Performance Modeling as the Key to Extreme Scale Computing

William Gropp

www.cs.illinois.edu/~wgropp

PARALLEL@ILLINOIS

Performance is Key

- Parallelism is (usually) used to get more performance
  - How do you know if you are making good (not even best) use of a parallel system?
- Even measurement-based approaches can be (and all to often are) performed without any real basis of comparison
  - The key questions are
    - Where is most of the time spent?
    - What is the achieveable performance, and how do I get there?
  - This latter is often overlooked, leading to erroneous conclusions based on the (immature) state of compiler / runtime / code implementations

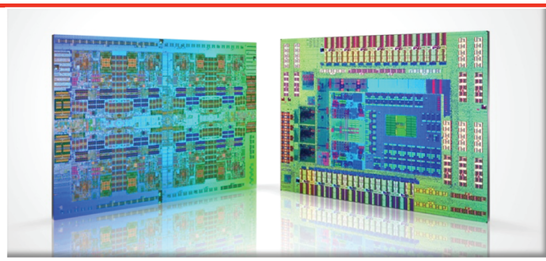
How Do We Know if there is a Performance Problem?

- My application scales well!
  - So what!
    - Is it efficient?
    - Making the scalar code more efficient *decreases* scalability
  - How can we *know?*
  - To what do we compare?

Tuning A Parallel Code

- Typical Approach
  - Profile code. Determine where most time is being spent
  - Study code. Measure absolute performance, look at performance counters, compare FLOP rates
  - Improve code that takes a long time, reduce time spent in “unproductive” operations
- Why this isn’t the right Approach:
  - How do you know when you are done?
  - How do you know how much performance improvement you can obtain?
- Why is it hard to know?

Heart of Blue Waters: Two New Chips



Power7 Chip	**PERCS Hub Chip**
Up to 256 GF peak performance	*1.128 TB/s total bandwidth*

Power7 Chip

Up to 256 GF peak performance

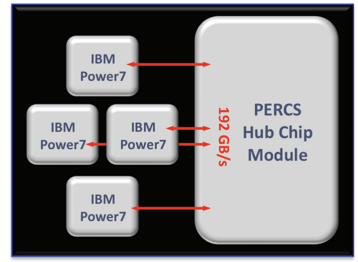
- 3.5–4.0 GHz
- Up to 8 cores, 32 SMT threads
- Caches
  - L1 (2x64 KB), L2 (256 KB),
  - L3 (32 MB, complex policy)
- Memory Subsystem
  - Two memory controllers
  - 128 GB/s memory bandwidth

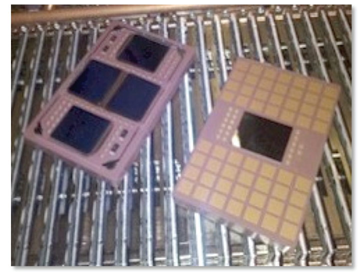
PERCS Hub Chip

1.128 TB/s total bandwidth

Connections:
- 192 GB/s QCM connection
- 896 GB/s to other QCMs
- 40 GB/s general purpose I/O

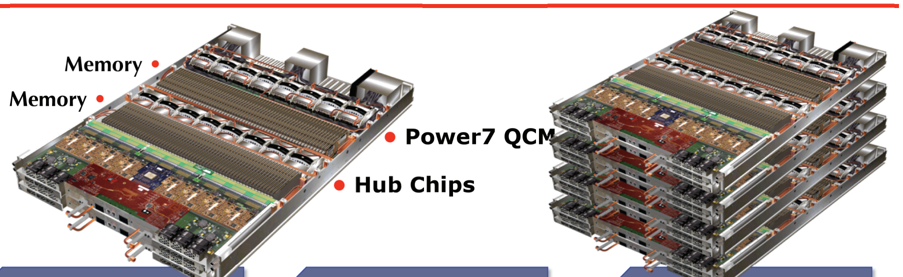
Quad-chip Module (QCM)





- Quad-chip Module
  - Four Power7 chips
    - 32 cores, 128 threads
    - ~1 TF peak performance
    - 128 GBytes memory
      - 512 GB/sec memory bw
  - One Hub Chip
    - 1.128 TB/sec bandwidth
      - 192 GB/s QCM connection
      - 896 GB/s to other QCMs
        - 336 GB/s to other QCMs in same drawer
        - 240 GB/s to QCMs in other drawers in same supernode
        - 320 GB/s to QCMs in other supernodes
      - 40 GB/s general purpose I/O

IH Server Node (Drawer) and Supernode

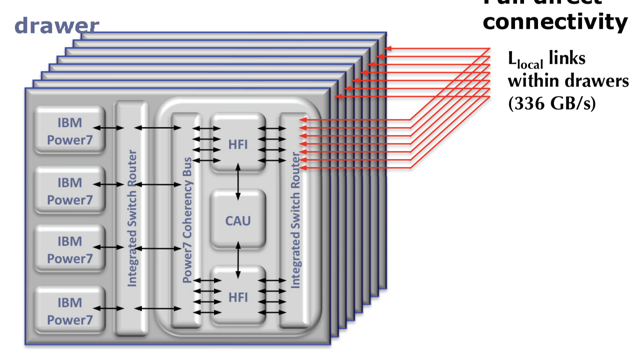


Specifications:
- 8 Quad-chip Modules
  - Up to 8 Tflops
- 1 TByte of memory
  - 4 TB/s memory
- 8 Hub Chips
  - 9 TB/sec/Hub

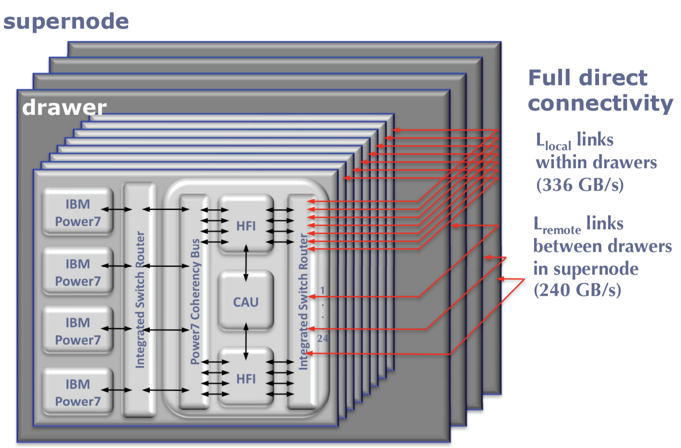
Packaging:
- 2U Drawer
- 39″w x 72″d
- > 300 lbs.
- *Fully water cooled (QCMs, Hubs, and Memory)*

Specifications:
- 4 IH Server Nodes
  - Up to 32 Tflops
  - 4 TBytes of memory
    - 16 TB/s
- 32 Hub Chips
  - 36 TB/sec

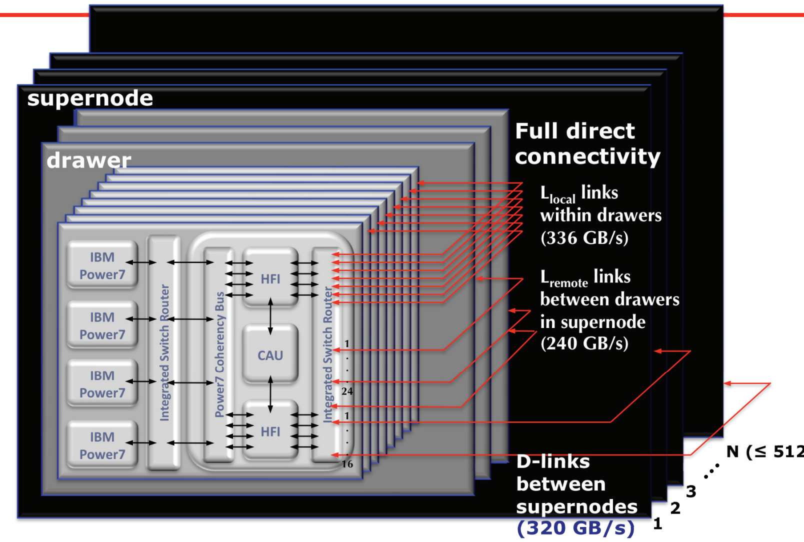
Logical View of Blue Waters Interconnect



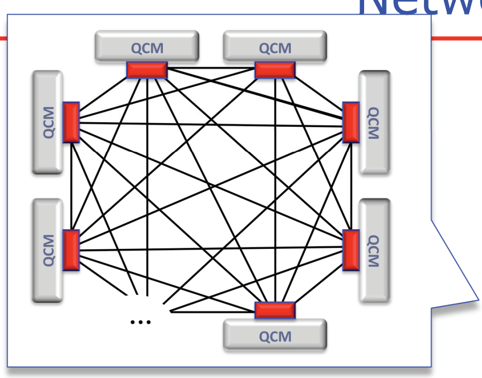
Logical View of Blue Waters Interconnect



Logical View of Blue Waters Interconnect



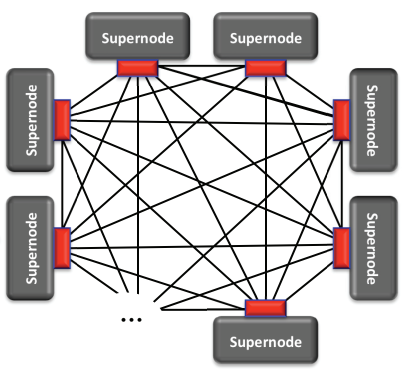
Two-level (L, D) Direct-connect Network



Each Supernode = 32 QCMs
(4 Drawers x 8 SMPs/Drawer)

Fully Interconnected with L_{local} and L_{remote} Links

Result: Very low hardware latency Very high bandwidth



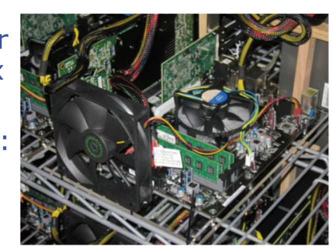
Blue Waters = 320 Supernodes
(40 BBs x 8 SNs/BB)

Fully Interconnected with D Links

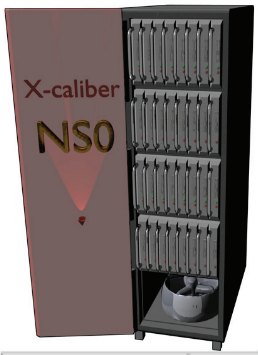
But complex, nonuniform network

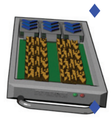
Another Example System

- 128 node GPU Cluster
- #3 on Green500
- Each node has
  - One Core i3 530 2.93 GHz dual-core CPU
  - One Tesla C2050 GPU per node
- 33.62 TFLOPS on HPL
- 934 MFLOPS/Watt
- How can we *engineer* codes for performance on these complex systems?
- And an exercise for the viewer: what do performance models tell you about the CPU/GPU comparisons you see?



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

NNSA

Sandia National Laboratories

Why Performance Modeling?

- What is the goal?
  - It is *not* precise predictions
  - It *is* insight into whether a code is achieving the performance it could, and if not, how to fix it
- Performance modeling can be used
  - To estimate the baseline performance
  - To estimate the potential benefit of a nontrivial change to the code
  - To identify the critical resource

1867

What do I mean by Performance Modeling?

- Actually two different models
  - First, an analytic expression based on the application code
  - Second, an analytic expression based on the application's *algorithm* and data structures
- Note that a series of measurements from benchmarks are *not* a performance model
- Why this sort of modeling
  - The obvious: extrapolation to other systems, such as scalability in nodes or different interconnect
  - Also: comparison of the two models with observed performance can identify
    - Inefficiencies in compilation/runtime
    - Mismatch in developer expectations

1867

Different Philosophies for Performance Models

- Simulation:
  - Very accurate prediction, little insight
- Traditional Performance Modeling (PM):
  - Focuses on accurate predictions
  - Tool for computer scientists, not application developers
- PM as part of the software engineering process (our view)
  - PM for design, tuning and optimization
  - PMs are developed with algorithms and used in each step of the development cycle
  - ➢ Performance Engineering

1867

Our Methodology

- Combine analytical methods and performance measurement tools
  - Programmer specifies parameterized expectation
    - E.g., $T = a+b*N^3$
  - Estimate coefficients with *appropriate* benchmarks
  - We derive the scaling analytically and fill in the constants with empirical measurements
  - Focus on upper and lower bounds rather than precise predictions
- Models must be as simple and effective as possible
  - Simplicity increases the insight
  - Precision needs to be just good enough to drive action.
- An example: Sparse matrix-vector multiply

Sparse Matrix-Vector Product

- Common operation for optimal (in floating-point operations) solution of linear systems
- Sample code (common CSR format):

```
for row=1,n
    m   = i[row] - i[row-1];
    sum = 0;
    for k=1,m
        sum += *a++ * x[*j++];
    y[i] = sum;
```

- Data structures are a[nnz], j[nnz], i[n], x[n], y[n]

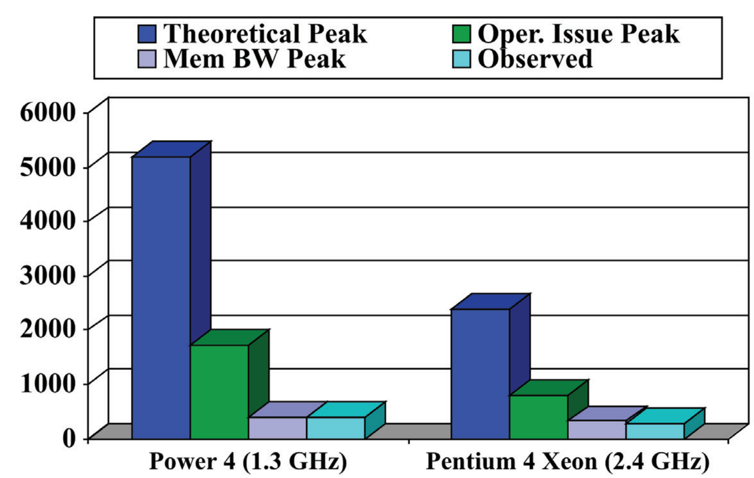
Simple Performance Analysis

- Memory motion:
  - nnz (sizeof(double) + sizeof(int)) + n (2*sizeof(double) + sizeof(int))
  - Assume a perfect cache (never load same data twice)
- Computation
  - nnz multiply-add (MA)
- Roughly 12 bytes per MA
- Typical node can move 1-4 bytes/MA
  - *Maximum* performance is 8-33% of peak
  - Use STREAM benchmark to get sustained memory bandwidth
- Similar analysis gives bound based on instruction issue rate
- Implementation improvements (tricks) cannot improve on these limits
- W. K. Anderson, William D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. Achieving high sustained performance in an unstructured mesh CFD application, SC'99 (Gordon Bell Prize)

Realistic Measures of Peak Performance

Sparse Matrix Vector Product

One vector, matrix size, m = 90,708, nonzero entries nz = 5,047,120



Note excellent match to simple performance model. Current systems show similar results (but there is a difference to be discussed later)

Thanks to Dinesh Kaushik; ORNL and ANL for compute time

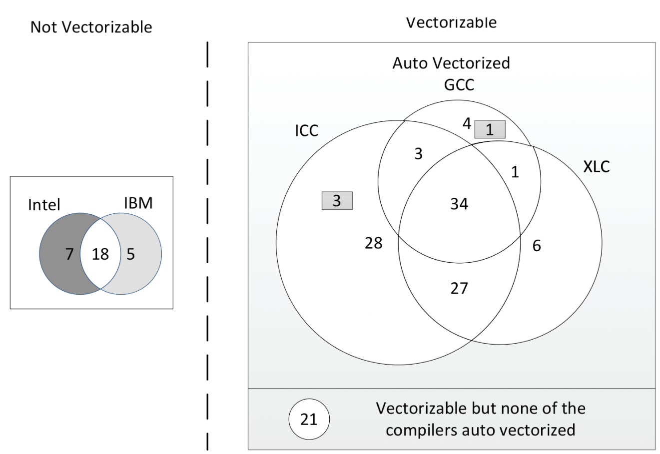
But the problem is so big!

- Real applications are much larger – isn't it hard to do this for the entire application?
- Yes, but it doesn't matter for runnable apps. Look at the parts that take the most time. Break the problem into digestible parts
- Contributions to performance issues from:
  - Single thread and node performance
  - Node and the Network
  - Placement in the Network

Utilizing the Processor

- Note rapidly growing numbers of functional units – Power7 has 2 multiply-add units per core; BG/Q has 4, accessed through "vector" instructions
- How do we know how well we are doing?
- How do we know how well the compiler is doing?
- We can model the expected performance, including vectorization!
- Using the model, we can also identify where manually applying well-known transformations will help
- Also identifies where extra constraints, such as alignment restrictions, may inhibit use of vectorization

How Good are Compilers at Vectorizing Codes?



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

Media Bench II Applications

Appl	XLC	ICC	GCC	XLC	ICC	GCC
	Automatic			Manual		
JPEG Enc	-	1.33	-	1.39	2.13	1.57
JEPG Dec	-	-	-	-	1.14	1.13
H263 Enc	-	-	-	1.25	2.28	2.06
H263 Dec	-	-	-	1.31	1.45	-
MPEG2 Enc	-	-	-	1.06	1.96	2.43
MPEG2 Dec	-	-	1.15	1.37	1.45	1.55
MPEG4 Enc	-	-	-	1.44	1.81	1.74
MPEG4 Dec	-	-	-	1.12	-	1.18

Table shows **whole program speedups** measured against unvectorized application

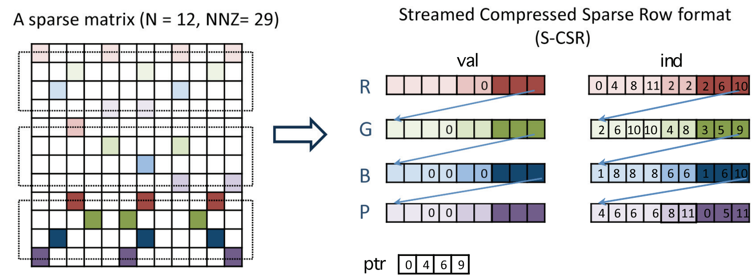
S. Maleki, Y. Gao, T. Wong, M. Garzarán, and D. Padua. An Evaluation of Vectorizing Compilers. In preparation. 2011.

Processes and Memory

- For many computations, sustained memory performance is the limiting resource
  - As in sparse matrix-vector multiply
- What is the appropriate sustained rate?
  - Memory bus bandwidth is nearly irrelevant – it is the sustained rate that is usually important
  - What about other ways to increase effective sustained performance, such as prefetch?
- Prefetch hardware can detect regular accesses and prefetch data, making use of otherwise idel memory bus time.
  - However, the hardware must be presented with enough independent data streams

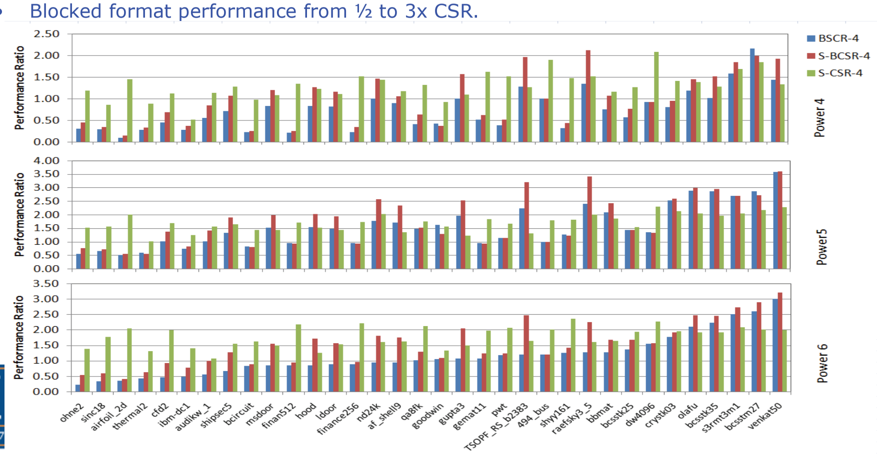
Streamed Compressed Sparse Row (S-CSR) format

- S-CSR format partitions the sparse matrix into blocks along rows with size of bs. Zeros are added in to keep the number of elements the same in each row of a blockThe first rows of all blocks are stored first, then second, third … and bs-th rows.
- For the sample matrix in the following Figure, NNZ = 29. Using a block size of bs = 4, it generates four equal length streams R, G, B and P. This new design only adds 7 zeros every 4 rows.



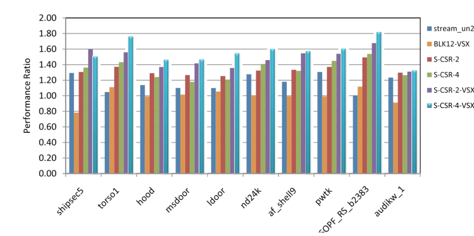
Performance Ratio Compared to CSR Format

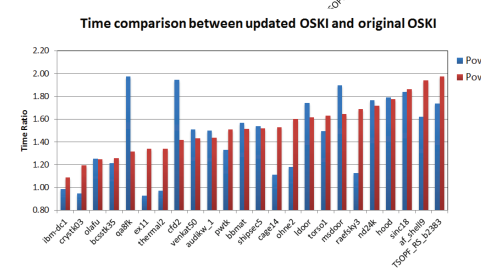
- S-CSR format is better than CSR format for all (on Power 5 and 6) or Most (on Power 4) matrices
- S-BCSR format is better than BCSR format for all (on Power 6) or Most (on Power 4 and 5) matrices
- Blocked format performance from ½ to 3x CSR.



Combining With Other Optimizations

- We can further modify the S-CSR and S-BCSR to match the requirements for vectorization
- We can use OSKI to optimize "within the loops"



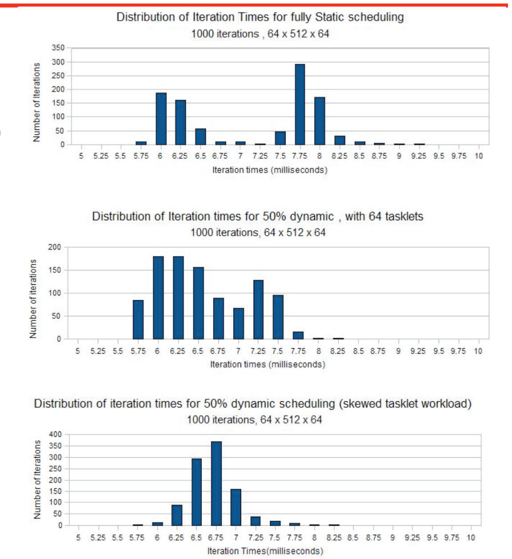


Processes and SMP nodes

- 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.
- Consider a simple Jacobi sweep on a regular mesh, with every core having the same amount of work. How are run times distributed?

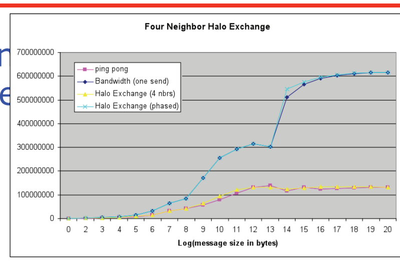
Sharing an SMP

- Having many cores available makes everyone think that they can use them to solve other problems ("no one would use all of them all of the time")
- However, compute-bound scientific calculations are often *written* as if all compute resources are owned by the application
- Such *static* scheduling leads to performance loss
- Pure dynamic scheduling adds overhead, but is better
- Careful mixed strategies are even better
- Recent results give 10-16% performance improvements on large, scalable systems
- Thanks to Vivek Kale



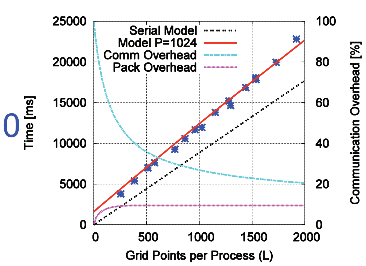
Processes and the Network

- How relevant is ping-pong bandwidth and real system?
- What are the correct parameters?
  - Model the real system, but abstractly
  - For Blue Gene, must model independent communication links
  - Impacts choice of communication algorithm (many benchmarks do not provide a relevant measurement)
- Data copies and MPI datatypes
  - How do you decide whether to even carry out the experiment?



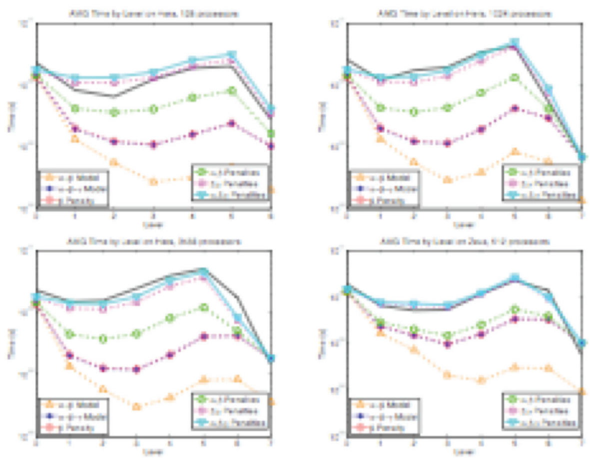
Model-guided Optimization

- Application is MILC, a lattice QCG code
- Analytic model showed possible improvement of 12% by eliminating the pack before communicating
- Torsten Hoefler implemented and analyzed in EuroMPI'10
  - Demonstrated benefit of up to **18%**
- Next bottleneck: CG phase
  - Investigating use of nonblocking collectives in a modified CG
  - Also model-driven



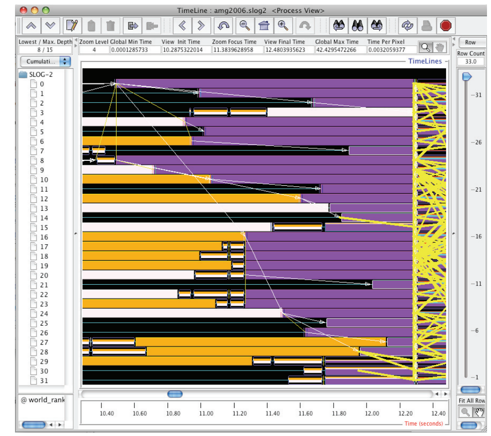
AMG Performance Model

- What if a model is too difficult? We can establish upper and lower bounds and compare performance
- Includes contention, bandwidth, multicore penalties
- 82% accuracy on Hera, 98% on Zeus
- Gahvari, Baker, Schulz, Yang, Jordan, Gropp (earlier this week)



How often do you hear "MPI Communication is too Slow"

- Often the real problem is that some process is "late" to a collective call or some send or receive is issued late
- "Fix" (used in PETSc and FPMPI2)
  - Test using
    - MPI_Barrier(comm) MPI_Allreduce(…,comm);
  - If Barrier time is too long (what's that), *hypothesis* is that there is load imbalance



Not Just Collectives

- So why do people see slow communication with regular mesh codes?
- One common culprit is the mapping of process topology to physical topology (network interconnect)
  - Note that this may be quite complex
  - We have used modeling to determine that a certain kind of random mapping is often preferable for Blue Waters
  - *Avoiding hot-spots on two-level direct networks,* Abhinav Bhatele, Nikhil Jain, William Gropp and Laxmikant V. Kale, submitted
- One common case is a halo exchange…

Halo Exchange on BG/P and Cray XT4

- 2048 doubles to each neighbor
- Rate is MB/Sec (for all tables)

BG/P	4 Neighbors		8 Neighbors	
	Irecv/Send	Irecv/Isend	Irecv/Send	Irecv/Isend
World	208	328	184	237
Even/Odd	219	327	172	243
Cart_create	301	581	242	410

Cray XT4	4 Neighbors			8 Neighbors	
	Irecv/Send	Irecv/Isend	Phased	Irecv/Send	Irecv/Isend
World	311	306	331	262	269
Even/Odd	257	247	279	212	206
Cart_create	265	275	266	236	232

Discovering Performance Opportunities

- Lets look at a single process sending to its neighbors.
- Based on our performance model, we *expect* the rate to be roughly twice that for the halo (since this test is only sending, not sending and receiving)

System	4 neighbors		8 Neighbors	
		Periodic		Periodic
BG/L	488	490	389	389
BG/L, VN	294	294	239	239
BG/P	1139	1136	892	892
BG/P, VN	468	468	600	601
XT3	1005	1007	1053	1045
XT4	1634	1620	1773	1770
XT4 SN	1701	1701	1811	1808

Discovering Performance Opportunities

- Ratios of a single sender to all processes sending (in rate)
- *Expect* a factor of roughly 2 (since processes must also receive)

System	4 neighbors		8 Neighbors	
		Periodic		Periodic
BG/L	2.24		2.01	
BG/L, VN	1.46		1.81	
BG/P	3.8		2.2	
BG/P, VN	2.6		5.5	
XT3	7.5	8.1	9.08	9.41
XT4	10.7	10.7	13.0	13.7
XT4 SN	5.47	5.56	6.73	7.06

- BG gives roughly double the halo rate. XTn is much higher
  - It should be possible to improve the halo exchange on the XT by scheduling the communication
  - Or improving the MPI implementation

Summary

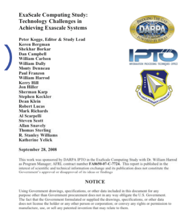
- Isn't this just a collection of tricks?
- Yes and no
  - Yes, a number of different approaches have been applied
  - No, the same quantitative approach, based on getting performance estimates for the resources under consideration and emphasizing a simple model that estimates *bounds*, is applied
  - **Quantitative Thinking**
    - **… must be based on having a hypothesis (model), not just measurements**

Performance Models Provide Insight

- SpMV, compiler vectorization
  - Model identifies limits of achievable performance
- Using prefetch in SpMV
  - Abstract model based on hardwaree identifies opportunity, led to new algorithm
- Jitter and adapting to runtime
  - Simple performance model identifies gap in achieved performance, leading to new approaches
- Using MPI Datatypes
  - Simple model suggests benefit; results show either success or problems in MPI implementation
- Topology
  - Simple model identifies performance gaps, even when multiple communication links involves

Why is Performance Modeling the Key to Extreme Scale?

- Measuring yesterday's applications, even with today's runtimes, is often irrelevant
  - ♦ Look at some of the CPU/GPU comparison (Vuduc et al)
- Focus on *achievable* performance at scale
  - ♦ Architectures are changing rapidly
    - Further reduces value of measurements on existing codes
  - ♦ Models permit quantitative evaluation of different approaches and a priori estimation of possible benefit to a major change
  - ♦ Only way to evaluate radical (and necessary!) architectural changes!


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.

Thanks

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