Engineering for Performance in High Performance Computing

William Gropp

www.cs.illinois.edu/~wgropp

PARALLEL@ILLINOIS

Doesn't Everyone Already Do This?

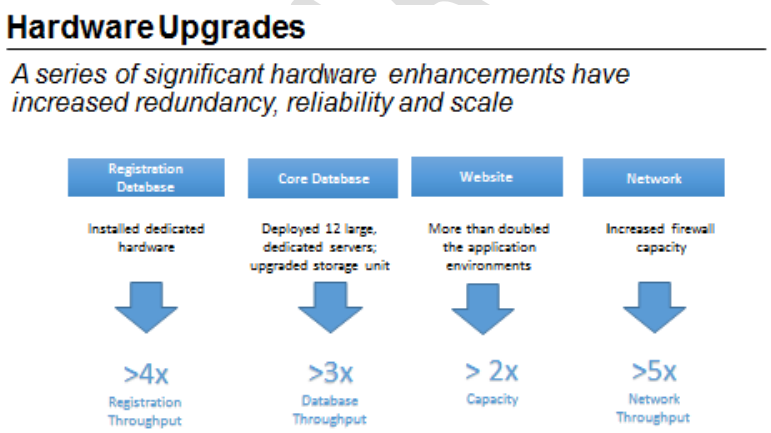
- Doesn't everyone design for performance?
  - ♦ Sadly, no
- Why?
  - ♦ Not always necessary (people time more important than machine time)
  - ♦ Too hard
    - Performance hard to predict
  - ♦ Can fix after the fact using tuning tools
- Why isn't that good enough?

Practical Performance Engineering is Essential

- One recent high-profile performance disaster: the US Affordable Care Act ("Obamacare") Web site
  - See one analysis at http://apmblog.compuware.com/2013/10/22/diagnosing-obamacare-website-performance-issues-with-apm-tools/
- A post mortem analysis (but many problem predictable a priori)
  - E.g., Points out loads 55 JS and 11 CSS (!!) files just to display registration page

Would Simple Performance Modeling Have Helped?

- Fixes to ACA Website should have been predictable in advance
- Initial report hints at that
  - http://www.hhs.gov/digitalstrategy/sites/digitalstrategy/files/pdf/healthcare.gov-progress-report.pdf



- An example of where we should be: Performance Evaluation and Model Checking Join Forces
  - http://cacm.acm.org/magazines/2010/9/98021-performance-evaluation-and-model-checking-join-forces/fulltext

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 achievable 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?

Systems are Increasingly Complicated

- End of Denard scaling, slow down in reduction of feature size, stagnation in latency imply improvements come from specializations, other tradeoffs
- Example: Blue Waters
  - 22,640 nodes with 2 AMD Interlagos
  - 4,224 nodes with 1 AMD Interlagos, 1 NVIDIA Kepler K20X
  - 24x24x24 mesh topology
  - I/O system with intermingled service nodes
  - Hierarchical storage system (1.5PB DRAM, 26 PB disk, 320 PB Tape)

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

What do I mean by Performance Modeling?

- 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 is *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

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

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

Preview: The Process

- Model algorithm and data structures
  - E.g., loads, stores, messages, operations
- Calibrate model
  - E.g., stream, pingpong
- Optional Step: Adjust algorithm, data structure choices
- Measure application; compare to model ***envelope***; explore discrepancies
  - Model missed something
  - Implementation missed something
- Decide if code fast enough (for expected environment and inputs)

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]

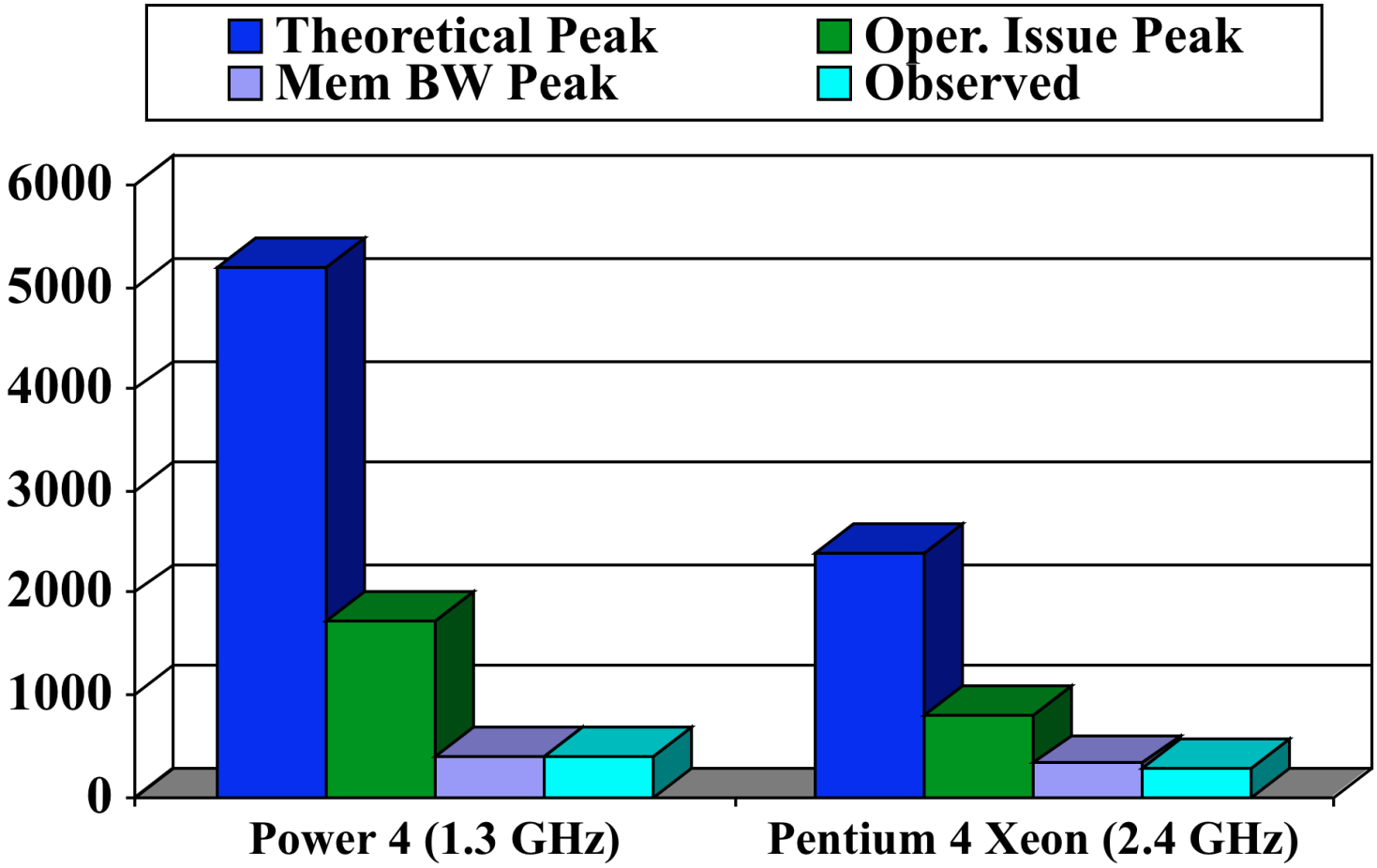
Simple Performance Analysis

- 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

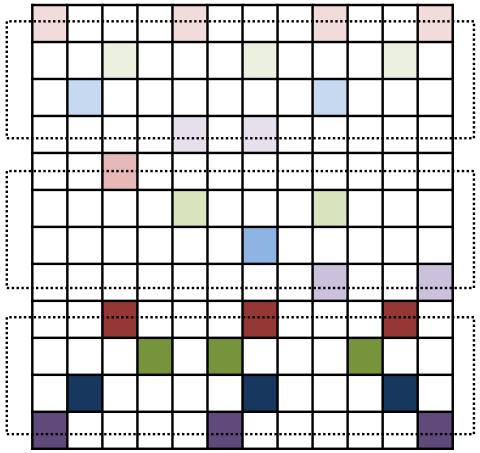
Processes and Memory

- 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 idle memory bus time.
  - However, the hardware must be presented with enough independent data streams

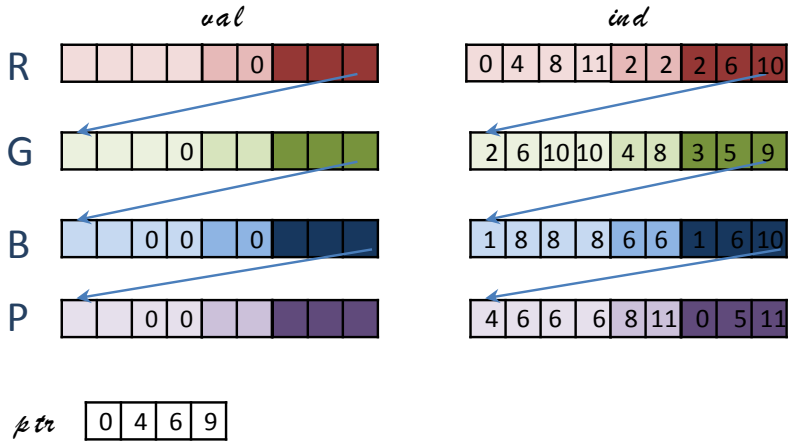
Streamed Compressed Sparse Row (S-CSR) format

- 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.

A sparse matrix (N = 12, NNZ= 29)

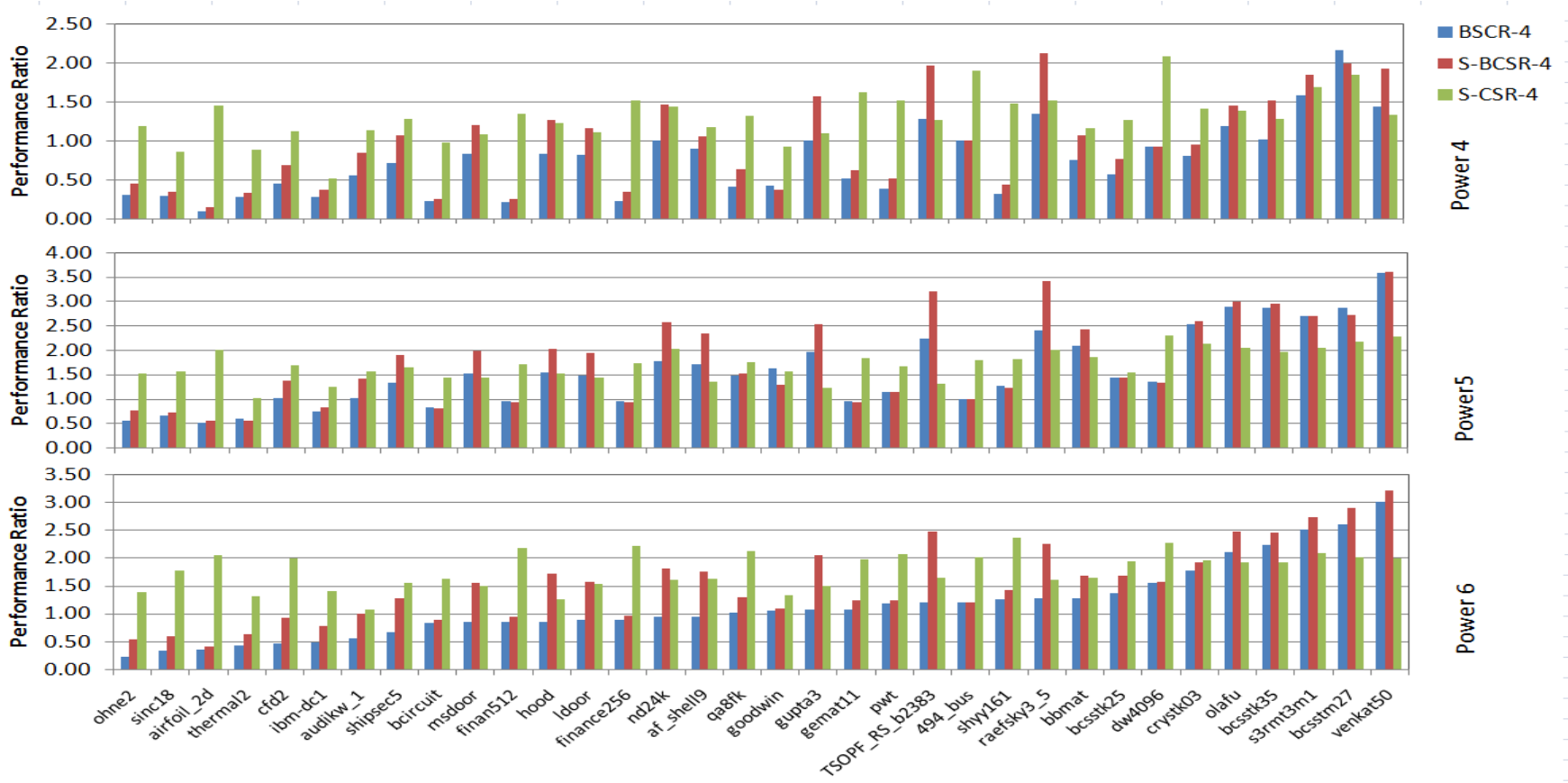


Streamed Compressed Sparse Row format (S-CSR)



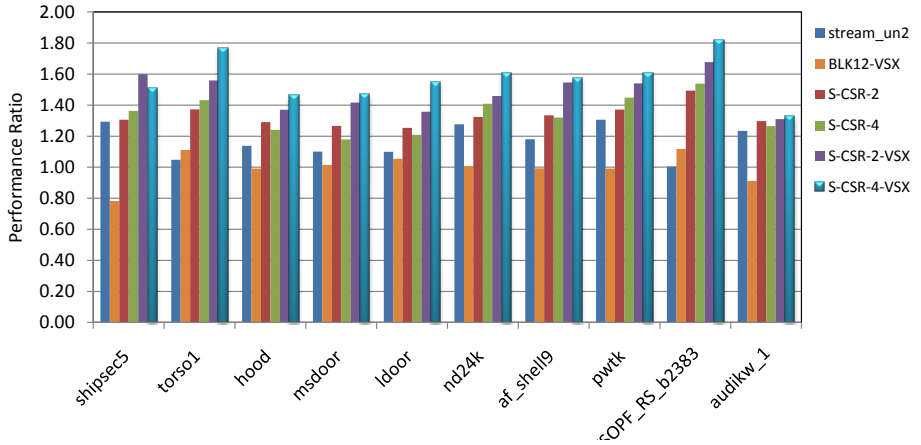
Performance Ratio Compared to CSR Format

- 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.

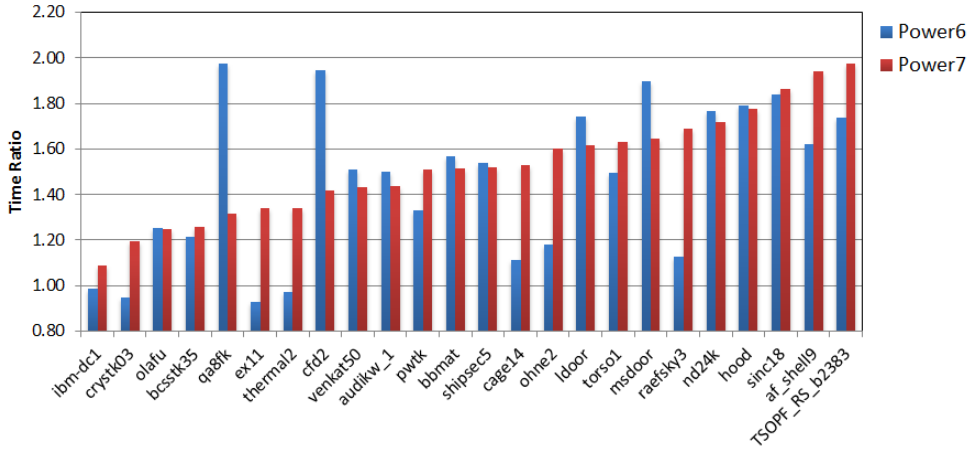


Combining With Other Optimizations

- 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"

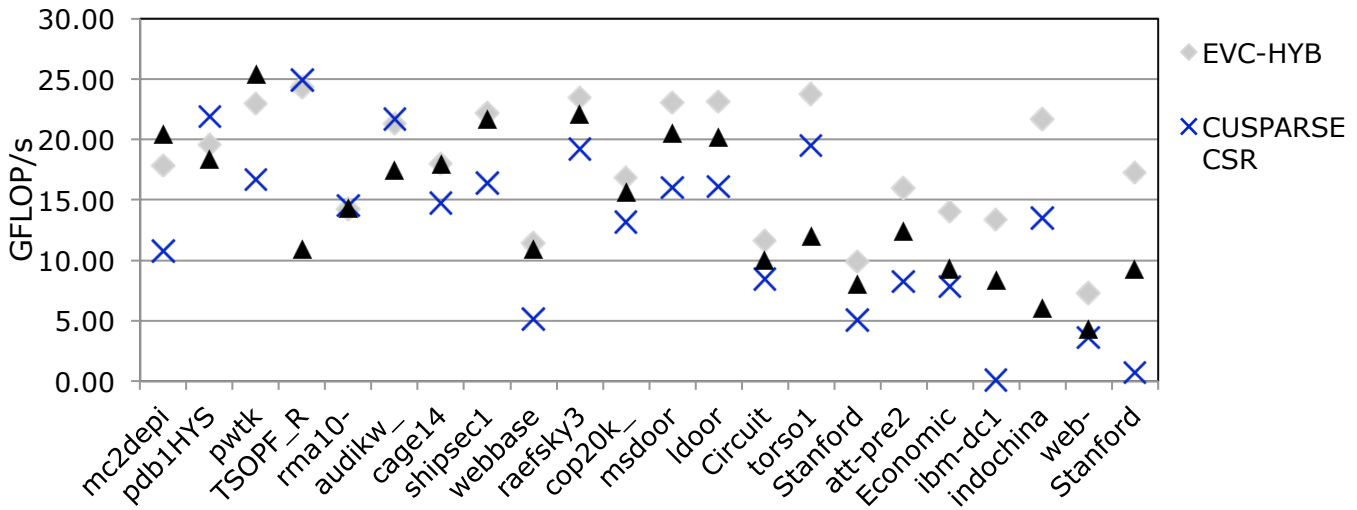


Time comparison between updated OSKI and original OSKI



Hybrid Sparse Matrix Representations

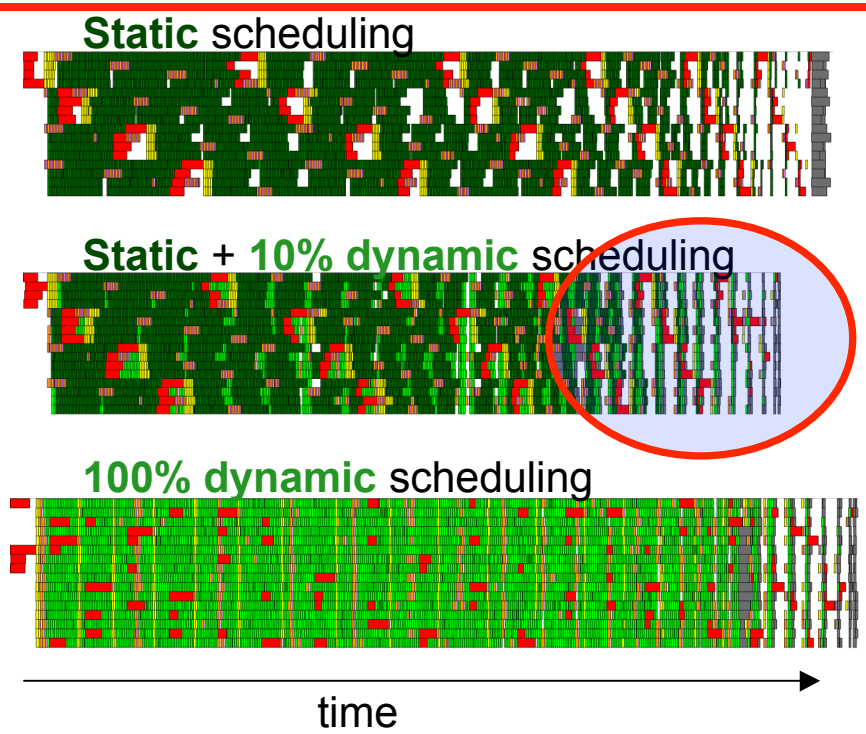
- Match hardware (for GPU, includes coalesced memory references)
- Adapt to matrix structure
  - ♦ May not be one best format for entire matrix
  - ♦ Not new, but penalties high in current systems



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

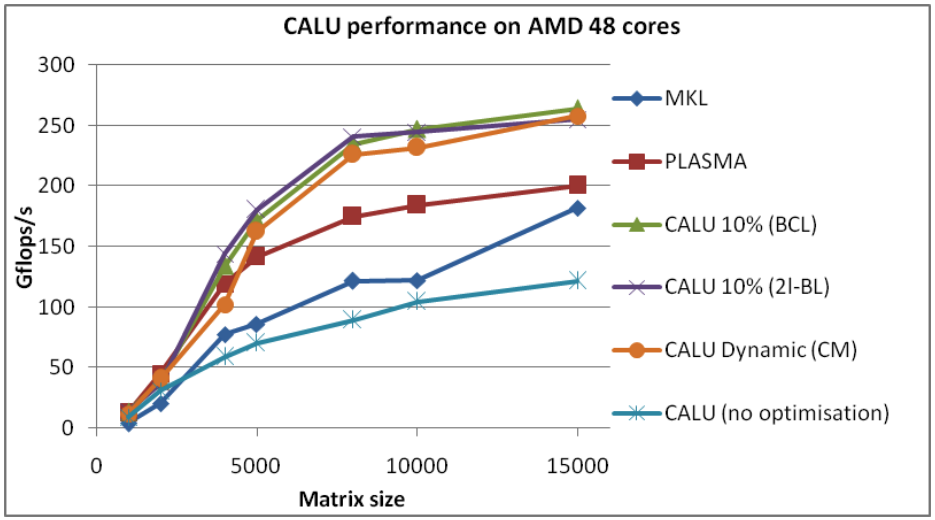


Scary Consequence: Static data decompositions *will not work at scale.*

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

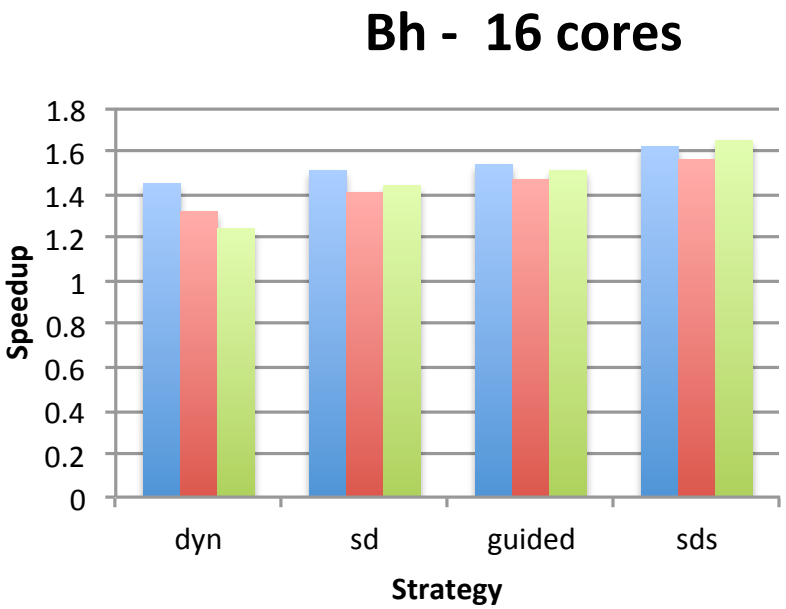
Performance irregularities introduce load-imbalance.
Pure dynamic has significant overhead; pure static too much imbalance.
Solution: combined static and dynamic scheduling

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

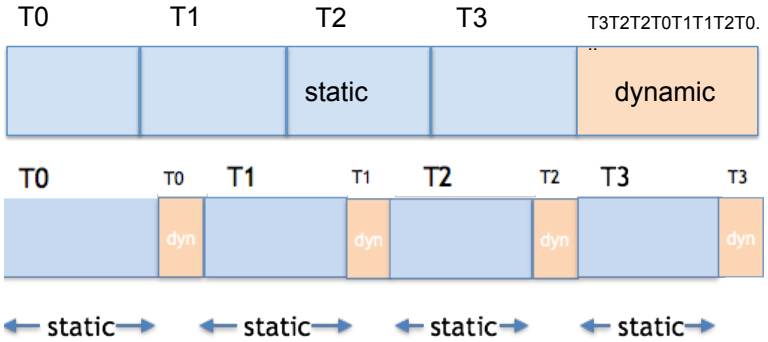


Integrating Locality With Thread Scheduling

- Model performance, costs (especially data motion)
- Can also use information from MPI waits to tune dynamic fraction
- Goal is *not* optimal – goal is better/near optimal



1000, 10k, 100k bodies



Work of Vivek Kale

Changing Requirements for Data Decomposition

- Paraphrasing either Lincoln or PT Barnum:

  You own some of the cores all of the time and all of the cores some of the time, but you don't own all of the cores all of the time

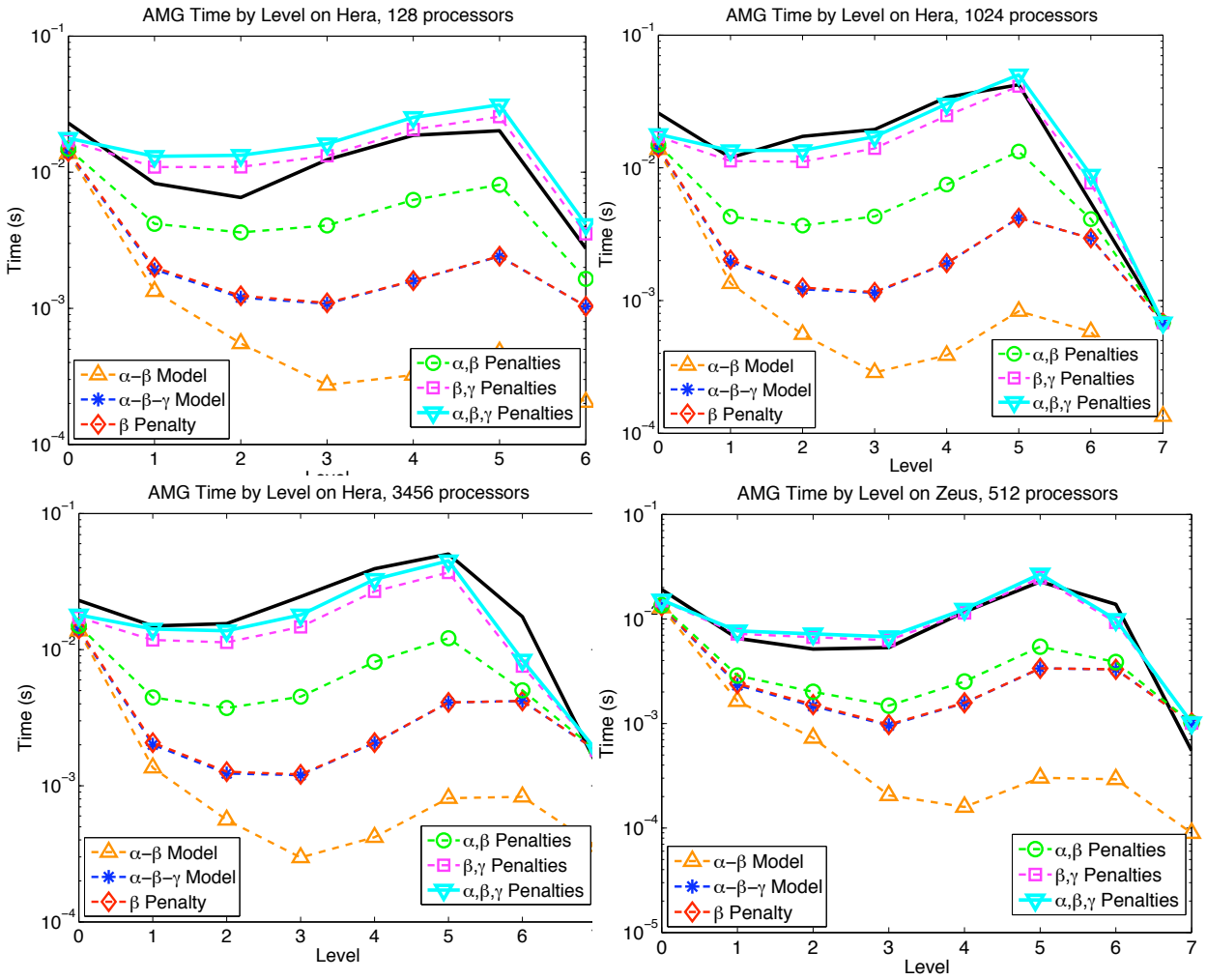
- Translation: a priori data decompositions that were effective on single core processors are no longer effective on multicore processors
- We see this in recommendations to "leave one core to the OS"
  - What about other users of cores, like … the runtime system?

Data Motion Can Dominate Cost

- This is not new
  - Even though floating point operations are still the way computations are usually compared
- Minimizing time may require more computations
- Many examples
  - Lesson here is that simple cost models are often sufficient

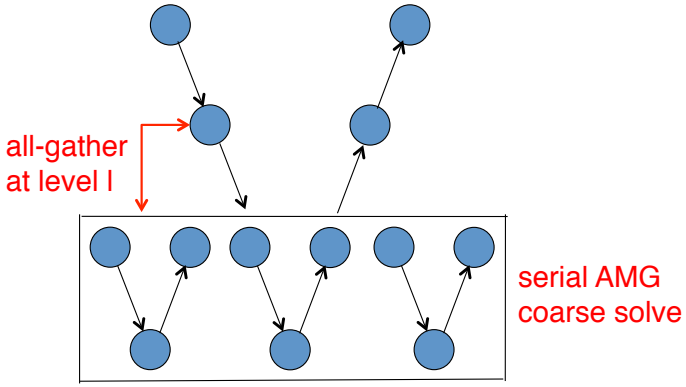
AMG Performance Model

- 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 (ICS'11)



Using Redundant Solvers

- AMG requires a solve on the coarse grid



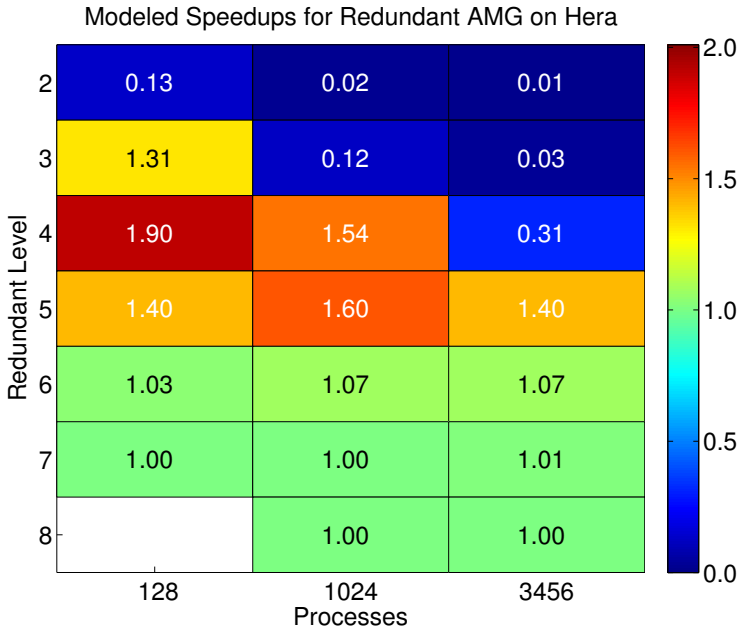
- Options:
  - ♦ Solve in parallel (too little work)
  - ♦ Solve in serial and distribute (Amdahl bottleneck + communication)
  - ♦ Solve redundantly

Redundant Solution

- Replace communication at levels ≥L with Allgather
- Every process now has complete information; no further communication needed
  - ♦ Solution is computed redundantly
- Performance analysis (based on Gropp & Keyes 1989) can guide selection of L
  - ♦ Must be modified by characteristics of modern CPUs and networks

Redundant Solves

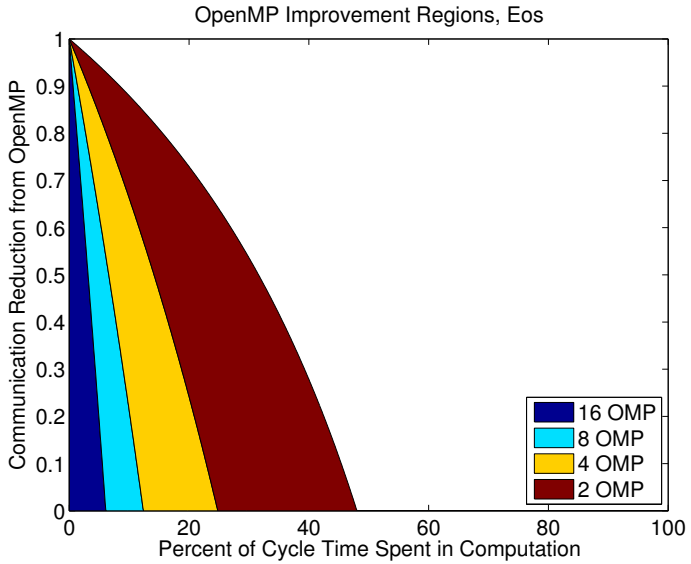
- Applied to Hera at LLNL, provides significant speedup

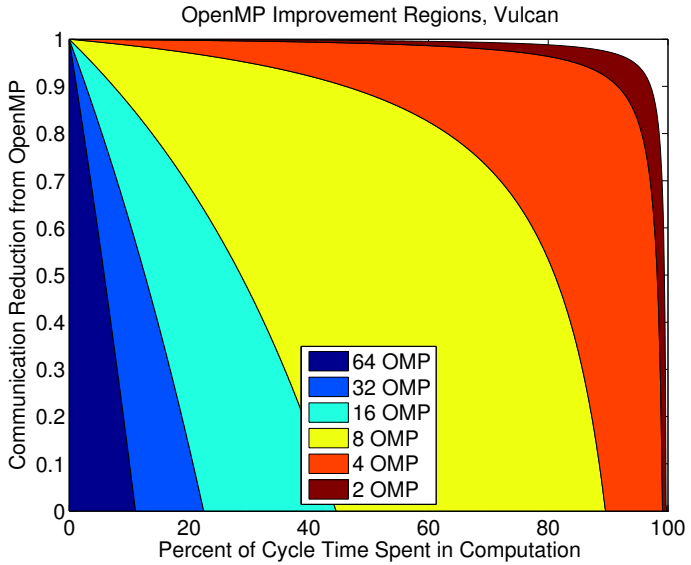


- Lesson: More work can be *faster*
- Key idea is to compute performance *envelope*
- Thanks to Hormozd Gahvari

Gauging the Benefit of Hybrid Programming

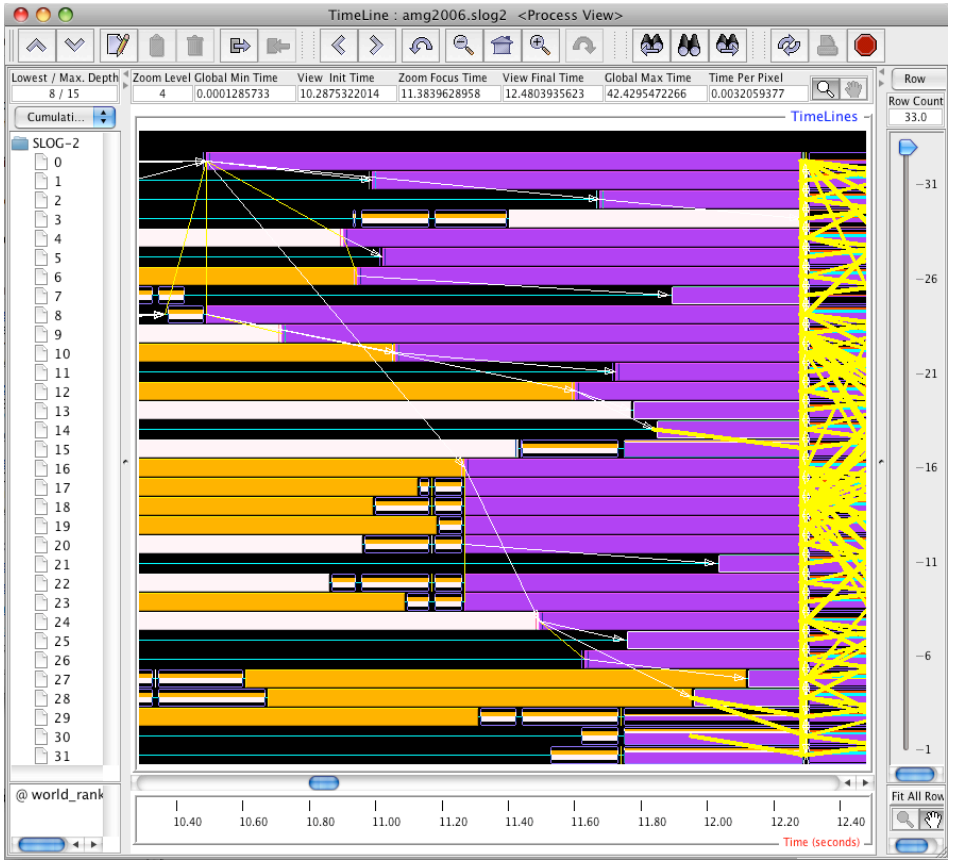
- Simple model:
  - Using OpenMP avoids some MPI communication
  - Using OpenMP introduces some additional costs
  - Compute the trade-off point
- These *guide* choices
  - Real performance depends on additional factors
- Reveals short comings in some *implementations*
- These models roughly matched measurements on real systems (for the values tried)
- Work of Hormozd Gahvari





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



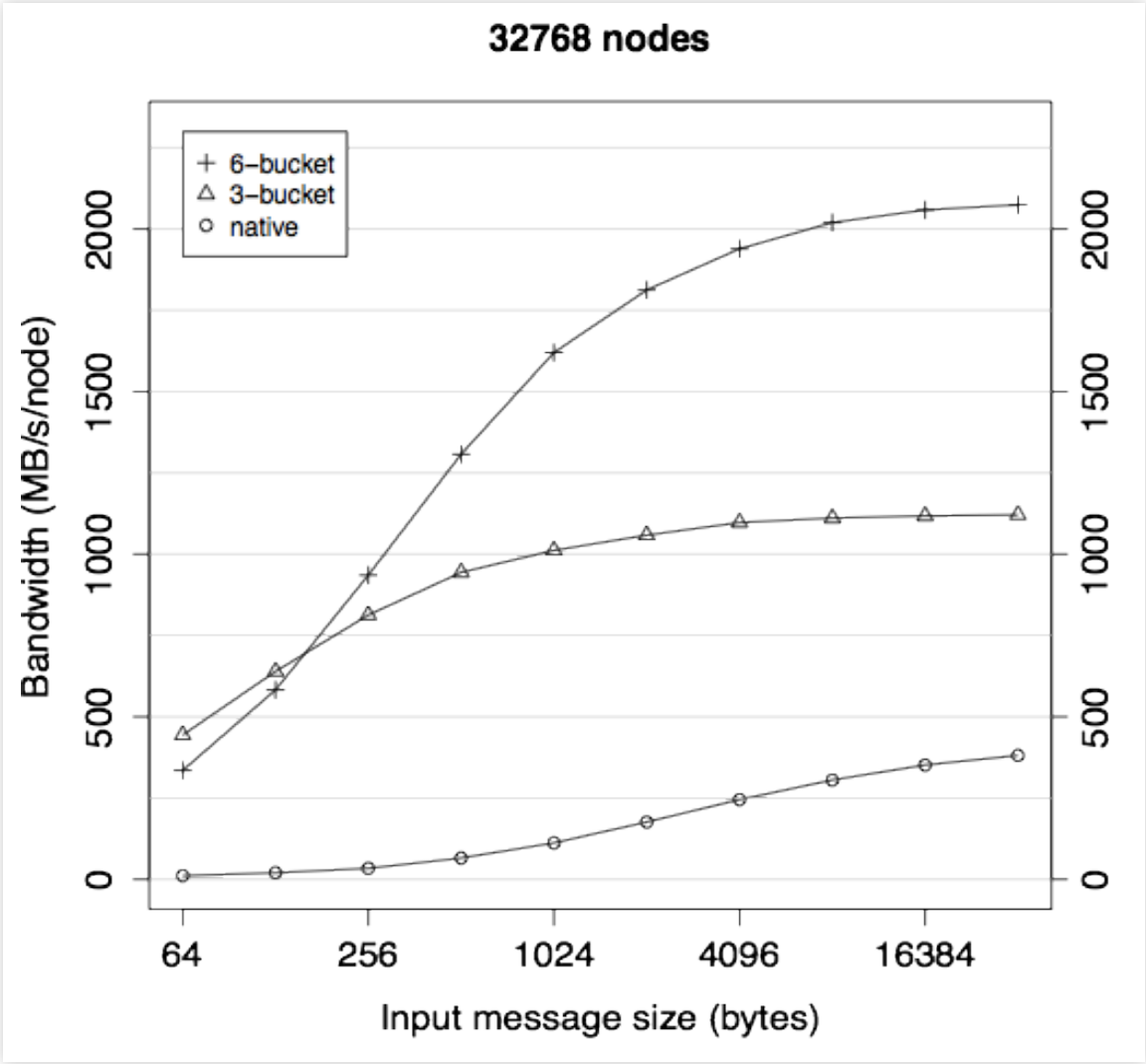
Is It Communication Avoiding Or Minimum Solution Time?

- Example: non minimum collective algorithms
- Work of Paul Sack; see "Faster topology-aware collective algorithms through non-minimal communication", Best Paper, PoPP 2012
  - ♦ Key ideas
    - Shortest messages should travel the longer distances
    - Shuffle of data in network can improve overall times
- Lesson: minimum communication *need not be optimal*

Evaluation: Intrepid BlueGene/P at ANL

- 40k-node system
  - Each is 4 x 850 MHz PowerPC 450
- 512+ nodes is 3d torus; fewer is 3d mesh
- xlc -O4
- 375 MB/s delivered per link
  - 7% penalty using all 6 links both ways

Allgather Performance



Notes on Allgather

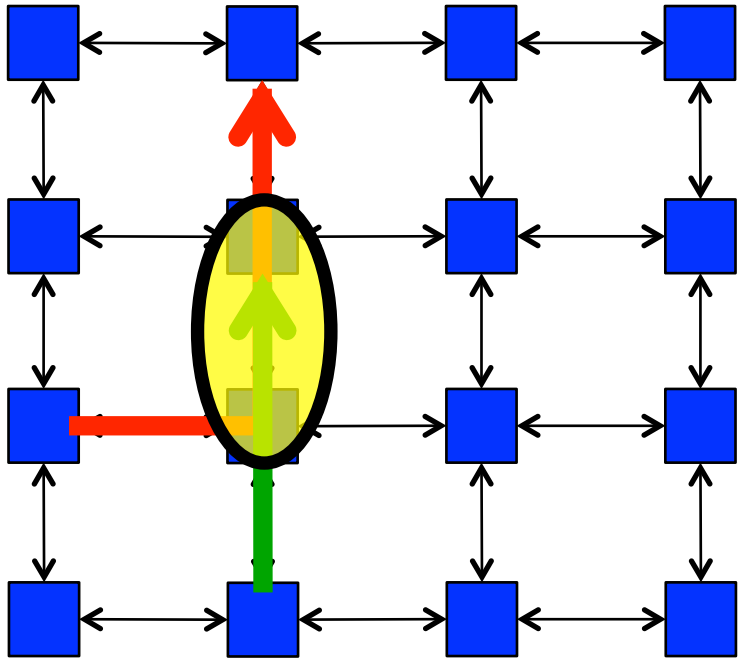
- Bucket algorithm (not described here) exploits multiple communication engines on BG
- *Analysis shows performance near optimal*
- Alternative to reorder data step is in-memory move; analysis shows similar performance and measurements show reorder step faster on tested systems

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…

Communication Cost Includes More than Latency and Bandwidth

- Communication does not happen in isolation
- Effective bandwidth on shared link is ½ point-to-point bandwidth
- Real patterns can involve many more (integer factors)
- Loosely synchronous algorithms ensure communication cost is worst case



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

Halo Exchange on BG/Q and Cray XE6

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

BG/Q	**8 Neighbors**	
	Irecv/Send	Irecv/Isend
World	662	1167
Even/Odd	711	1452
1 sender		2873

Cray XE6	**8 Neighbors**	
	Irecv/Send	Irecv/Isend
World	352	348
Even/Odd	338	324
1 sender		5507

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

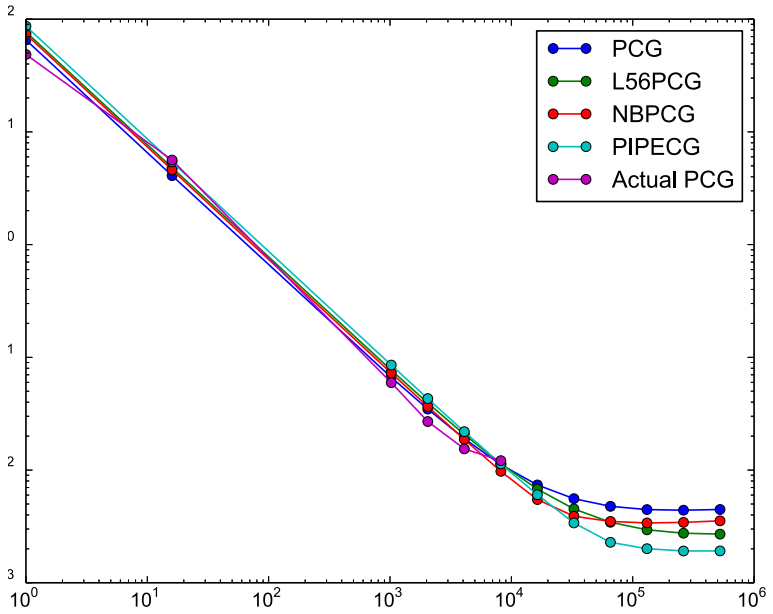
- Ratios of a single sender to all processes sending (in rate)
- *Expect* a factor of roughly 2 (since processes must also receive)

System	4 neighbors		8 Neighbors	
		Periodic		Periodic
BG/L	2.24		2.01	
BG/P	3.8		2.2	
BG/Q			1.98	
XT3	7.5	8.1	9.08	9.41
XT4	10.7	10.7	13.0	13.7
XE6			15.6	15.9

- BG gives roughly double the halo rate. XTn and XE6 are much higher.
  - It should be possible to improve the halo exchange on the XT/E/K by scheduling the communication
  - Or improving the MPI implementation

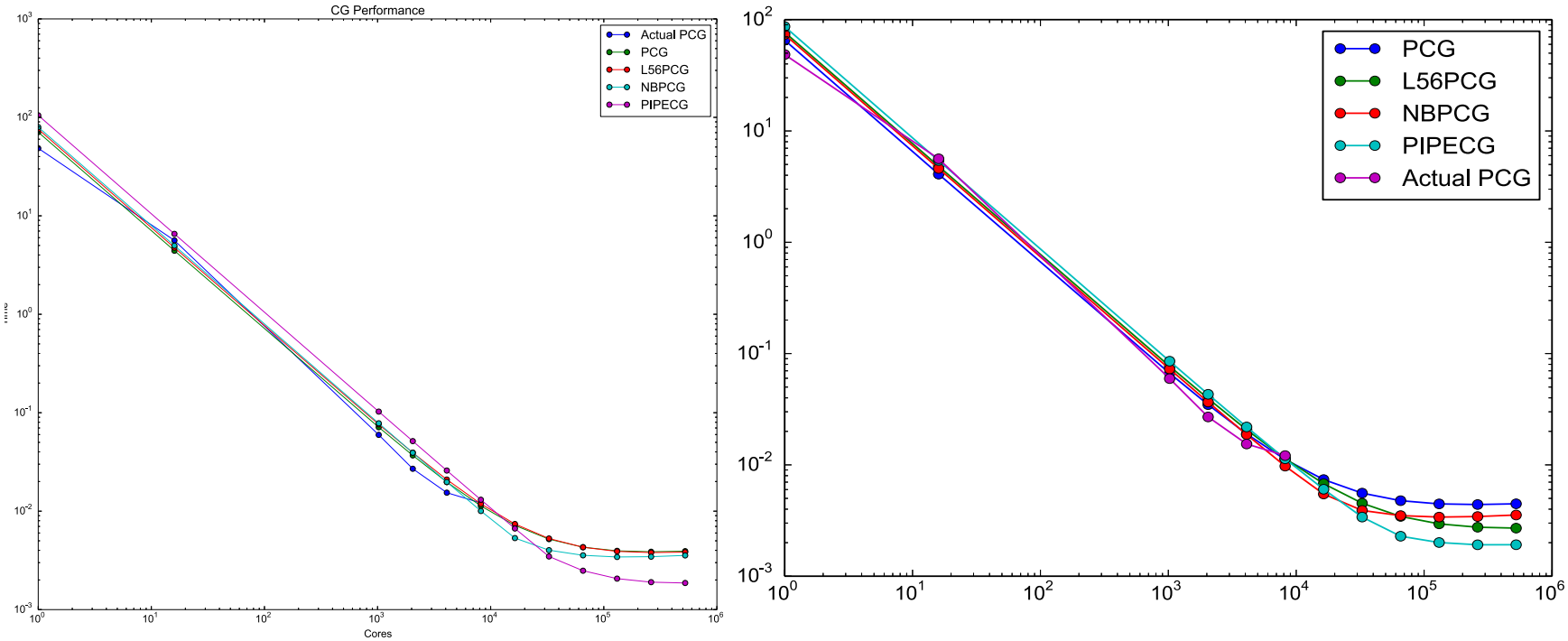
Scaling PCG

- PCG often considered not scalable due to the Allreduce
- Various reformulations trade work for communication
- Model shows benefit of combined vector operations and overlapped Allreduce
- Work of Paul Eller



Method	Dot Prod	AXPYs	Vec Load	Vec Store	Col Flops
PCG	2	3	14	5	6
L56PCG	1	4	14	6	7
PIPECG	1	8	14	10	11
NBPCG	2	5	13	7	8

The Importance of Memory Motion Optimization



Left: methods as described
Right: methods with merged vector operations
Lessons:

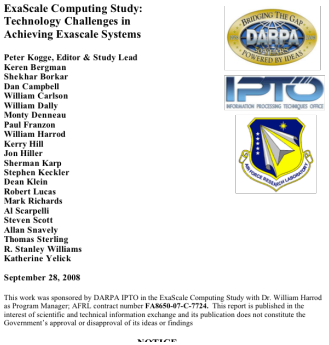
- Don't count FLOPS!
- Don't count DAXPY/BLAS – minimum memory references
- No solution, but real improvements

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**

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

Thanks

- Torsten Hoefler
  - Performance modeling advocate; MPI datatype
- Dahai Guo
  - Streamed format exploiting prefetch
- Paul Eller
  - Nonblocking PCG
- Marc Snir and William Kramer
  - Performance model advocates
- Vivek Kale
  - SMP work partitioning
- Abhinav Bhatele
  - Process/node mapping
- Hormozd Gahvari
  - AMG application modeling
- Funding provided by:
  - Blue Waters project (State of Illinois and the University of Illinois)
  - Department of Energy, Office of Science
  - National Science Foundation