Performance Modeling as the Key to Extreme Scale Computing

William Gropp

www.cs.illinois.edu/∼wgropp
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 too 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?**
An Extreme System

**Power7 Chip**

*Nearly 256 GF peak performance*

- Over 3.5 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 (4 P7/QCM) connection
  - 896 GB/s to other QCMs
  - 40 GB/s general purpose I/O
Two-level (L, D) Direct-connect Network

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

Fully Interconnected with $L_{\text{local}}$ and $L_{\text{remote}}$ Links

But complex, nonuniform network; full system (too?) costly

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

Fully Interconnected with D Links
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
- 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

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Nodes</th>
<th>Topology</th>
<th>Compute</th>
<th>Mem BW</th>
<th>Injection BW</th>
<th>Bisection BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>1</td>
<td>N/A</td>
<td>8 TF/s</td>
<td>3 TB/s</td>
<td>1 TB/s</td>
<td>N/A</td>
</tr>
<tr>
<td>Deployable Cage</td>
<td>22</td>
<td>All-to-All</td>
<td>176 TF/s</td>
<td>67.5 TB/s</td>
<td>22.5 TB/s</td>
<td>31 TB/s</td>
</tr>
<tr>
<td>Rack</td>
<td>128</td>
<td>Flat. Butterfly</td>
<td>1 PF/s</td>
<td>.4 PB/s</td>
<td>0.13 PB/s</td>
<td>0.066 PB/s</td>
</tr>
<tr>
<td>Group Cluster</td>
<td>512</td>
<td>Flat. Butterfly</td>
<td>4.1 PF/s</td>
<td>1.6 PB/s</td>
<td>0.52 PB/s</td>
<td>0.26 PB/s</td>
</tr>
<tr>
<td>National Resource</td>
<td>128k</td>
<td>Hier. All-to-All</td>
<td>1 EF/s</td>
<td>0.4 EB/s</td>
<td>0.13 EB/s</td>
<td>16.8 PB/s</td>
</tr>
<tr>
<td>Max Configuration</td>
<td>2048k</td>
<td>Hier. All-to-All</td>
<td>16 EF/s</td>
<td>6.4 EB/s</td>
<td>2.1 EB/s</td>
<td>0.26 EB/s</td>
</tr>
</tbody>
</table>
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 resource limiting performance
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
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., $T = a + b \times N^3$
  - 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):
  
  ```c
  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:
  ♦ $\text{nnz} \ (\text{sizeof(double)} + \text{sizeof(int)}) + n \ (2\times\text{sizeof(double)} + \text{sizeof(int)})$
  ♦ Assume a perfect cache (never load same data twice)

• Computation
  ♦ $\text{nnz}$ 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

Realistic Measures of Peak Performance

Sparse Matrix Vector Product
One vector, matrix size, $m = 90,708$, nonzero entries $nz = 5,047,120$

Thanks to Dinesh Kaushik; ORNL and ANL for compute time

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

### Media Bench II Applications

<table>
<thead>
<tr>
<th>Appl</th>
<th>XLC</th>
<th>ICC</th>
<th>GCC</th>
<th>XLC</th>
<th>ICC</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Automatic</td>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPEG Enc</td>
<td>-</td>
<td>1.33</td>
<td>-</td>
<td>1.39</td>
<td>2.13</td>
<td>1.57</td>
</tr>
<tr>
<td>JPEG Dec</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.14</td>
<td>1.13</td>
</tr>
<tr>
<td>H263 Enc</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.25</td>
<td>2.28</td>
<td>2.06</td>
</tr>
<tr>
<td>H263 Dec</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.31</td>
<td>1.45</td>
<td>-</td>
</tr>
<tr>
<td>MPEG2 Enc</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.06</td>
<td>1.96</td>
<td>2.43</td>
</tr>
<tr>
<td>MPEG2 Dec</td>
<td>-</td>
<td>-</td>
<td>1.15</td>
<td>1.37</td>
<td>1.45</td>
<td>1.55</td>
</tr>
<tr>
<td>MPEG4 Enc</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.44</td>
<td>1.81</td>
<td>1.74</td>
</tr>
<tr>
<td>MPEG4 Dec</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.12</td>
<td>-</td>
<td>1.18</td>
</tr>
</tbody>
</table>

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

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

• Guo and Gropp, IJHPCA 2011
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 block. The 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”
- Guo and Gropp, submitted
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 (EuroMPI’10)
Processes and the Network

- How relevant is ping-pong bandwidth and real systems?
- 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
  - Up to 18% faster!
- Next bottleneck: CG phase
  - Investigating use of nonblocking collectives in a modified CG
  - Also model-driven (because involves more floating point but same or less data motion)
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)
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
- Same issue with thread programs – “cost” of barriers, locks, ...
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
- 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)

<table>
<thead>
<tr>
<th>BG/P</th>
<th>4 Neighbors</th>
<th>8 Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Irecv/Send</td>
<td>Irecv/Isend</td>
</tr>
<tr>
<td>World</td>
<td>208</td>
<td>328</td>
</tr>
<tr>
<td>Even/Odd</td>
<td>219</td>
<td>327</td>
</tr>
<tr>
<td>Cart_create</td>
<td>301</td>
<td>581</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cray XT4</th>
<th>4 Neighbors</th>
<th>8 Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Irecv/Send</td>
<td>Irecv/Isend</td>
</tr>
<tr>
<td>World</td>
<td>311</td>
<td>306</td>
</tr>
<tr>
<td>Even/Odd</td>
<td>257</td>
<td>247</td>
</tr>
<tr>
<td>Cart_create</td>
<td>265</td>
<td>275</td>
</tr>
</tbody>
</table>
Discovering Performance Opportunities

- Let's 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)

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

<table>
<thead>
<tr>
<th>System</th>
<th>4 neighbors</th>
<th>8 Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Periodic</td>
<td>Periodic</td>
</tr>
<tr>
<td>BG/L</td>
<td>2.24</td>
<td>2.01</td>
</tr>
<tr>
<td>BG/L, VN</td>
<td>1.46</td>
<td>1.81</td>
</tr>
<tr>
<td>BG/P</td>
<td>3.8</td>
<td>2.2</td>
</tr>
<tr>
<td>BG/P, VN</td>
<td>2.6</td>
<td>5.5</td>
</tr>
<tr>
<td>XT3</td>
<td>7.5</td>
<td>8.1</td>
</tr>
<tr>
<td>XT4</td>
<td>10.7</td>
<td>10.7</td>
</tr>
<tr>
<td>XT4 SN</td>
<td>5.47</td>
<td>5.56</td>
</tr>
</tbody>
</table>

- 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 involve
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 (see Vuduc et al for good examples)

• 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!
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
- Paul Sack
  ♦ Contention-reducing collectives
- 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