Performance and Productivity Opportunities using Global Address Space Programming Models

Kathy Yelick
Lawrence Berkeley National Laboratory
and UC Berkeley

Joint work with

The Titanium Group: S. Graham, P. Hilfinger, P. Colella, D. Bonachea, K. Datta, E. Givelberg, A. Kamil, N. Mai, A. Solar, J. Su, T. Wen

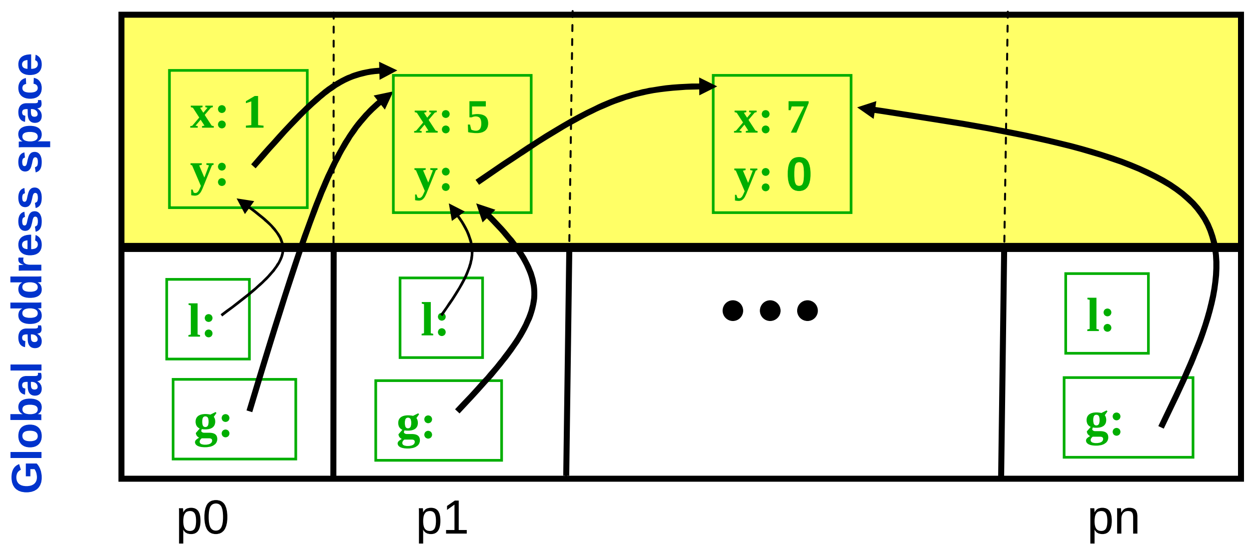
The Berkeley UPC Group: C. Bell, D. Bonachea, W. Chen, J. Duell, P. Hargrove, P. Husbands, C. Iancu, R. Nishtala, M. Welcome

Partitioned Global Address Space (PGAS) Languages

- **Productivity**
  - Global address space supports construction of complex shared data structures
  - High level constructs (e.g., multidimensional arrays) simplify programming
- **Performance**
  - PGAS Languages are Faster than two-sided MPI
  - Some surprising hints on performance tuning
  - Compilers can optimize parallel constructs
- **Portability**
  - These languages are nearly ubiquitous

Partitioned Global Address Space

- ***Global address space:*** **any thread/process may directly read/write data allocated by another**
- ***Partitioned:*** **data is designated as local (near) or global (possibly far); programmer controls layout**



By default:
- **Object heaps are shared**
- **Program stacks are private**

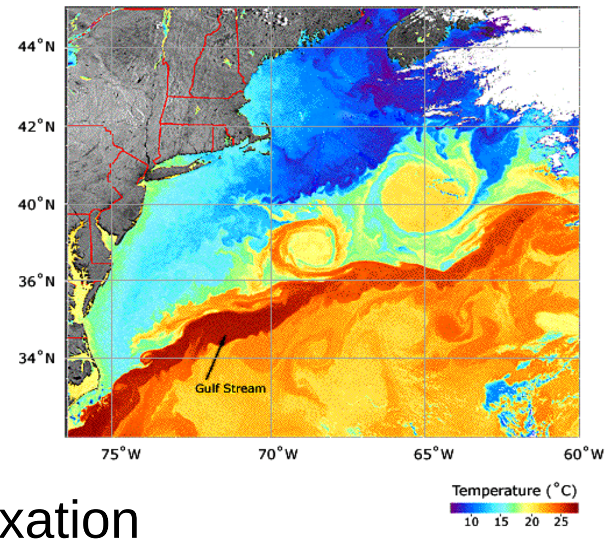
- ***3 Current languages:*** **UPC, CAF, and Titanium**
  - Emphasis in this talk on UPC & Titanium (based on Java)

PGAS Language Overview

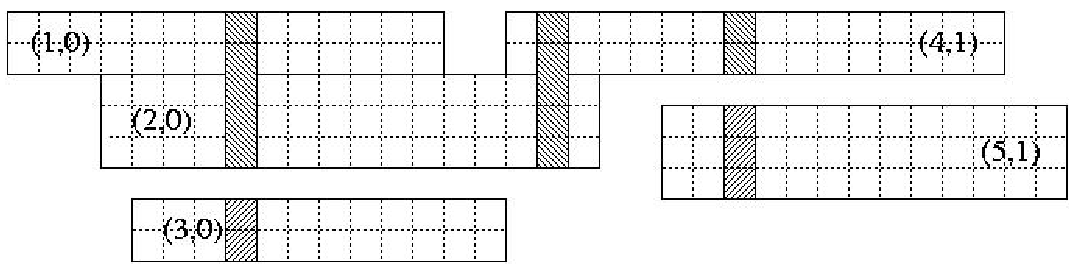
- **Many common concepts, although specifics differ**
  - Consistent with base language
- **Both private and shared data**
  - **int x[10];** and **shared int y[10];**
- **Support for distributed data structures**
  - Distributed arrays; local and global pointers/references
- **One-sided shared-memory 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 (split-phase in UPC), locks, memory fences
- **Collective Communication, IO libraries, etc.**
  - Overlapping, possibly non-blocking collectives

Case Study 1: AMR in Titanium

- Ocean modeling with AMR:
  - **Horizontal range from 10km (ocean) t .1 km (coast)**
  - **High grid aspect ratio horizontal to vertical**



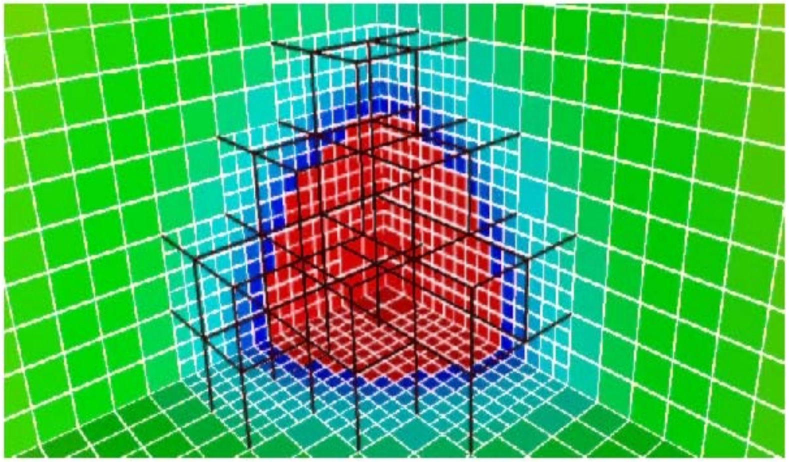
- For elliptic problems:
  - Multigrid algorithms remain the same
  - But point relaxation replaced by line relaxation

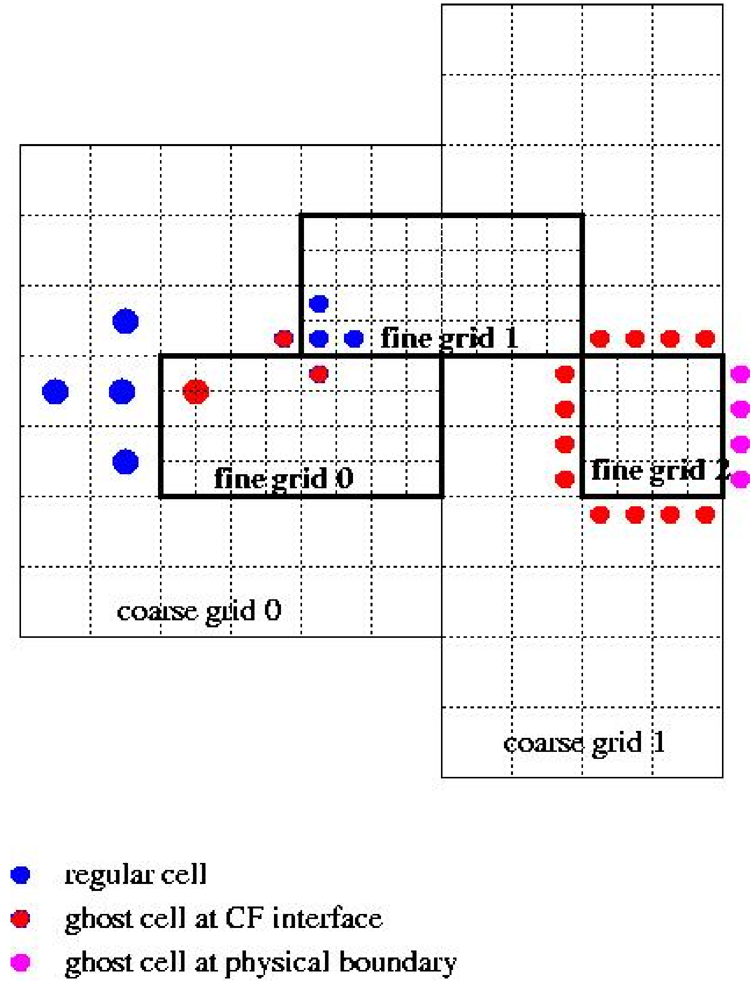


  - Developed fully in Titanium
  - Benchmark based on point-relaxation 3D Poisson

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

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++/Fortran/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

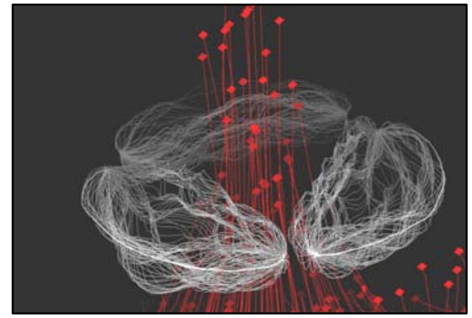
Elliptic PDE solver running time (secs)		
PDE Solver Time (secs)	C++/Fortran/MPI	Titanium
Serial, Opteron	57	53
Parallel, SP3 (28 procs)	113	112

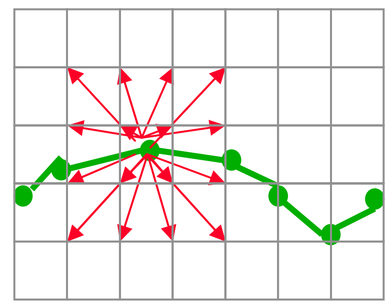
10X reduction in lines of code!

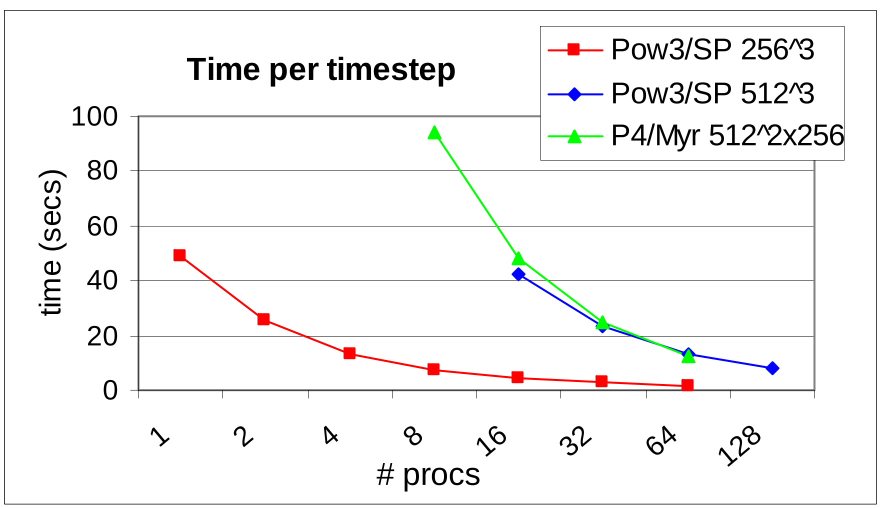
Comparable running time (both being tuned)

Immersed Boundary Simulation in Titanium

- **Modeling elastic structures in an incompressible fluid.**
  - Blood flow in the heart, blood clotting, inner ear, embryo growth, and many more
- **Complicated parallelization**
  - Particle/Mesh method
  - "Particles" connected into materials



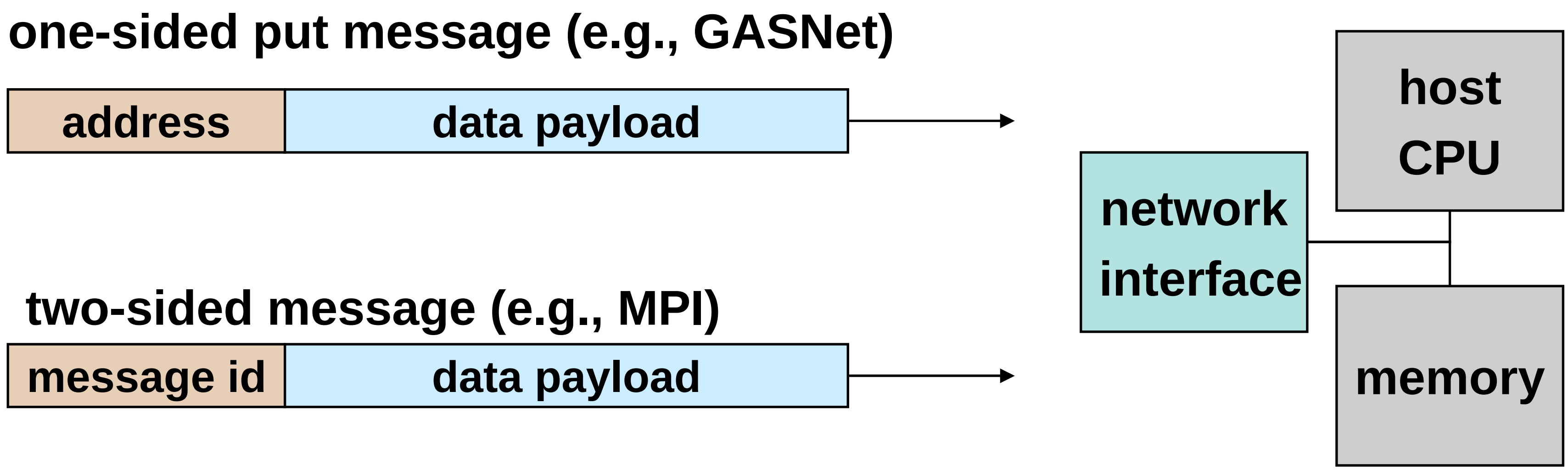




Code Size in Lines	
Fortran	**Titanium**
8000	**4000**

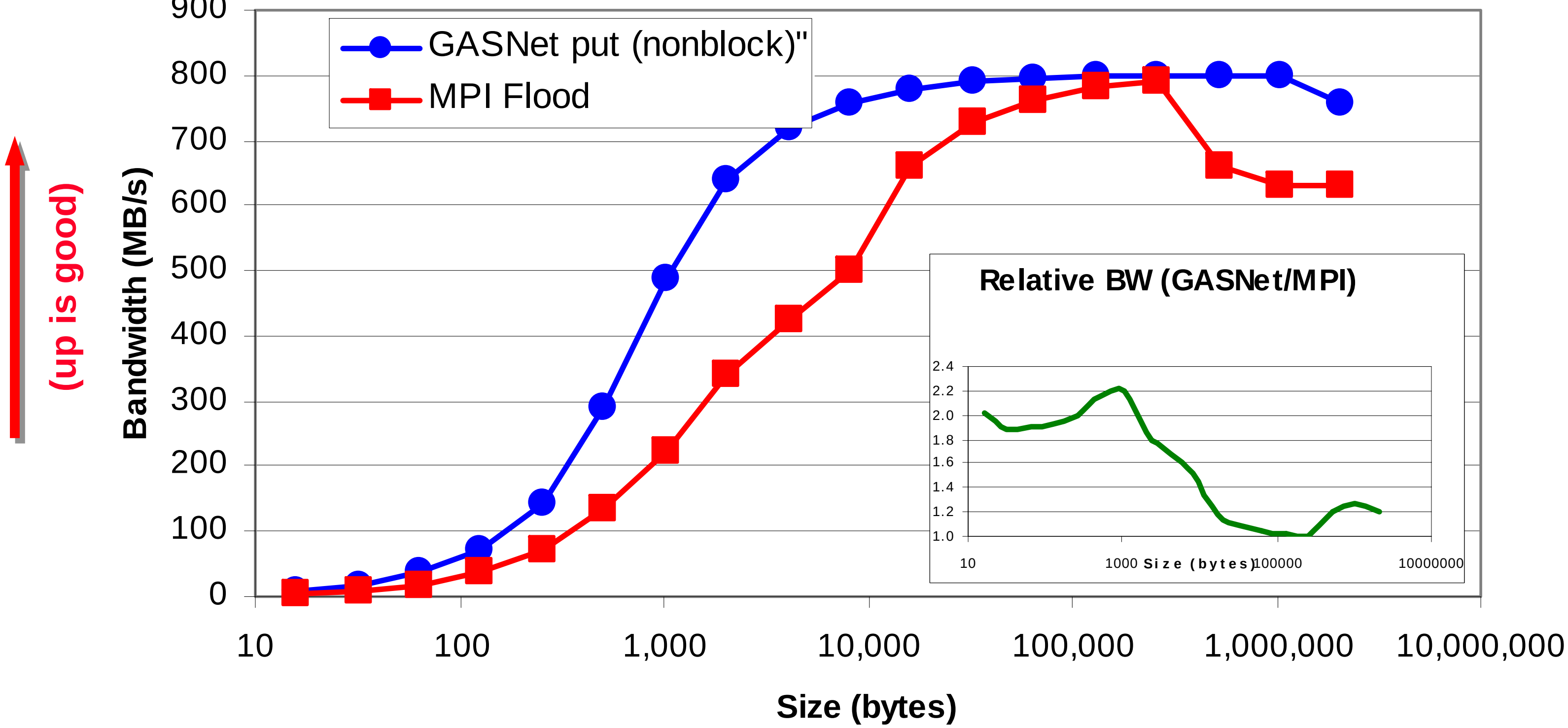
No support for parallelism in the Fortran code

One-Sided vs Two-Sided



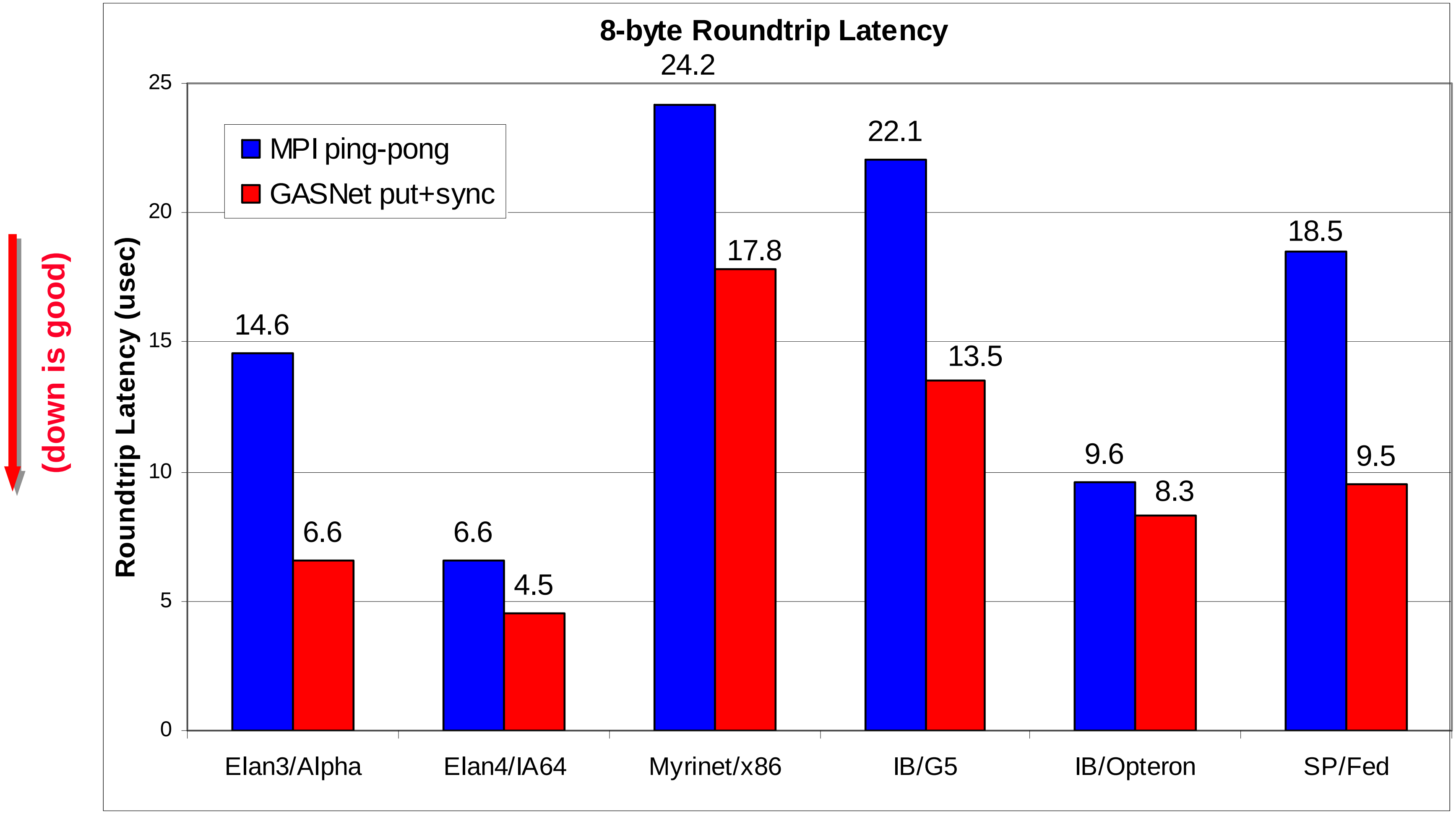
- **A one-sided put/get message can be handled directly by a network interface with RDMA support**
  - Avoid interrupting the CPU or storing data from CPU (preposts)
- **A two-sided messages needs to be matched with a receive to identify memory address to put data**
  - Offloaded to Network Interface in networks like Quadrics
  - Need to download match tables to interface (from host)
- **MPI has added costs associated with ordering to make it usable as a end-user programming model**

Performance Advantage of One-Sided Communication: GASNet vs MPI



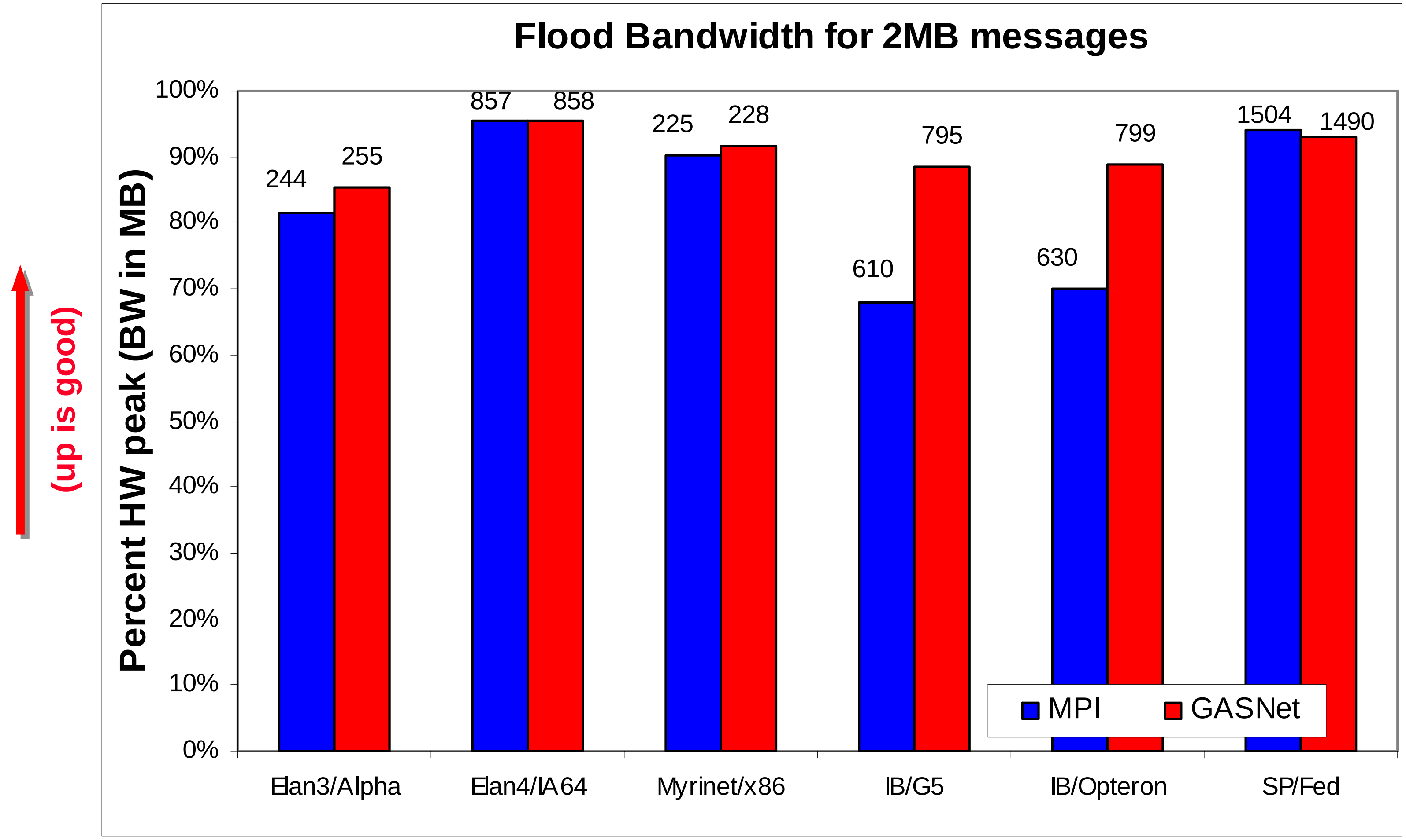
- **Opteron/InfiniBand (Jacquard at NERSC):**
  - GASNet's vapi-conduit and OSU MPI 0.9.5 MVAPICH
- **Half power point (N ½) differs by *one order of magnitude***
- ***Note: this is a very good MPI implementation!!***

GASNet: Portability and *High-Performance*



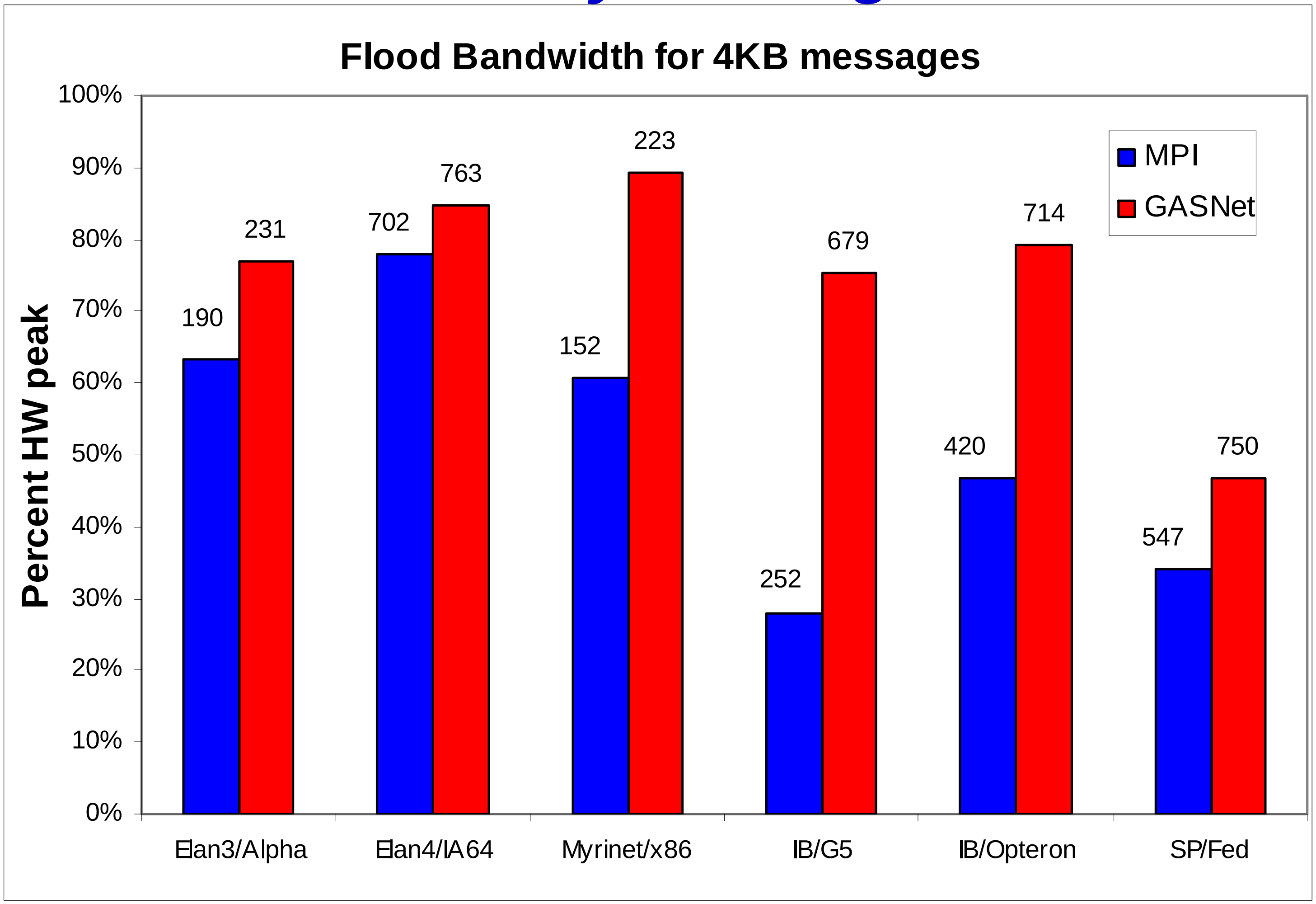
GASNet better for latency across machines

GASNet: Portability and *High-Performance*



GASNet at least as high (comparable) for large messages

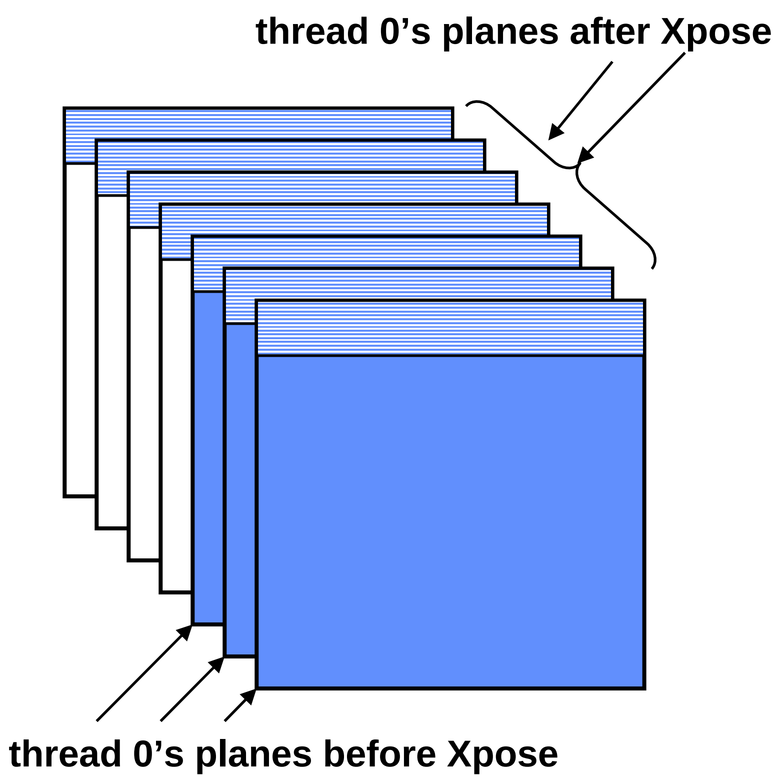
GASNet: Portability and *High-Performance*



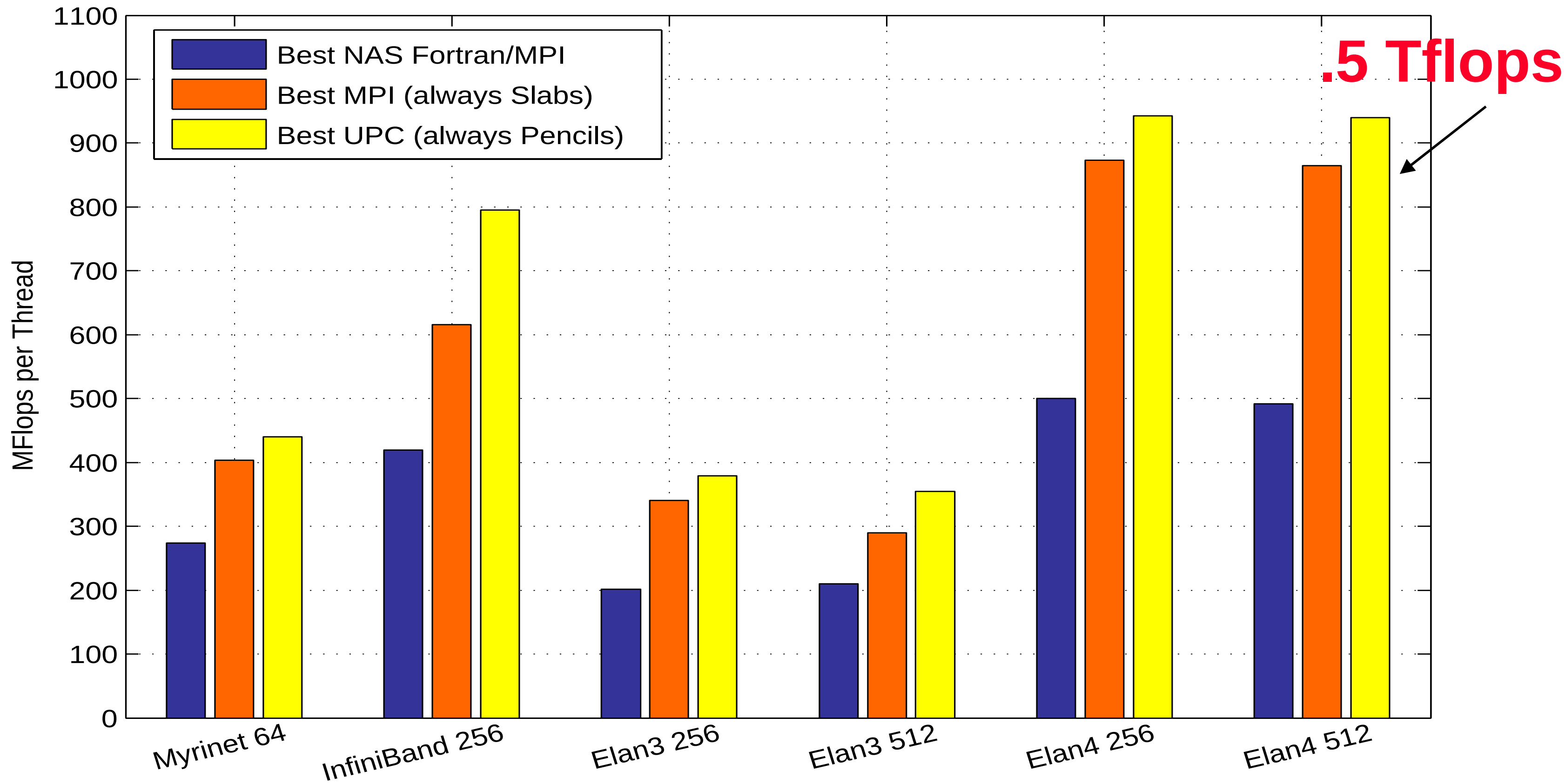
GASNet excels at mid-range sizes: important for overlap

Case Study 2: NAS FT

- **Performance of Exchange (Alltoall) is critical**
  - 1D FFTs in each dimension, 3 phases
  - Transpose after first 2 for locality
  - Bisection bandwidth-limited
    - Problem as #procs grows
- **Three implementations:**
  - **Exchange:**
    - wait for 2nd dim FFTs to finish, send 1 message per processor pair
  - **Slab:**
    - wait for chunk of rows destined for 1 proc, send when ready
  - **Pencil:**
    - send each row as it completes



NAS FT Variants Performance Summary



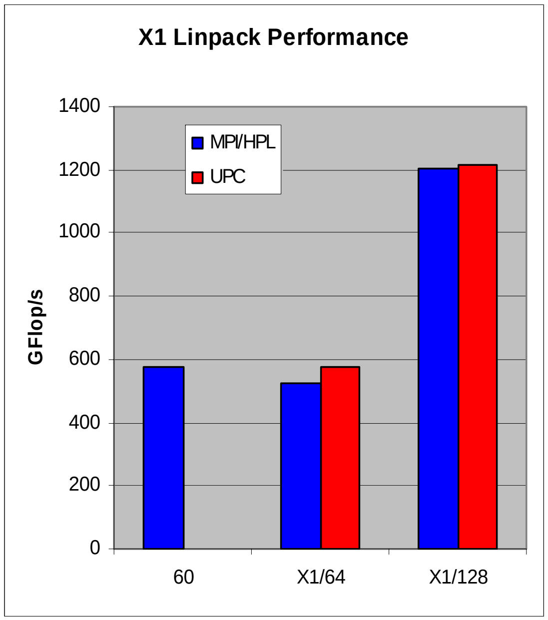
- **Slab is always best for MPI; small message cost too high**
- **Pencil is always best for UPC; more overlap**

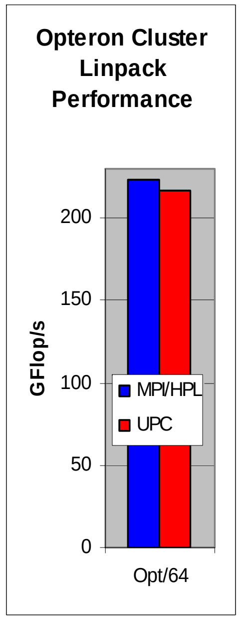
Case Study 3: LU Factorization

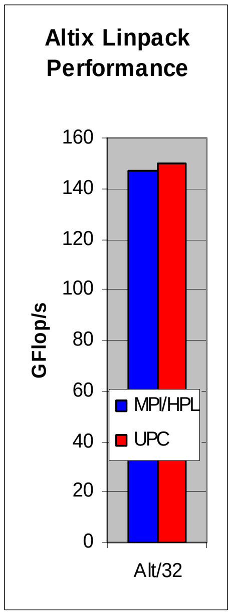
- **Direct methods have complicated dependencies**
  - Especially with pivoting (unpredictable communication)
  - Especially for sparse matrices (dependence graph with holes)
- **LU Factorization in UPC**
  - Use overlap ideas and multithreading to mask latency
  - Multithreaded: UPC threads + user threads + threaded BLAS
    - Panel factorization: Including pivoting
    - Update to a block of U
    - Trailing submatrix updates
- **Status:**
  - Dense LU done: HPL-compliant
  - Sparse version underway

UPC HPL Performance

- **Comparison to High Performance Linpack**







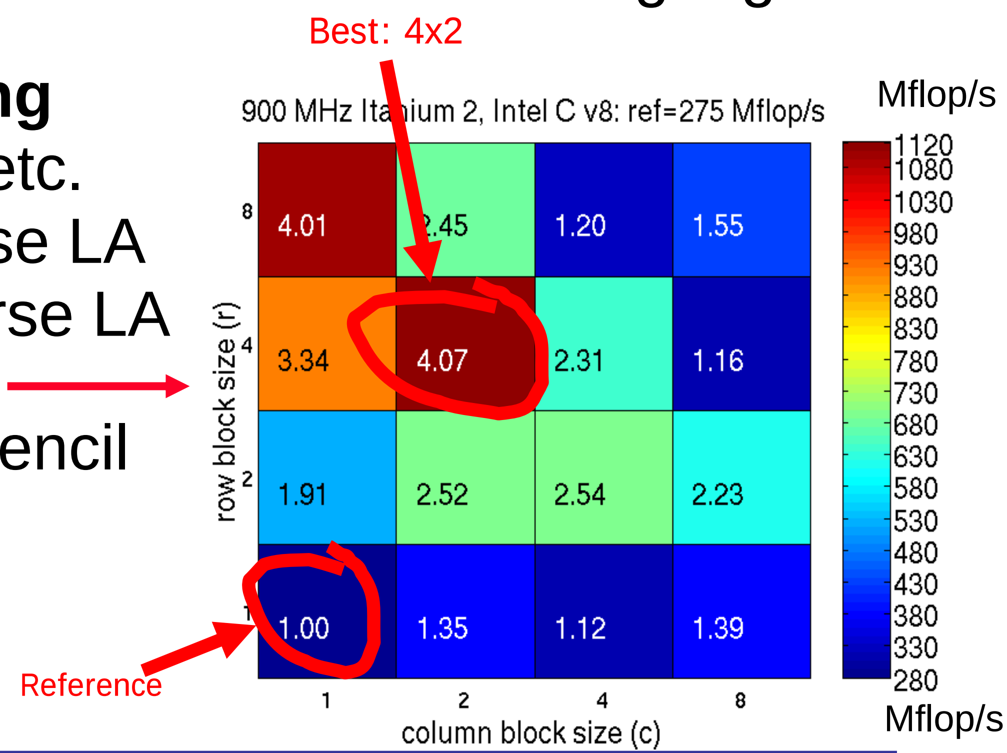
- **MPI HPL numbers from HPCC database**
- **Large scaling:**
  - 2.2 TFlops on 512p,
  - 4.4 TFlops on 1024p (Thunder)
- ***UPC is ½ the code size***

- **Comparison to ScaLAPACK on an Altix**
  - 2 x 4 process grid (best of several block sizes for both versions)
    - ScaLAPACK **25.25** GFlop/s (block size 64)
    - UPC LU **33.60** GFlop/s (block size 256)
  - 4x4 process grid
    - ScaLAPACK - **43.34 GFlop/s** (block size = 64)
    - UPC - **70.26 Gflop/s** (block size = 200)

Joint work with Parry Husbands

What About Serial Performance?

- **In general, UPC and Titanium serial performance are comparable to C**
  - Differences between tuning effort and C/Fortran compilers are more significant than an overhead from languages
- **Strategy: Empirical tuning**
  - *FFTW & Spiral*: FFTs, etc.
  - *Atlas (& PHiPAC):* dense LA
  - *OSKI (& Sparsity):* sparse LA
    - NASA example shown
  - Currently working on stencil optimizations
- **Not currently tied to PGAS languages**



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**
  - Support Operating Systems:
    - Linux, FreeBSD, Tru64, AIX, IRIX, HPUX, Solaris, Cygwin, MacOSX, Unicos, SuperUX
    - UPC translator somewhat less portable: we provide a http-based compile server
  - Supported CPUs:
    - x86, Itanium, Alpha, Sparc, PowerPC, PA-RISC, Opteron
  - GASNet communication:
    - Myrinet GM, Quadrics Elan, Mellanox Infiniband VAPI, IBM LAPI, Cray X1, SGI Altix, Cray/SGI SHMEM, and (for portability) MPI and UDP
  - Specific supercomputer platforms:
    - HP AlphaServer, Cray X1, IBM SP, NEC SX-6, Cluster X (Big Mac), SGI Altix 3000
    - In progress: Cray XT3, BG/L (both run over MPI)
- **Can be mixed with MPI, C/C++, Fortran**

Portability of PGAS Languages

Other compilers also exist for PGAS Languages

- **UPC**
  - Gcc/UPC by Intrepid: runs on GASNet
  - HP UPC for AlphaServers, clusters, …
  - MTU UPC uses HP compiler on MPI (source to source)
  - Cray UPC
- **Co-Array Fortran:**
  - Cray CAF Compiler: X1, X1E
  - Rice CAF Compiler (on ARMCI or GASNet), John Mellor-Crummey
    - Source to source
    - Processors: Pentium, Itanium2, Alpha, MIPS
    - Networks: Myrinet, Quadrics, Altix, Origin, Ethernet
    - OS: Linux32 RedHat, IRIS, Tru64

NB: source-to-source requires cooperation by backend compilers

Summary

- **PGAS languages offer performance advantages**
  - Expose best-possible network performance
    - Shared memory on machines like SGI Altix
    - Remote load/store on GAS hardware like Cray X1 and Quadrics
    - Remote load/store with "registration" on Infiniband, Myrinet
  - Smaller messages may be faster:
    - make better use of network: postpone bisection bandwidth pain
    - can also prevent cache thrashing for packing
- **PGAS languages offer productivity advantage**
  - Order of magnitude in line counts for grid-based code in Titanium
  - Push decisions about packing/not into runtime for portability (advantage of language with translator vs. library approach)
- **Source-to-source translation**
  - The way to ubiquity
  - Complement highly tuned machine-specific compilers