

Autotuning in the Exascale Era

Kathy Yelick

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

EECS Professor, UC Berkeley

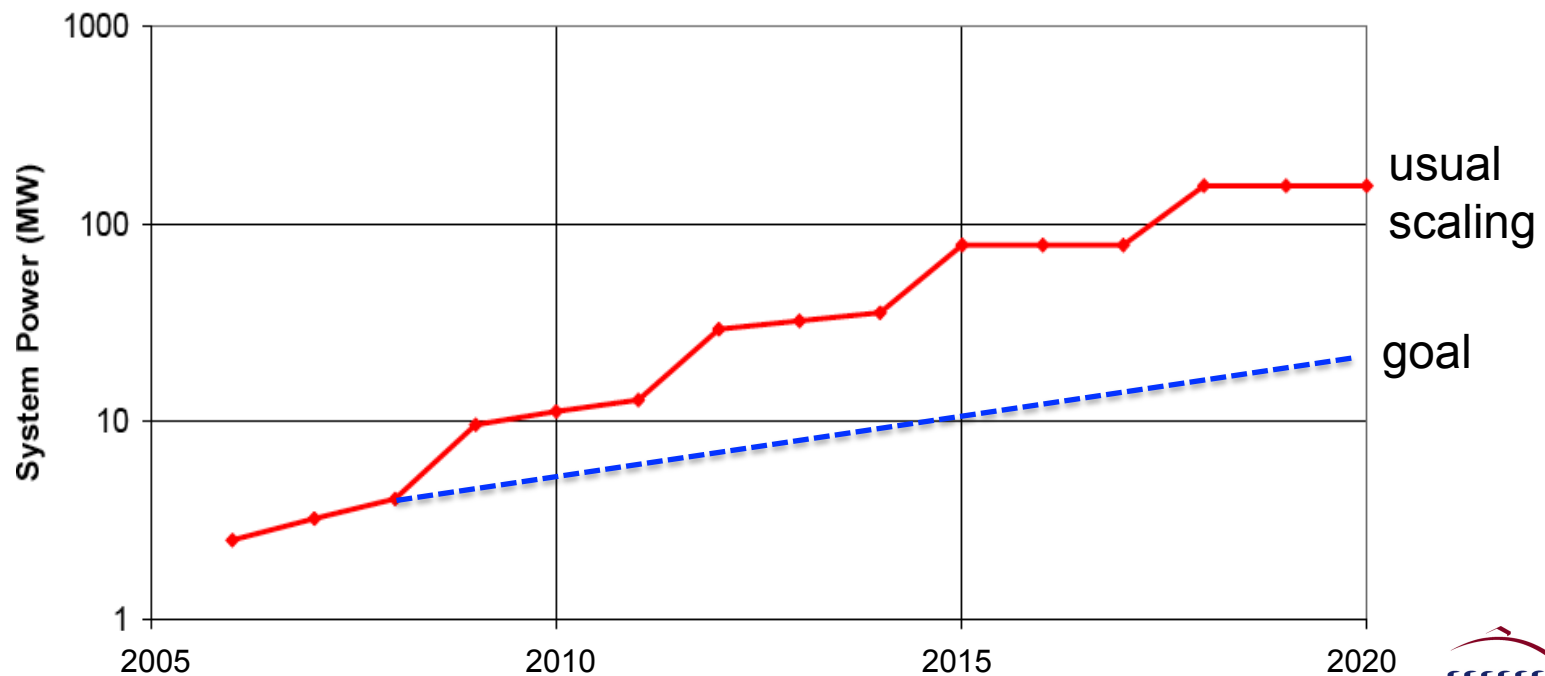




Energy Cost Challenge for Computing Facilities

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



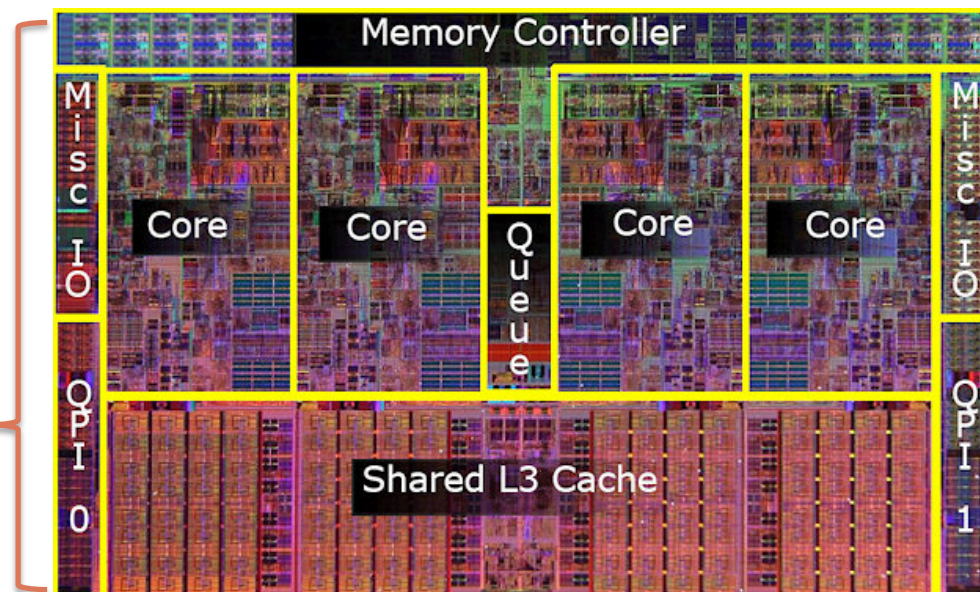


New Processor Designs are Needed to Save Energy



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

Server processor
(100 Watts, 50 Gflop/s)



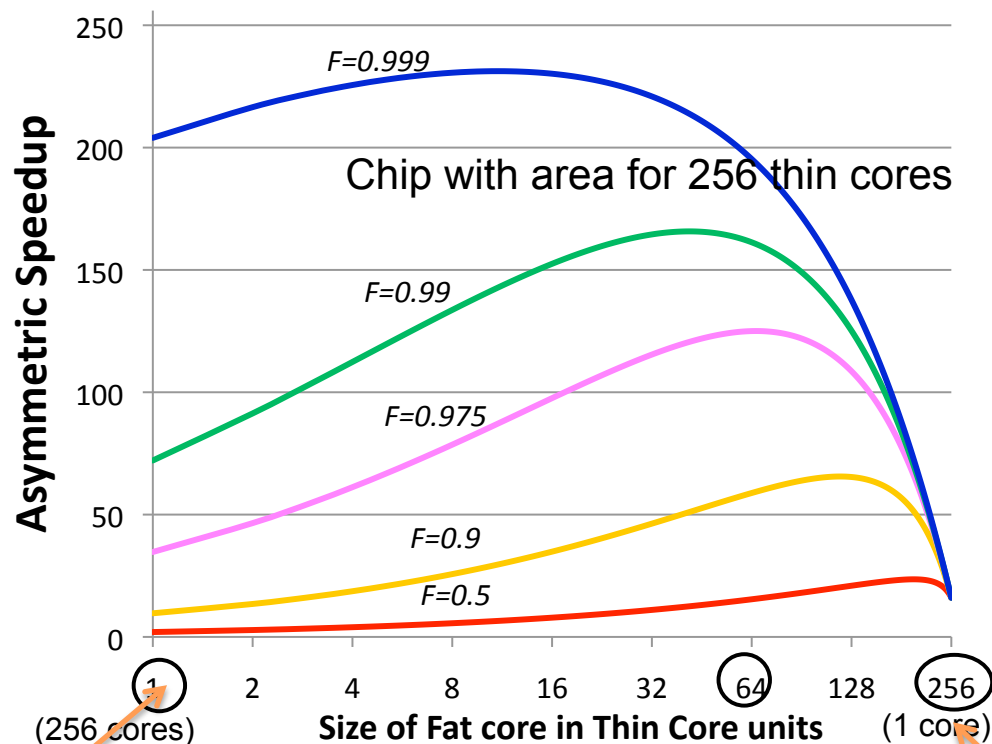
- **Server processors have been designed for performance, not energy**
 - Graphics processors are 10-100x more efficient
 - Embedded processors are 100-1000x
 - Need manycore chips with thousands of cores





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

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



6/7/11

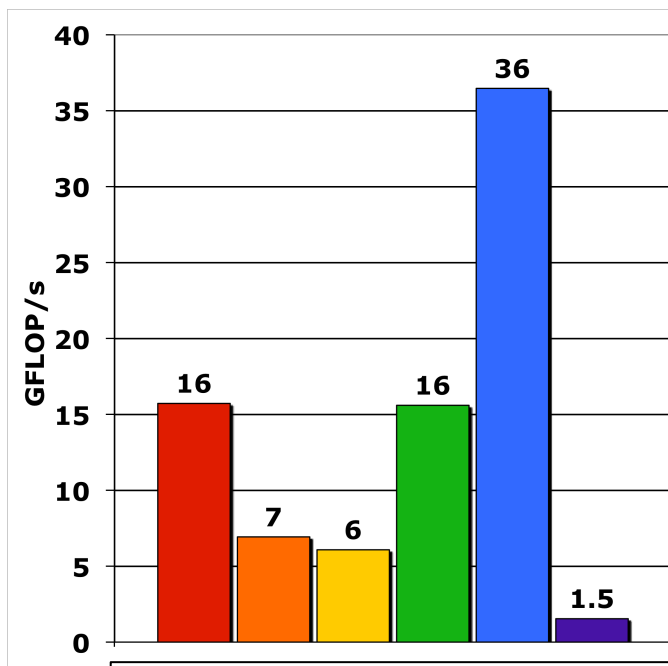




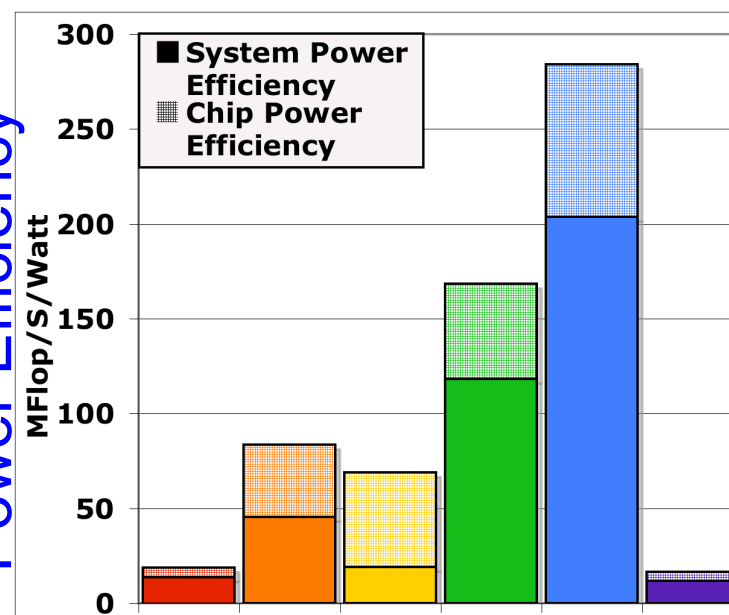
Energy Efficiency of Applications

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

Performance



Power Efficiency



Cache-based

- Gainestown
- Barcelona
- Victoria Falls

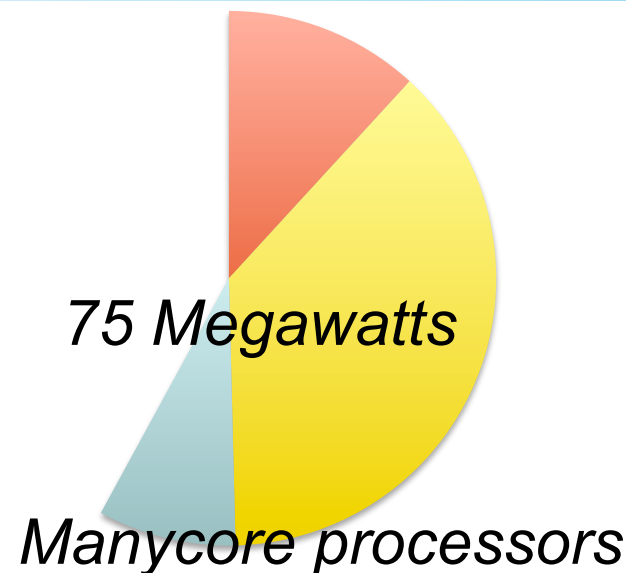
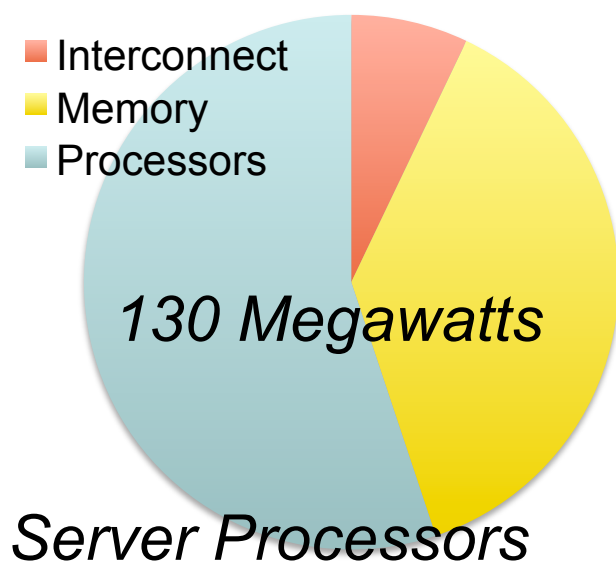
Local store-based

- Cell Blade
- GTX280
- GTX280-Host





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**





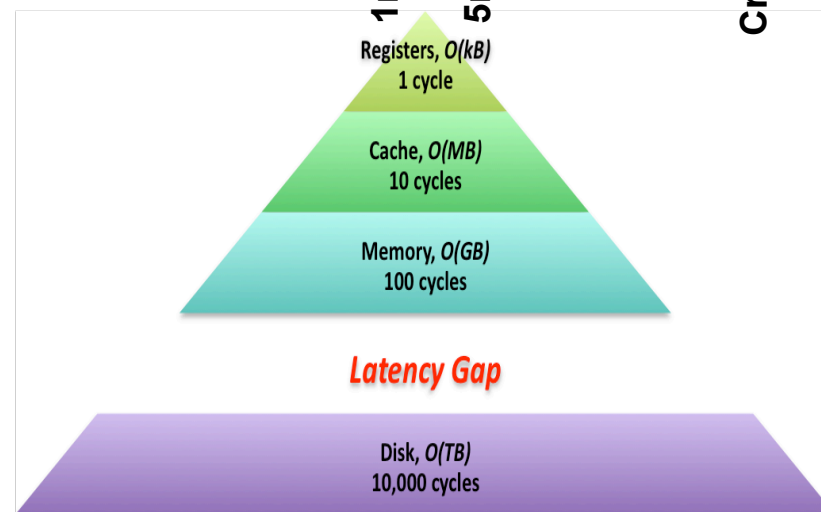
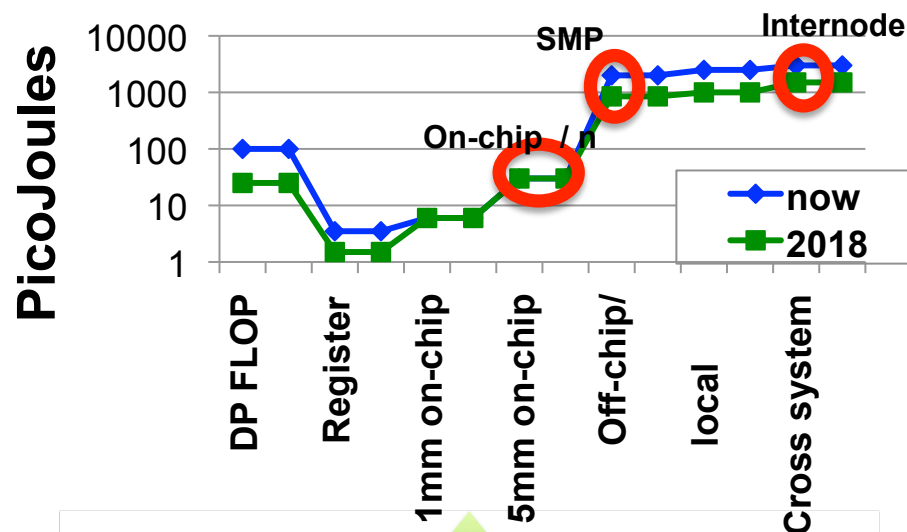
Memory and Storage Challenges for Exascale

• Challenges:

- Height of the memory wall is growing
- Off-chip bandwidth, latency, and poor concurrency throttle performance
- Per-disk performance not improving

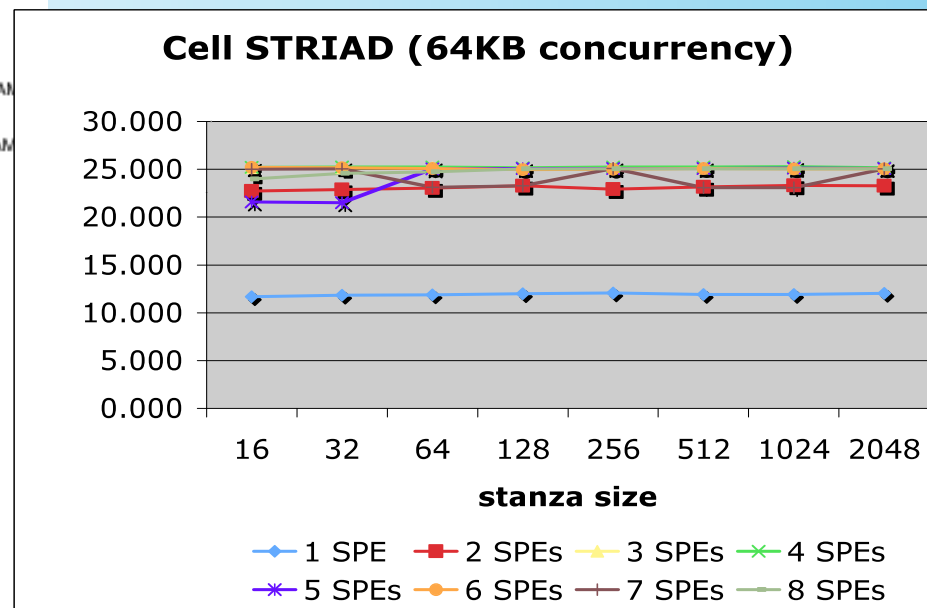
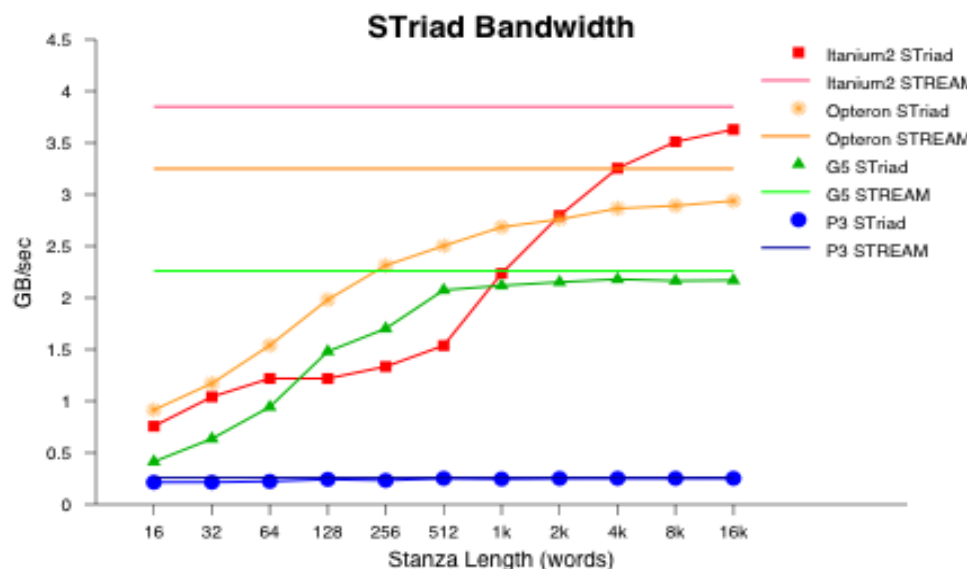
• Approaches:

- 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

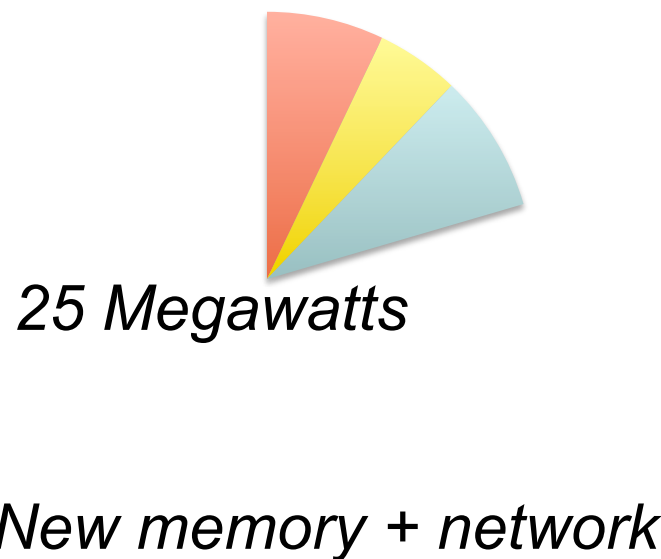
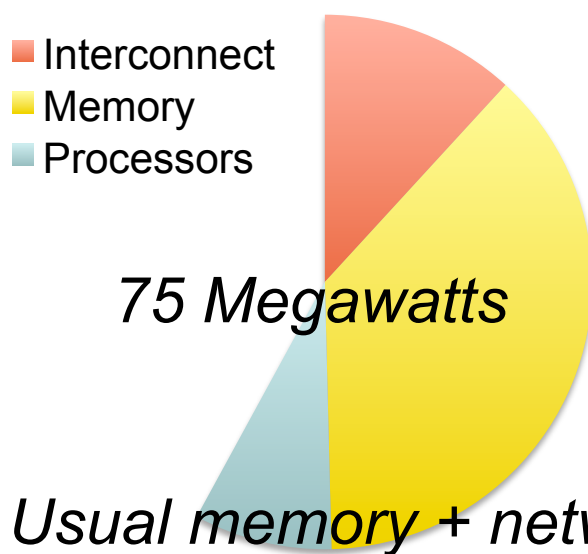


Joint work with Shoaib Kamil, Lenny Oliker, John Shalf, Kaushik Datta





New Memory and Network Technology to Lower Energy



- **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

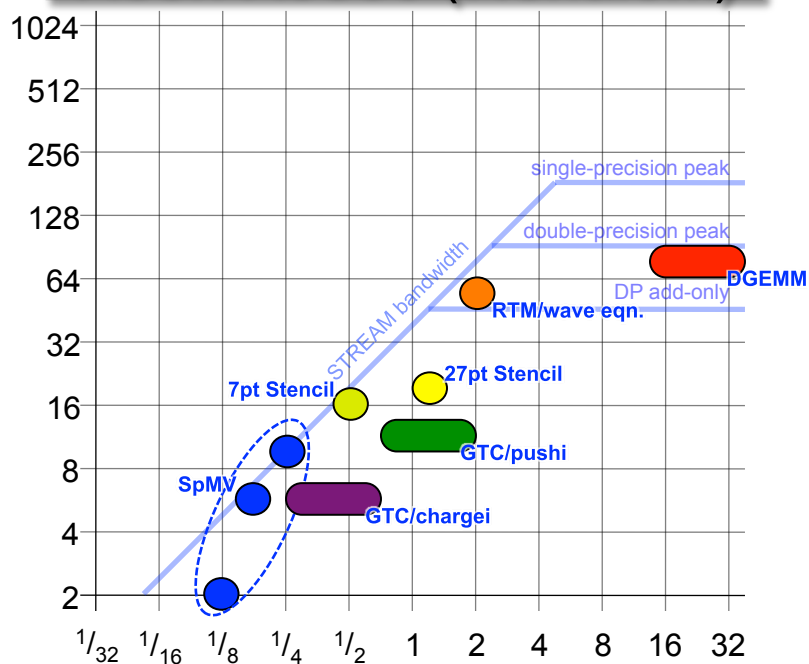




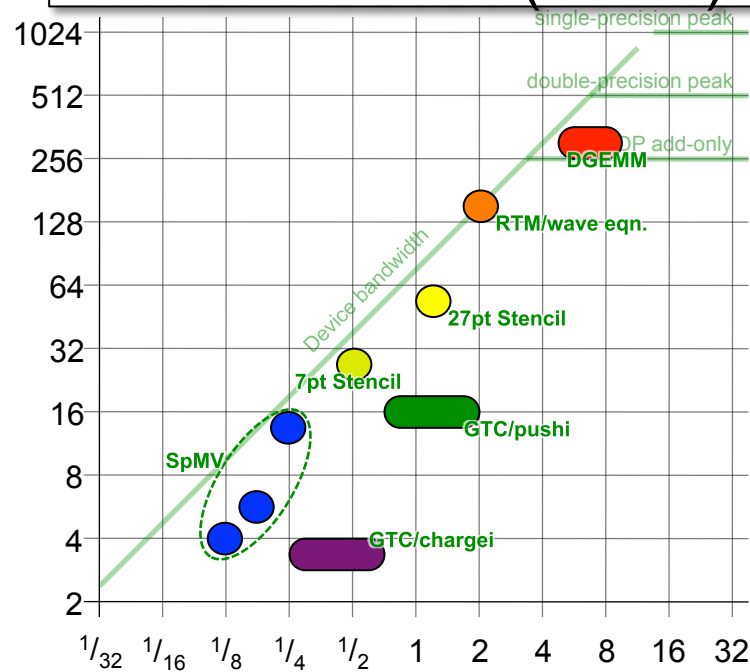
Autotuning Gets Kernel Performance Near Optimal

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

Xeon X5550 (Nehalem)



NVIDIA C2050 (Fermi)



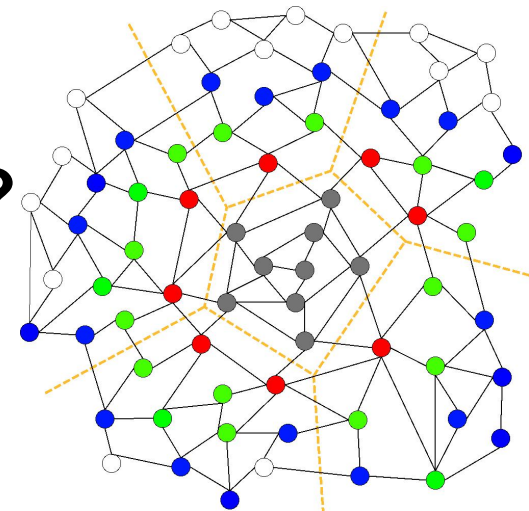
Work by Williams, Olikar, Shalf, Madduri, Kamil, Im, Ethier,...





Exascale Will Require New Algorithmic Thinking

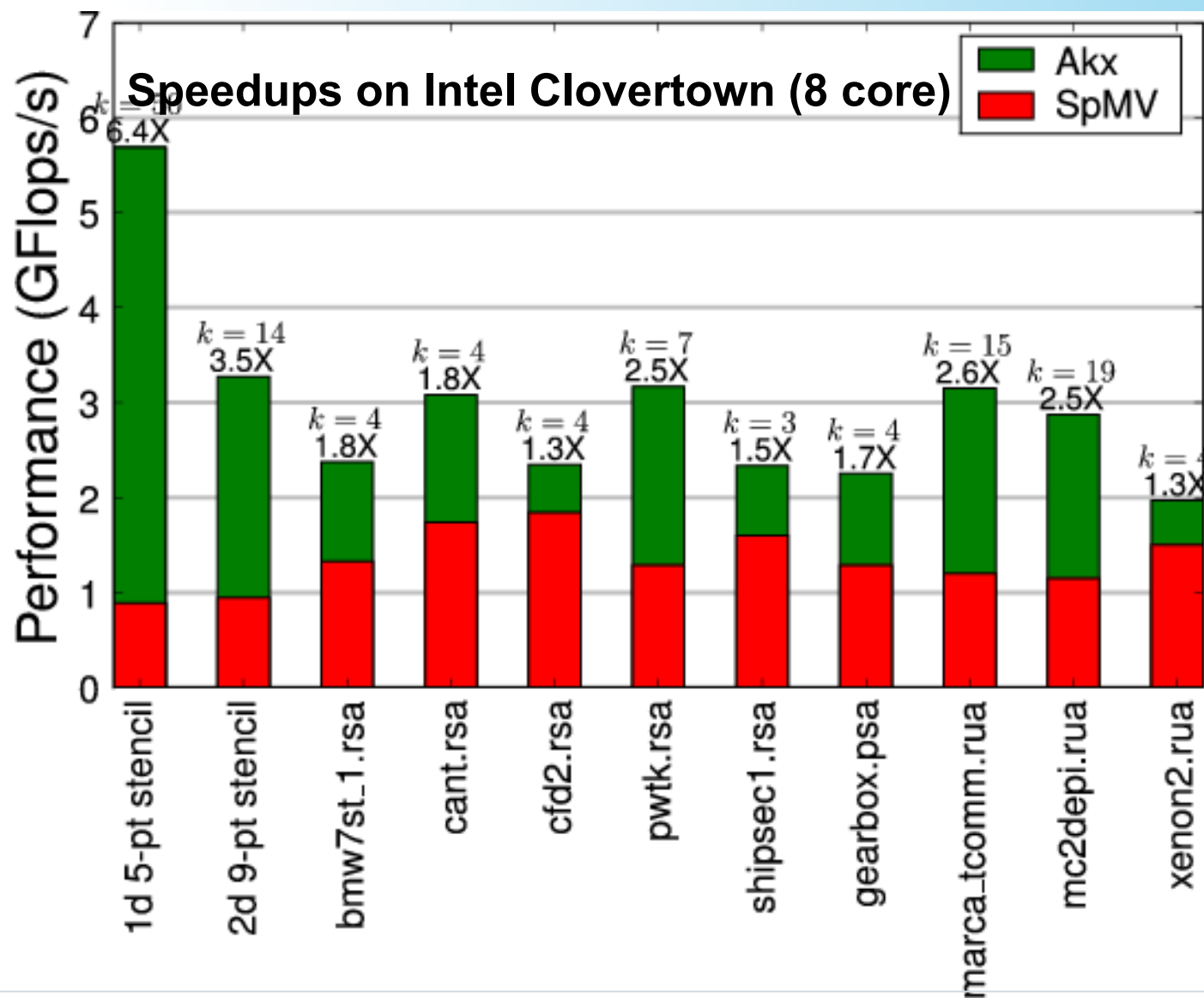
- **Sparse Iterative (Krylov Subspace) Methods**
 - Nearest neighbor communication on a mesh
 - Dominated by time to read matrix (edges) from DRAM
 - And (small) communication and global synchronization events at each step
- **Can we lower data movement costs?**
 - Take k steps with one matrix read from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

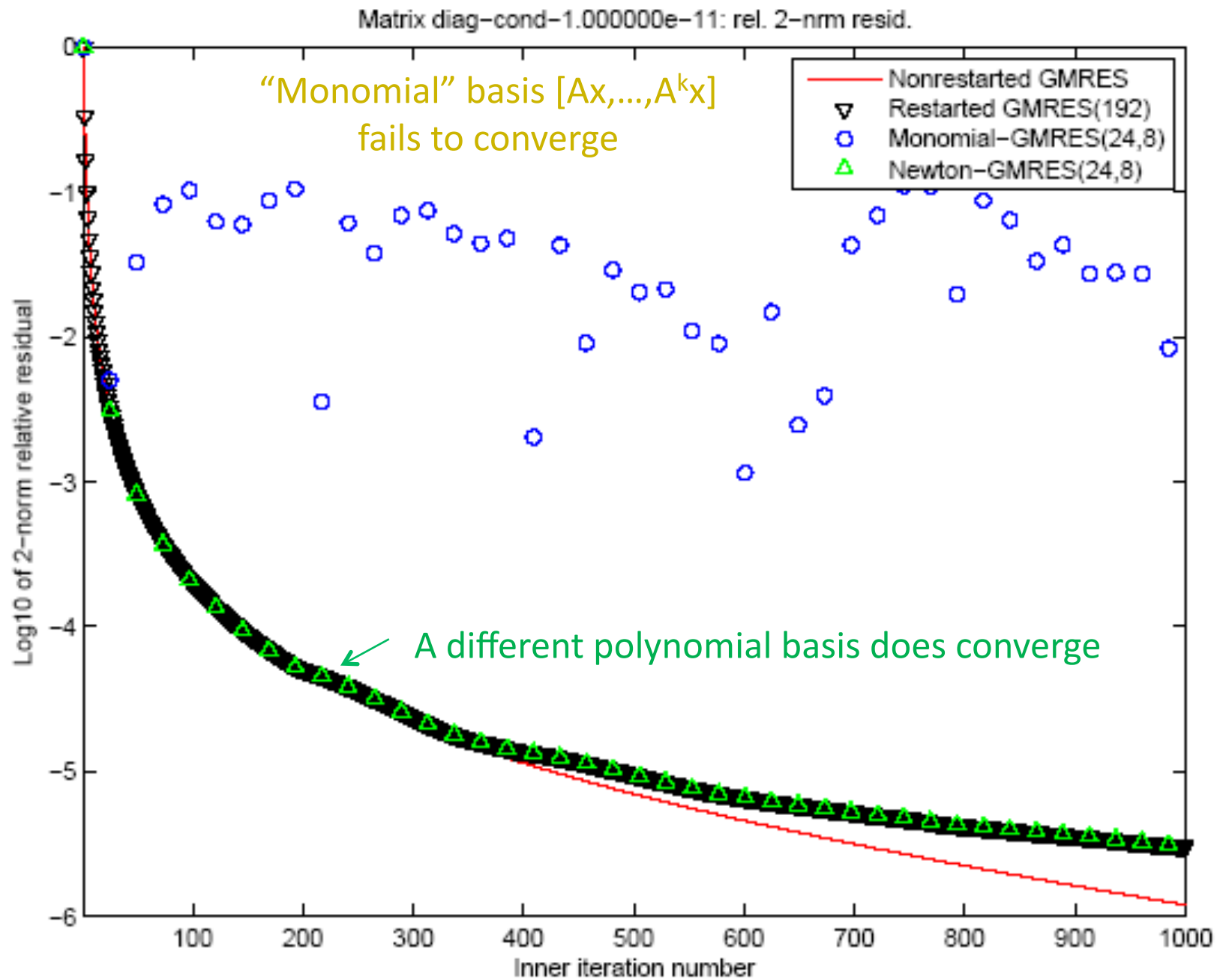


Bigger Kernel ($A^k x$) Runs at Faster Speed than Simpler (Ax)



Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Kathy Yelick



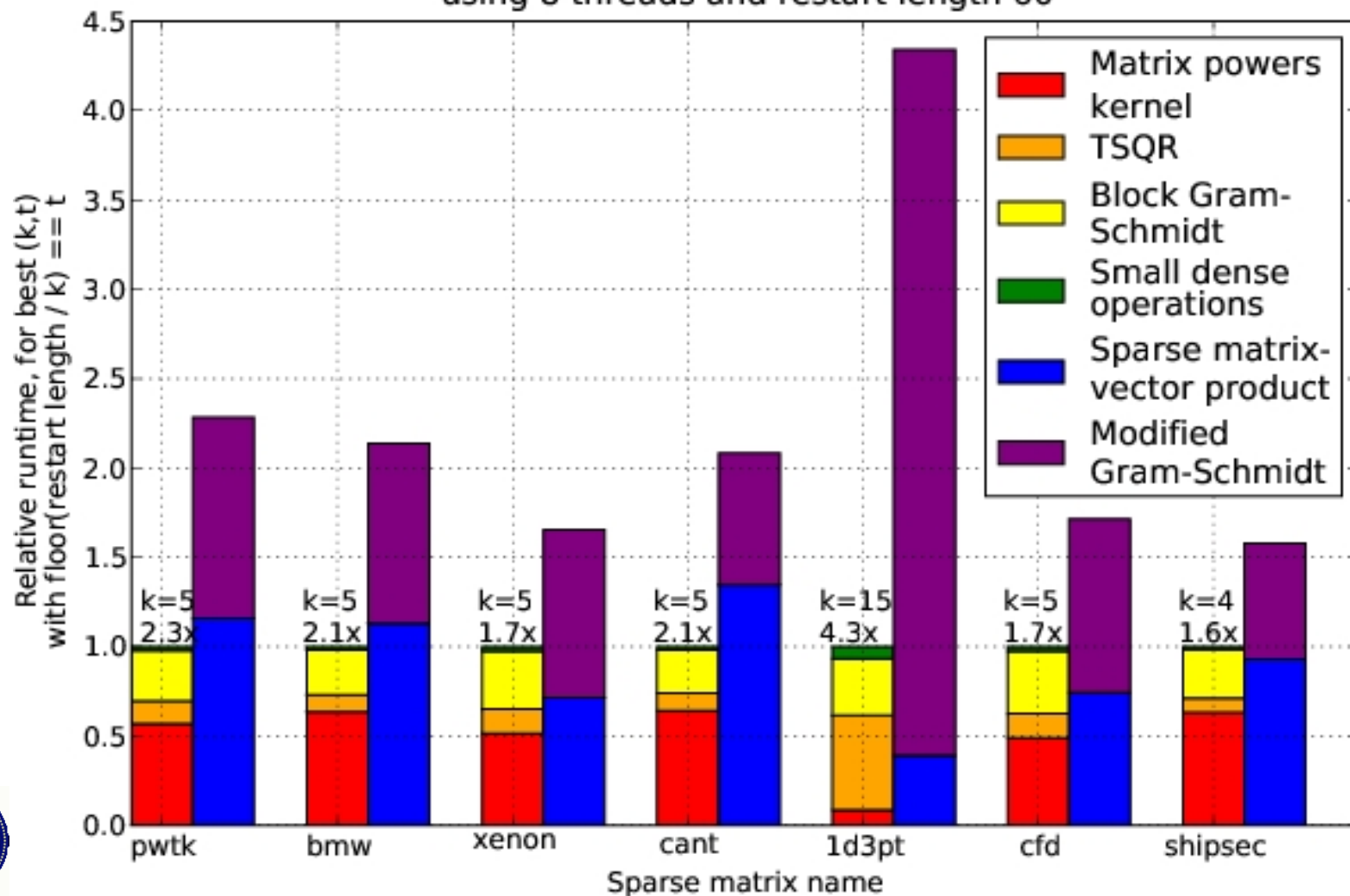




Communication-Avoiding Krylov Method (GMRES)

Performance on 8 core Clovertown

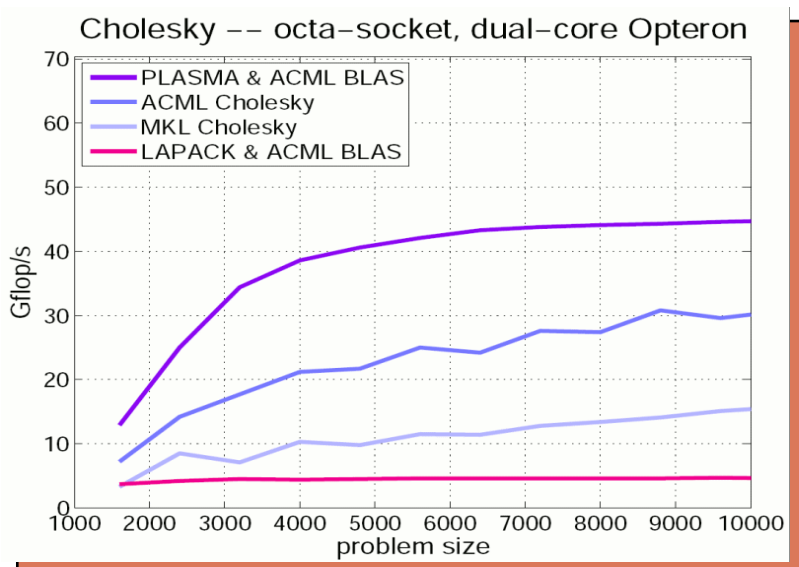
Runtime per kernel, relative to CA-GMRES(k,t), for all test matrices, using 8 threads and restart length 60



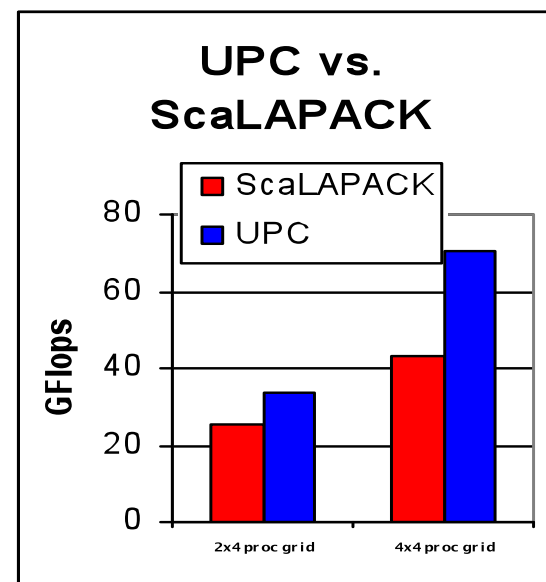


DAG Scheduling Outperforms Bulk-Synchronous Style

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
Parry Husbands





Hardware and Software Scaling Require New Resilience Models

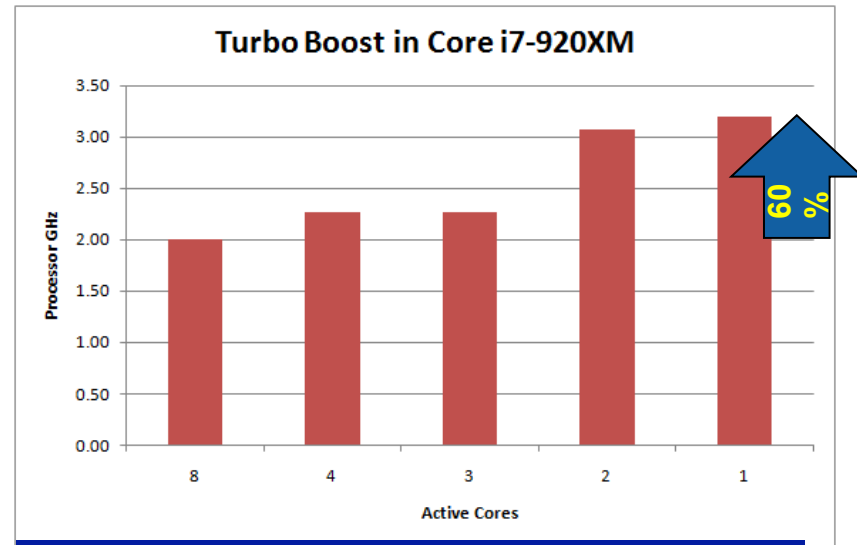
- **Resiliency challenges**

- Chance of component failure grows with system
- Failure and power management → irregular performance behavior

- **Hardware / software**

- Component values should not cause system-wide or application-wide outages

Affects all software levels



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

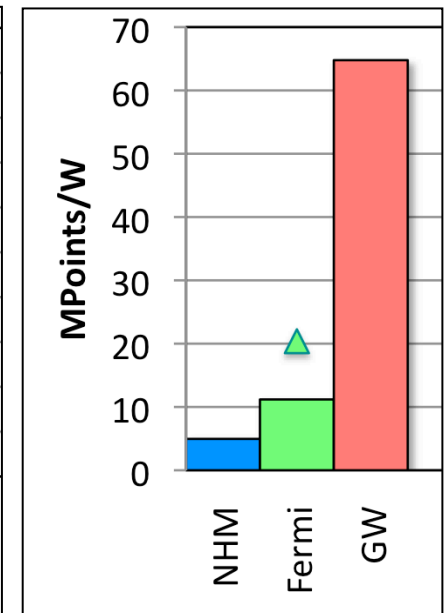
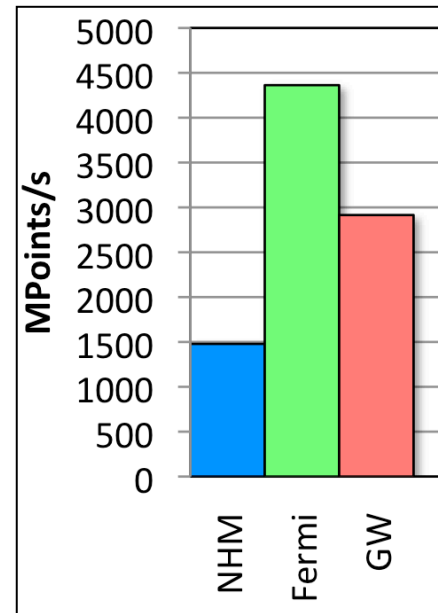
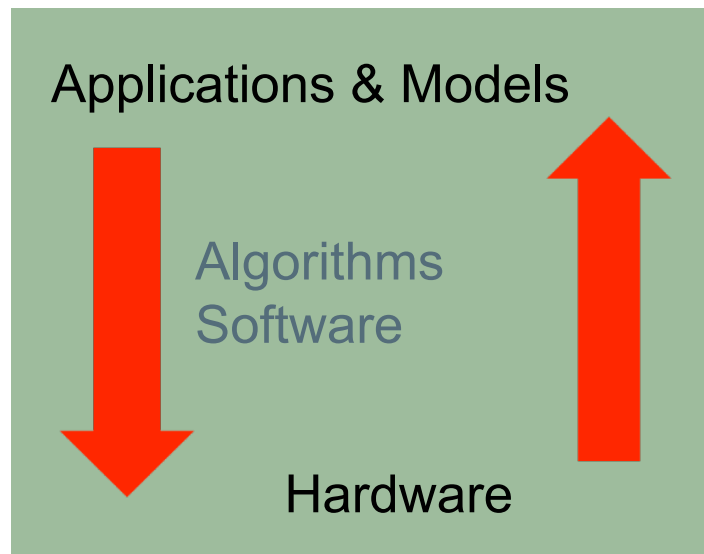




CoDEx: Co-Design for Exascale

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

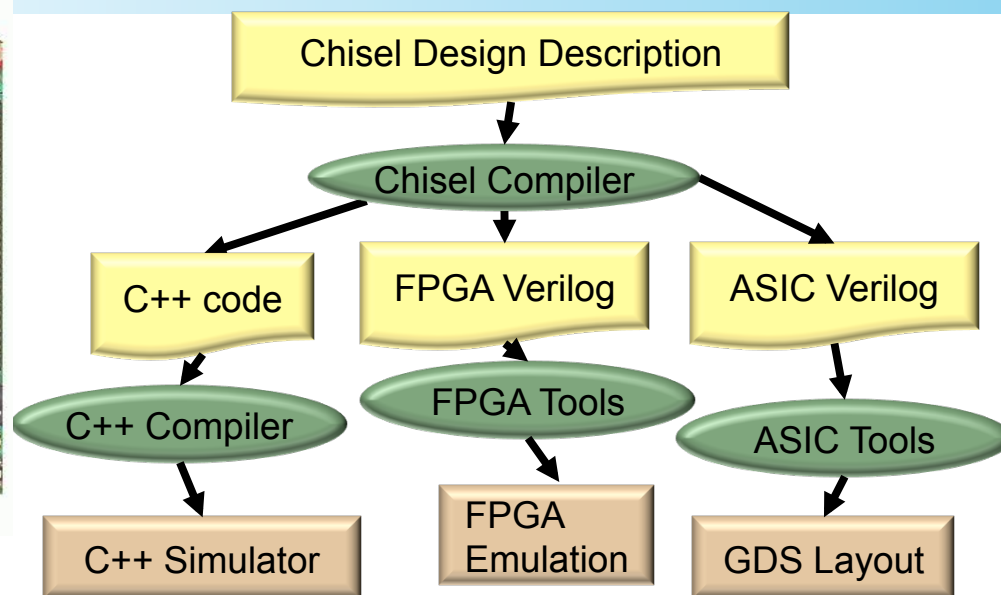




RAMP: Enabling Manycore Architecture Research



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

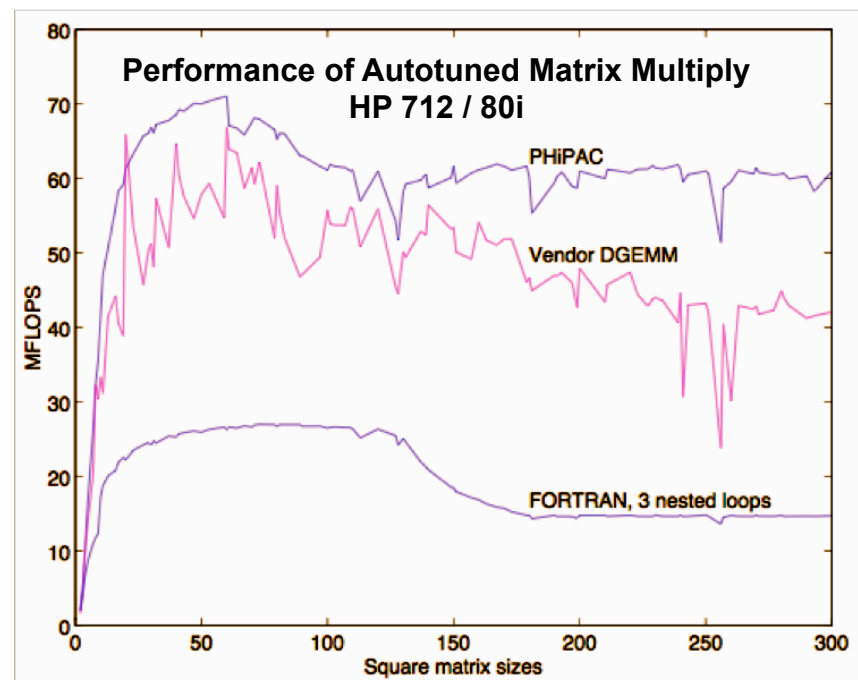
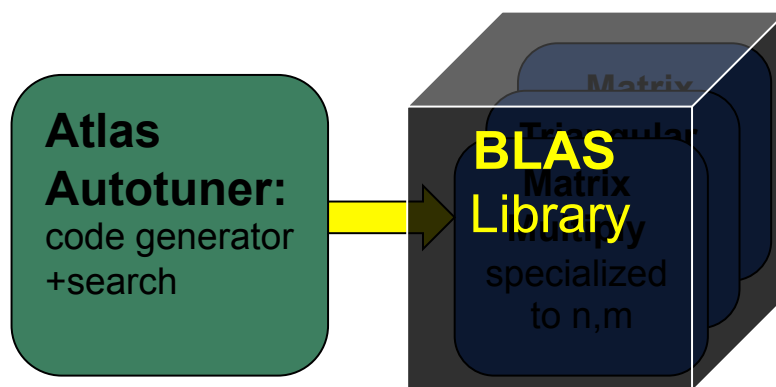
PIs: John Wawrzynek and Krste Asanovic, UC Berkeley





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



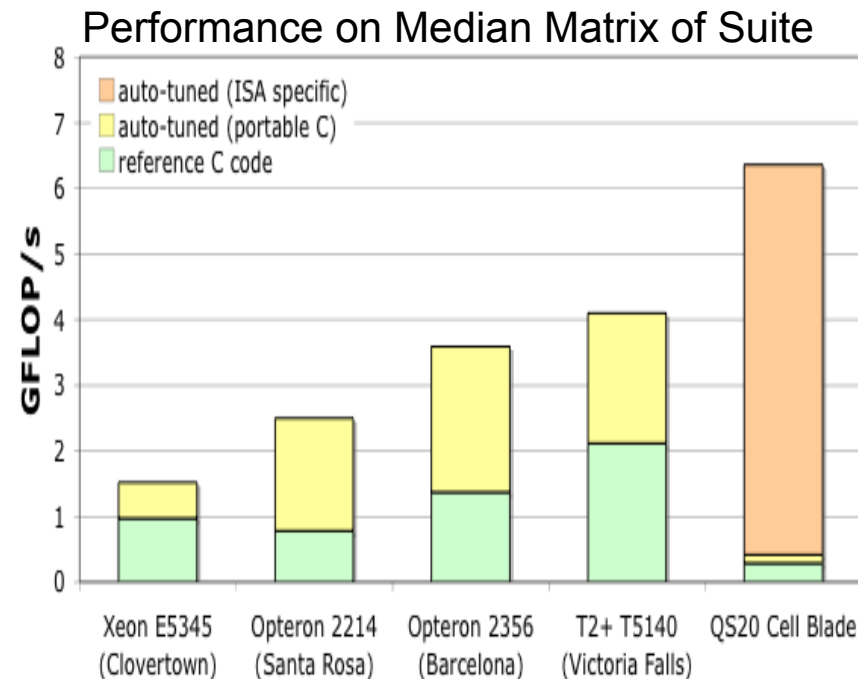
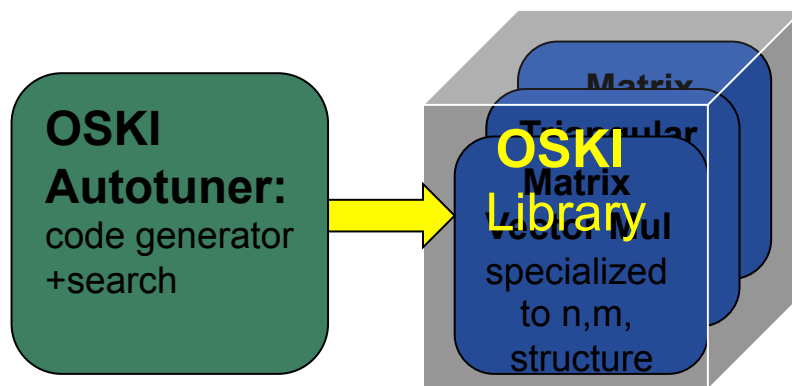
BLAS = Basic Linear Algebra Subroutine: matrix multiply, etc.





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
  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 Challenges

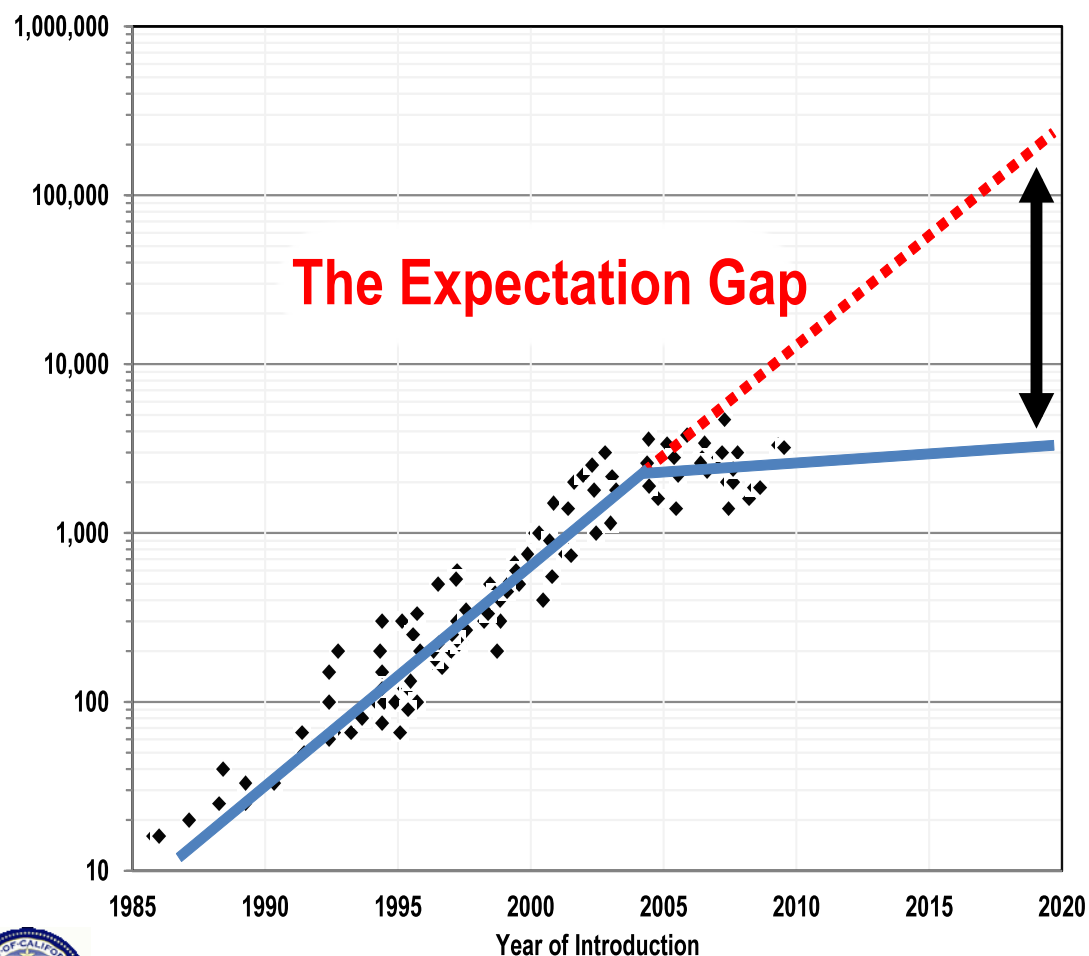
- 1) **Generality of autotuners** for DS CG or Compiler CG
- 2) **Irregular data structures**, e.g., sparse matrices, graphs
- 3) **Higher order functions**, e.g., stencils
- 4) **Larger kernels**, communication/synchronization avoidance
- 5) **Simplify writing of domain-specific code generators or automate in compilers**: teach applications to fish
- 6) **Portability**, at what level?
- 7) **Heterogeneity** of architectures
- 8) **Dynamic parallelism** from algorithms or hardware
- 9) **Multiple optimization criteria**: power (avoid overheating), energy (race to stop), performance
- 10) **Resilience**





Computing Growth is Not Just an Exascale Problem

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

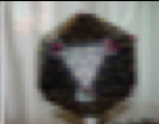




Autotuning Motif Coverage

(App importance: **Red Hot** → **Blue Cool**)

- Autotuning goal to cover the Motifs
- UC Berkeley Parlab work on several

		Embed	SPEC	DB	Games	ML	CAD	HPC	 Health	 Image	 Speech	 Music	 Browser
1	Finite State Mach.	Red	Red	Red	Yellow	Yellow	Yellow	Blue	Blue	Blue	Blue	Blue	Red
2	Circuits	Red	Blue	Green	Blue	Green	Blue	Blue	Blue	Blue	Blue	Blue	Red
3	Graph Algorithms	Red	Yellow	Yellow	Yellow	Red	Red	Blue	Red	Blue	Red	Green	Green
4	Structured Grid	Red	Red	Blue	Yellow	Blue	Blue	Red	Blue	Red	Blue	Blue	Blue
5	Dense Matrix	Red	Red	Yellow	Red	Red	Red	Red	Blue	Red	Red	Red	Blue
6	Sparse Matrix	Yellow	Yellow	Blue	Red	Red	Red	Red	Red	Blue	Blue	Red	Blue
7	Spectral (FFT)	Yellow	Blue	Blue	Yellow	Yellow	Yellow	Red	Blue	Green	Red	Red	Red
8	Dynamic Prog	Yellow	Blue	Red	Blue	Red	Red	Blue	Blue	Blue	Yellow	Blue	Red
9	Particle Methods	Blue	Yellow	Blue	Yellow	Blue	Blue	Red	Blue	Blue	Blue	Blue	Blue
10	Backtrack/ B&B	Blue	Blue	Yellow	Blue	Red	Red	Blue	Blue	Blue	Blue	Yellow	Blue
11	Graphical Models	Blue	Blue	Yellow	Blue	Red	Blue	Blue	Blue	Blue	Blue	Red	Blue
12	Unstructured Grid	Blue	Blue	Blue	Yellow	Yellow	Yellow	Red	Red	Blue	Blue	Red	Blue

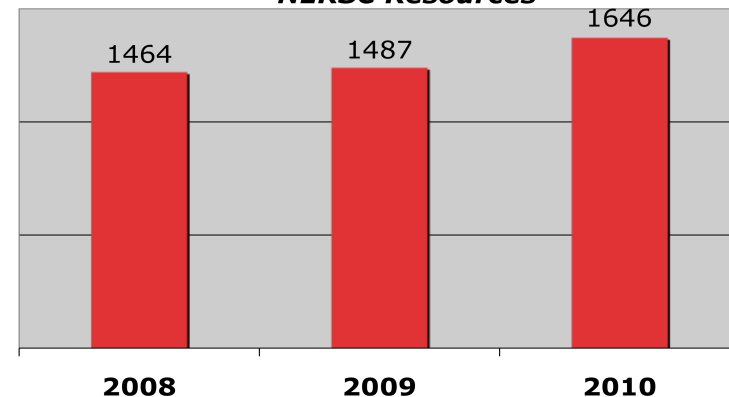


Measuring Efficiency

- 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



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!

