Petascale Meets Multicore: Programming Model Challenges and Opportunities

Kathy Yelick

EECS Professor, U.C. Berkeley and

NERSC Director, Lawrence Berkeley National Laboratory

Petaflop with ~1M Cores By 2008

1Eflop/s
100 Pflop/s
10 Pflop/s
1 Pflop/s
100 Tflop/s
10 Tflops/s
1 Tflop/s
100 Gflop/s
10 Gflop/s
1 Gflop/s
100 MFlop/s

1 EFlop system in 2019?

1 PFlop system in 2008

6-8 years

#1
#500

Data from top500.org

1993 1996 1999 2002 2005 2008 2011 2014 2017

Slide source Horst Simon, LBNL

Software Issues at Scale

- DARPA Study on Exascale led by Peter Kogge
  - Power discussion dominates all others
    - concurrency is the only significant approach
    - lower clock and increase parallelism
  - Power density and system power (20-155MW)
- Summary Issues for Software
  - Exascale will require billion-way concurrency with 1K cores per chip
  - Departmental scale (1 PF) systems will require millions of threads with 1K cores per chip
  - The memory/core ratio will drop by at least an order of magnitude across machine size
    - Note: Weak Scaling may not work!
  - A new model for fault tolerant software is needed; checkpoints to disk will be impractical
- These issues will creep into Petascale

BERKELEY LAB

Need a Fundamentally New Approach

- Rethink programming models
  - Massive parallelism
  - Eliminate scaling bottlenecks replication, synchronization
- Rethink algorithms
  - Massive parallelism and locality
  - Counting Flops is the wrong measure
- Rethink hardware
  - What limits performance
  - How to build efficient hardware

Rethink Programming Models

Strategies for Manycore in Exascale

- There are multiple approaches we could take in the HPC community
- They have different cost to us:
  - Software infrastructure investment
  - Application software investment
- And different risks of working
  - At all
  - Or at the performance level we demand

1) MPI Everywhere

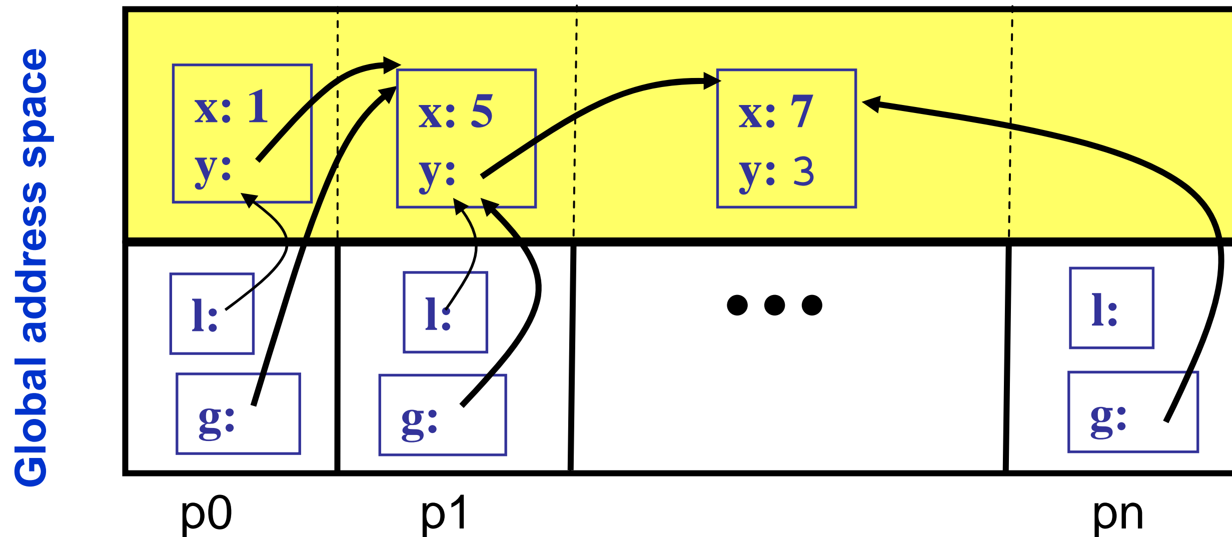
- We can run 1 MPI process per core
  - This works now (for CMPs) and will work for a while
- How long will it continue working?
  - 4 - 8 cores? Probably. 128 - 1024 cores? Probably not.
  - Depends on performance expectations -- more on this later
- What is the problem?
  - Latency: some copying required by semantics
  - Memory utilization: partitioning data for separate address space requires some replication
    - How big is your per core subgrid? At 10x10x10, over 1/2 of the points are surface points, probably replicated
  - Memory bandwidth: extra state means extra bandwidth
  - Weak scaling will not save us -- not enough memory per core
  - Heterogeneity: MPI per CUDA thread-block?
- Advantage: no new apps work; modest infrastructure work (multicore-optimized MPI)

2) Mixed MPI and OpenMP

- This is the obvious next step
- Problems
  - Will OpenMP performance scale with the number of cores / chip?
  - OpenMP does not support locality optimizations
  - More investment in infrastructure than MPI, but can leverage existing technology
  - Heterogeneity support unclear
  - Do people want two programming models?
- Advantages
  - Incremental work for applications
- Variation: await a silver bullet from industry
  - Will this be at all helpful in scientific applications?
  - Do they know enough about parallelism/algorithms

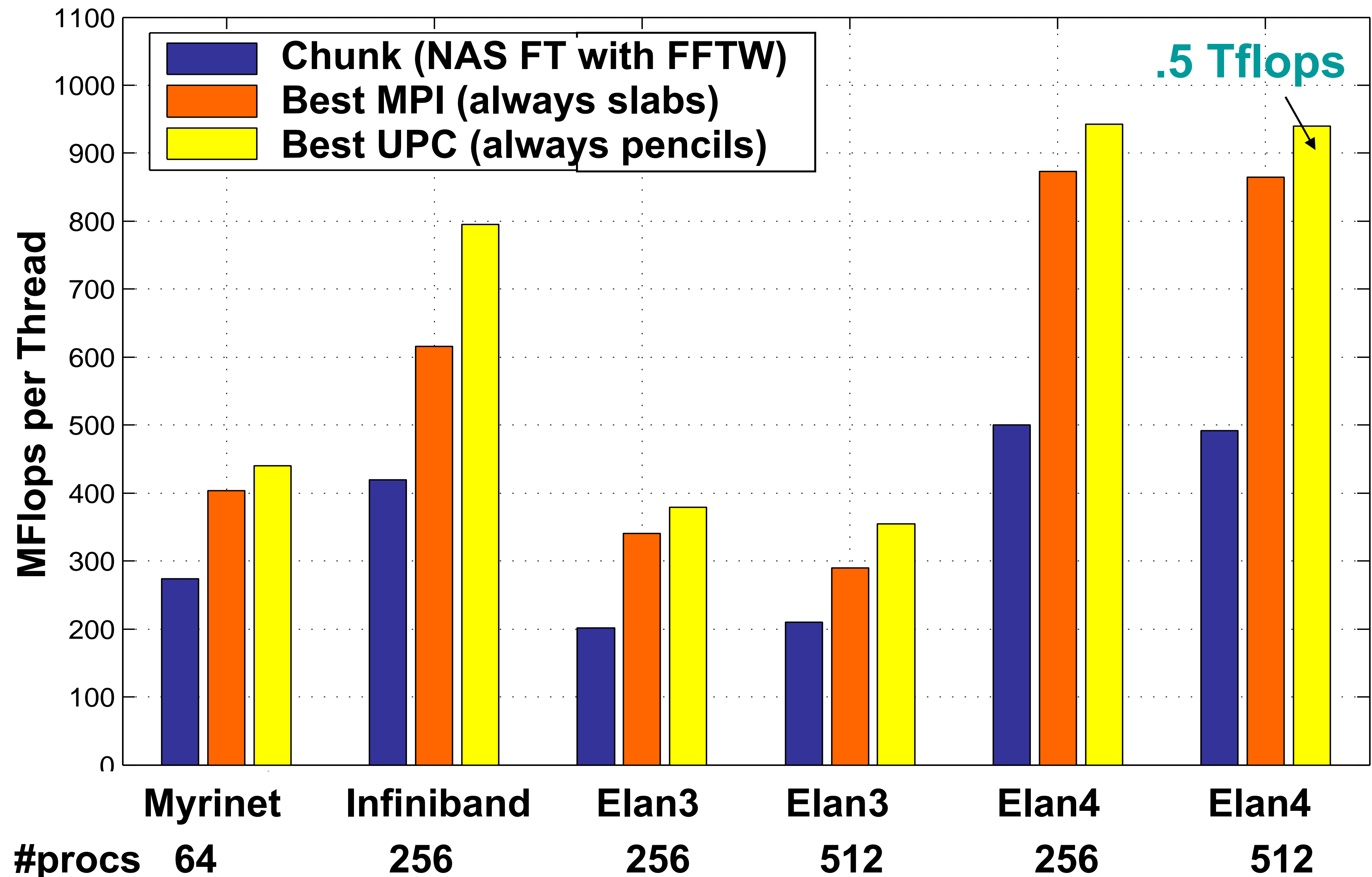
3) PGAS Languages

- *Global address space:* thread may directly read/write remote data
- *Partitioned:* data is local or global: critical for scaling
  - Maps directly to shared memory hardware
  - Maps to one-sided communication on distributed memory hardware
  - One programming model for inter and intra node parallelism!



- *UPC, CAF, Titanium: Static parallelism (1 thread per proc)*
  - Does not virtualize processors; main difference from HPCS languages which have many/dynamic threads

NAS FT Variants Performance Summary



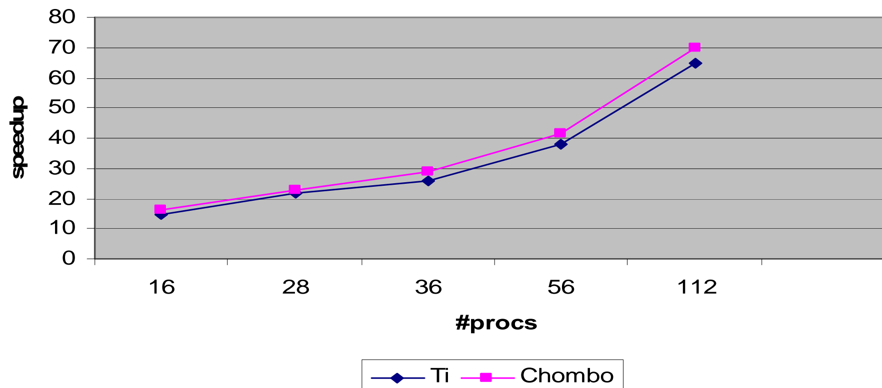
Languages Support Helps Productivity

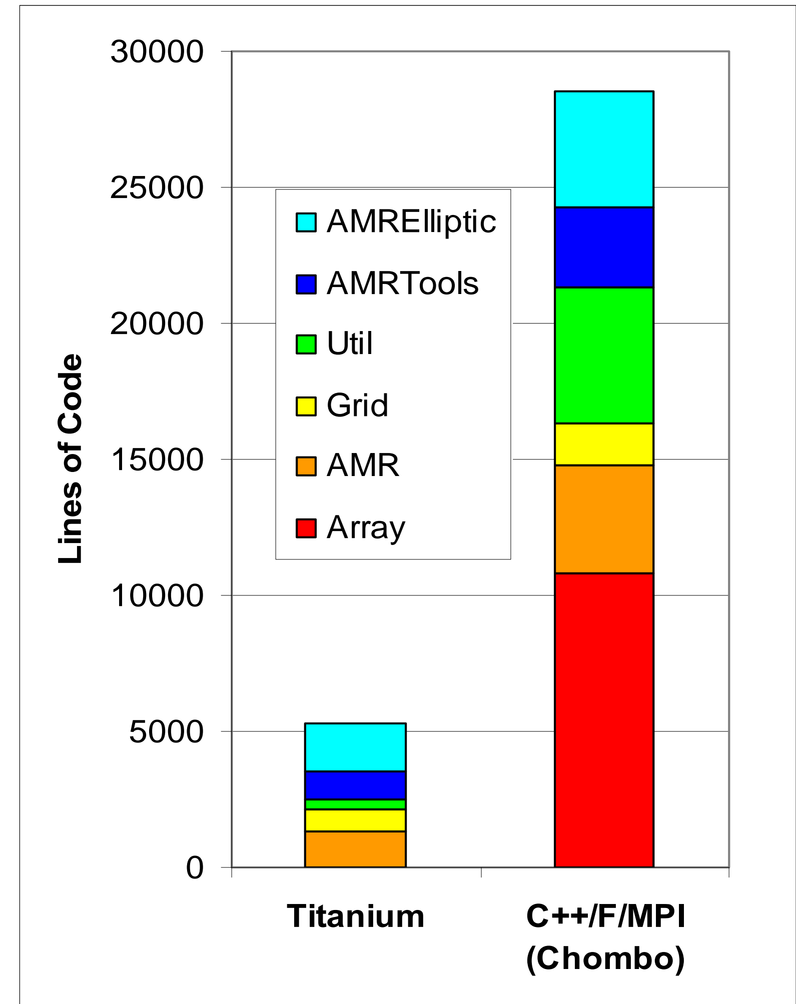
C++/Fortran/MPI AMR

- 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



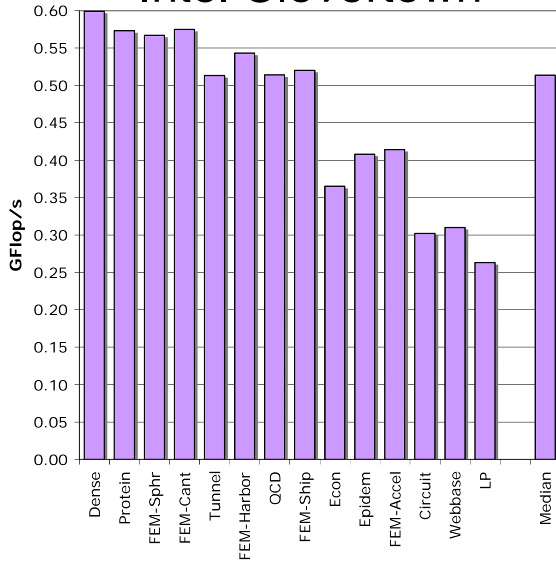


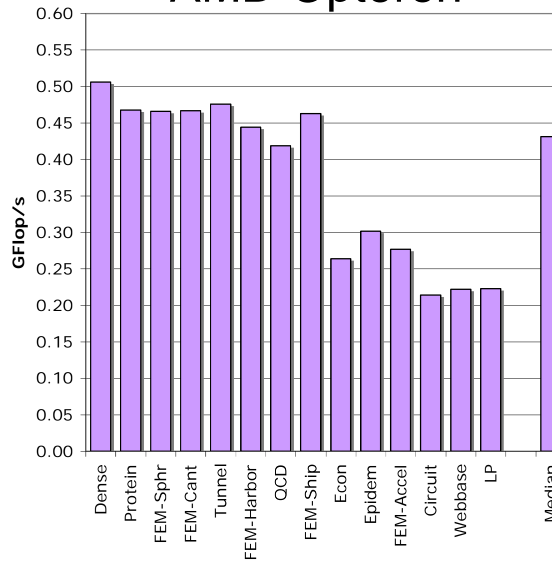
Work by Tong Wen and Philip Colella; Communication optimizations joint with Jimmy Su

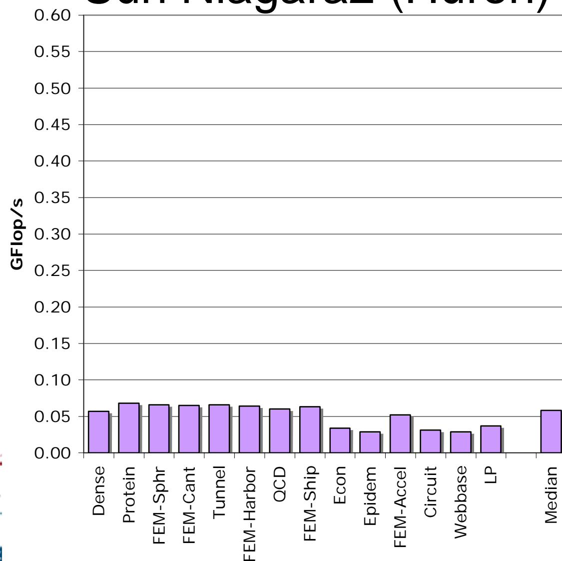
Autotuning for Multicore: Extreme Performance Programming

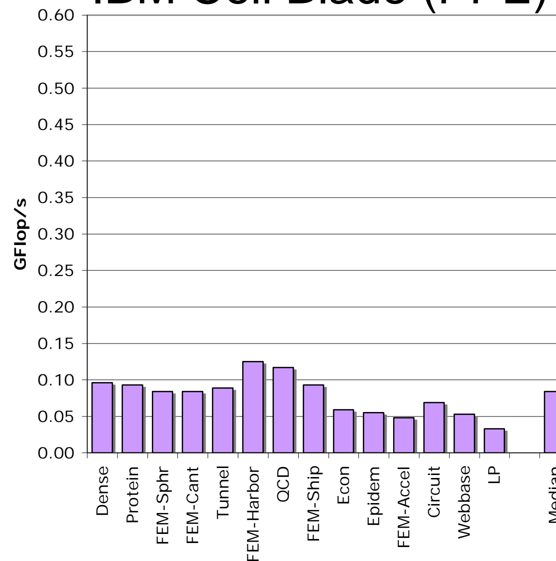
- Automatic performance tuning
  - Use machine time in place of human time for tuning
  - Search over possible implementations
  - Use performance models to restrict search space
- Programmers should write programs to generate code, not the code itself
- Autotuning finds a good performance solution be heuristics or exhaustive search
  - Perl script generates many versions
  - Generate SIMD-optimized kernels
  - Autotuner analyzes/runs kernels
  - Uses search and heuristics
- Can do this in libraries (Atlas, FFTW, OSKI) or compilers (ongoing research)

Naïve Serial Implementation









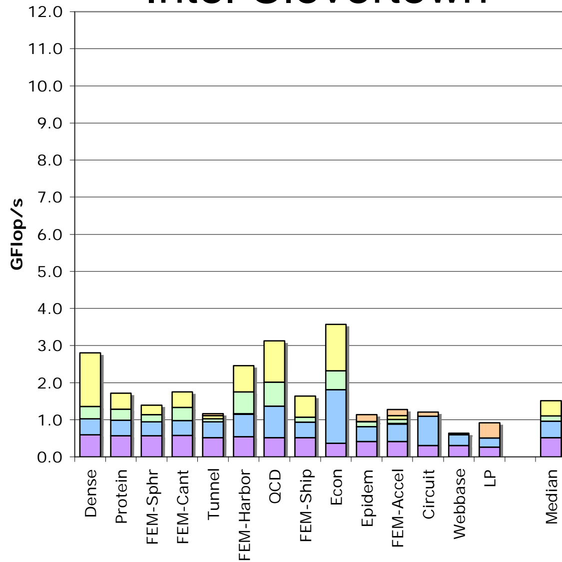
- **Vanilla C implementation**
- **Matrix stored in CSR (compressed sparse row)**
- **Explored compiler options, but only the best is presented here**
- **x86 core delivers > 10x the performance of a Niagara2 thread**

- **Work by Sam Williams with Vuduc, Oliker, Shalf, Demmel, Yelick**

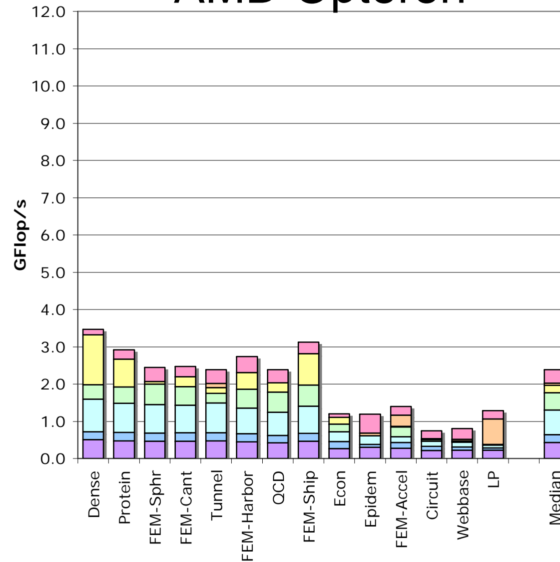
Autotuned Performance

(+Cell/SPE version)

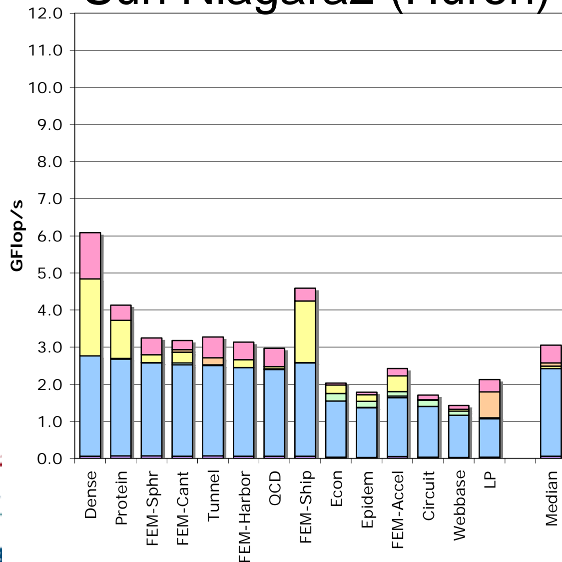
Intel Clovertown



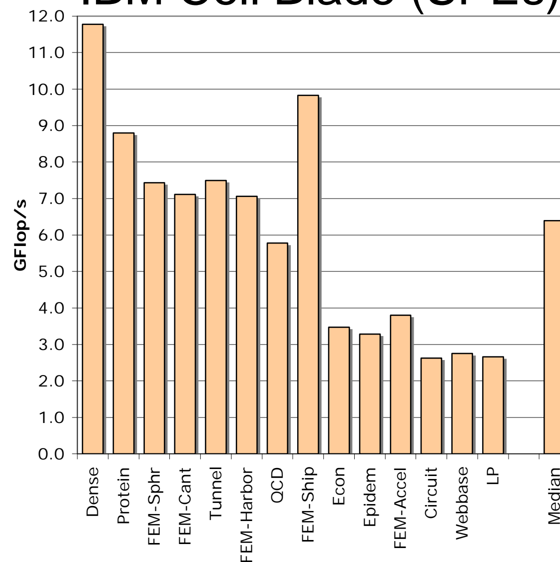
AMD Opteron



Sun Niagara2 (Huron)



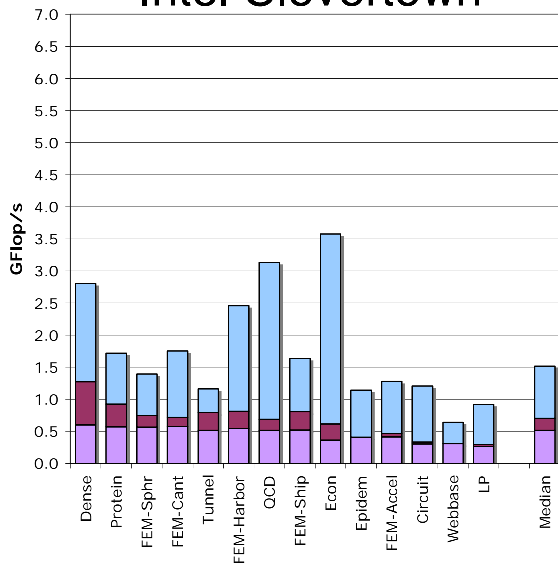
IBM Cell Blade (SPEs)

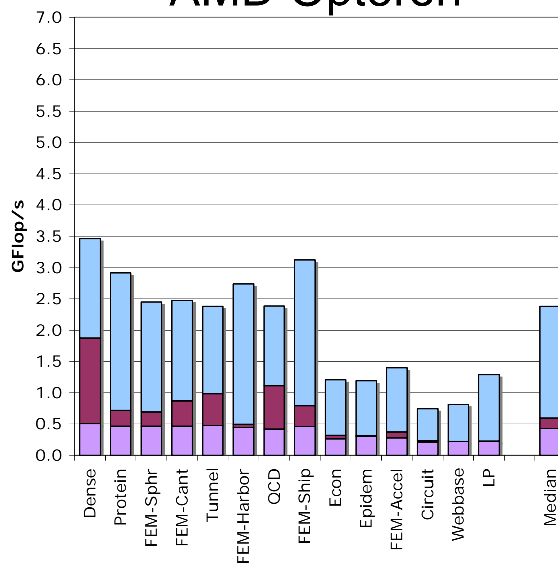


- **Wrote a double precision Cell/SPE version**
- **DMA, local store blocked, NUMA aware, etc…**
- **Only 2x1 and larger BCOO**
- **Only the SpMV-proper routine changed**
- **About 12x faster (median) than using the PPEs alone.**

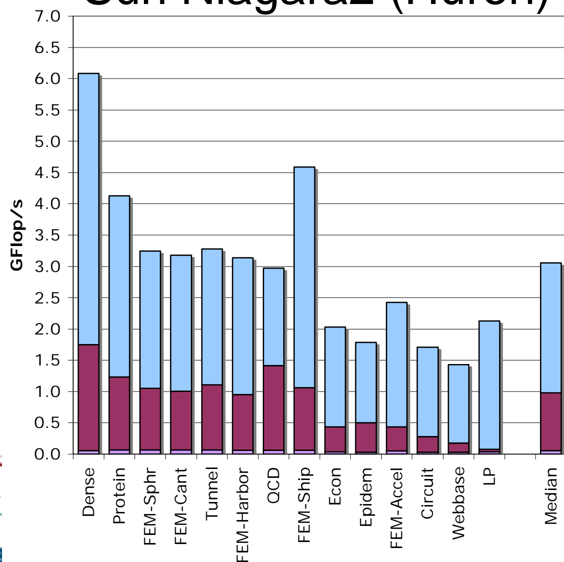
- +More DIMMs(opteron), +FW fix, array padding(N2), etc…
- +Cache/TLB Blocking
- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve

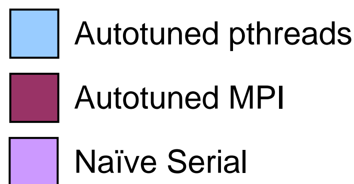
MPI vs. Threads





- **On x86 machines, autotuned shared memory MPICH implementation rarely scales beyond 2 threads**
- **Still debugging MPI issues on Niagara2, but so far, it rarely scales beyond 8 threads.**





Rethinking Algorithms

Count memory, not Flops

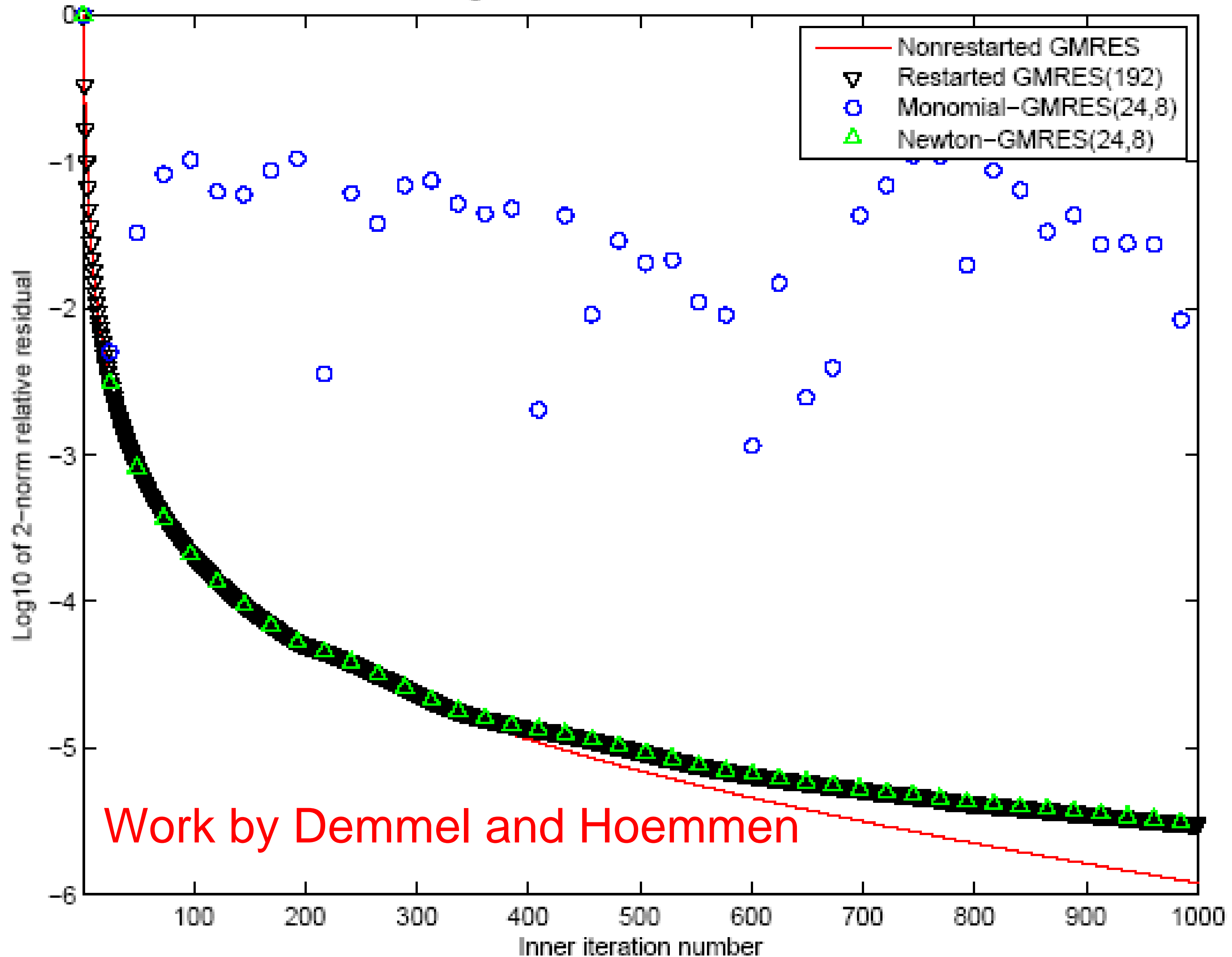
BERKELEY LAB

THE UNIVERSITY OF CALIFORNIA BERKELEY 1868

Latency and Bandwidth-Avoiding

- Many iterative algorithms are limited by
  - Communication latency (frequent messages)
  - Memory bandwidth
- New optimal ways to implement Krylov subspace methods on parallel and sequential computers
  - Replace $x \rightarrow Ax$ by $x \rightarrow [Ax, A^2x, \ldots A^kx]$
    - One read of the matrix or mesh, multiple steps
  - Change GMRES, CG, Lanczos, … accordingly
- Theory
  - Minimizes network latency costs on parallel machine
  - Minimizes memory bandwidth on sequential machine
- Performance models for 2D problem
  - Up to 7x speedups on "petascale"
  - Measured speedup: 3.2x for out-of-core

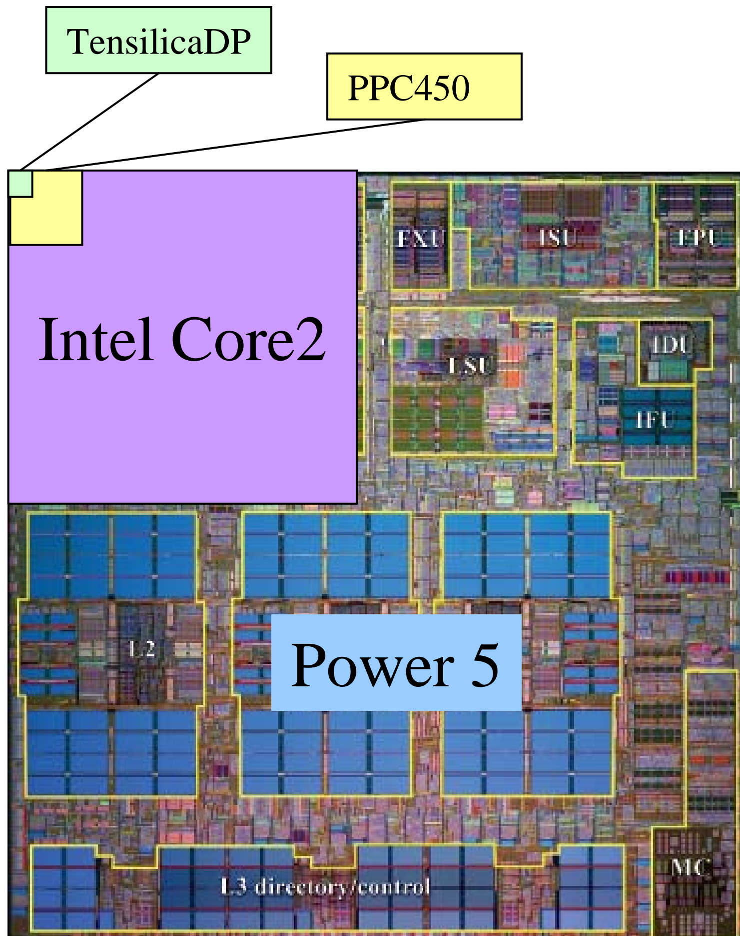
Can use Matrix Power Kernel, but change Algorithms



Rethink Hardware

(Design for Power)

Design for Power: More Concurrency

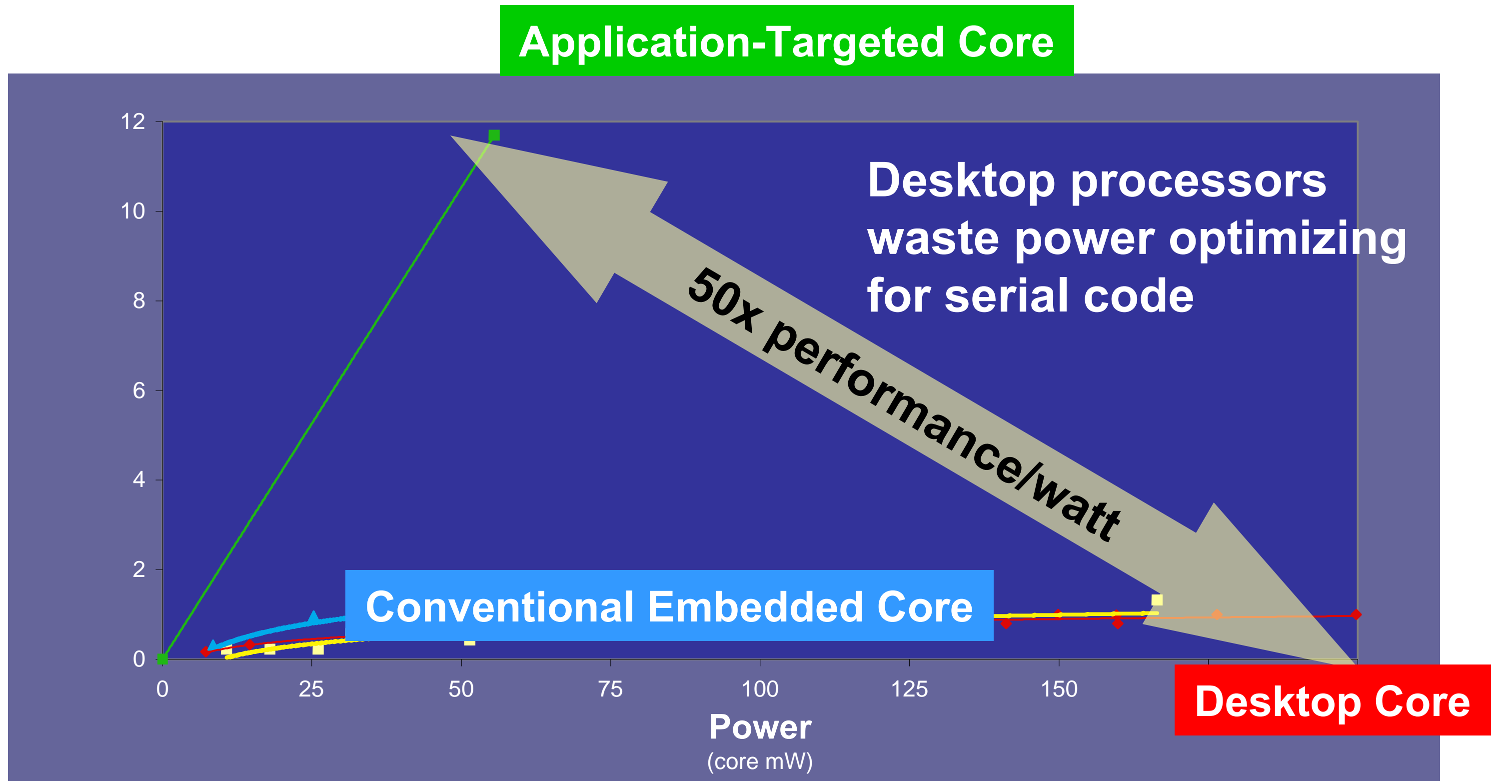


- **Power5 (Server)**
  - $389\ mm^2$
  - 120 W @ 1900 MHz
- **Intel Core2 sc (Laptop)**
  - $130\ mm^2$
  - 15 W @ 1000 MHz
- **PowerPC450 (BlueGene/P)**
  - $8\ mm^2$
  - 3 W @ 850 MHz
- **Tensilica DP (cell phones)**
  - $0.8\ mm^2$
  - 0.09 W @ 650 MHz

Each core operates at 1/3 to 1/10th efficiency of largest chip, but you can pack 100x more cores onto a chip and consume 1/20 the power!

Specialization Saves Power

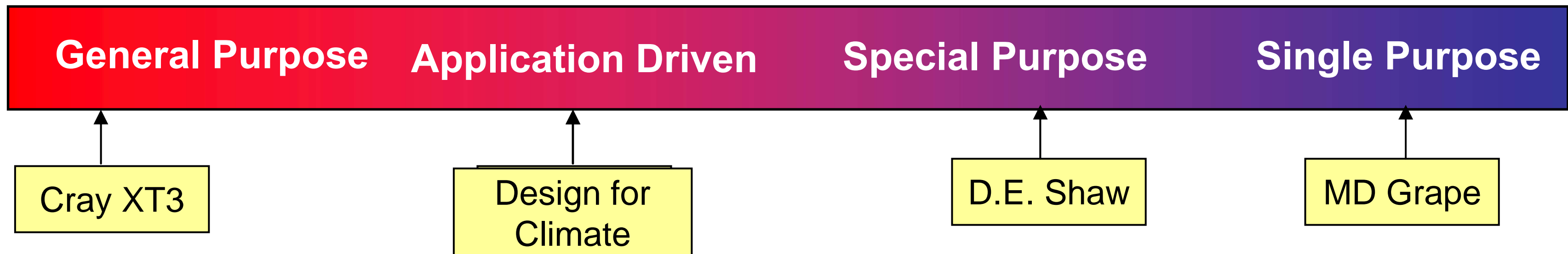
Graph courtesy of Chris Rowen, Tensilica Inc.



Performance on EEMBC benchmarks aggregate for Consumer, Telecom, Office, Network, based on ARM1136J-S (Freescale i.MX31), ARM1026EJ-S, Tensilica Diamond 570T, T1050 and T1030, MIPS 20K, NECVR5000). MIPS M4K, MIPS 4Ke, MIPS 4Ks, MIPS 24K, ARM 968E-S, ARM 966E-S, ARM926EJ-S, ARM7TDMI-S scaled by ratio of Dhrystone MIPS within architecture family. All power figures from vendor websites, 2/23/2006.

BERKELEY LAB

Strawman 1km Climate Computer
(Shalf, Oliker, Wehner)

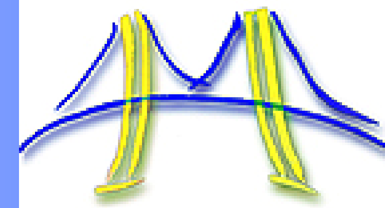


- **Computation**
  - .015°X.02°X100L (note 4X more vertical levels)
- **Hardware:**
  - ~10 Petaflops sustained (300 Pflops peak?)
  - ~100 Terabytes total memory
  - ~20 million processors using true commodity (embedded cores)
- **Massively parallel algorithms with autotuning**
  - E.g., scalable data structure, e.g., Icosahedral
  - ~20,000 nearest neighbor send-receive pairs per subdomain per simulated hour of ~10KB each
- **Upside result:**
  - 1K scale model running in O(5 years)!
- **Worse case:**
  - Better understand how to build/buy climate systems

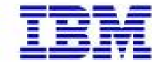
Conclusions

- Rethink Programming Models
  - PGAS: Avoid replication in favor of sharing
  - Autotuning: write self-tuning applications
- Rethink Algorithms
  - Design for bottlenecks: latency and bandwidth
- Rethink Hardware
  - More concurrency to get to Exascale
  - Specialization: Design for climate modeling!
- Rethink Application

Autotuning Sparse Matrix Vector Multiply (Memory Bound)



Sun



Name	Clovertown	Opteron	Niagra2	Cell
Chips*Cores	2*4 = 8	2*2 = 4	2*8	1*8 = 8
Architecture	4-/3-issue, 2-/1-SSE3, OOO, caches, prefetch		multithreaded	2-VLIW, SIMD, local store + DMA
Clock Rate	2.3 GHz	2.2 GHz		3.2 GHz
Peak MemBW	21.3 GB/s	21.3		25.6 GB/s
Peak GFLOPS	74.6 GF	17.6 GF		14.6 (DP Fl. Pt.)
Naïve (median)	1.0 GF	0.6 GF	2.5 GF	--
Efficiency %	1%	3%		--
Autotune SpMV	1.5 GF	1.9 GF		3.4 GF
Auto Speedup	1.5X	3.2X		∞

A Brief History of Languages

- **When vector machines were king**
  - Parallel "languages" were loop annotations (IVDEP)
  - Performance was fragile, but there was good user support
- **When SIMD machines were king**
  - Data parallel languages popular and successful (CMF, *Lisp, C*, …)
  - Quite powerful: can handle irregular data (sparse mat-vec multiply)
  - Irregular computation is less clear (multi-physics, adaptive meshes, backtracking search, sparse matrix factorization)
- **When shared memory machines (SMPs) were king**
  - Shared memory models, e.g., OpenMP, Posix Threads, are popular
- **When clusters took over**
  - Message Passing (MPI) became dominant

We are at the mercy of HW, but is this true today?