

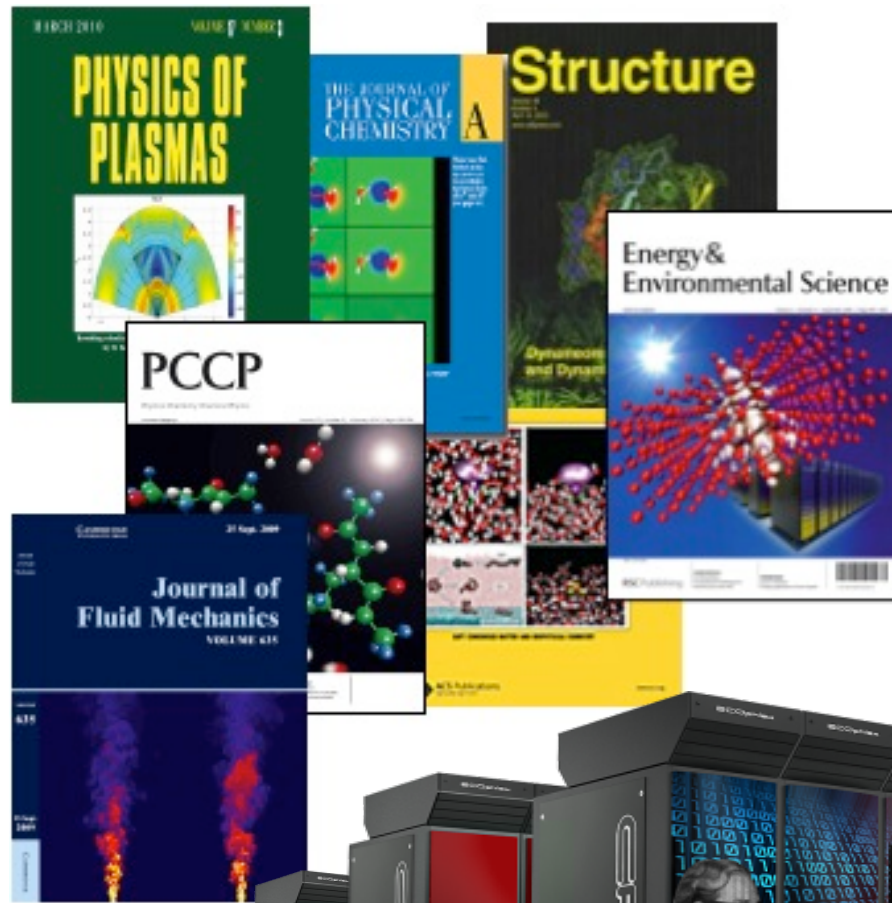
# Exascale Computing: More and Moore?

**Kathy Yelick**

**Associate Laboratory Director and NERSC Director  
Lawrence Berkeley National Laboratory**

**EECS Professor, UC Berkeley**

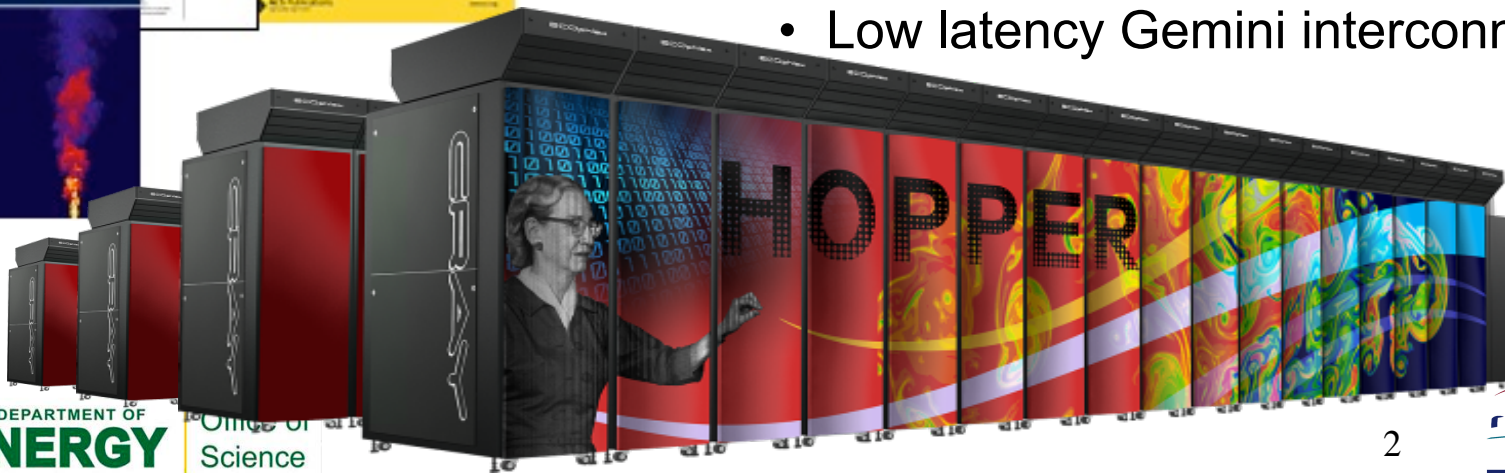
# NERSC Overview



- NERSC represents science needs
- 4000 users, 500 projects, 700 code instances
  - Over 1,500 publications annually
  - Time is used by university researchers (65%), DOE Labs (25%) and others

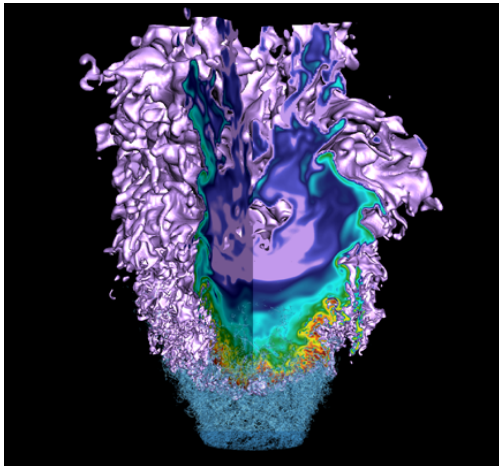
## Petaflop Hopper system

- High application performance
- Nodes: 2 12-core AMD processors
- Low latency Gemini interconnect

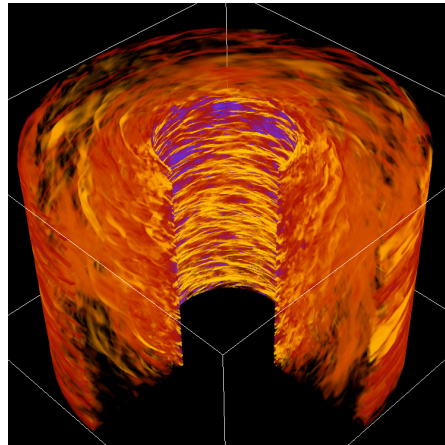




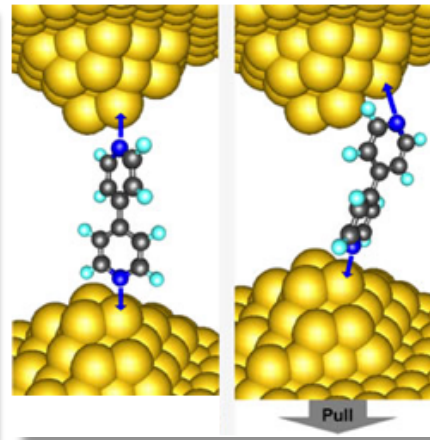
# Energy Science at NERSC



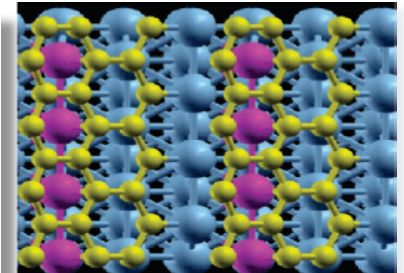
**Combustion:** New algorithms (AMR) coupled to experiments



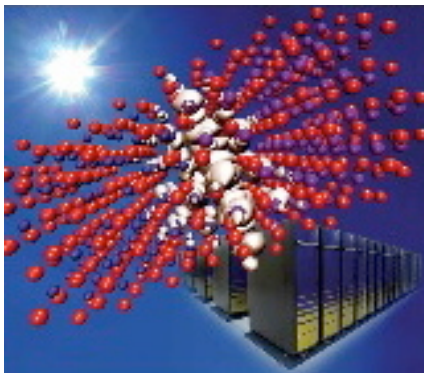
**Fusion:** Simulations of Fusion devices at ITER scale



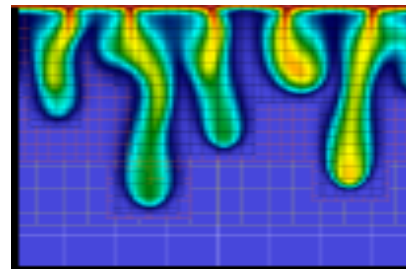
**Nano devices:** New single molecule switching element



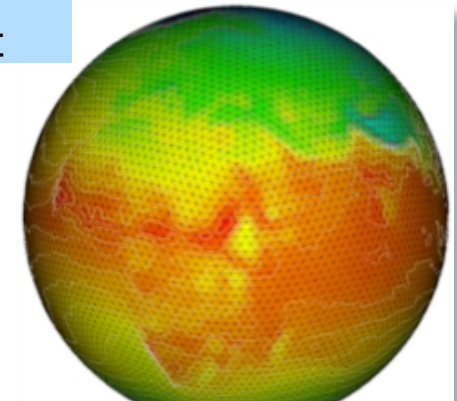
**Energy storage:** Catalysis for improved batteries and fuel cells



**Materials:** For solar panels and other applications.



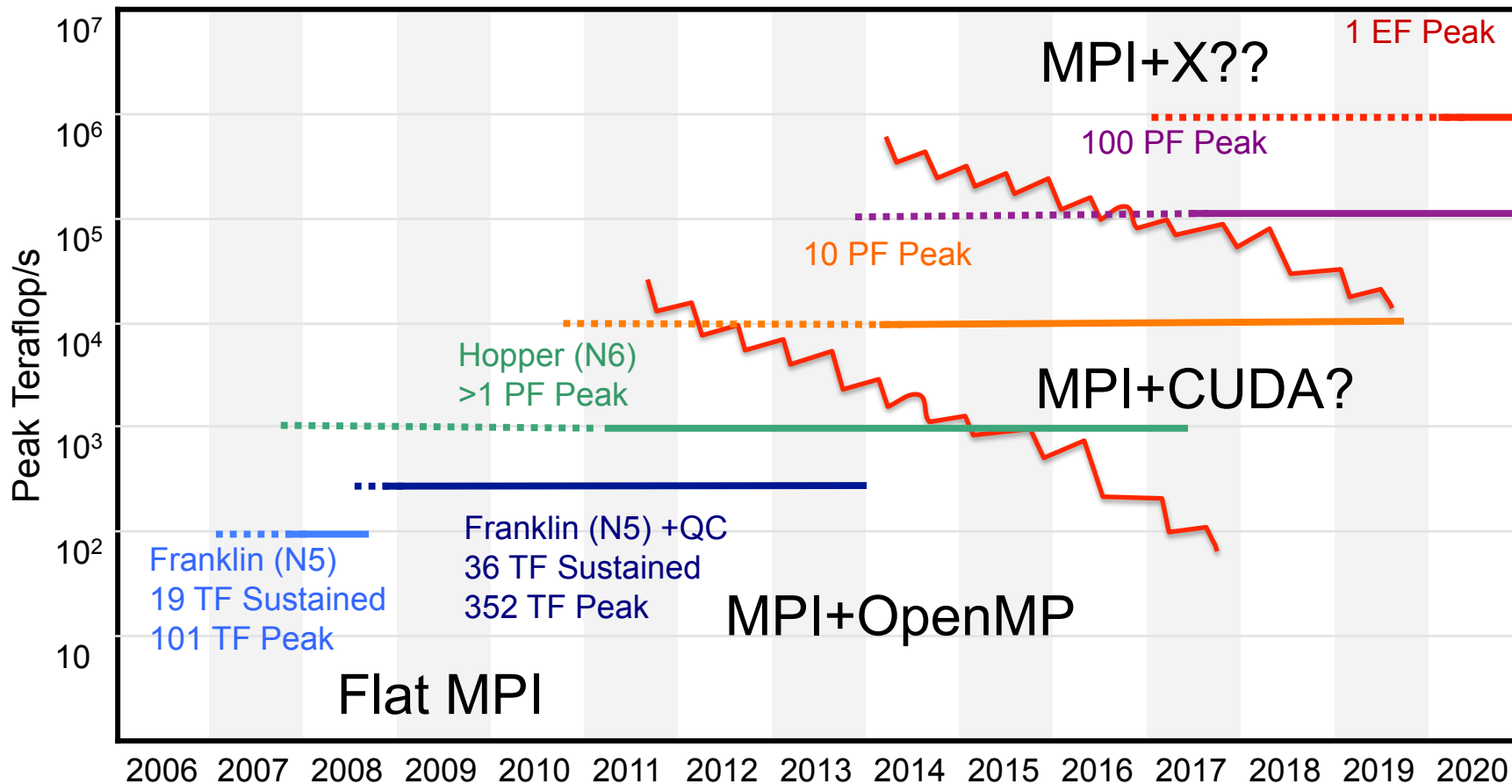
**Capture & Sequestration:** EFRCs



**Climate modeling:** Work with users on scalability of cloud-resolving models



# How and When to Move Users



Want to avoid two paradigm disruptions on road to Exa-scale

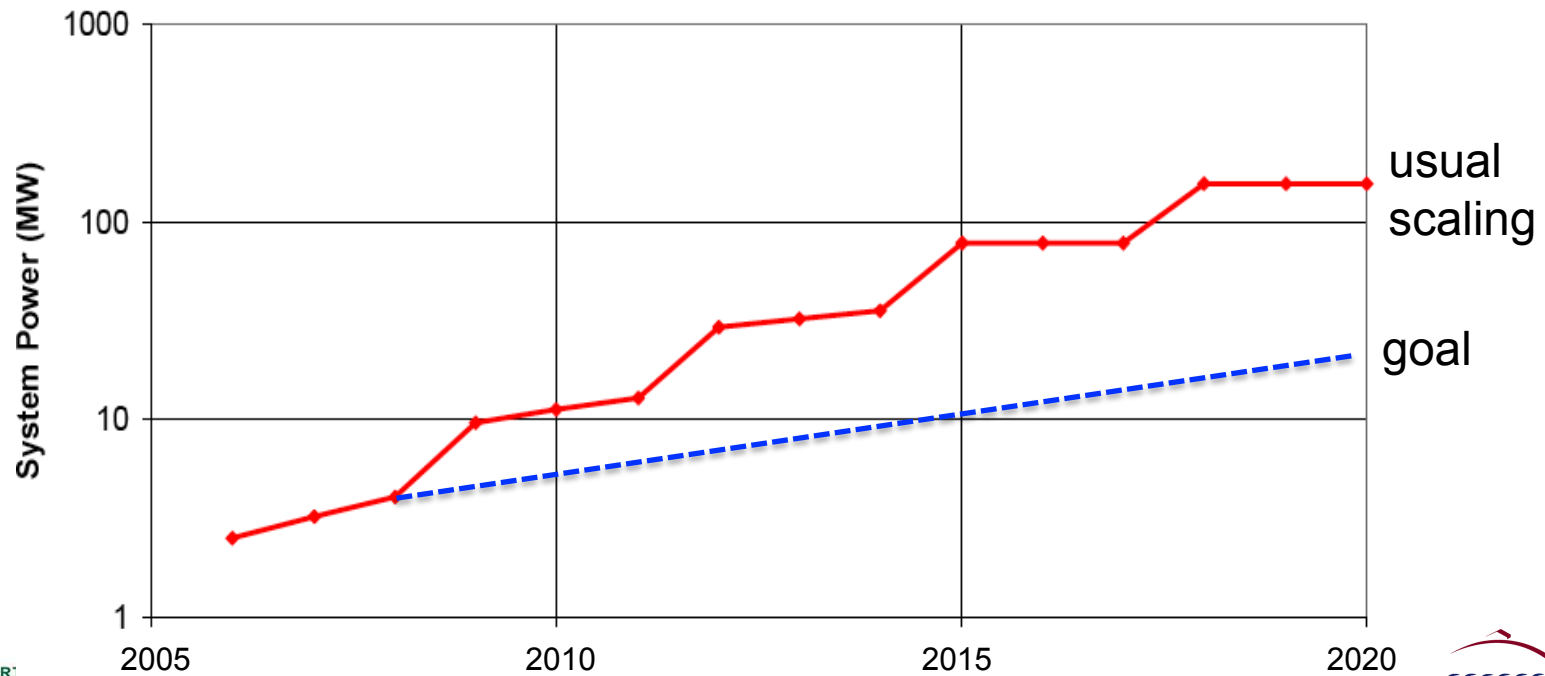




# Exascale is about Energy Efficient Computing

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

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





# Energy Efficiency of Computing is a Global Problem

## Worldwide IT Footprints

Emissions by sub-sector, 2020

820m tons CO<sub>2</sub>

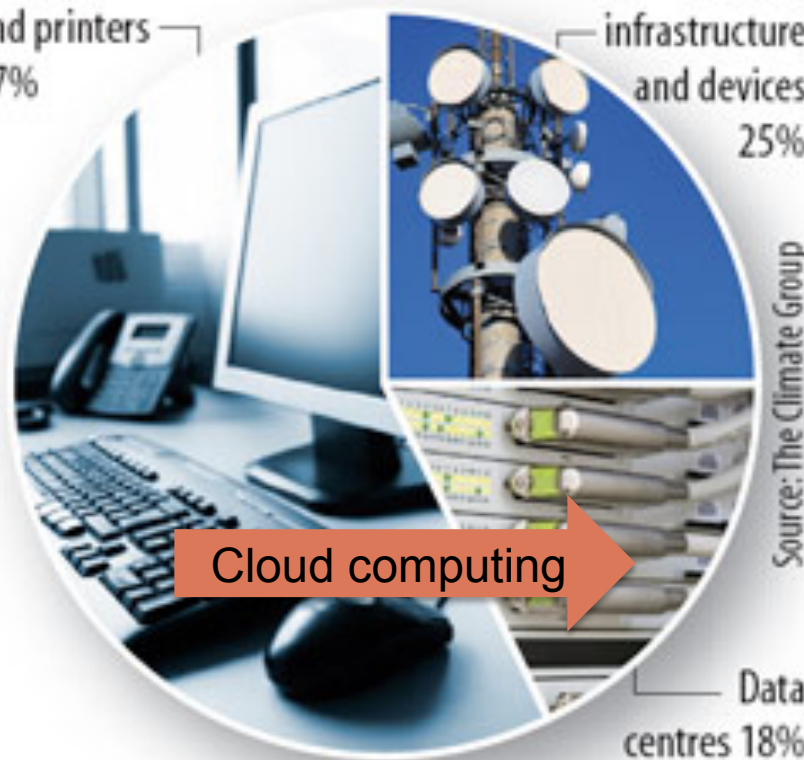
PCs, peripherals and printers 57%

Telecoms infrastructure and devices 25%

360m tons CO<sub>2</sub>

2007 Worldwide IT carbon footprint: 2% = 1.43 billion tons CO<sub>2</sub>, comparable to the global aviation industry

Expected to grow to 4% by 2020

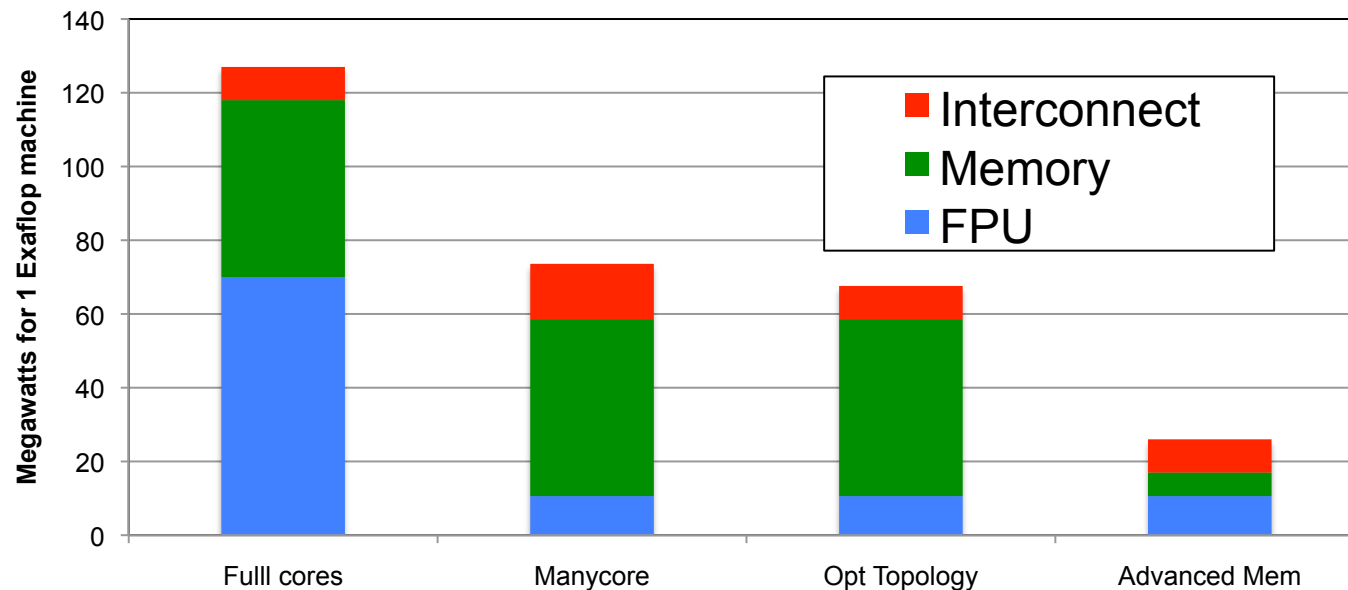


260m tons CO<sub>2</sub>

Total emissions: 1.43bn tonnes CO<sub>2</sub> equivalent



# 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)

# 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 an open question
- 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*



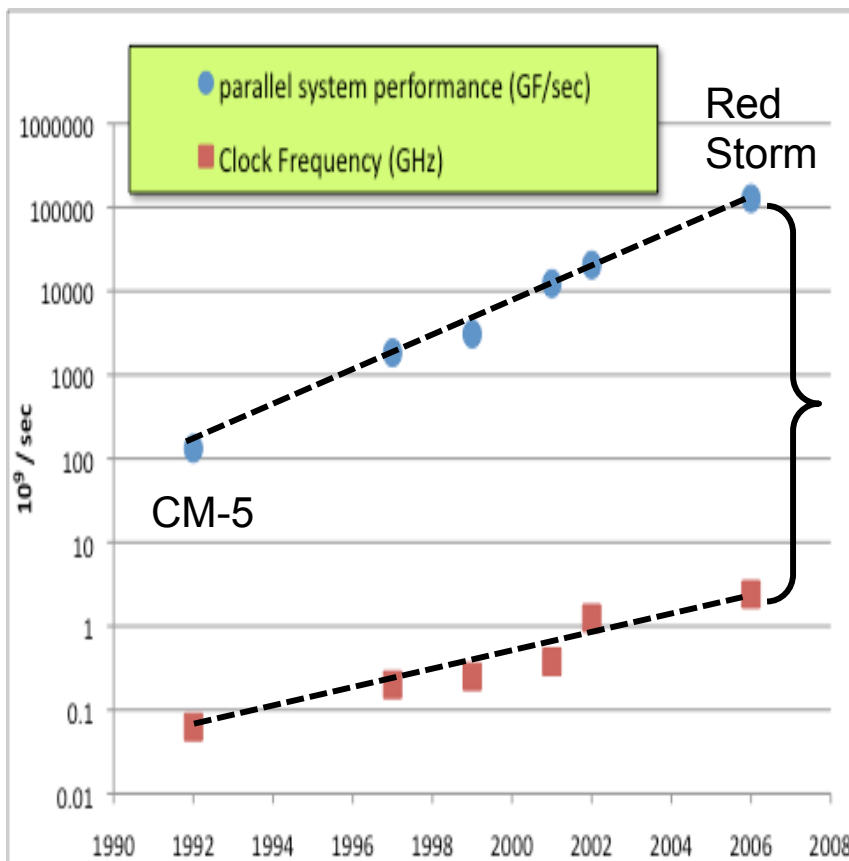


# Anticipating and Influencing the Future

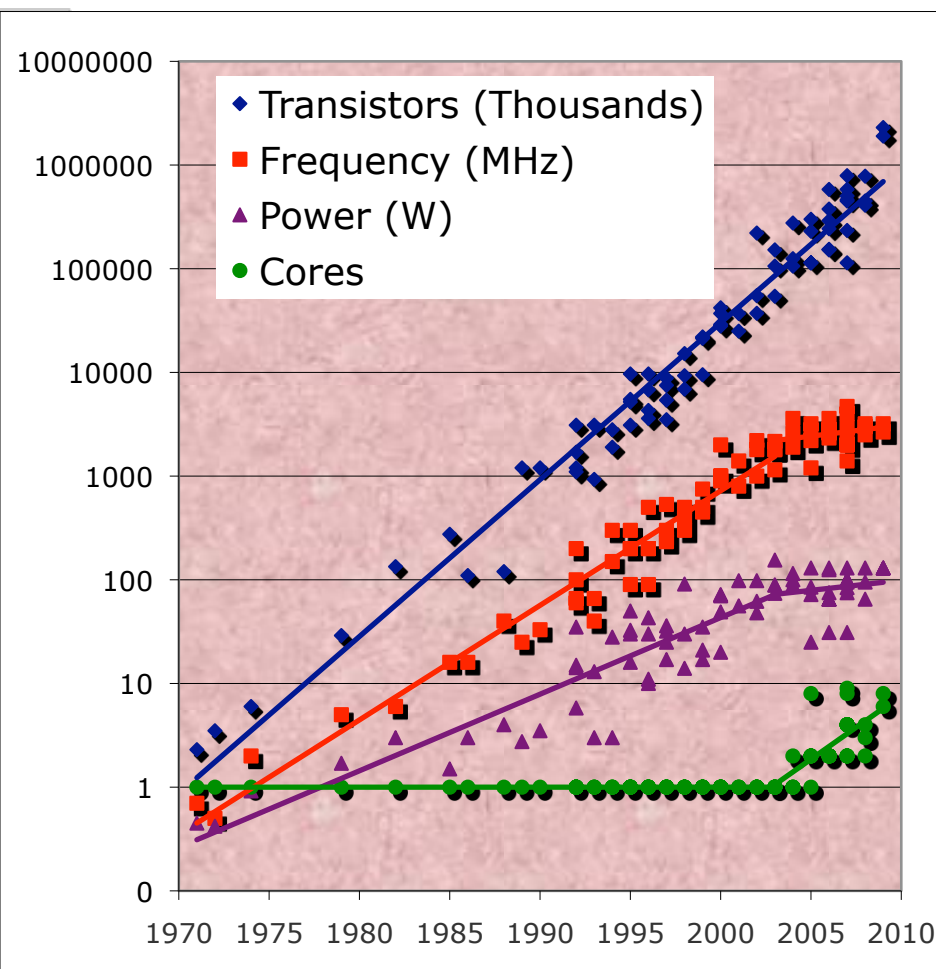
## Hardware Design



# Moore's Law Continues, but Only with Added Concurrency



1000x performance increase was  
40x clock speed x 25x concurrency



Can you double concurrency every 2 years?



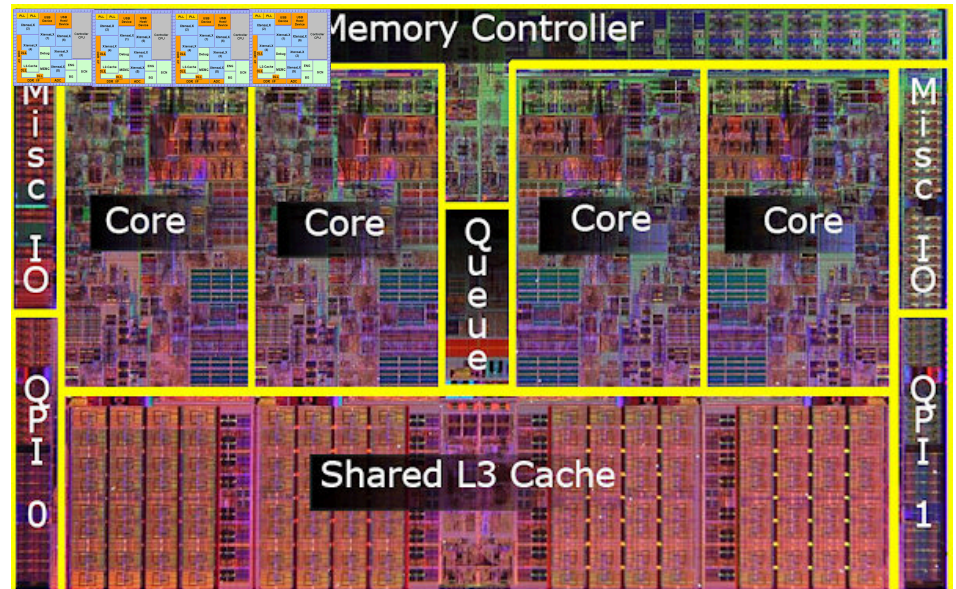
# Manycore/Embedded Approach

	Intel QC Nehalem	Tensilica	Overall Gain
Power (W)	100	.1	$10^3$
Area (mm <sup>2</sup> )	240	2	$10^2$
DP Gflops	50	4	.1
<b>Overall</b>			$10^4$

Lightweight (thin) cores improve energy efficiency

## Tensilica Xtensa with double-precision

- 2mm<sup>2</sup> chip surface area
- 0.1 watts
- 4GFLOPs

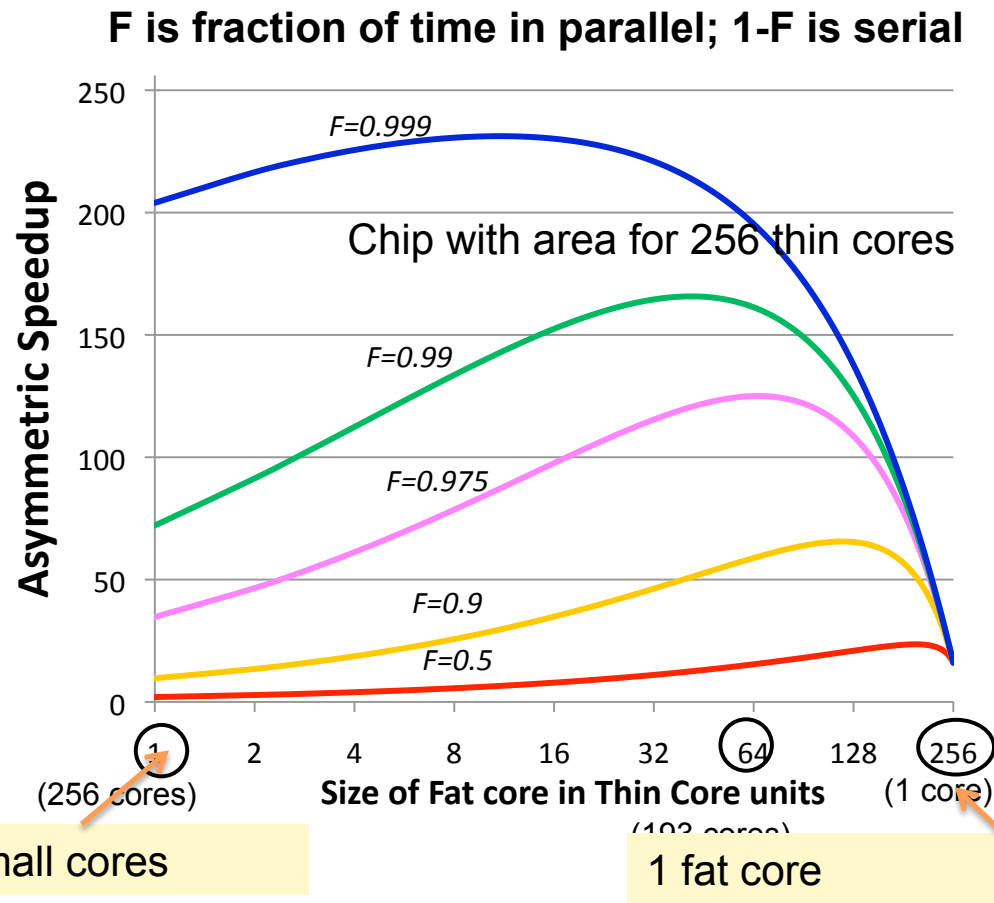


## Intel Quad Core Nehalem

- 240mm<sup>2</sup> chip surface area
- 100 watts TDP
- 50 GFLOPs



# 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

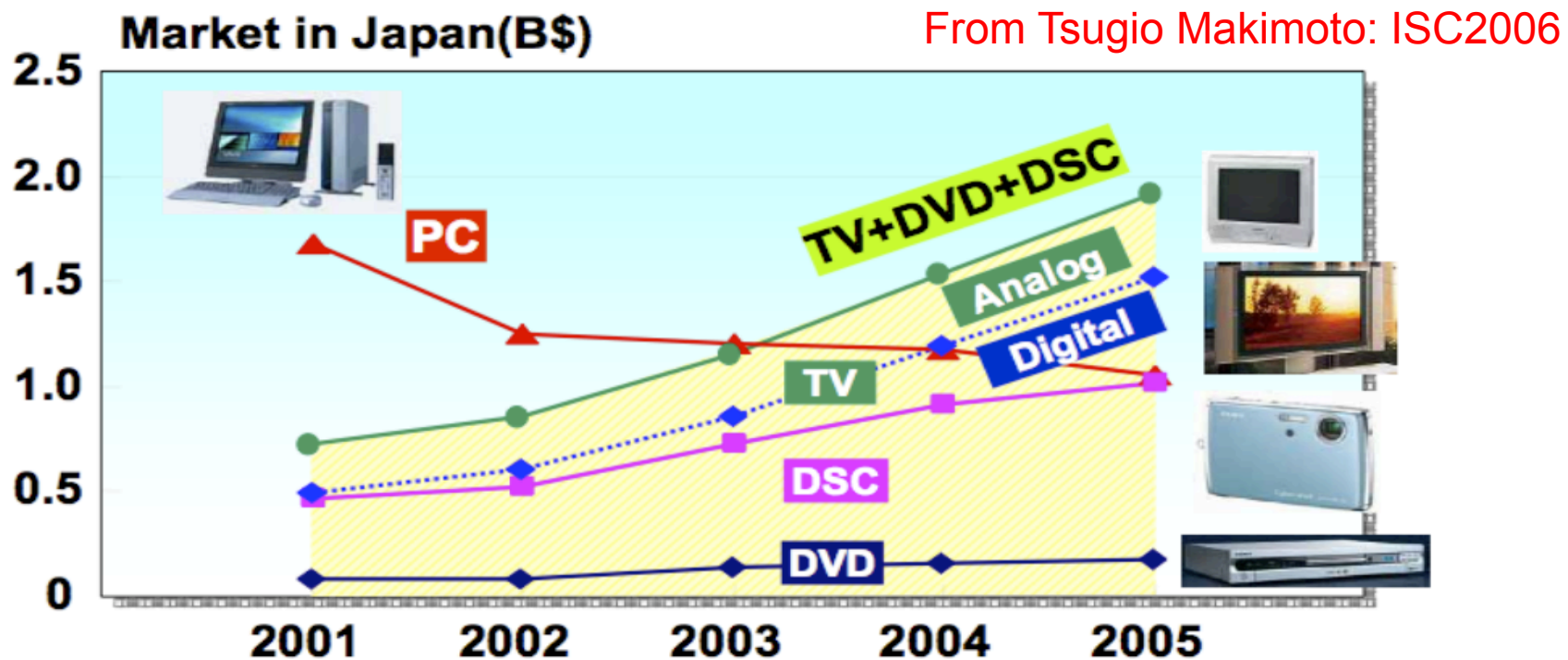
# Technology Investment Trends

**1990s: Computing R&D dominated by desktop/COTS**

- Learned to use COTS technology for HPC

**2010s: Computing R&D moving to consumer electronics**

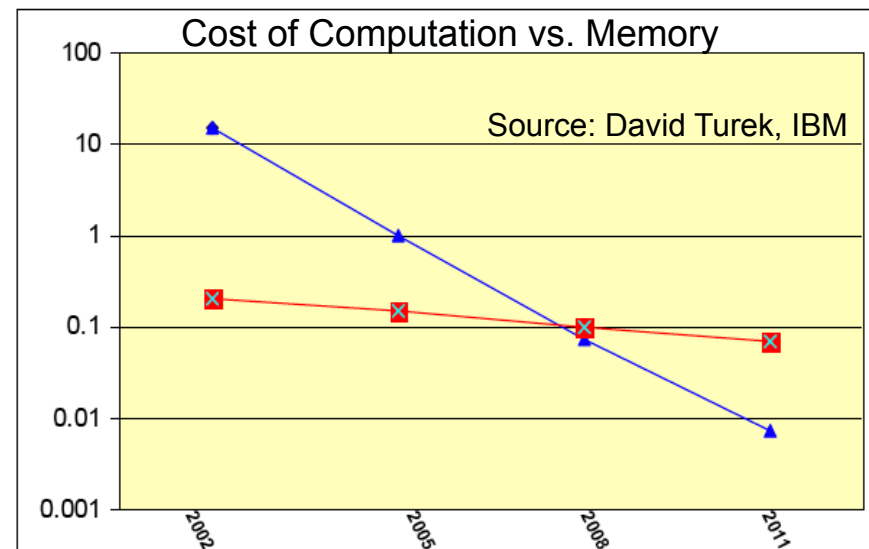
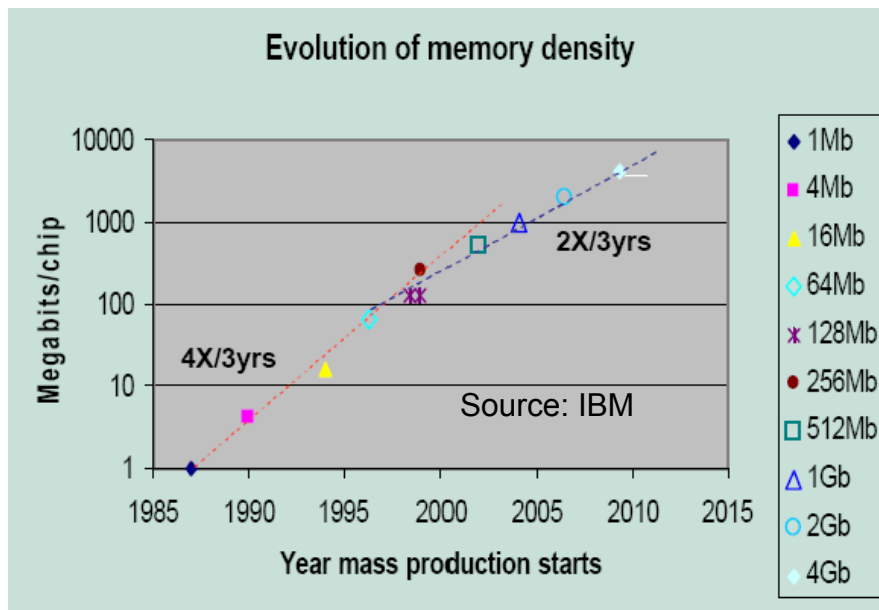
- Need to leverage embedded/consumer technology for HPC



# Memory is Not Keeping Pace

**Technology trends against a constant or increasing memory per core**

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

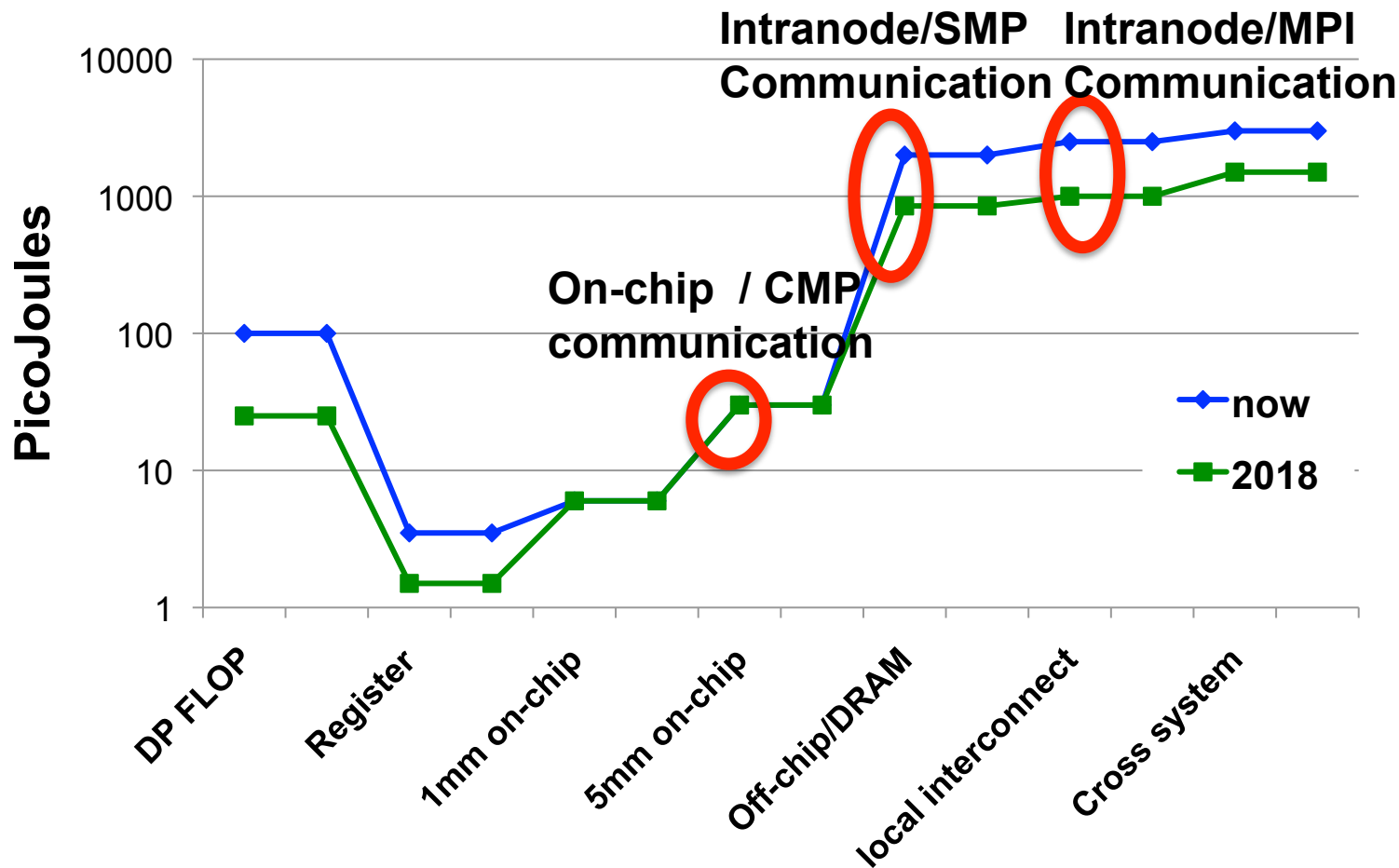


The cost to sense, collect, generate and calculate data is declining much faster than the cost to access, manage and store it

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



# Where does the Power Go?

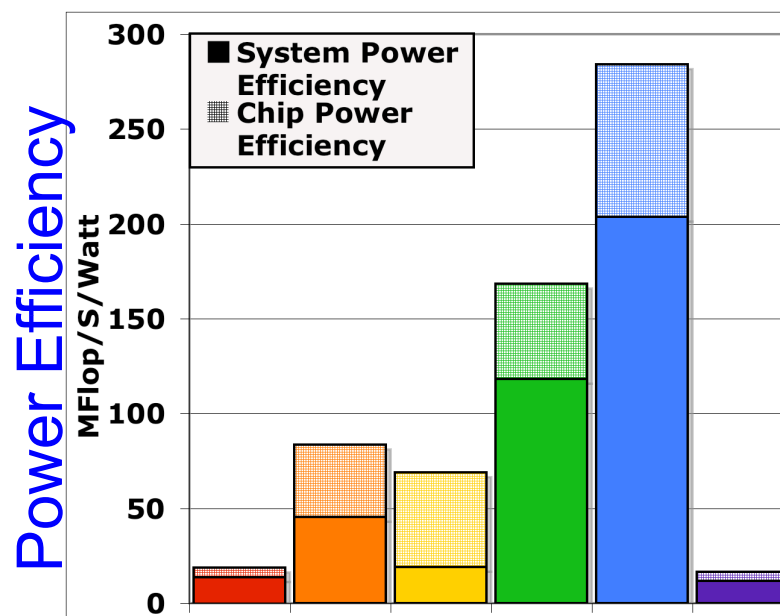
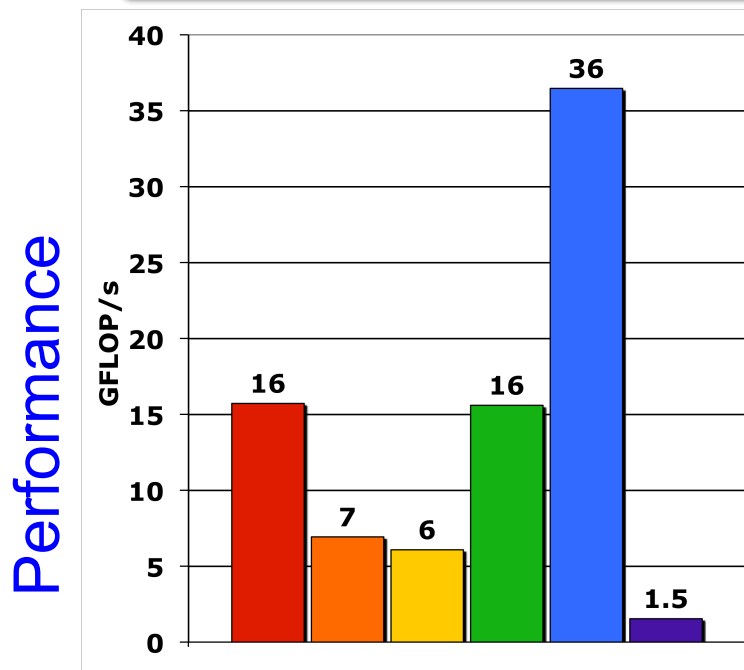




# Energy Efficiency of Applications (Includes Cell and GPU)

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

1.7x speedup versus optimized Nehalem (C2050 w/ECC)



## Cache-based

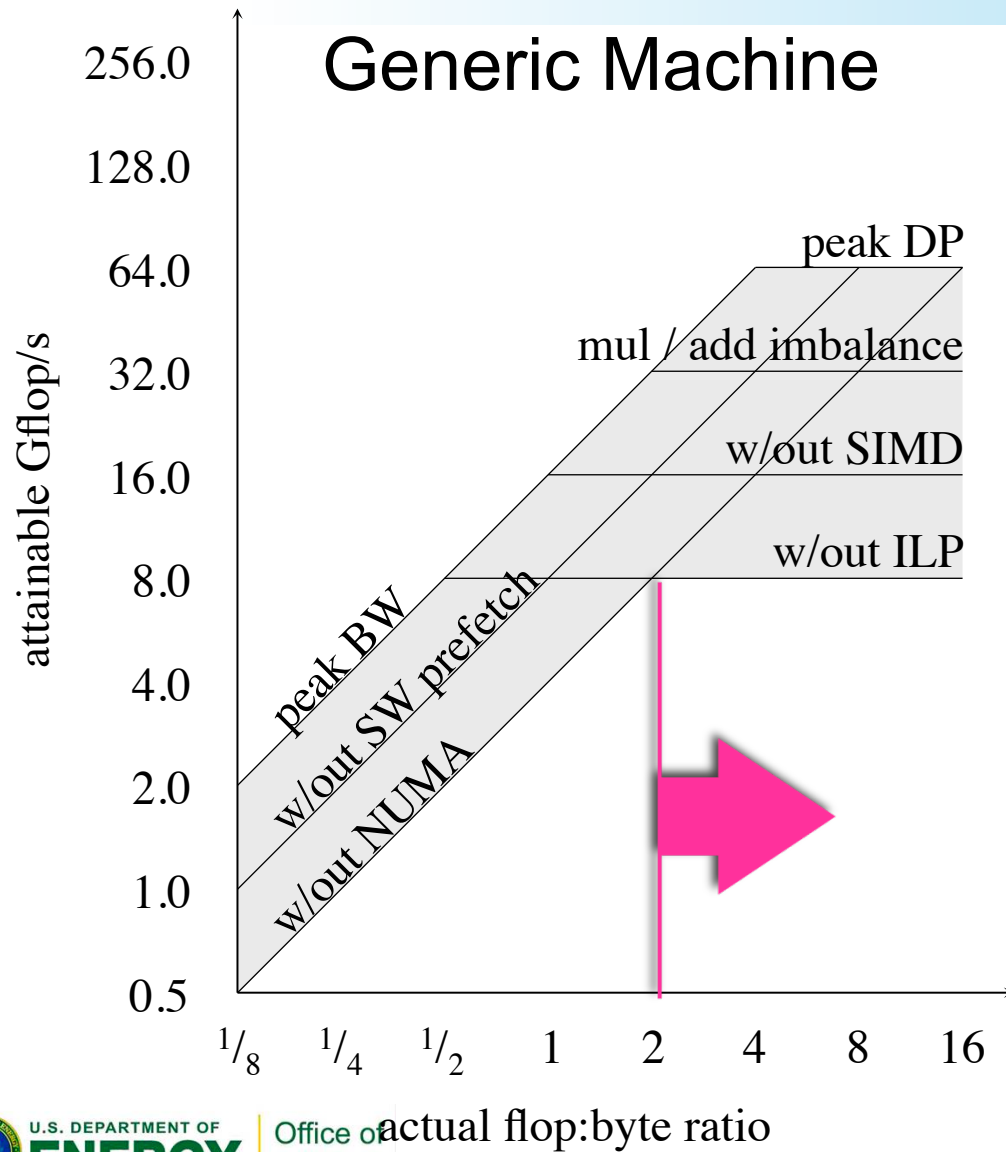
- Gainestown
- Barcelona
- Victoria Falls

## Local store-based

- Cell Blade
- GTX280
- GTX280-Host



# The Roofline Performance Model

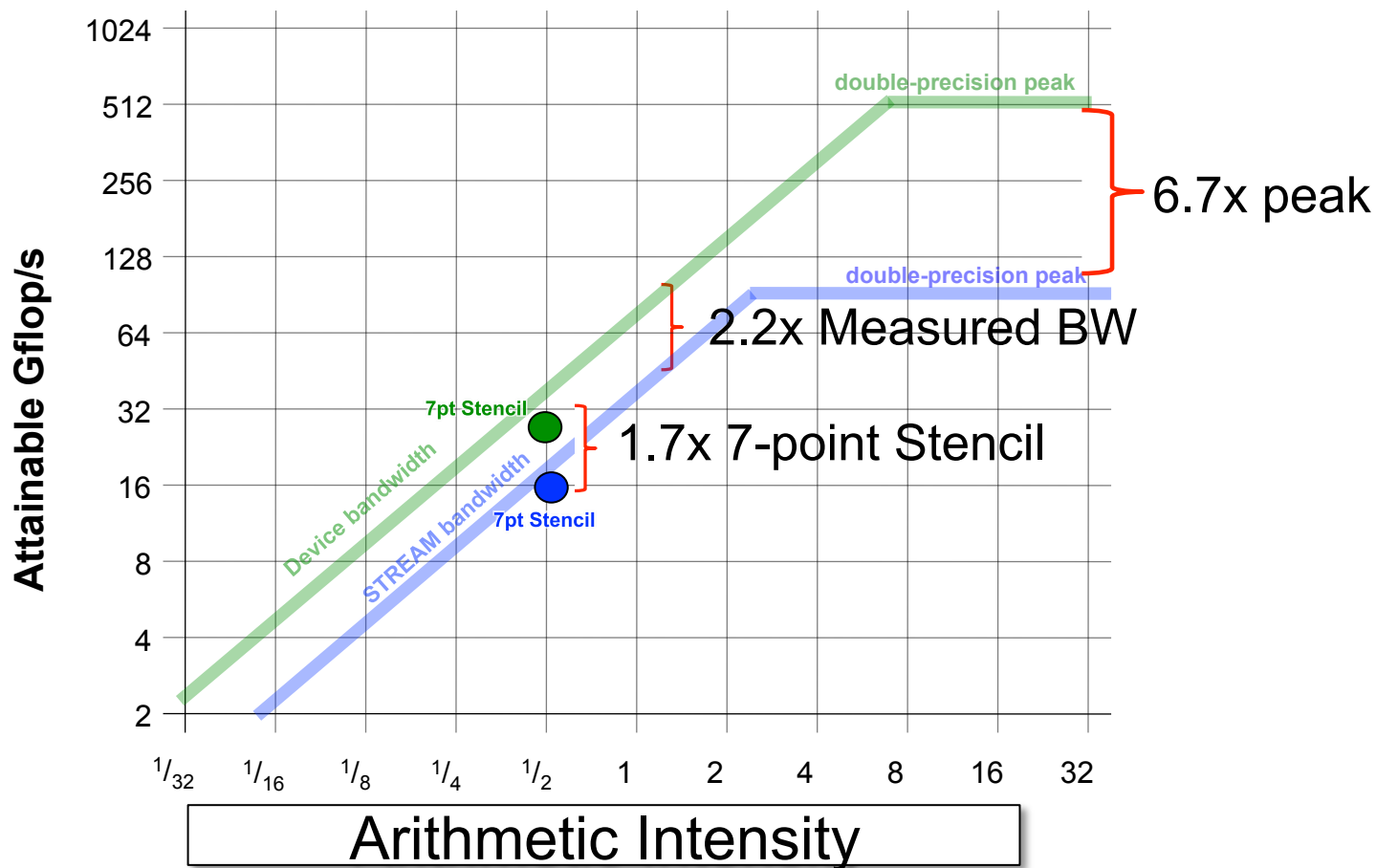


- ❖ The flat room is determined by arithmetic peak and instruction mix
- ❖ The sloped part of the roof is determined by peak DRAM bandwidth (STREAM)
- ❖ X-axis is the computational intensity of your computation



# Relative Performance Expectations

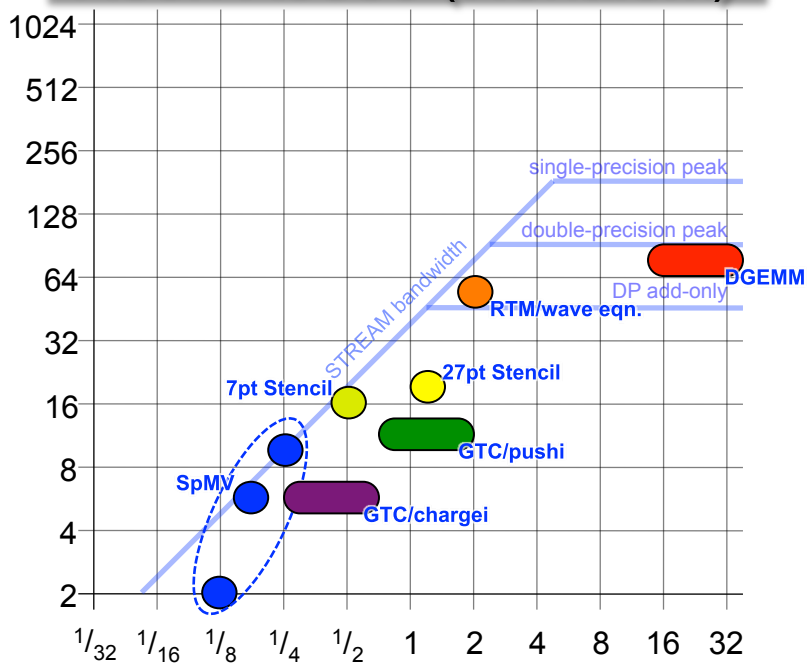
## Fermi & Nehalem Roofline



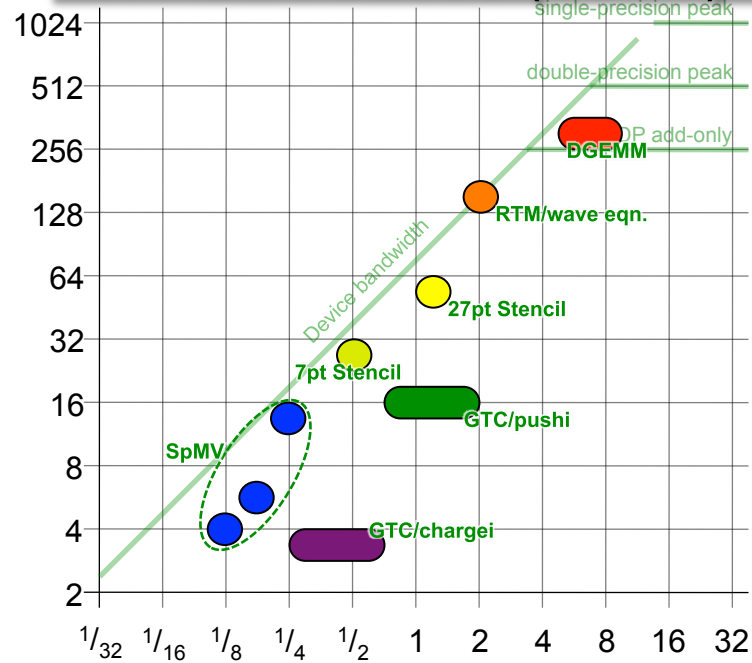


# Relative Performance Across Kernels

## Xeon X5550 (Nehalem)



## NVIDIA C2050 (Fermi)





# 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 Programming Models



# Open Problems in Software

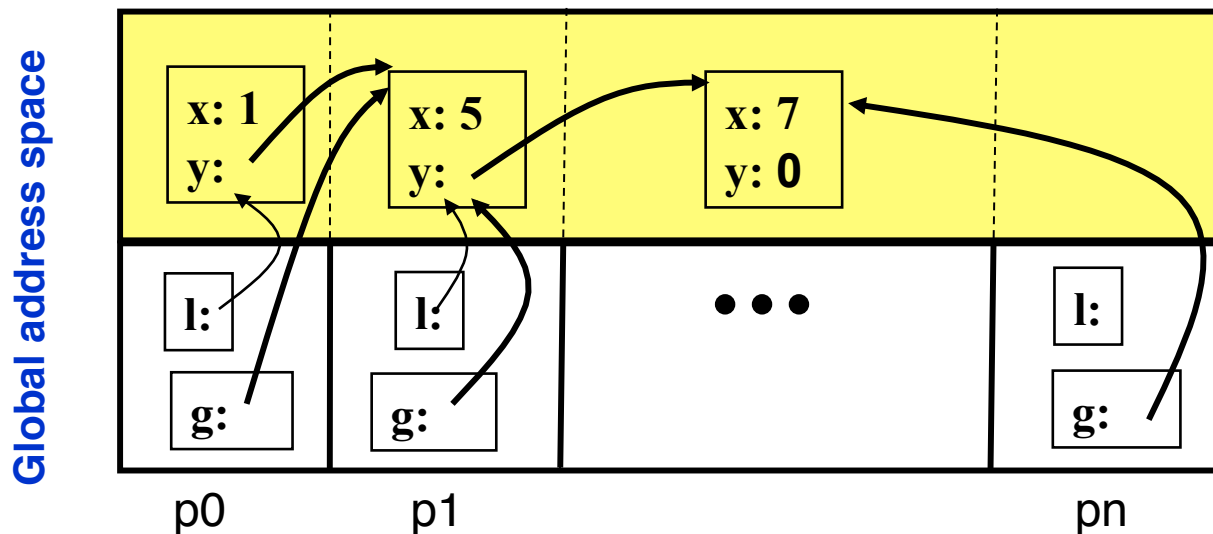
- **Goal: performance through parallelism**
- **Locality is equally important**
- **Heroic compilers unlikely solution:**
- **Need better programming models that:**
  - **Abstract machine variations**
  - **Provide for control over what is important**
- **Data movement (“communication”) dominates running time and power**



# Partitioned Global Address Space Languages

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

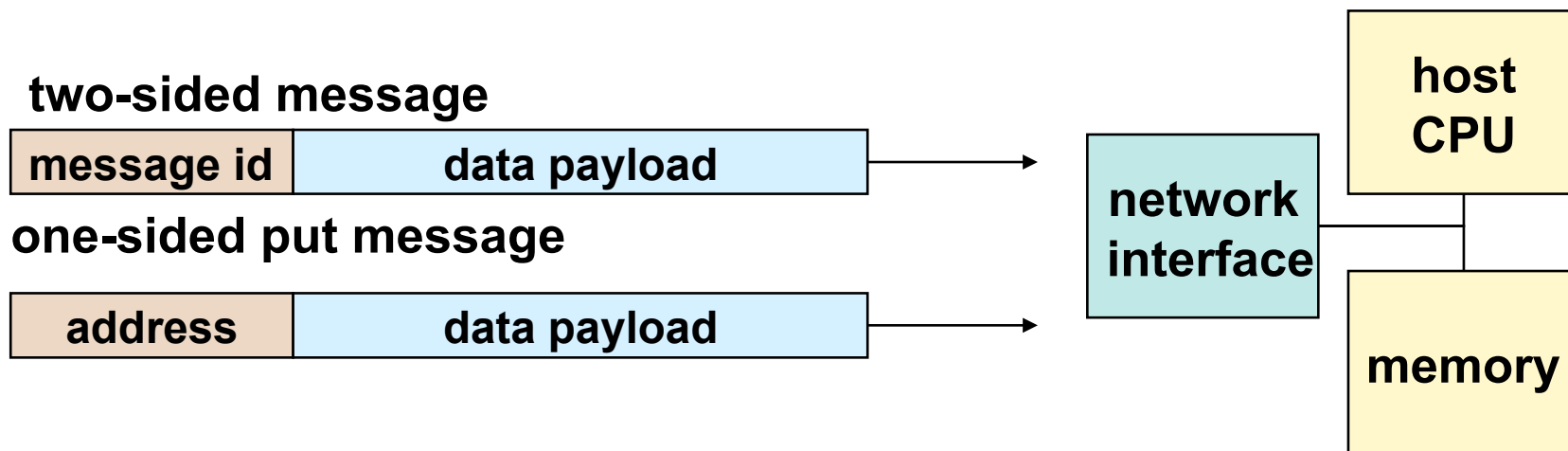
**Partitioned:** data is designated as local or global



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



# Two-sided vs One-sided 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



U.S. DEPARTMENT OF  
**ENERGY**

Office of  
Science

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

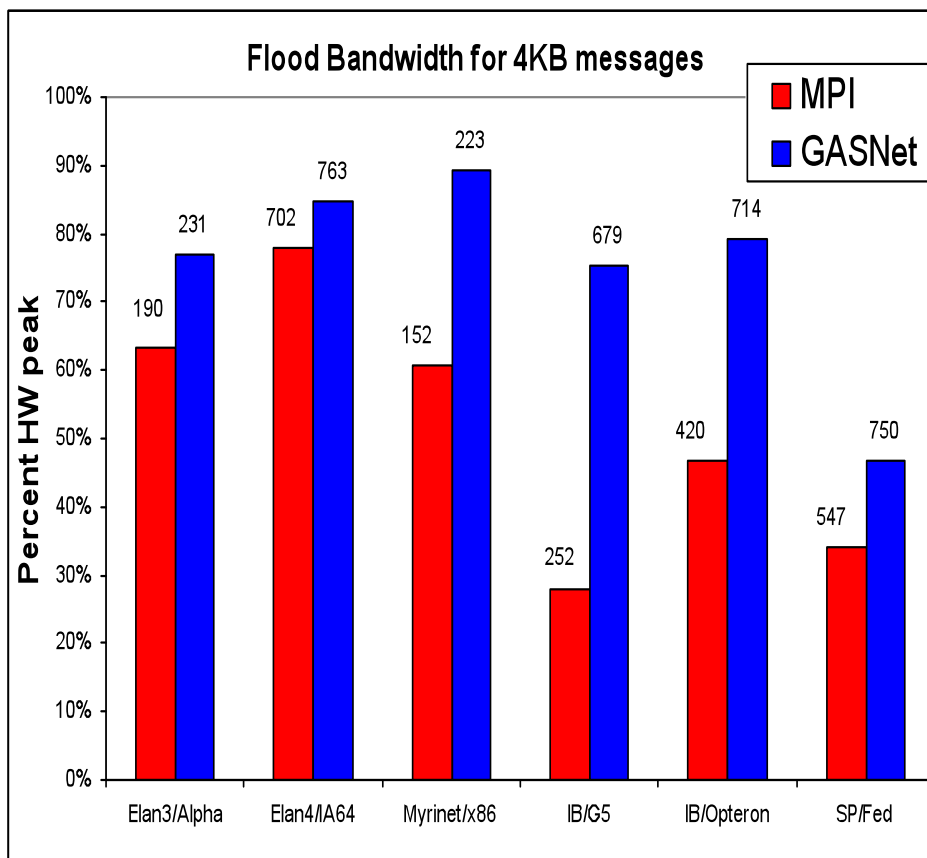
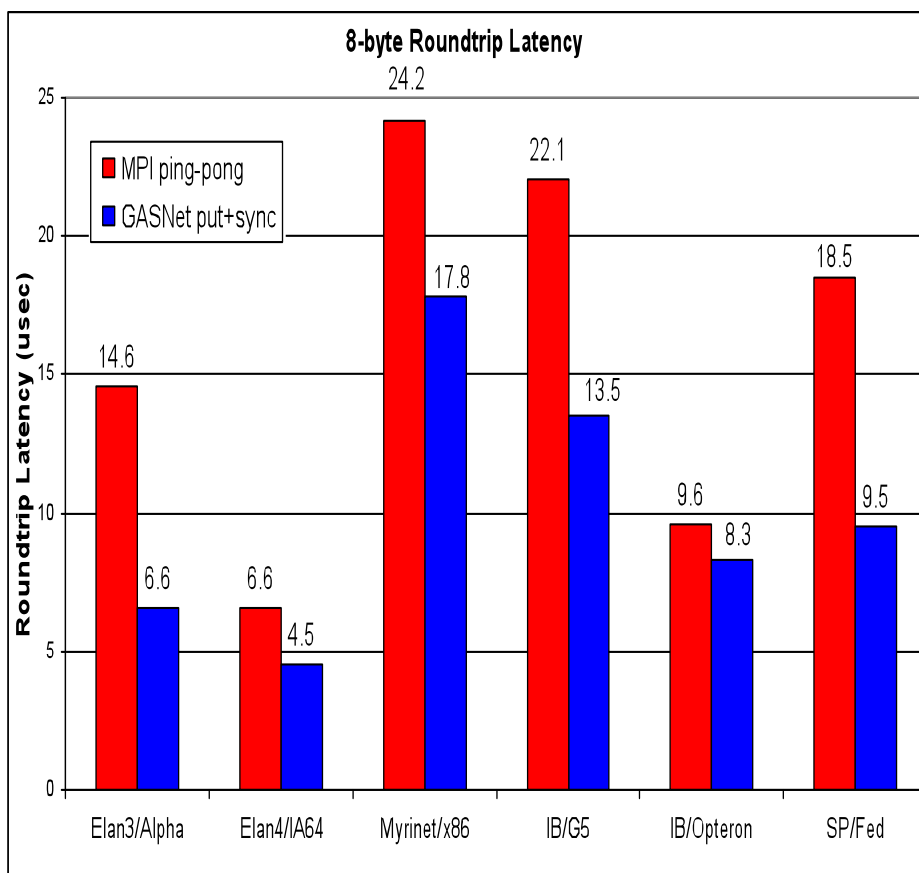






# One-Sided Communication Avoids Unnecessary Overheads

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



U.S. DEPARTMENT OF  
**ENERGY**

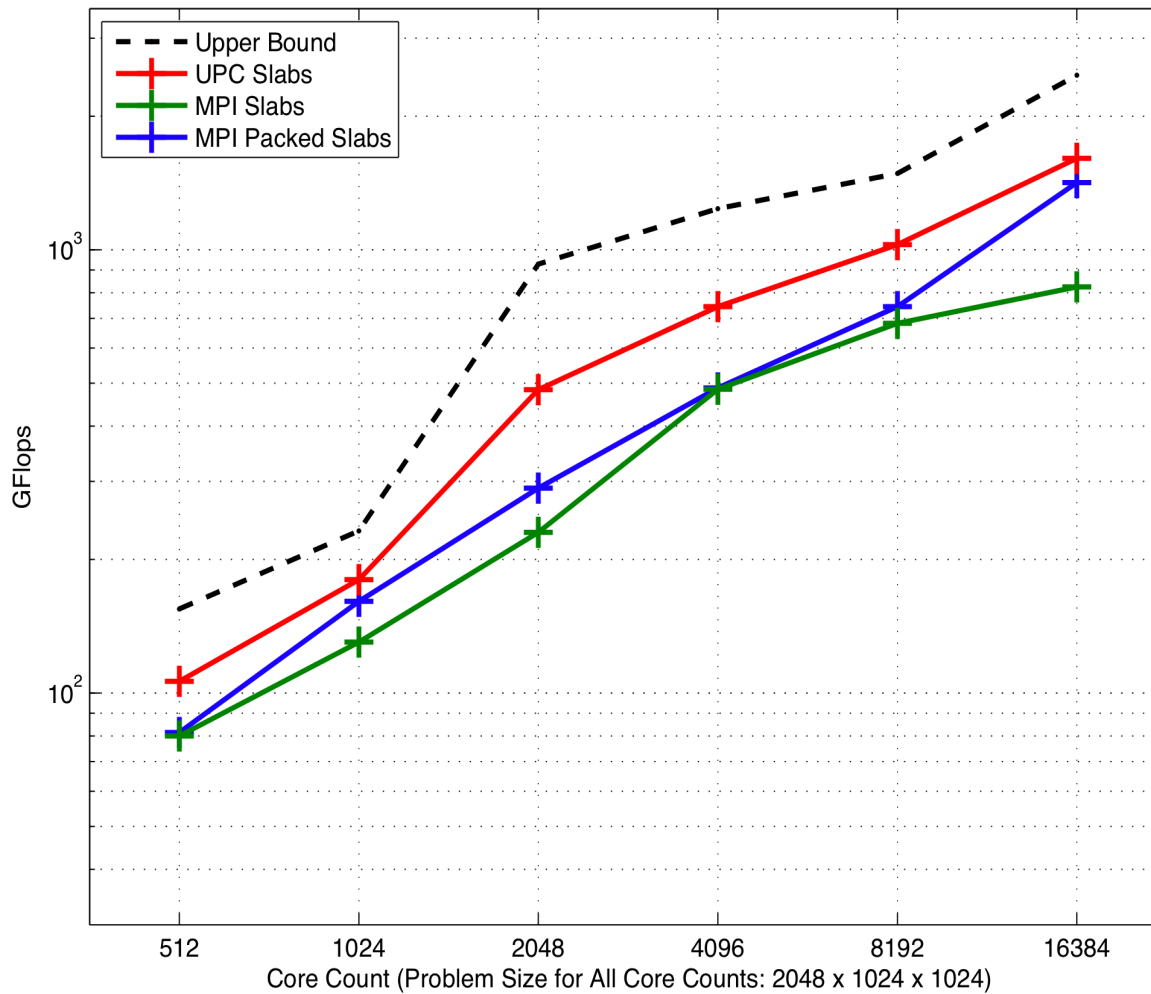
Office of  
Science

Joint work with Berkeley UPC Group





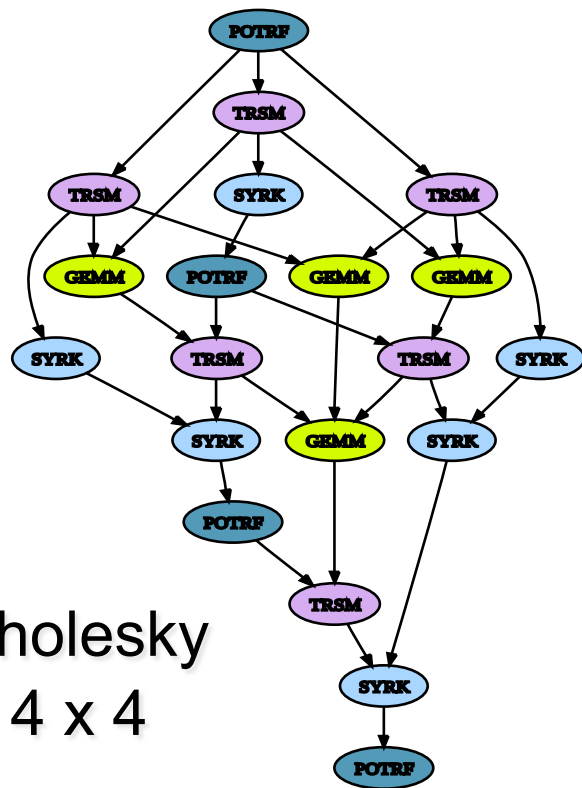
# 3D FFT on BlueGene/P



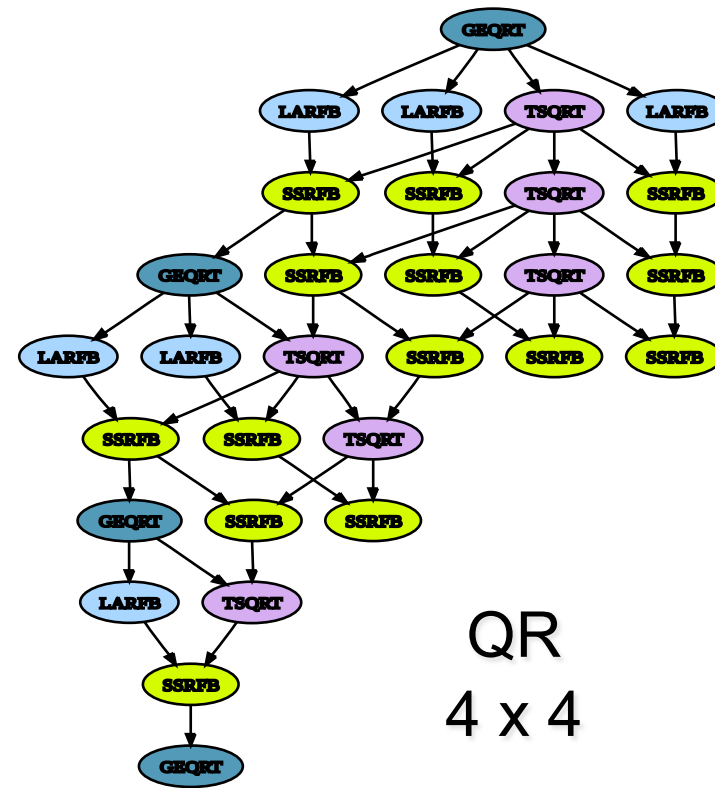
# Avoid Synchronization

## Computations as DAGs

View parallel executions as the directed acyclic graph of the computation



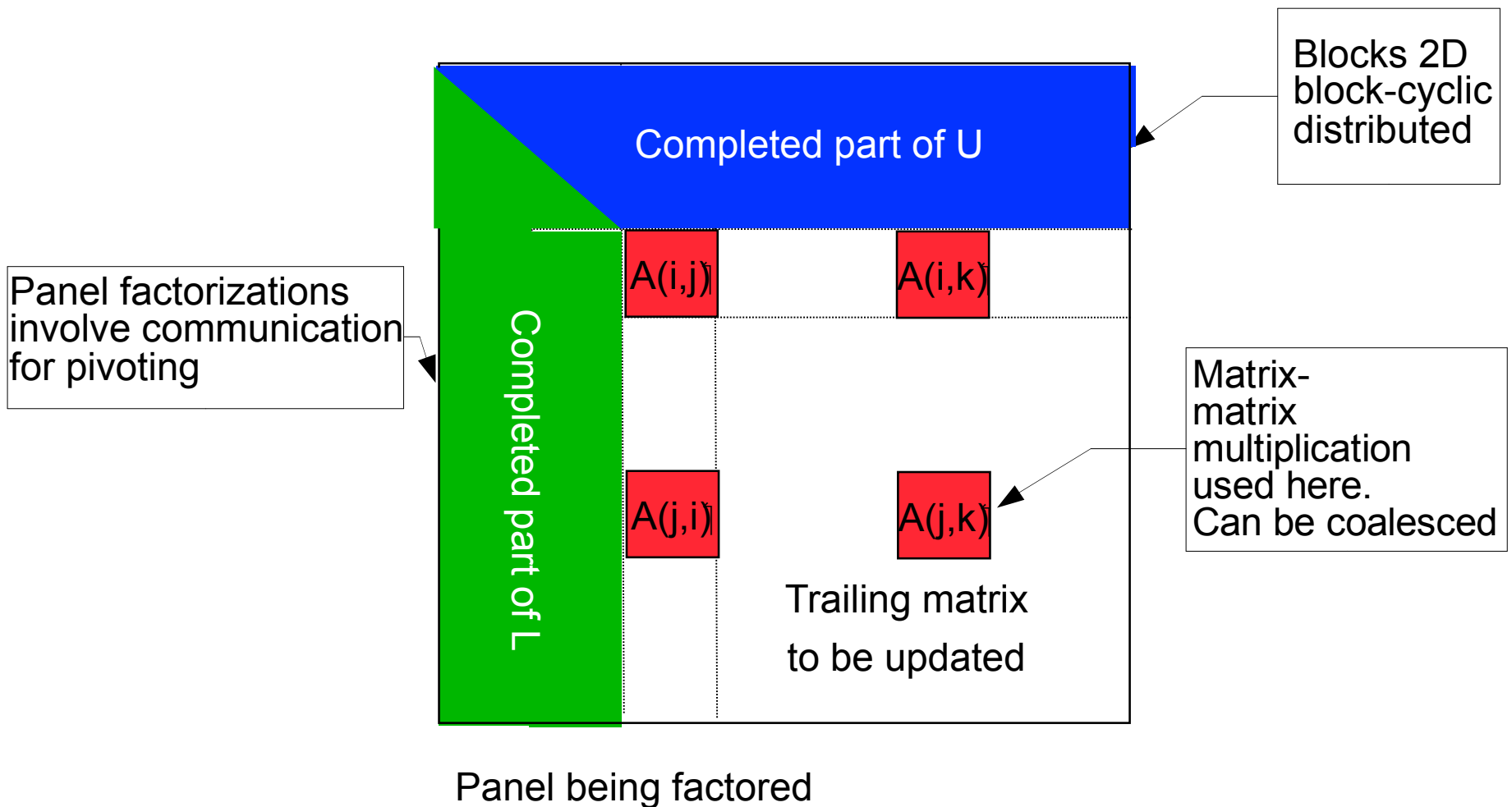
Cholesky  
4 x 4



QR  
4 x 4

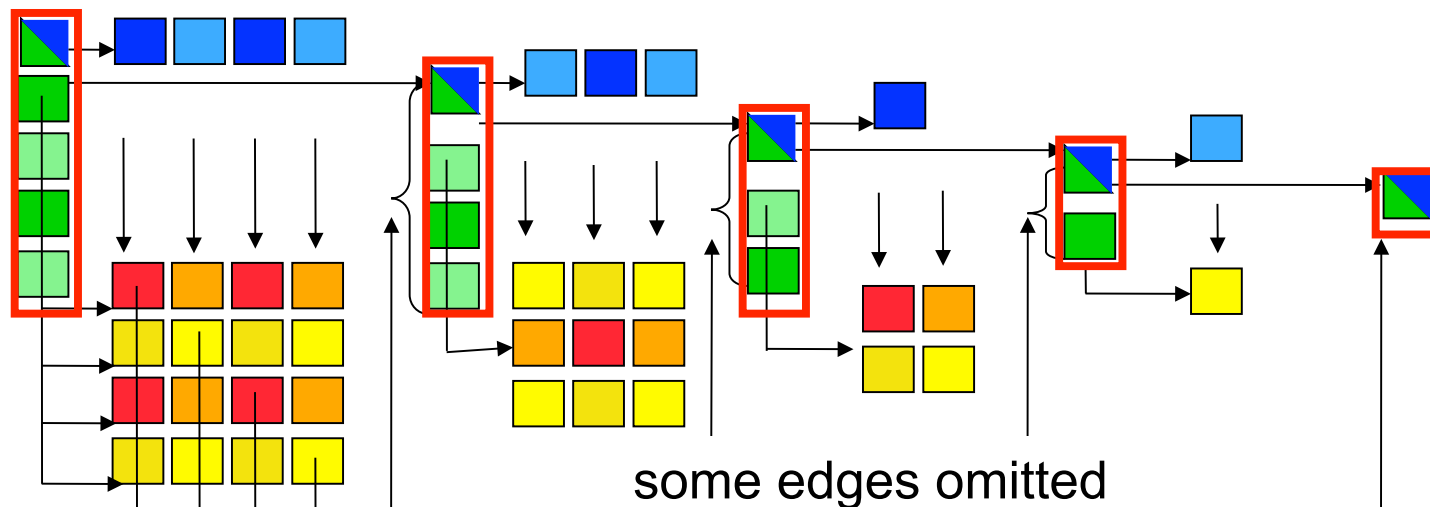


# Parallel LU Factorization



# Event Driven Execution of LU

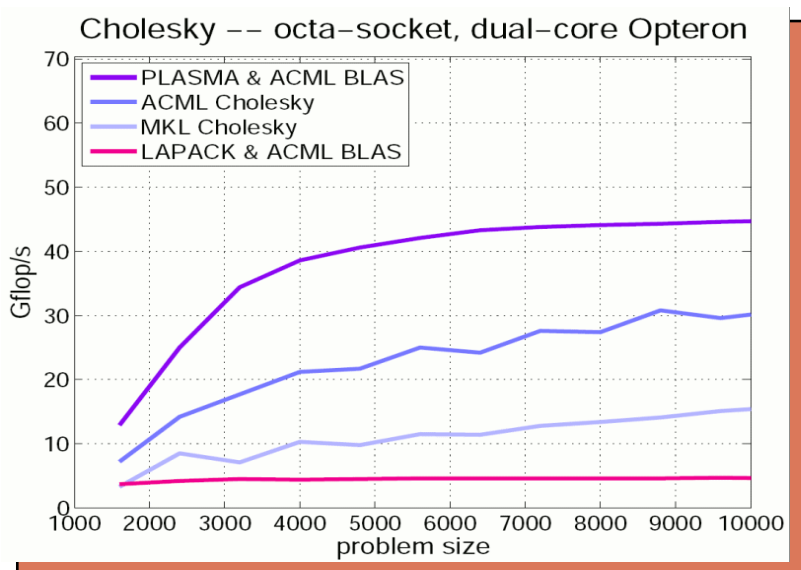
- Ordering needs to be imposed on the schedule
- Critical operation: Panel Factorization
  - need to satisfy its dependencies first
  - perform trailing matrix updates with low block numbers first
  - “memory constrained” lookahead
- General issue: dynamic scheduling in partitioned memory
  - Can deadlock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



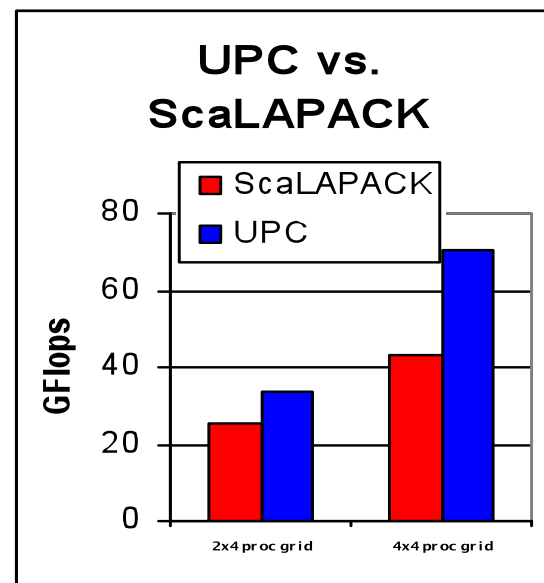


# DAG Scheduling Outperforms Bulk-Synchronous Style

PLASMA on shared memory



UPC on partitioned memory



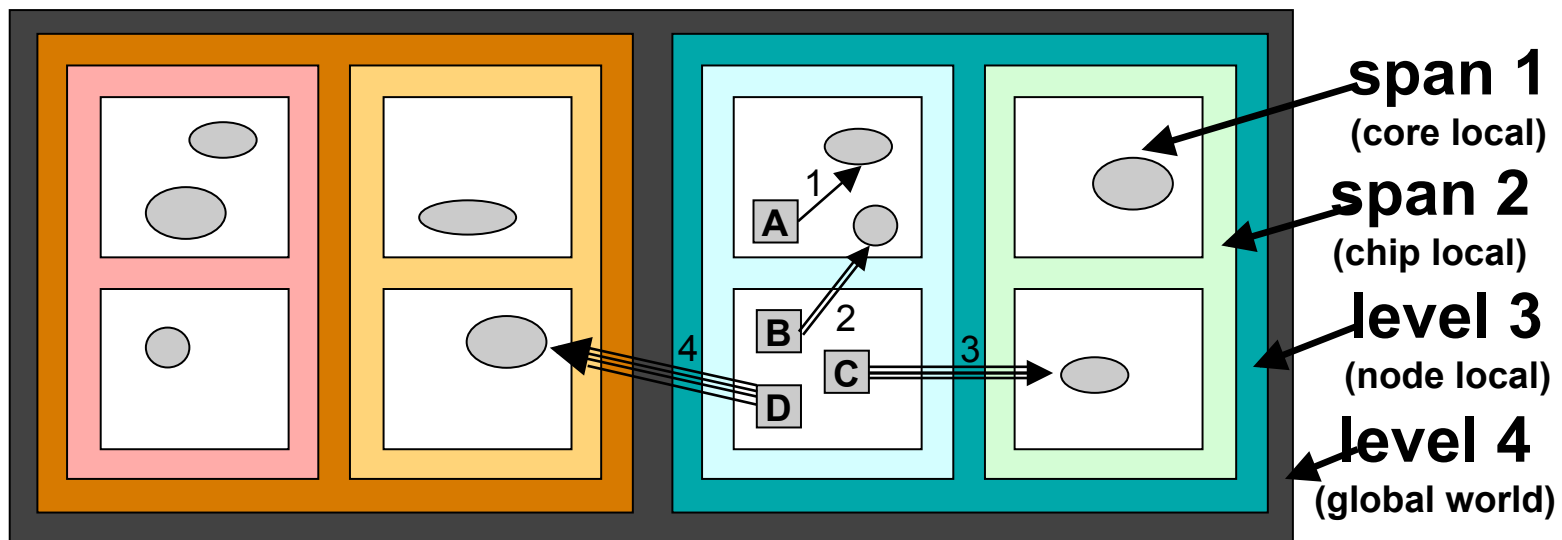
**UPC LU factorization code adds cooperative (non-preemptive) threads for latency hiding**

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



# Hierarchical PGAS Memory Model

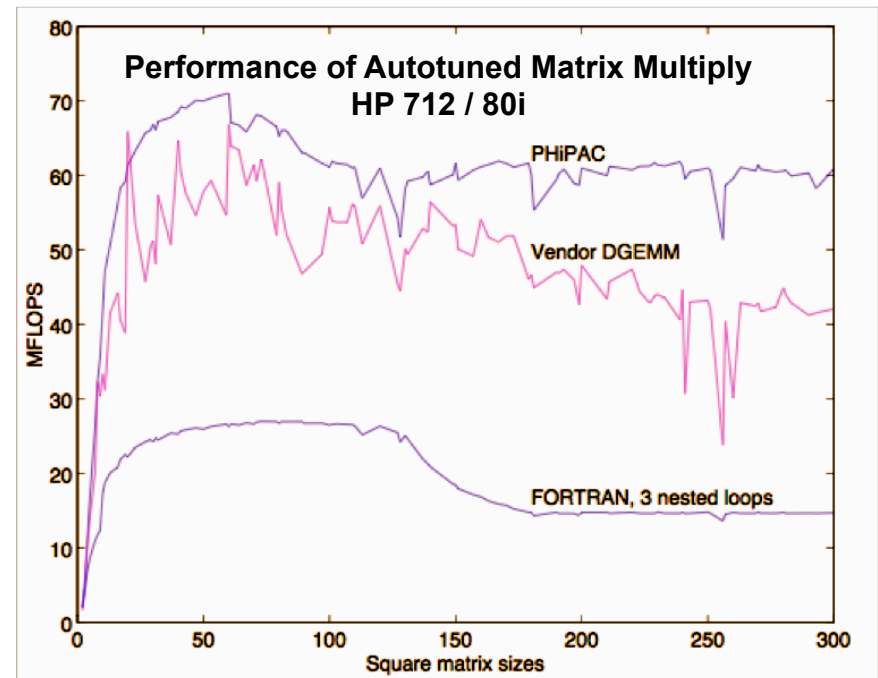
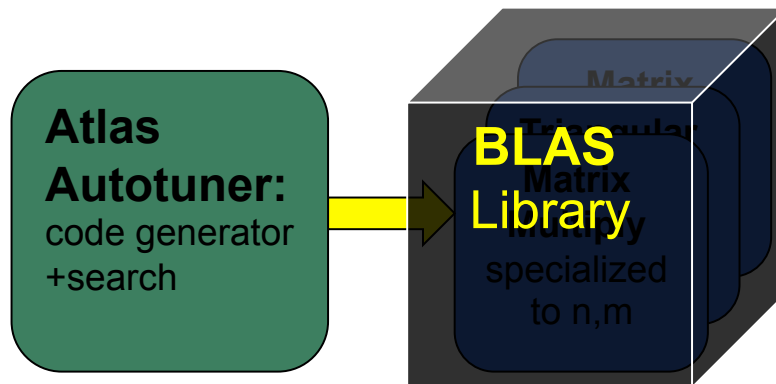
- A global address space for hierarchical machines may have multiple kinds of pointers
- These can be encoded by programmers in type system or hidden, e.g., all global or only local/global
- This partitioning is about pointer span, not control / parallelism





# Autotuning: Write Code Generators

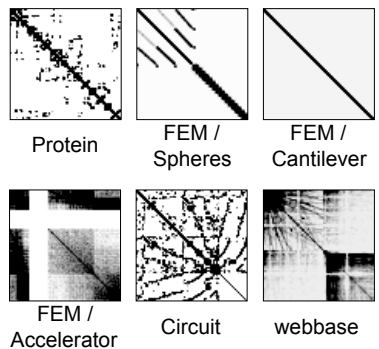
- Autotuners are code generators plus search algorithms to find best code
- Avoids compiler problems of dependence analysis and approximate performance models
- ❖ *Functional portability* from C
- ❖ *Performance portability* from search at install time







# Autotuners for Input-Dependence Optimizations

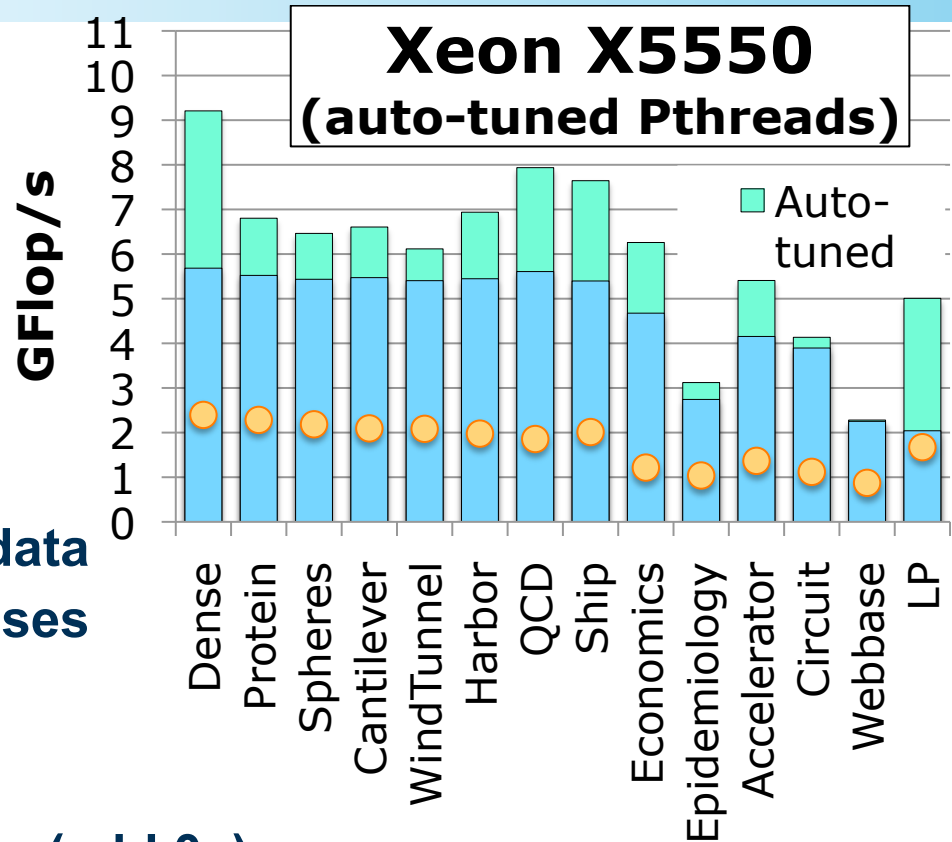


- **Sparse Matrix**

- Significant index meta data
- Irregular memory accesses
- Memory bound

- **Autotuning**

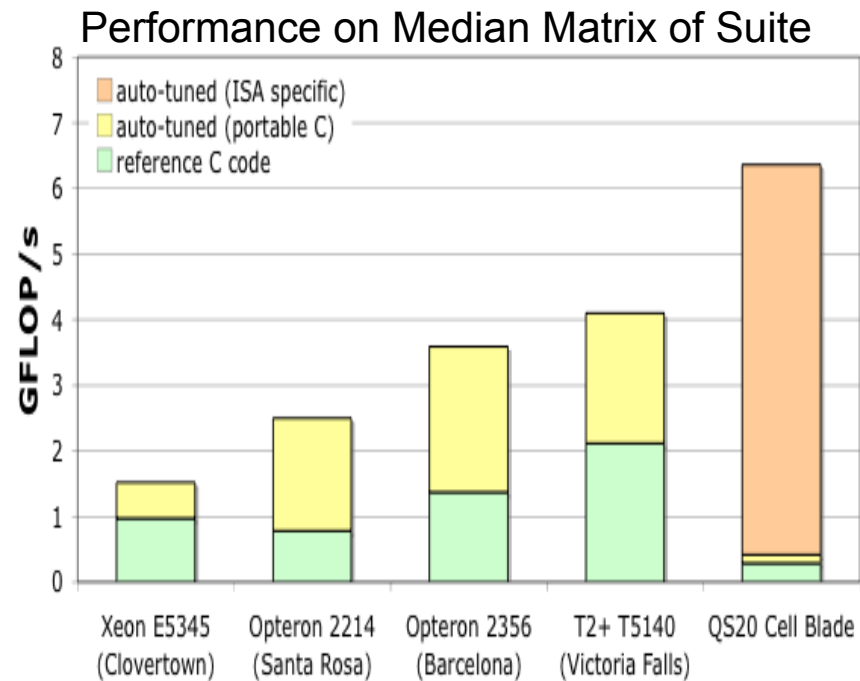
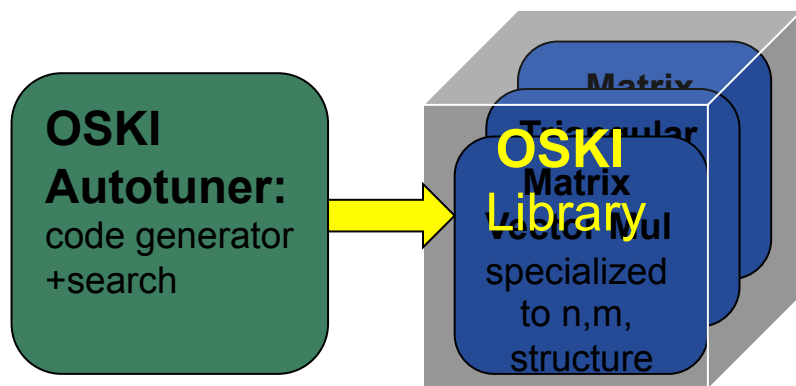
- Tune over data structures (add 0s)
- Delayed tuning decisions until runtime
- Still use significant install-time tuning (dense matrix in sparse format) with online specialization based on matrix





# 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






# 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
  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;}

```



# Algorithms to Optimize for Communication



# Choose Scalable Algorithms

- Algorithmic gains in last decade have far outstripped Moore's Law

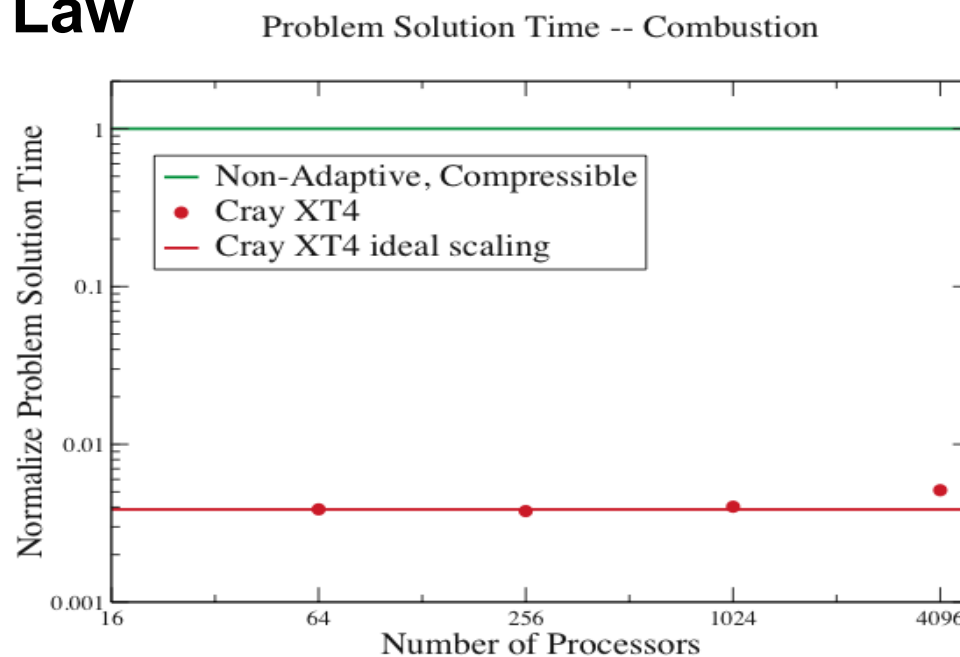
- Adaptive meshes rather than uniform
- Sparse matrices rather than dense
- Reformulation of problem back to basics

- Two kinds of scalability

- In problem size ( $n$ )
- 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





# Communication-Avoiding Algorithms

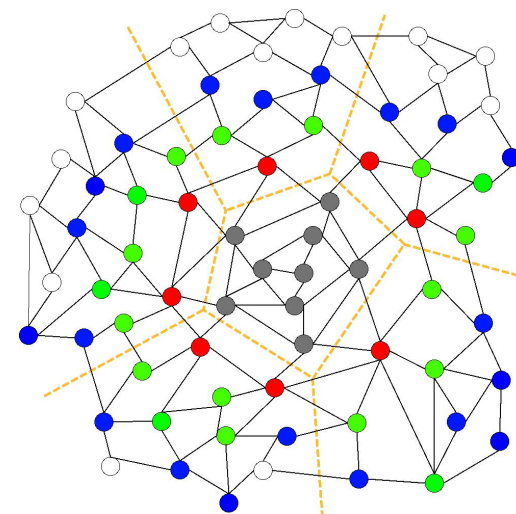
## Consider Sparse Iterative Methods

- 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?

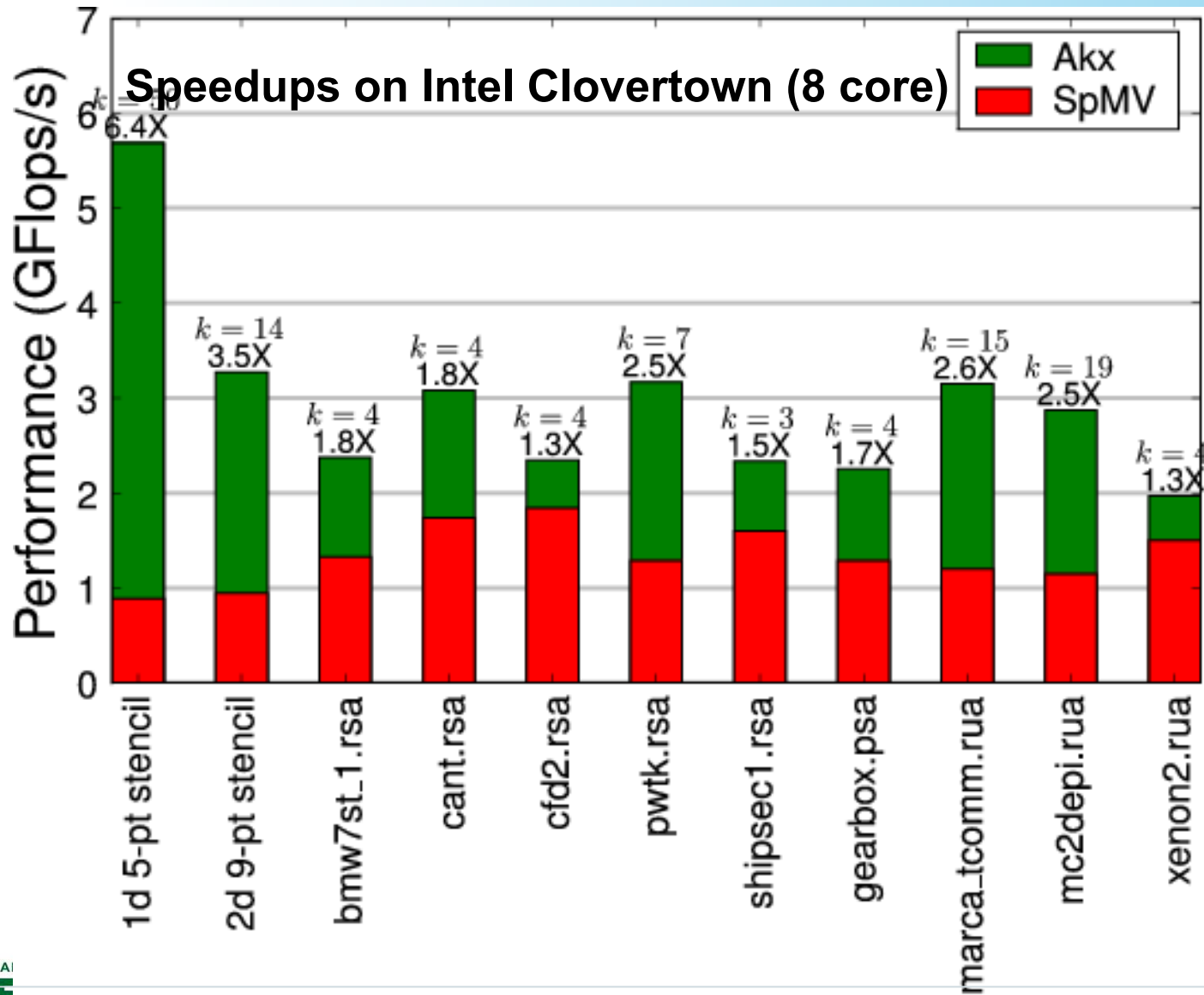
- Take  $k$  steps “at once” with one matrix read from DRAM and one communication phase
- **Parallel implementation**  
 $O(\log p)$  messages vs.  $O(k \log p)$
- **Serial implementation**  
 $O(1)$  moves of data moves vs.  $O(k)$

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin



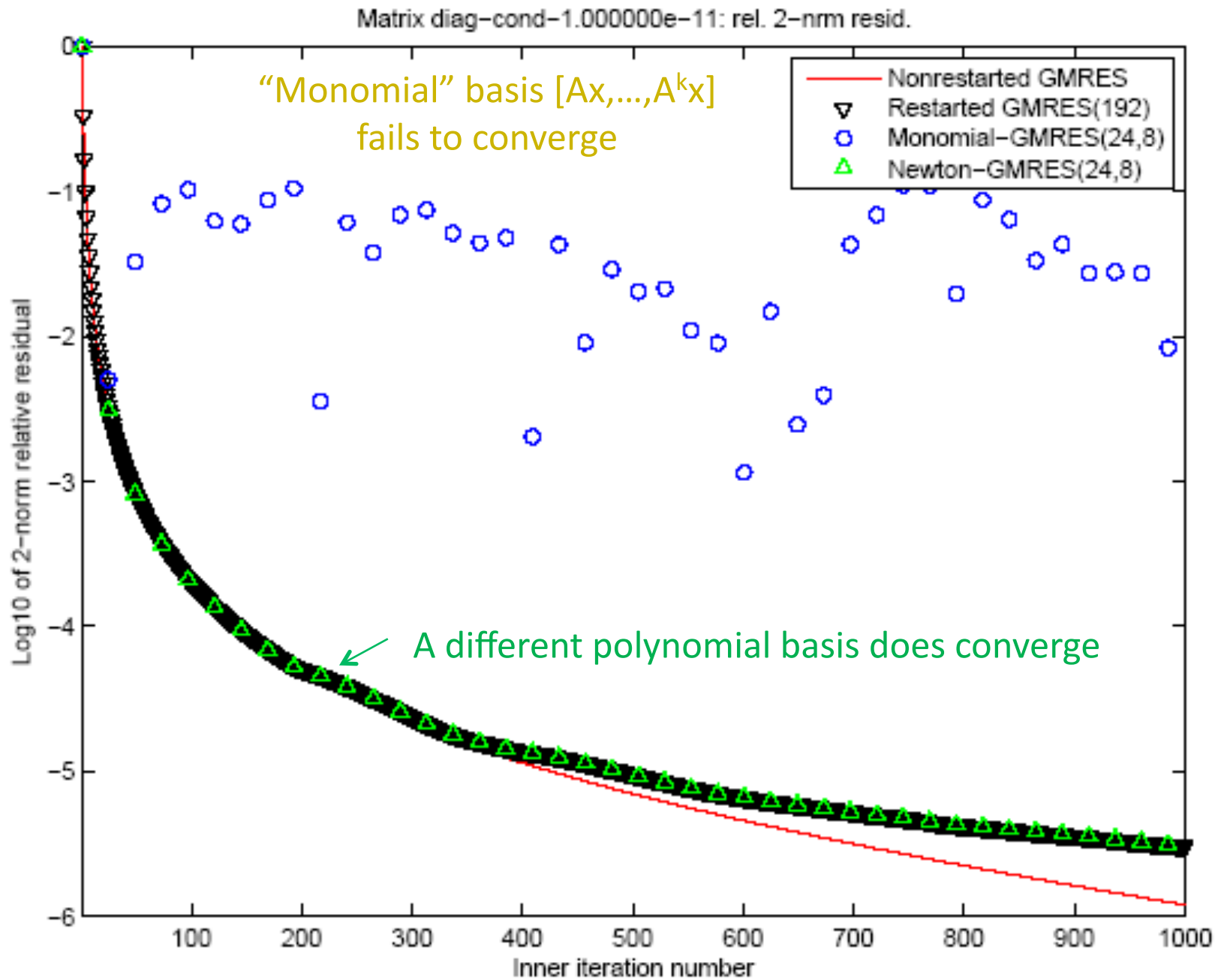


# Bigger Kernel ( $A^kx$ ) Runs at Faster Speed than Simpler ( $Ax$ )



Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Kathy Yelick





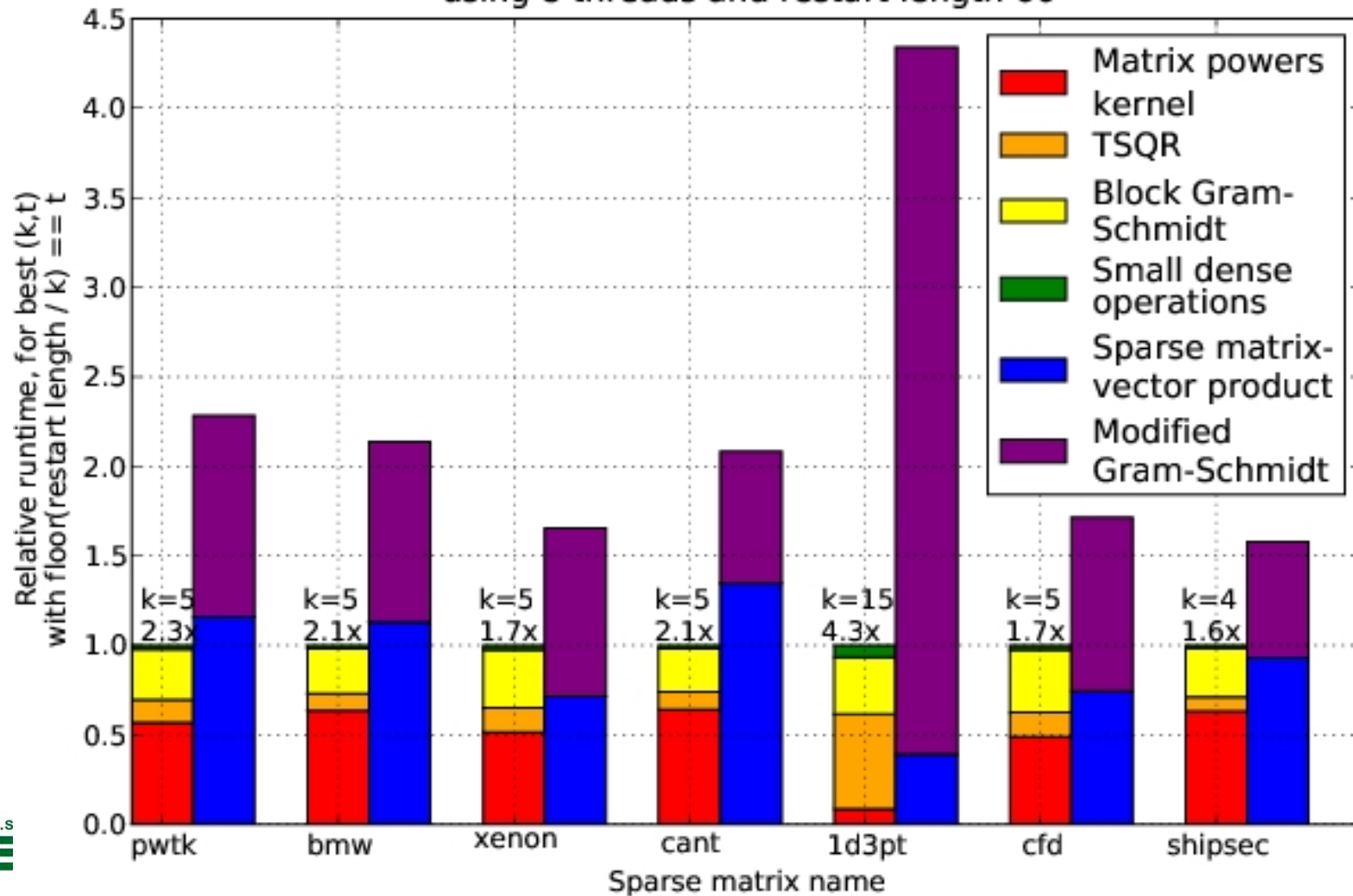




# Communication-Avoiding Krylov Method (GMRES)

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





# Communication-Avoiding Dense Linear Algebra

- **Well known why BLAS3 beats BLAS1/2: Minimizes communication = data movement**
  - Attains lower bound  $\Omega (n^3 / \text{cache\_size}^{1/2})$  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 ( $n^3 \Rightarrow \#flops$  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



## 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**



Questions?