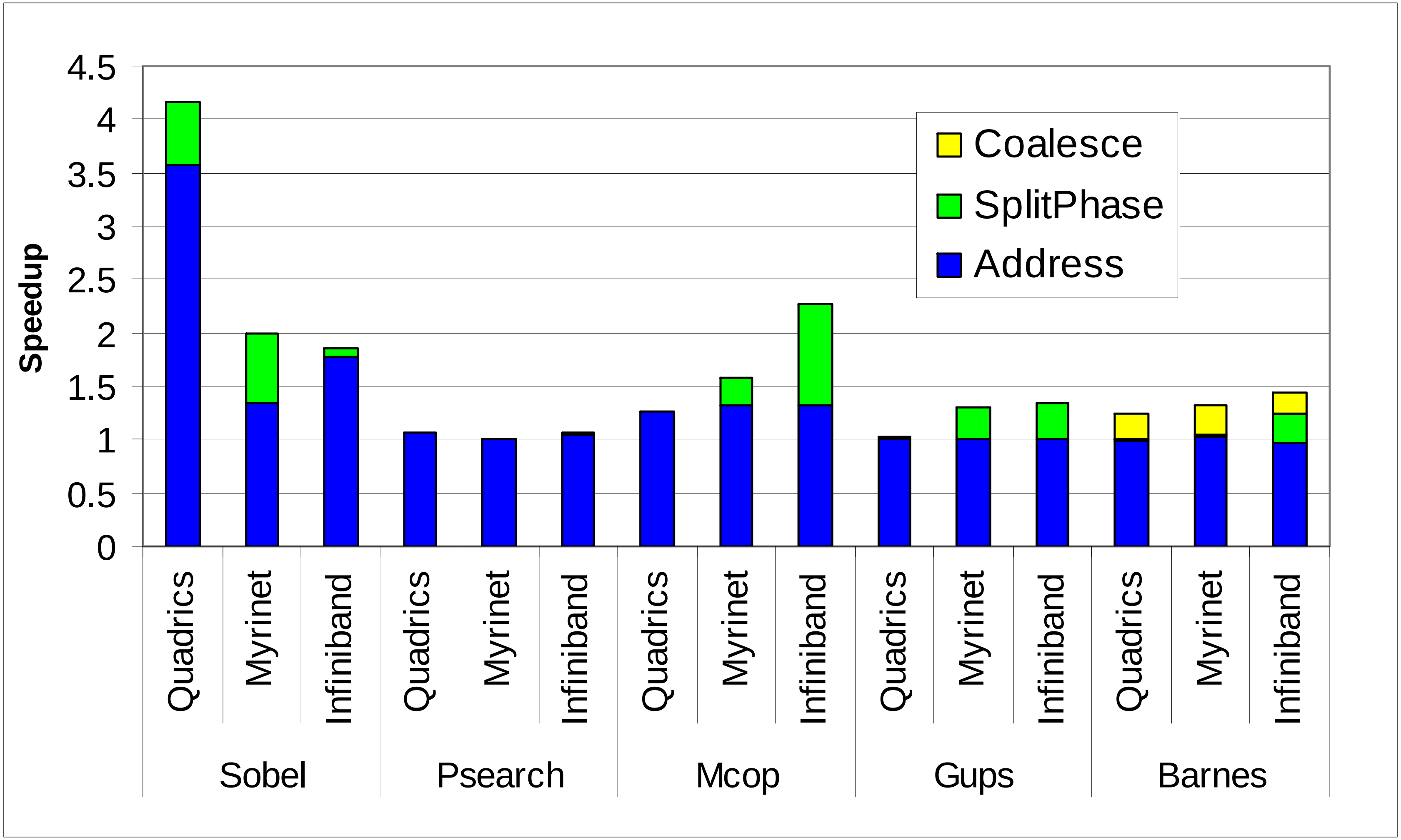
NAS FT Variants Performance Summary



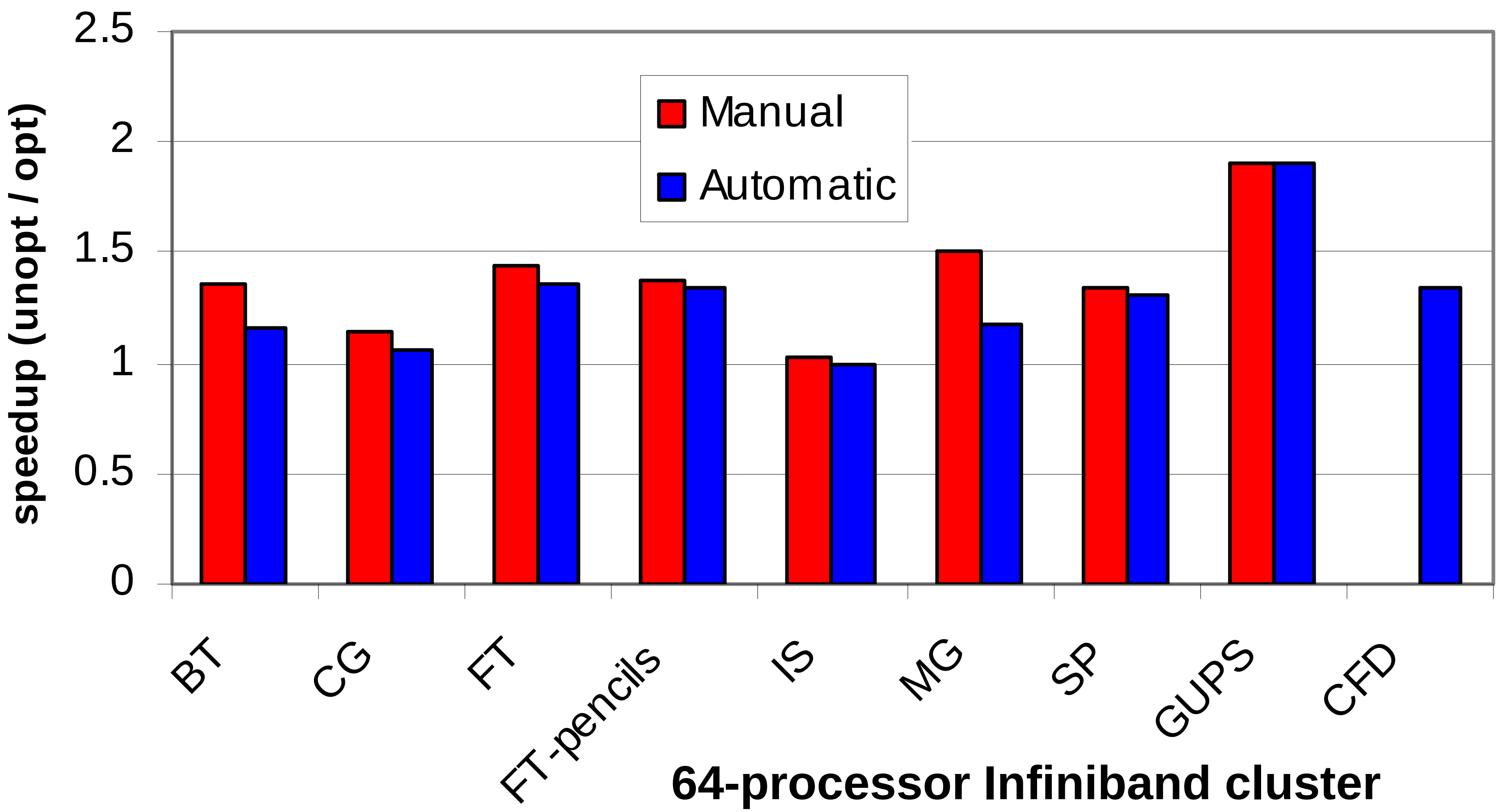
Automating Overlap

- **UPC philosophy: language is expressive enough to allow programmers to hand-optimize**
- **Compiler can improve productivity by making simpler (less optimized) programs run faster**
- **Three communication optimizations:**
  - Overlap and coalescing of fine-grained accesses
  - Overlap of operations that use bulk put/get
  - Scheduling (reduce contention through pipelining) of bulk operations
  - Dynamic optimizations for irregular (a[b[i]]) accesses (implemented in Titanium rather than UPC)

Optimizing Fine-Grained Programs



Overlapping Bulk Communication



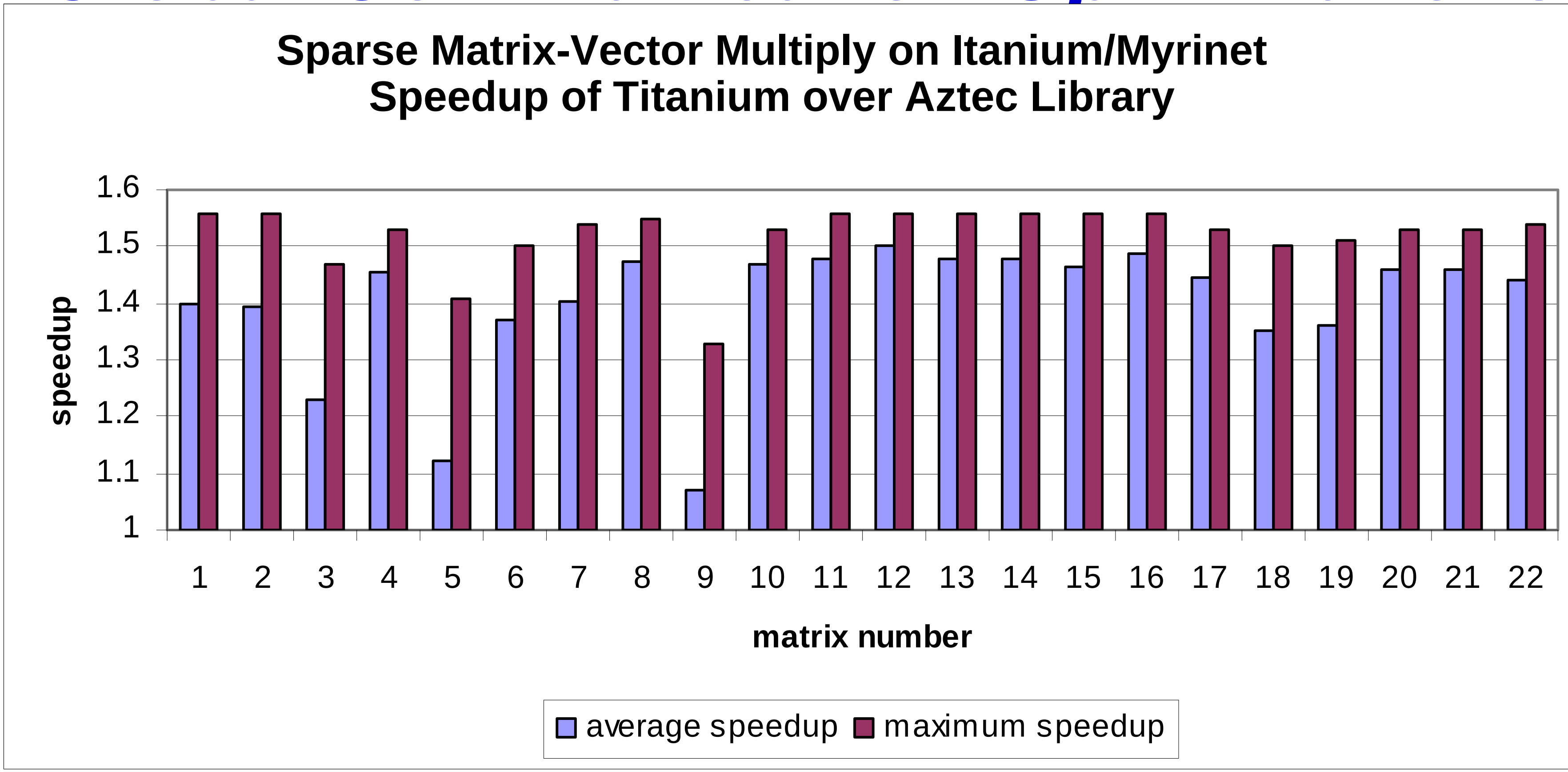
64-processor Infiniband cluster

Optimizations in Titanium

- **Communication optimizations are done**
- **Analysis in Titanium is easier than in UPC:**
  - Strong typing helps with alias analysis
  - Single analysis identifies global execution points that all threads will reach "together" (in same synch phase)
    - I.e., a barrier would be legal here
- **Allows global optimizations**
  - Convert remote reads to remote writes by other side
  - Perform global runtime analysis (inspector-executor)
  - Especially useful for sparse matrix code with indirection:

```
y [i] = … a[b[i]]
```

Global Communication Optimizations

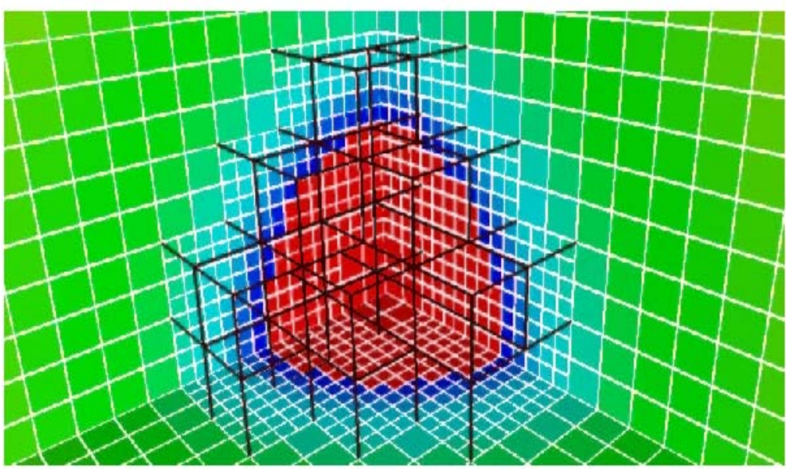


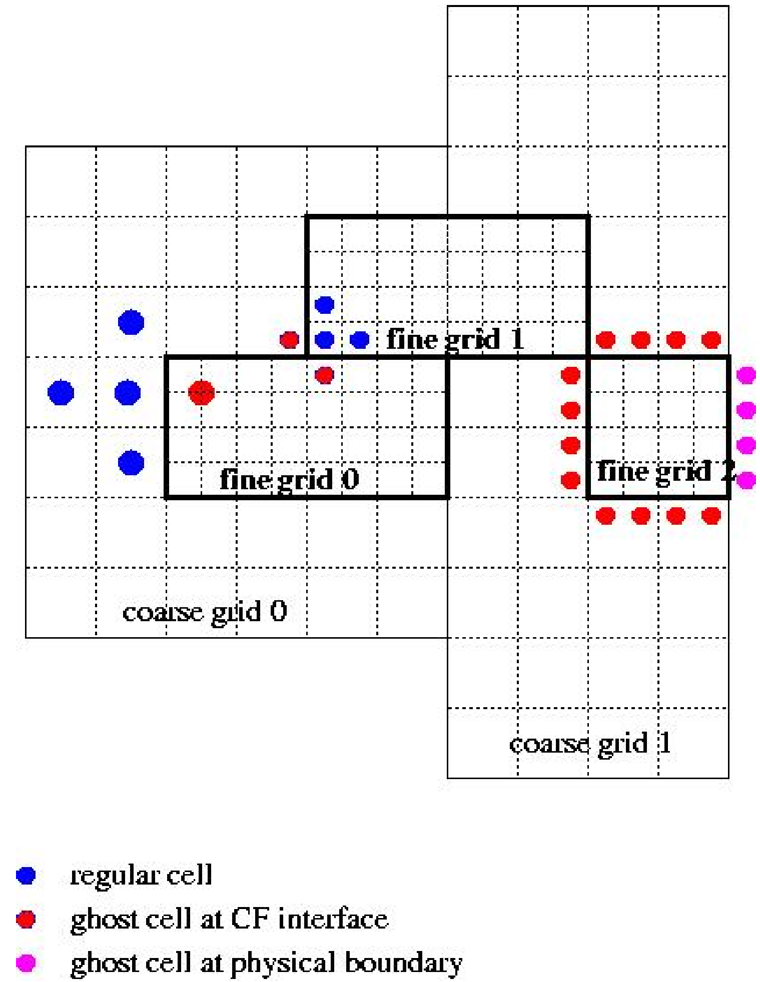
- Titanium code is written with fine-grained remote accesses
- Compile identifies legal “inspector” points
- Runtime selects (pack, bounding box) per machine / matrix / thread pair

PGAS Productivity

Coding Challenges: Block-Structured AMR

- **Adaptive Mesh Refinement (AMR) is challenging**
  - Irregular data accesses and control from boundaries
  - Mixed global/local view is useful





Titanium AMR benchmark available

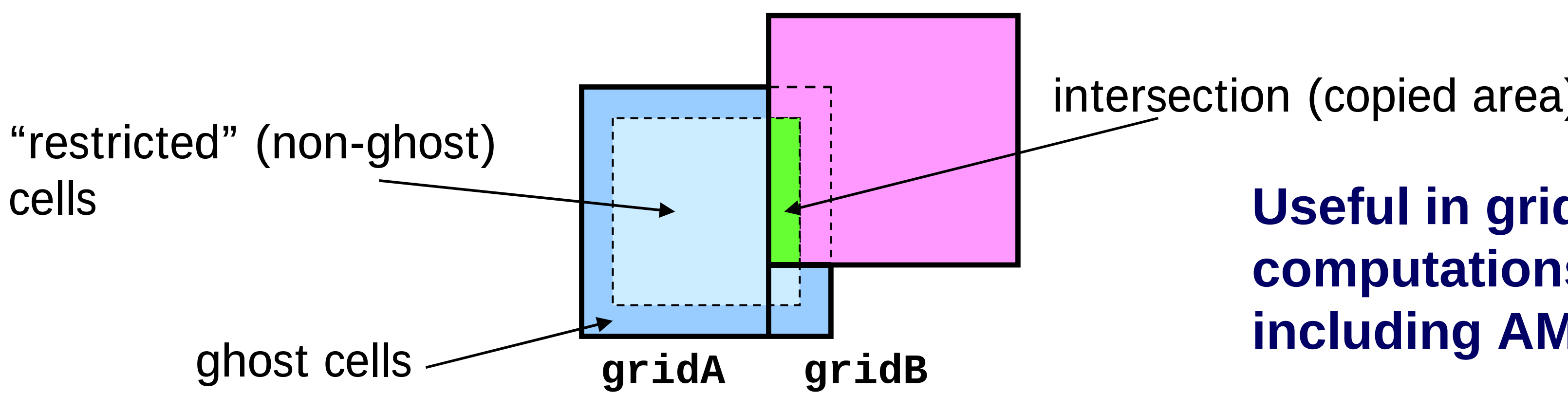
Arrays in a Global Address Space

- **Key features of Titanium arrays**
  - Generality: indices may start/end and any point
  - Domain calculus allow for slicing, subarray, transpose and other operations without data copies
- **Use domain calculus to identify ghosts and iterate:**

```
foreach (p in gridA.shrink(1).domain()) ...
```

- **Array copies automatically work on intersection**

```
gridB.copy(gridA.shrink(1));
```



Useful in grid computations including AMR

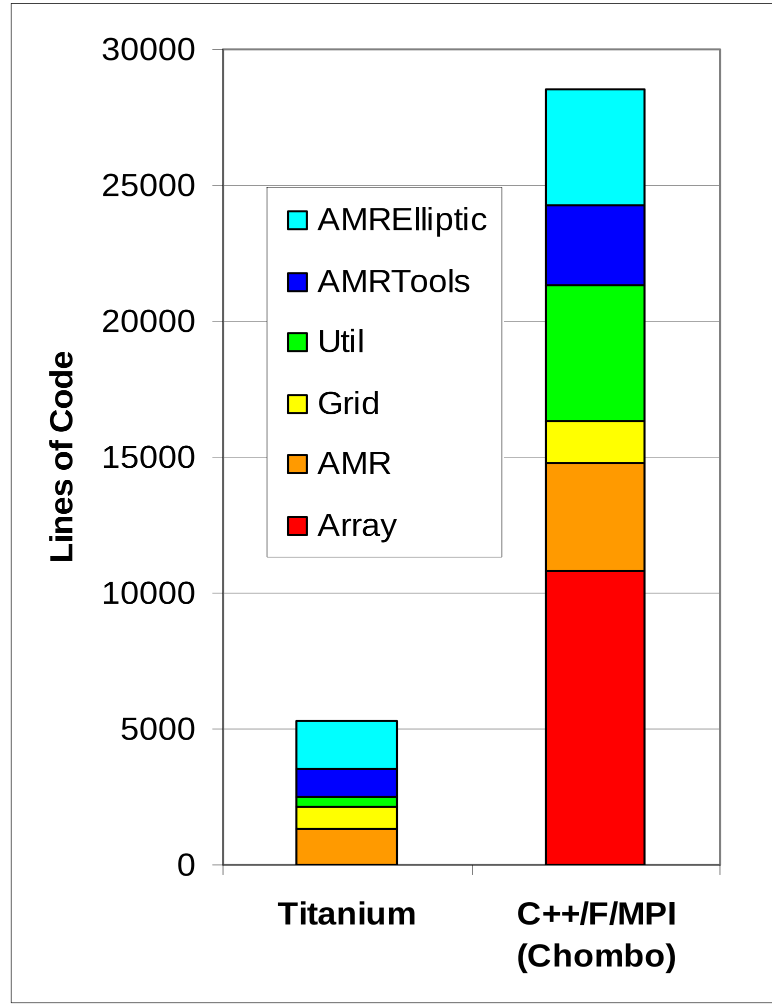
Languages Support Helps Productivity

C++/Fortran/MPI AMR

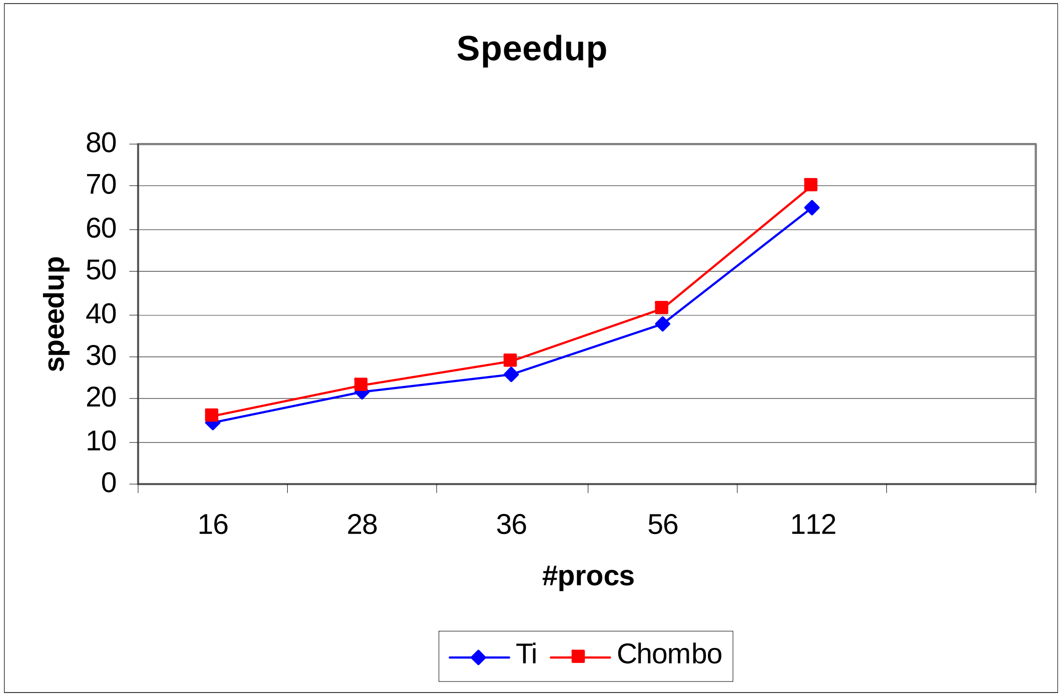
- **Chombo package from LBNL**
- **Bulk-synchronous comm:**
  - Pack boundary data between procs
  - All optimizations done by programmer

Titanium AMR

- **Entirely in Titanium**
- **Finer-grained communication**
  - No explicit pack/unpack code
  - Automated in runtime system
- **General approach**
  - Language allow programmer optimizations
  - Compiler/runtime does some automatically



Performance of Titanium AMR



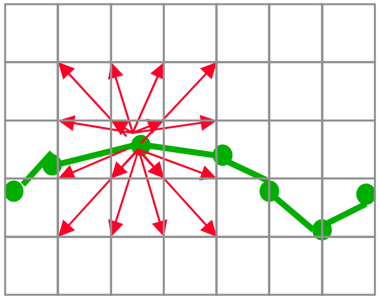
Comparable parallel performance

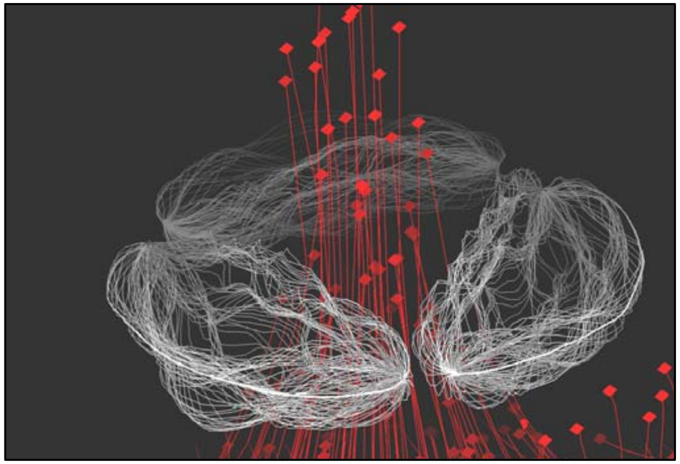
- **Serial: Titanium is within a few % of C++/F; sometimes faster!**
- **Parallel: Titanium scaling is comparable with generic optimizations**
  - **optimizations (SMP-aware) that are not in MPI code**
  - **additional optimizations (namely overlap) not yet implemented**

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

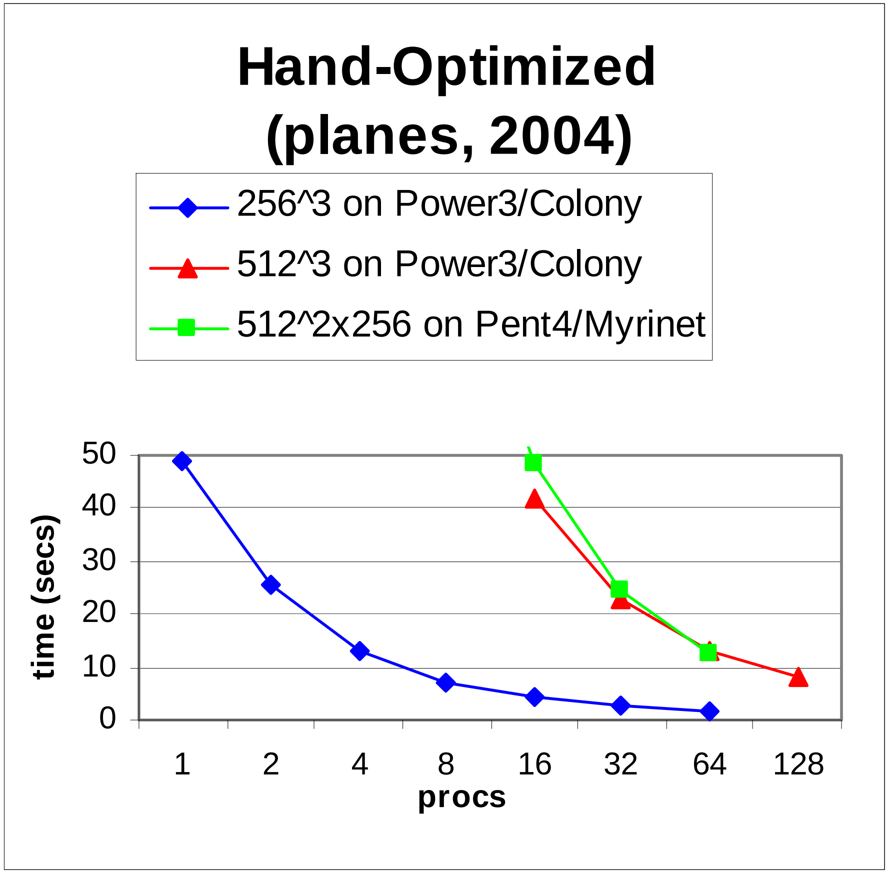


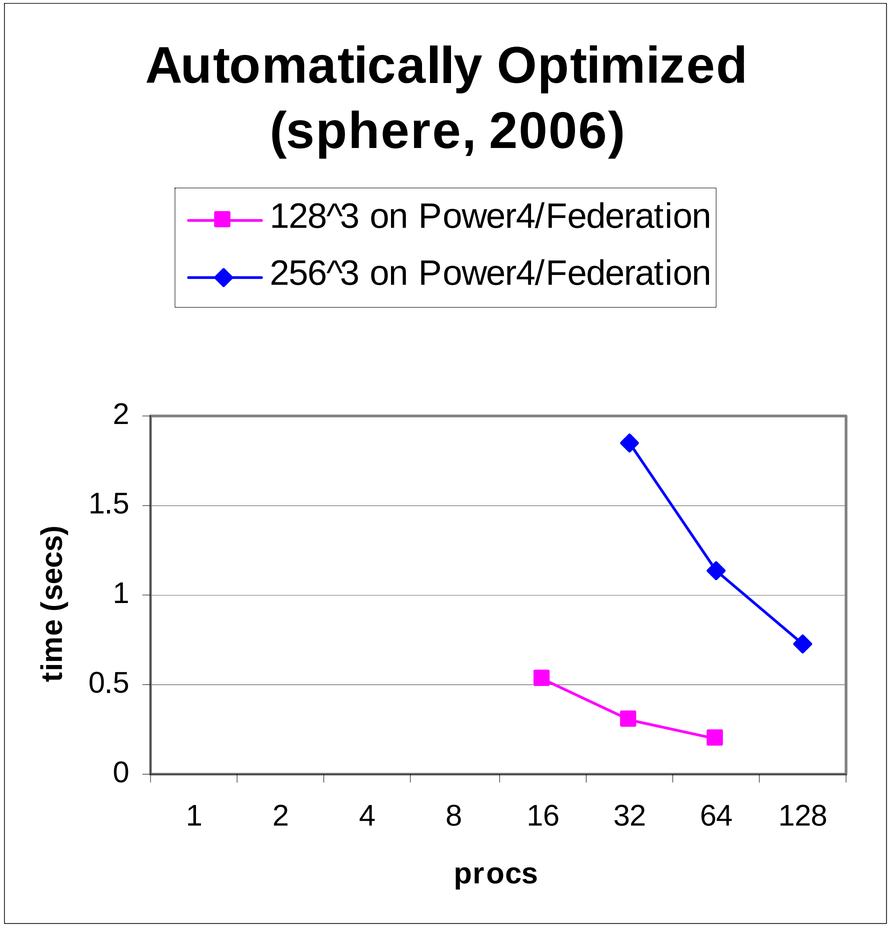


Code Size in Lines	
Fortran	**Titanium**
8000	**4000**

Note: Fortran code is not parallel

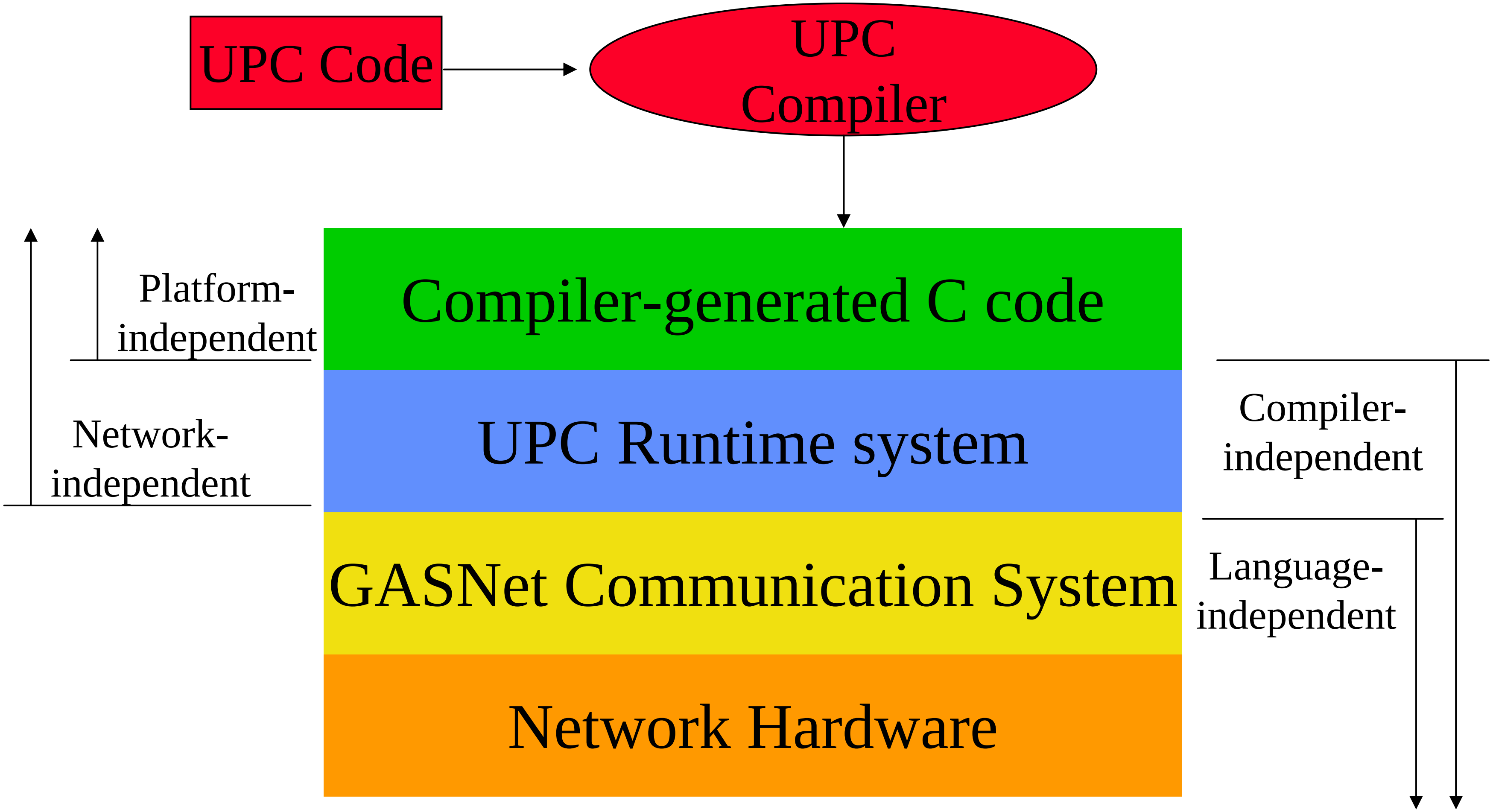
Immersed Boundary Method Performance



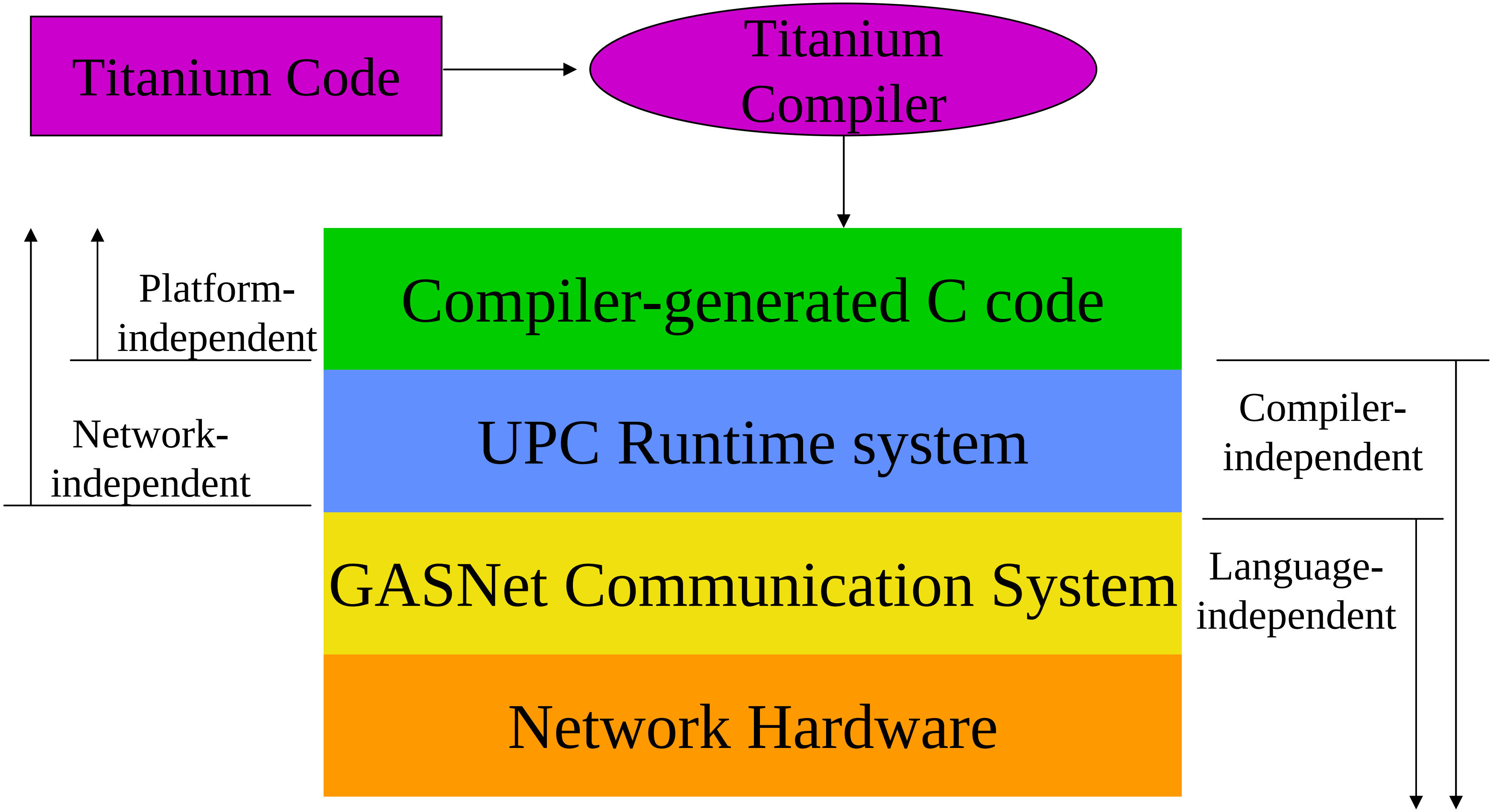


PGAS Portability

Titanium and Berkeley UPC Compiler



Titanium and Berkeley UPC Compiler



Berkeley UPC Compiler Portability

- **Portable, high-performance open-source UPC compiler**
  - **Fully UPC spec 1.2 compliant**
  - **Includes UPC collectives and UPC-I/O**
- **Many extensions for performance and programmability**
  - **Non-blocking and non-contiguous memcpy functions**
  - **Semaphores and signaling put**
  - **Fine granularity timers**
  - **Value-based collectives**
  - **Atomic memory operations**
  - **Hierarchical layout query**
  - **Call to/from MPI (C++, F, etc.)**
- **Entirely free & open source**
  - **Binary installer for Windows/Mac/UNIX** **http://upc.lbl.gov/download/**
  - **Source code download too**
  - **Remote compile server simplifies install**

Titanium Compiler Portability

- Portable, high-performance open-source Titanium compiler
  - Includes value-based collectives and bulk I/O
  - Support for checkpoint
- Many extensions for performance and programmability
  - Non-blocking array copy functions
  - Array copies do strided acceses
  - Hierarchical layout query
  - Call to MPI (C++, F, etc.)
- Entirely free & open source
  - http://titanium.cs.berkeley.edu/download/

Berkeley UPC and Titanium Portability

- Platform-independent generated code supports:
  - Network Hardware (supported through GASNet):
    - SMP, Myrinet, Quadrics Elan 3/4, Infiniband, IBM LAPI, Dolphin SCI, MPI, Ethernet, X1/Altix shmem (UPC only), Cray XT3 Portals (new, UPC only, Titanium soon)
    - BlueGene via MPI (working on native version)
  - Operating Systems:
    - Linux, Mac OSX, Windows/Cygwin, AIX, Solaris, IRIX, HPUX, FreeBSD, NetBSD, Tru64, Unicos, Catamount, CNL (new)
  - CPU / System Architecture:
    - Opteron, Itanium, x86, Athlon, Blue Gene, Cray XT3, X1, T3E, Alpha, PowerPC, MIPS, PA-RISC, SPARC, SX-6
- UPC-to-C Translator runs on Linux, Tru64, OSX, AIX
  - Opteron, x86, Itanium, PowerPC and Alpha
  - Seamless cross-compilation for other systems
    - using Berkeley internet translate server or your own

Recent Work on Extending the Language Model

(ongoing)

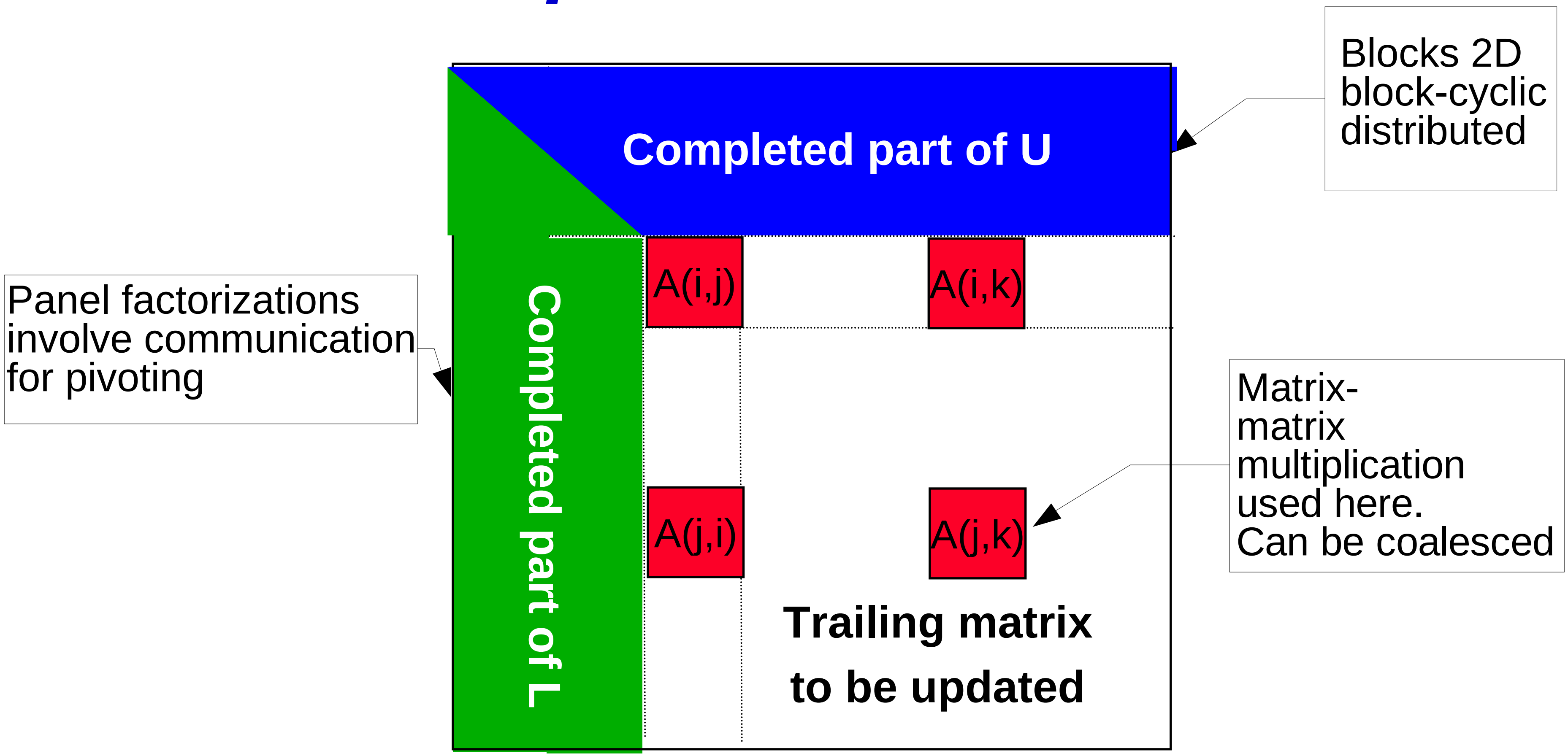
Beyond the SPMD Model: Mixed Parallelism

- **UPC and Titanium uses a static threads (SPMD) programming model**
  - General, performance-transparent
  - Criticized as “local view” rather than “global view”
    - “for all my array elements”, or “for all my blocks”
- **Adding extension for data parallelism**
  - Based on collective model:
    - Threads gang together to do data parallel operations
    - Or (from a different perspective) single data-parallel thread can split into P threads when needed
  - Compiler proves that threads are aligned at barriers, reductions and other collective points
    - Already used for global optimizations: read → writes transform
    - Adding support for other data parallel operations

Beyond the SPMD Model: Dynamic Threads

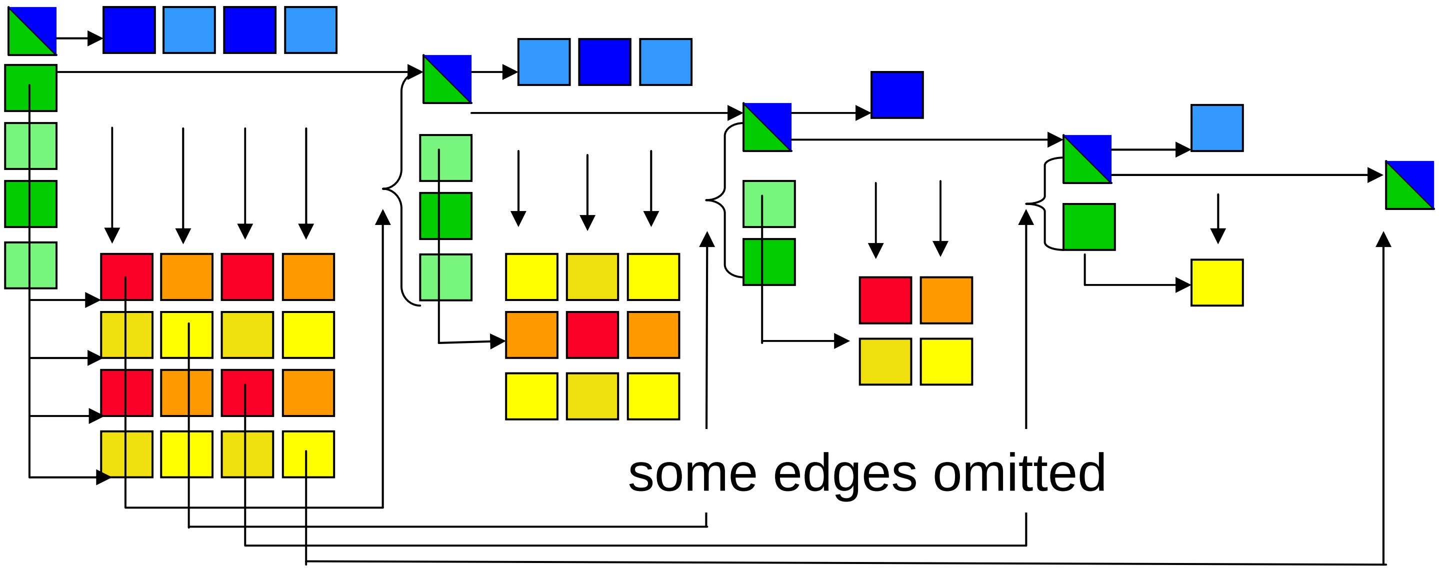
- **UPC uses a static threads (SPMD) programming model**
  - No dynamic load balancing built-in, although some examples (Delaunay mesh generation) of building it on top
  - Berkeley UPC model extends basic memory semantics (remote read/write) with active messages
  - AM have limited functionality (no messages except acks) to avoid deadlock in the network
- **A more dynamic runtime would have many uses**
  - Application load imbalance, OS noise, fault tolerance
- **Two extremes are well-studied**
  - Dynamic load balancing (e.g., random stealing) without locality
  - Static parallelism (with threads = processors) with locality
- **Charm++ has virtualized processes with locality**
  - How much “unnecessary” parallelism can it support?

Dense and Sparse Matrix Factorization



Panel being factored

Parallel Tasks in LU

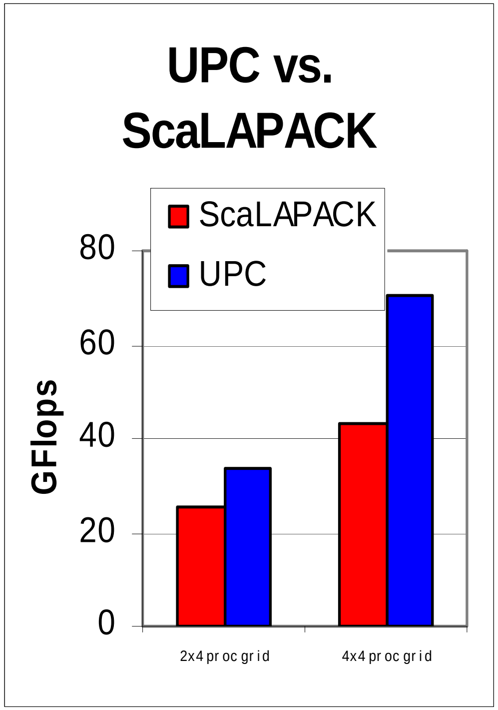


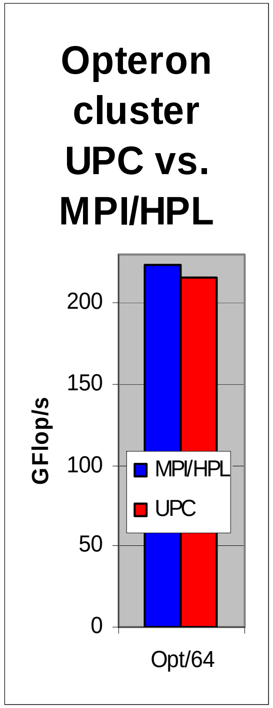
- **Theoretical and practical problem: Memory deadlock**
  - Not enough memory for all tasks at once. (Each update needs two temporary blocks, a green and blue, to run.)
  - If updates are scheduled too soon, you will run out of memory
  - If updates are scheduled too late, critical path will be delayed.

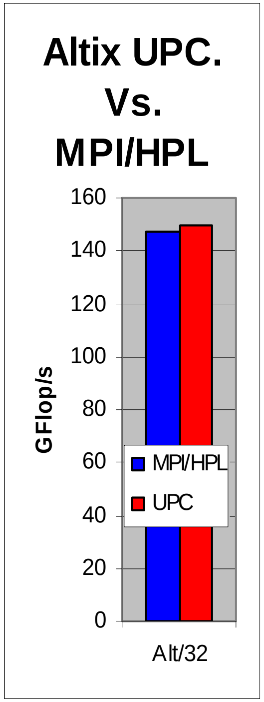
LU in UPC + Multithreading

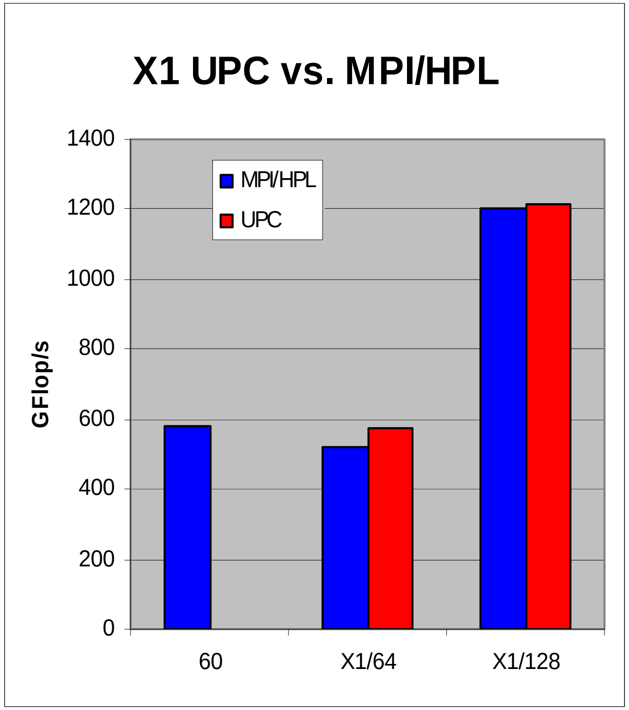
- **UPC uses a static threads (SPMD) programming model**
  - Multithreading used to mask latency and to mask dependence delays
  - Remote enqueue used to spawn remote threads
  - Three levels of threads:
    - UPC threads (data layout, each runs an event scheduling loop)
    - Multithreaded BLAS (boost 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**
  - Event-driven sparse ChoHard 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
  - Collectives (asynchronous reductions for pivoting) need high priority

UPC HP Linpack Performance









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

Conclusions and Future Plans

- **Current PGAS Languages**
  - Good fit for shared and distributed memory
  - Good control over locality
  - High productivity, especially in higher level Titanium
- **Role of optimizing compiler**
  - Language provides enough control for hand-optimizations (heroic compilers not needed)
  - Analysis and optimizations for productivity
  - Goal: allow for algorithm experimentation by users
- **Need to break out of strict SPMD model**
  - Load imbalance, OS noise, faults tolerance, etc.
  - Encapsulate LU techniques as language extension