Architectural Trends and Programming Model Strategies for Large-Scale Machines

Katherine Yelick

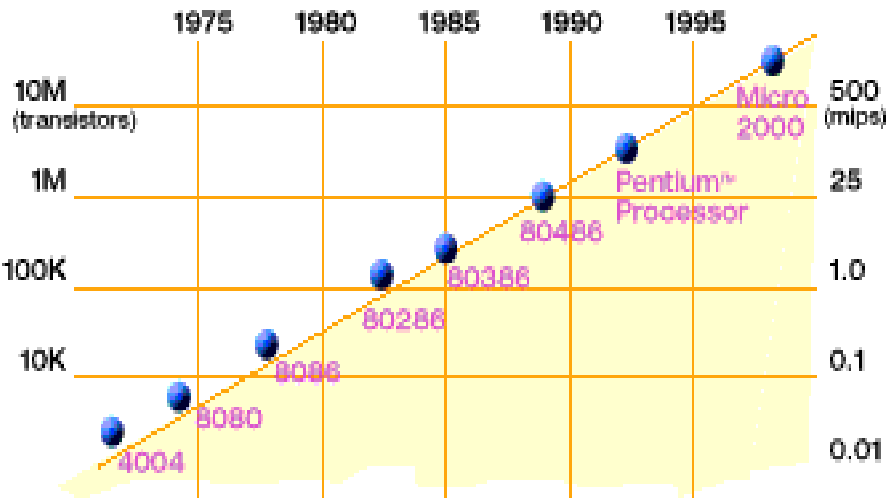
U.C. Berkeley and Lawrence Berkeley National Lab

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

Architecture Trends

- **What happened to Moore's Law?**
- **Power density and system power**
- **Multicore trend**
- **Game processors and GPUs**
- **What this means to you**

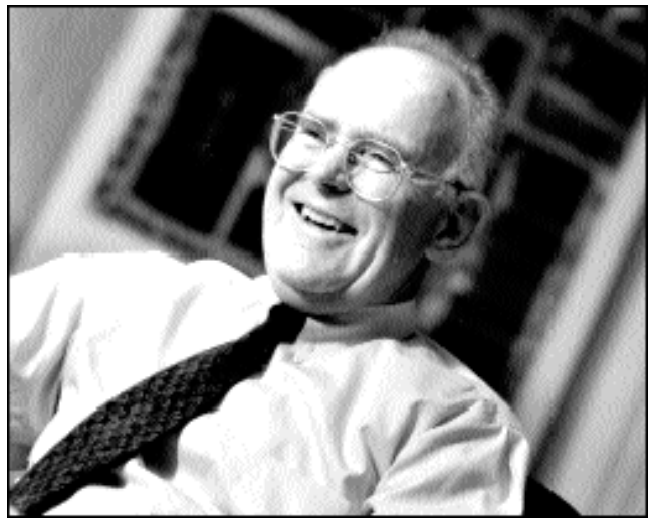
Moore's Law is Alive and Well



2X transistors/Chip Every 1.5 years

Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.



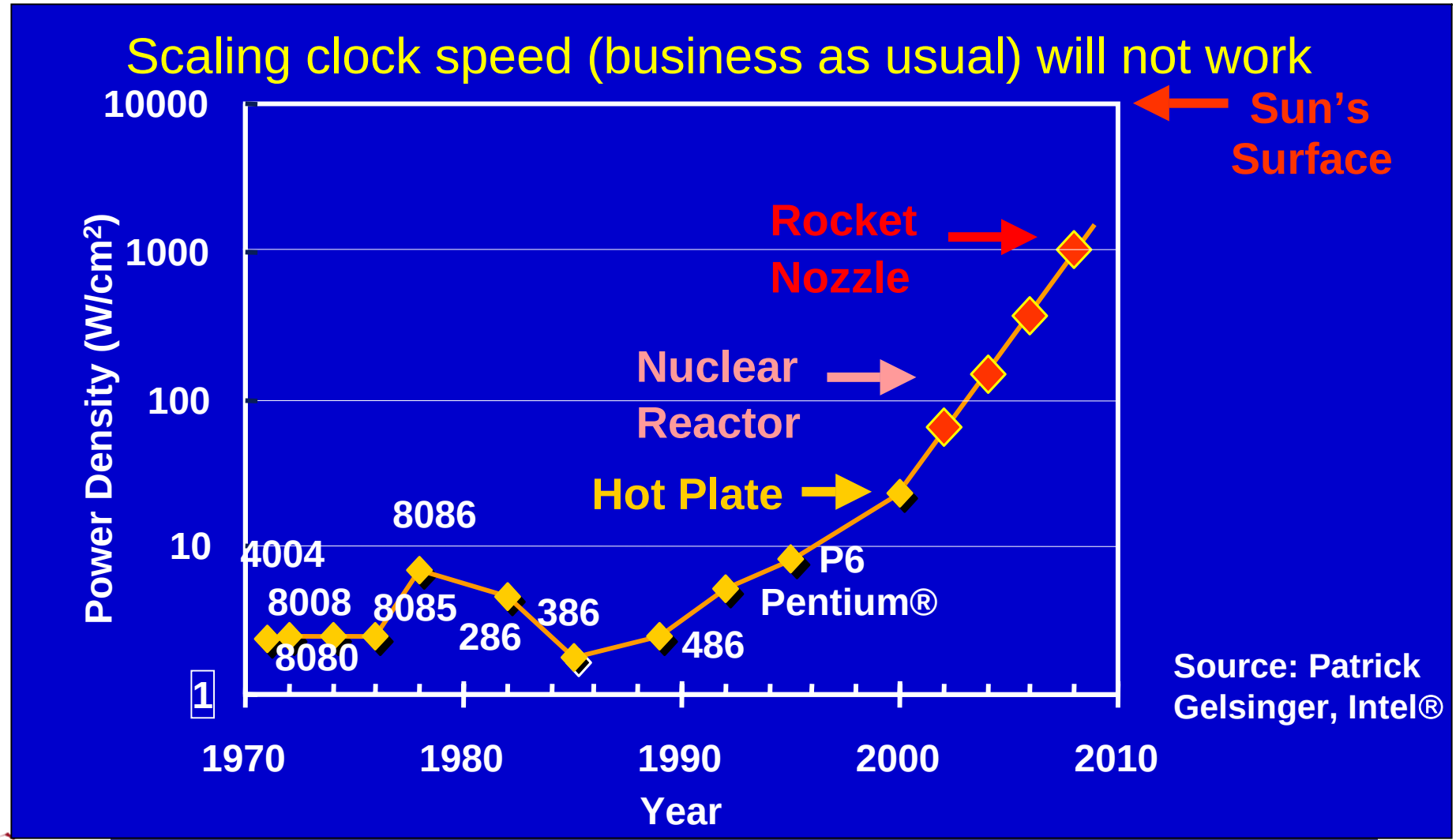
Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

But Clock Scaling Bonanza Has Ended

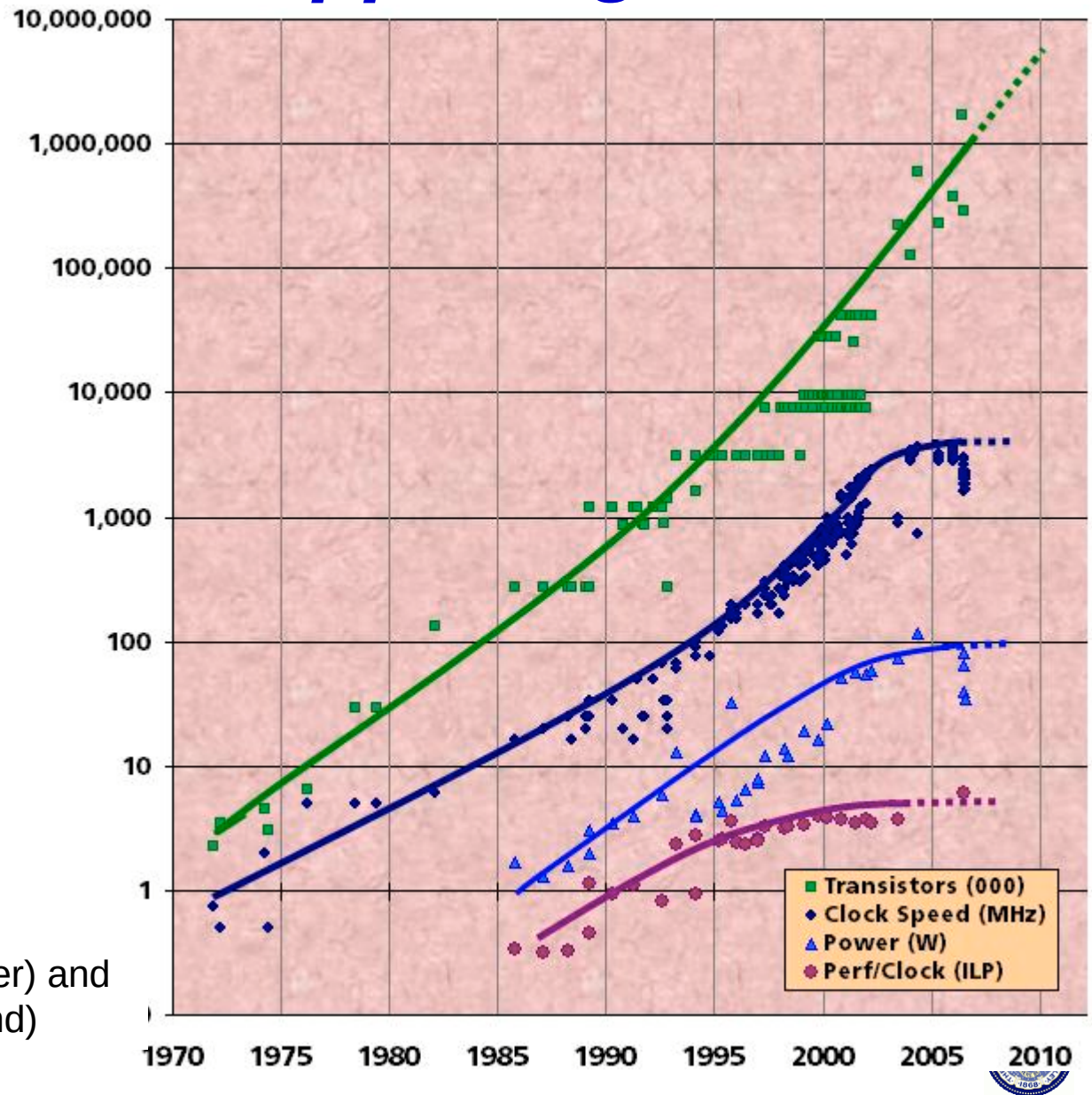
- **Processor designers forced to go "multicore":**
  - Heat density: faster clock means hotter chips
    - more cores with lower clock rates burn less power
  - Declining benefits of "hidden" Instruction Level Parallelism (ILP)
    - Last generation of single core chips probably over-engineered
    - 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

Clock Scaling Hits Power Density Wall



Revolution is Happening Now

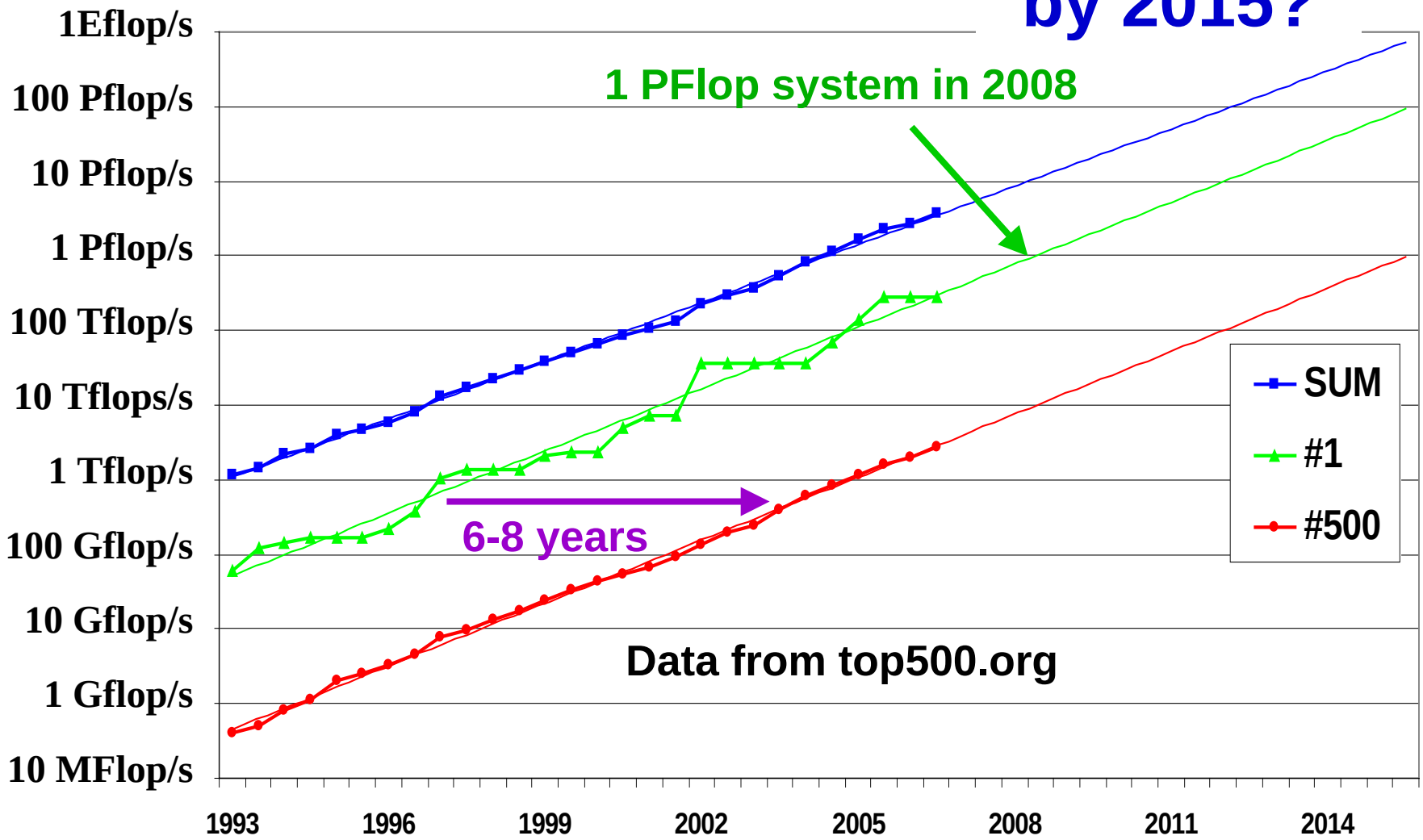
- **Chip density is continuing increase ~2x every 2 years**
  - Clock speed is not
  - Number of processor cores may double instead
- **There is little or no hidden parallelism (ILP) to be found**
- **Parallelism must be exposed to and managed by software**



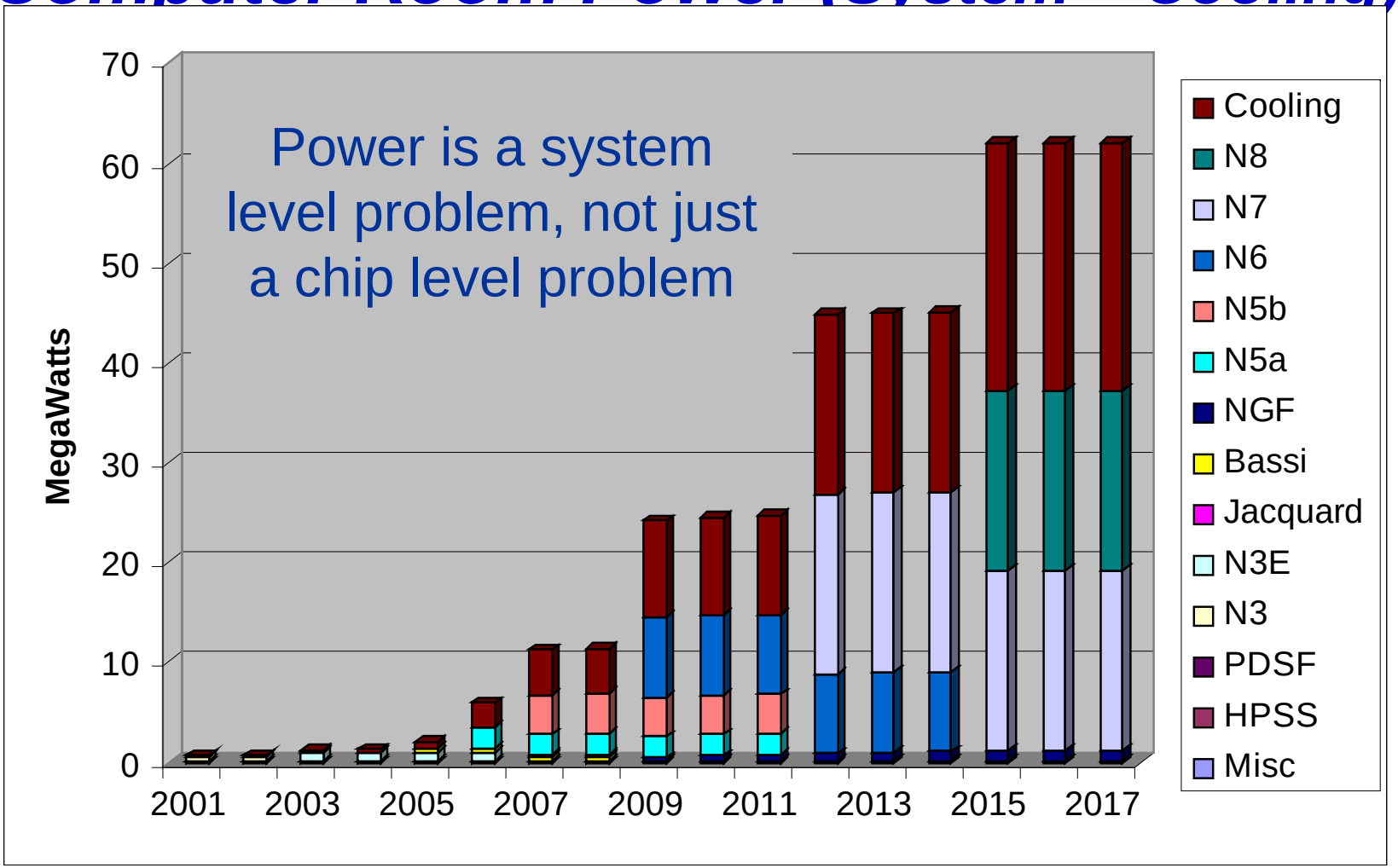
Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)

Petaflop with ~1M Cores

Common by 2015?



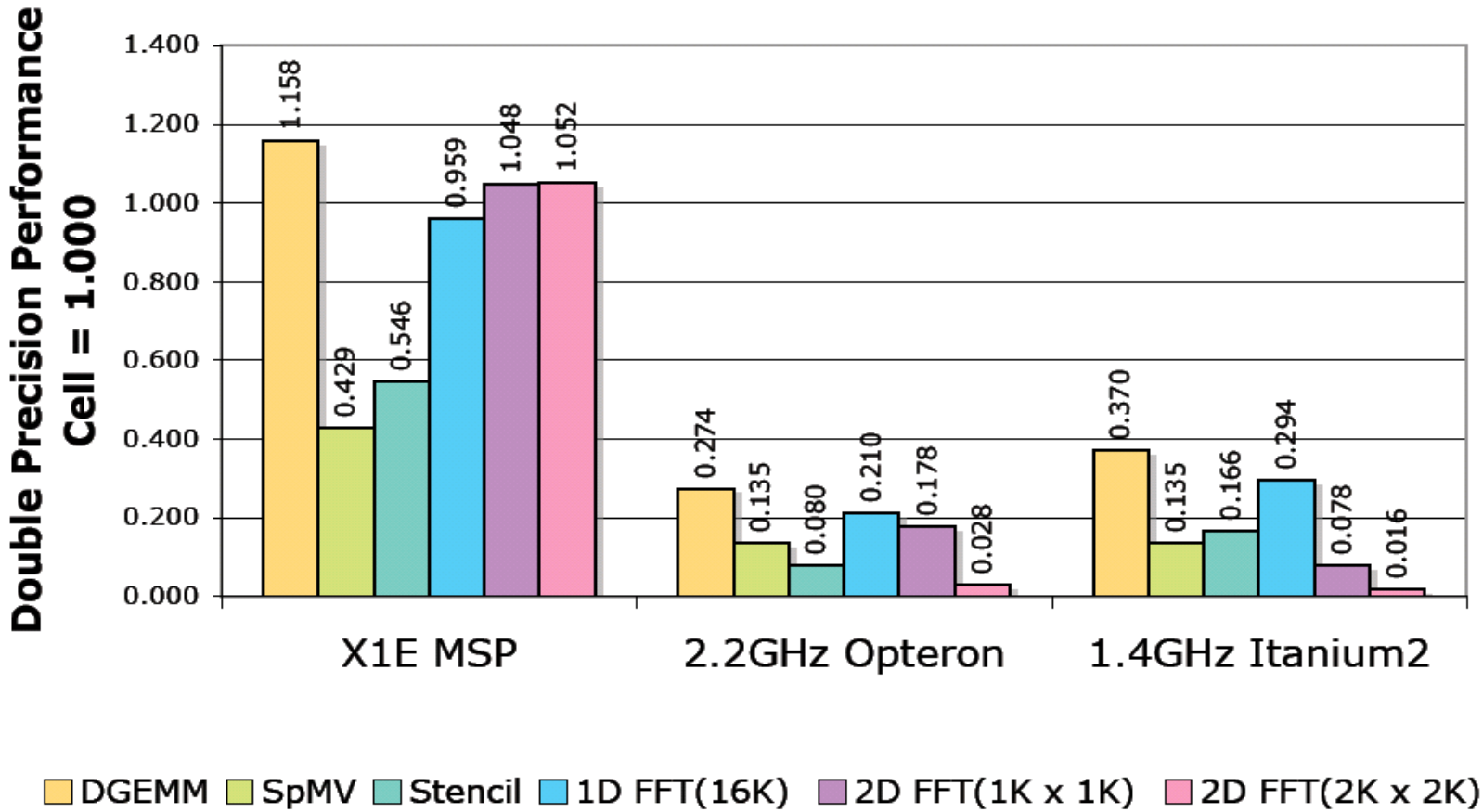
NERSC 2005 Projections for Computer Room Power (System + Cooling)



Concurrency for Low Power

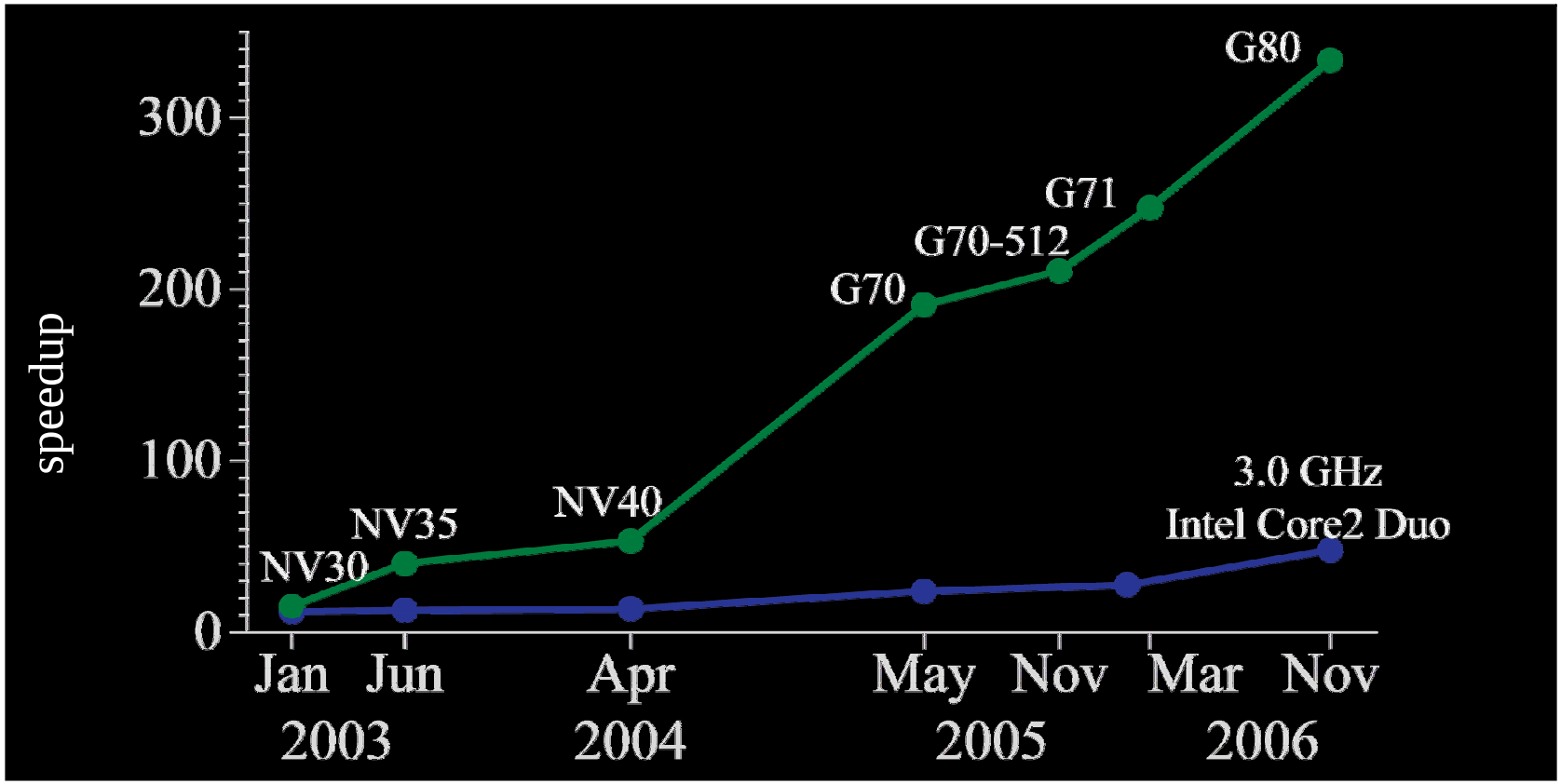
- **Highly concurrent systems are more power efficient**
  - *Dynamic power is proportional to V^2fC*
  - *Increasing frequency (f) also increases supply voltage (V): more than linear effect*
  - *Increasing cores increases capacitance (C) but has only a linear effect*
- **Hidden concurrency burns power**
  - Speculation, dynamic dependence checking, etc.
  - Push parallelism discover to software (compilers and application programmers) to save power
- **Challenge: *Can you double the concurrency in your algorithms every 18 months?***

Cell Processor (PS3) on Scientific Kernels



- Very hard to program: explicit control over memory hierarchy

Game Processors Outpace Moore's Law



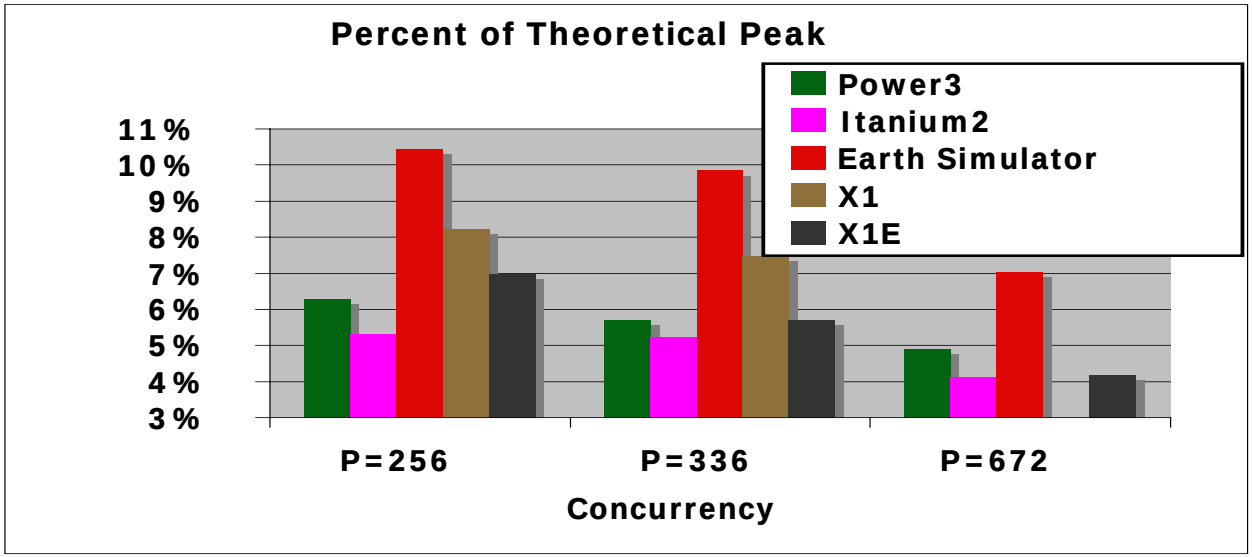
- Traditionally too specialized (no scatter, no inter-core communication, no double fp) but trend to generalize
- Still have control over memory hierarchy/partitioning

What Does this Mean to You?

- **The world of computing is going parallel**
  - Number of cores will double every 12-24 months
  - Petaflop (1M processor) machines common by 2015
- **More parallel programmers to hire ☺**
- **Climate codes must have more parallelism ☹**
  - Need for fundamental rewrite and new algorithms
  - Added challenge of combining with adaptive algorithms
- **New programming model or language likely ☺**
  - Can the HPC benefit from investments in parallel languages and tools?
  - One programming model for all?

Performance on Current Machines

Oliker, Wehner, Mirin, Parks and Worley



- **Current state-of-the-art systems attain around 5% of peak at the highest available concurrencies**
  - Note current algorithm uses OpenMP when possible to increase parallelism
- **Unless we can do better, peak performance of system must be 10-20x of sustained requirement**
- **Limitations (from separate studies of scientific kernels)**
  - Not just memory bandwidth: latency, compiler code generation, ..

Strawman 1km Climate Computer

Oliker, Shalf, Wehner

- **Cloud system resolving global atmospheric model**
  - .015^{o}X.02^{o} horizontally with 100 vertical levels
- **Caveat: A *back-of-the-envelope* calculation, with many assumptions about scaling current model**
- **To acheive simulation time 1000x faster than real time:**
  - ~10 Petaflops sustained performance requirement
  - ~100 Terabytes total memory
  - ~2 million horizontal subdomains
  - ~10 vertical domains
  - ~20 million processors at 500Mflops each sustained inclusive of communications costs.
  - 5 MB memory per processor
  - ~20,000 nearest neighbor send-receive pairs per subdomain per simulated hour of ~10KB each

A Programming Model Approach: Partitioned Global Address Space (PGAS) Languages

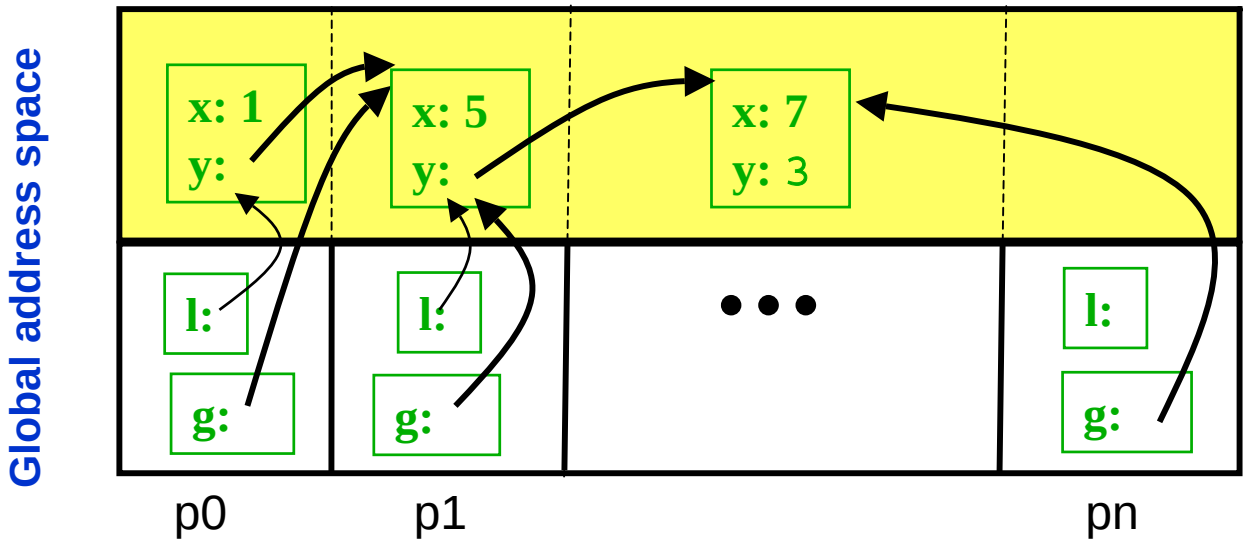
What, Why, and How

Parallel Programming Models

- Easy parallel software is still an unsolved problem !
- **Goals:**
  - Making parallel machines easier to use
  - Ease algorithm experiments
  - Make the most of your machine (network and memory)
  - Enable compiler optimizations
- **Partitioned Global Address Space (PGAS) Languages**
  - Global address space like threads (programmability)
  - One-sided communication
  - Current static (SPMD) parallelism like MPI
  - Local/global distinction, i.e., layout matters (performance)

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:

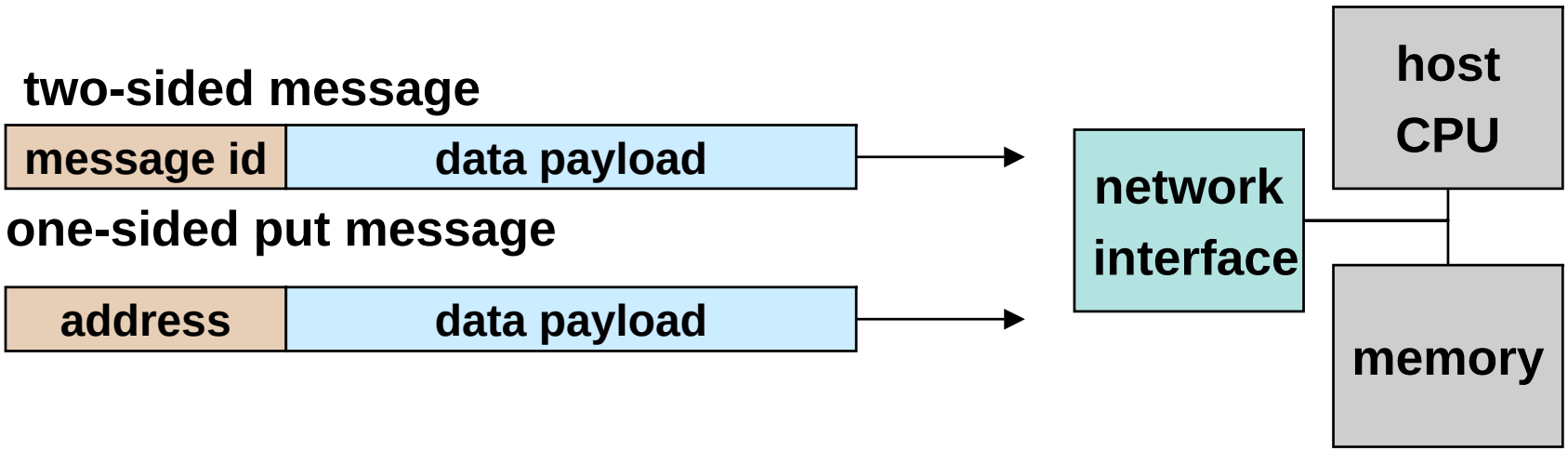
- Object heaps are shared
- Program stacks are private

- *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 for Hybrid Parallelism

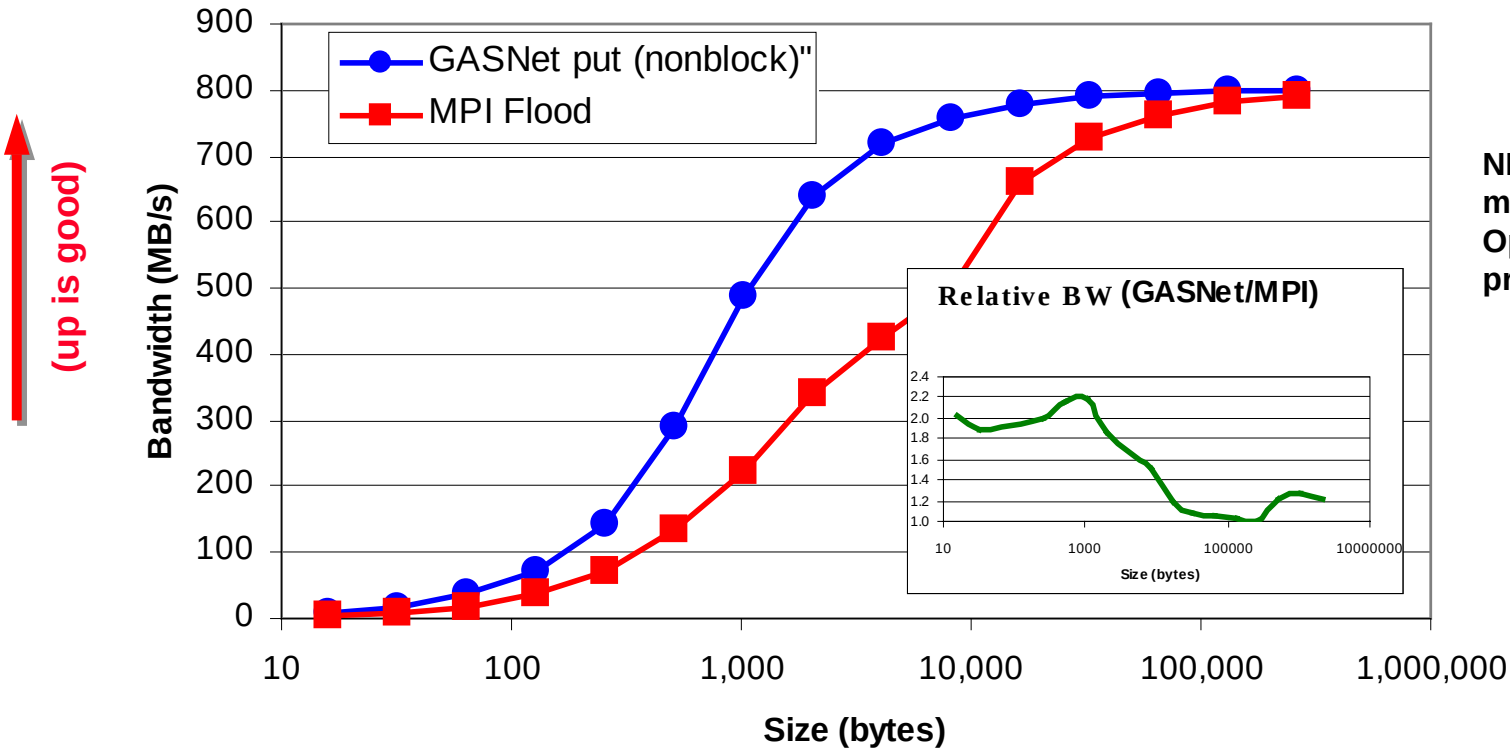
- **PGAS languages are a good fit to shared memory machines**
  - Global address space implemented as reads/writes
  - Current UPC and Titanium implementation uses threads
  - Working on System V shared memory for UPC
- **PGAS languages are a good fit to distributed memory machines and networks**
  - Good match to modern network hardware
  - Decoupling data transfer from synchronization can improve bandwidth

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)

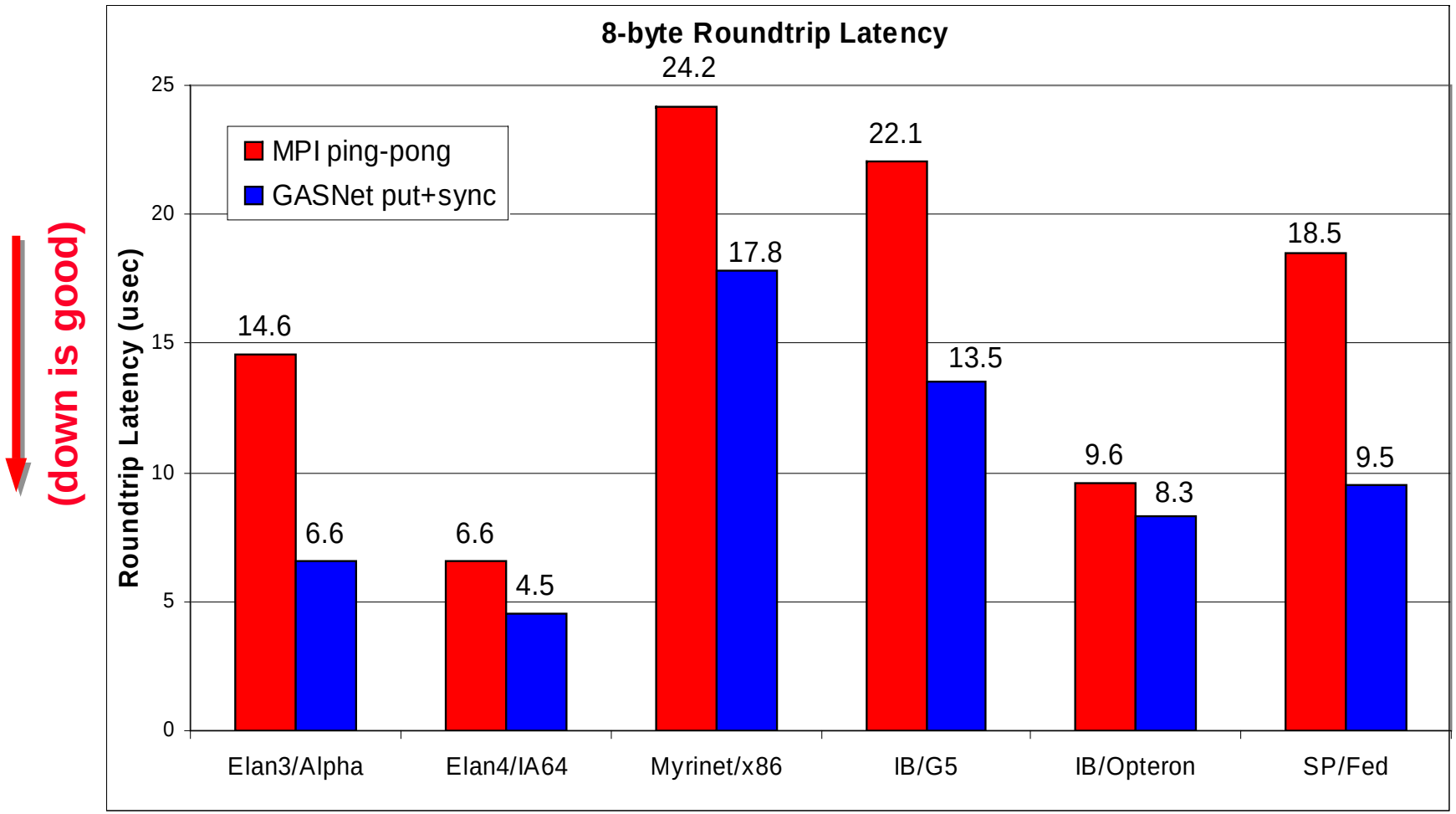
One-Sided vs. Two-Sided: Practice



NERSC Jacquard machine with Opteron processors

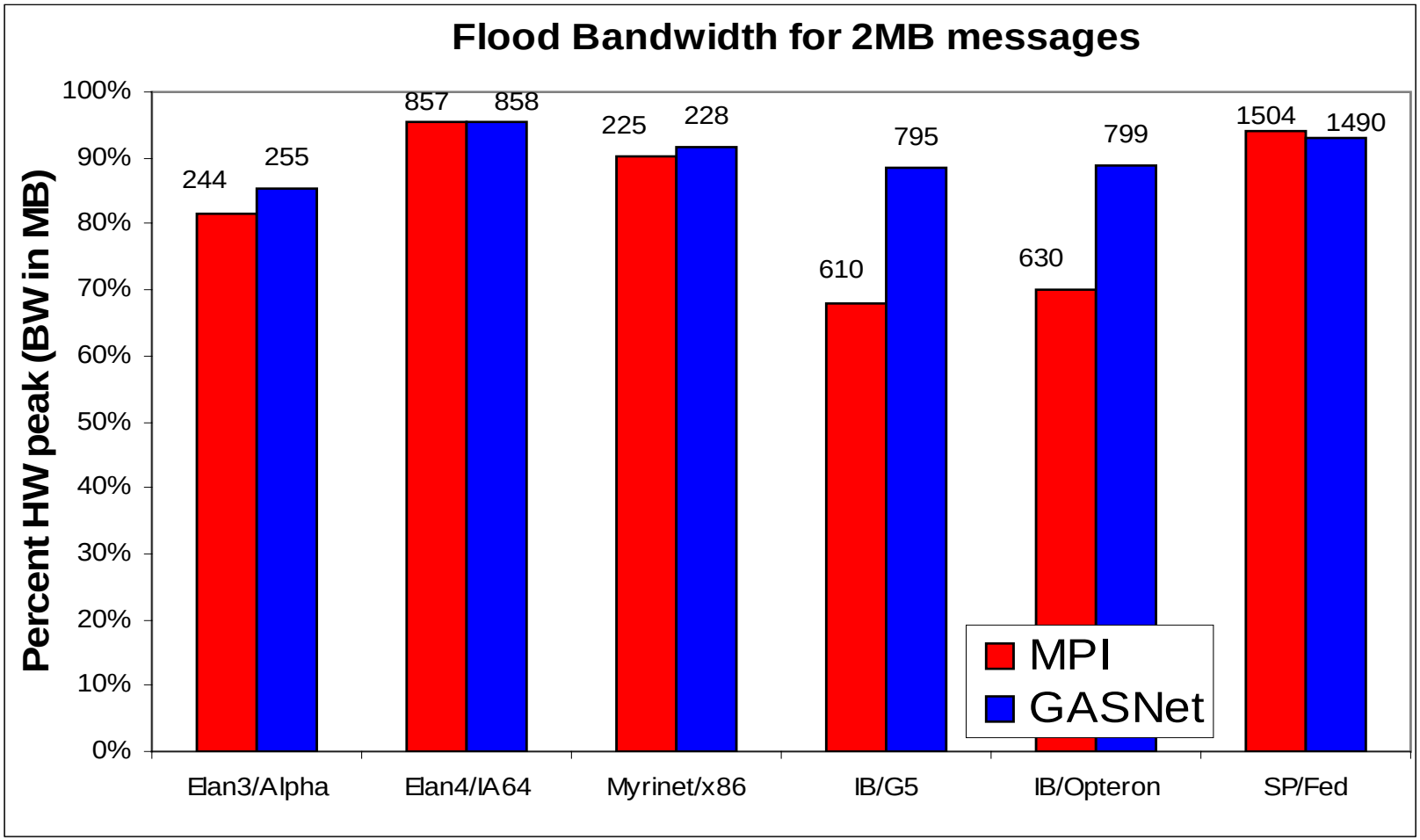
- InfiniBand: GASNet vapi-conduit and OSU MVAPICH 0.9.5
- Half power point (N ½) differs by *one order of magnitude*
- This is not a criticism of the implementation!

GASNet: Portability and *High-Performance*



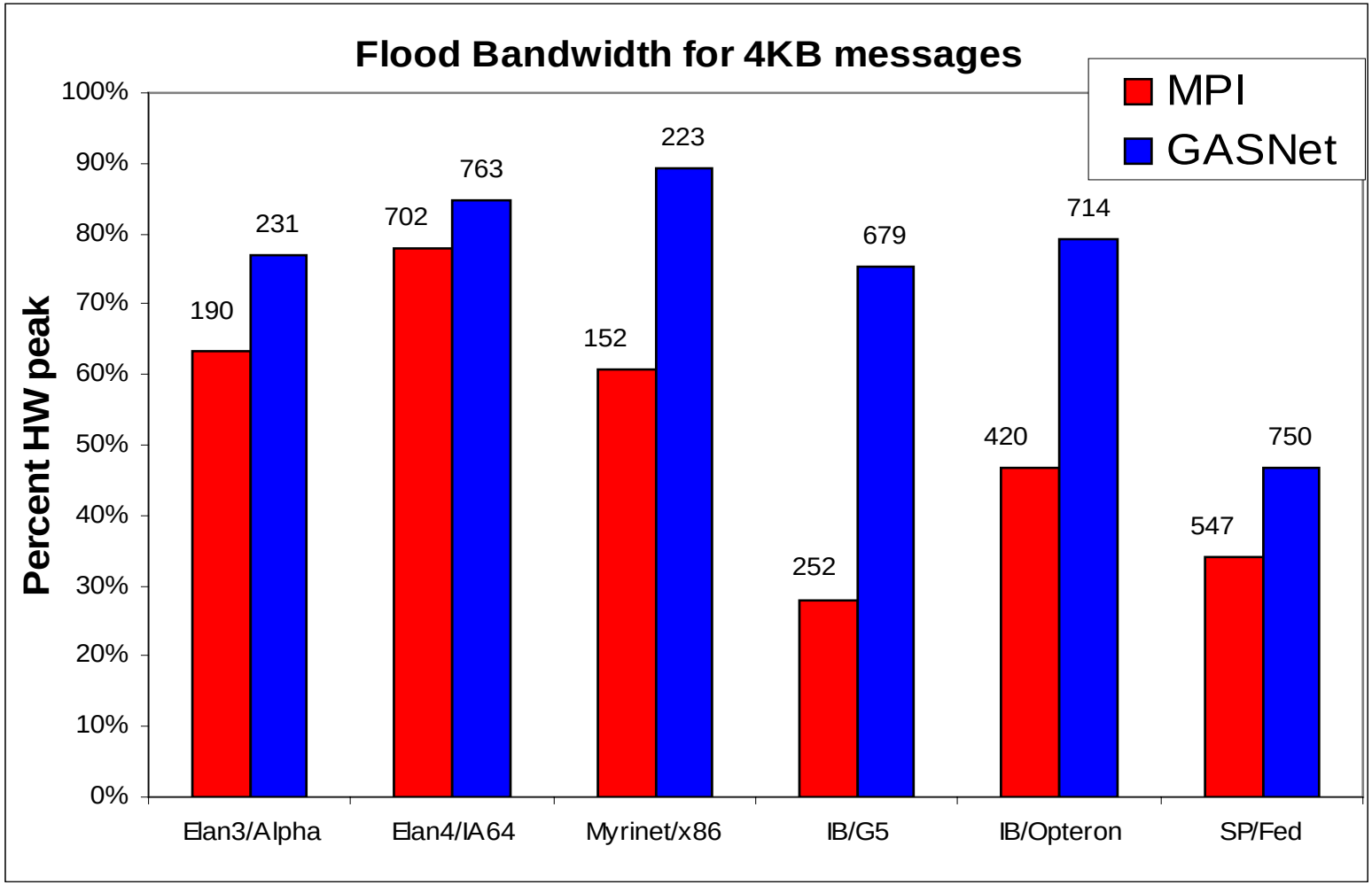
GASNet better for latency across machines

GASNet: Portability and *High-Performance*



GASNet at least as high (comparable) for large messages

GASNet: Portability and *High-Performance*

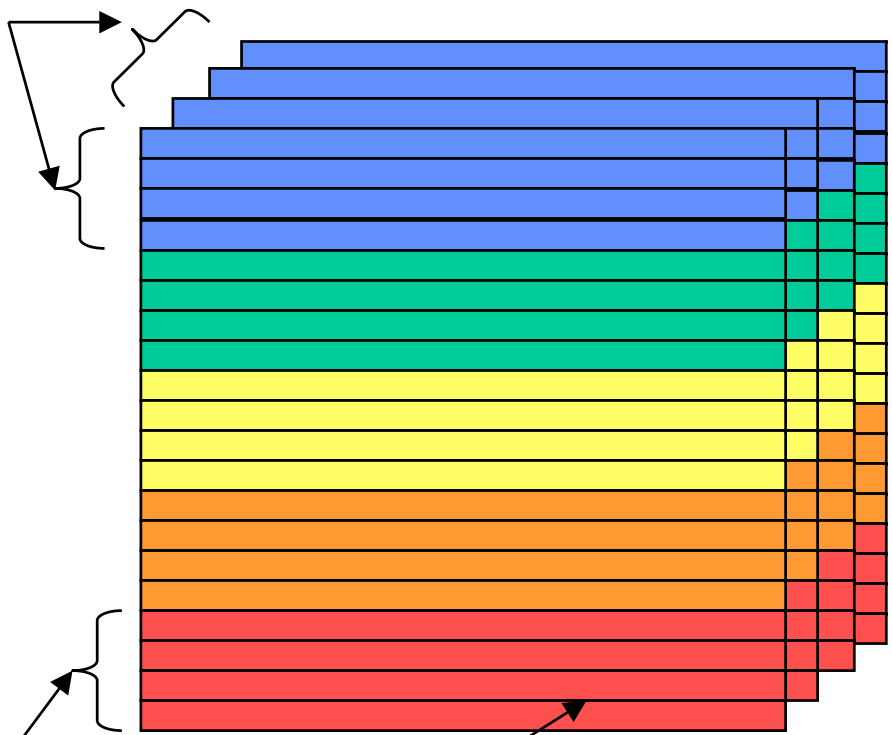


(up is good)

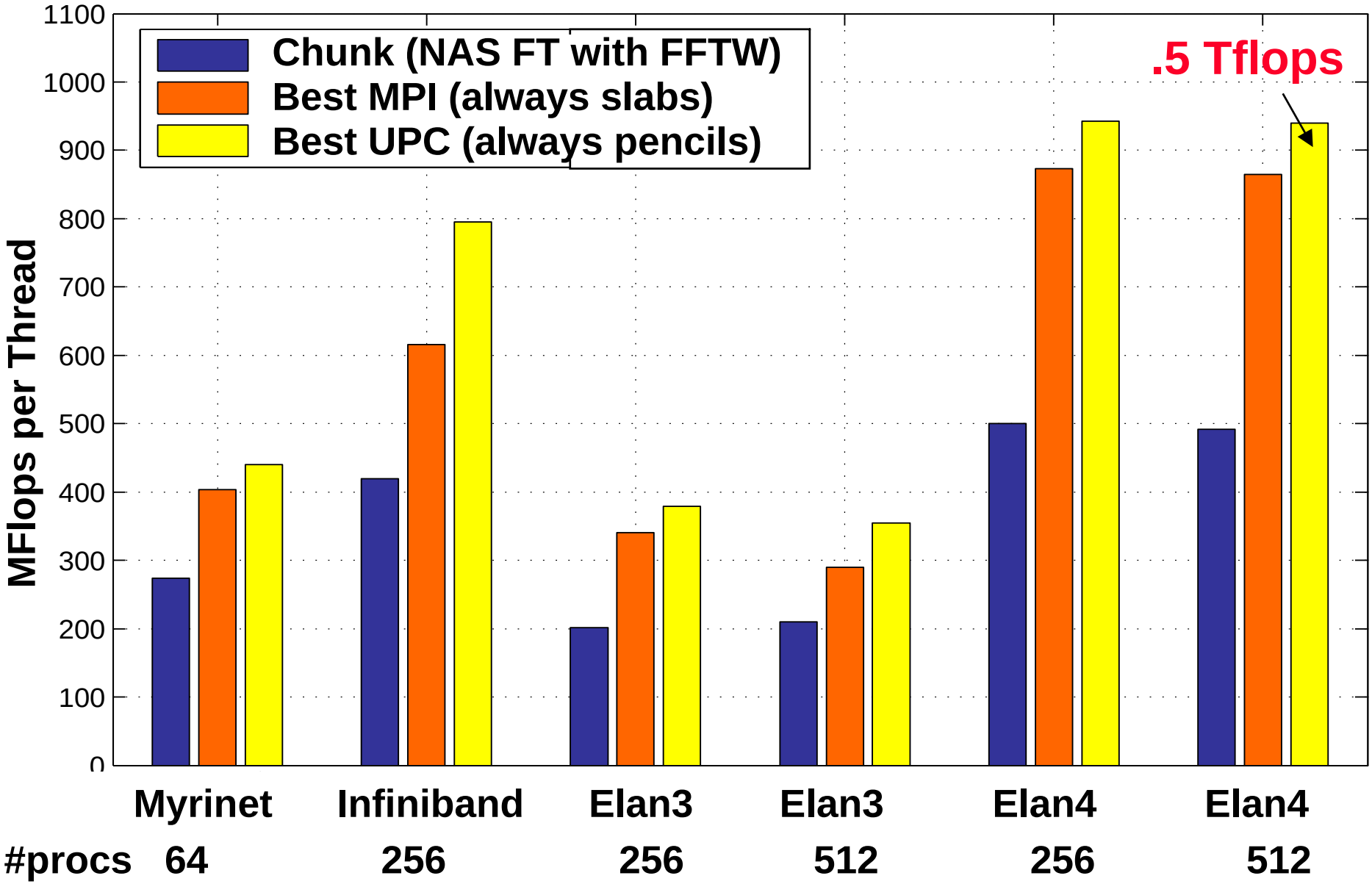
GASNet excels at mid-range sizes: important for overlap

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



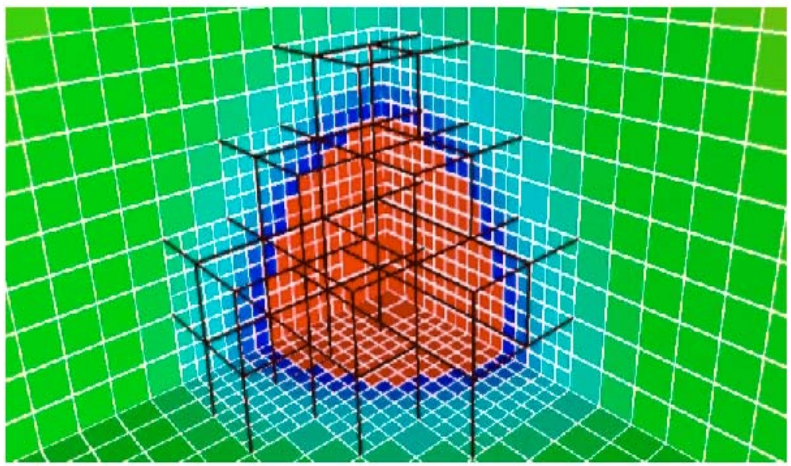
NAS FT Variants Performance Summary



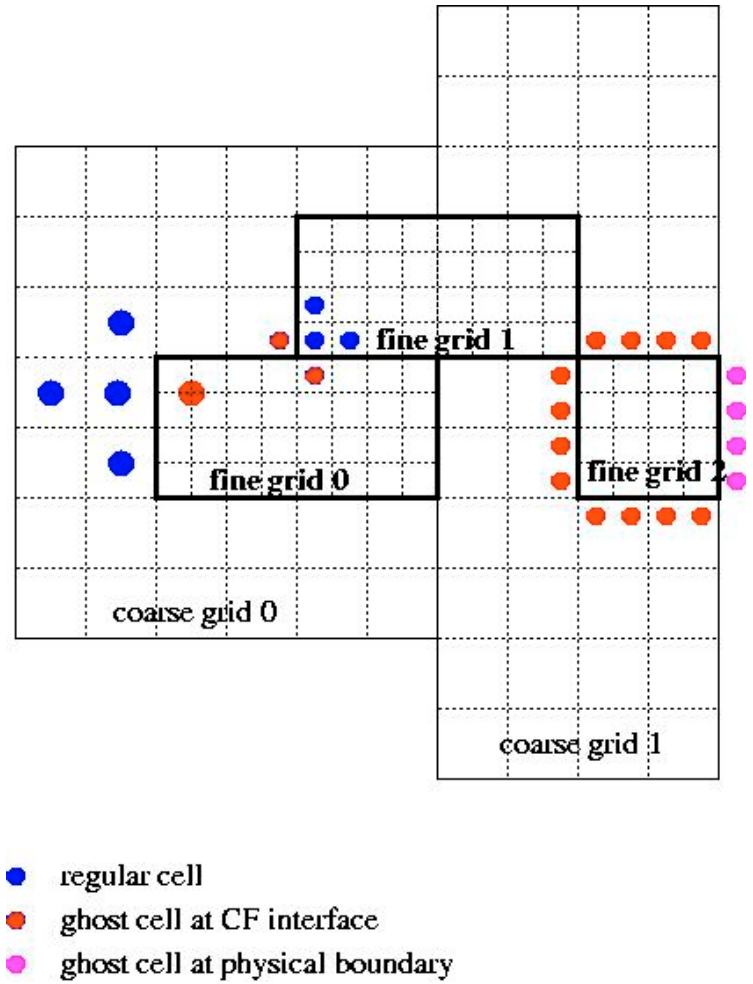
Making PGAS Real: Applications and Portability

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



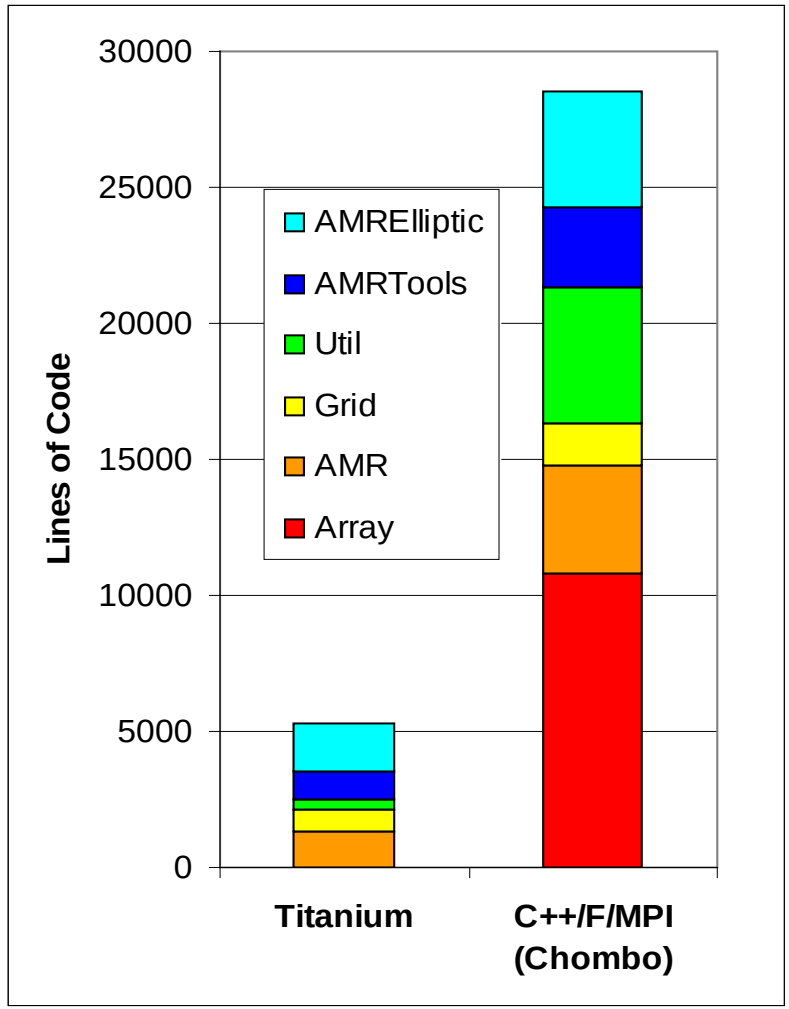
Languages Support Helps Productivity

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



Conclusions

- **Future hardware performance improvements**
  - Mostly from concurrency, not clock speed
  - New commodity hardware coming (sometime) from games/GPUs
- **PGAS Languages**
  - Good fit for shared and distributed memory
  - Control over locality and (for better or worse) SPMD
  - Available for download
    - Berkeley UPC compiler: http://upc.lbl.gov
    - Titanium compiler: http://titanium.cs.berkeley.edu
- **Implications for Climate Modeling**
  - Need to rethink algorithms to identify more parallelism
  - Need to match needs of future (efficient) algorithms

Extra Slides

Parallelism Saves Low Power

- **Exploit explicit parallelism for reducing power**

Power = C * V^2 * F	Performance = Cores * F
Power = 2C * V^2 * F	Performance = 2Cores * F
Power = 4C * V^2/4 * F/2	Performance = 4Lanes * F/2
Power = (C * V^2 * F)/2	Performance = 2Cores * F

  - **Using additional cores**
    - Allows reduction in frequency and power supply without decreasing (original) performance
    - Power supply reduction lead to large power savings
  - **Additional benefits**
    - Small/simple cores → more predictable performance

Where is Fortran?

Source: Tim O'Reilly