

Algorithmic Challenges of Exascale Computing

Kathy Yelick

**Associate Laboratory Director for Computing Sciences
and Acting NERSC Center Director
Lawrence Berkeley National Laboratory**

EECS Professor, UC Berkeley



Obama for Communication-Avoiding Algorithms

“New Algorithm Improves Performance and Accuracy on Extreme-Scale Computing Systems. **On modern computer architectures, communication between processors takes longer than the performance of a floating point arithmetic operation by a given processor.** ASCR researchers have developed a new method, derived from commonly used linear algebra methods, to **minimize communications between processors and the memory hierarchy, by reformulating the communication patterns specified within the algorithm.** This method has been implemented in the TRILINOS framework, a highly-regarded suite of software, which provides functionality for researchers around the world to solve large scale, complex multi-physics problems.”

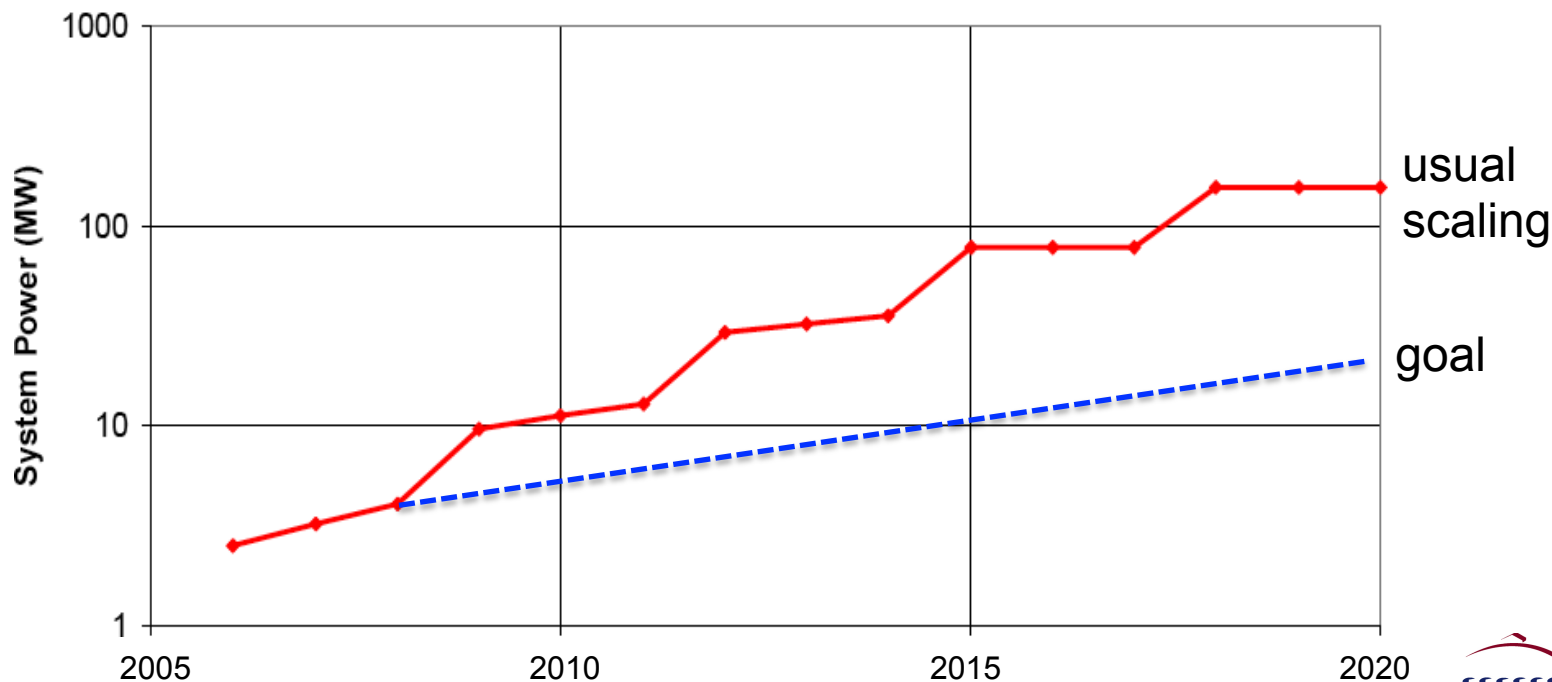
FY 2012 Congressional Budget Request, Volume 4, FY2010 Accomplishments, Advanced Scientific Computing Research (ASCR), pages 65-67.



Energy Cost Challenge for Computing Facilities

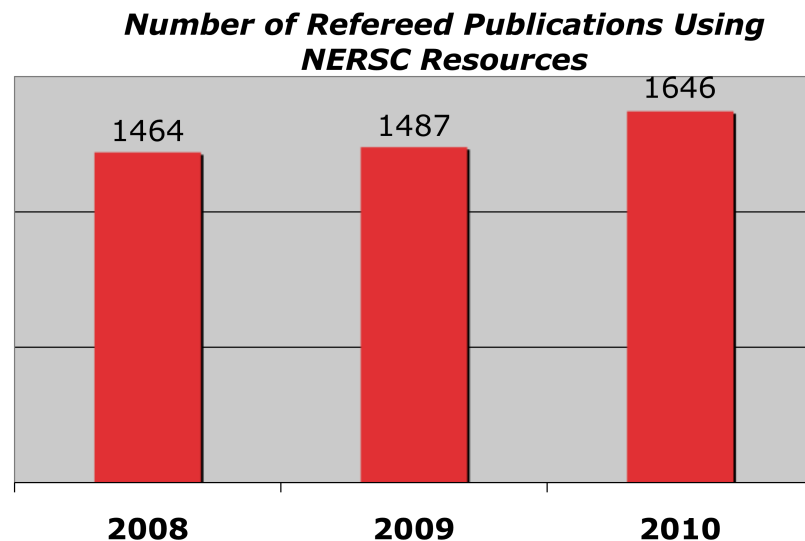
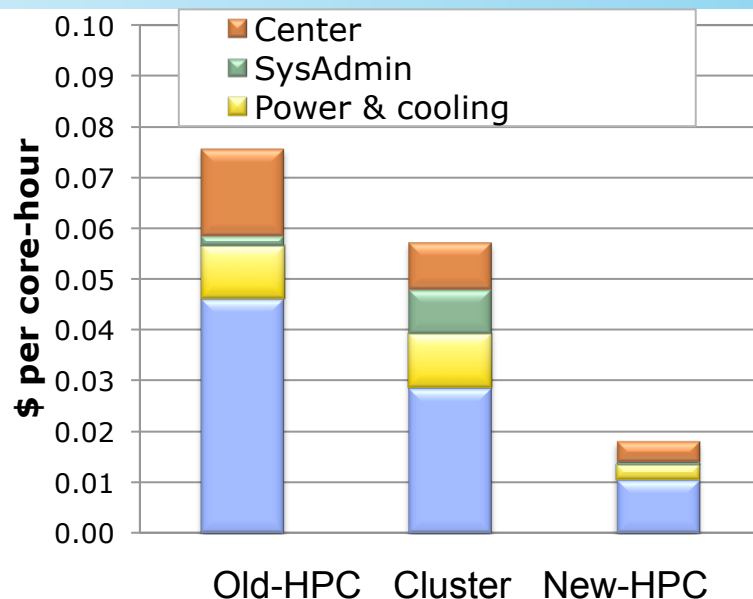
At ~\$1M per MW, energy costs are substantial

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



Measuring Efficiency

- Race-to-Halt generally minimized energy use
- For Scientific Computing centers, the metric should be science output per Watt....
 - *NERSC in 2010 ran at 450 publications per MW-year*
 - But that number drops with each new machine
- Next best: application performance per Watt
 - Newest, largest machine is best
 - Lower energy and cost per core

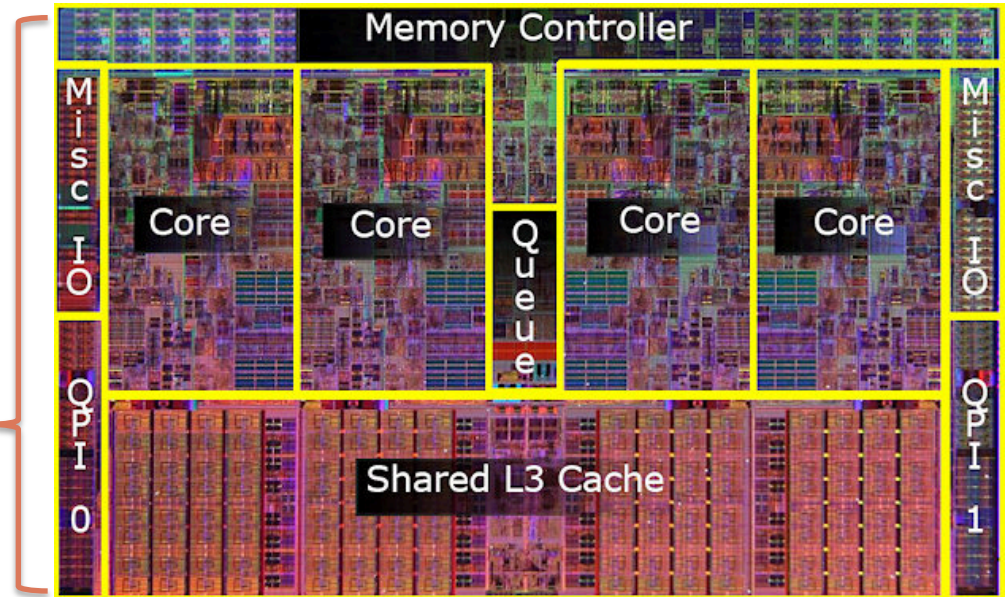


New Processor Designs are Needed to Save Energy



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

Server processor
(100 Watts, 50 Gflop/s)

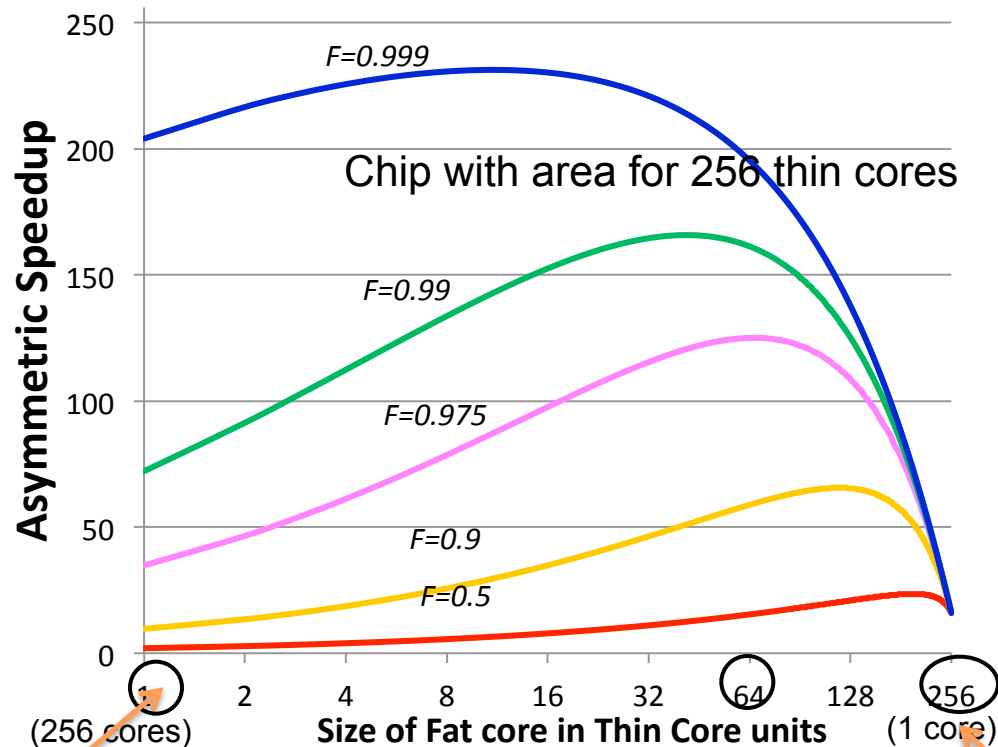


- **Server processors have been designed for performance, not energy**
 - Graphics processors are 10-100x more efficient
 - Embedded processors are 100-1000x
 - Need manycore chips with thousands of cores



The Amdahl Case for Heterogeneity

F is fraction of time in parallel; $1-F$ is serial



Assumes
speedup for
Fat / Thin =
Sqrt of Area
advantage

256 small cores

1 fat core

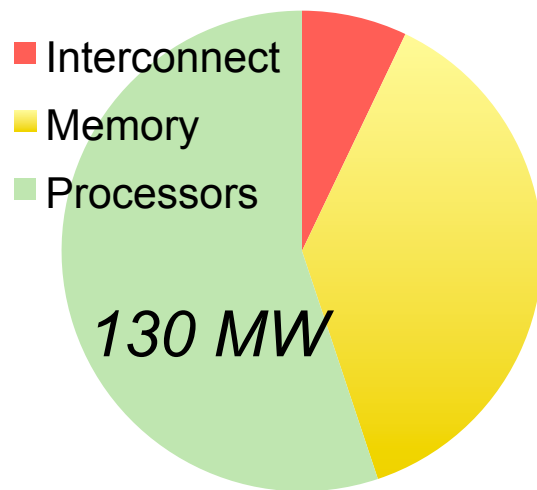
A Chip with up to 256 “thin” cores and “fat” core that uses some of the some of the thin core area

Heterogeneity Analysis by: Mark Hill, U. Wisc

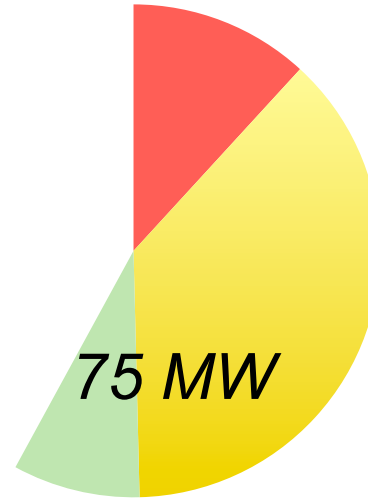


1/11/12

New Processors Means New Software



Server Processors



Manycore

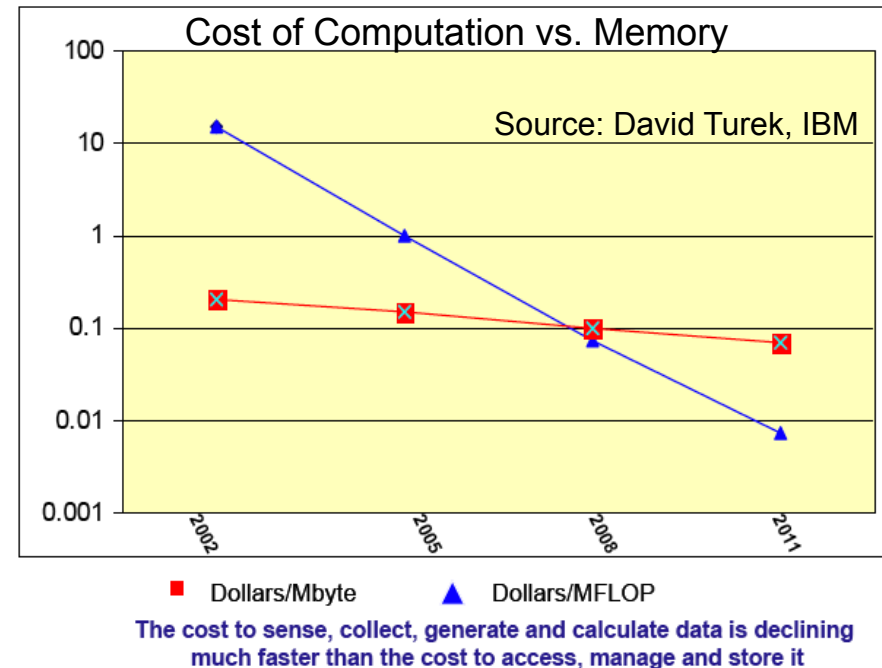
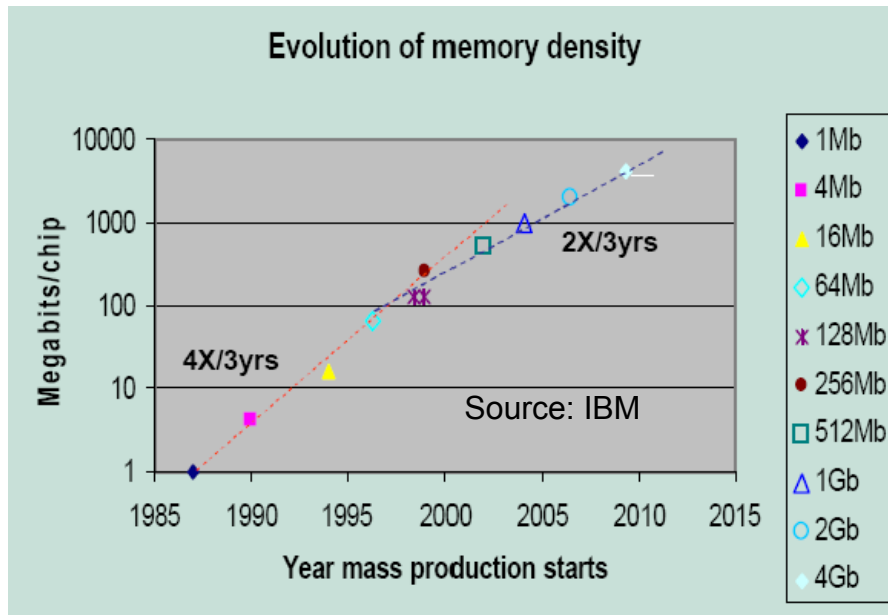
- **Exascale will have chips with thousands of tiny processor cores, and a few large ones**
- **Architecture is an open question:**
 - sea of embedded cores with heavyweight “service” nodes
 - Lightweight cores are accelerators to CPUs



Memory Capacity 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



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



Why avoid communication?

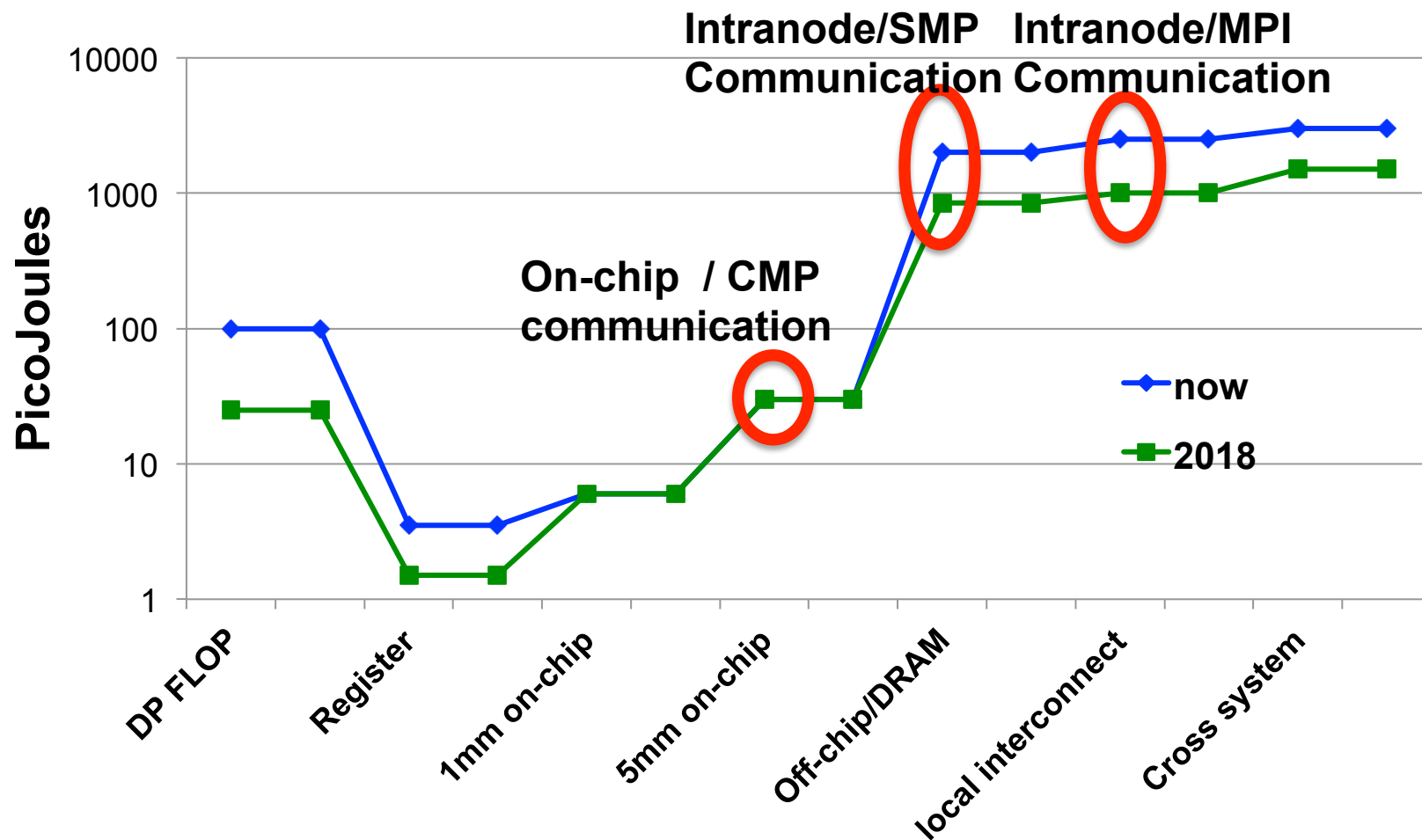
- Running time of an algorithm is sum of 3 terms:
 - $\# \text{ flops} * \text{time_per_flop}$
 - $\# \text{ words moved} / \text{bandwidth}$
 - $\# \text{ messages} * \text{latency}$ } communication
- $\text{Time_per_flop} \ll 1 / \text{bandwidth} \ll \text{latency}$
 - Gaps growing exponentially with time [FOSC]

Annual improvements			
Time_per_flop		Bandwidth	Latency
59%	Network	26%	15%
	DRAM	23%	5%

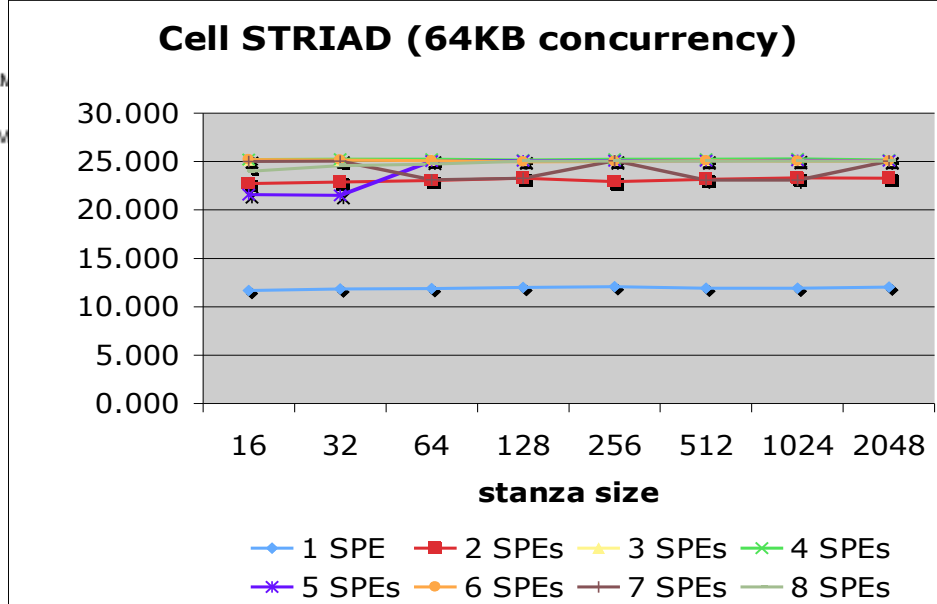
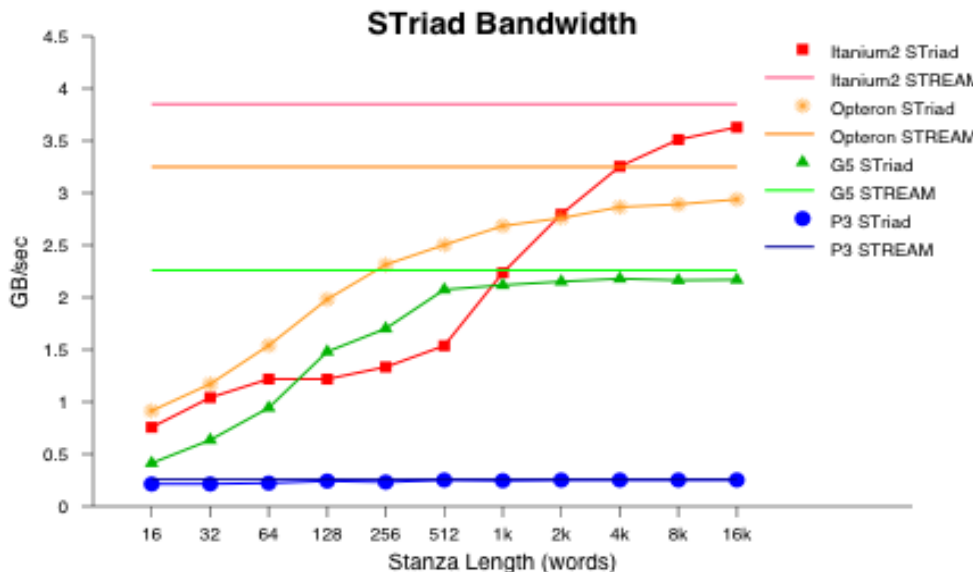
- And these are hard to change:
 - “Latency is physics, bandwidth is money”



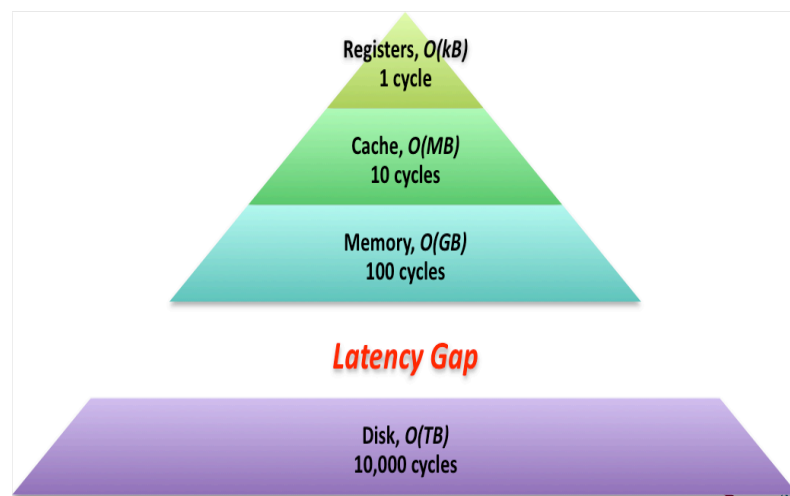
Bandwidth (to Memory and Remote Nodes) is an Energy Hog



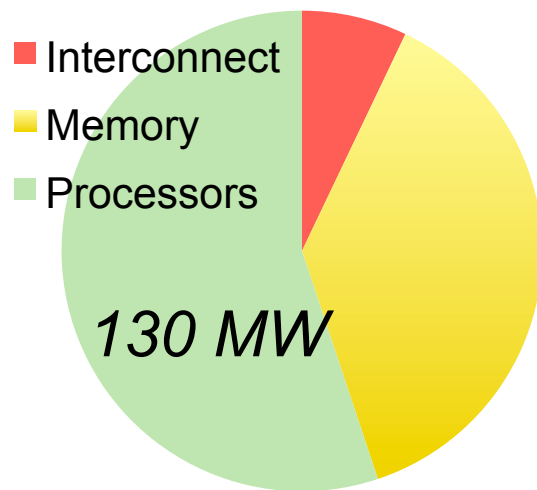
Value of Local Store Memory



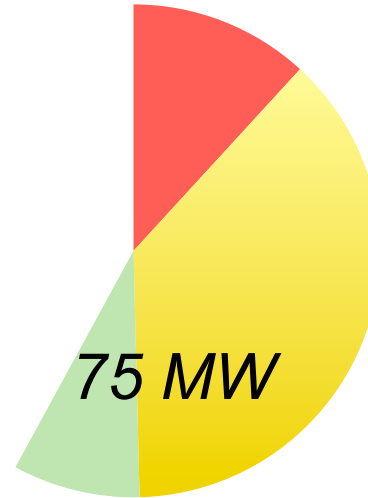
- Unit stride as important as cache hits on hardware with prefetch
 - Don't cut unit stride when tiling
- Software controlled memory gives more control ("scrathpad")
 - May also be more for new level of memory between DRAM and disk



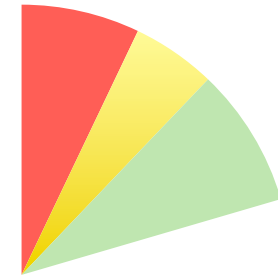
New Processors Means New Software



Server Processors



Manycore



25 Megawatts

*Low power memory
and interconnect*

- **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
- **Low power memory and storage technology are key**

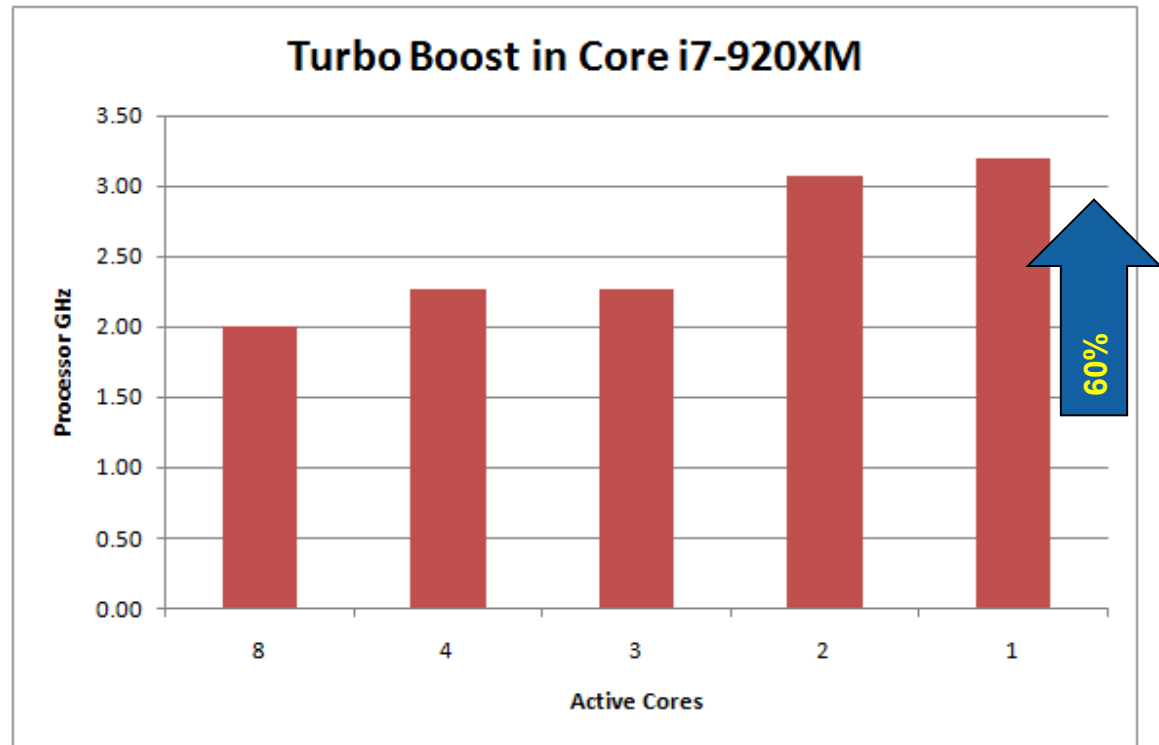


Why Avoid Synchronization?

- **Processors do not run at the same speed**
 - Never did, due to caches
 - Power / temperature management makes this worse

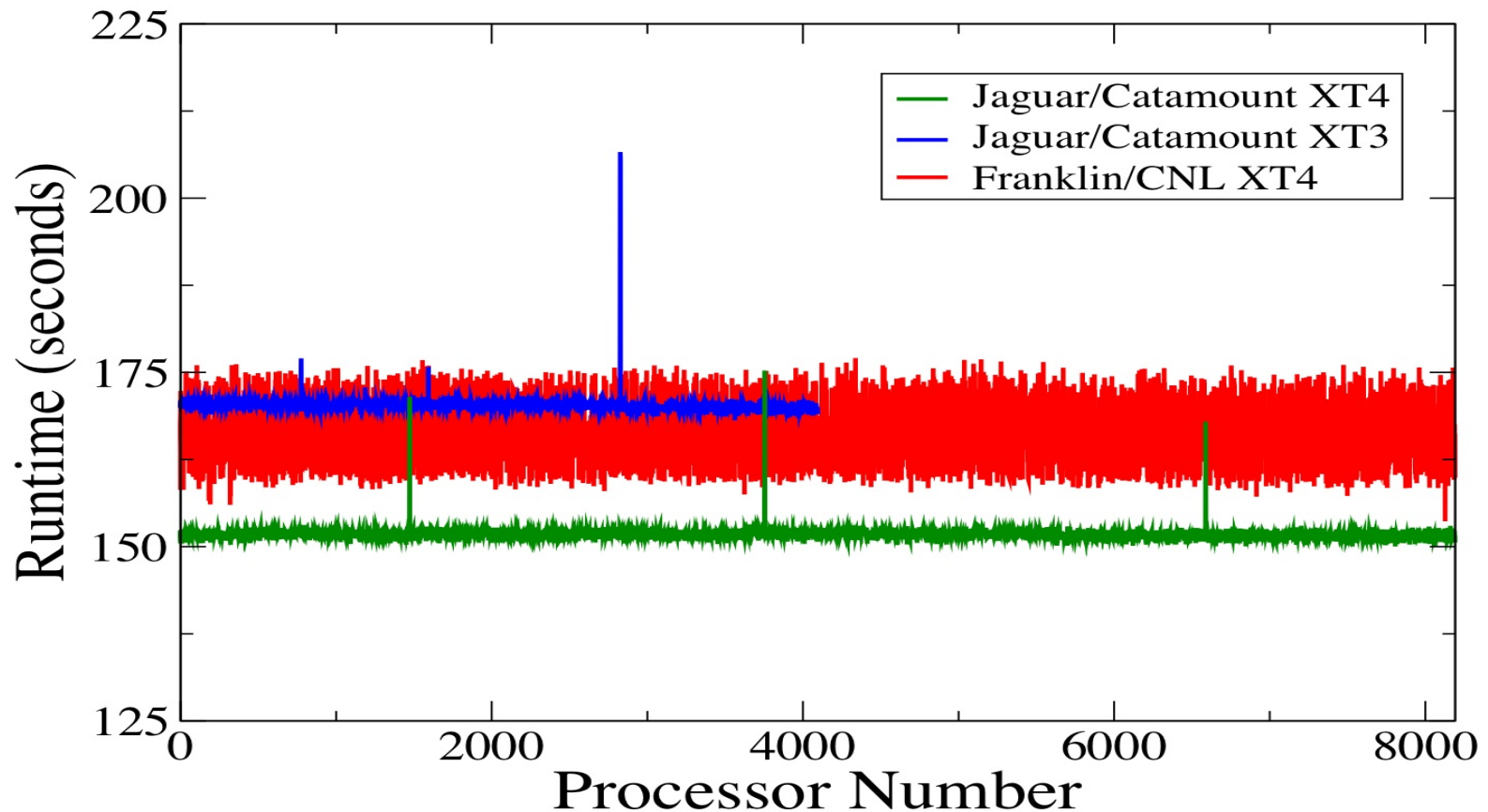
HPC can't turn this off

- Power swings of 50% on systems
- At \$3M/MW of capital costs, don't want 50% headroom



Errors Can Turn into Performance Problems

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



Slide source: John Shalf



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) **Resiliency** critical at large scale (in time or processors)
- 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



Algorithms to Optimize for Communication



Avoiding Communication in Iterative Solvers

- **Consider Sparse Iterative Methods for $Ax=b$**
 - **Krylov Subspace Methods: GMRES, CG,...**
 - **Can we lower the communication costs?**
 - Latency of communication, i.e., reduce # messages by computing multiple reductions at once
 - Bandwidth to memory hierarchy, i.e., compute Ax , A^2x , ... A^kx with one read of A
- **Solve time dominated by:**
 - **Sparse matrix-vector multiple (SPMV)**
 - Which even on one processor is dominated by “communication” time to read the matrix
 - **Global collectives (reductions)**
 - **Global latency-limited**

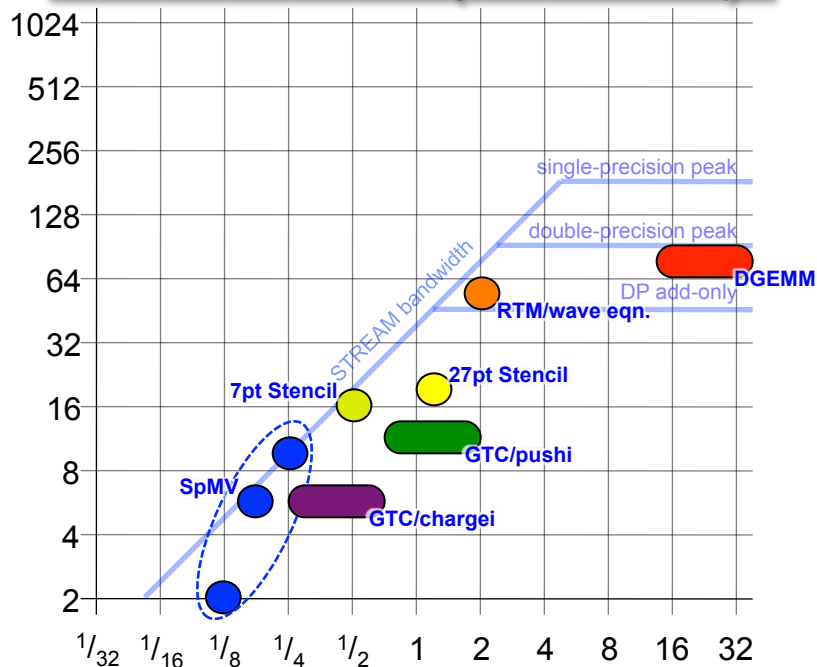
Joint work with Jim
Demmel, Mark Hoemman,
Marghoob Mohiyuddin



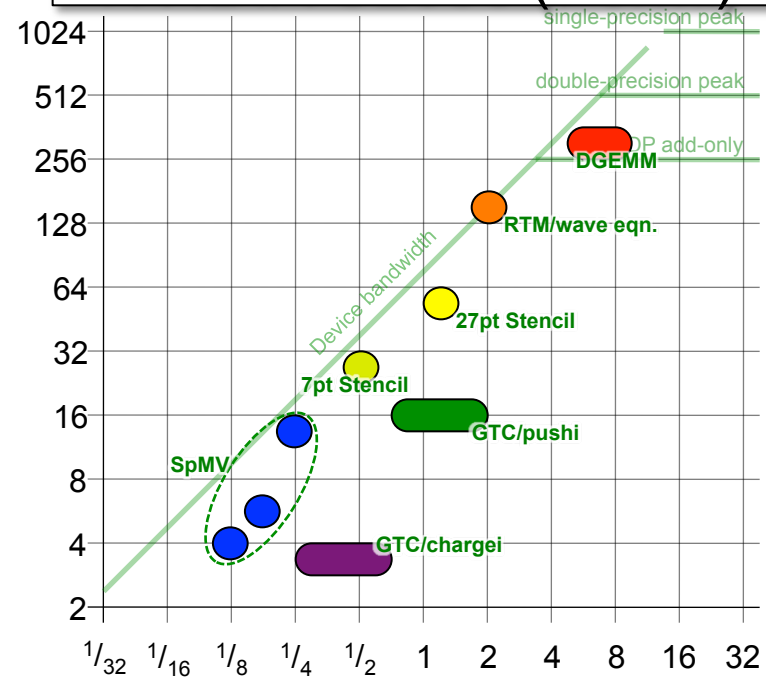
Autotuning Gets Kernel Performance Near Optimal

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

Xeon X5550 (Nehalem)



NVIDIA C2050 (Fermi)



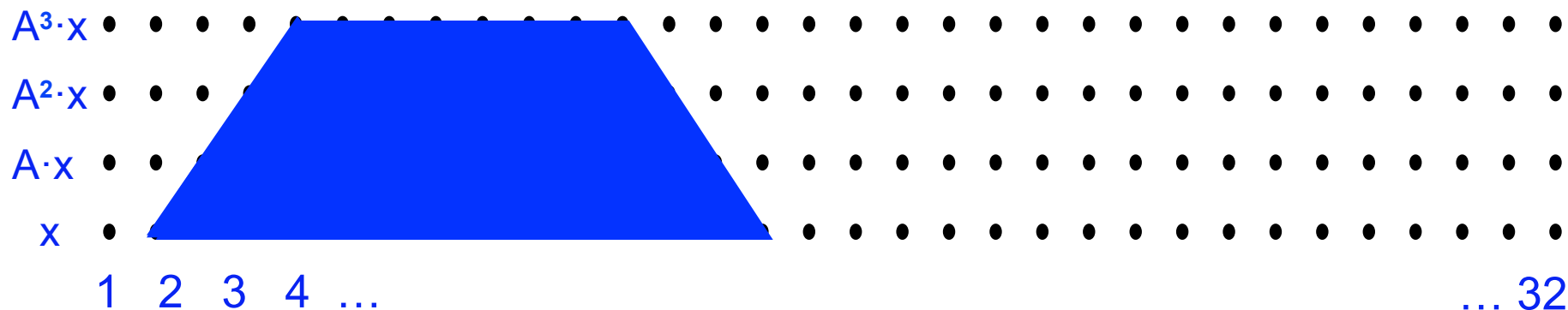
Work by Williams, Oliker, Shalf, Madduri, Kamil, Im, Ethier,...



Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \dots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \dots, A^kx]$



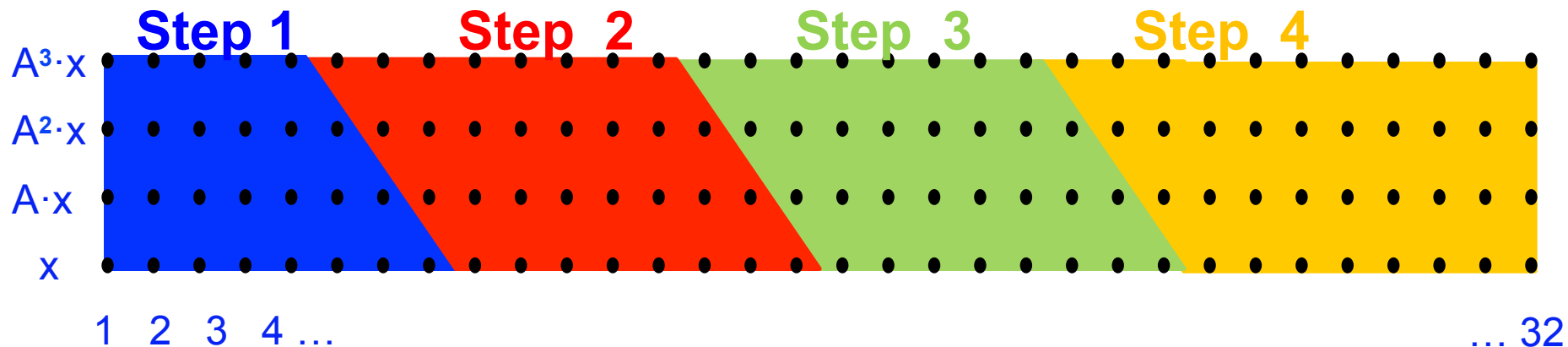
- Idea: pick up part of A and x that fit in fast memory, compute each of k products
- Example: A tridiagonal, $n=32$, $k=3$
- Works for any “well-partitioned” A



Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \dots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \dots, A^kx]$
- **Sequential Algorithm**



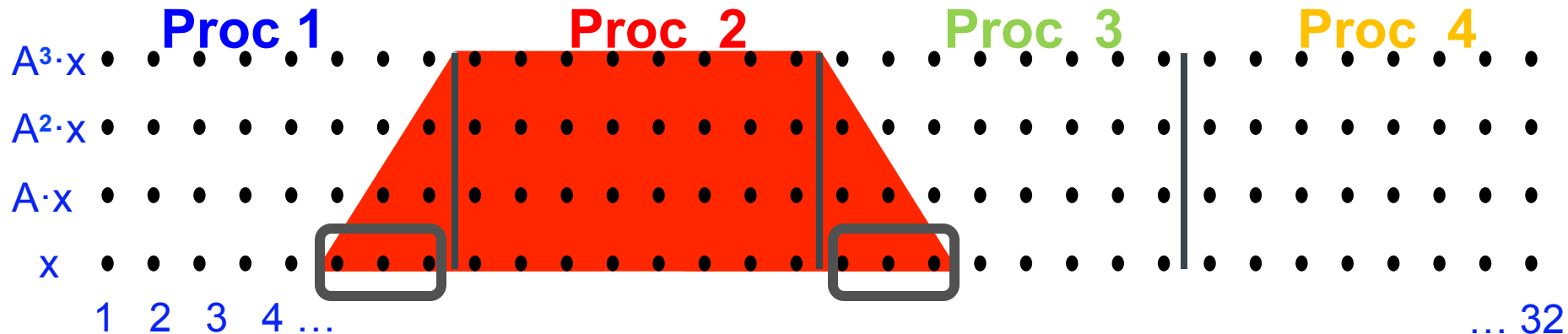
- Example: A tridiagonal, $n=32$, $k=3$
- Saves bandwidth (one read of $A \& x$ for k steps)
- Saves latency (number of independent read events)



Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \dots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \dots, A^kx]$
- **Parallel Algorithm**



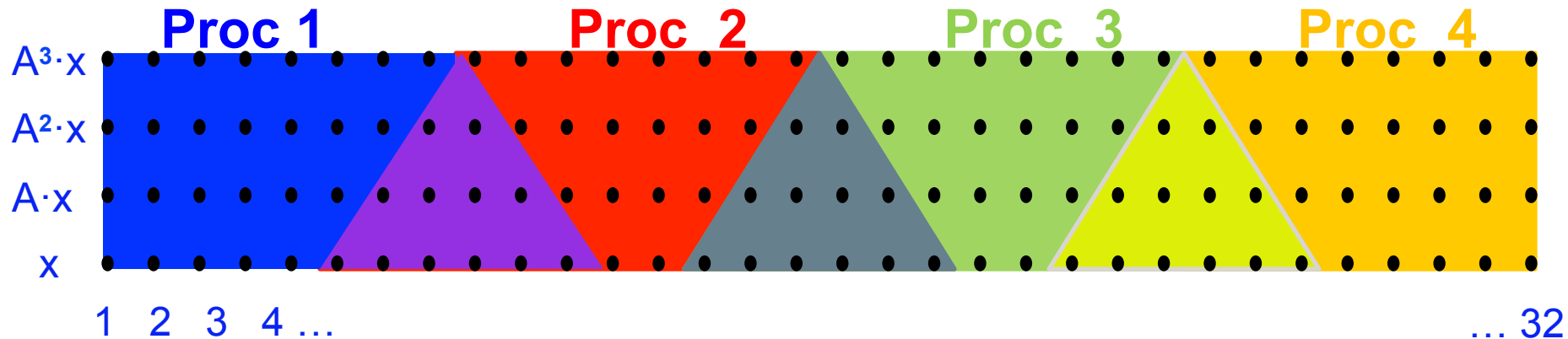
- Example: A tridiagonal, $n=32$, $k=3$
- Each processor communicates once with neighbors



Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \dots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \dots, A^kx]$
- **Parallel Algorithm**

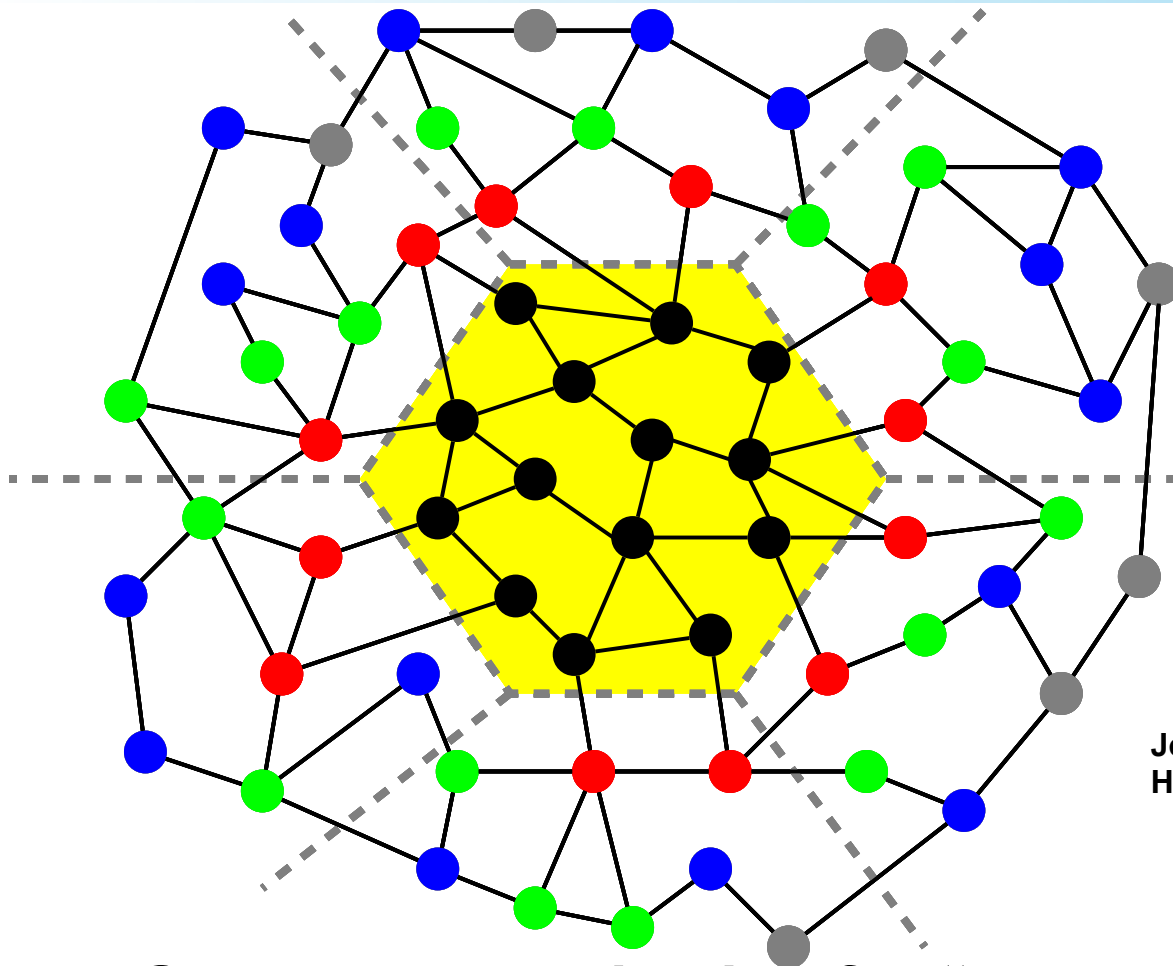


- Example: A tridiagonal, $n=32$, $k=3$
- Each processor works on (overlapping) trapezoid
- Saves latency (# of messages); Not bandwidth



But adds redundant computation

Matrix Powers Kernel on a General Matrix



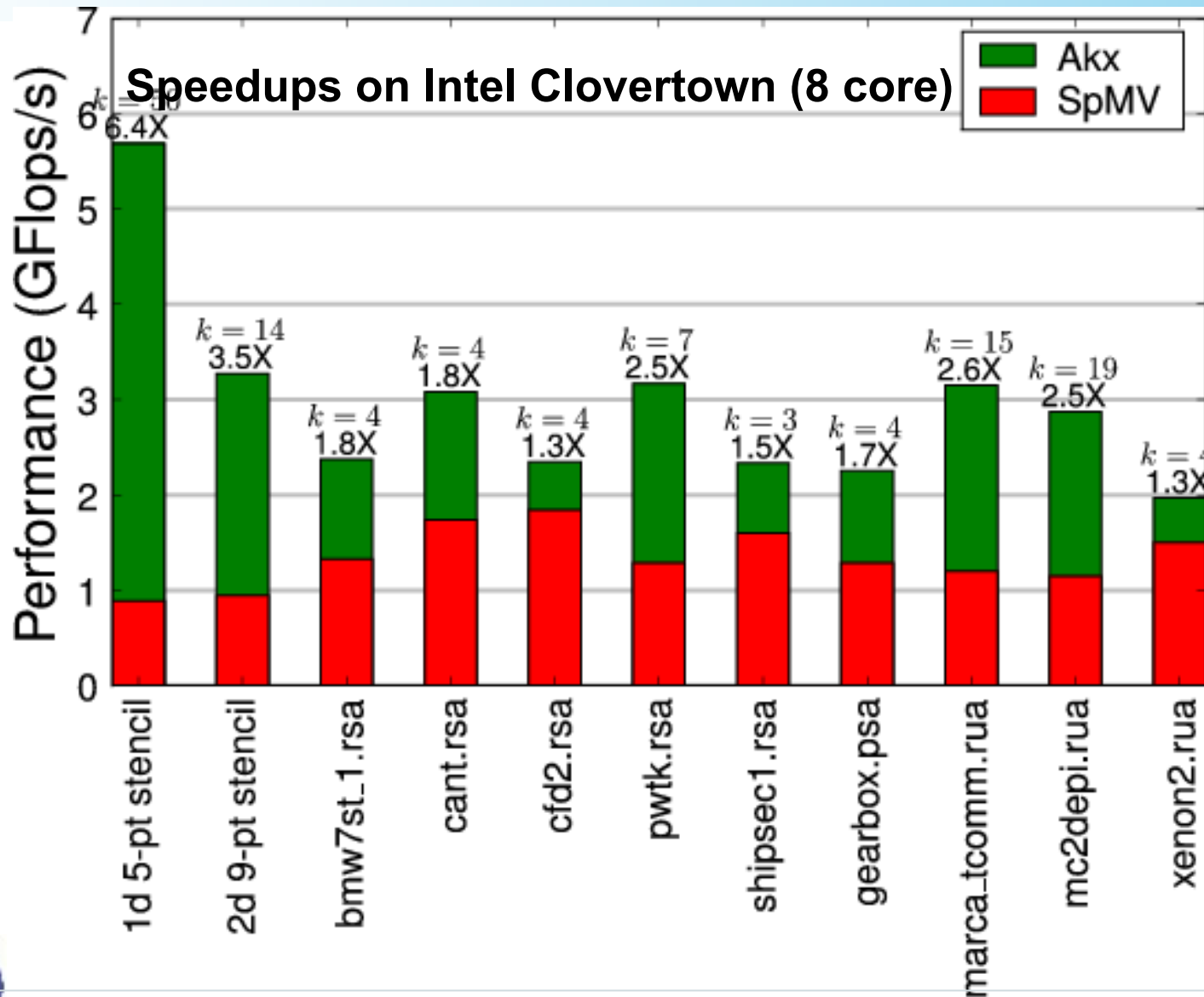
For implicit memory management (caches) uses a TSP algorithm for layout

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin

- **Saves communication for “well partitioned” matrices**
 - Serial: $O(1)$ moves of data moves vs. $O(k)$
 - Parallel: $O(\log p)$ messages vs. $O(k \log p)$



Bigger Kernel (A^kx) Runs at Faster Speed than Simpler (Ax)



Minimizing Communication of GMRES to solve $Ax=b$

- **GMRES:** find x in $\text{span}\{b, Ab, \dots, A^k b\}$ minimizing $\|Ax - b\|_2$

Standard GMRES

for $i=1$ to k

$w = A \cdot v(i-1) \dots \text{SpMV}$

$\text{MGS}(w, v(0), \dots, v(i-1))$

update $v(i)$, H

endfor

solve LSQ problem with H

Communication-avoiding GMRES

$W = [v, Av, A^2v, \dots, A^k v]$

$[Q, R] = \text{TSQR}(W)$

\dots “Tall Skinny QR”

build H from R

solve LSQ problem with H

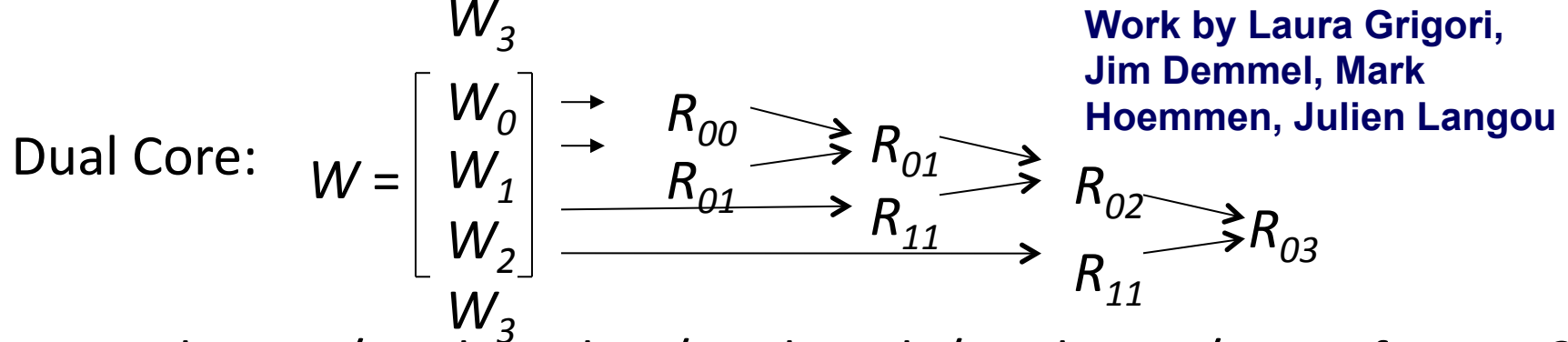
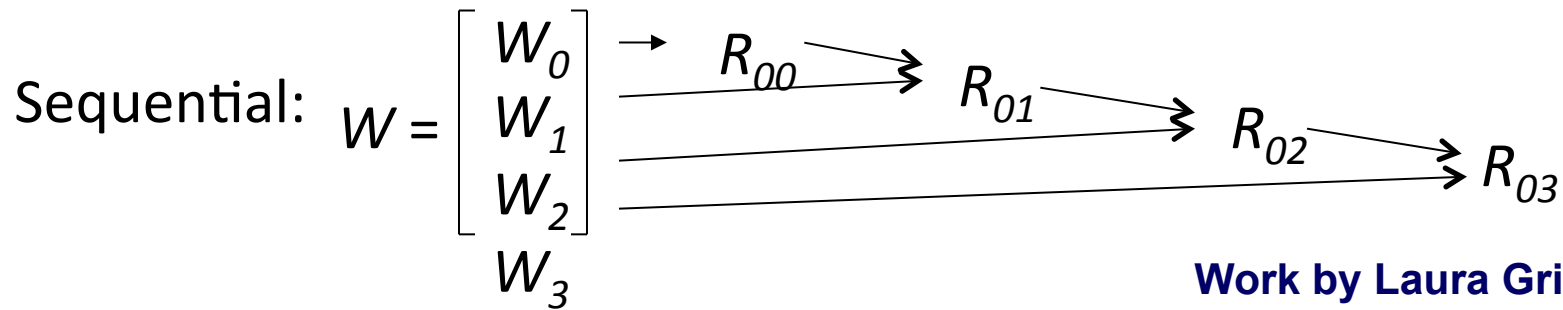
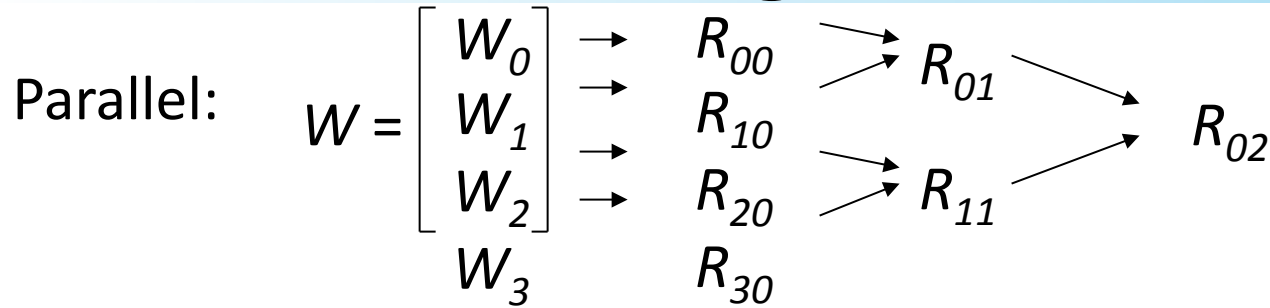
Sequential case: #words moved decreases by a factor of k

Parallel case: #messages decreases by a factor of k

- **Oops – W from power method, precision lost!**



TSQR: An Architecture-Dependent Algorithm



Multicore / Multisocket / Multirack / Multisite / Out-of-core: ?

Can choose reduction tree dynamically



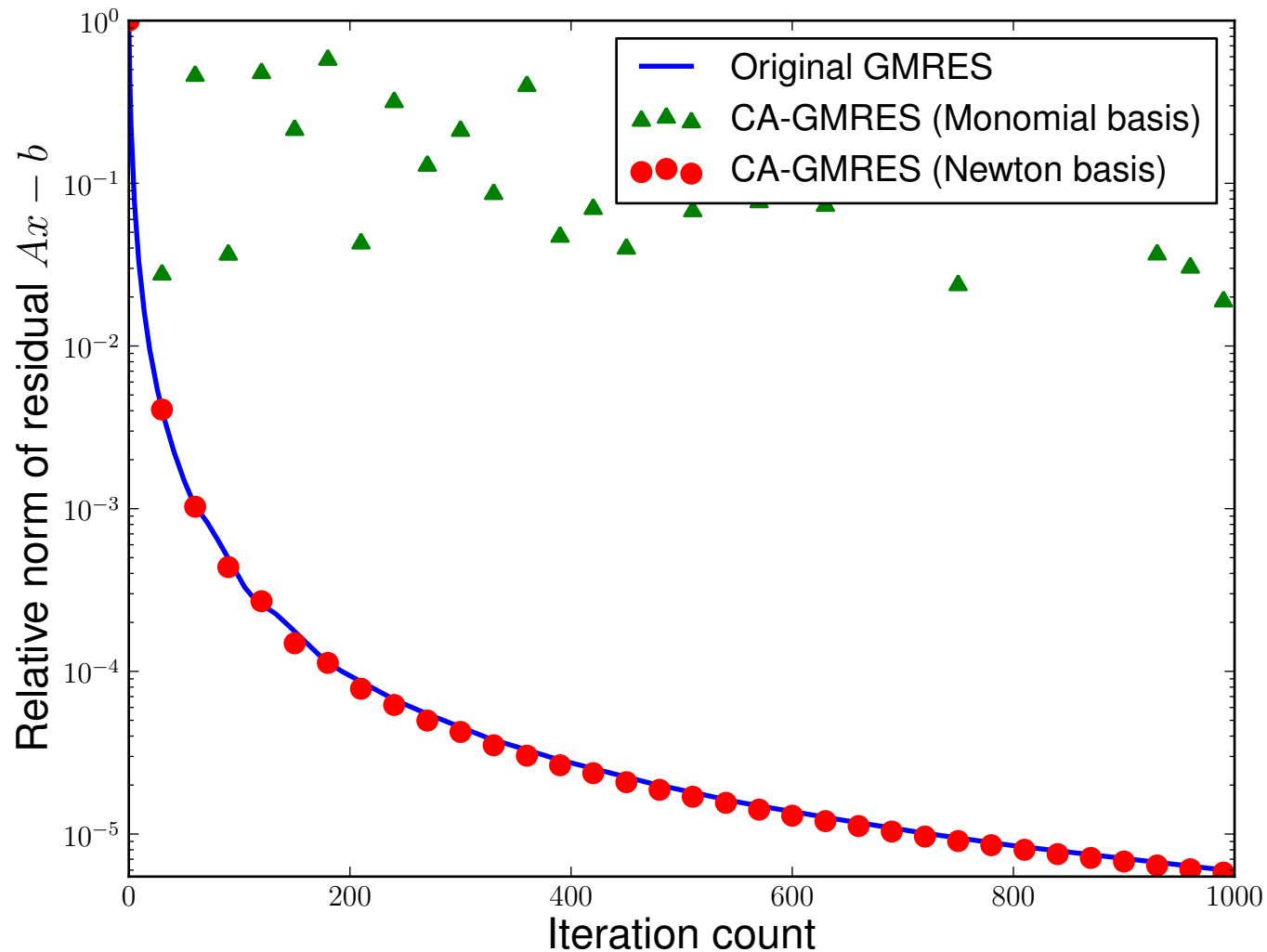
TSQR Performance Results

- **Parallel**
 - **Intel Clovertown**
 - Up to 8x speedup (8 core, dual socket, 10M x 10)
 - **Pentium III cluster, Dolphin Interconnect, MPICH**
 - Up to 6.7x speedup (16 procs, 100K x 200)
 - **BlueGene/L**
 - Up to 4x speedup (32 procs, 1M x 50)
 - **Grid – 4x on 4 cities (Dongarra et al)**
 - **Cloud – early result – up and running using Mesos**
- **Sequential**
 - **Out-of-Core on PowerPC laptop**
 - As little as 2x slowdown vs (predicted) infinite DRAM
 - LAPACK with virtual memory never finished

Data from Grey Ballard, Mark Hoemmen, Laura Grigori, Julien Langou, Jack Dongarra, Michael Anderson



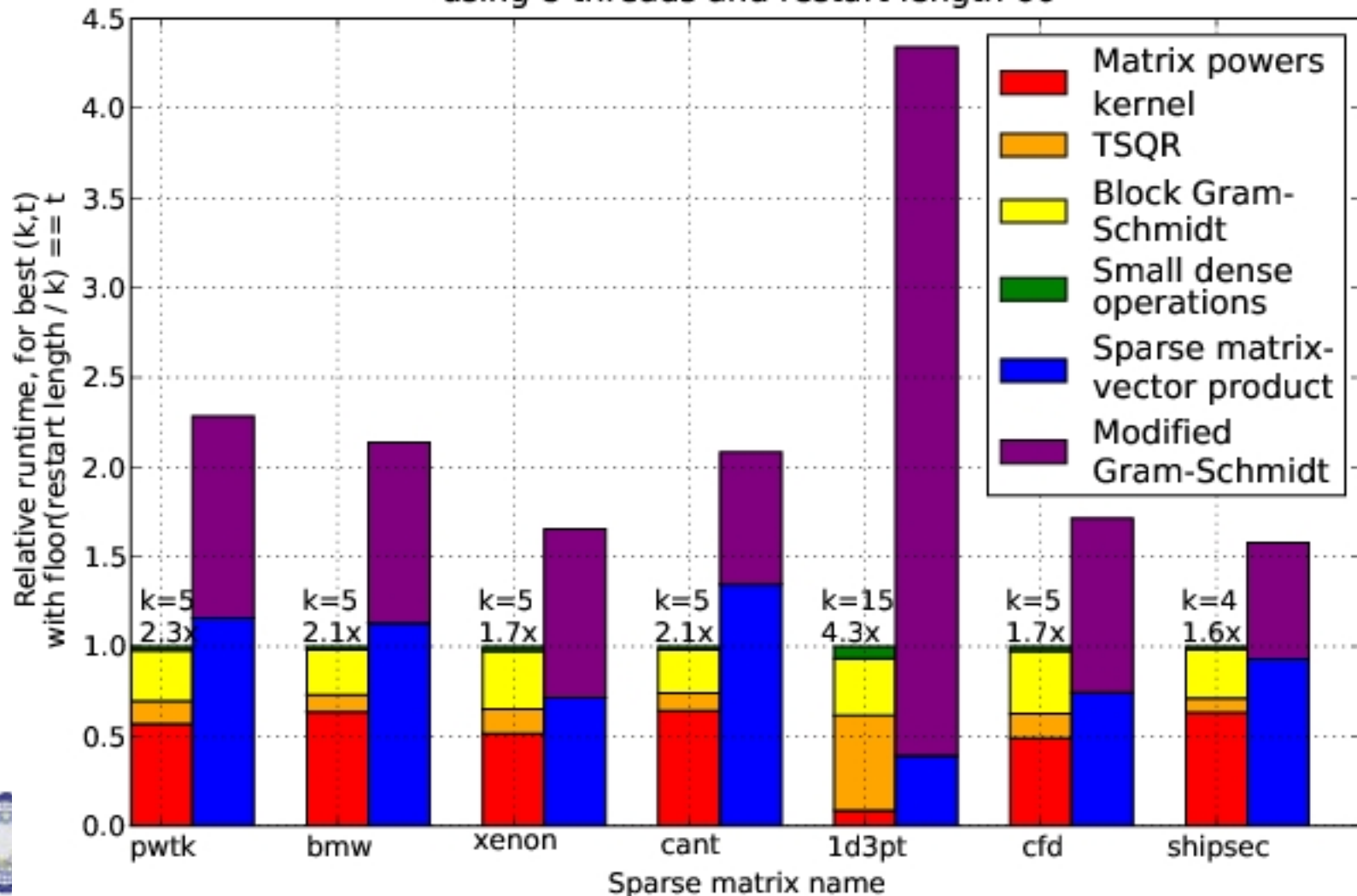
Matrix Powers Kernel (and TSQR) in GMRES



Communication-Avoiding Krylov Method (GMRES)

Performance on 8 core Clovertown

Runtime per kernel, relative to CA-GMRES(k,t), for all test matrices, using 8 threads and restart length 60



CA-Krylov Methods Summary and Future Work

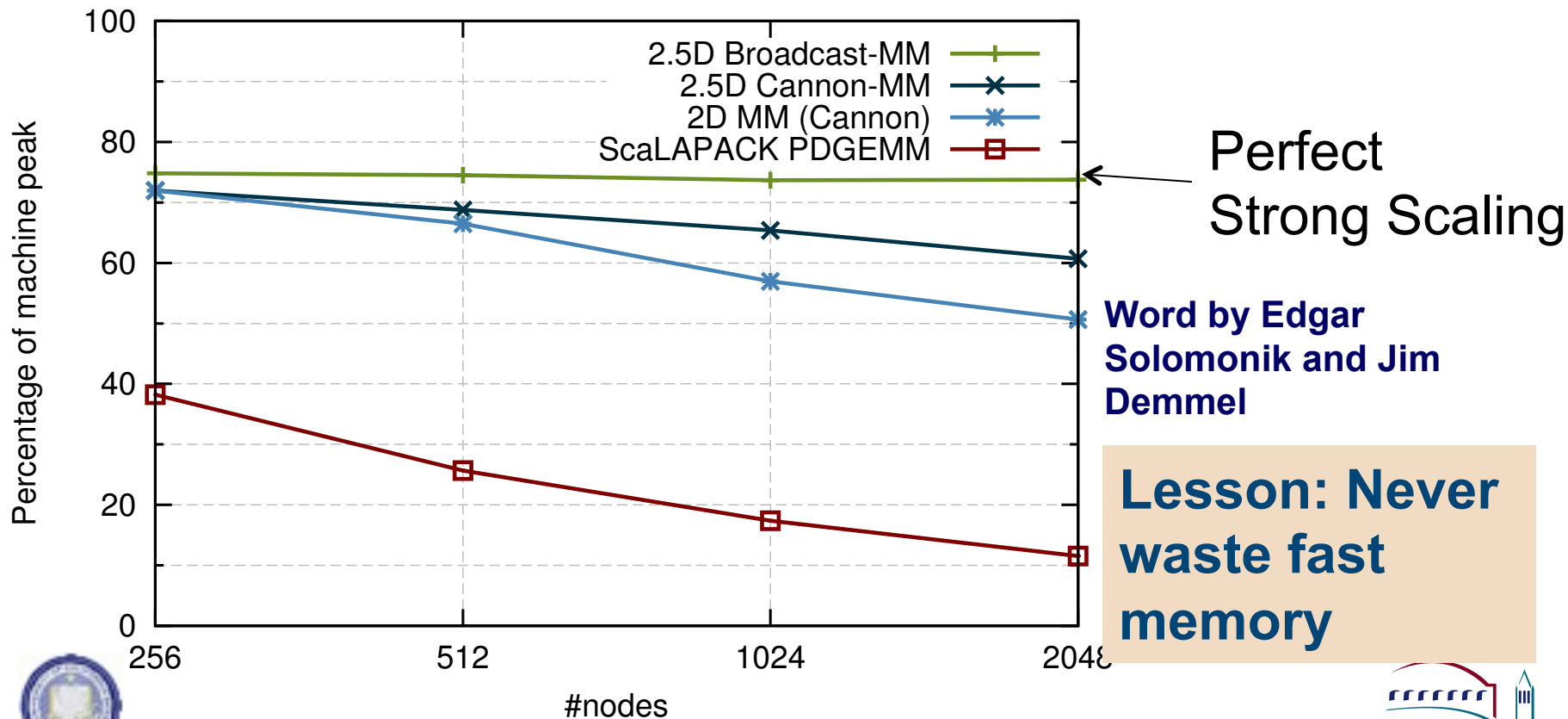
- **The Communication-Avoidance works**
 - Provably optimal
 - Faster in practice
- **Ongoing work for “hard” matrices**
 - Partition poorly (high surface to volume)
 - Idea: separate out dense rows (HSS matrices)
 - [Erin Carson, Nick Knight, Jim Demmel]




Beyond Domain Decomposition: 2.5D Matrix Multiply

- Conventional “2D algorithms” use $P^{1/2} \times P^{1/2}$ mesh and minimal memory
- New “2.5D algorithms” use $(P/c)^{1/2} \times (P/c)^{1/2} \times c^{1/2}$ mesh and c -fold memory
 - Matmul sends $c^{1/2}$ times fewer words – lower bound
 - Matmul sends $c^{3/2}$ times fewer messages – lower bound

2.5D MM on BG/P (n=65,536)



Communication Avoidance in Multigrid: Method of Local Corrections (MLC) for Poisson's Equation

$$\Delta\varphi = \rho \quad , \quad \varphi(\mathbf{x}) = \int_{\Omega} G(\mathbf{x} - \mathbf{y})\rho(\mathbf{y})d\mathbf{y} \quad , \quad \mathbf{x} \in \Omega$$


Real analytic, with rapidly convergent Taylor expansion

MLC uses domain decomposition, plus a noniterative form of multigrid, to represent the solution in a way that minimizes global communication and increases computational intensity.

- Local domains fit in cache; solves computed using FFT and a simplified version of FMM. For $16^3 - 32^3$, this yields ~ 5 flops/byte.
- The (global) coarse problem is small. Communication comparable to single relaxation step.
- Weak scaling: 95% efficiency up to 1024 processors.

Work by Phil Colella et al



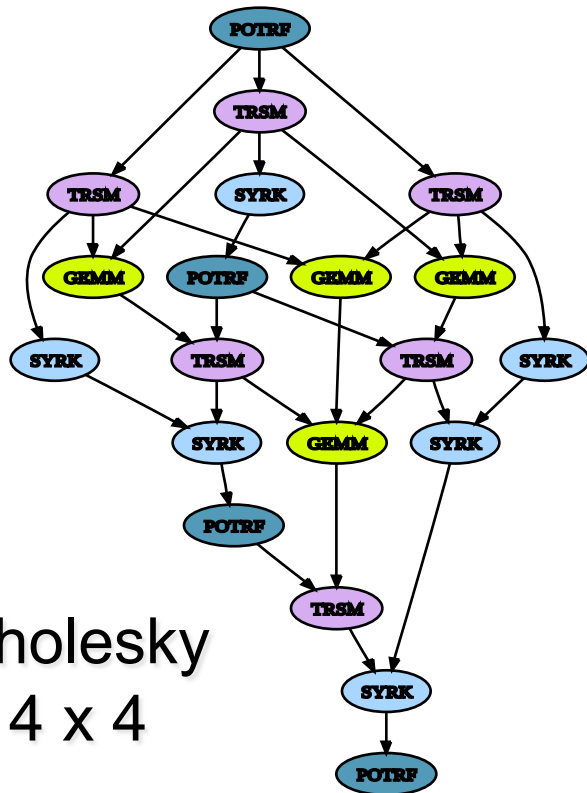
Avoiding Synchronization



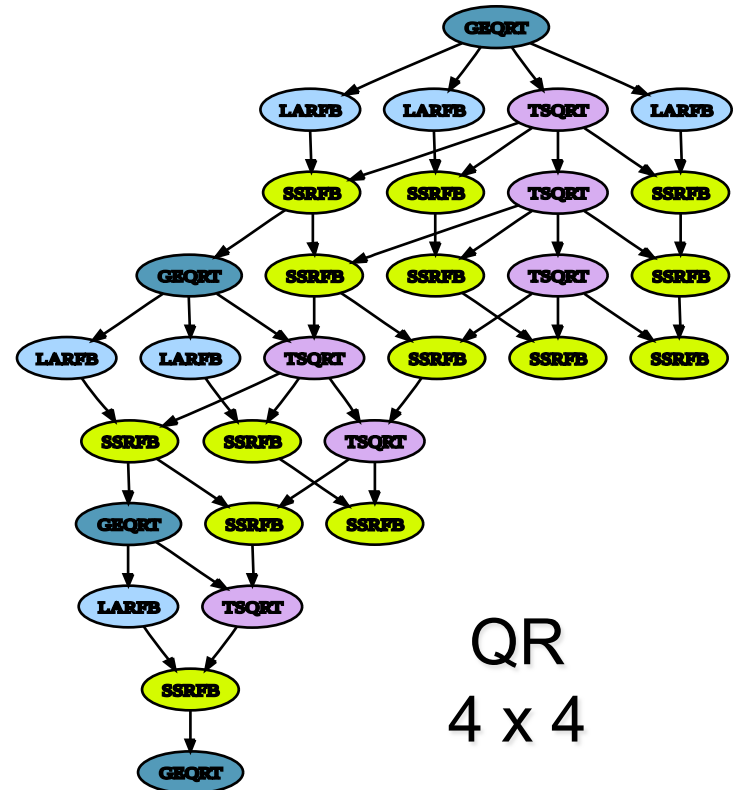
Avoid Synchronization from Applications

Computations as DAGs

View parallel executions as the directed acyclic graph of the computation



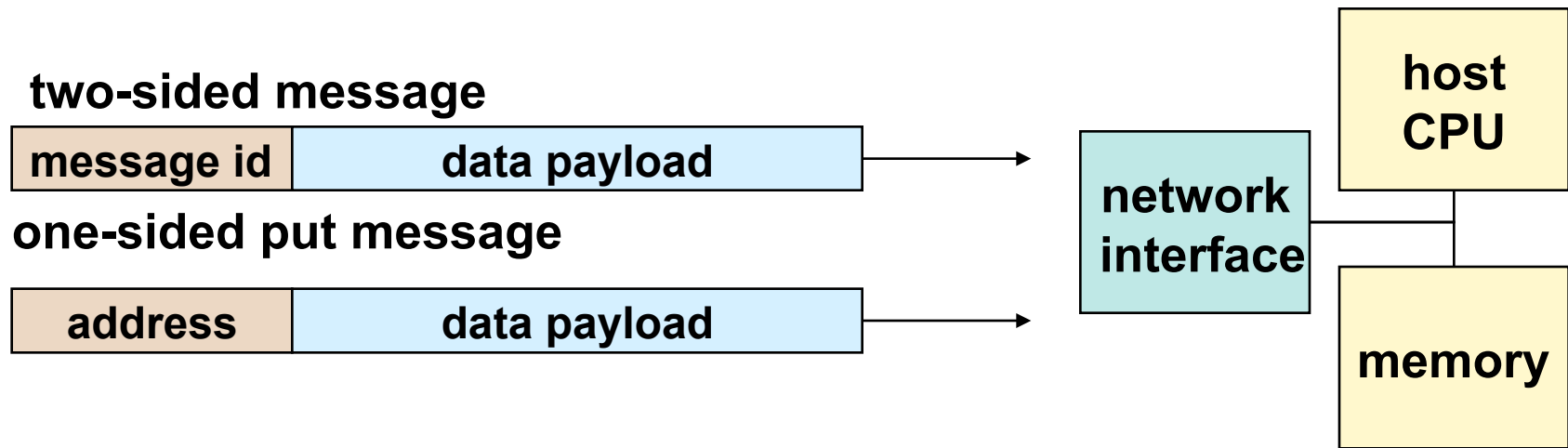
Cholesky
4 x 4



QR
4 x 4



Avoiding Synchronization in Communication

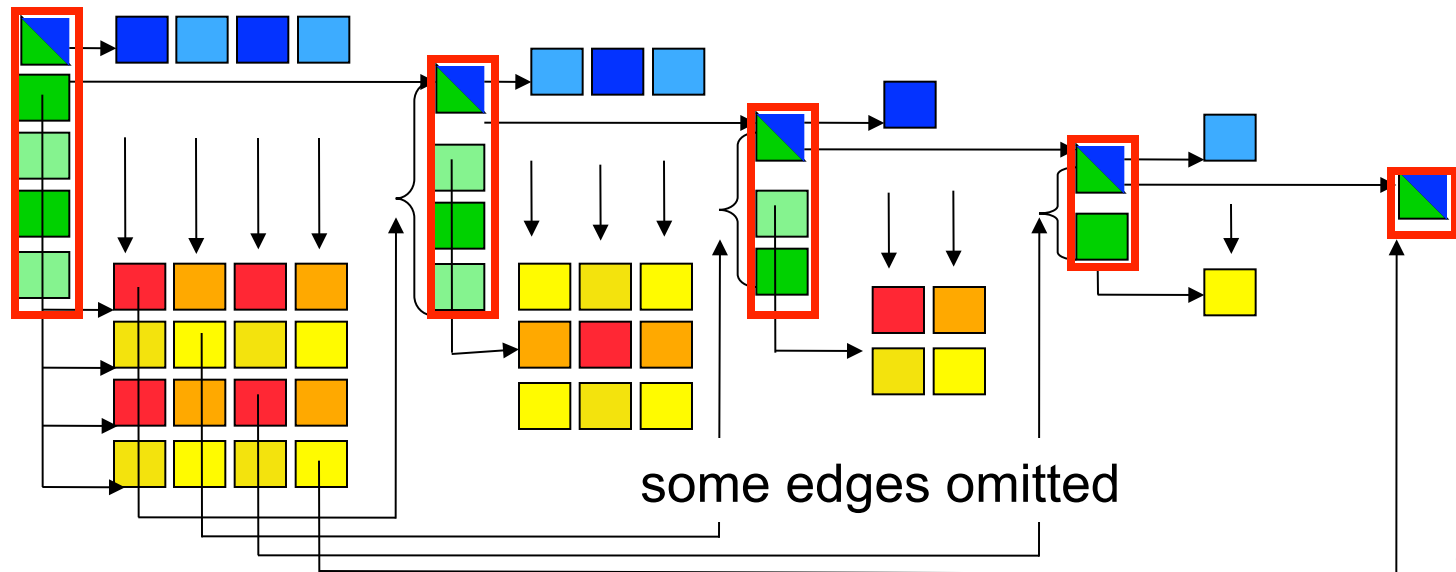


- **Two-sided message passing (e.g., MPI) requires matching a send with a receive to identify memory address to put data**
 - Wildly popular in HPC, but cumbersome in some applications
 - Couples data transfer with synchronization
- **Using global address space decouples synchronization**
 - Pay for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



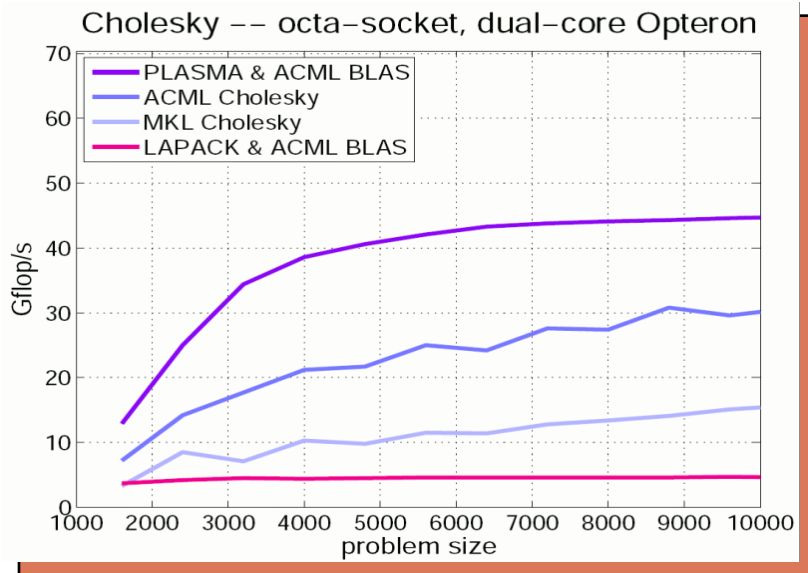
Event Driven LU in UPC

- Assignment of work is static; schedule is dynamic
- Ordering needs to be imposed on the schedule
 - Critical path operation: Panel Factorization
- General issue: dynamic scheduling in partitioned memory
 - Can deadlock in memory allocation
 - “memory constrained” lookahead

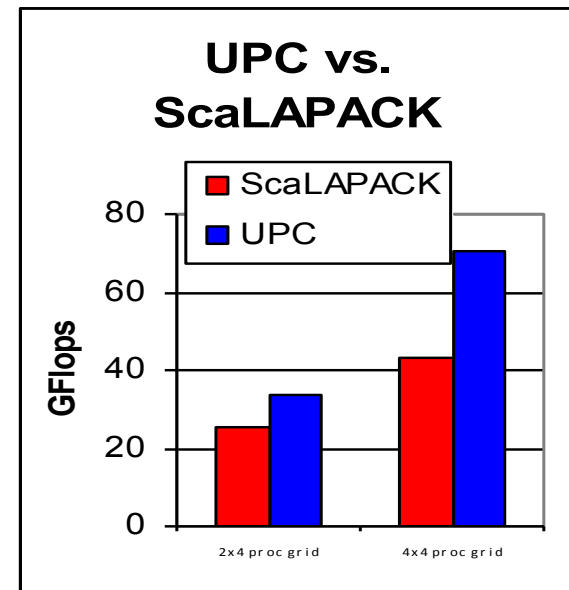


DAG Scheduling Outperforms Bulk-Synchronous Style

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



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

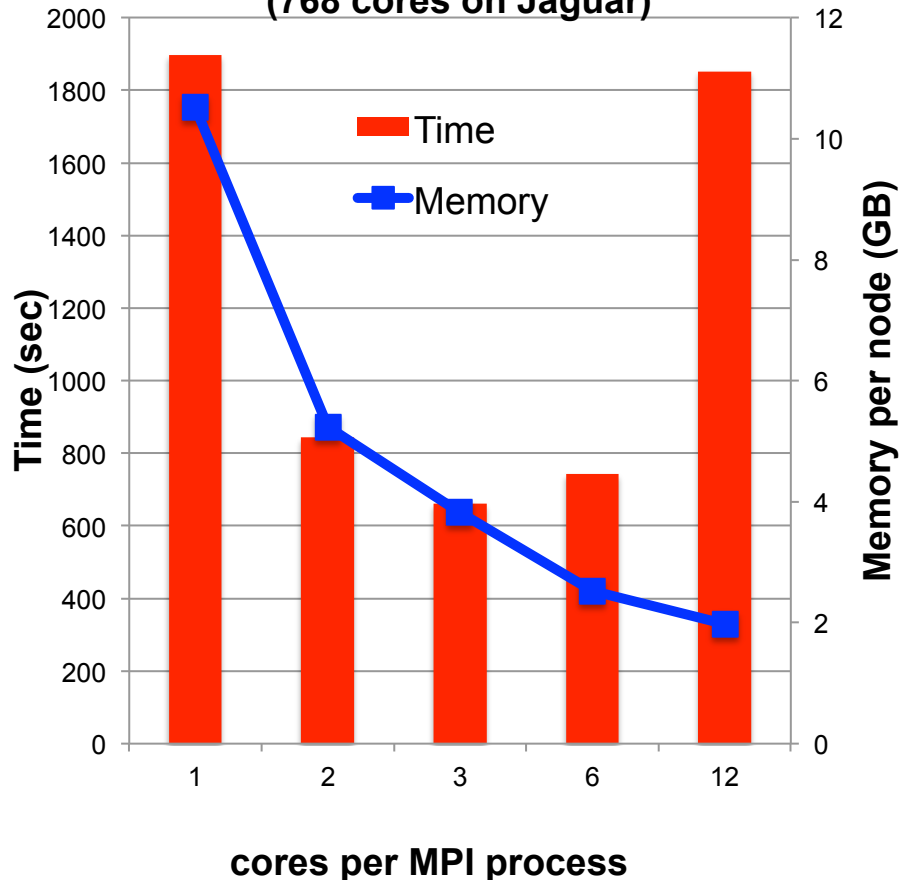
- **Locality is important:**
 - When memory hierarchies are deep
 - When computational intensity is low
- **Most (all?) successful examples of locality-important applications/machines use static scheduling**
 - Unless they have a irregular/dynamic task graph
- **Two extremes are well-studied**
 - Dynamic parallelism without locality
 - Static parallelism (with threads = processors) with locality
- **Dynamic scheduling and locality control don't mix**
 - 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



Use Two Programming Models to Match Machine

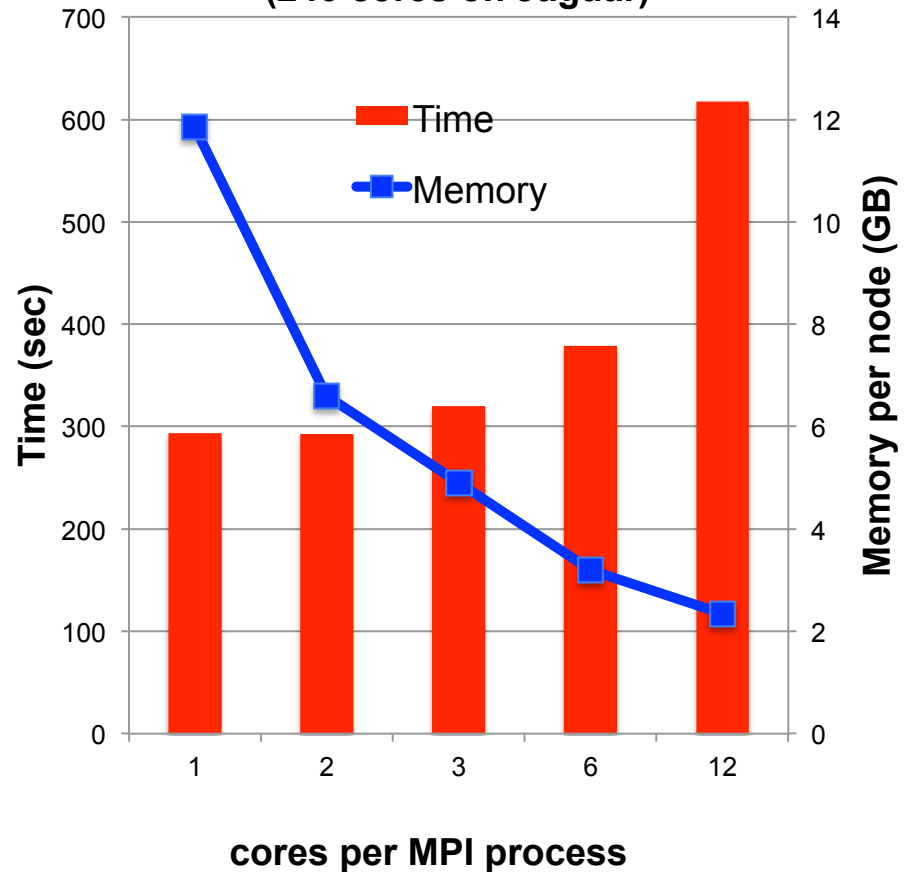
PARATEC

(768 cores on Jaguar)



fvCAM

(240 cores on Jaguar)



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



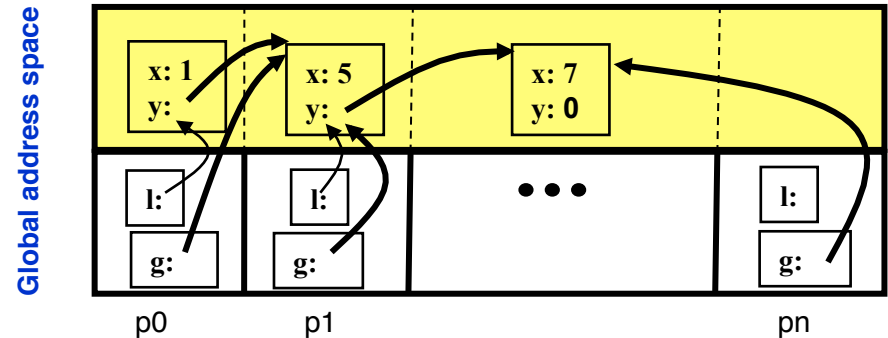
Getting the Best of Each

PGAS for locality and convenience

Global address space: directly read/write remote data

Partitioned: data is designated as local or global

- Affinity control
- Scalability
- Never say “receive”



Debate is around the control model:

- Dynamic thread creation vs. SPMD
- Hierarchical SPMD compromise
 - “Think parallel” and group as needed



Hierarchical SPMD in Titanium

- Thread teams may execute distinct tasks

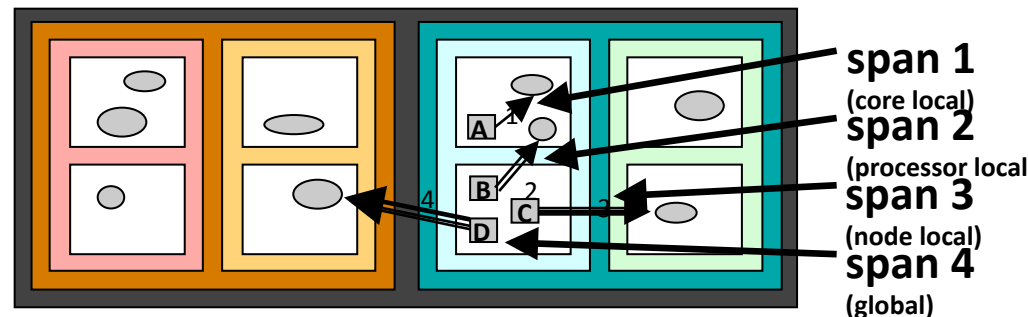
```
partition(T) {  
    { model_fluid(); }  
    { model_muscles(); }  
    { model_electrical(); }  
}
```

- Hierarchy for machine / tasks

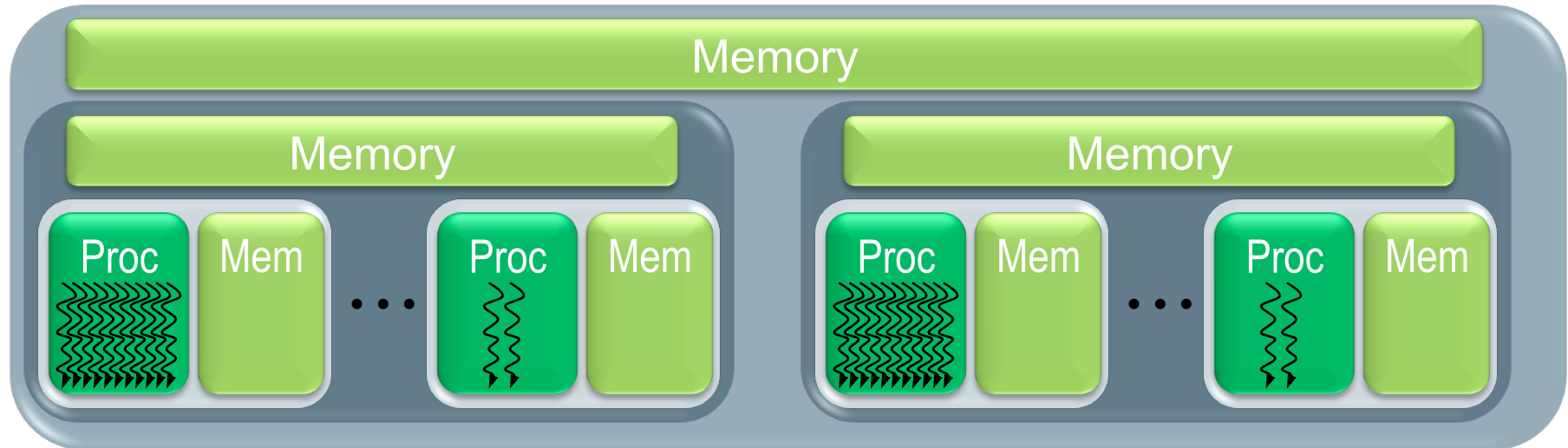
- Nearby: access shared data
- Far away: copy data

- Advantages:

- Provable pointer
- Mixed data / task style
- Lexical scope prevents some deadlocks



Hierarchical 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



Stepping Back

- **Communication avoidance as old as tiling**
- **Communication optimality as old as Hong/Kung**
- **What's new?**
 - Raising the level of abstraction at which we optimize
 - BLAS2 → BLAS3 → LU or SPMV/DOT → Krylov
 - Changing numerics in non-trivial ways
 - Rethinking methods to models
- **Communication and synchronization avoidance**
- **Software engineering: breaking abstraction**
- **Compilers: inter-procedural optimizations**



Exascale Views

- **“Exascale” is about continuing growth in computing performance for science**
 - Energy efficiency is key
 - Job size is irrelevant
- **Success means:**
 - Influencing market: HPC, technical computing, clouds, general purpose
 - Getting more science from data and computing
- **Failure means:**
 - few big machines for a few big applications
- **Not all computing problems are exascale, but they should all be exascale-technology aware**

Thank You!

