The Next Generation of High Performance Computing

William Gropp
www.cs.illinois.edu/~wgropp

Extrapolation is Risky

- 1989 – T – 23 years
  - Intel introduces 486DX
  - Eugene Brooks writes "Attack of the Killer Micros"
  - 4 years *before* TOP500
  - Top systems at about 2 GF Peak
- 1999 – T – 13 years
  - NVIDIA introduces its GPU (GeForce 256)
    - Programming GPUs still a challenge 13 years later
  - Top system – ASCI Red, 9632 cores, 3.2 TF Peak (about 3 GPUs in 2012)
  - MPI is 7 years old

HPC Today

- High(est)-End systems
  - 1 PF (10^{15} Ops/s) achieved on a few "peak friendly" applications
  - Much worry about scalability, how we're going to get to an ExaFLOPS
  - Systems are all oversubscribed
    - DOE INCITE awarded almost 900M processor hours in 2009; 1600M-1700M hours in 2010-2012; (big jump planned in 2013 – over 5B hours)
    - NSF PRAC awards for Blue Waters similarly competitive
- Widespread use of clusters, many with accelerators; cloud computing services
  - These are transforming the low and midrange
- Laptops (far) more powerful than the supercomputers I used as a graduate student

HPC in 2012

- Sustained PF systems
  - ♦ K Computer (Fujitsu) at RIKEN, Kobe, Japan (2011)
  - ♦ "Sequoia" Blue Gene/Q at LLNL
  - ♦ NSF Track 1 "Blue Waters" at Illinois
  - ♦ Undoubtedly others (China, …)
- Still programmed with MPI and MPI+other (e.g., MPI+OpenMP or MPI+OpenCL/CUDA)
  - ♦ But in many cases using toolkits, libraries, and other approaches
    - And not so bad – applications will be able to run when the system is turned on
  - ♦ Replacing MPI will require some compromise – e.g., domain specific (higher-level but less general)
    - Lots of evidence that fully automatic solutions won't work

HPC in 2020-2023

- Exascale systems are likely to have
  - ♦ Extreme power constraints, leading to
    - Clock Rates similar to today's systems
    - A wide-diversity of simple computing elements (simple for hardware but complex for software)
    - Memory per core and per FLOP will be much smaller
    - Moving data anywhere will be expensive (time and power)
  - ♦ Faults that will need to be detected and managed
    - Some detection may be the job of the programmer, as hardware detection takes power
  - ♦ Extreme scalability and performance irregularity
    - Performance will require enormous concurrency
    - Performance is likely to be variable
      - Simple, static decompositions will not scale
  - ♦ A need for latency tolerant algorithms and programming
    - Memory, processors will be 100s to 10000s of cycles away. Waiting for operations to complete will cripple performance

ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems

Peter Kogge, Editor & Study Lead
Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley Williams
Katherine Yelick

September 28, 2008

This work was sponsored by DARPA IPTO in the ExaScale Computing Study with Dr. William Harrod as Program Manager; AFRL contract number **FA8650-07-C-7724.** This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings

NOTICE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

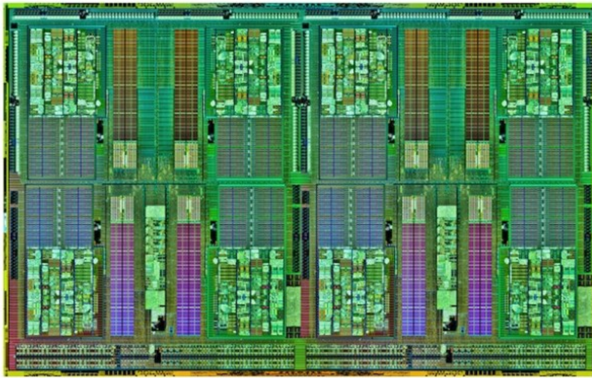
What Do Current Systems Tell Us?

- Examples of trends
  - ♦ Supercomputers: Blue Waters
  - ♦ Exploiting Commodity Computing: GPU Clusters
  - ♦ Post GPU: Radical architectures
- Parallelism is about getting *performance*
  - ♦ Productivity is important, but only if performance is achieved
  - ♦ All systems **already** “heterogeneous”
    - “Vector” instructions really a separate unit
  - ♦ Sustained performance is the goal

Blue Waters Science Team Characteristics

Science Area	Number of Teams	Codes	Structured Grids	Unstructured Grids	Dense Matrix	Sparse Matrix	N-Body	Monte Carlo	FFT	Significant I/O
Climate and Weather	3	CESM, GCRM, CM1, HOMME	X	X		X		X		
Plasmas/ Magnetosphere	2	H3D(M), OSIRIS, Magtail/ UPIC	X				X		X	X
Stellar Atmospheres and Supernovae	2	PPM, MAESTRO, CASTRO, SEDONA	X			X		X		X
Cosmology	2	Enzo, pGADGET	X			X	X			
Combustion/ Turbulence	1	PSDNS	X						X	
General Relativity	2	Cactus, Harm3D, LazEV	X			X				
Molecular Dynamics	4	AMBER, Gromacs, NAMD, LAMMPS			X		X		X	
Quantum Chemistry	2	SIAL, GAMESS, NWChem			X	X	X	X		X
Material Science	3	NEMOS, OMEN, GW, QMCPACK			X	X	X	X		
Earthquakes/ Seismology	2	AWP-ODC, HERCULES, PLSQR, SPECFEM3D	X	X			X			X
Quantum Chromo Dynamics	1	Chroma, MILC, USQCD	X		X	X	X		X	
Social Networks	1	EPISIMDEMICS								
Evolution	1	Eve								
Computer Science	1			X	X	X			X	X

Heart of Blue Waters: Two New Chips

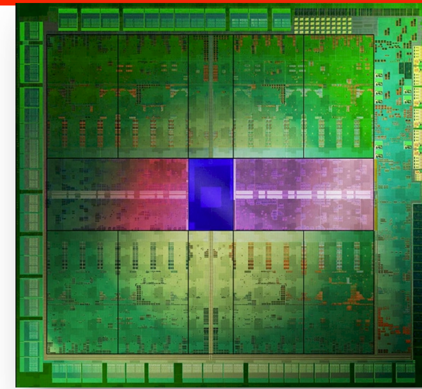


AMD Interlagos

157 GF peak performance

Features:

- 2.3-2.6 GHz
- 8 core modules, 16 threads
- On-chip Caches
  - L1 (I:8x64KB; D:16x16KB)
  - L2 (8x2MB)
- Memory Subsystem
  - Four memory channels
  - 51.2 GB/s bandwidth



NVIDIA Kepler

1,400 GF peak performance

Features:

- 15 Streaming multiprocessors (SMX)
  - SMX: 192 sp CUDA cores, 64 dp units, 32 special function units
  - L1 caches/shared mem (64KB, 48KB)
  - L2 cache (1536KB)
- Memory subsystem
  - Six memory channels
  - 180 GB/s bandwidth

Blue Waters and Titan Computing Systems

System Attribute	NCSA Blue Waters	ORNL Titan
Vendors	Cray/AMD/NVIDIA	Cray/AMD/NVIDIA
Processors	Interlagos/Kepler	Interlagos/Kepler
Total Peak Performance (PF)	11.9	>20
Total Peak Performance (CPU/GPU)	7.6/4.3	3/17
Number of CPU Chips	48,576	18,688
Number of GPU Chips	3,072	14,592
Amount of CPU Memory (TB)	1,510	688
Interconnect	3D Torus	3D Torus
Amount of On-line Disk Storage (PB)	26	20(?)
Sustained Disk Transfer (TB/sec)	>1	0.4-0.7
Amount of Archival Storage	300	15-30
Sustained Tape Transfer (GB/sec)	100	7

Blue Waters and K Computing Systems

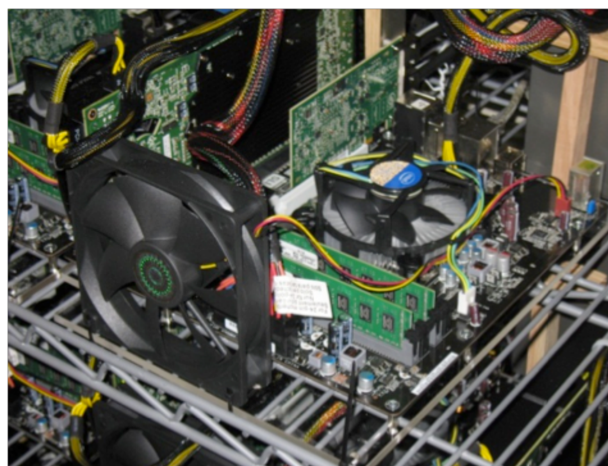
System Attribute	NCSA Blue Waters	RIKEN K
Vendors	Cray/AMD/NVIDIA	Fujitsu
Processors	Interlagos/Kepler	SPARC64 VIIIfx
Total Peak Performance (PF)	11.9	11.3
Total Peak Performance (CPU/GPU)	7.6/4.3	11.3/0.0
Number of CPU Chips	48,576	88,128
Number of GPU Chips	3,072	0
Amount of CPU Memory (TB)	1,510	1,410
Interconnect	3D Torus	6D Torus
Amount of On-line Disk Storage (PB)	26	11/30
Sustained Disk Transfer (TB/sec)	>1	?
Amount of Archival Storage	300	?
Sustained Tape Transfer (GB/sec)	100	?

Blue Waters and Sequoia Computing Systems

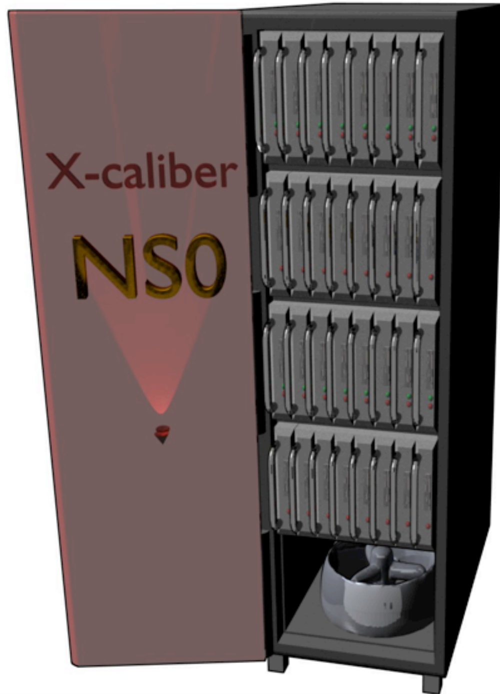
System Attribute	NCSA Blue Waters	LLNL Sequoia
Vendors	Cray/AMD/NVIDIA	IBM
Processors	Interlagos/Kepler	PowerPCA2 variant
Total Peak Performance (PF)	11.9	20.1
Total Peak Performance (CPU/GPU)	7.6/4.3	20.1/0.0
Number of CPU Chips (8, 16 cores/chip)	48,576	98,304
Number of GPU Chips	3,072	0
Amount of CPU Memory (TB)	1,510	1,572
Interconnect	3D Torus	5D Torus
Amount of On-line Disk Storage (PB)	26	50(?)
Sustained Disk Transfer (TB/sec)	>1	0.5-1.0
Amount of Archival Storage	300	?
Sustained Tape Transfer (GB/sec)	100	?

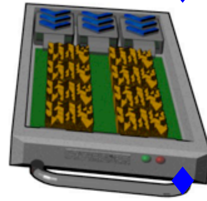
Another Example System

- 128 node GPU Cluster
- #3 on Green500 in 2010
- Each node has
  - One Core i3 530 2.93 GHz dual-core CPU
  - One Tesla C2050 GPU per node
- 33.62 TFLOPS on HPL (10x ASCI Red)
- 934 MFLOPS/Watt
- But how do you program it?



An Even More Radical System





- Rack Scale
  - Processing:128 Nodes, 1 (+) PF/s
  - Memory:
    - 128 TB DRAM
    - 0.4 PB/s Aggregate Bandwidth
  - NV Memory
    - 1 PB Phase Change Memory (addressable)
    - Additional 128 for Redundancy/RAID
  - Network
    - 0.13 PB/sec Injection, 0.06 PB/s Bisection

Deployment	Nodes	Topology	Compute	Mem BW	Injection BW	Bisection BW
Module	1	N/A	8 TF/s	3 TB/s	1 TB/s	N/A
Deployable Cage	22	All-to-All	176 TF/s	67.5 TB/s	22.5 TB/s	31 TB/s
Rack	128	Flat. Butterfly	1 PF/s	.4 PB/s	0.13 PB/s	0.066 PB/s
Group Cluster	512	Flat. Butterfly	4.1 PF/s	1.6 PB/s	0.52 PB/s	0.26 PB/s
National Resource	128k	Hier. All-to-All	1 EF/s	0.4 EB/s	0.13 EB/s	16.8 PB/s
Max Configuration	2048k	Hier. All-to-All	16 EF/s	6.4 EB/s	2.1 EB/s	0.26 EB/s

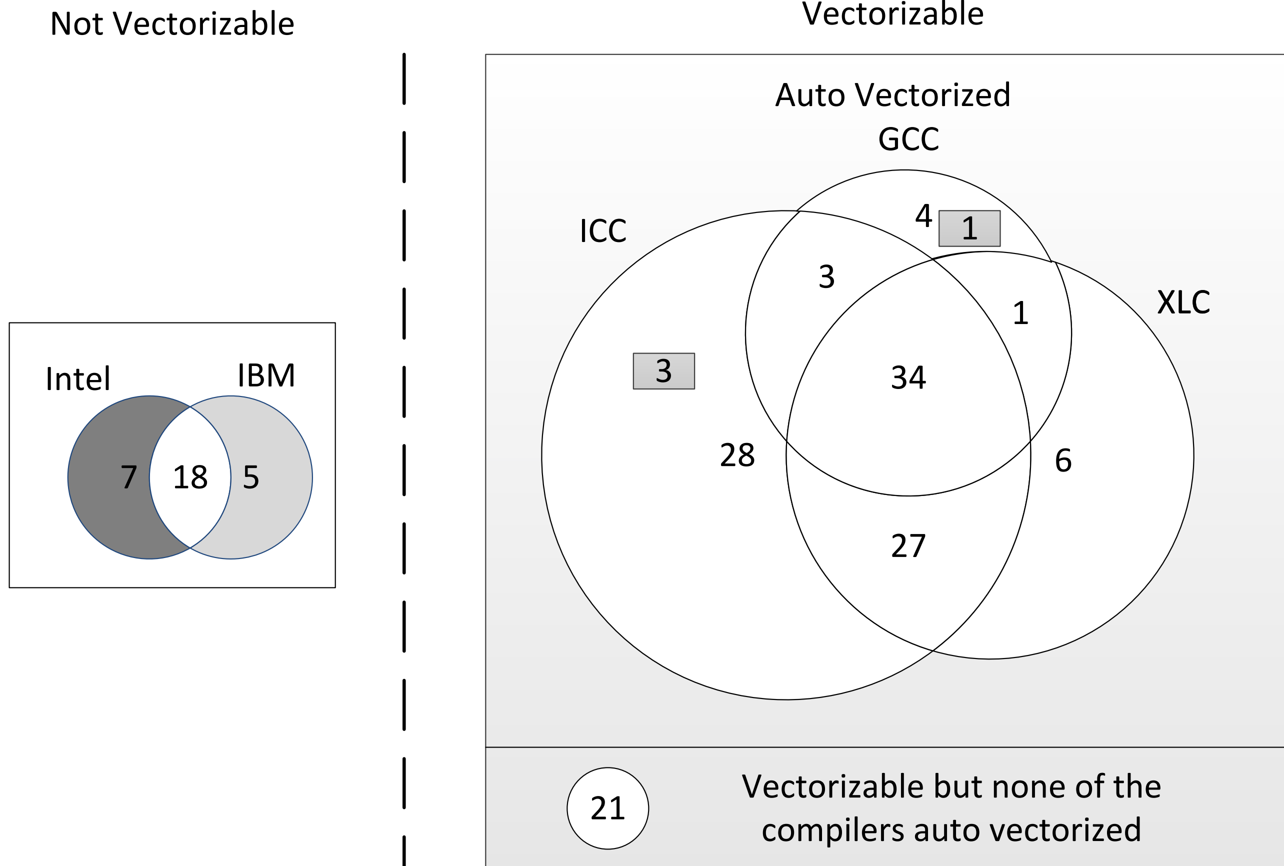
How Do We Make Effective Use of These Systems?

- Better use of our existing systems
  - Blue Waters will provide a sustained PF, but that typically requires ~10PF peak (BW over 11PF peak)
- Improve node performance
  - Make the compiler better
  - Give better code to the compiler
  - Match algorithms/data structures to real hardware
- Improve parallel performance/scalability
- Improve productivity of applications
  - Better tools and interoperable languages, not a (single) new programming language
- Improve algorithms wrt real hardware
  - Optimize for the real issues – data movement, power, resilience, …

Make the Compiler Better

- It remains the case that most compilers cannot compete with hand-tuned or autotuned code on simple code
  - Just look at dense matrix-matrix multiplication or matrix transpose
  - Try it yourself!
    - Matrix multiply on my laptop:
    - N=100 (in cache): 1818 MF (1.1ms)
    - N=1000 (not): 335 MF (6s)

How Good are Compilers at Vectorizing Codes?



S. Maleki, Y. Gao, T. Wong, M. Garzarán, and D. Padua. *An Evaluation of Vectorizing Compilers*. PACT 2011.

How Do We Change This?

- Short term: Help improve compilers by testing against best hand-tuned or auto-tuned code
  - ♦ Code for results on previous slide taken from Blue Waters project
- Medium term: Give "Better" code to the compiler
  - ♦ Augment (not replace) current programming languages to exploit advanced techniques for program optimization
    - Enable autotuning; specialized code generation
  - ♦ Challenge: Develop suitable performance abstractions, rather than prescriptive commands
  - ♦ Challenge: Overcome practical issues (e.g., correctness of multiple versions, debugging)

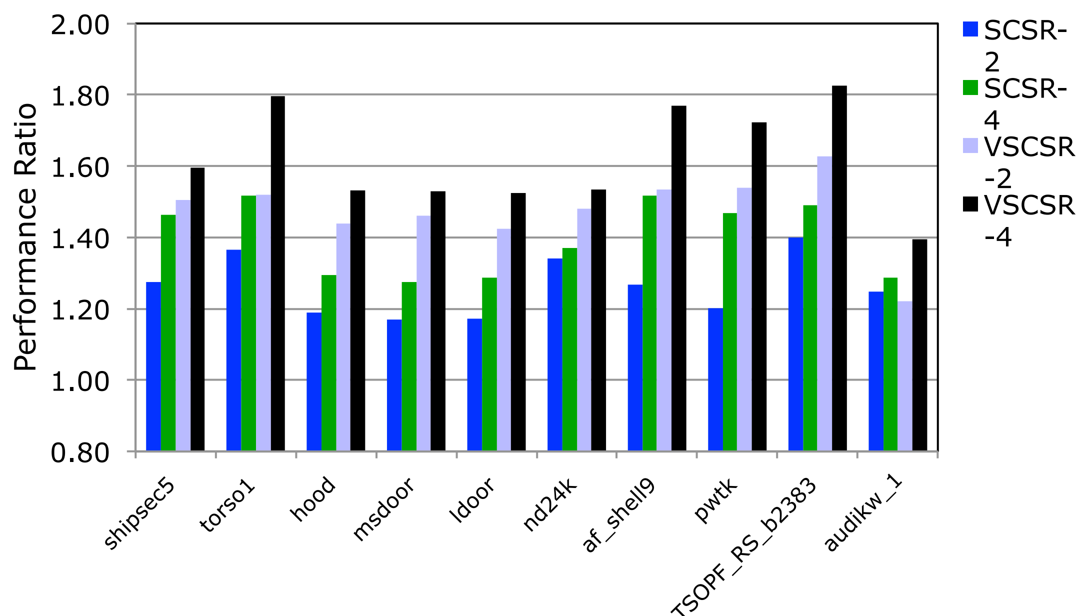
Better Algorithms and Data Structures

- Autotuning only offers the best performance with the given data structure and algorithm
  - That's a big constraint
- Processors include hardware to address performance challenges
  - "Vector" function units
  - Memory latency hiding/prefetch
  - Atomic update features for shared memory
  - Etc.

Sparse Matrix-Vector Multiply

Barriers to faster code

- "Standard" formats such as CSR do not meet requirements for prefetch or vectorization
- Modest changes to data structure enable both vectorization, prefetch, for 20-80% improvement on P7



Prefetch results in *Optimizing Sparse Data Structures for Matrix Vector Multiply*
http://hpc.sagepub.com/content/25/1/115

What Does This Mean For You?

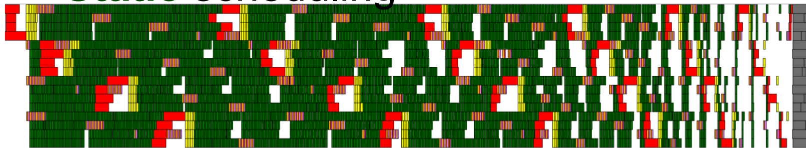
- It is time to rethink data structures and algorithms to match the realities of memory architecture
  - We have results for x86 where the benefit is smaller but still significant
  - Better match of algorithms to prefetch hardware is necessary to overcome memory performance barriers
- Similar issues come up with heterogeneous processing elements (someone needs to *design* for memory motion and concurrent and nonblocking data motion)

Processes and SMP nodes

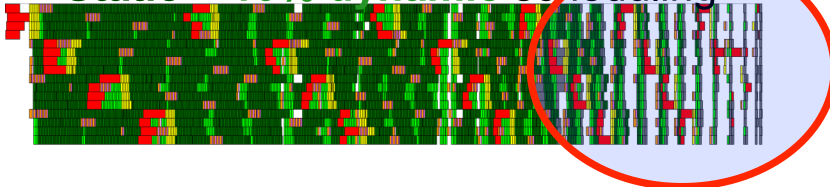
- HPC users typically believe that their code "owns" all of the cores all of the time
  - The reality is that was never true, but they did have all of the cores the same fraction of time when there was one core /node
- We can use a simple performance model to check the assertion and then use measurements to identify the problem and suggest fixes.
- Based on this, we can tune a state-of-the-art LU factorization….

Happy Medium Scheduling

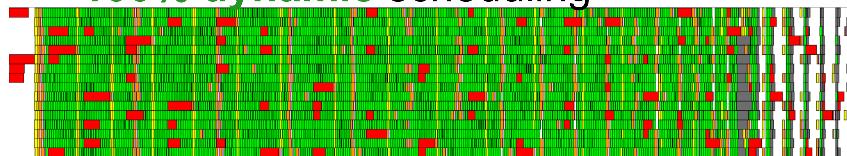
Static scheduling



Static + **10% dynamic** scheduling



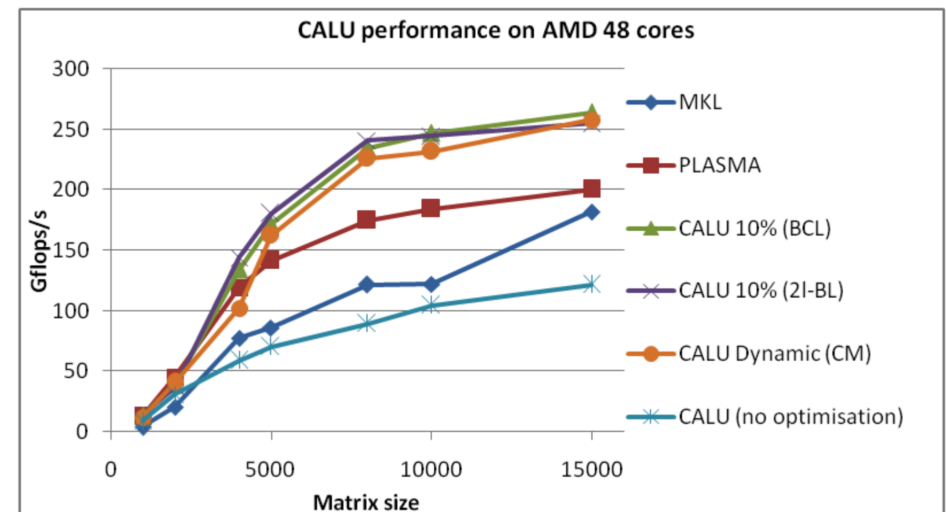
100% dynamic scheduling



time

Performance irregularities introduce load-imbalance.
Pure dynamic has significant overhead; pure static too much imbalance.
Solution: combined static and dynamic scheduling

Communication Avoiding LU factorization (CALU) algorithm, S. Donfack, L .Grigori, V. Kale, WG, IPDPS '12

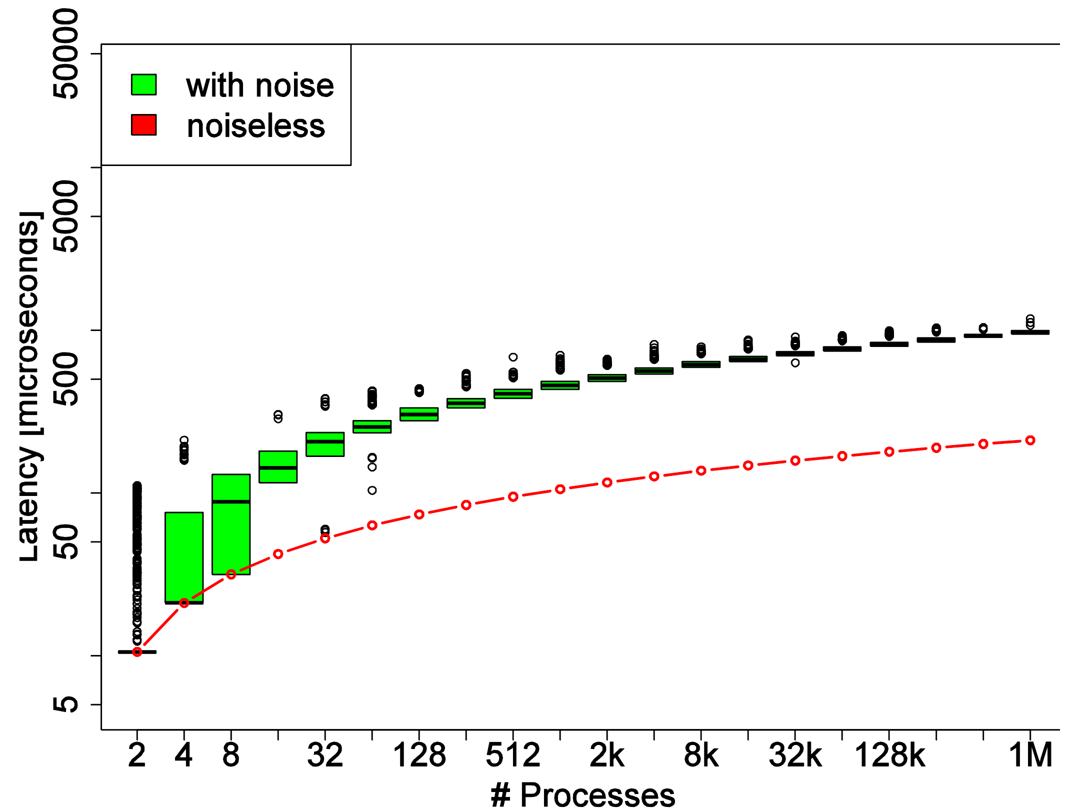


Scary Consequence: Static data decompositions *will not work at scale.*

Corollary: programming models with static task models *will not work at scale*

Synchronization and Performance Irregularities

- Communication dependencies between processes or threads can introduce cascading delays if performance is "irregular"
- Total delay related to
  - Local delay
  - Probability of delay
  - Number of threads
- Probability of impact goes from near 0 on small system to near 1 on large system
- Not just an "OS Noise" issue



"Characterizing the Influence of System Noise on Large-Scale Applications by Simulation," Hoefler, Schneider, Lumsdaine; Best Paper, SC10

The Problem is *Blocking* Operations

- Simple, data-parallel algorithms easy to reason about but inefficient
  - True for decades, but ignored (memory)
- One solution: fully asynchronous methods
  - Very attractive, yet efficiency is low and there are good reasons for that
  - Blocking can be due to fully collective (e.g., Allreduce) or neighbor communications (halo exchange)
  - Can we save methods that involve global, synchronizing operations?

PARALLEL@ILLINOIS

Saving Allreduce

- One common suggestion is to avoid using Allreduce
  - ♦ But algorithms with dot products are among the best known
  - ♦ Can sometimes aggregate the data to reduce the number of separate Allreduce operations
  - ♦ But better is to reduce the impact of the synchronization by hiding the Allreduce behind other operations (in MPI, using MPI_Iallreduce)
- We can adapt CG to nonblocking Allreduce with some added floating point (but perhaps little time cost)

The Conjugate Gradient Algorithm

- While (not converged)

```
  niters += 1;
  s     = A * p;
  t     = p' *s;
  alpha = gmma / t;
  x     = x + alpha * p;
  r     = r - alpha * s;
  if rnorm2 < tol2 ; break ; end
  z     = M * r;
  gmmaNew = r' * z;
  beta  = gmmaNew / gmma;
  gmma = gmmaNew;
  p     = z + beta * p;
end
```

The Conjugate Gradient Algorithm

- While (not converged)

```
  niters += 1;
  s     = A * p;
  t     = p' *s;
  alpha = gmma / t;
  x     = x + alpha * p;
  r     = r - alpha * s;
  if rnorm2 < tol2 ; break ; end
  z     = M * r;
  gmmaNew = r' * z;
  beta  = gmmaNew / gmma;
  gmma = gmmaNew;
  p     = z + beta * p;
end
```

CG Reconsidered

- By reordering operations, nonblocking dot products (MPI_Iallreduce in MPI-3) can be overlapped with other operations
- Trades extra local work for overlapped communication
  - ♦ On a pure floating point basis, the nonblocking version requires 2 more DAXPY operations
  - ♦ A closer analysis shows that some operations can be merged
- *More work does not imply more time*

What's Different at Peta/Exascale

- Performance Focus
  - Only a little – basically, the resource is expensive, so a premium placed on making good use of resource
  - Quite a bit – node is more complex, has more features that must be exploited
- Scalability
  - Solutions that work at 100-1000 way often inefficient at 100,000-way
  - Some algorithms scale well
    - Explicit time marching in 3D
  - Some don't
    - Direct implicit methods
  - Some scale well for a while
    - FFTs (communication volume in Alltoall)
  - Load balance, latency are critical issues
- Fault Tolerance becoming important
  - Now: Reduce time spent in checkpoints
  - Soon: Lightweight recovery from transient errors

Preparing for the Next Generation of HPC Systems

- Better use of existing resources
  - Performance-oriented programming
  - Dynamic management of resources at all levels
  - Embrace hybrid programming models (you have already if you use SSE/VSX/OpenMP/…)
- Focus on results
  - Adapt to available network bandwidth and latency
  - Exploit I/O capability (available space crew faster than processor performance!)
- Prepare for the future
  - Fault tolerance
  - Hybrid processor architectures
  - Latency tolerant algorithms
  - Data-driven systems

Recommended Reading

- Bit reversal on uniprocessors (Alan Karp, SIAM Review, 1996)
- Achieving high sustained performance in an unstructured mesh CFD application (W. K. Anderson, W. D. Gropp, D. K. Kaushik, D. E. Keyes, B. F. Smith, Proceedings of Supercomputing, 1999)
- Experimental Analysis of Algorithms (Catherine McGeoch, Notices of the American Mathematical Society, March 2001)
- Reflections on the Memory Wall (Sally McKee, ACM Conference on Computing Frontiers, 2004)

Thanks

- Torsten Hoefler
  - Performance modeling lead, Blue Waters; MPI datatype
- David Padua, Maria Garzaran, Saeed Maleki
  - Compiler vectorization
- Dahai Guo
  - Streamed format exploiting prefetch, vectorization, GPU
- Vivek Kale
  - SMP work partitioning
- Hormozd Gahvari
  - AMG application modeling
- Marc Snir and William Kramer
  - Performance model advocates
- Abhinav Bhatele
  - Process/node mapping
- Van Bui
  - Performance model-based evaluation of programming models
- Funding provided by:
  - Blue Waters project (State of Illinois and the University of Illinois)
  - Department of Energy, Office of Science
  - National Science Foundation

SC12

Salt Lake City, Utah

November 10-16, 2012

SC13 Denver, CO 2013

sighpc

ACM Special Interest Group on High Performance Computing

parallel

COMPUTING INSTITUTE

The power of many working as one.