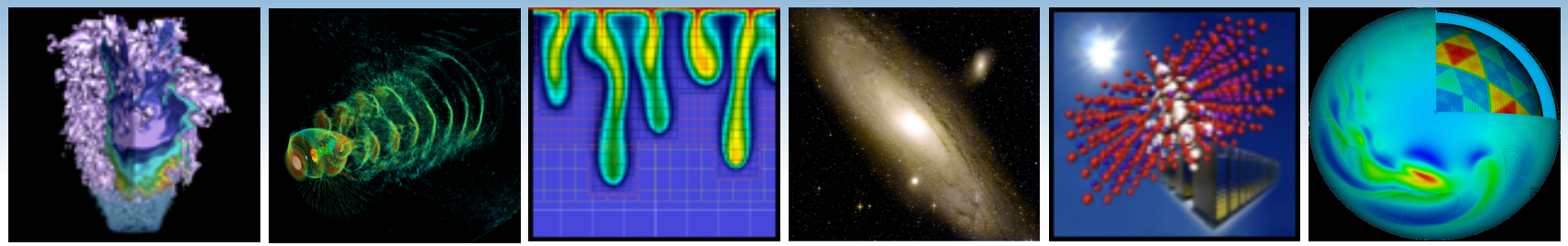


Exascale Computing: Opportunities and Challenges

Kathy Yelick

Associate Laboratory Director for Computing Sciences and NERSC Center Director

Lawrence Berkeley National Laboratory

EECS Professor, UC Berkeley

NERSC Facility Leads DOE in Scientific Computing Productivity

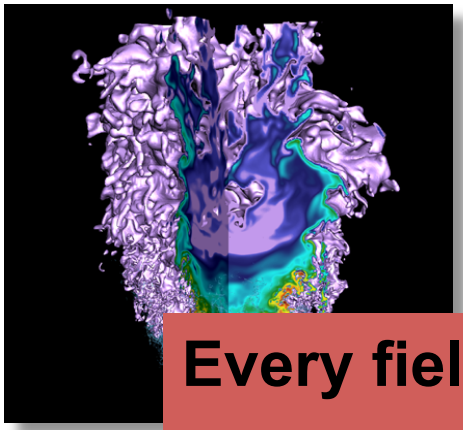
NERSC computing for science

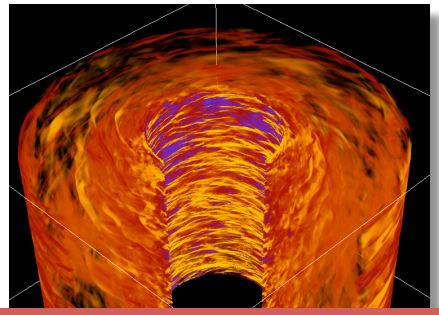
- 4000 users, 500 projects
- 1500 publications per year
- Outstanding user services, computing and data systems

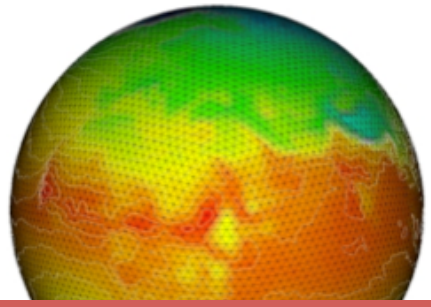
Systems designed for science

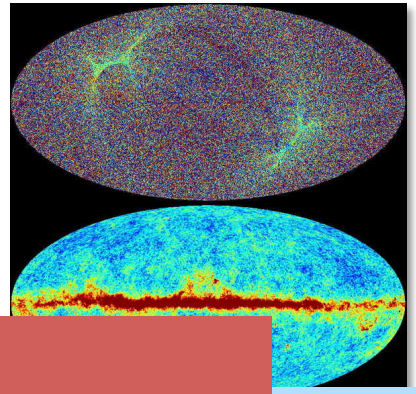
- 1.3 Petaflop Hopper system
- Purchased for application performance per $ and per Watt
- Designed for reliability and productivity

Exascale: Who Needs It?







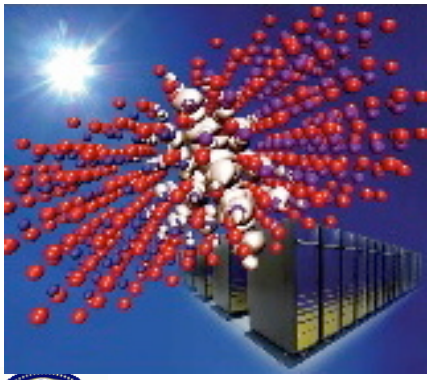


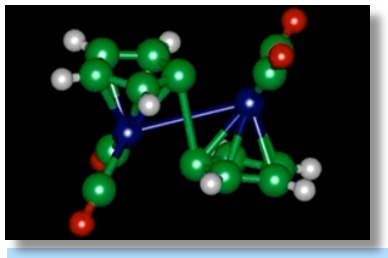
Every field needs more computing!

1) To quantify and reduce uncertainty in simulations
2) Analyze data from experiments and simulations

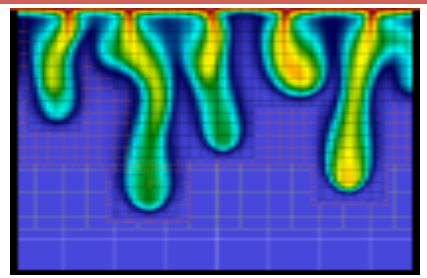
Combu
comple
engine

origins
e





Materials: solar panels to database of materials-by-design.



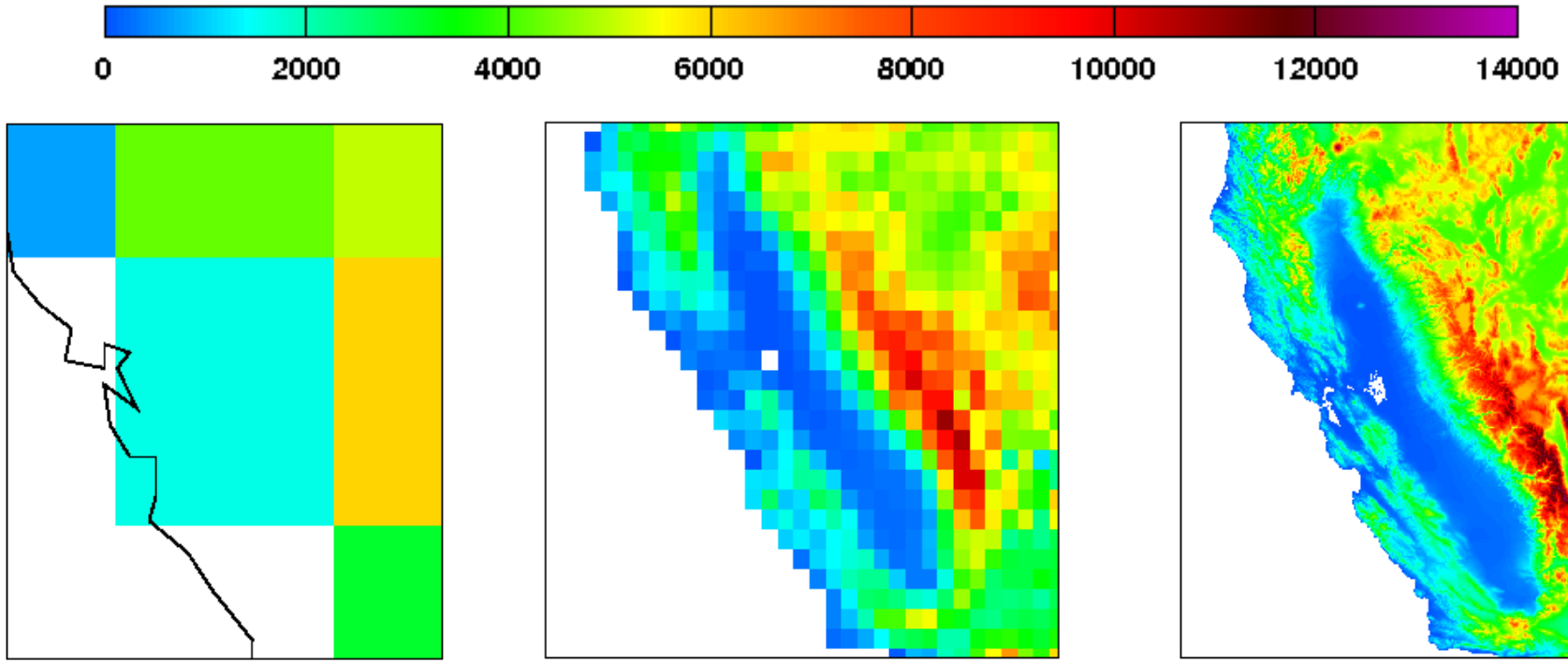
Sequestration: Understanding fluid flow & chemistry

Protein structures: From Biofuels to Alzheimers

BERKELEY LAB

Exascale Science: *Global Cloud Resolving Climate Model*

Surface Altitude (feet)



200km
Typical resolution of IPCC AR4 models

25km
Upper limit of climate models with cloud parameterizations

1km
Cloud resolving models are a transformational change

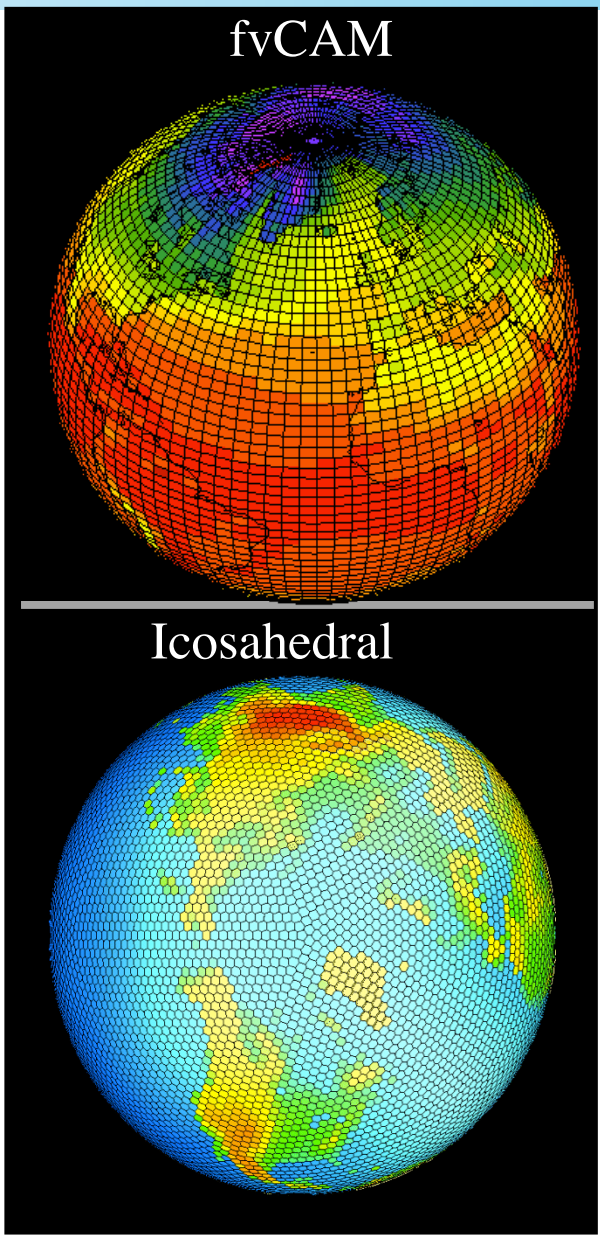
Computational Requirements for 1km Climate Model

Must maintain 1000x faster than real time for practical climate simulation

- *Multiple simulations to improve confidence*
- *Requires Exascale computing*

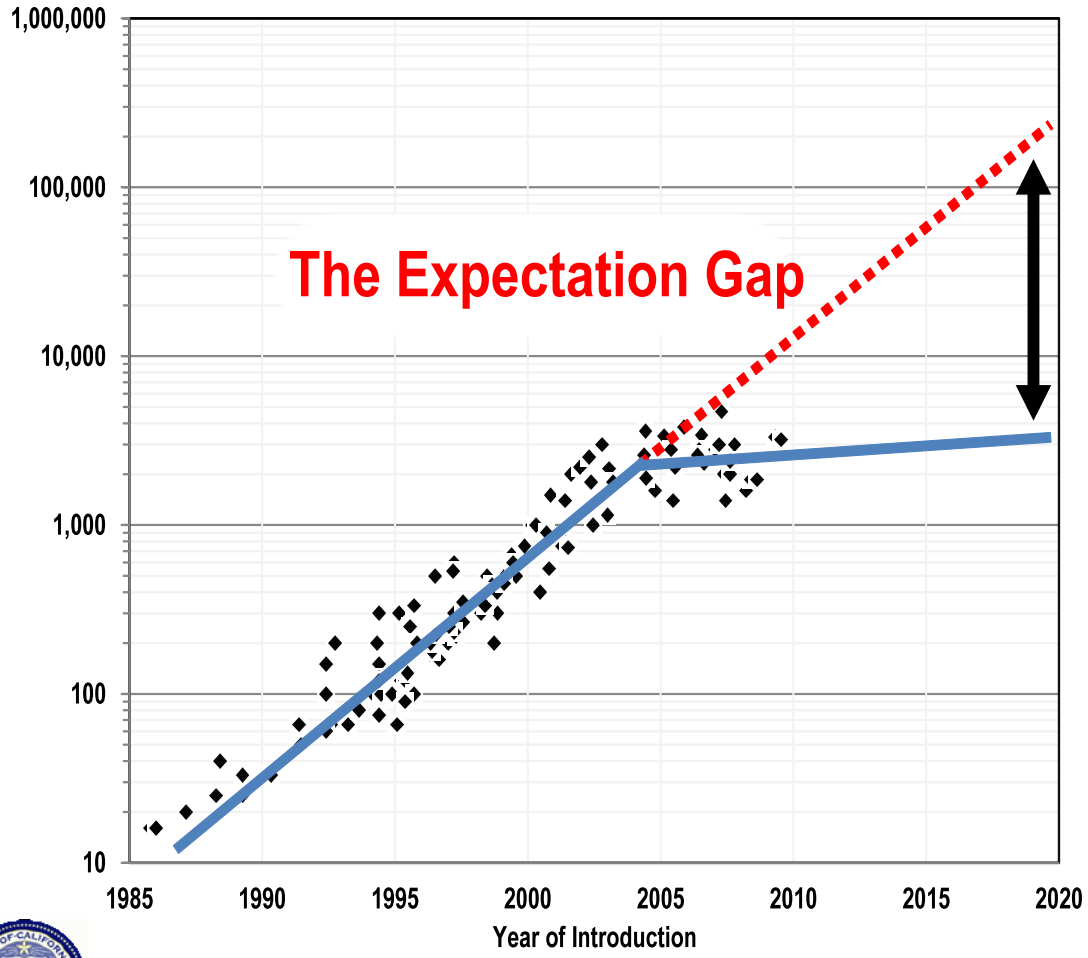
Not just faster computers

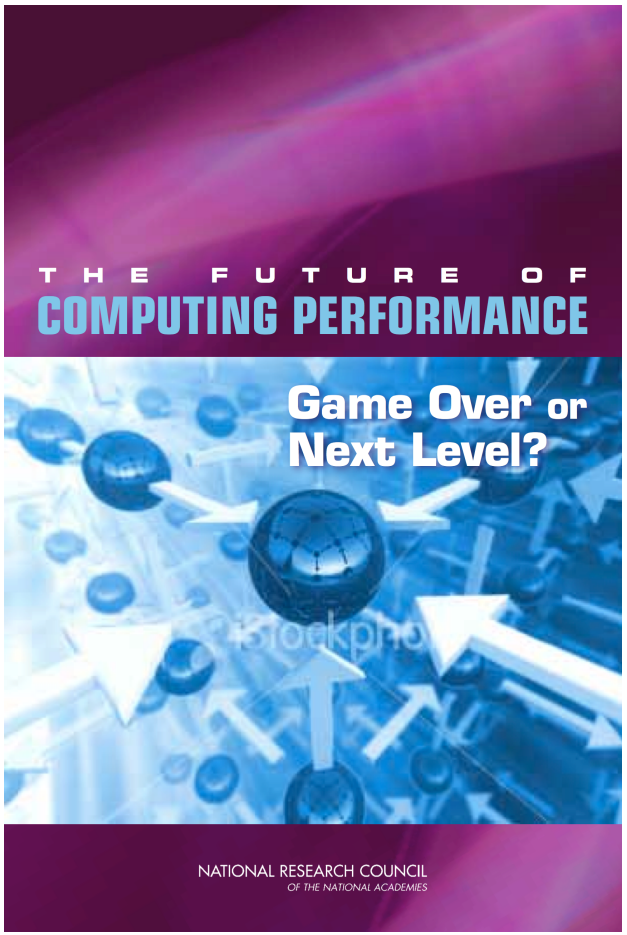
- *New models of how clouds (and ice, and…) behave*
- *New algorithms that scale in both problem size and parallelism*
- *New software to use new machines*



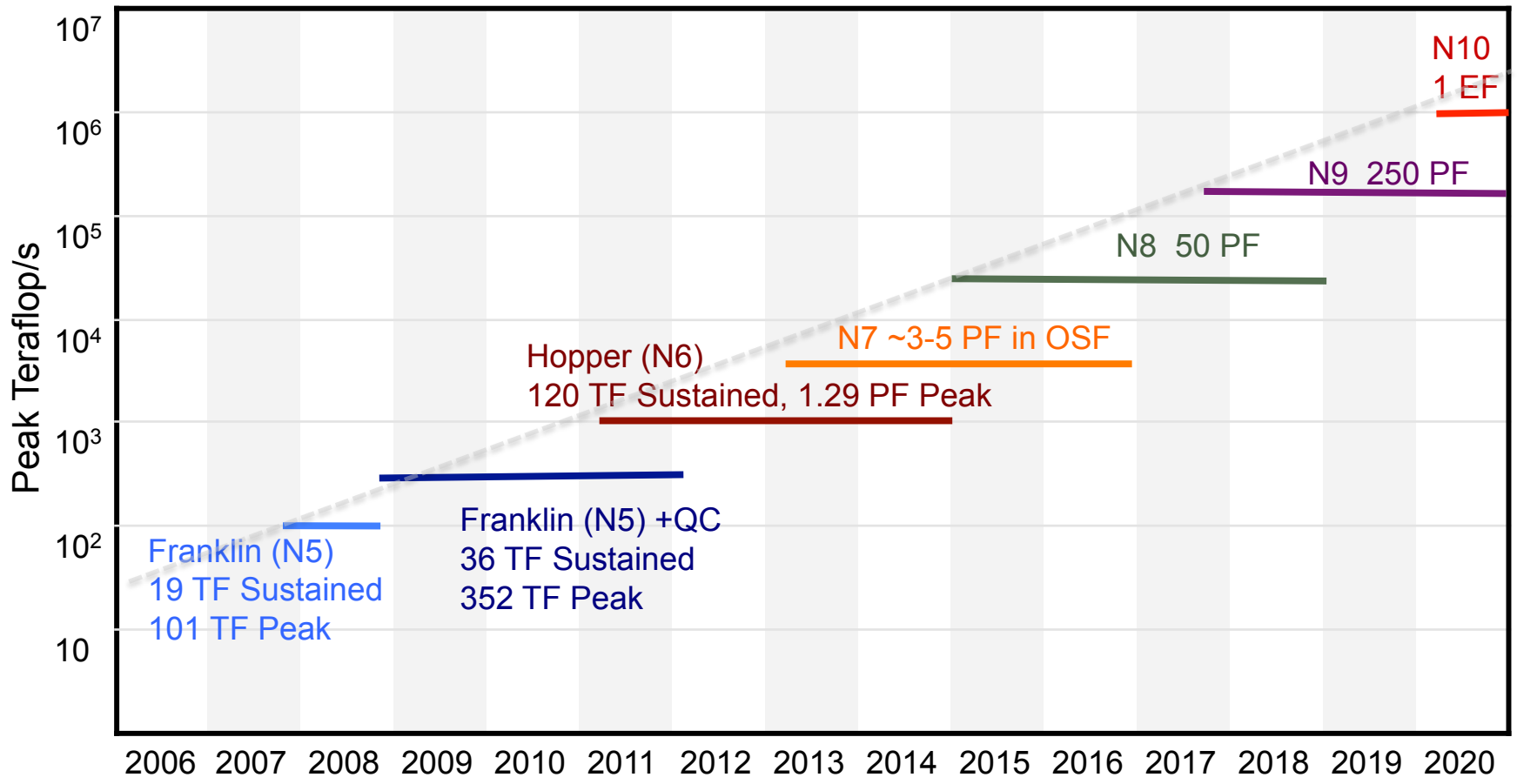
Computing Growth is Not Just an HPC Problem

Microprocessor Performance "Expectation Gap" over Time
(1985-2020 projected)





Expectation Leads to Exascale: NERSC Roadmap



NERSC performance has traditionally grown at 10x every 3-4 years

The Exascale Challenge

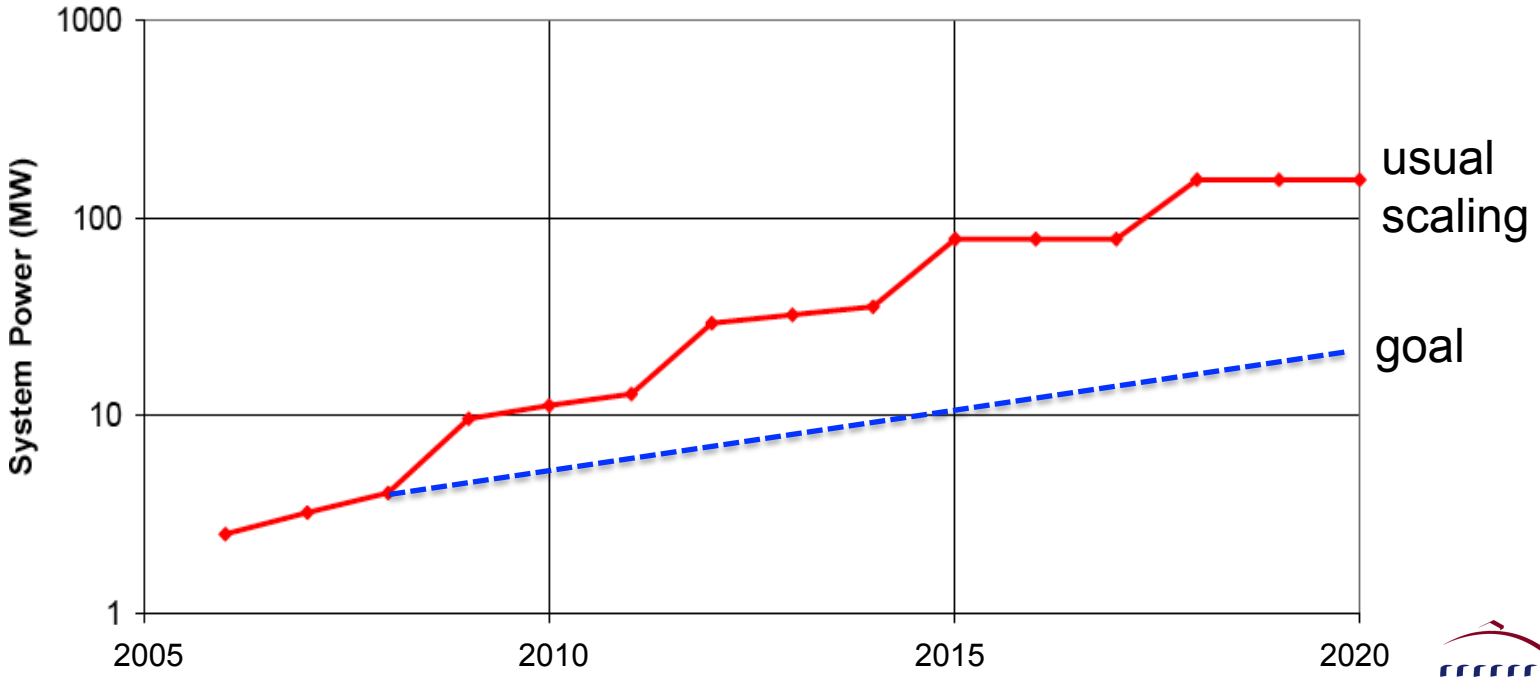
Energy Efficiency

8

Energy Cost Challenge for Computing Facilities

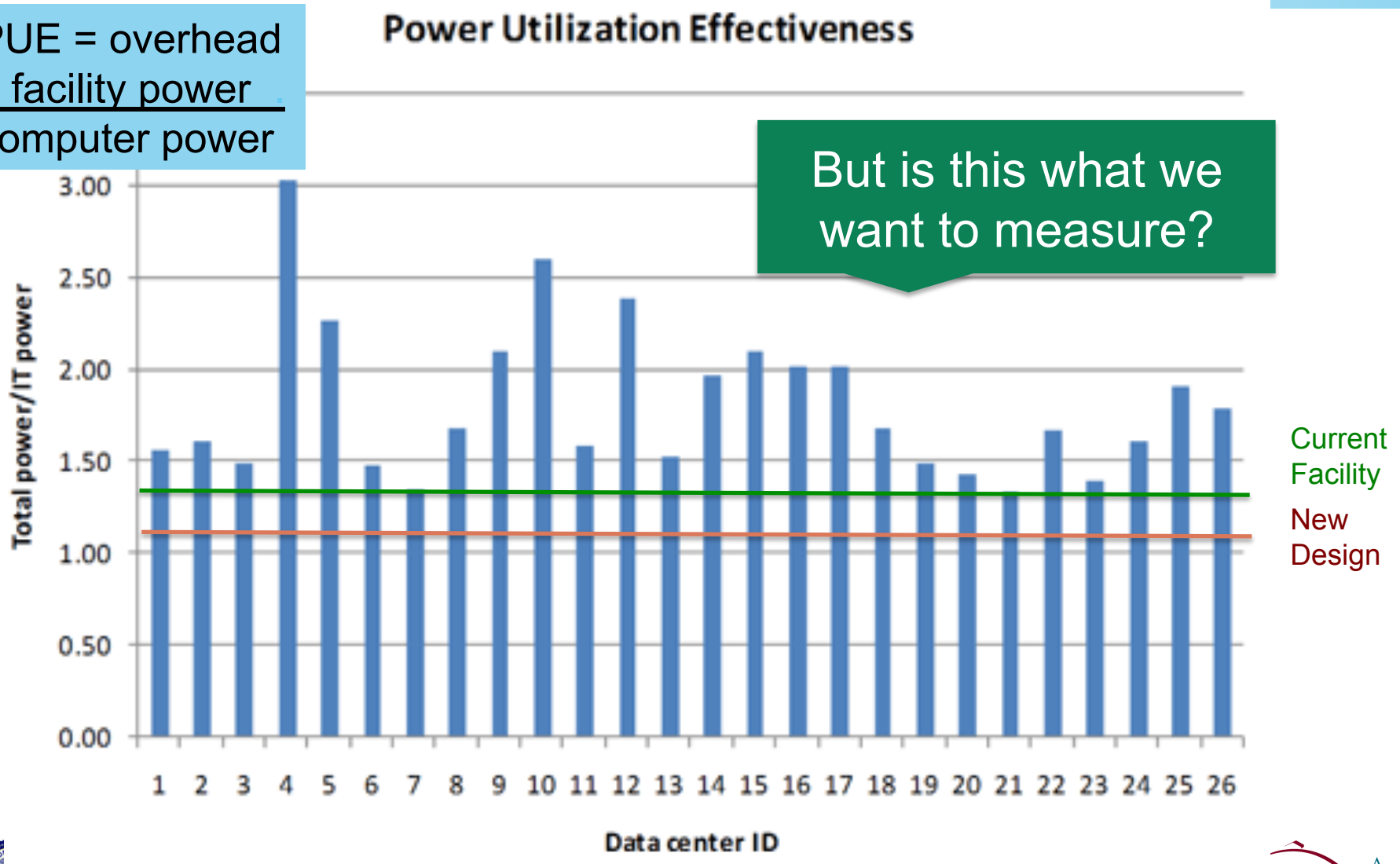
At ~$1M per MW, energy costs are substantial

- 1 petaflop in 2010 will use 3 MW
- 1 exaflop in 2018 possible in 200 MW with “usual” scaling
- 1 exaflop in 2018 at 20 MW is DOE target



PUE of Data Centers

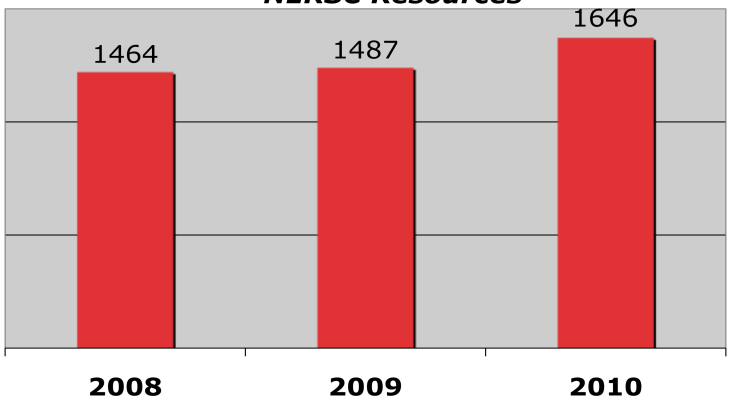
PUE = overhead / computer power
facility power



Measuring Efficiency

- For Scientific Computing centers, the metric should be science output per Watt, if only we could measure that
- If we measure productivity by publications…
- *NERSC in 2010 ran at 450 publications per MW-year*
- Next best: application performance per Watt

Number of Refereed Publications Using NERSC Resources



Reducing power is about architecture & process technology

- **Memory (2x-5x)**
  - New memory interfaces (optimized memory control and xfer)
  - Extend DRAM with non-volatile memory
- **Processor (10x-20x)**
  - Reducing data movement (functional reorganization, > 20x)
  - Domain/Core power gating and aggressive voltage scaling
- **Interconnect (2x-5x)**
  - More interconnect on package
  - Replace long haul copper with integrated optics
- **Data Center Energy Efficiencies (10%-20%)**
  - Higher operating temperature tolerance
  - 480V to the rack and free air/water cooling efficiencies

Slide source: Mark Seager (LLNL)

Anticipating and Influencing the Future

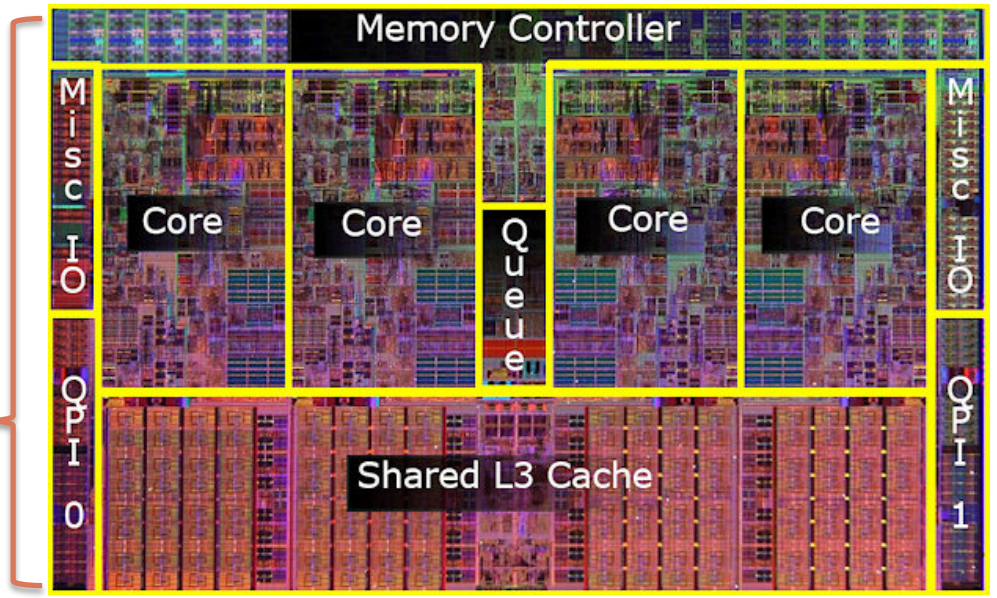
Hardware Design

13

New Processor Designs are Needed to Save Energy

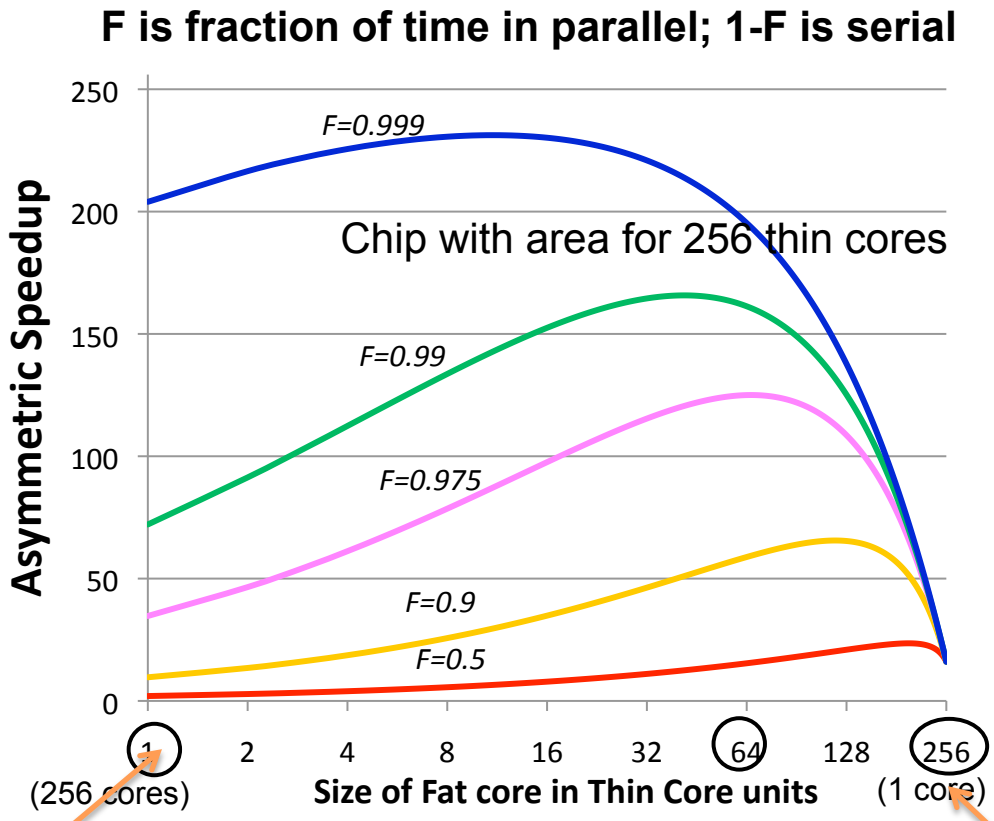
Cell phone processor
(0.1 Watt, 4 Gflop/s)

Server processor
(100 Watts, 50 Gflop/s)



- **Server processors have been designed for performance, not energy**
  - **Graphics processors are 10-100x more efficient**
  - **Embedded processors are 100-1000x**
  - **Need manycore chips with thousands of cores**

The Amdahl Case for Heterogeneity

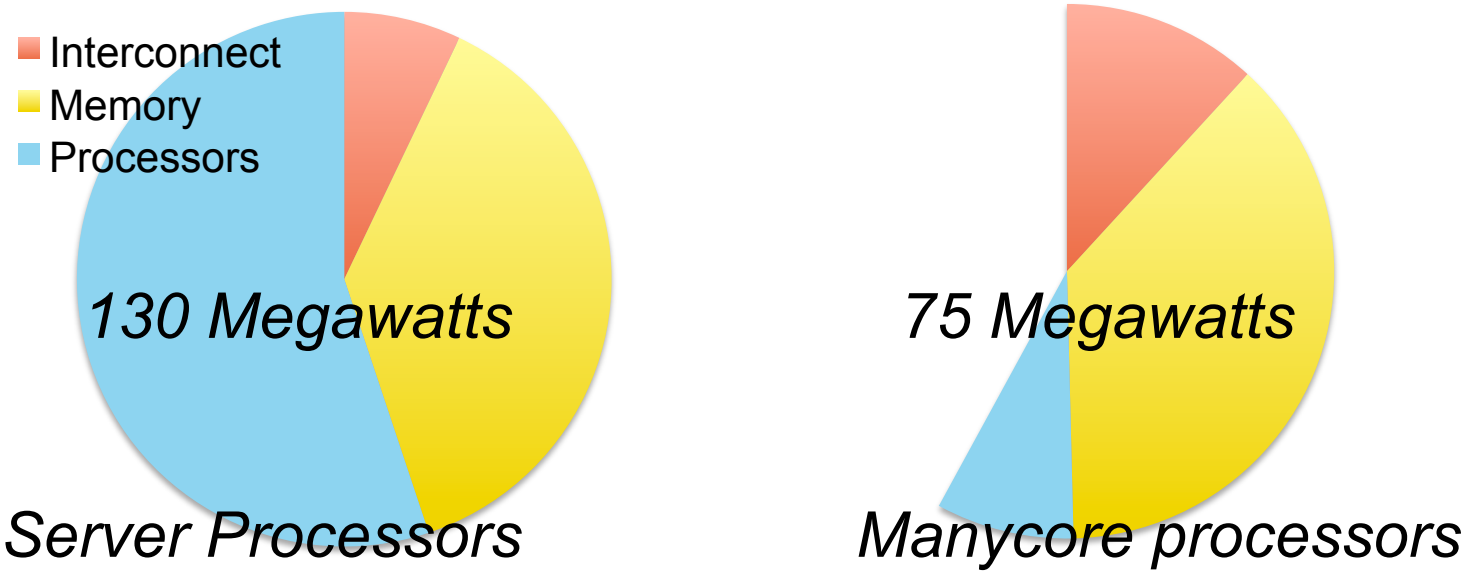


Assumes speedup for Fat / Thin = Sqrt of Area advantage

A Chip with up to 256 “thin” cores and “fat” core that uses some of the some of the thin core area

Heterogeneity Analysis by: Mark Hill, U. Wisc

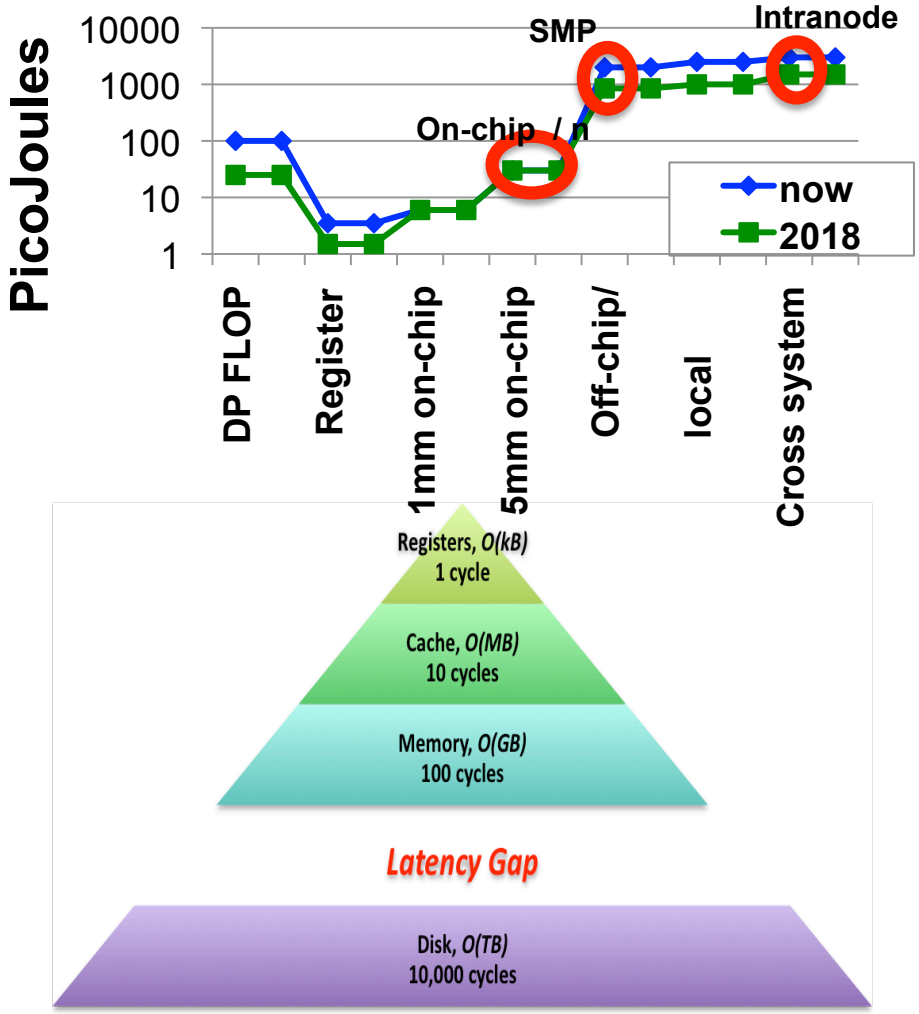
New Processors Means New Software



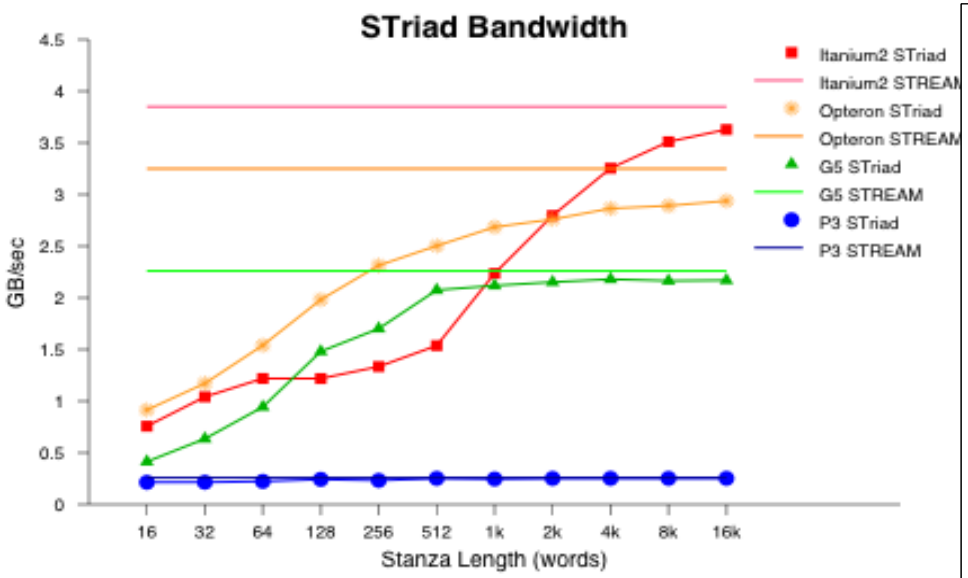
- **Exascale will have chips with thousands of tiny processor cores, and a few large ones**
- **Architecture is an open question:**
  - sea of embedded cores with heavyweight "service" nodes
  - Lightweight cores are accelerators to CPUs
- **Autotuning eases code generation for new architectures**

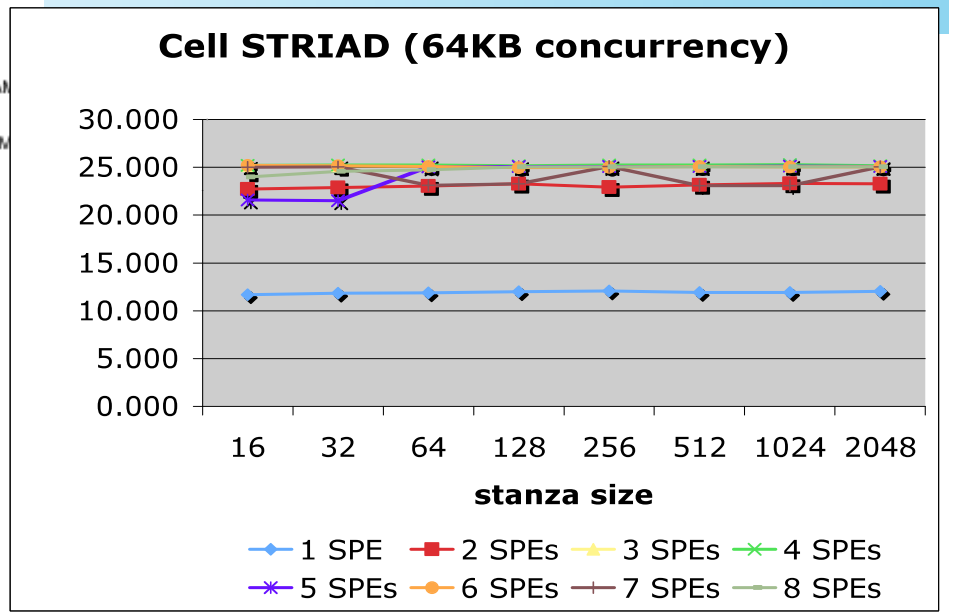
Memory and Storage Challenges for Exascale

- **Challenges:**
  - *Height of the memory wall is growing*
  - *Off-chip bandwidth, latency, and poor concurrency throttle performance*
  - *Per-disk performance not improving*
- **Approaches:**
  - *New memory and storage technologies*
    - Advanced packaging (chip stacking)
    - Photonic DRAM interfaces
    - Optical interconnects / routers
    - Non-volatile memory gap fillers
  - *New Storage Approaches*
    - Software-managed memory hierarchies
    - Communications optimal algorithms
    - Storage efficient programming models (Global Address Space)



Value of Local Store Memory



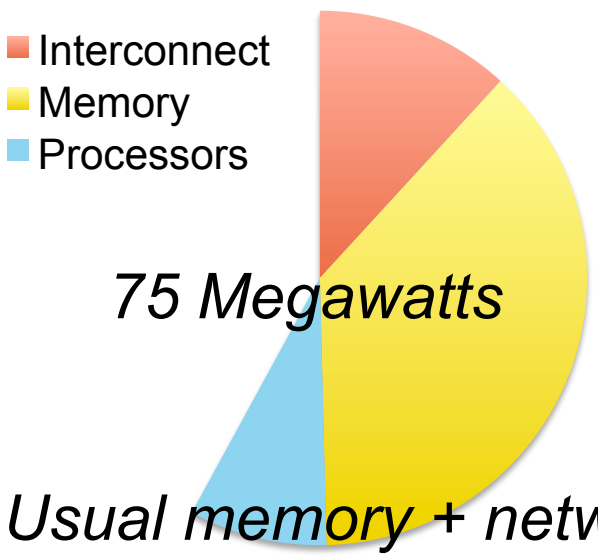


- Unit stride access is as important as cache utilization on processors that rely on hardware prefetch
  - Tiling in unit stride direction is counter-productive: improves reuse, but kills prefetch effectiveness
- Software controlled memory gives programmers more control
  - Spend bandwidth on what you use; bulk moves (DMA) hide latency

Joint work with Shoaib Kamil, Lenny Oliker, John Shalf, Kaushik Datta

BERKELEY LAB

New Memory and Network Technology to Lower Energy



75 Megawatts

Usual memory + network

25 Megawatts

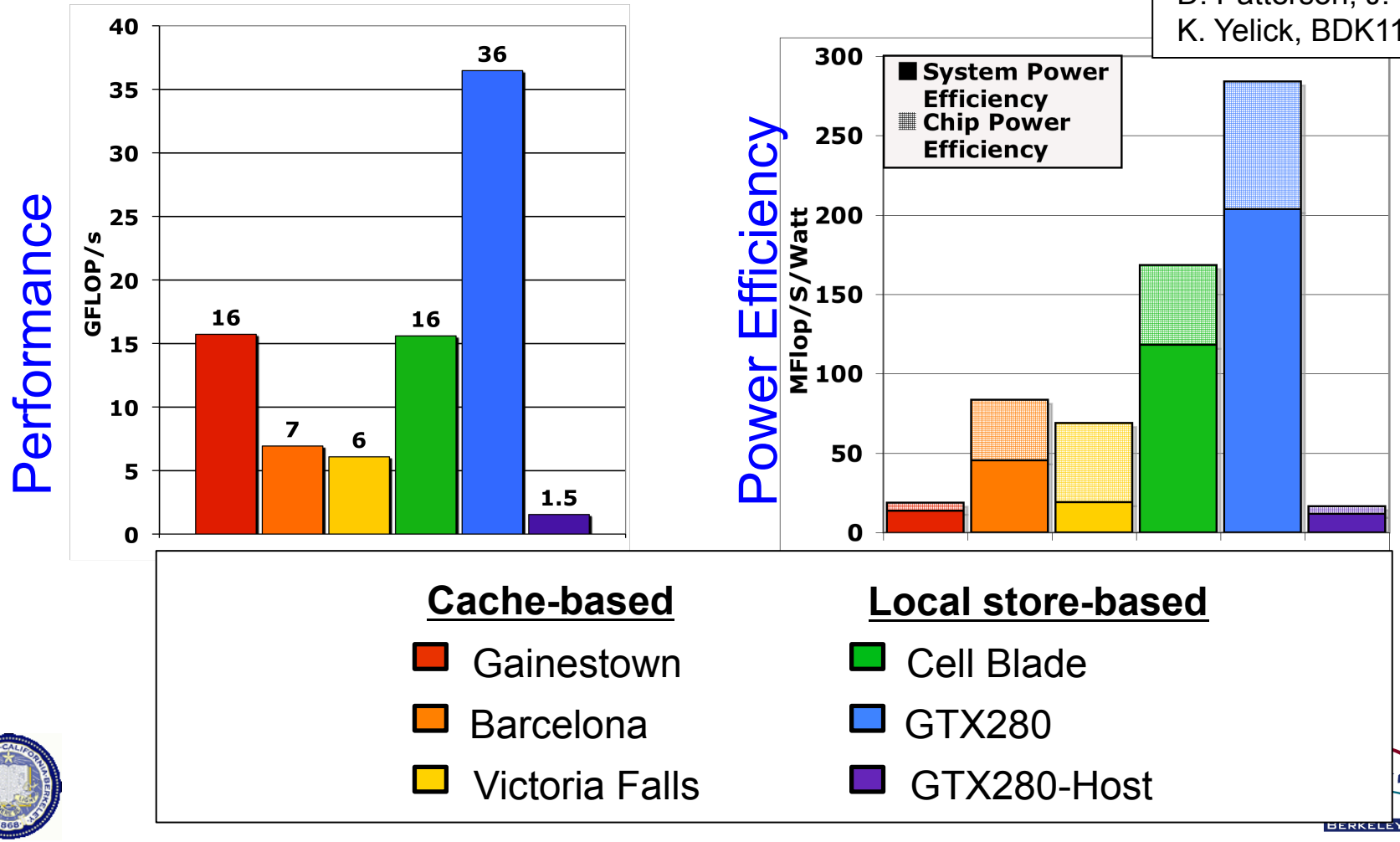
New memory + network

- **Memory as important as processors in energy**
  - Latency is physics, bandwidth is money
  - Software managed memory or cache hybrids
  - Autotuning has helped with that management
  - Need to raise level of autotuning to higher level kernels

Energy Efficiency of Applications

K. Datta, M. Murphy, V. Volkov, S. Williams , J. Carter, L. Oliker. D. Patterson, J. Shalf, K. Yelick, BDK11 book

7-point stencil aggressively autotuned



What Heterogeneity Means to Me

- **Case for heterogeneity**
  - Many small cores are needed for energy efficiency and power density; could have their own PC or use a wide SIMD
  - Need one fat core (at least) for running the OS
- **Local store, explicitly managed memory hierarchy**
  - More efficient (get only what you need) and simpler to implement in hardware
- **Co-Processor interface between CPU and Accelerator**
  - Market: GPUs are separate chips for specific domains
  - Control: Why are the minority CPUs in charge?
  - Communication: The bus is a significant bottleneck.
  - *Do we really have to do this? Isn't parallel programming hard enough*

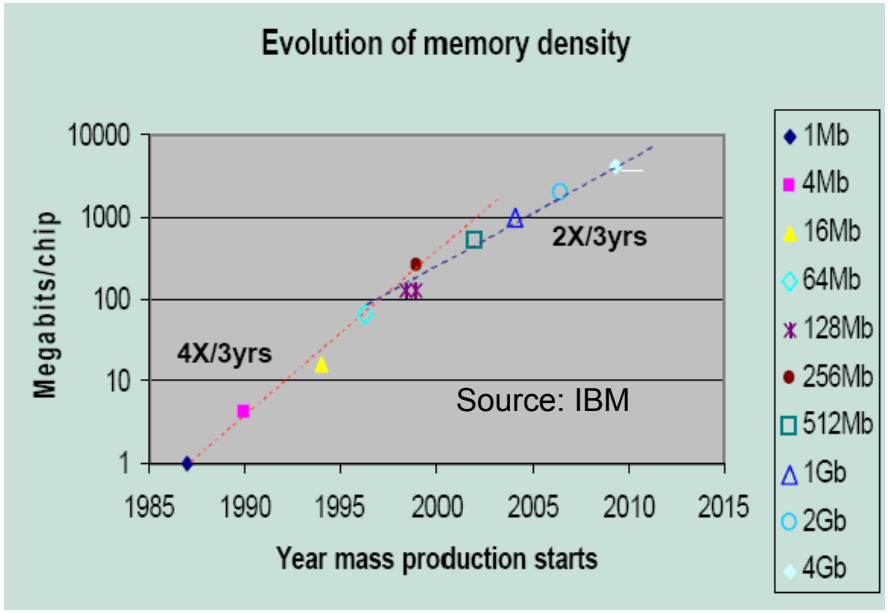
The Future of Software Design and Programming Models

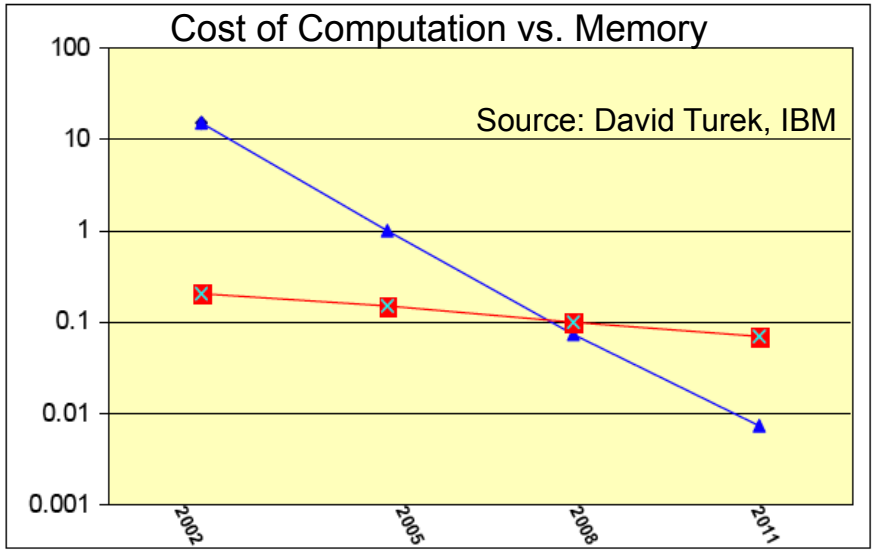
- Memory model
- Control model
- Resilience

Memory is Not Keeping Pace

Technology trends against a constant or increasing memory per core

- Memory density is doubling every three years; processor logic is every two
- Storage costs (dollars/Mbyte) are dropping gradually compared to logic costs





The cost to sense, collect, generate and calculate data is declining much faster than the cost to access, manage and store it

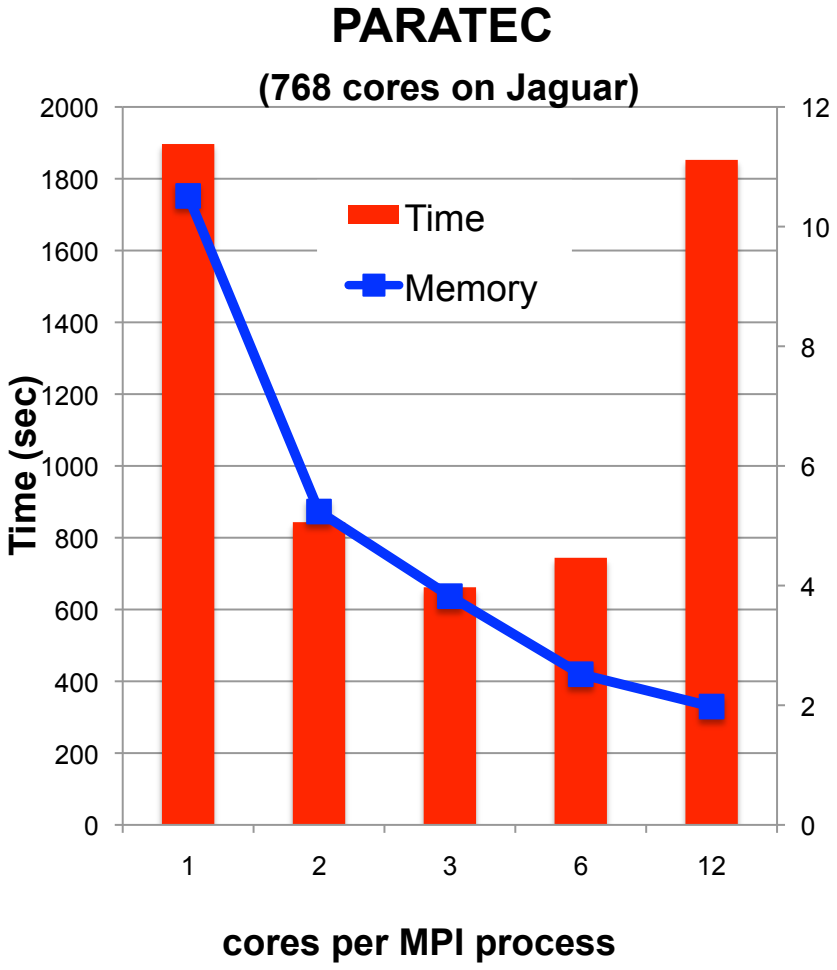
Question: *Can you double concurrency without doubling memory?*

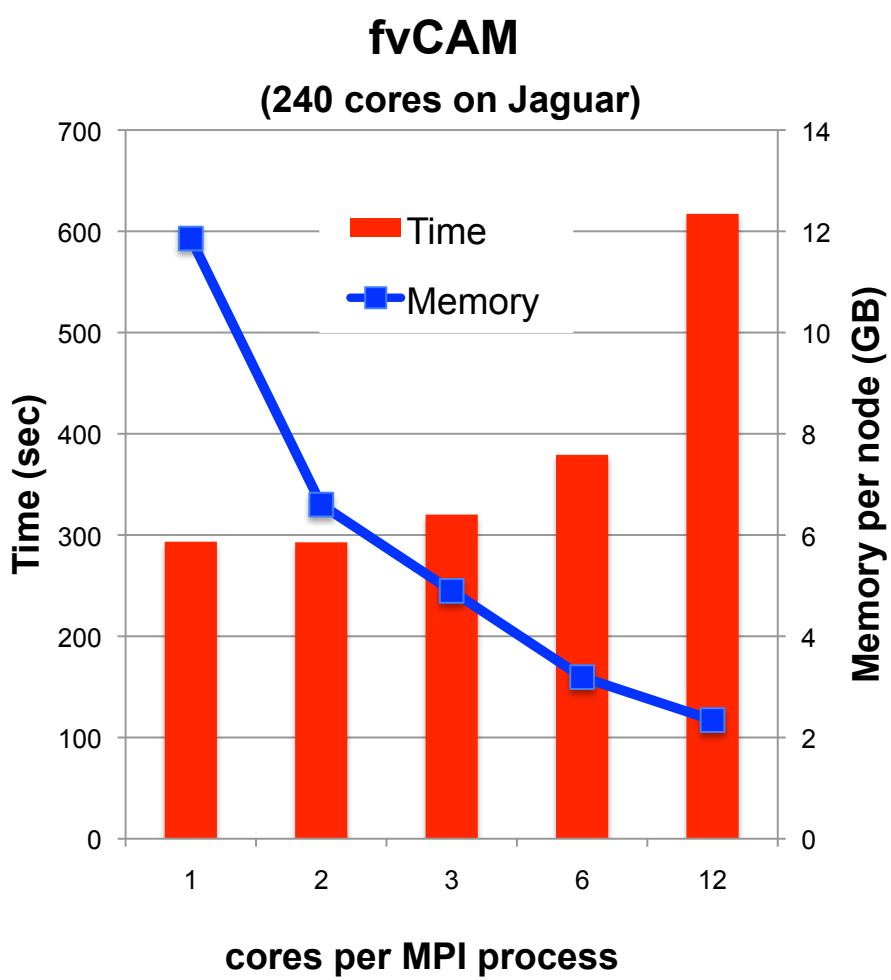
23

What's Wrong with Flat MPI?

- We can run 1 MPI process per core
  - This works now for Quad-Core on Franklin
- How long will it continue working?
  - 4 - 8 cores? Probably. 128 - 1024 cores? Probably not.
- 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?
- This means a "new" model for most NERSC users

What's Wrong with Flat MPI?



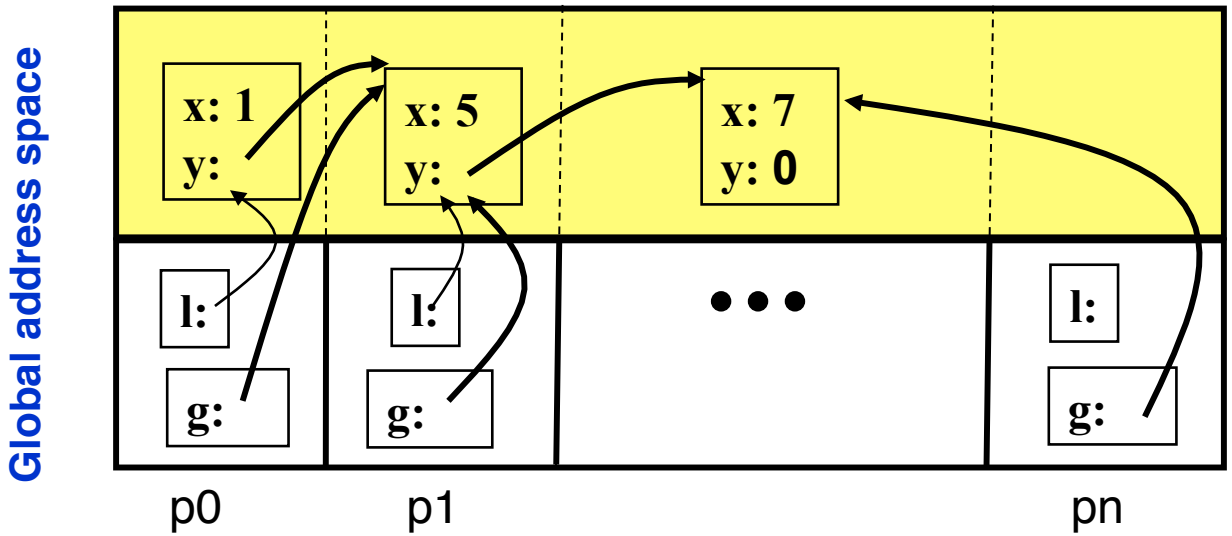


Hybrid Programming is key to saving memory (2011) and sometimes improves performance

Why Use 2 Programming Models When 1 Will Do?

Global address space: thread may directly read/write remote data

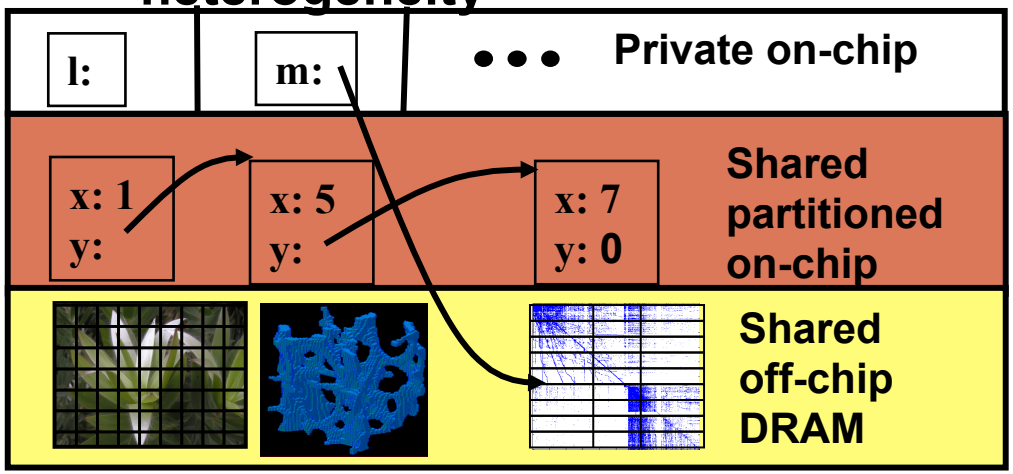
Partitioned: data is designated as local or global

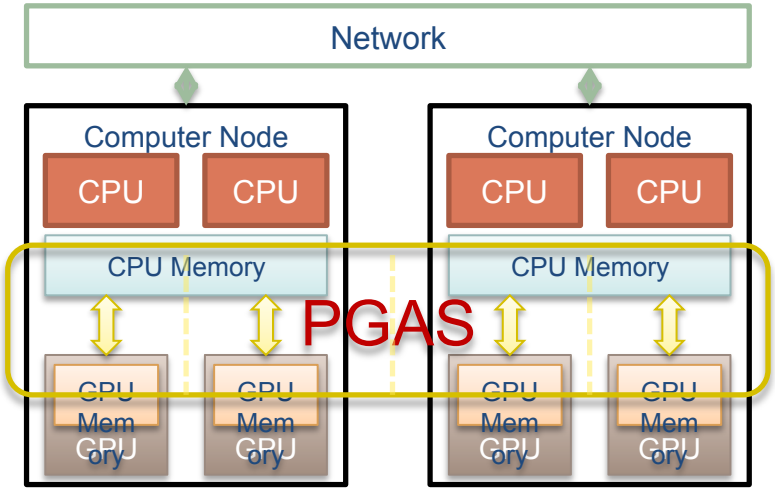


- Affinity control for shared and distributed memory
- No less scalable than message passing
- Permits sharing, unlike message passing
- One-sided communication: never say “receive”

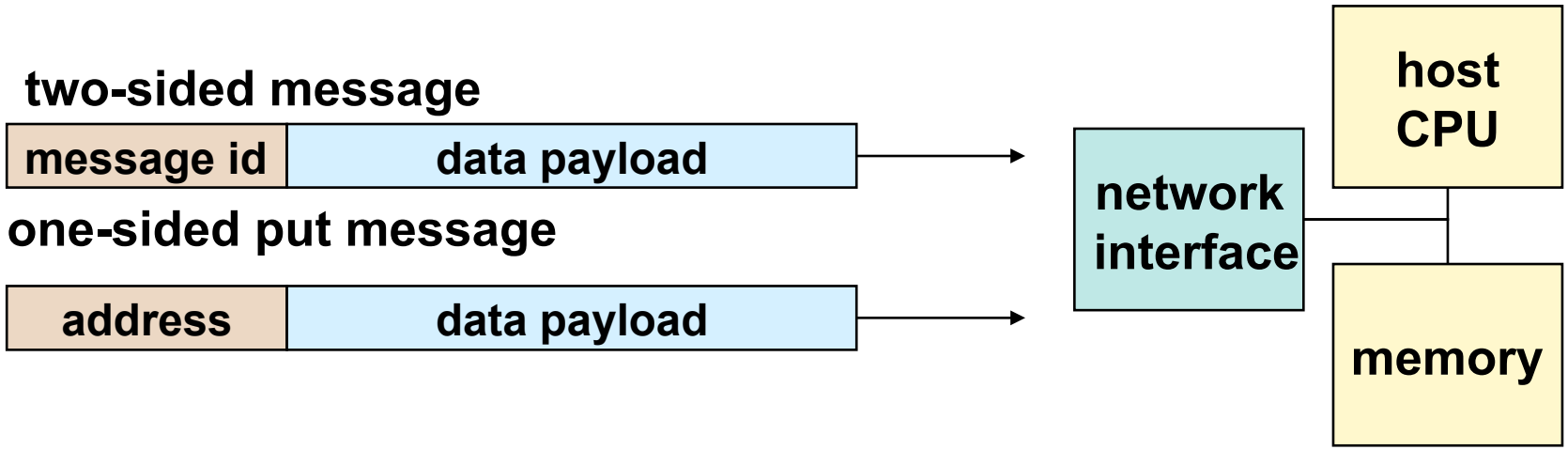
DMA ~~PGAS~~ Languages for Manycore

- PGAS memory are a good fit to machines with explicitly managed memory (local store)
  - Global address space implemented as DMA reads/writes
  - New "vertical" partition of memory needed for on/off chip, e.g., upc_offchip_alloc
  - Non-blocking features of UPC put/get are useful
- SPMD execution model needs to be adapted to heterogeneity





Avoiding Synchronization in Communication

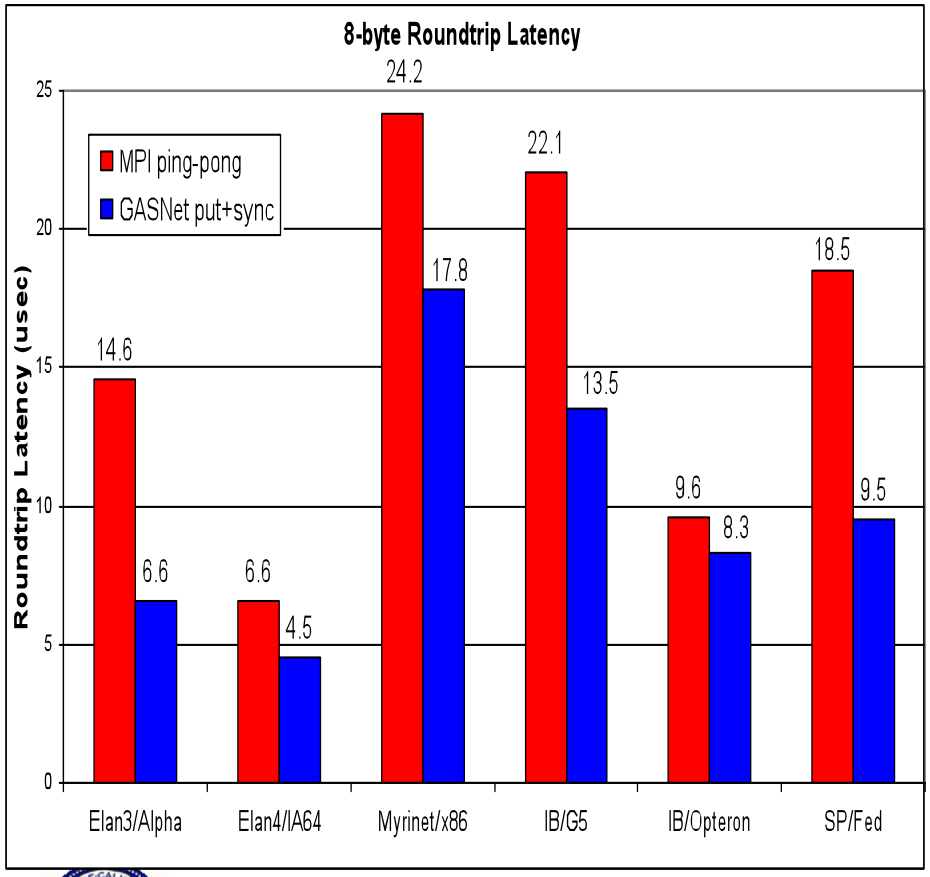


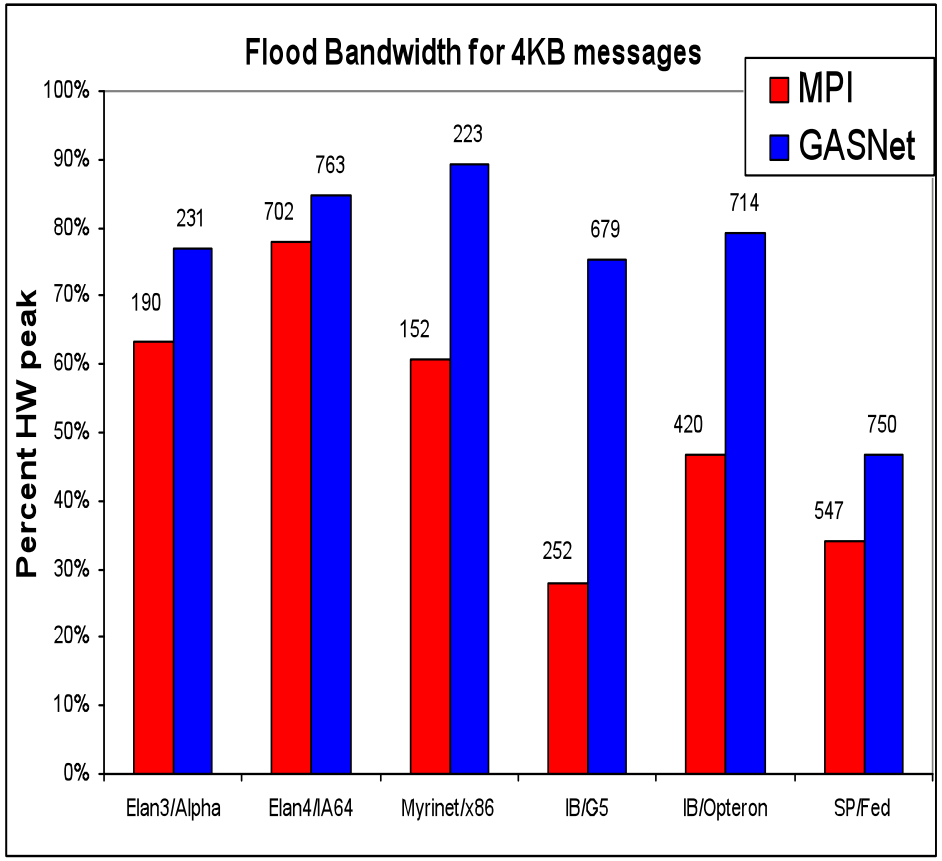
- Two-sided message passing (e.g., MPI) requires matching a send with a receive to identify memory address to put data
  - Wildly popular in HPC, but cumbersome in some applications
  - Couples data transfer with synchronization
- Using global address space decouples synchronization
  - Pay for what you need!
  - Note: Global Addressing ≠ Cache Coherent Shared memory

Joint work with Dan Bonachea, Paul Hargrove, Rajesh Nishtala and rest of UPC group

One-Sided Communication Avoids Unnecessary Overheads

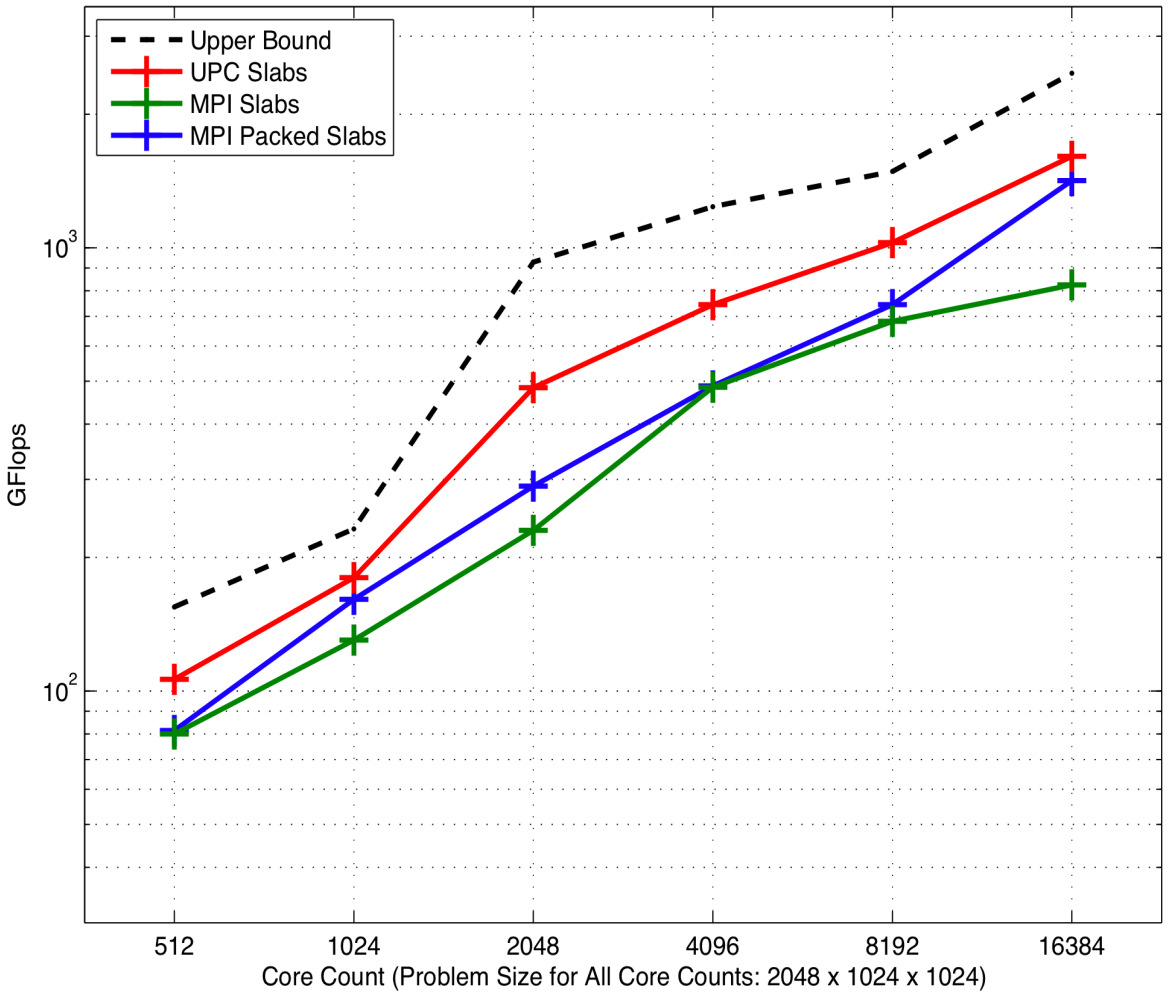
Comparison of MPI to GASNet (LBNL/UCB one-sided communication layer)





Joint work with Berkeley UPC Group

3D FFT on BlueGene/P

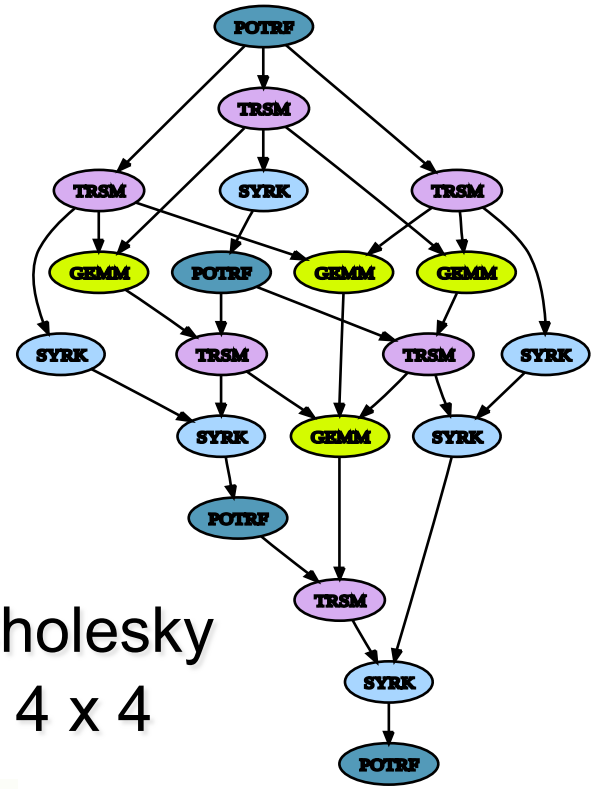


Joint work with Rajesh Nishtala, Dan Bonachea, Paul Hargrove,and rest of UPC group

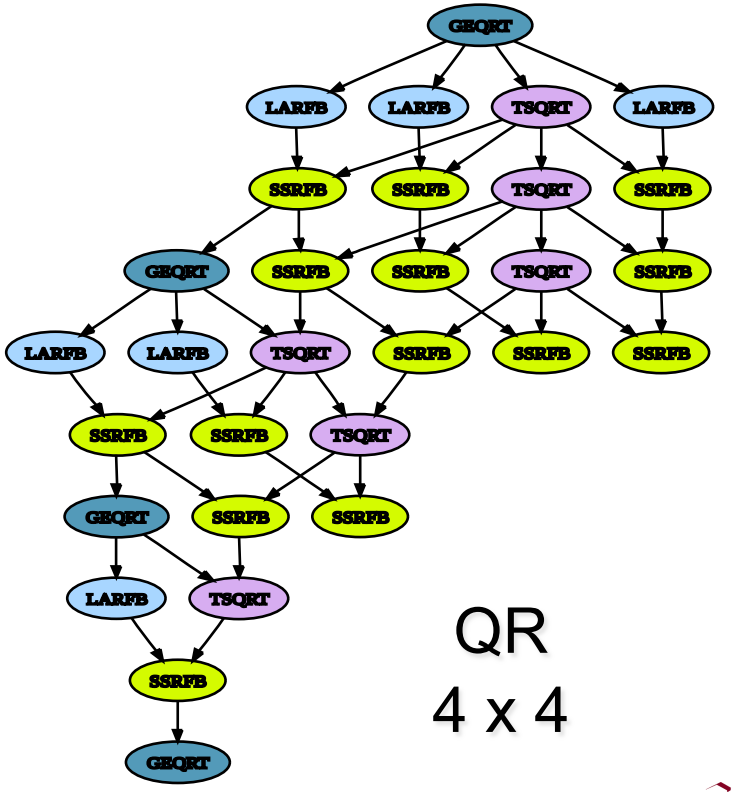
Avoid Synchronization in Applications

Computations as DAGs

View parallel executions as the directed acyclic graph of the computation



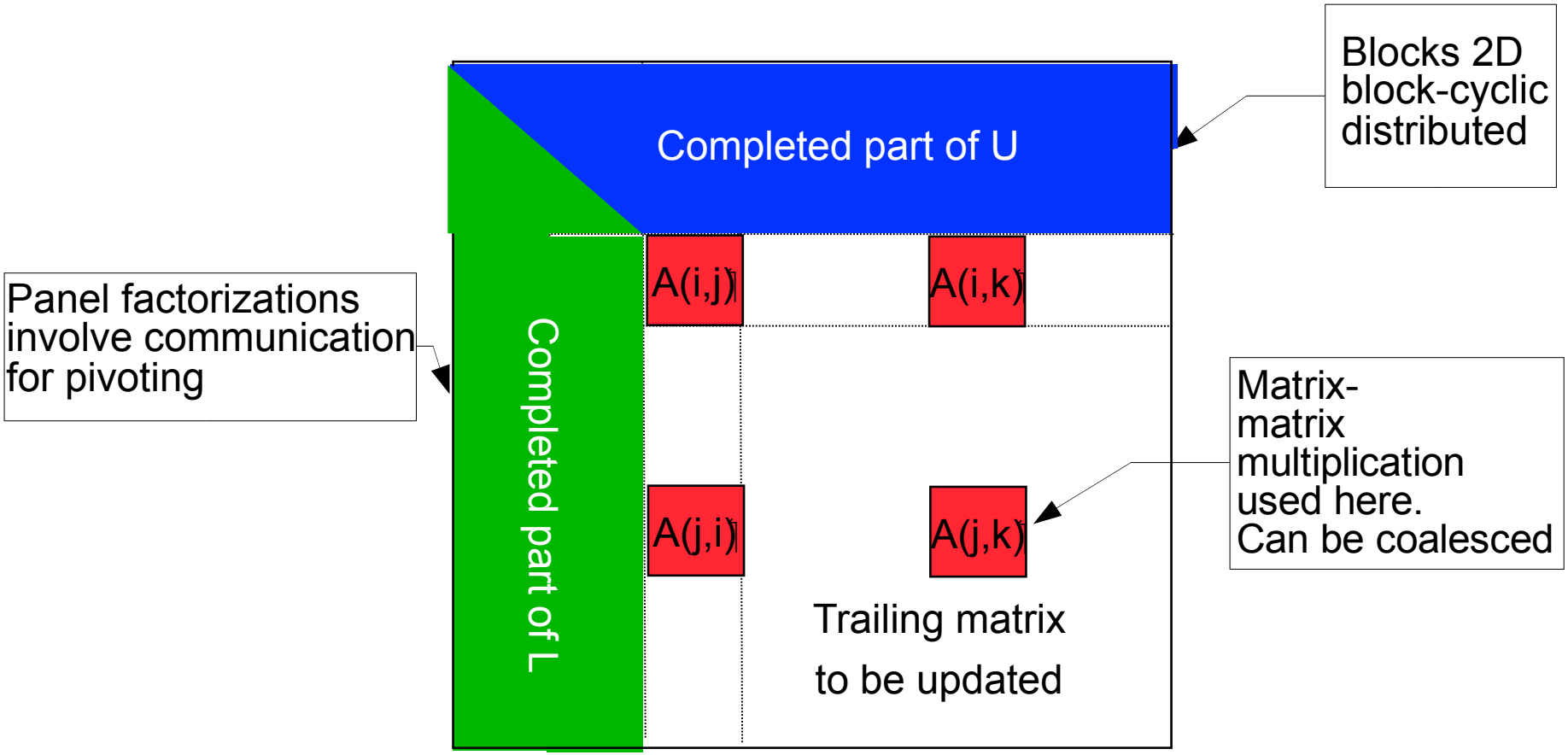
Cholesky
4 x 4



QR
4 x 4

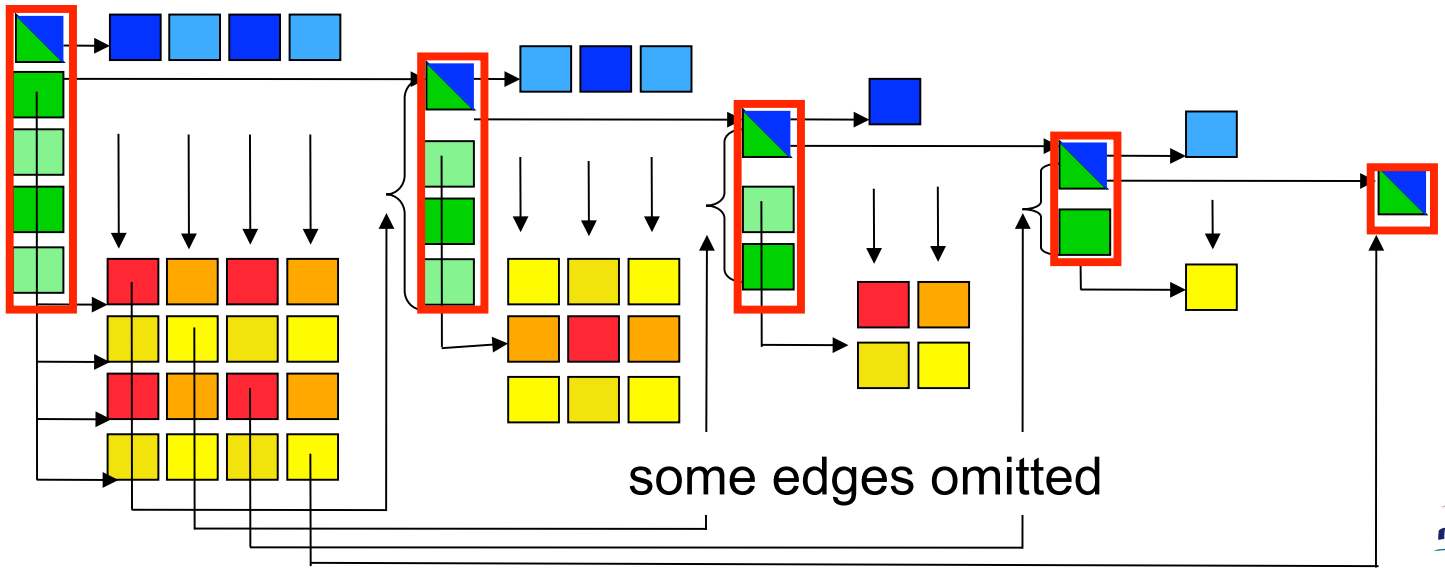
Slide source: Jack Dongarra

Parallel LU Factorization



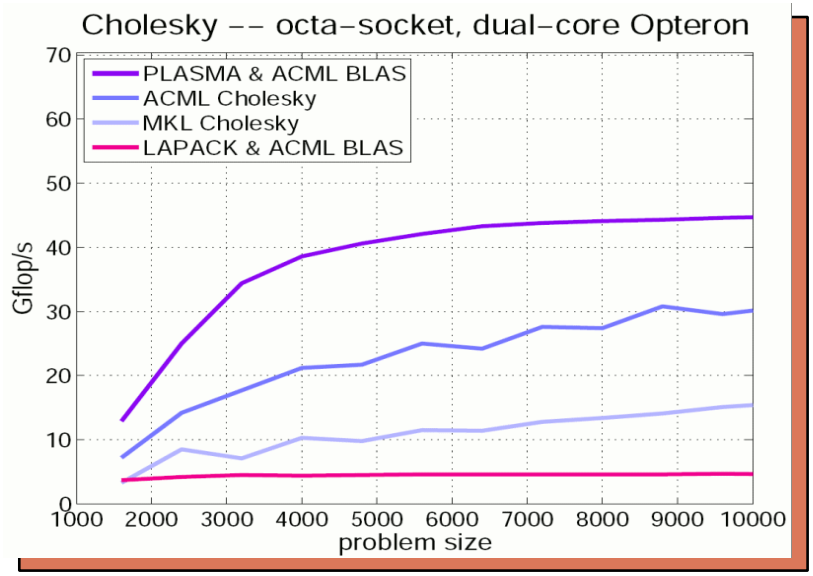
Event Driven Execution of LU Factorization

- Ordering needs to be imposed on the schedule
- Critical operation: Panel Factorization
  - need to satisfy its dependencies first
  - perform trailing matrix updates with low block numbers first
  - “memory constrained” lookahead
- General issue: dynamic scheduling in partitioned memory
  - Can deadlock memory allocator!

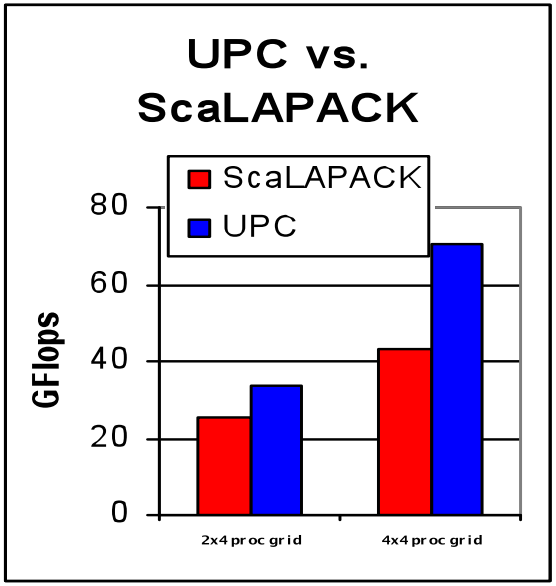


DAG Scheduling Outperforms Bulk-Synchronous Style

PLASMA on shared memory



UPC on partitioned memory



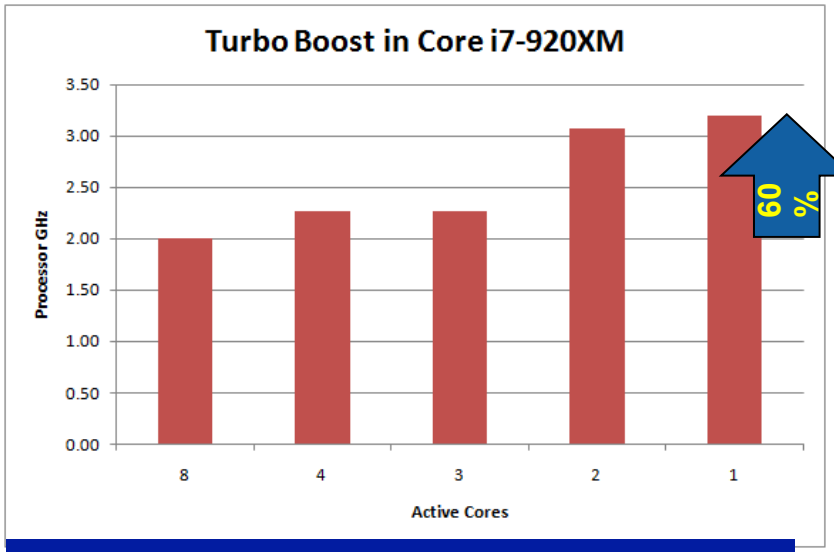
UPC LU factorization code adds cooperative (non-preemptive) threads for latency hiding

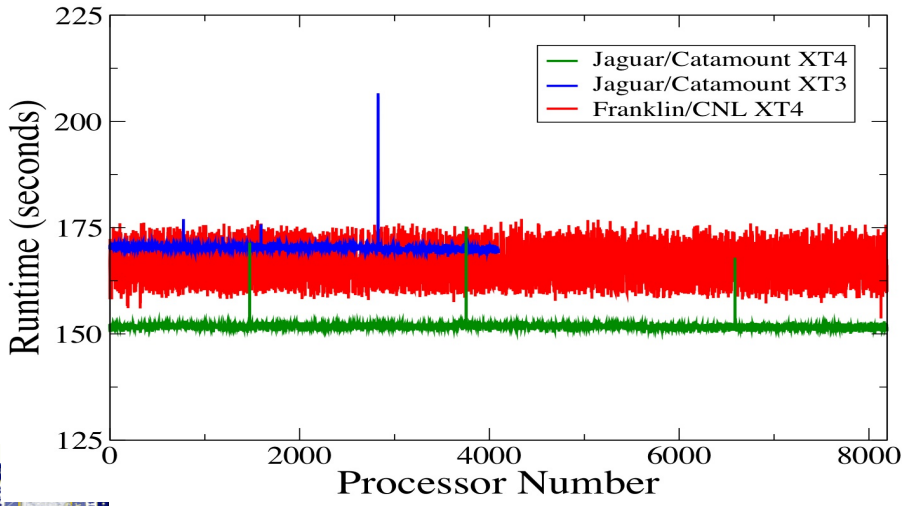
- **New problem in partitioned memory: allocator deadlock**
- **Can run on of memory locally due tounlucky execution order**

PLASMA by Dongarra et al; UPC LU joint with Parray Husbands

Hardware and Software Scaling Require New Resilience Models

- **Resiliency challenges**
  - Chance of component failure grows with system / job size
  - Failure and power management → irregular performance behavior
- **Hardware / software**
  - Component values should not cause system-wide or application-wide outages





Software assumption that all processors run at the same speed:

- Clock speed may change due to temperature and power
- Failures in memory system may also affect performance

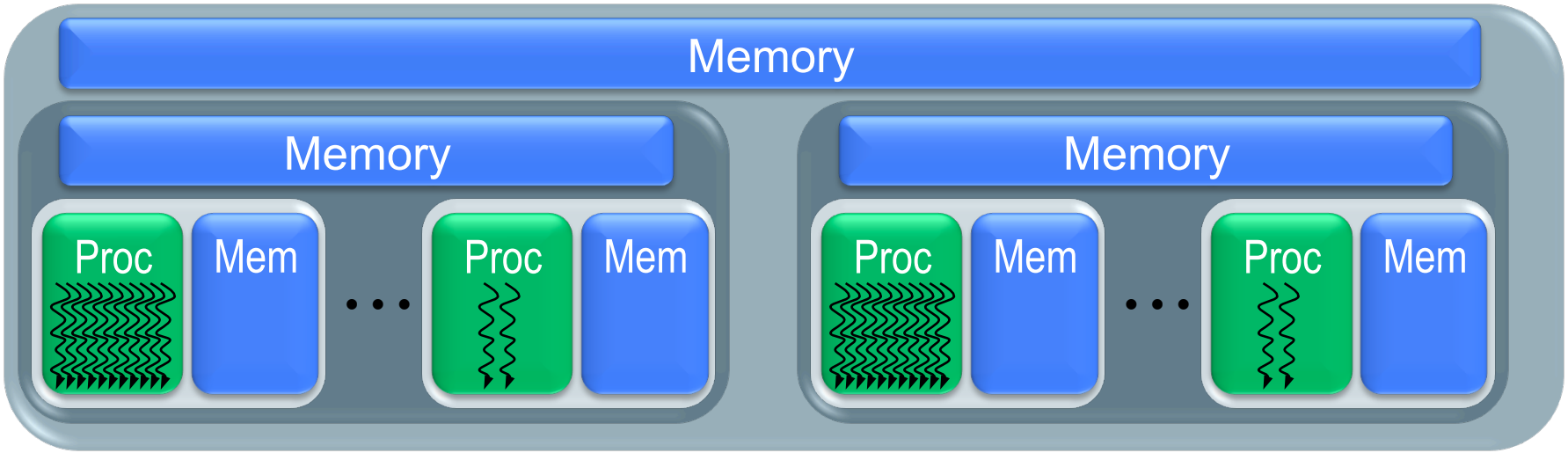
To Virtualize or Not

- The fundamental question facing in parallel programming models is:

  What should be virtualized?

- Hardware has finite resources
  - Processor count, number of registers, caches size, are finite
  - Hide this from programmer (express all parallelism and locality) or expose it (express what is important)
- Locality vs Load balance is fundamental trade-off:
  - Most successful examples of locality-important applications/ machines use static scheduling
  - Unless they have a dynamic task graph so it is impossible
- Two extremes are well-studied
  - Dynamic parallelism without locality
  - Static parallelism (with threads = processors) with locality

Hierarchical Partitioned Global Address Space Programming

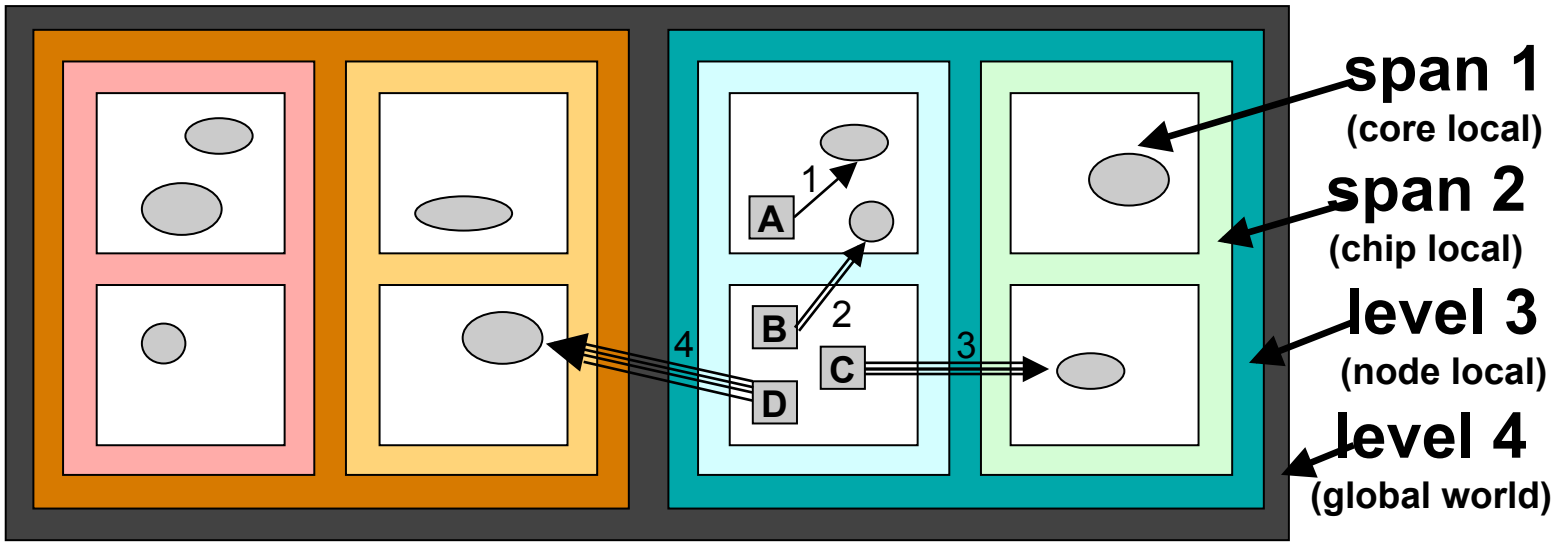


- **Partitioned Global Address Space languages**
  - **Ability to control data location and movement through partitioning**
  - **Ability to move data without synchronizing**
- **Challenges for future systems**
  - **Heterogeneity: fine and coarse-grained parallelism**
  - **Data parallel (efficiency & simplicity) and task parallel (generality)**
  - **More than one level of partitioning**

Echelon ProgSys Team: Michael Garland, Alex Aiken, Brad Chamberlain, Mary Hall, Greg Titus, Kathy Yelick

Hierarchical Pointers in a PGAS Memory Model

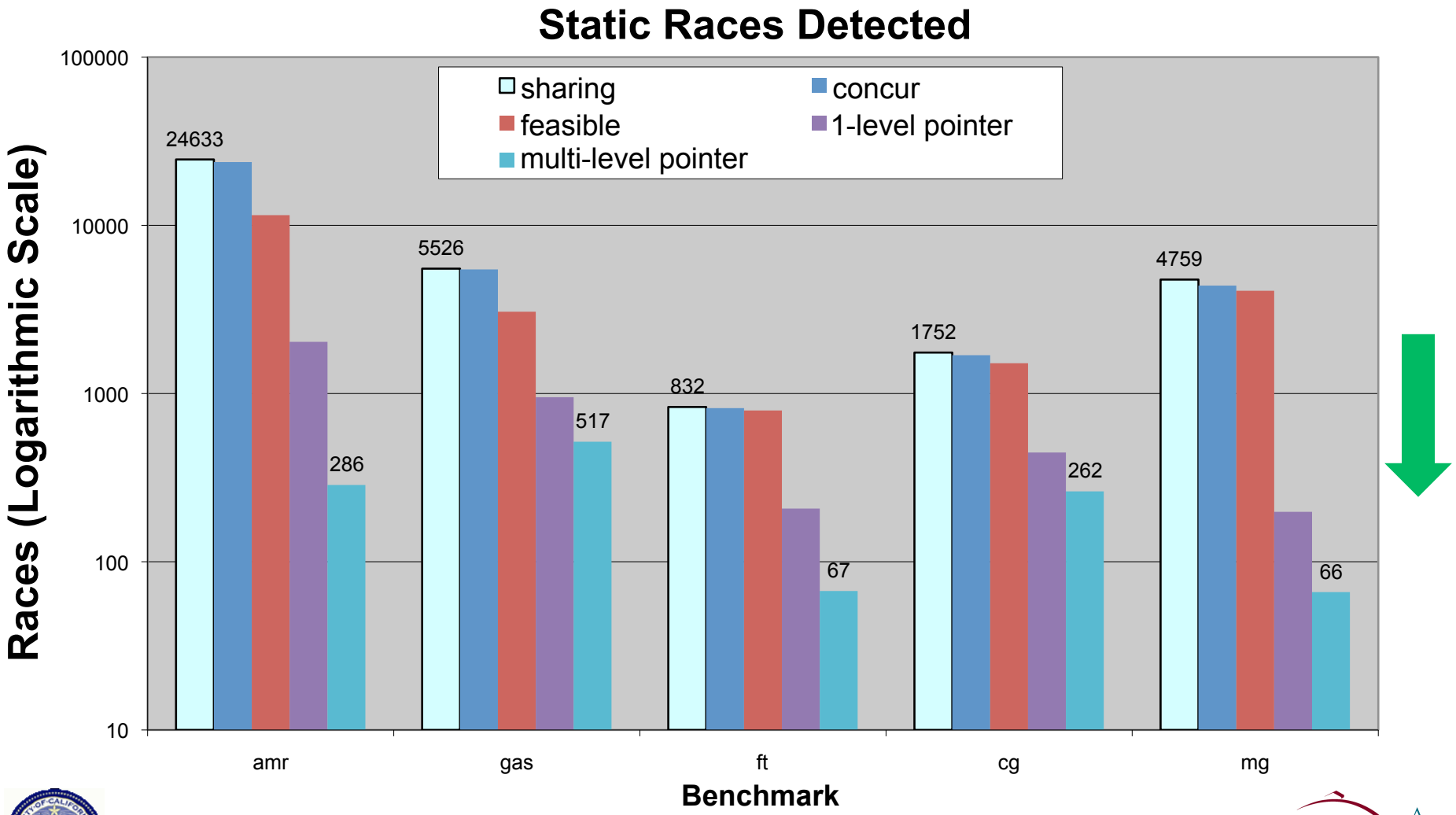
- A global address space for hierarchical machines may have multiple kinds of pointers
- These can be encode by programmers in type system or hidden, e.g., all global or only local/global
- This partitioning is about pointer span, not control / parallelism



Type Systems Help: Hierarchical Pointer Analysis

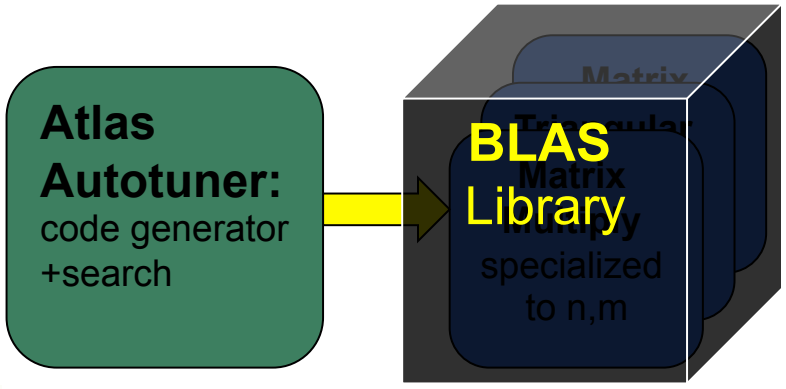
- Demonstrated in Titanium (Java-based PGAS)
- Statically determines what pointer reach
- Allocation sites approximated as *abstract locations (alocs)*
- All explicit and implicit program variables have points-to sets
  - The set of alocs that the variable may reference
- Abstract locations have reach or "span"
  - Thread local locations created by local thread
  - Process local locations reside in same memory space
  - Global locations can be anywhere

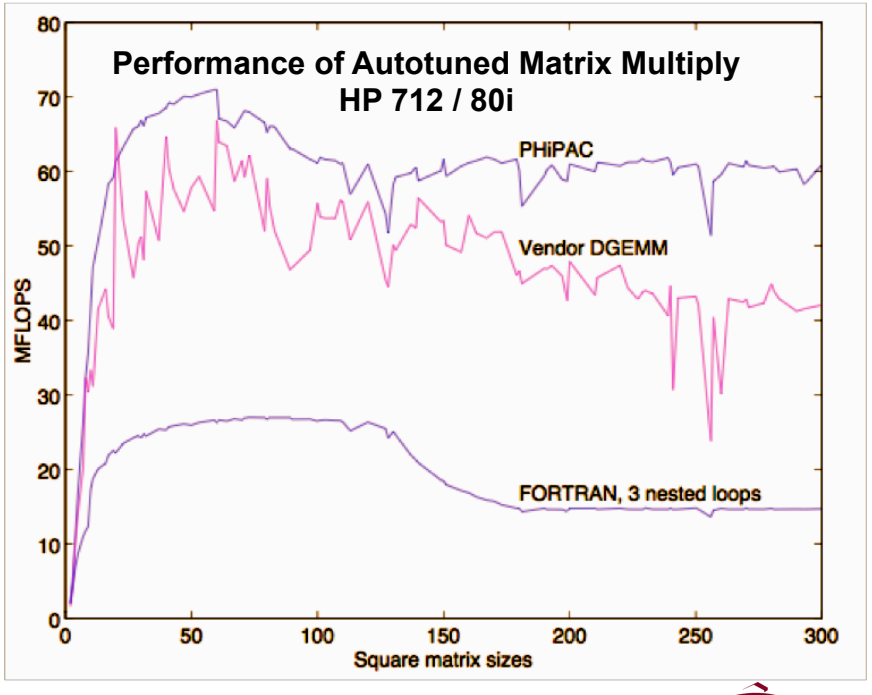
Race Detection Results



Autotuning: Write Code Generators

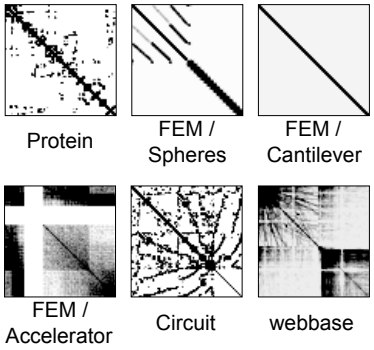
- **Autotuners are code generators plus search algorithms to find best code**
- **Avoids compiler problems of dependence analysis and approximate performance models**
  - ***Functional portability*** from C
  - ***Performance portability*** from search at install time

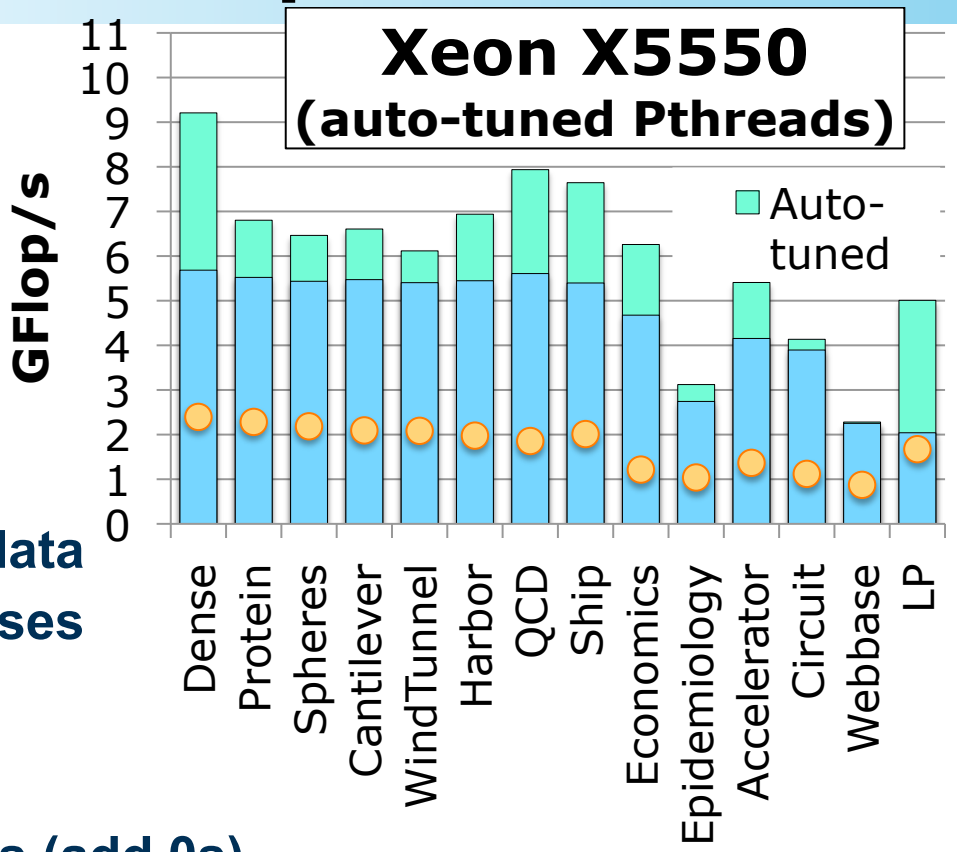




BLAS = Basic Linear Algebra Subroutine: matrix multiply, etc.

Autotuners for Input-Dependence Optimizations





- **Sparse Matrix**
  - Significant index meta data
  - Irregular memory accesses
  - Memory bound
- **Autotuning**
  - Tune over data structures (add 0s)
  - Delayed tuning decisions until runtime
  - Still use significant install-time tuning (dense matrix in sparse format) with online specialization based on matrix structure

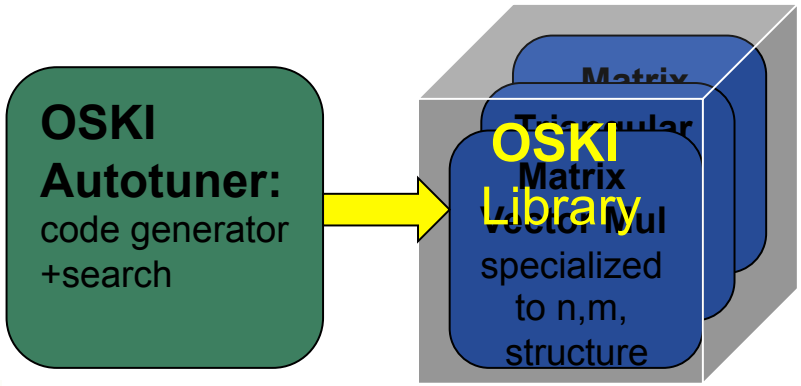
See theses from Im, Vuduc, Williams, and Jain

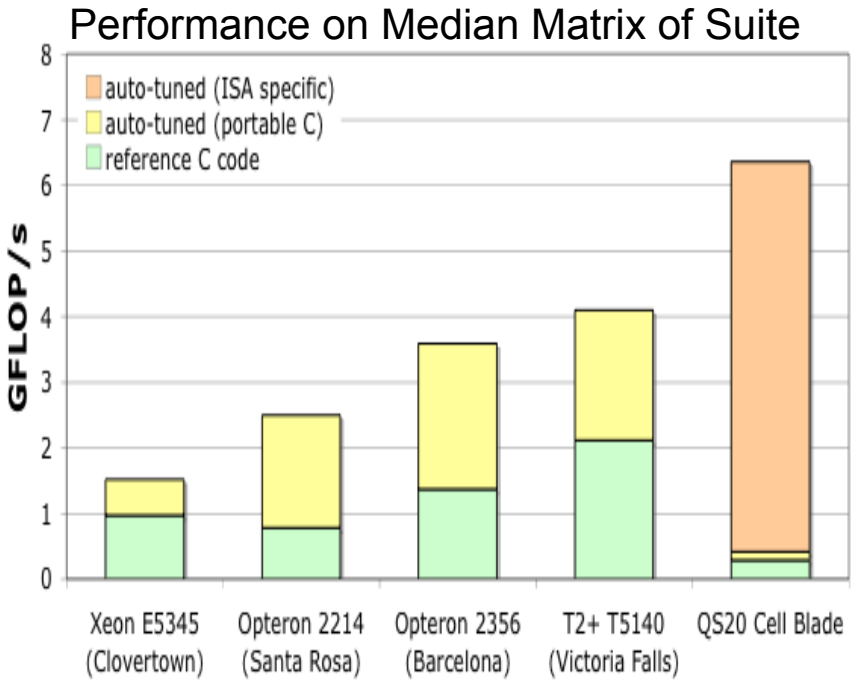
Recent Past Autotuners: Sparse Matrices

- **OSKI: Optimized Sparse Kernel Interface**
- **Optimized for: size, machine, and matrix structure**
- ***Functional portability* from C (except for Cell/GPUs)**

❖ ***Performance portability*** from install time search and model evaluation at runtime

❖ Later tuning, less opaque interface





See theses from Im, Vuduc, Williams, and Jain

Future: Improving Support for Writing Autotuners

- Ruby class encapsulates SG pattern
  - body of anonymous lambda specifies filter function
- Code generator produces OpenMP
  - ~1000-2000x faster than Ruby
  - Minimal per-call runtime overhead

Joint with Shoaib Kamil, Armando Fox, John Shalf.

```
class LaplacianKernel < Kernel
 def kernel(in_grid, out_grid)
  in_grid.each_interior do |point|
   in_grid.neighbors(point,1).each
      do |x|
     out_grid[point] += 0.2*x.val
   end
 end
end
```

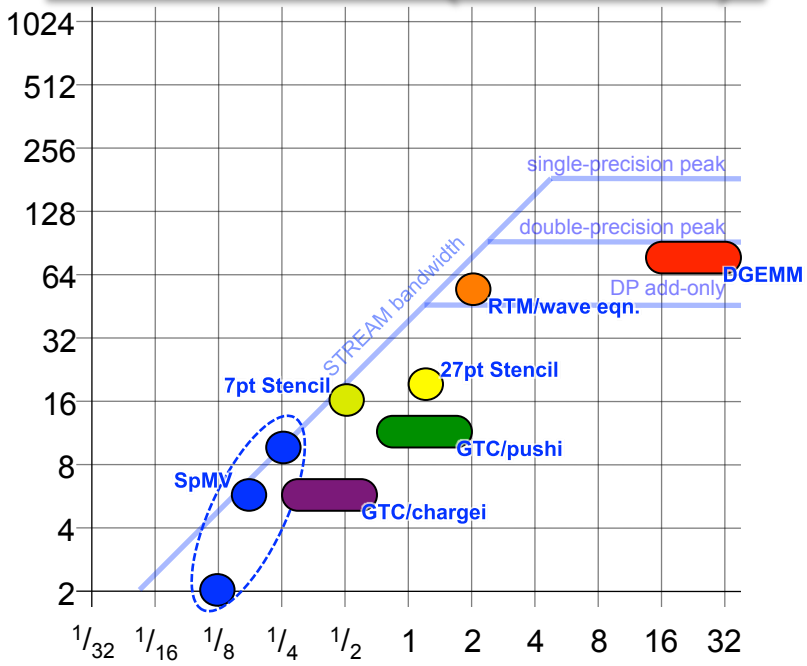
```
VALUE kern_par(int argc, VALUE* argv, VALUE
self) {
unpack_arrays into in_grid and out_grid;

#pragma omp parallel for default(shared)
private (t_6,t_7,t_8)
for (t_8=1; t_8<256-1; t_8++) {
 for (t_7=1; t_7<256-1; t_7++) {
  for (t_6=1; t_6<256-1; t_6++) {
   int center = INDEX(t_6,t_7,t_8);
   out_grid[center] = (out_grid[center]
      +(0.2*in_grid[INDEX(t_6-1,t_7,t_8)]));
   ...
   out_grid[center] = (out_grid[center]
      +(0.2*in_grid[INDEX(t_6,t_7,t_8+1)]));
;}}}
return Qtrue;}
```

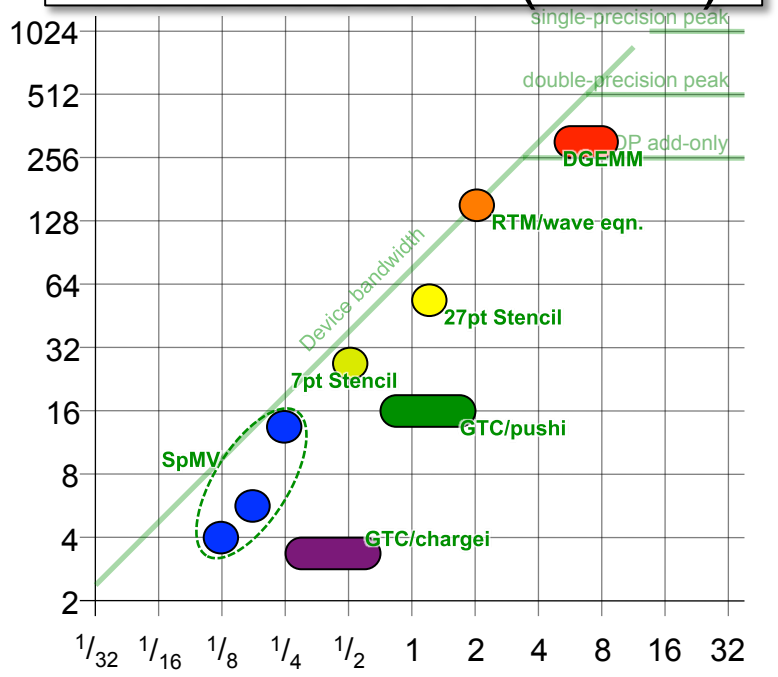
Autotuning Gets Kernel Performance Near Optimal

- Roofline model captures bandwidth and computation limits
- Autotuning gets kernels near the roof

Xeon X5550 (Nehalem)



NVIDIA C2050 (Fermi)



See Sam Williams PhD Thesis & papers for Roofline
Tuning results by large number of Berkeley / LBNL folks

Algorithms to Optimize for Communication

"Moore's Law" for combustion simulations

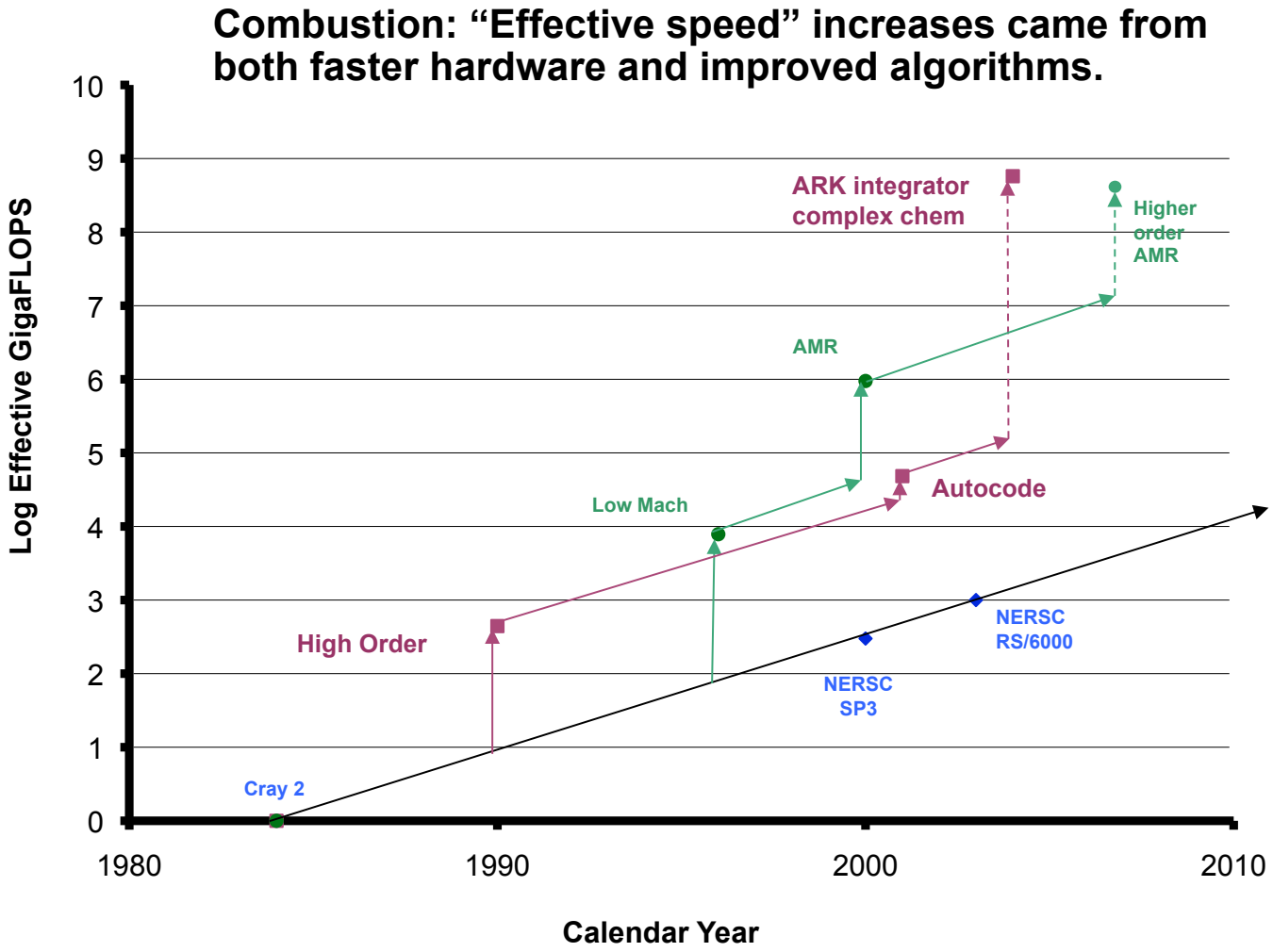
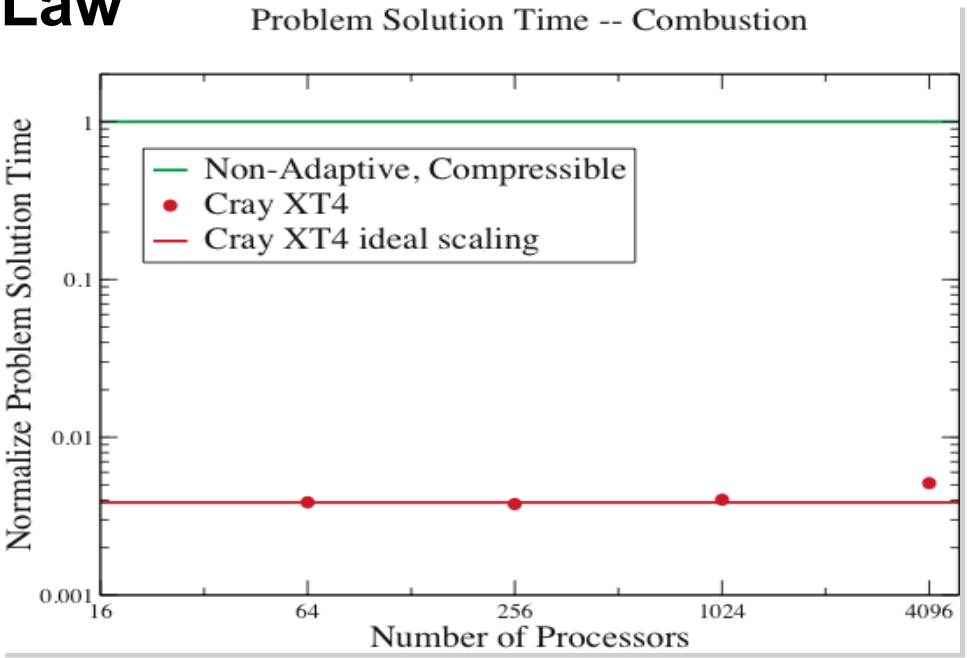


Figure from "SCaLeS report," Volume 2

Choose Scalable Algorithms

- Algorithmic gains in last decade have far outstripped Moore's Law
  - Adaptive meshes rather than uniform
  - Sparse matrices rather than dense
  - Reformulation of problem back to basics
- Two kinds of scalability
  - In problem side (n)
  - In machine size (p)

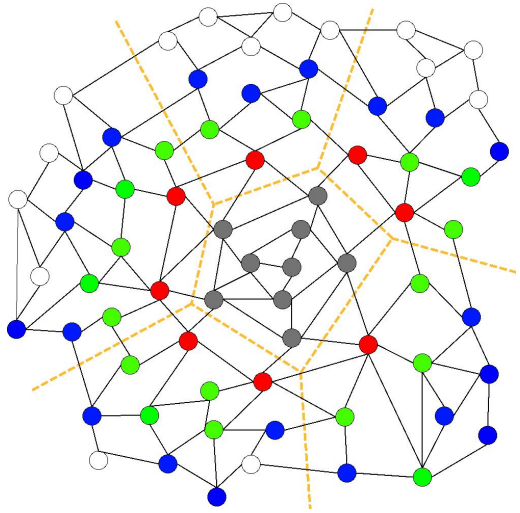


- Example of canonical "Poisson" problem on n points:
  - Dense LU: most general, but $O(n^3)$ flops on $O(n^2)$ data
  - Multigrid: fastest/smallest, O(n) flops on O(n) data

Performance results: John Bell et al

Communication-Avoiding Algorithms

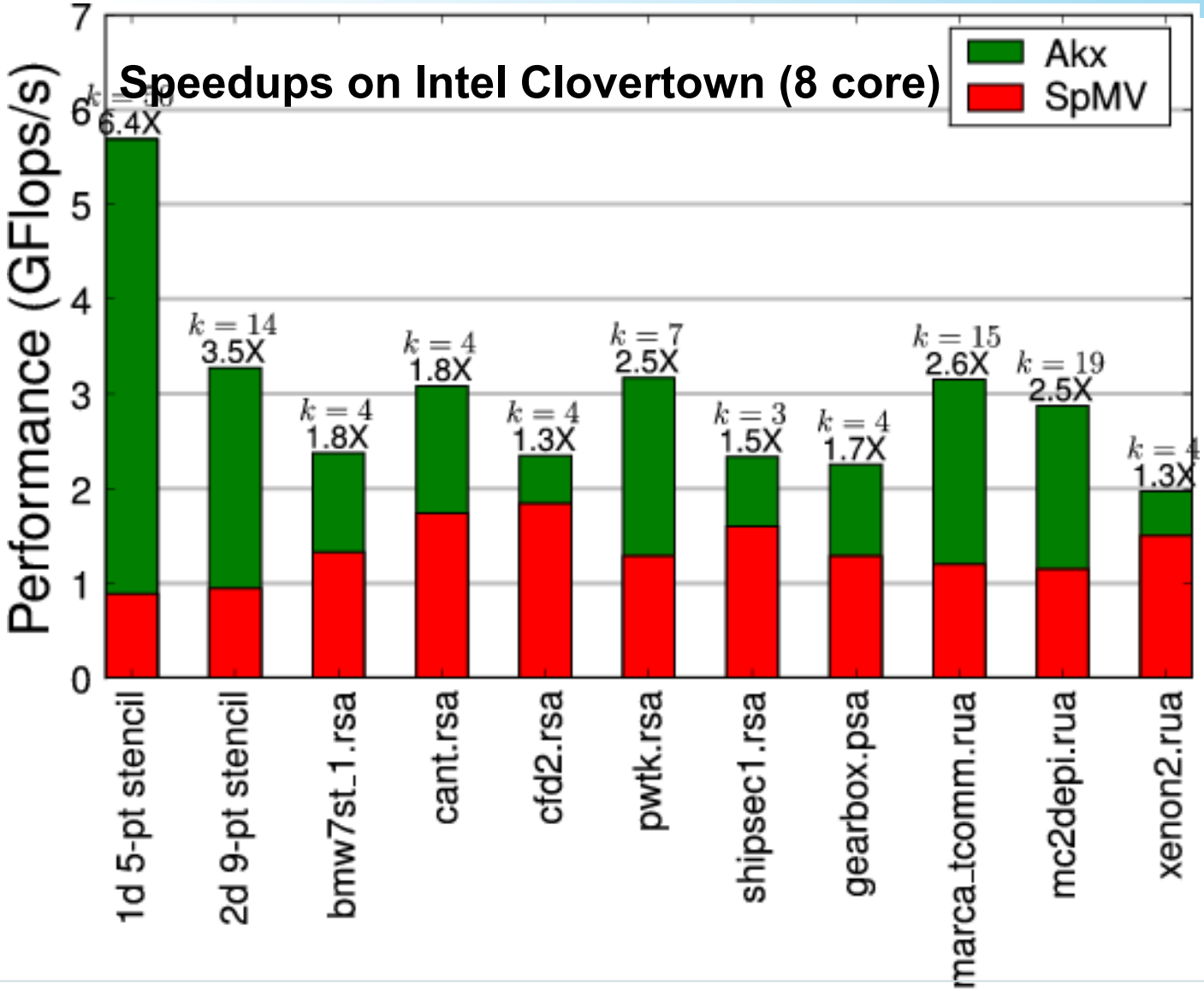
- Sparse Iterative (Krylov Subpace) Methods
  - Nearest neighbor communication on a mesh
  - Dominated by time to read matrix (edges) from DRAM
  - And (small) communication and global synchronization events at each step
- Can we lower data movement costs?
  - Take *k* steps with one matrix read from DRAM and one communication phase
    - Serial: O(1) moves of data moves vs. O(k)
    - Parallel: O(log p) messages vs. O(k log p)



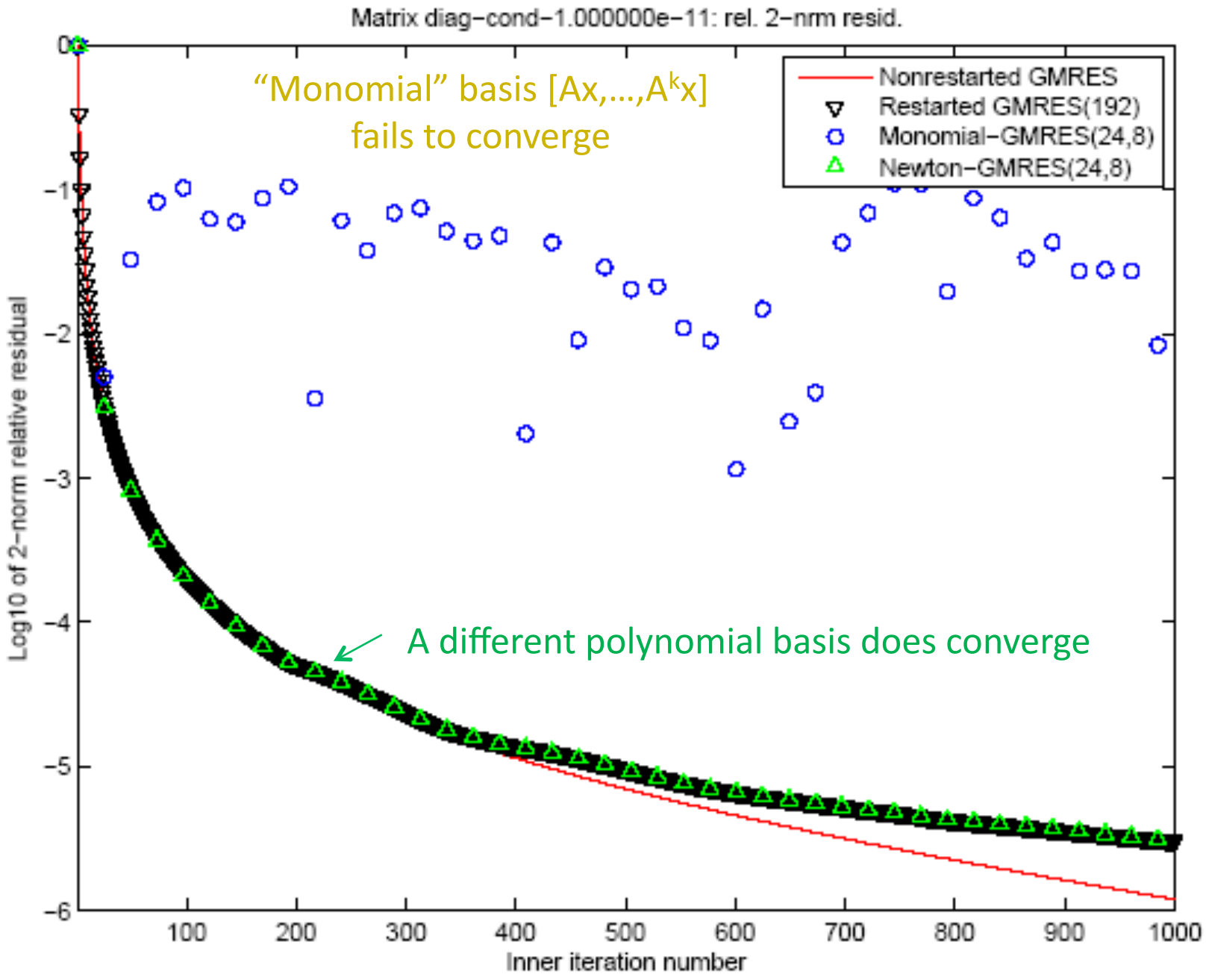
- Can we make communication provably optimal?
  - Communication both to DRAM and between cores
  - Minimize independent accesses ('latency')
  - Minimize data volume ('bandwidth')

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin

Bigger Kernel ($A^k x$) Runs at Faster Speed than Simpler (Ax)

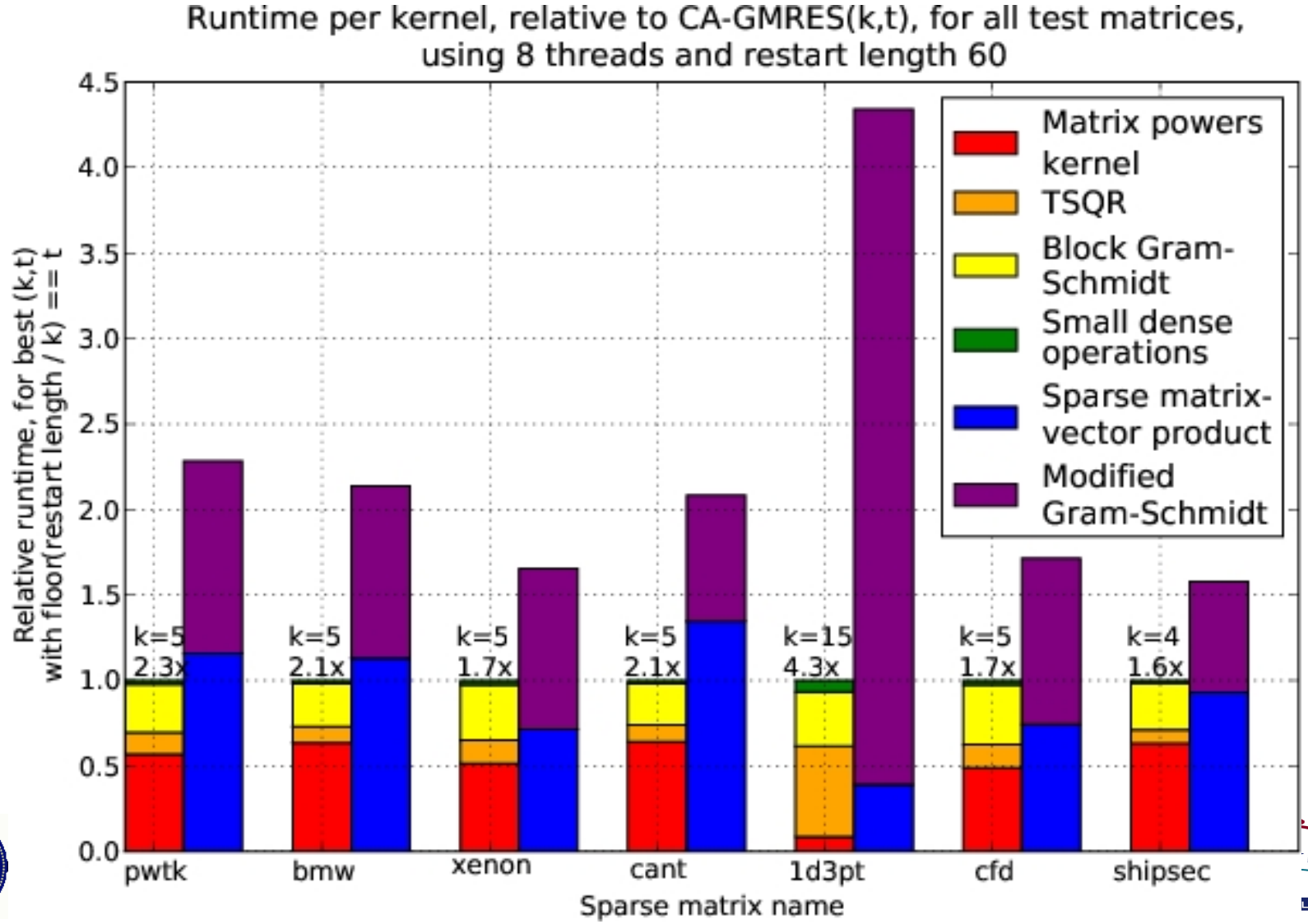


Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Kathy Yelick


Matrix diag−cond−1.000000e−11: rel. 2−nrm resid.
"Monomial" basis [Ax,…,A^k x]
fails to converge
A different polynomial basis does converge
Nonrestarted GMRES
Restarted GMRES(192)
Monomial−GMRES(24,8)
Newton−GMRES(24,8)
Log10 of 2−norm relative residual
Inner iteration number
0
−1
−2
−3
−4
−5
−6
100
200
300
400
500
600
700
800
900
1000

Communication-Avoiding Krylov Method (GMRES)

Performance on 8 core Clovertown



Communication-Avoiding Dense Linear Algebra

- **Well known why BLAS3 beats BLAS1/2: Minimizes communication = data movement**
  - **Attains lower bound Ω (n^3 / cache_size$^{1/2}$) words moved in sequential case; parallel case analogous**
- **Same lower bound applies to *all* linear algebra**
  - **BLAS, LU, Cholesky, QR, eig, svd, compositions…**
  - **Sequential or parallel**
  - **Dense or sparse (n^3 ⇒ #flops in lower bound)**
- **Conventional algs (Sca/LAPACK) do much more**
- **We have new algorithms that meet lower bounds**
  - **Good speed ups in prototypes (including on cloud)**
  - **Lots more algorithms, implementations to develop**

Research by Demmel, Anderson, Ballard, Carson, Dumitriu, Grigori, Hoemmen, Holtz, Keutzer, Knight, Langou, Mohiyuddin, Schwartz, Solomonik, Williams, Xiang,Yelick

Challenges to ~~Exascale~~ Performance Growth

1) **System power** is the primary constraint
2) **Concurrency** (1000x today)
3) **Memory** bandwidth and capacity are not keeping pace
4) **Processor** architecture is open, but likely heterogeneous
5) **Programming model** heroic compilers will not hide this
6) **Algorithms** need to minimize data movement, not flops
7) **I/O bandwidth** unlikely to keep pace with machine speed
8) **Reliability and resiliency** will be critical at this scale
9) **Bisection bandwidth** limited by cost and energy

Unlike the last 20 years most of these (1-7) are equally important across scales, e.g., 1000 1-PF machines

General Lessons

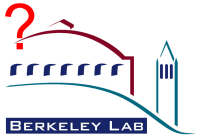
- Early intervention with hardware designs
- Optimize for what is important:

  energy → data movement
- Anticipating and changing the future
  - Influence hardware designs
  - Use languages that reflect abstract machine
  - Write code generators / autotuners
  - Redesign algorithms to avoid communication
- These problems are essential for computing performance in general

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

Are we still at the mercy of hardware with multicore?



Thank You!

Fault Resilience

Chip with FIT rate 1000 fails once every 16 years

A room full of them will fail every few minutes

Industry Trends in Fault Resilience

- Industry must maintain constant FIT rate per node
  - ~10^3 failures in 10^9 hours
- Moore's law gets us 100x improvement
  - Cores are increased, but FITs per chip ~constant
  - But still have to increase node count by 10x
- → 10x worse FIT rate
  - MTTI 1 week to 1 day
  - MTTI 1 day to 1 hour
- Localized checkpointing
  - LLNL SCR to node-local NVRAM
  - More user-assistance in identifying what data to checkpoint

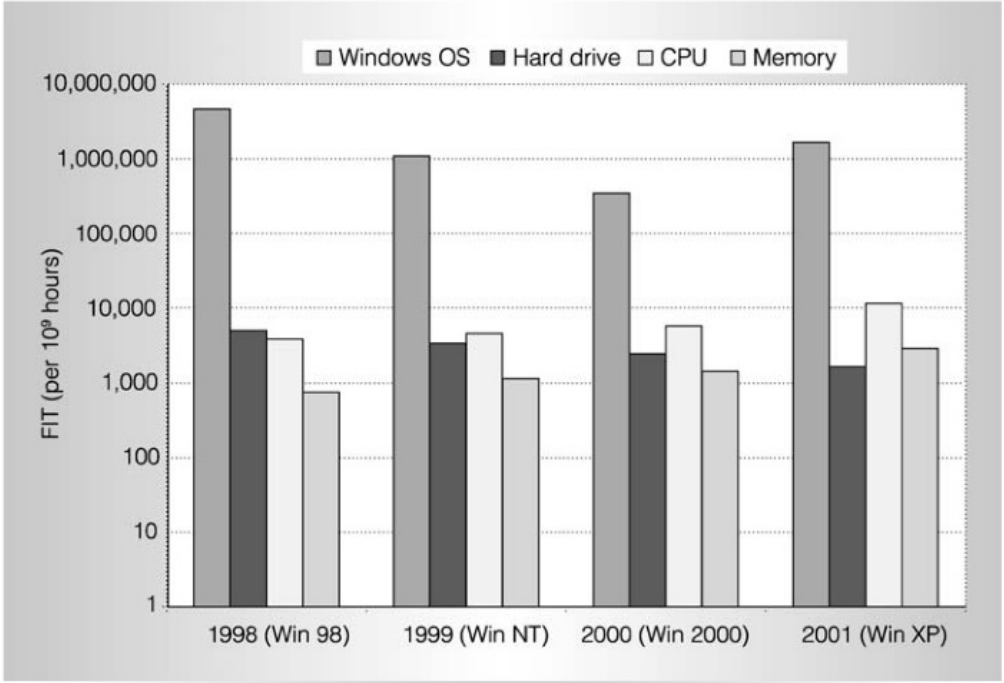


Figure 2. Failures in billions of hours of operation.[2–5]

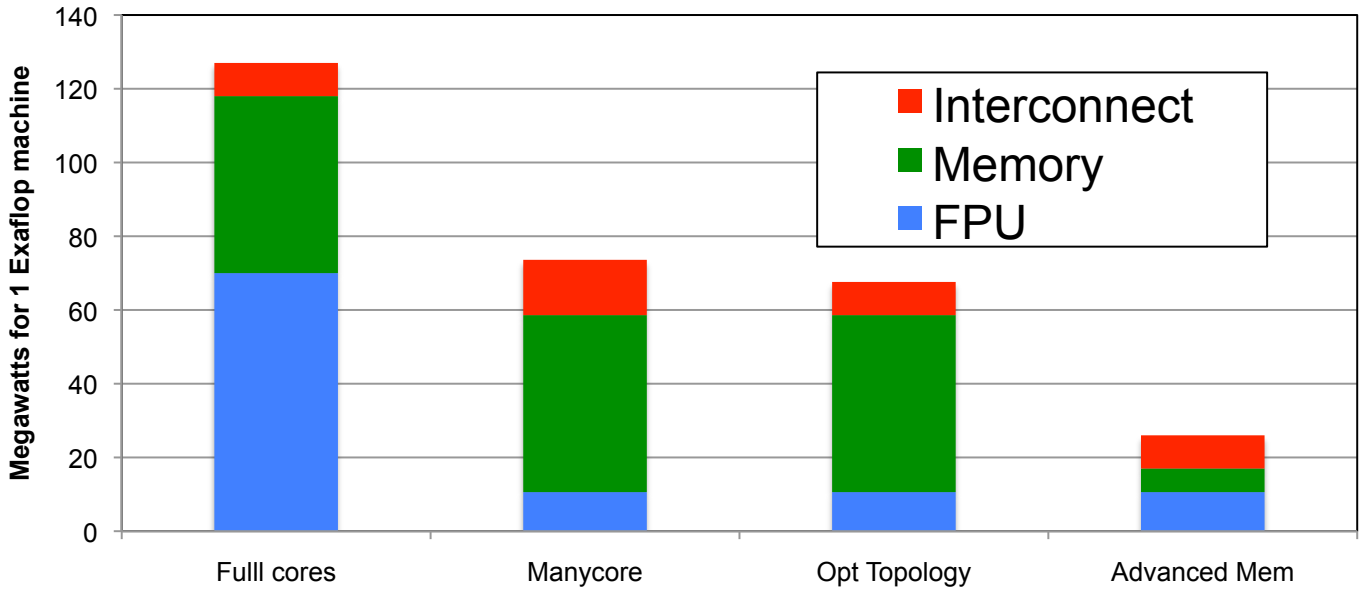
Fault Tolerance/Resilience

- **Hard Errors: *proportional to component count***
  - Spare cores in design (Cisco Metro)
  - SoC design (fewer components and fewer sockets)
  - Use solder (not sockets)
  - Fewer sockets (pushes us to 10TF chip to keep # sockets const.)
- **Soft Errors: *cosmic rays randomly flip bits***
  - Simpler low-power cores expose less surface area
  - ECC for memory and caches
  - On-board NVRAM controller for localized checkpoint
  - Checkpoint to neighbor for rollback (LLNL SCR)
- **Silent errors: Sometimes RAID & ECC are not enough**
  - End-to-End protection schemes (ZFS)
  - Byzantine Fault Tolerance (BFT)

Abstract Machine Model

- Programming models have traditionally reflected the underlying hardware
  - SIMD hardware → Data parallel
  - Shared memory → OpenMP / threads
  - Distributed memory → MPI
- Need a portable abstract machine model for future architectures

Architecture Paths to Exascale



- **Leading Technology Paths *(Swim Lanes)***
  - ~~**Multicore: Replicate traditional cores (x86 and Power7)**~~
  - **Manycore/Embedded: Use many simpler, low power cores from embedded space (BlueGene)**
  - **GPU/Accelerator*:* Use highly specialized processors from gaming space (NVidia Fermi, Cell)**

Exascale Basic Arithmetic

- Exascale Options ("Swim Lanes")
  - 1 GHz proc 10^9
  - 1 K to 10K FPUs / socket (10^3 to 10^4)
  - 1 M sockets / system (10^6 to 10^5)

  = 10^{18}
- Constraints
  - Higher clock speeds the system will melt
  - More sockets the system will never stay up
    - And more money in the interconnect
  - More FPUs / chip: insufficient memory bw
    - 4-8 way SIMD/VLIW? Enough fine-grained parallelism?