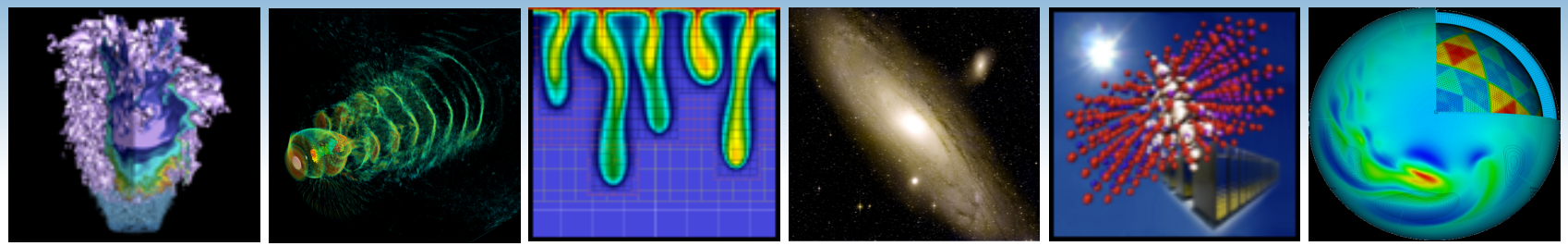


Exascale Computing: More and Moore?

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

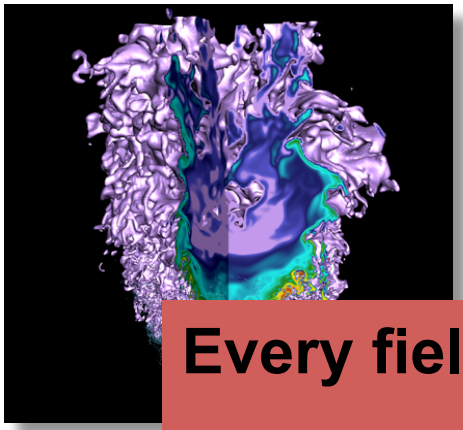
NERSC computing for science

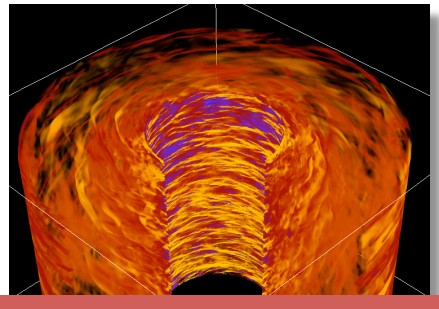
- 4000 users, 500 projects
- 1500 publications per year
- Outstanding user services, computing and data systems

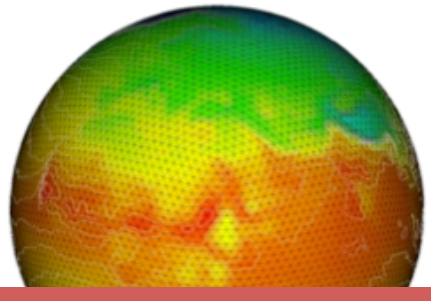
Systems designed for science

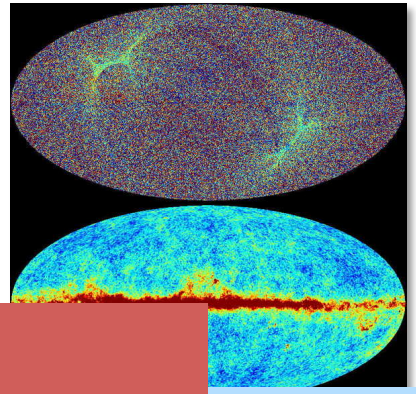
- 1.29 Petaflop Hopper system
- Best application performance per $ and per Watt
- Designed for reliability and productivity

Exascale: Who Needs It?







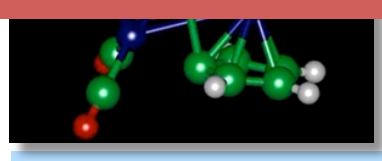


Combu
comple
engine

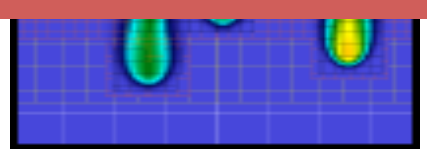
ny: origins
verse

Every field needs more computing!

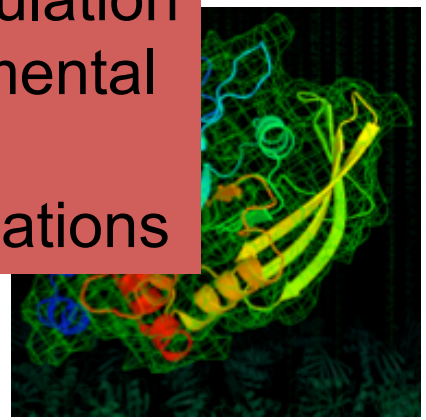
1) To quantify and reduce uncertainty in simulation and to analyze the data sets from experimental devices (increase number of simulations)
2) Analyze data from experiments and simulations



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



Sequestration: Understanding fluid flow & chemistry

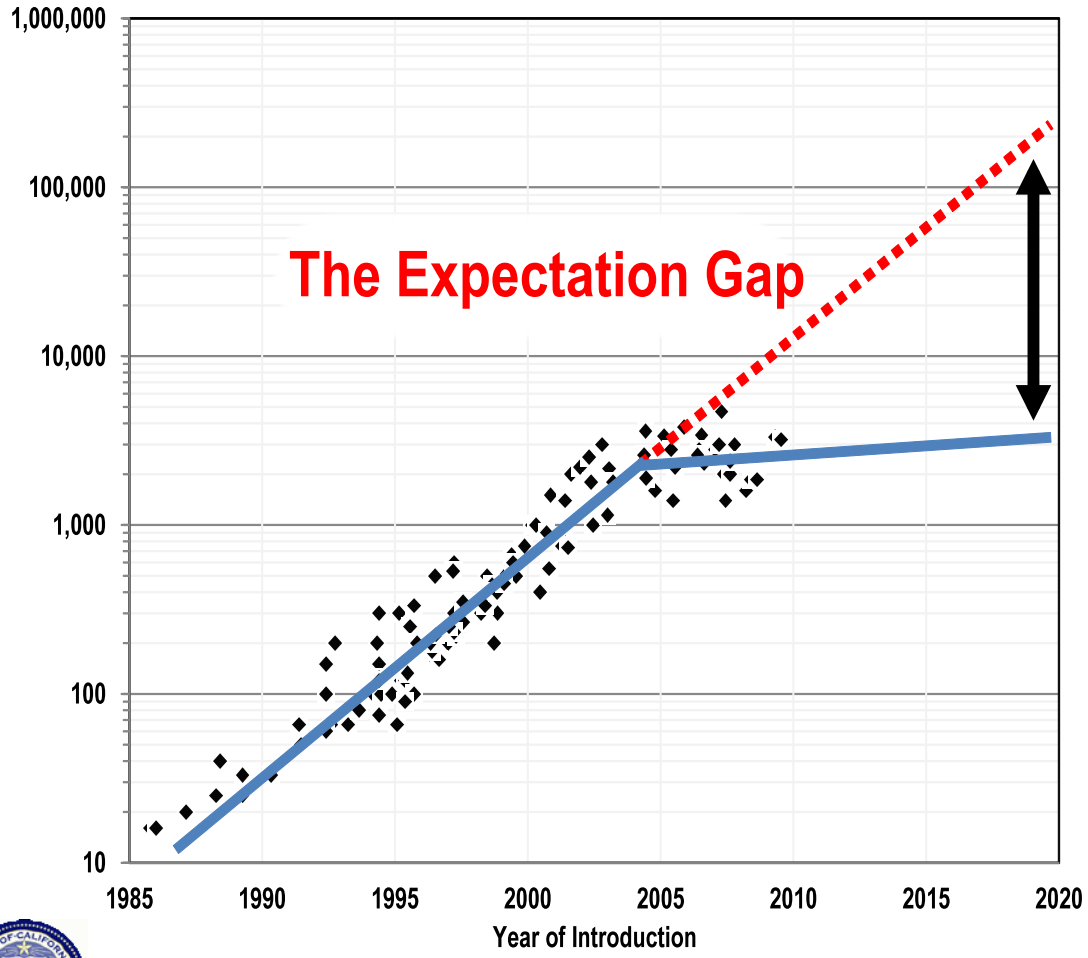


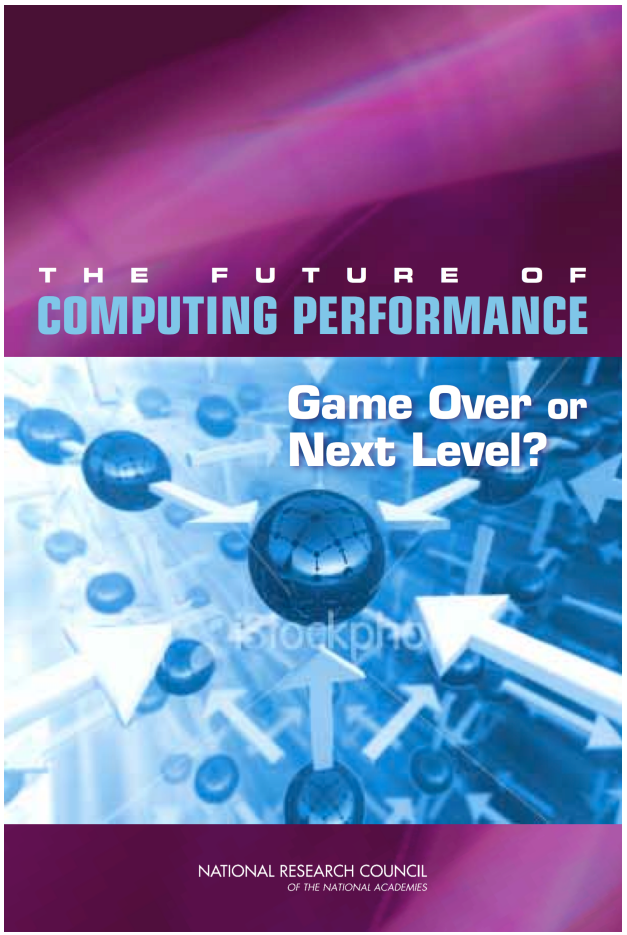
Protein structures: From Biofuels to Alzheimers

BERKELEY LAB

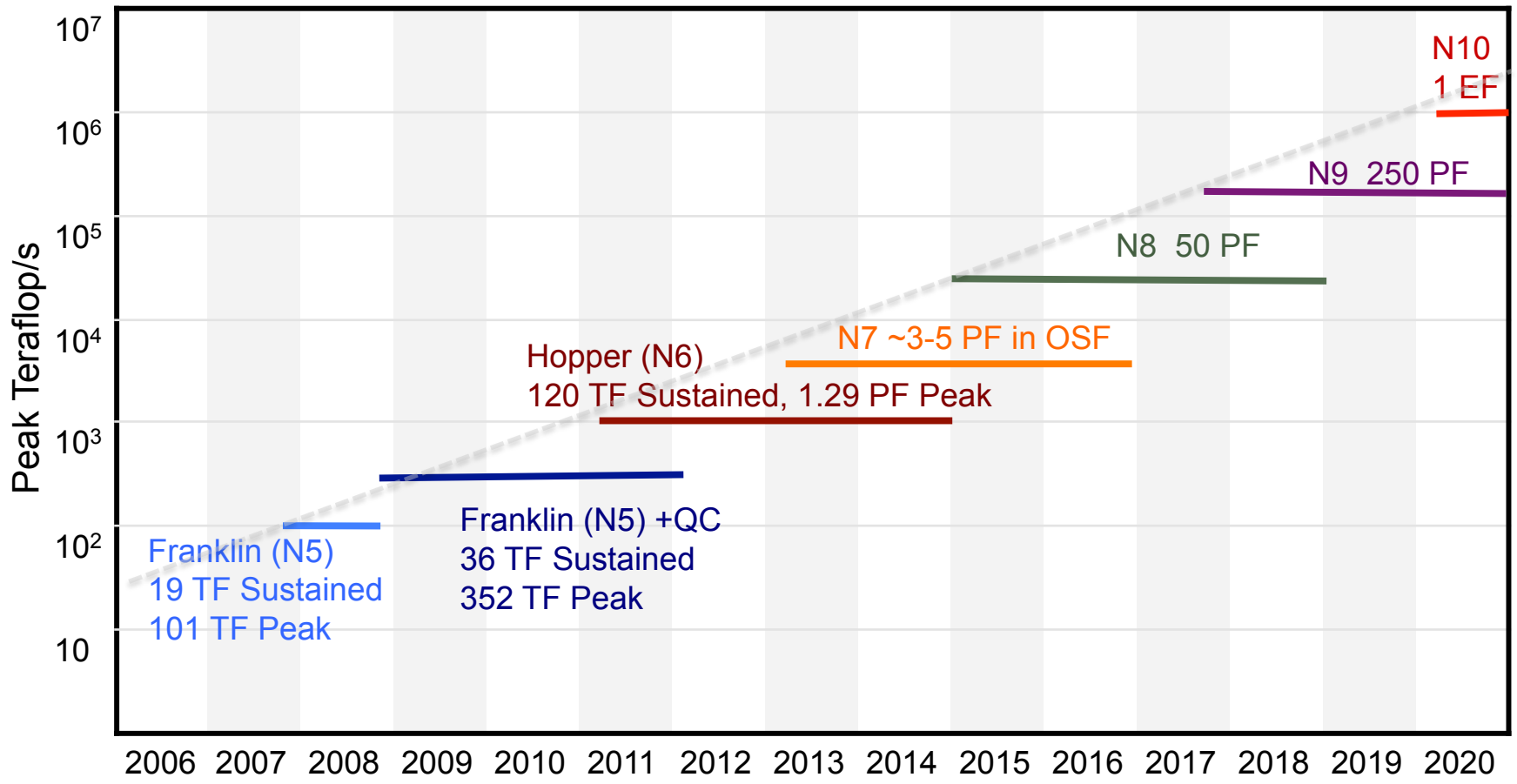
Computing Growth is Not Just an HPC Problem

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



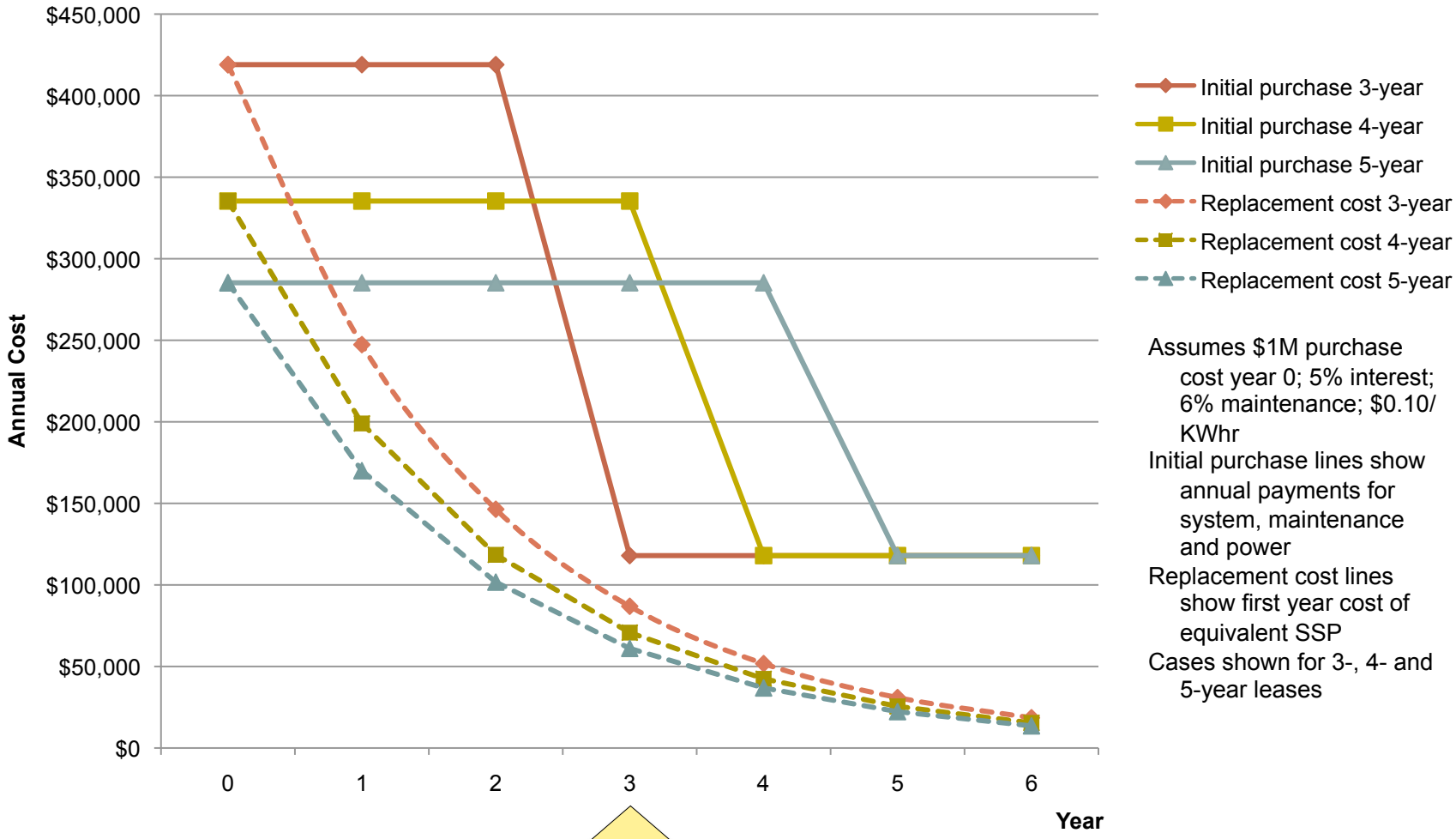


Expectation Leads to Exascale: NERSC Roadmap



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

Life Cycle Economics



Assumes $1M purchase cost year 0; 5% interest; 6% maintenance; $0.10/ KWhr

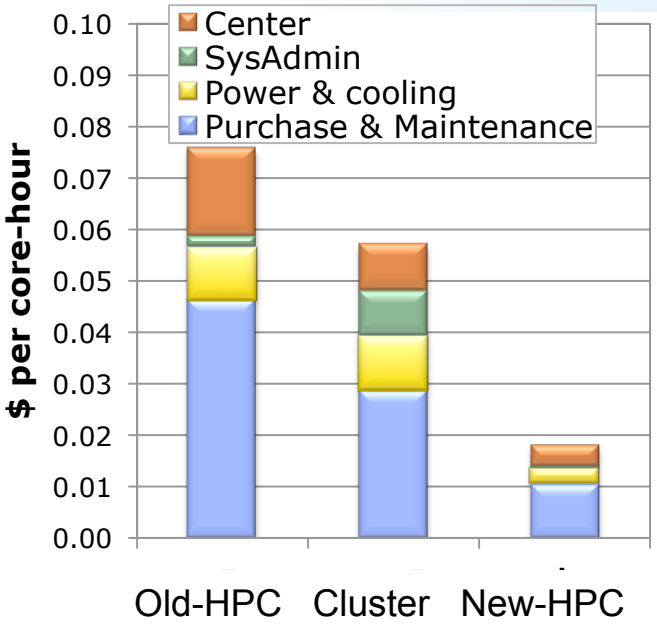
Initial purchase lines show annual payments for system, maintenance and power

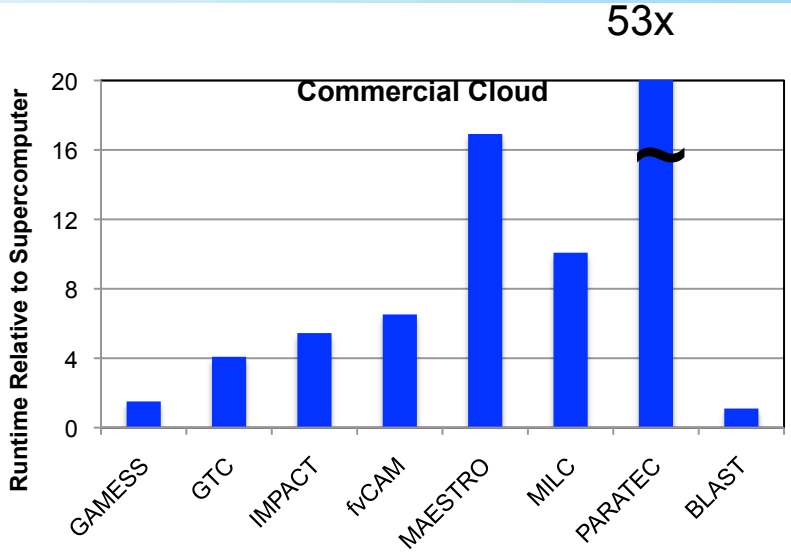
Replacement cost lines show first year cost of equivalent SSP

Cases shown for 3-, 4- and 5-year leases

Analysis by
Jeff Broughton, NERSC

Exascale for Thousands of Users





- **Minimum cost per core (or app flop) are:**
  - Newest machines with largest core count per node (power)
  - Largest machine: amortize personnel costs
- **But commercial clouds are slower & more expensive**
  - Price not dropping with Moore's Law (18% in 5 years)
  - 6-7x cost to buy NERSC compute + storage in 2011 cloud

The Exascale Challenge

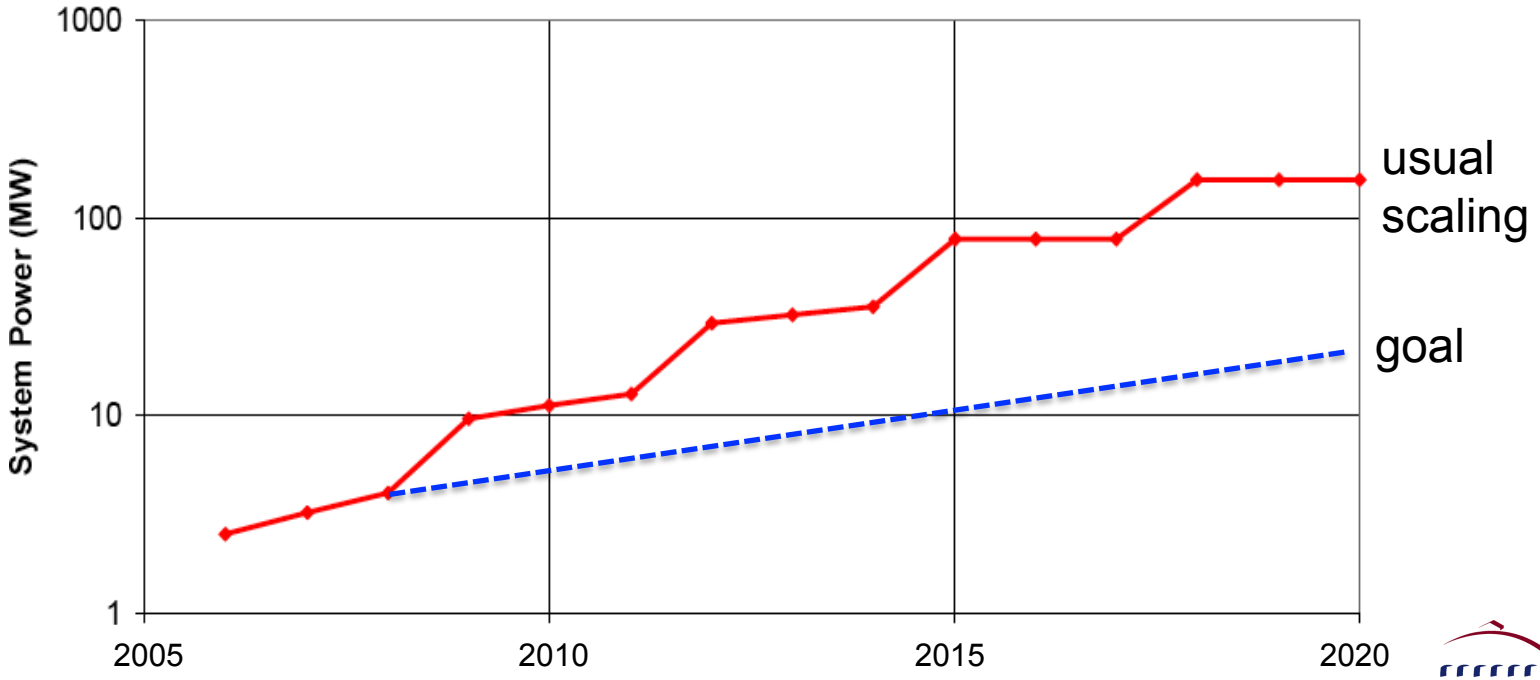
Energy Efficiency

8

Energy Cost Challenge for Computing Facilities

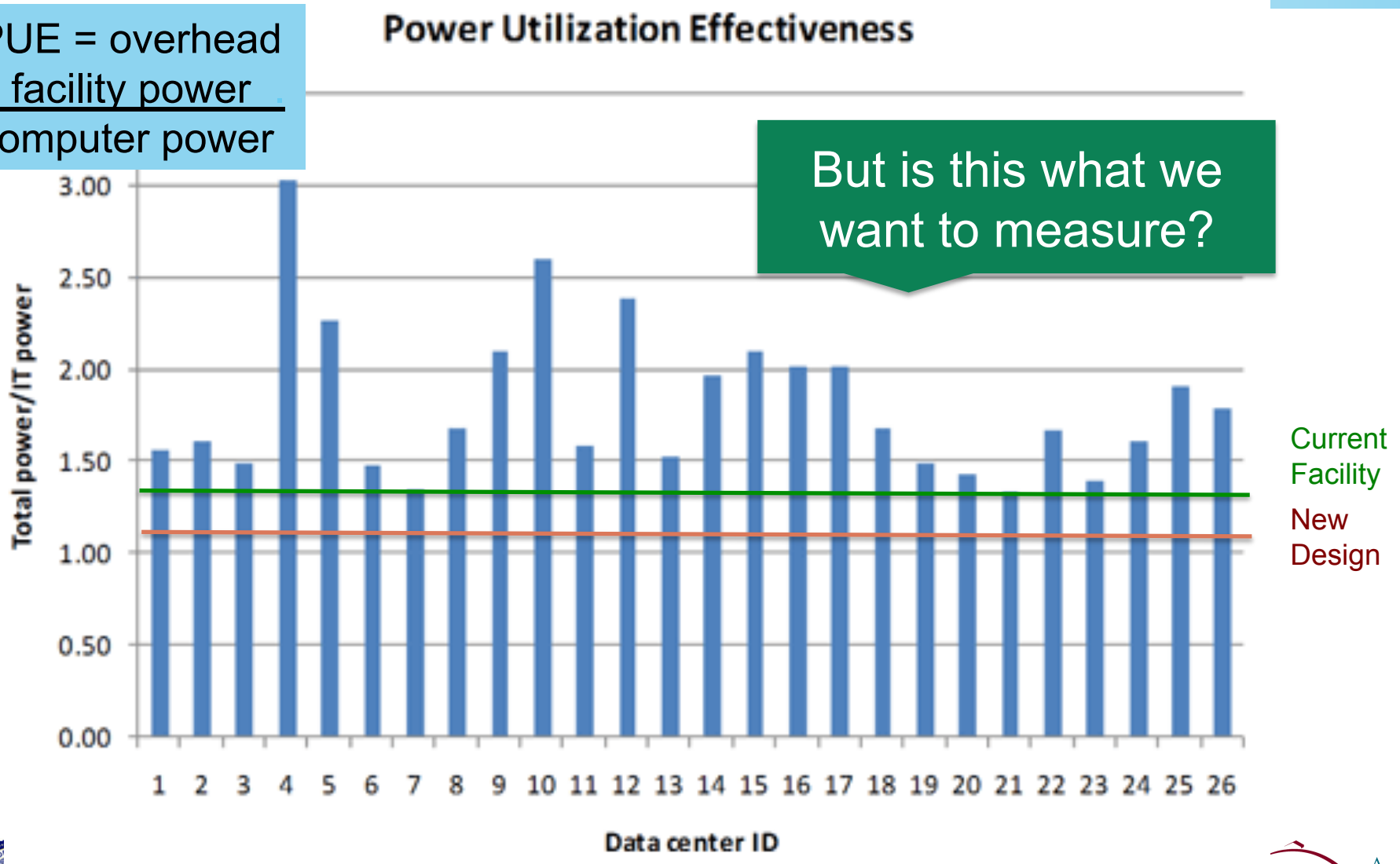
At ~$1M per MW, energy costs are substantial

- 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



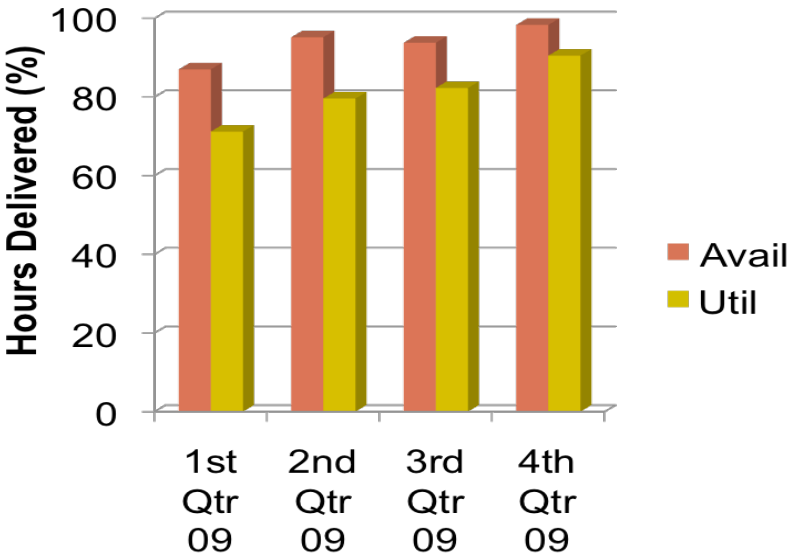
PUE of Data Centers

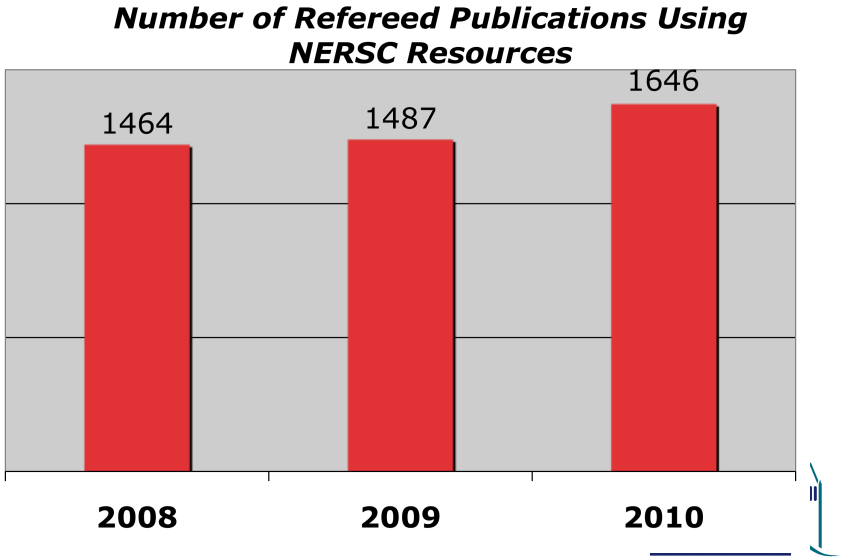
PUE = overhead / computer power
facility power



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





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

Potential Exascale System Architecture Targets

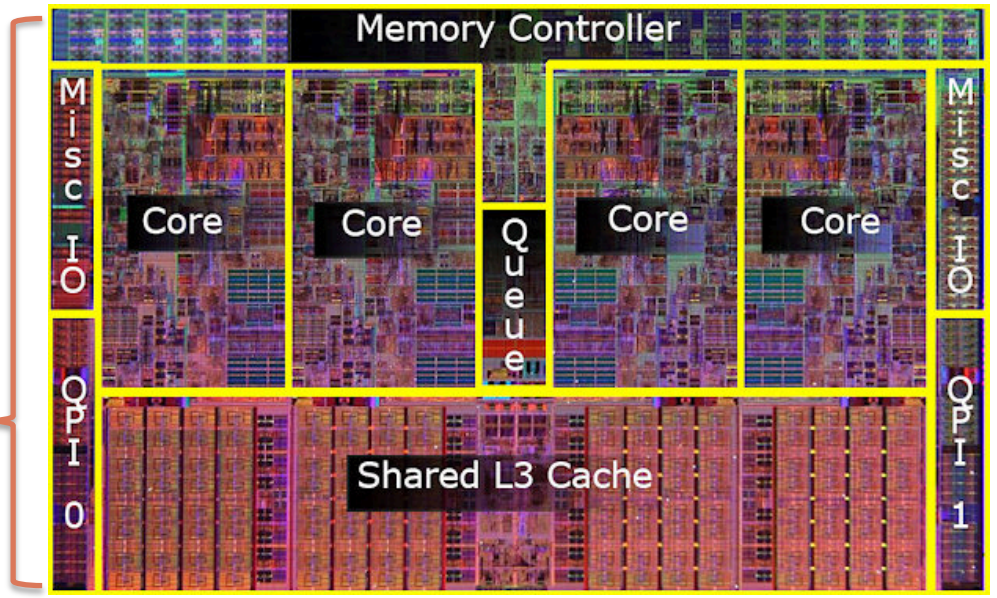
2 “swimlanes”: fast cores (>2 GHz) and slow cores (<.5 GHz) eliminated

System attributes	2010	“2015”		“2018”	
System peak	2 Peta	200 Petaflop/sec		1 Exaflop/sec	
Power	6 MW	15 MW		20 MW	
System memory	0.3 PB	5 PB		32-64 PB	
Node performance	125 GF	0.5 TF	7 TF	1 TF	10 TF
Node memory BW	25 GB/s	0.1 TB/sec	1 TB/sec	0.4 TB/sec	4 TB/sec
Node concurrency	12	O(100)	O(1,000)	O(1,000)	O(10,000)
System size (nodes)	18,700	50,000	5,000	1,000,000	100,000
Total Node Interconnect BW	1.5 GB/s	20 GB/sec		200 GB/sec	
MTTI	days	O(1day)		O(1 day)	

New Processor Designs are Needed to Save Energy

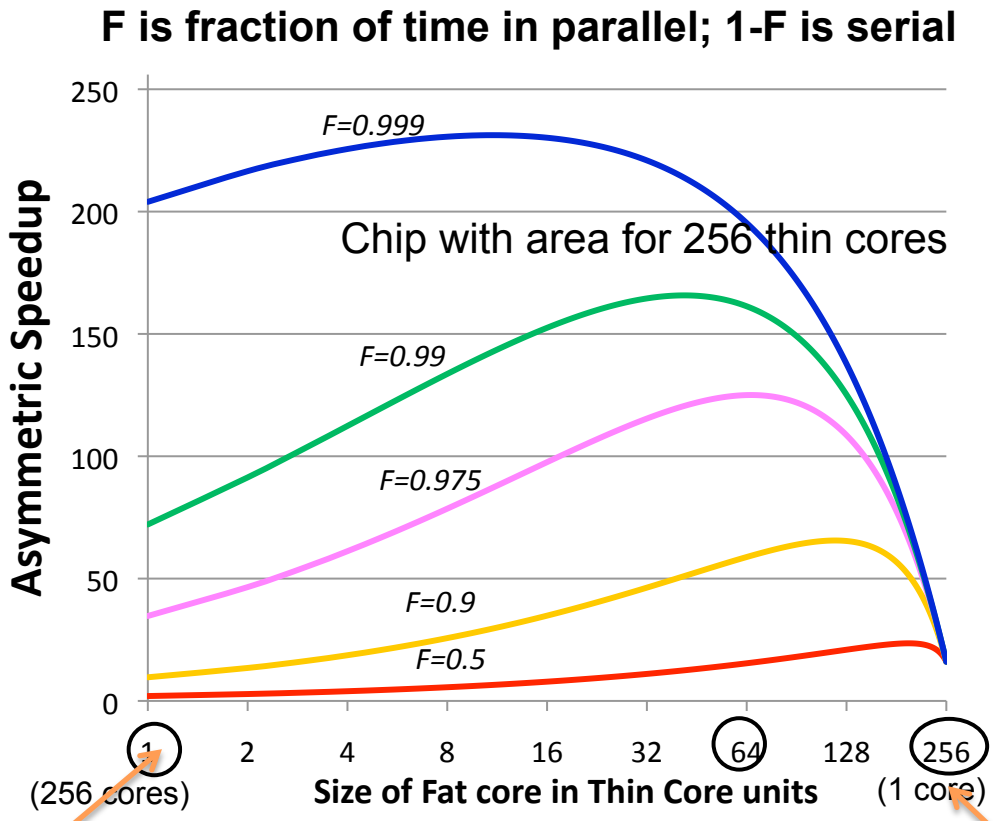
Cell phone processor
(0.1 Watt, 4 Gflop/s)

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



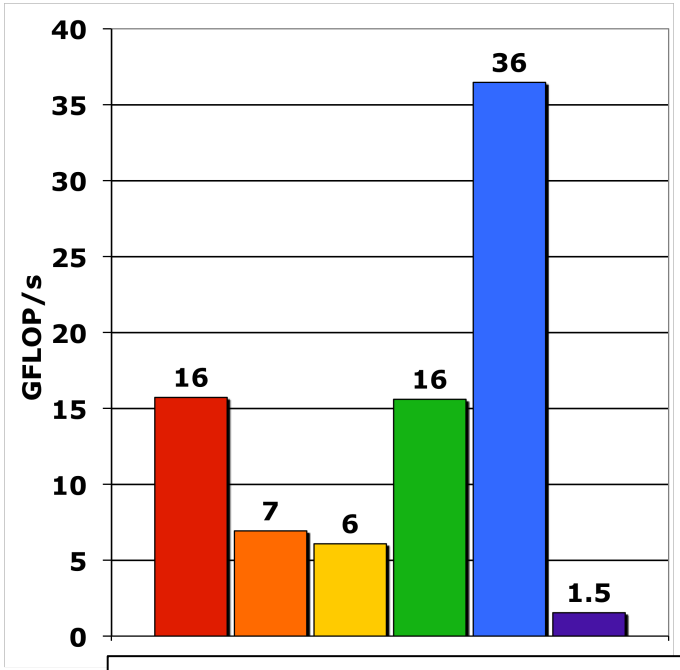
Assumes speedup for Fat / Thin = Sqrt of Area advantage

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

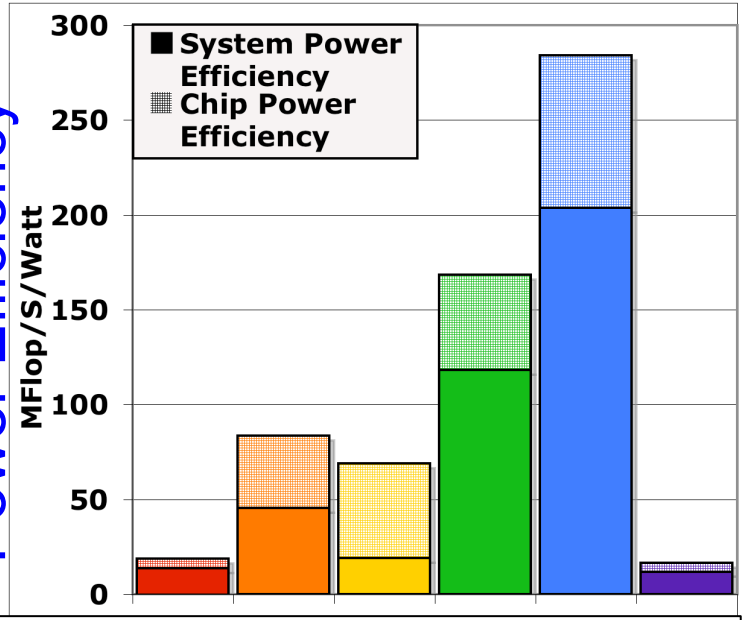
Energy Efficiency of Applications

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

Performance



Power Efficiency



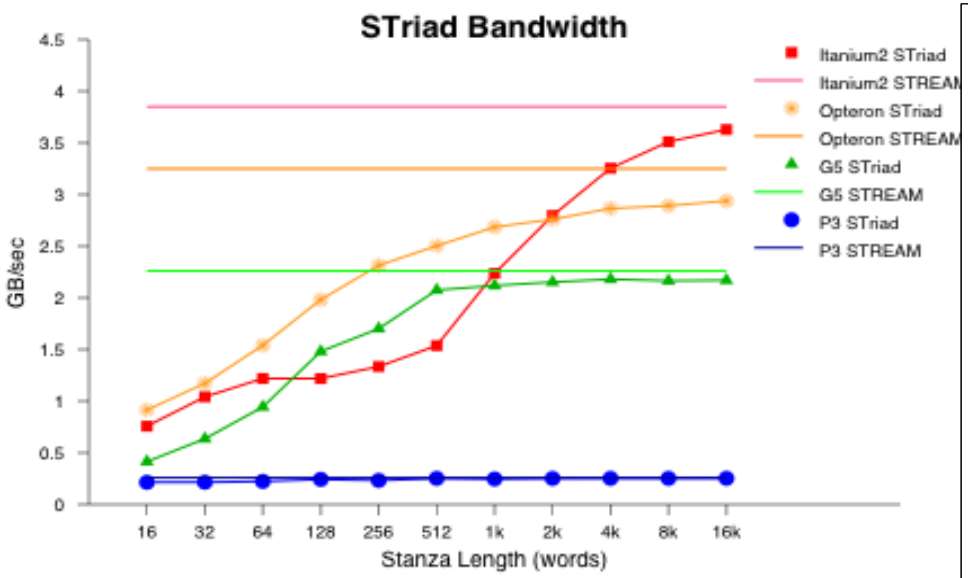
Cache-based

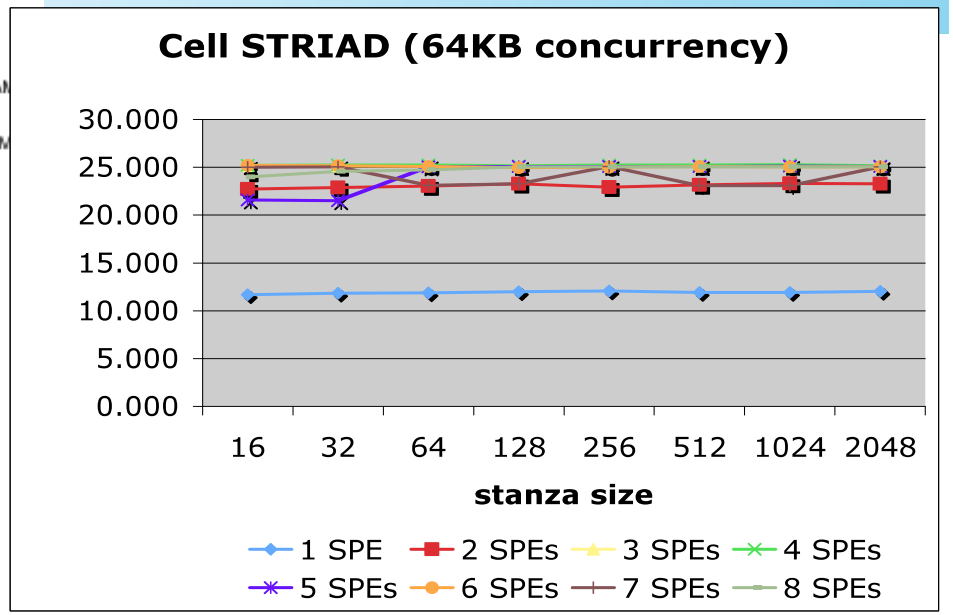
- Gainestown
- Barcelona
- Victoria Falls

Local store-based

- Cell Blade
- GTX280
- GTX280-Host

Value of Local Store Memory



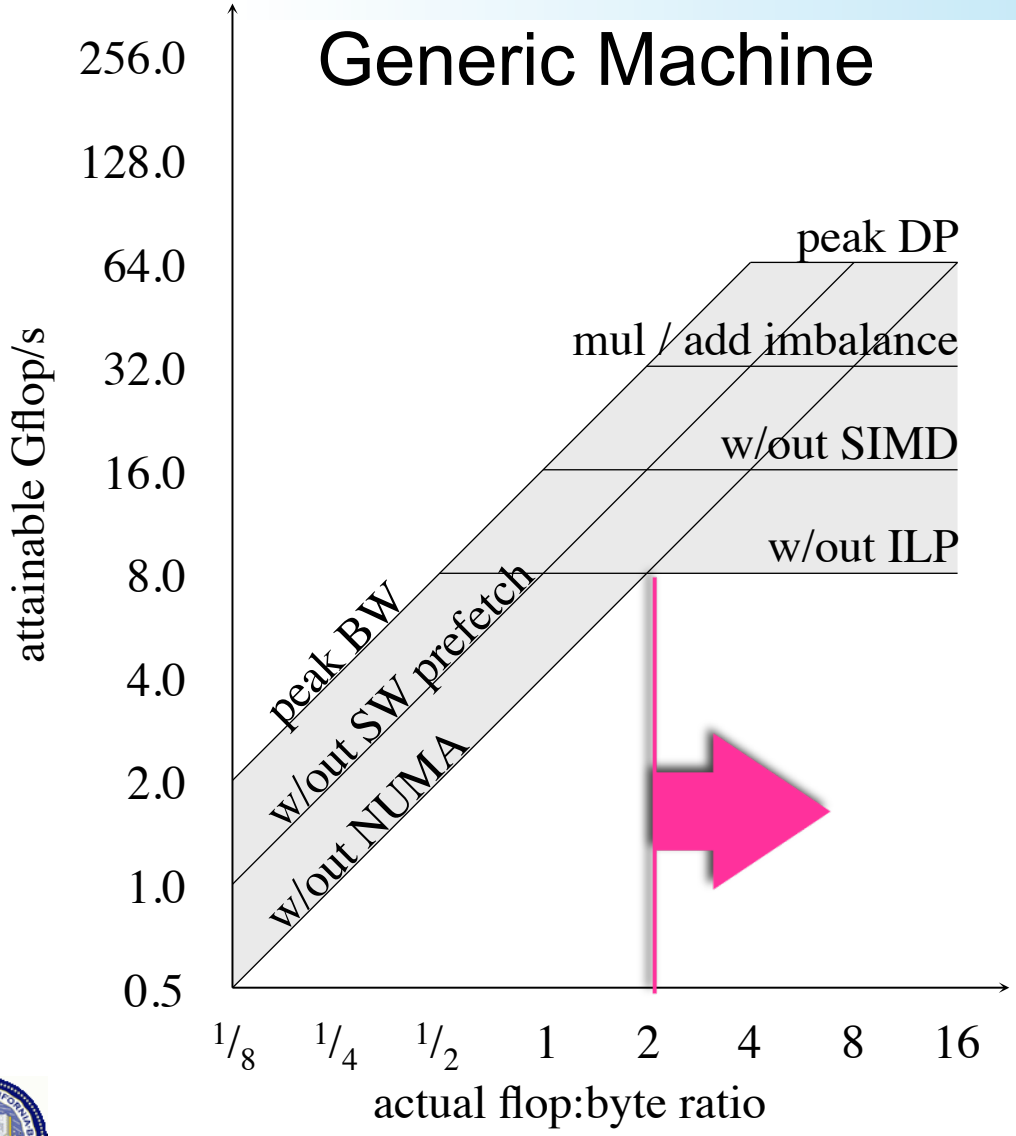


- Unit stride access is as important as cache utilization on processors that rely on hardware prefetch
  - Tiling in unit stride direction is counter-productive: improves reuse, but kills prefetch effectiveness
- Software controlled memory gives programmers more control
  - Spend bandwidth on what you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

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

BERKELEY LAB

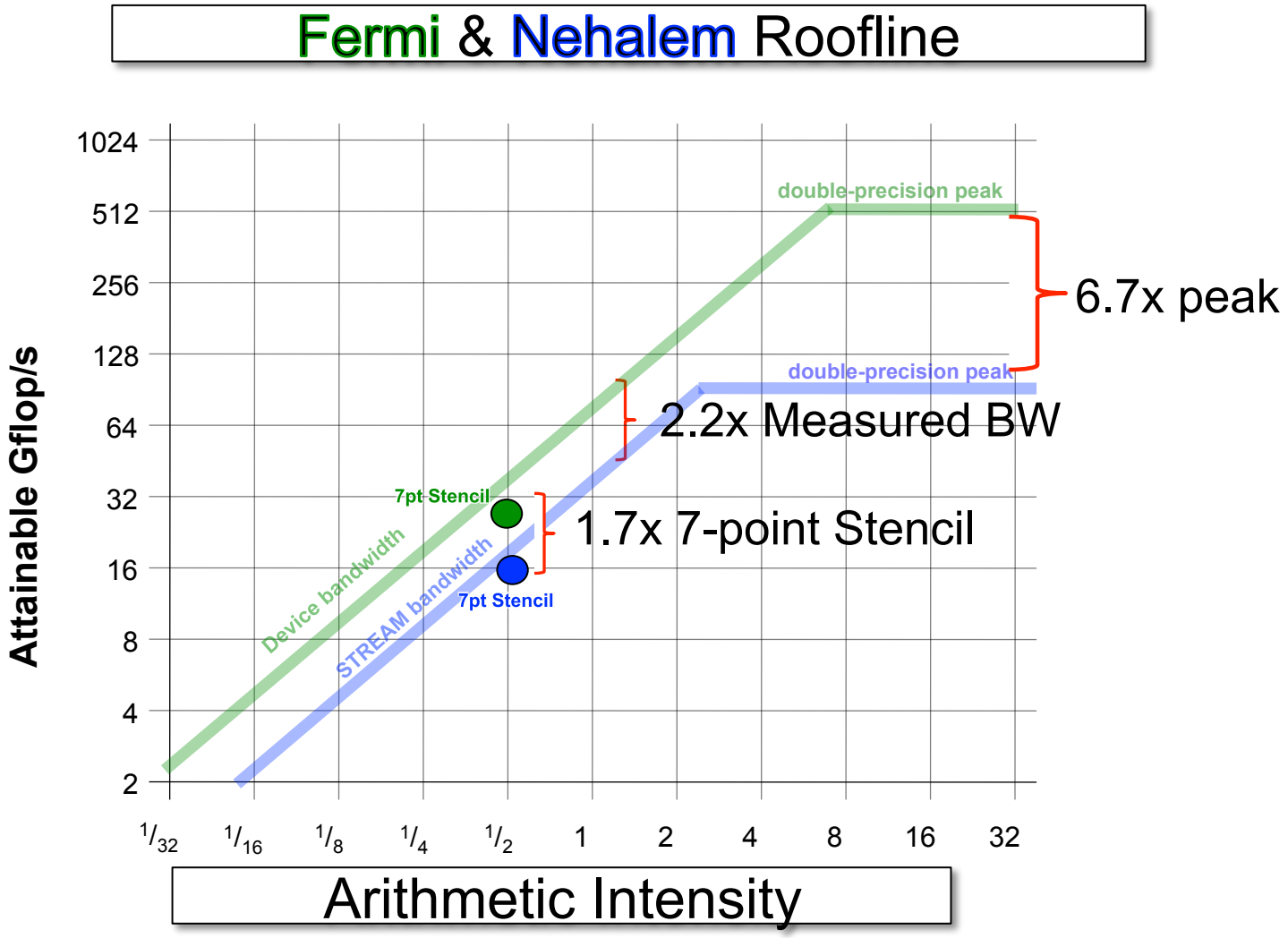
Understanding Node Performance: Roofline Model



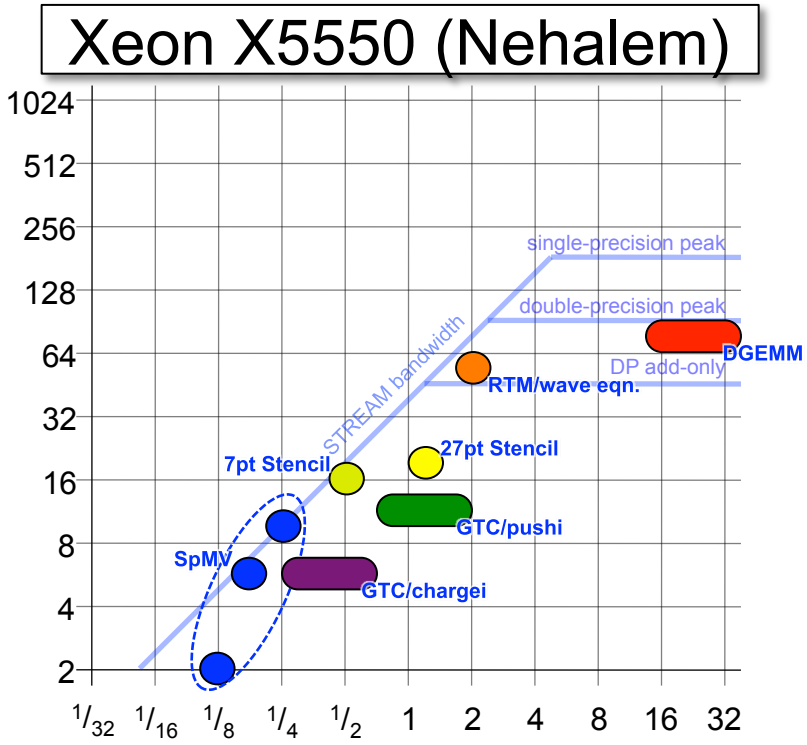
- The flat room is determined by arithmetic peak and instruction mix
- The sloped part of the roof is determined by peak DRAM bandwidth (STREAM)
- X-axis is the computational intensity of your computation

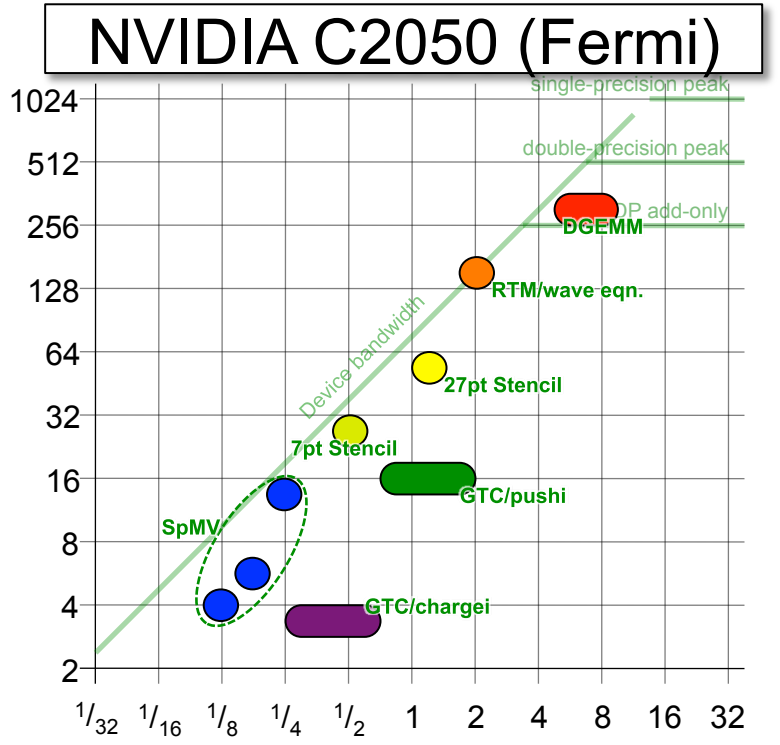
See Sam Williams PhD Thesis & papers

Relative Performance Expectations



Relative Performance Across Kernels



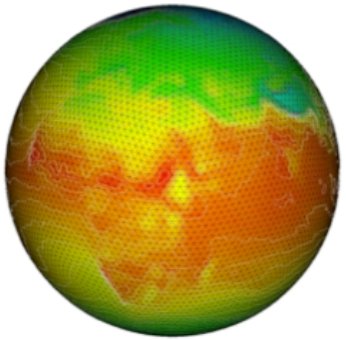


What Heterogeneity Means to Me

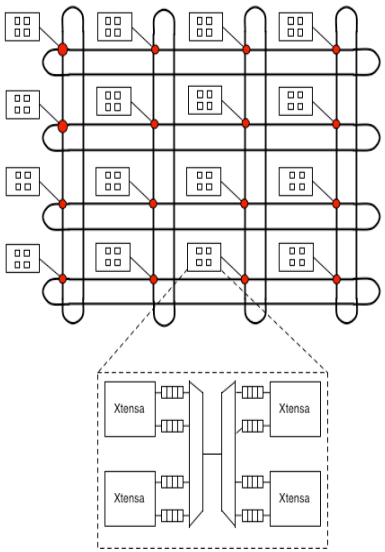
- **Case for heterogeneity**
  - Many small cores are needed for energy efficiency and power density; could have their own PC or use a wide SIMD
  - Need one fat core (at least) for running the OS
- **Local store, explicitly managed memory hierarchy**
  - More efficient (get only what you need) and simpler 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*

Co-Design Hardware & Software

- Green Flash Demo
- CSU atmospheric model ported to low-power core design
  - Dual Core Tensilica processors running atmospheric model at 25MHz
  - MPI Routines ported to custom Tensilica Interconnect
- Memory and processor Stats available for performance analysis
- Emulation performance advantage
  - 250x Speedup over merely function software simulator
- ***Actual code running - not representative benchmark***



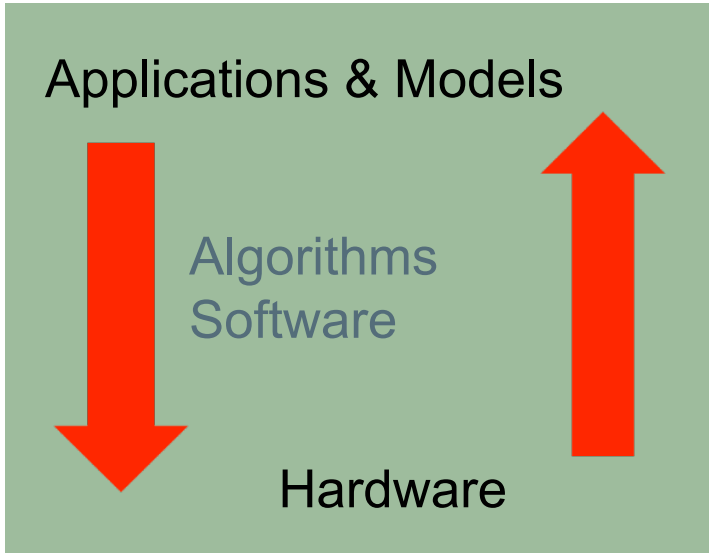
Icosahedral mesh for algorithm scaling

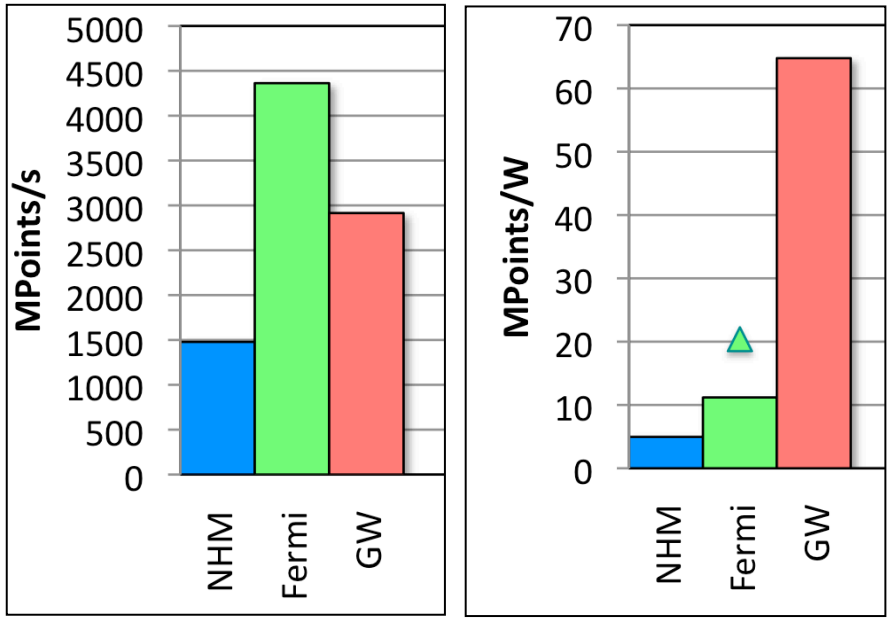


CoDEx: Co-Design for Exascale

Co-design for science applications (kernels or full)

- "GreenWave" example
  - Core (XTensa in-order core)
  - Cache hierarchy
  - Network On Chip (NoC)
  - Interconnect





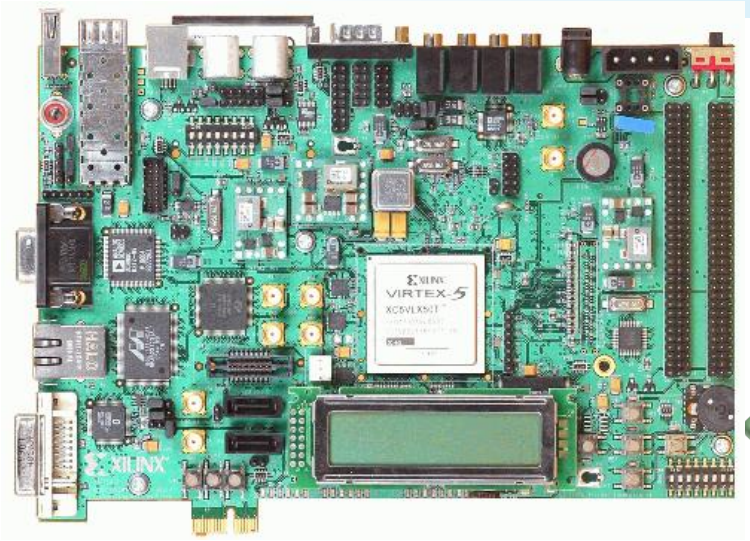
Seismic Time Migration algorithm (RTM) shown

Design hardware to match science needs and algorithms to match hardware

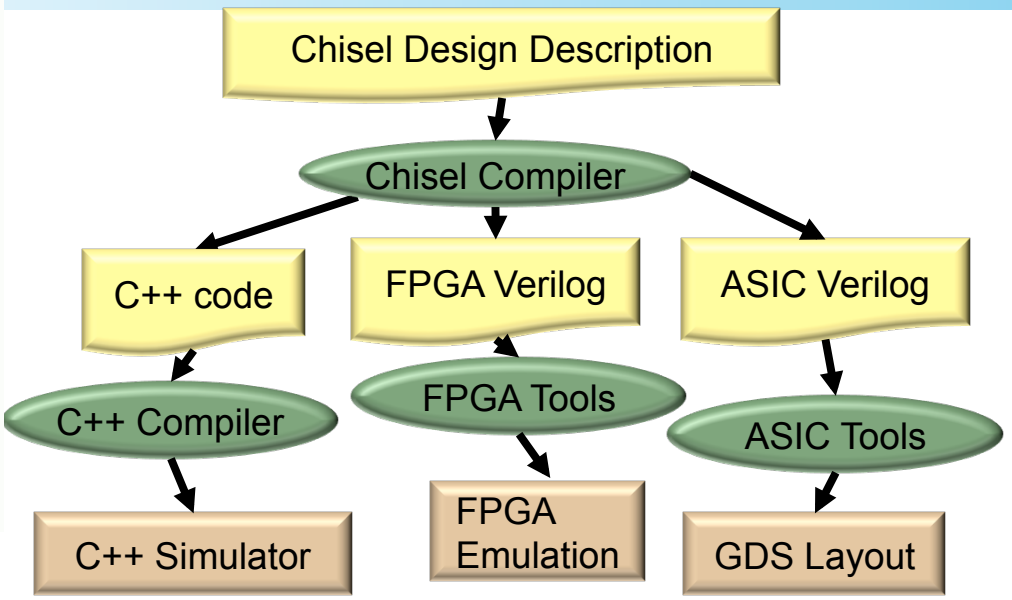
7 Co-Design Centers identified in DOE program

PI: John Shalf, LBNL

RAMP: Enabling Manycore Architecture Research



ISIS builds on Berkeley RAMP project. Ramp Gold shown here which models 64 cores of SPARC v8 with shared memory on $750 board. Has hardware FPU, MMU; boots OS.



ISIS Hardware description language based on Scala, modern OO/Functional language that compiles to JVM.

- **ISIS: rapid, accurate FPGA emulation of manycore chips**
- **Spans VLSI design and simulation and includes chip fab**
  - **Trains students in real design trade-offs, power and area costs**
- **Mapping RTL to FPGAs for algorithm/software co-design**
  - **100x faster than software simulators and more accurate**

PIs: John Wawrzynek and Krste Asanovic, UC Berkeley

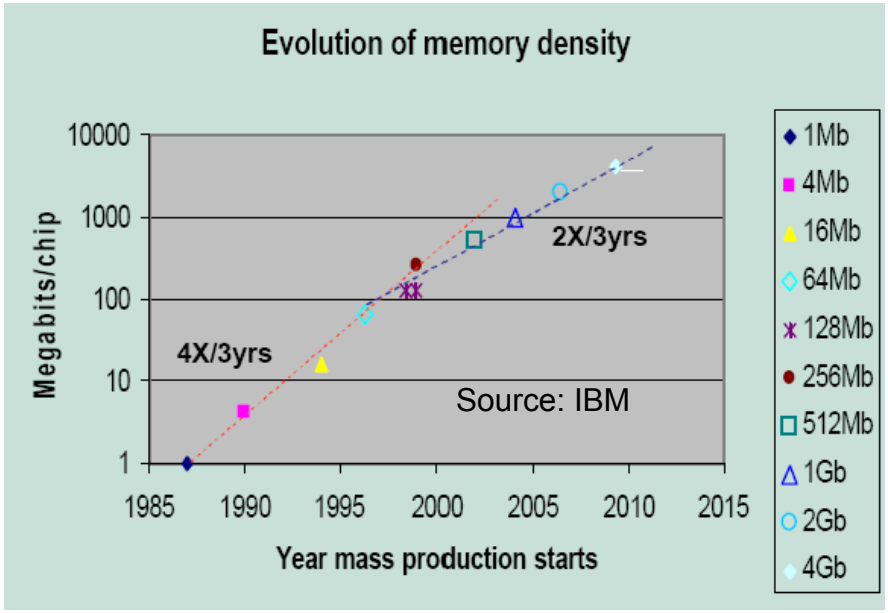
The Future of Software Design and Programming Models

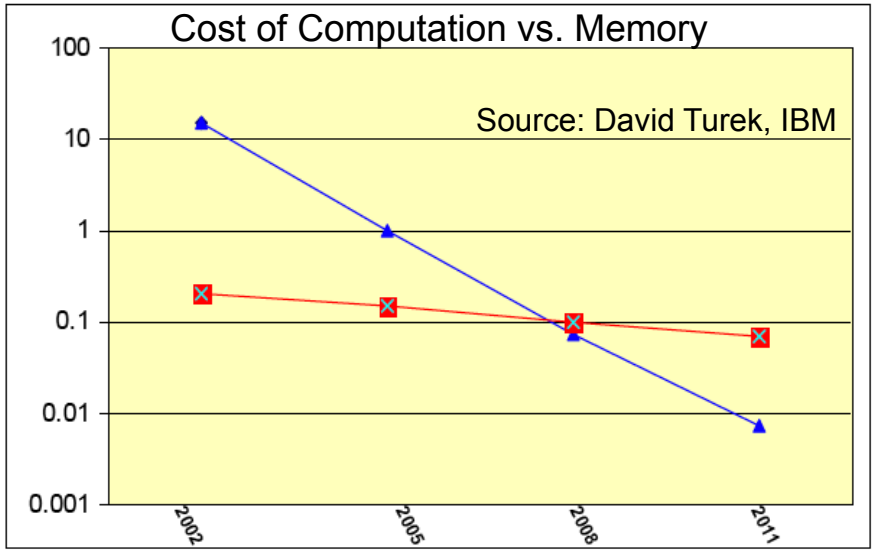
- Memory model
- Control model
- Resilience

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

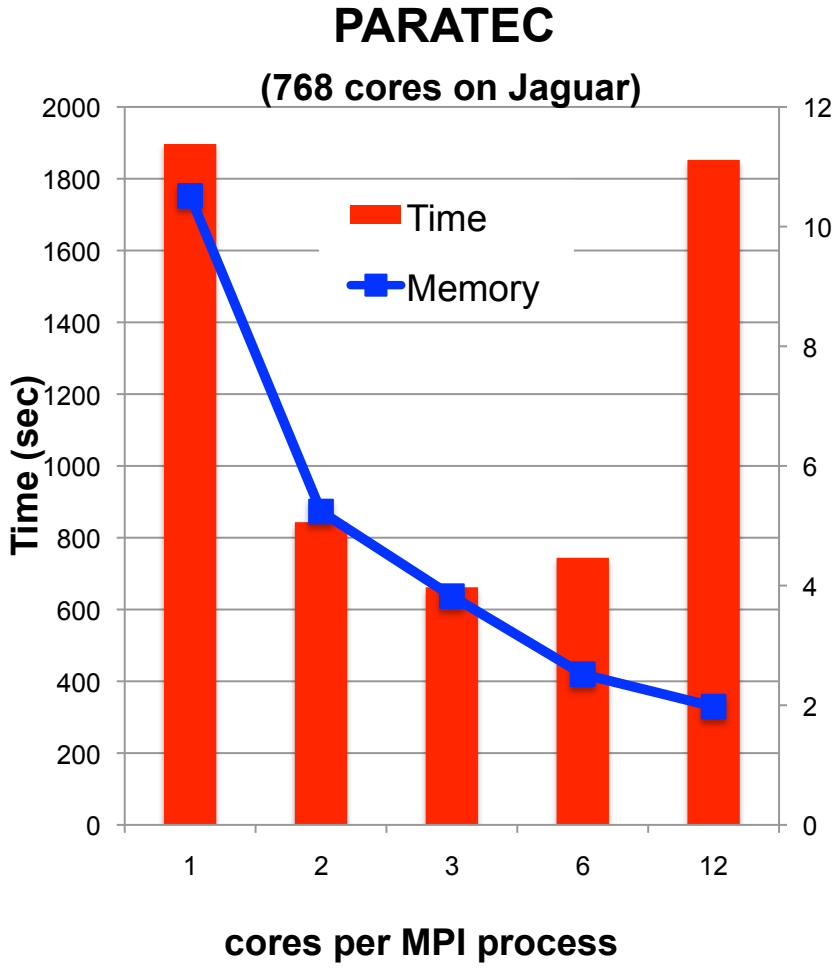
Question: *Can you double concurrency without doubling memory?*

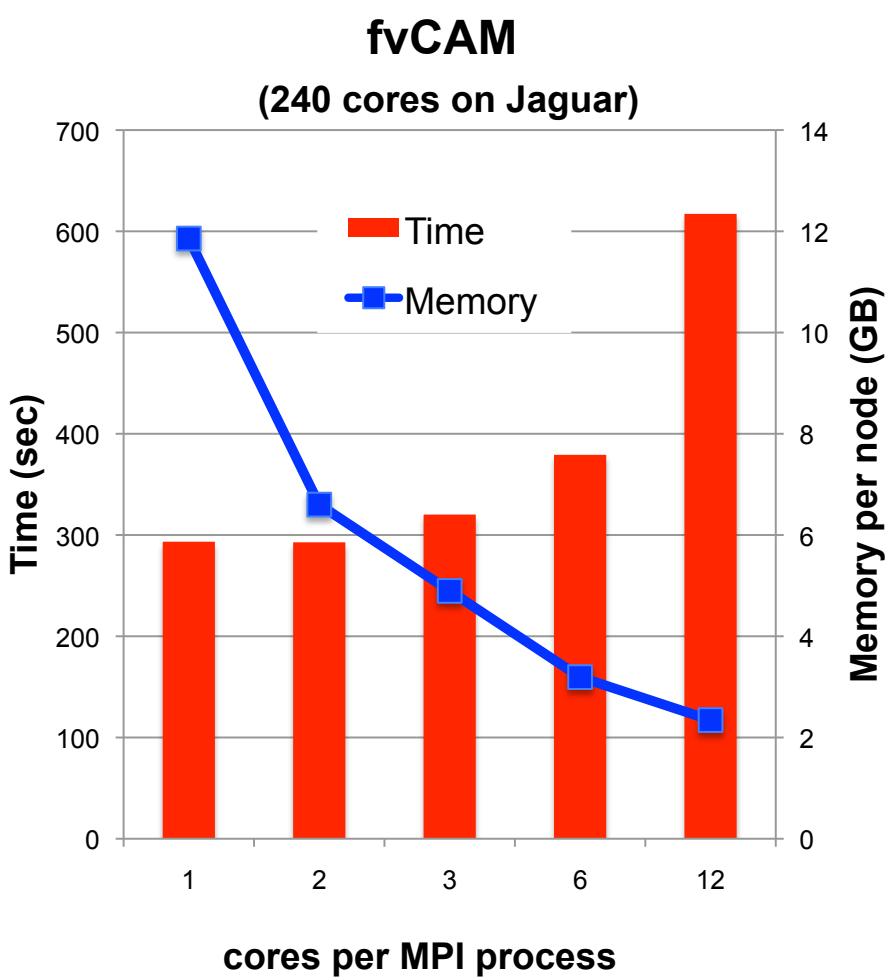
27

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? (circa 2008)
  - 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
- This means a "new" model for most NERSC users

Develop Best Practices in Multicore Programming



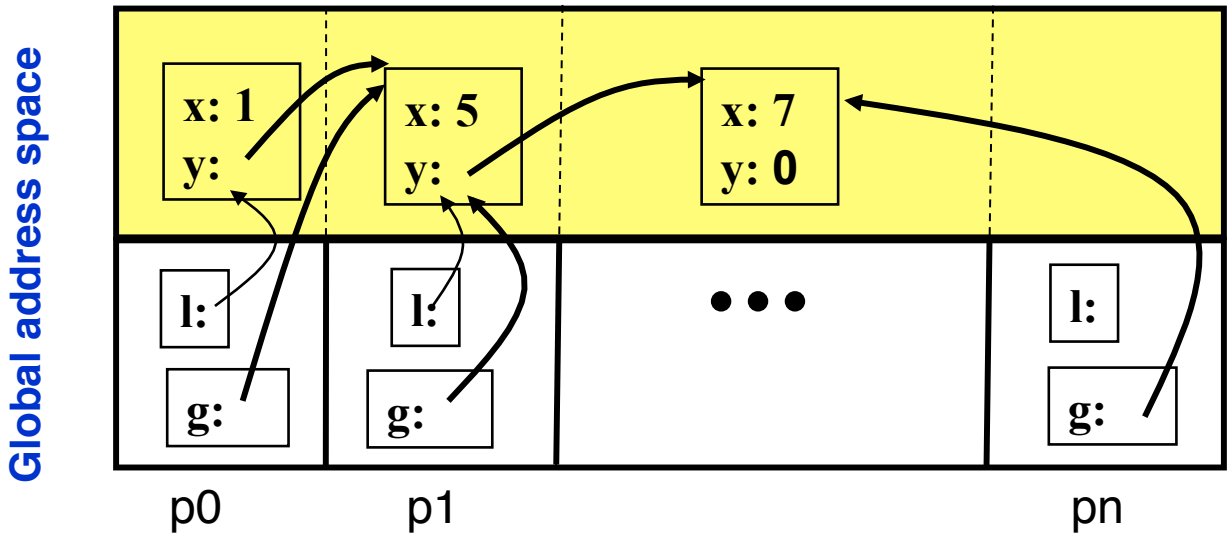


Hybrid Programming is key to saving memory (2011) and sometimes improves performance

Why Use 2 Programming Models When 1 Will Do?

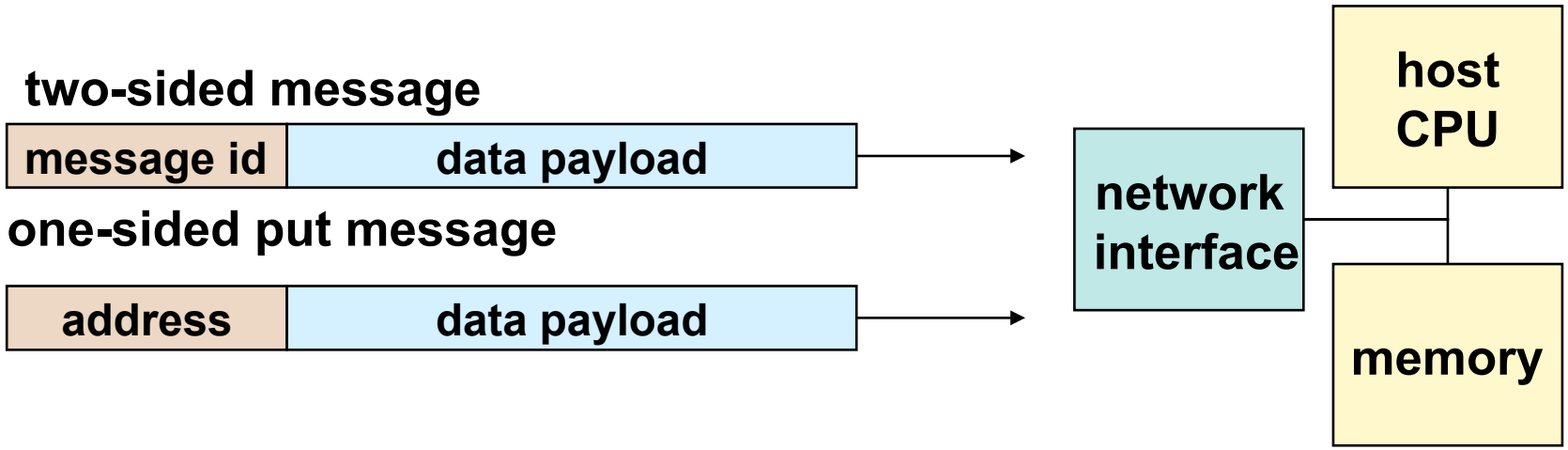
Global address space: thread may directly read/write remote data

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”

Avoiding Synchronization in Communication



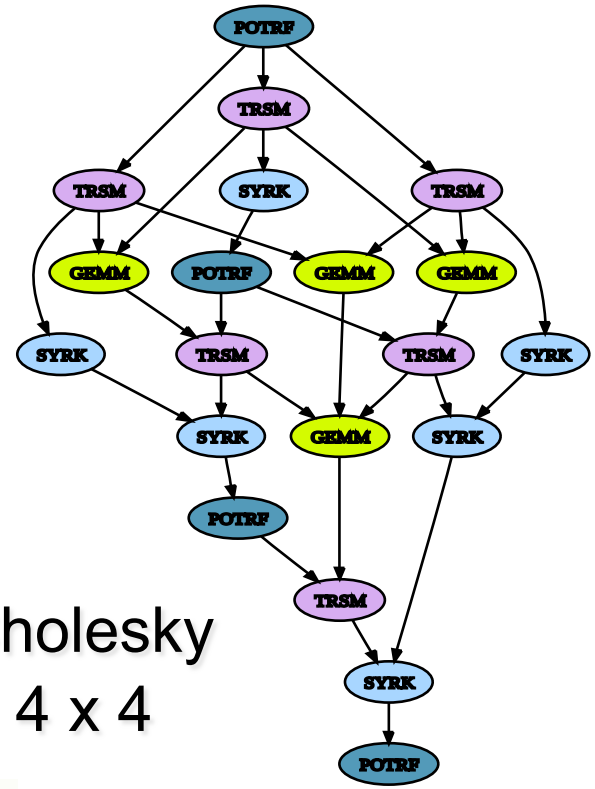
- Two-sided message passing (e.g., MPI) requires matching a send with a receive to identify memory address to put data
  - Wildly popular in HPC, but cumbersome in some applications
  - Couples data transfer with synchronization
- Using global address space decouples synchronization
  - Pay for what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

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

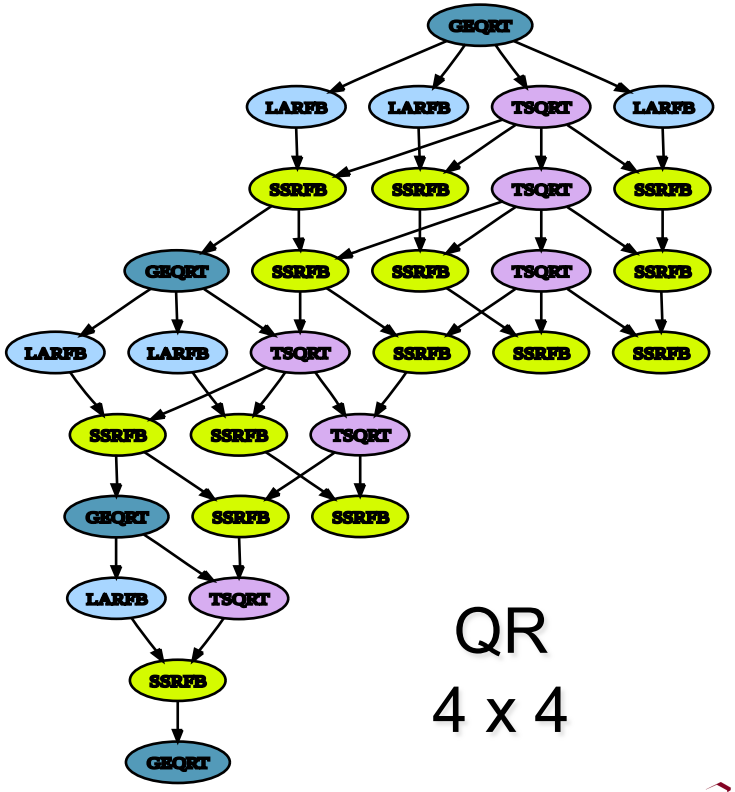
Avoid Synchronization from Applications

Computations as DAGs

View parallel executions as the directed acyclic graph of the computation



Cholesky
4 x 4

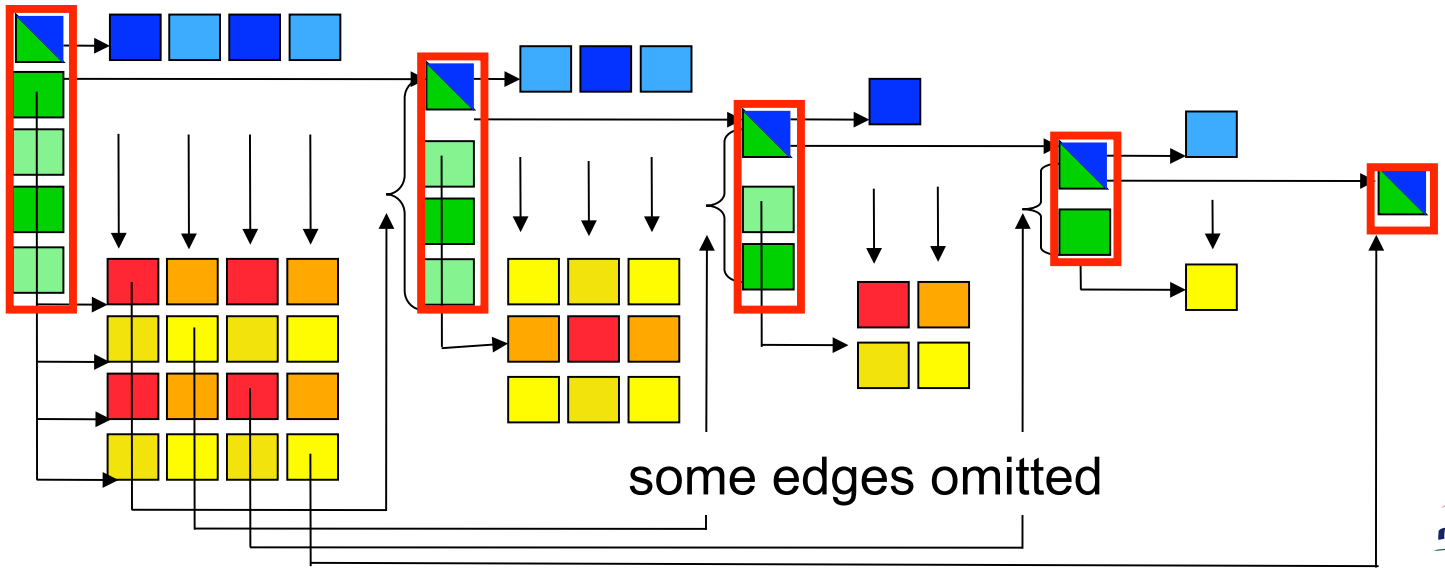


QR
4 x 4

Slide source: Jack Dongarra

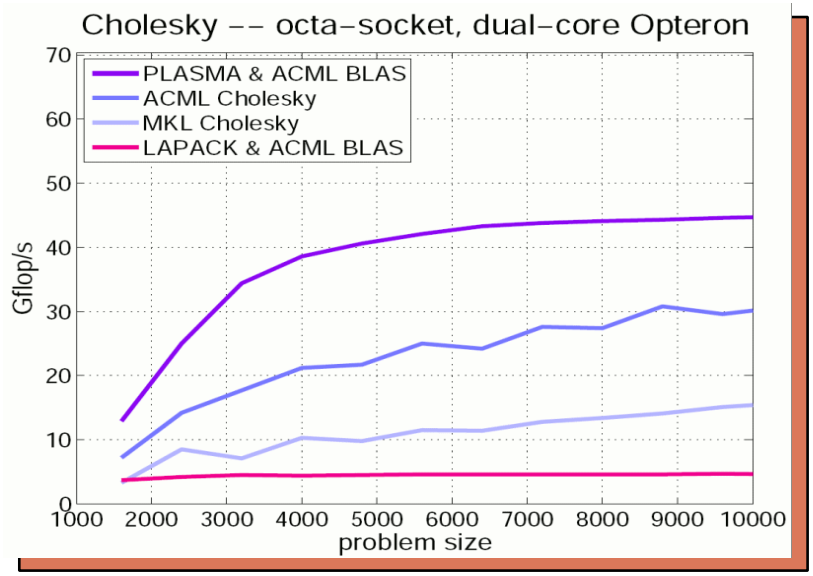
Event Driven Execution of LU

- 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!

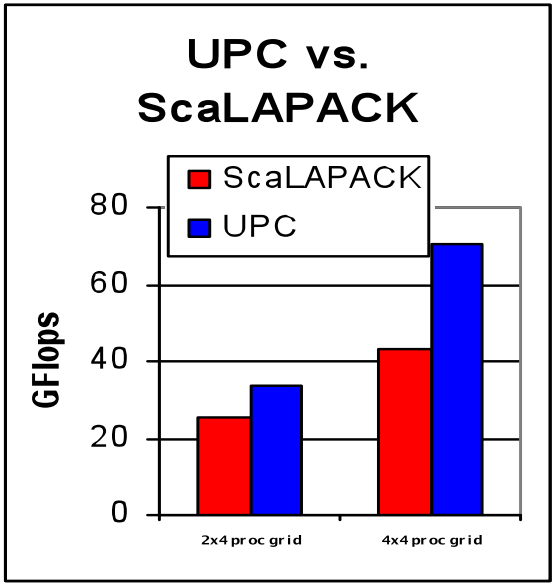


DAG Scheduling Outperforms Bulk-Synchronous Style

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

To Virtualize or Not

- The fundamental question facing in parallel programming models is:

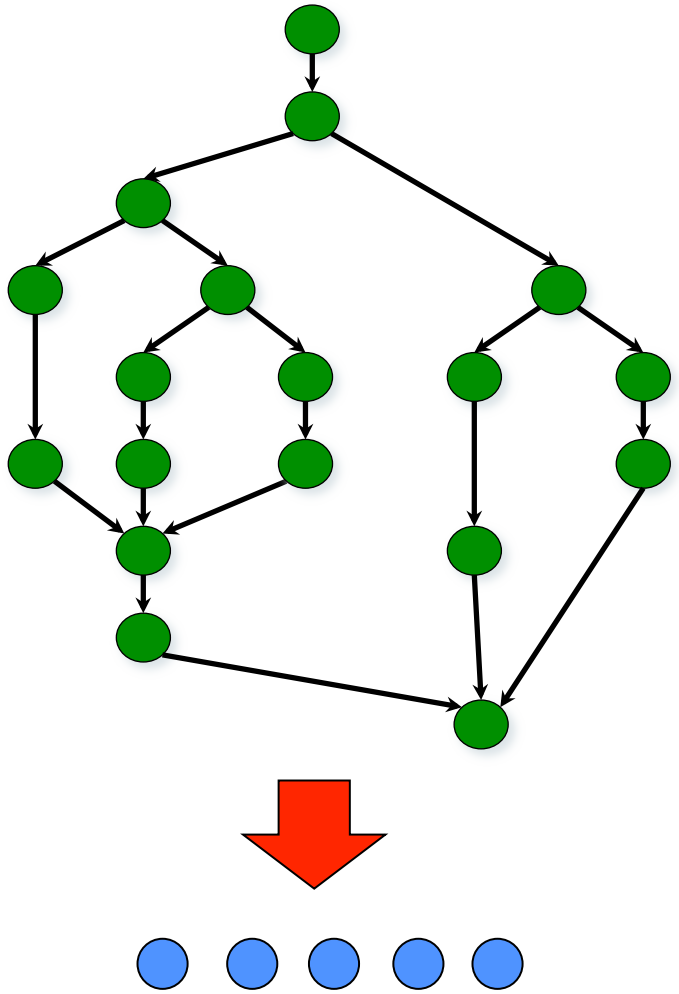
  What should be virtualized?

- Hardware has finite resources
  - Processor count is finite
  - Registers count is finite
  - Fast local memory (cache and DRAM) size is finite
  - Links in network topology are generally < n^2
- Does the programming model (language+libraries) expose this or hide it?
  - E.g., one thread per core, or many?
    - Many threads may have advantages for load balancing, fault tolerance and latency-hiding
    - But one thread is better for deep memory hierarchies
- How to get the most out of your machine?

Reasons to Virtualize

- Simplicity for Programmer
- Potential to hide problems:
  - load imbalance in hardware, e.g., jitter
  - faults
  - wierd memory structures (local stores)
- Effective use of system resources
  - in a space-shared environment
  - multiple jobs sharing resources

Virtualization of Processors



- A parallel computation is defined by its task graph
- Many possible graphs, depending on how much parallelism is exposed
- Where does the mapping of the graph to a particular number of processors happen?
  - The compiler: auto parallelization, NESL, ZPL
  - The runtime system : Cilk, Charm ++ (sometimes), OpenMP, X10
  - The programmer: MPI, UPC

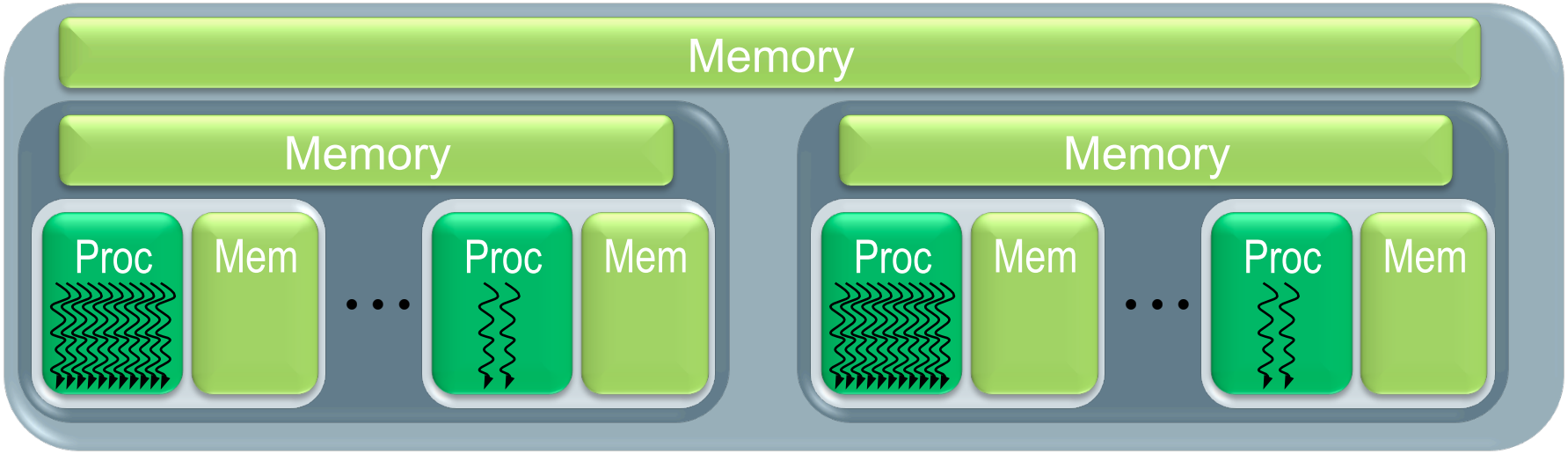
Irregular vs. Regular Parallelism

- Computations with known task graphs can be mapped to resources in an offline manner (before computation starts)
  - Regular graph: By a compiler (static) or runtime (semi-static)
  - Irregular graphs: By a DAG scheduler
  - No need for online scheduling
- If graphs are not known ahead of time (structure, task costs, communication costs), then dynamic scheduling is needed
  - Task stealing / task sharing
  - Demonstrated on shared memory
- Conclusion: If your task graph is dynamic, the runtime needs to be, but what if it static?

Load Balancing with Locality

- Locality is important:
  - When memory hierarchies are deep
  - When computational intensity is low (expensive move cost cannot be amortized)
- Most (all?) successful examples of locality-important applications/ machines use static scheduling
  - Unless they have a irregular/dynamic task graph so it is impossible
- Two extremes are well-studied
  - Dynamic parallelism without locality
  - Static parallelism (with threads = processors) with locality
- Dynamic scheduling (task stealing) with locality control can cause problems
  - Locality control can cause non-optimal task schedule, which can blow up memory use (breadth vs. depth first traversal)
  - Can run out of memory locally when you don't globally

Efficiency Programming Model: Phalanx

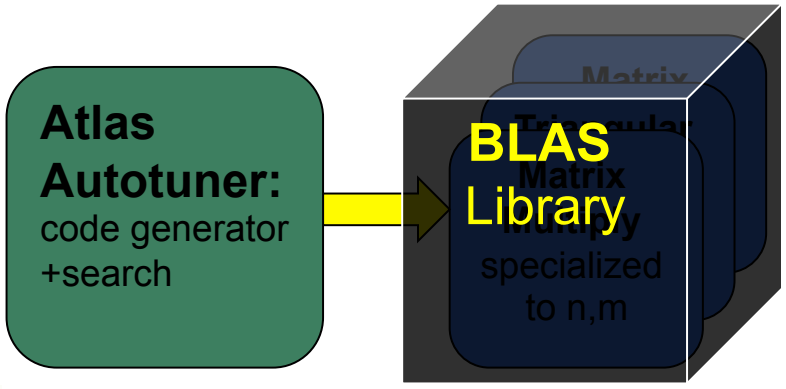


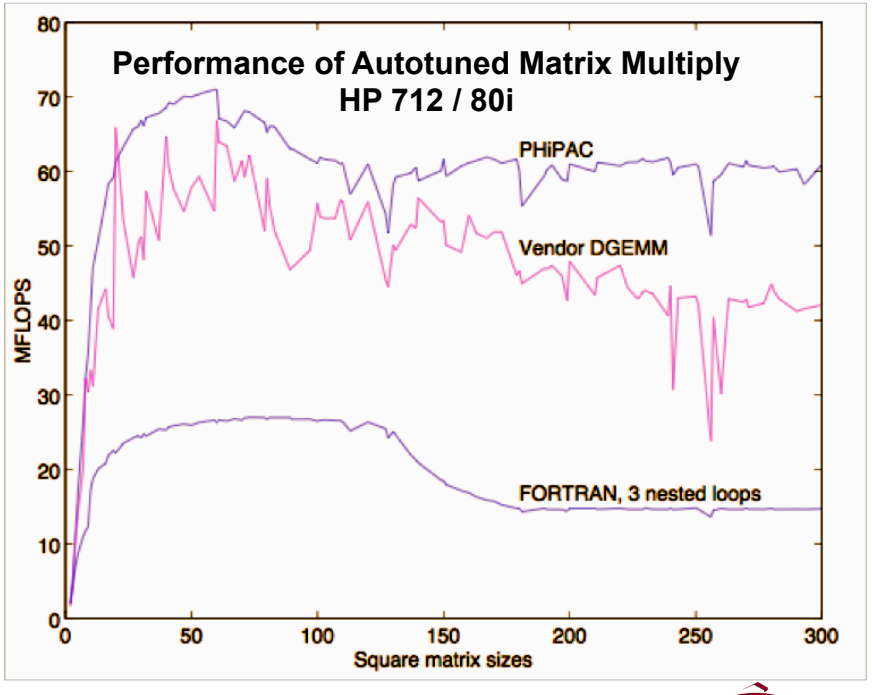
- **Invoke functions on set of cores and set of memories**
- **Hierarchy of memories**
  - **Can query to get (some) aspects of the hierarchical structures**
- **Functionally homogeneous cores (on Echelon)**
  - **Can query to get (performance) properties of cores**
- **Hierarchy of thread blocks**
  - **May be aligned with hardware based on queries**

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

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





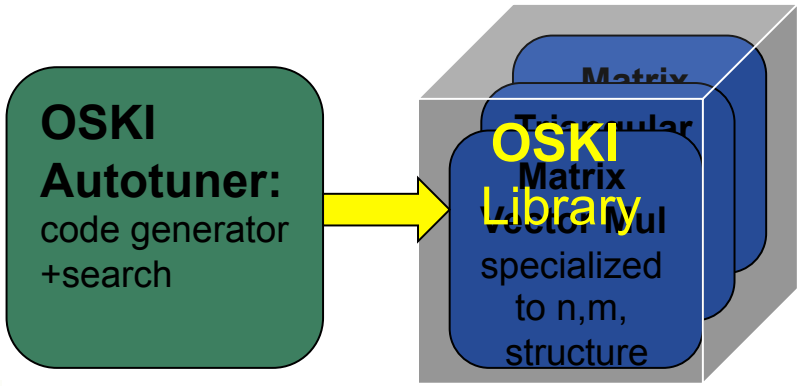
BLAS = Basic Linear Algebra Subroutine: matrix multiply, etc.

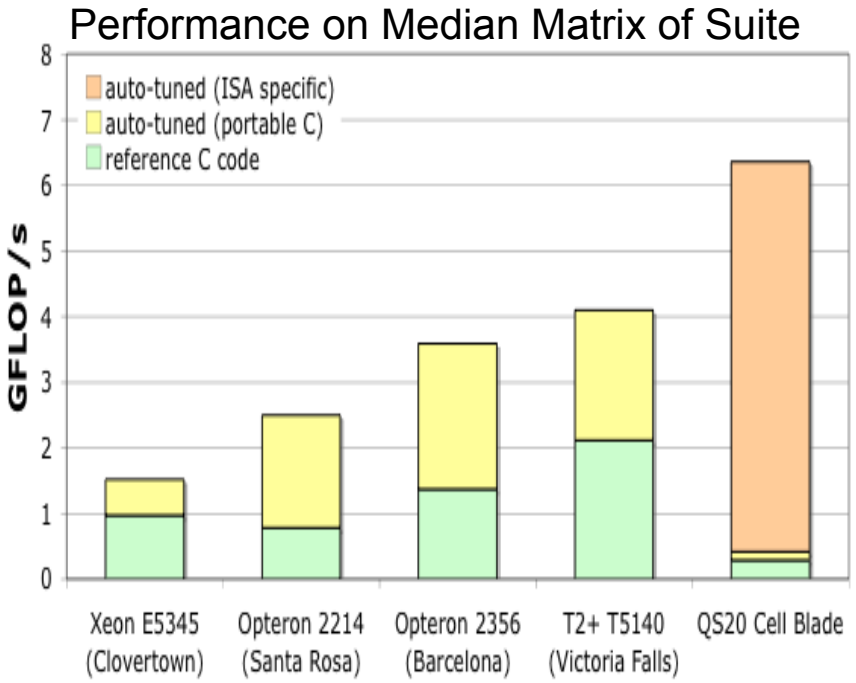
Recent Past Autotuners: Sparse Matrices

- **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
```

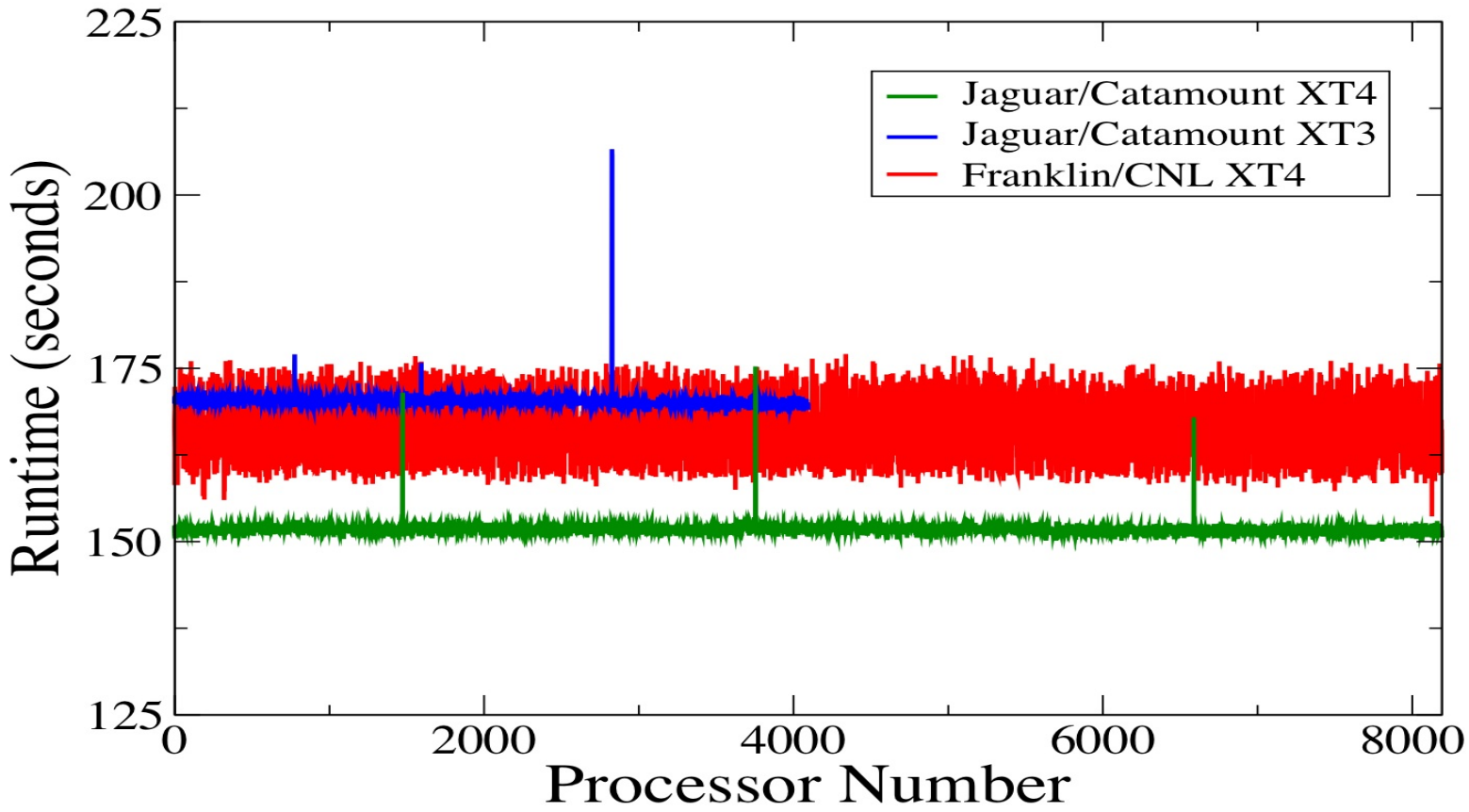
```
VALUE kern_par(int argc, VALUE* argv, VALUE
self) {
unpack_arrays into in_grid and out_grid;

#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;}
```

Resilience at Exascale

- More analysis needed on what faults are most likely and their impact
  - Node / component failures, O(1 day)
    - Kills a job, hopefully not a system
  - System wide outages, O(1 month)
    - Kills all jobs, O(hours) to restart
    - Weakest links: network, file system
- How much to virtualize?
  - Detection of errors visible on demand
  - Automatic recovery: maybe

Errors Can Turn into Performance Problems

- **Fault resilience introduces inhomogeneity in execution rates *(error correction is not instantaneous)***

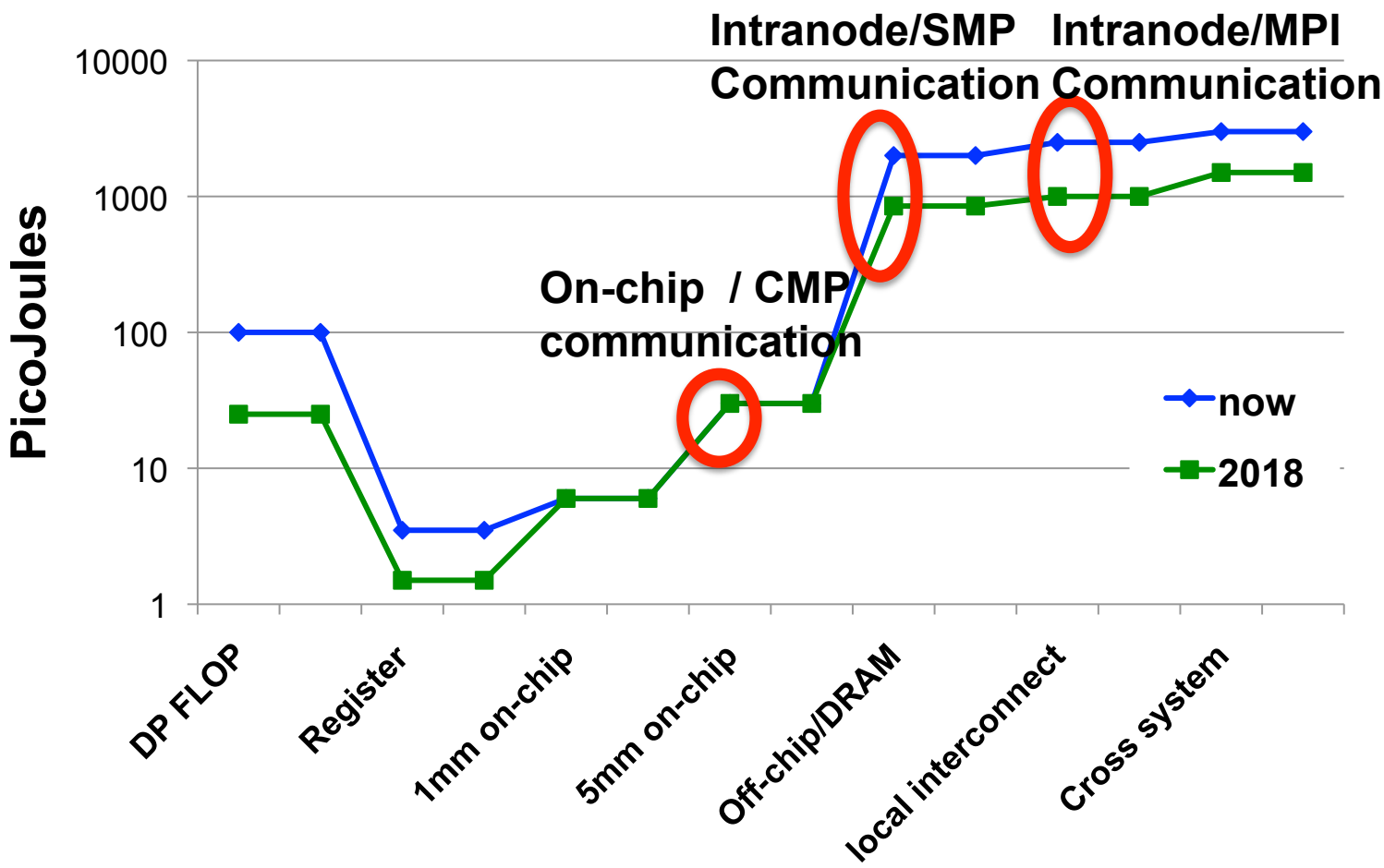


Slide source: John Shalf

Algorithms to Optimize for Communication

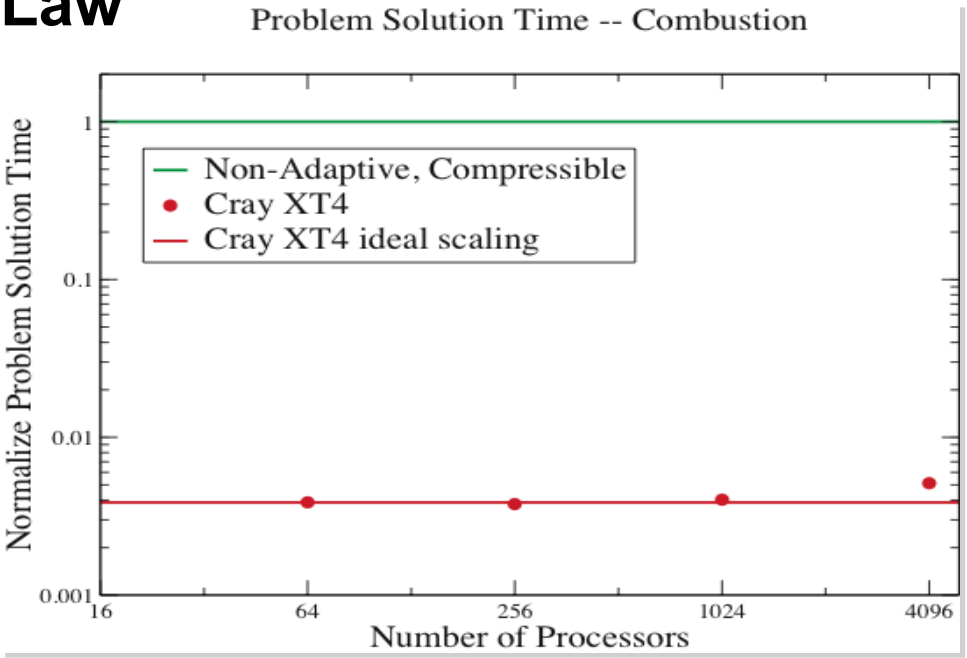
46

Where does the Power Go?



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)

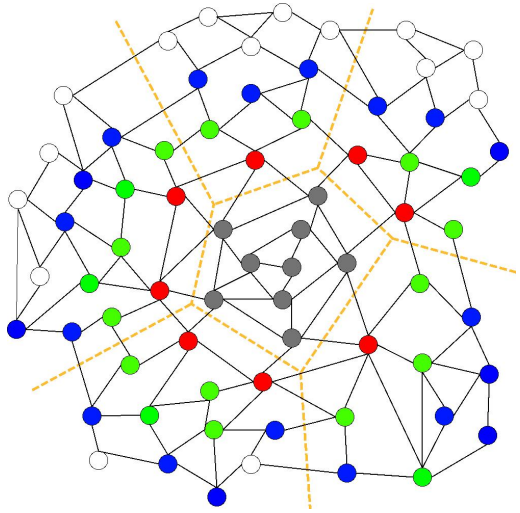


- Example of canonical "Poisson" problem on n points:
  - Dense LU: most general, but $O(n^3)$ flops on $O(n^2)$ data
  - Multigrid: fastest/smallest, O(n) flops on O(n) data

Performance results: John Bell et al

Communication-Avoiding Algorithms

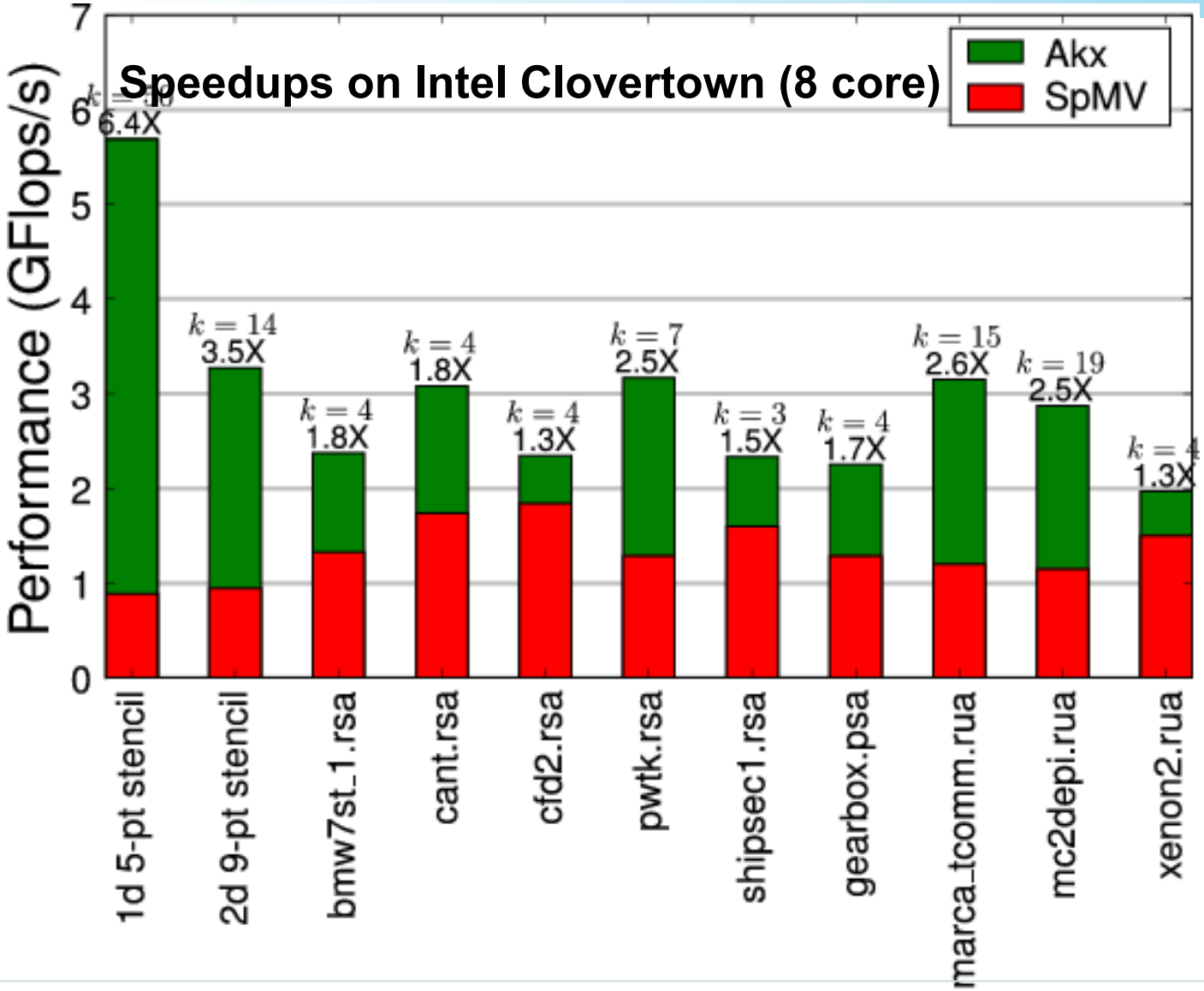
- Sparse Iterative (Krylov Subpace) Methods
  - Nearest neighbor communication on a mesh
  - Dominated by time to read matrix (edges) from DRAM
  - And (small) communication and global synchronization events at each step
- Can we lower data movement costs?
  - Take *k* steps with one matrix read from 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)



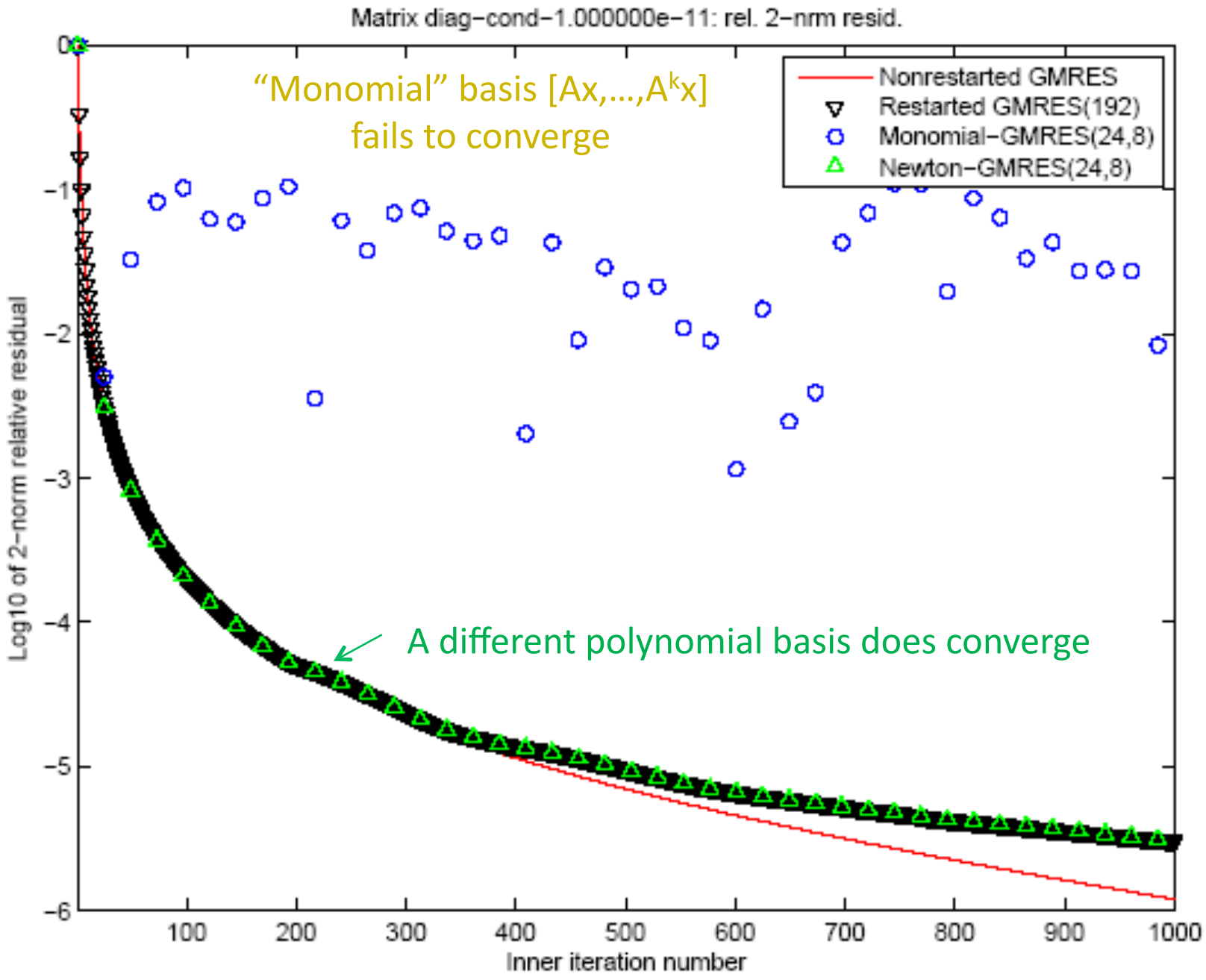
- Can we make communication provably optimal?
  - Communication both to DRAM and between cores
  - Minimize independent accesses ('latency')
  - Minimize data volume ('bandwidth')

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin

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

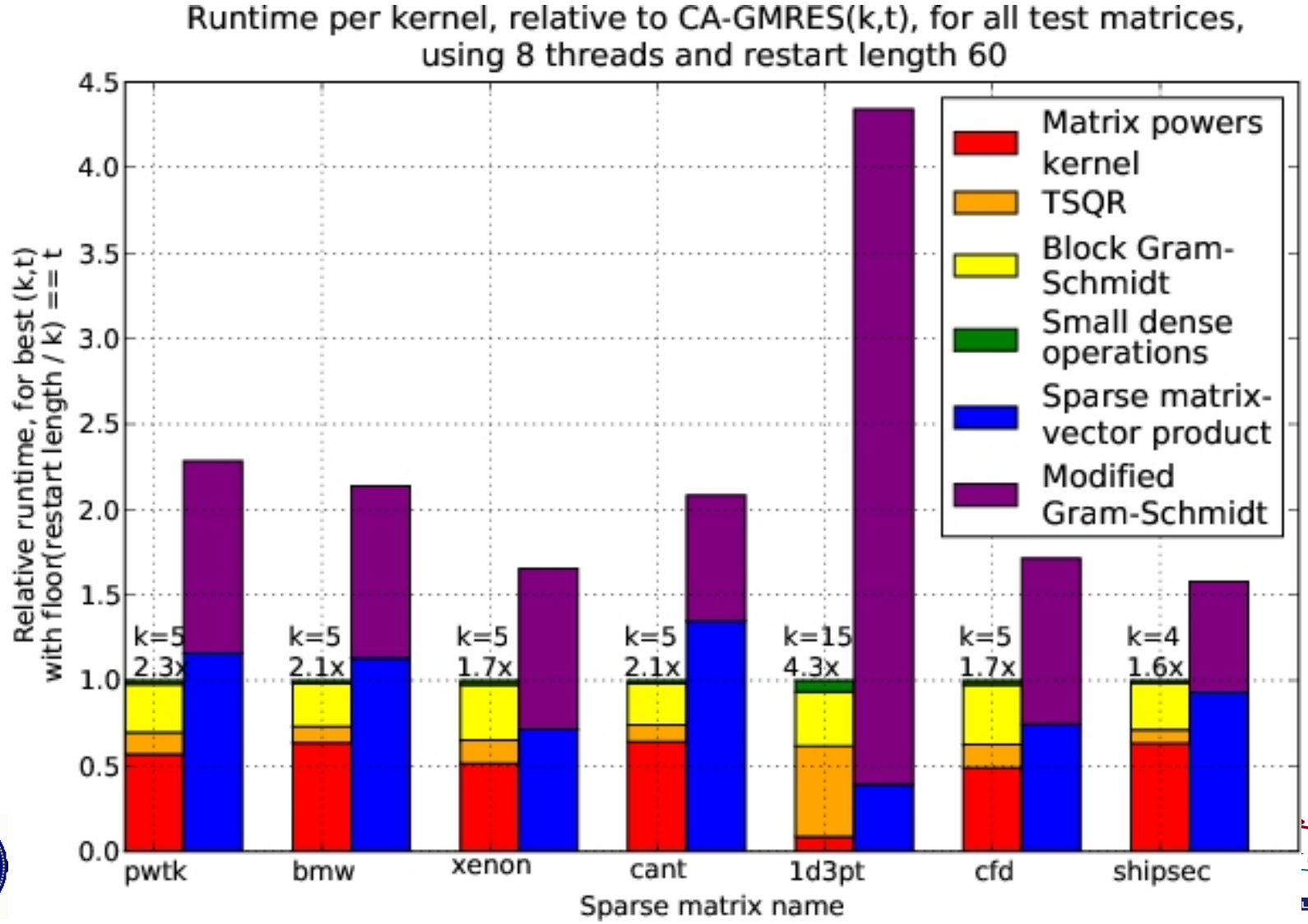


Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Kathy Yelick


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

Thank You!