Assessing Sustained System Performance - A Practical Solution for Heterogeneous Systems

Torsten Hoefler, Bill Kramer, Greg Bauer, & AUS@BW

All used images belong to the owner/creator!
The State of Performance Measurements

- Most used metric: Floating Point Performance
  - That’s what limited performance in the 80’s!
  - Systems were balanced, peak was easy!
  - FP performance was the limiting factor
- Architecture Update (2012):
  - Deep memory hierarchies
    - Hard to predict and model
  - Algorithmic structure and data locality matters
    - Complicates things further
Rough Computational Algorithm Classification

- High locality, moderate locality, low locality
- Highly Structured
  - Dense linear algebra (HPL)
  - FFT
  - Stencil
- Semi-structured
  - Adaptive refinements
  - Sparse linear algebra
- Unstructured
  - Graph computations (Graph500)
How do we assess performance?

• Microbenchmarks
  • Libraries (DGEMM, FFT)
  • Communication (p2p, collective)
  • …

• Application Microbenchmark
  • HPL (for historic reasons?)
  • NAS (outdated)
  • …

• Applications
We still somehow agree on FLOPS

• … because that’s what we always did
  • And it’s an OK metric
• But the benchmarks should reflect the workload
  • “Sustained performance”
  • Cf. “real application performance”
• In the Blue Waters context
  • “Sustained Petascale Performance” (SPP)
  • Reflects the NSF workload
The SPP Metric

- Enables us to
  - compare different computer systems
  - Verify system performance and correctness
  - Monitor performance through lifetime
  - Guide design of future systems
- It has to represent the “average workload” and must still be of manageable size
  - We chose ten applications (8 x86, 4 GPU)
  - Performance is geometric mean of all apps
Validating a System Model – Memory I

- Stride-1 word load/store/copy (32 MiB data):
  - 1 int core r/w/c: 3.8 / 4 / 3 GB/s
  - 16 int cores (1 IL) r/w/c: 32 / 16 / 9.6 GB/s
  - 32 int cores (2 IL) r/w/c: 32 / 16 / 9.6 GB/s

- Comments:
  - Very **high fairness** between cores
  - Very **low variance** between measurements

Measured with Netgauge 2.4.7, pattern memory/stream
Validating a System Model – Memory II

- CL latency (random pointer chase, 1 GiB data):
  - 1 int core: 110 ns
  - 16 int cores (1 IL): 257 ns
  - 32 int cores (2IL): 258 ns

- Comments:
  - High fairness between cores
  - Low variance between measurements

Measured with Netgauge 2.4.7, pattern memory/pchase
Validating a System Model – Memory III

- Random word access bandwidth (32 MiB data):
  - 1 int core r/w/c: 453 / 422 / 228 MiB/s
  - 16 int cores (1 IL) r/w/c: 241 / 119 / 77 MiB/s
  - 32 int cores (2IL) r/w/c: 241 / 119 / 77 MiB/s

- Comments:
  - 96% of stream bandwidth
  - Very high fairness between cores
  - Very low variance between measurements

Measured with Netgauge 2.4.7, pattern memory/rand
Validating a System Model – Network Scaling

- Average random latency and variance

Measured with Netgauge 2.4.7, pattern ebb
Validating a System Model – Collectives

- Large message (4k) alltoall performance
- Model: unclear (depends on mapping etc.)

Measured with Netgauge 2.4.7, pattern nbcolls
The SPP Application Mix

- Representative Blue Waters applications:
  - NAMD – molecular dynamics
  - MILC, Chroma – Lattice Quantum Chromodynamics
  - VPIC, SPECFEM3D – Geophysical Science
  - WRF – Atmospheric Science
  - PPM – Astrophysics
  - NWChem, GAMESS – Computational Chemistry
  - QMCPACK – Materials Science
The Grand Modeling Vision

- Our **very** high-level strategy consists of the following six steps:

  1) Identify input parameters that influence runtime
  2) Identify application kernels
  3) Determine communication pattern
  4) Determine communication/computation overlap
  5) Determine sequential baseline
  6) Determine communication parameters

---

Hoefler, Gropp, Snir, Kramer: Performance Modeling for Systematic Performance Tuning, SC11
A Simplified Modeling Method

- Fix input problem (omit step 1)
- No fancy tools, simple library using PAPI (libPGT)
- Determine performance-critical kernels
  - We demonstrate a simple method to identify kernels
- Analyze kernel performance
  - Using black-box counter approach
  - More accurate methods if time permits
- Establish system bounds, roofline
  - What can be improved? Are we hitting a bottleneck?
NAMD

<table>
<thead>
<tr>
<th>phase</th>
<th>MIPS</th>
<th>MFLOPS/s</th>
<th>MiBPS</th>
<th>CI</th>
<th>AI</th>
<th>IPC</th>
<th>effGHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonbonded</td>
<td>2460</td>
<td>1377</td>
<td>7506</td>
<td>1.1</td>
<td>0.2</td>
<td>1.1</td>
<td>2.3</td>
</tr>
<tr>
<td>PME</td>
<td>1772</td>
<td>1408</td>
<td>3299</td>
<td>1.7</td>
<td>0.4</td>
<td>0.8</td>
<td>2.3</td>
</tr>
<tr>
<td>bonded</td>
<td>1617</td>
<td>723</td>
<td>1821</td>
<td>0.8</td>
<td>0.4</td>
<td>0.7</td>
<td>2.3</td>
</tr>
<tr>
<td>integrate</td>
<td>1394</td>
<td>581</td>
<td>4573</td>
<td>0.8</td>
<td>0.1</td>
<td>0.6</td>
<td>2.3</td>
</tr>
</tbody>
</table>

- Dynamic scheduling complicates model
- Excellent cache locality
- PME performs well but will slow down at scale (alltoall)
- Good IPC
Five phases, CG most critical at scale
Low FLOPs and IPC
  Turbo boost seems to help here!
Low FLOPs are under investigation (already using SSE)
Many micro-phases
Hard to instrument
Very highly optimized by science team
  - Cache blocking
  - High FLOP rate
  - High locality
QMCPACK

<table>
<thead>
<tr>
<th>phase</th>
<th>MIPS</th>
<th>MFLOPS/s</th>
<th>MiBPS</th>
<th>CI</th>
<th>AI</th>
<th>IPC</th>
<th>effGHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>2083</td>
<td>943</td>
<td>1933</td>
<td>1.1</td>
<td>0.5</td>
<td>0.9</td>
<td>2.3</td>
</tr>
<tr>
<td>uw</td>
<td>1902</td>
<td>1177</td>
<td>2433</td>
<td>1.5</td>
<td>0.5</td>
<td>0.8</td>
<td>2.3</td>
</tr>
<tr>
<td>LB</td>
<td>3155</td>
<td>0</td>
<td>18</td>
<td>0.0</td>
<td>0.0</td>
<td>1.4</td>
<td>2.3</td>
</tr>
</tbody>
</table>

- Variational Monte Carlo initializes
- Performance issues are investigated
- Diffusion Monte Carlo:
  - load balance (LB)
  - update walker (uw)
- Microphysics dominates
  - Low performance, many branches
- Planet Boundary Layer also problematic
  - Turbo Boost helps!
- Runge Kutta is fast
  - High locality
SPECFEM3D

<table>
<thead>
<tr>
<th>phase</th>
<th>MIPS</th>
<th>MFLOPS/s</th>
<th>MiBPS</th>
<th>CI</th>
<th>AI</th>
<th>IPC</th>
<th>effGHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>tiso</td>
<td>1973</td>
<td>2010</td>
<td>1197</td>
<td>1.9</td>
<td>1.8</td>
<td>0.8</td>
<td>2.3</td>
</tr>
<tr>
<td>forces</td>
<td>1602</td>
<td>1736</td>
<td>4577</td>
<td>1.5</td>
<td>0.4</td>
<td>0.7</td>
<td>2.3</td>
</tr>
<tr>
<td>iso</td>
<td>1474</td>
<td>1396</td>
<td>1617</td>
<td>1.6</td>
<td>0.9</td>
<td>0.6</td>
<td>2.3</td>
</tr>
</tbody>
</table>

- Two phases, both do small mat-mat mult
- Internal forces perform well
NWCHEM

<table>
<thead>
<tr>
<th>phase</th>
<th>MIPS</th>
<th>MFLOPS/s</th>
<th>MiBPS</th>
<th>CI</th>
<th>AI</th>
<th>IPC</th>
<th>effGHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2616</td>
<td>431</td>
<td>5464</td>
<td>0.3</td>
<td>0.1</td>
<td>1.0</td>
<td>2.6</td>
</tr>
<tr>
<td>2</td>
<td>2660</td>
<td>398</td>
<td>4818</td>
<td>0.3</td>
<td>0.1</td>
<td>1.0</td>
<td>2.6</td>
</tr>
<tr>
<td>3+4</td>
<td>2463</td>
<td>1246</td>
<td>6030</td>
<td>0.9</td>
<td>0.2</td>
<td>1.0</td>
<td>2.6</td>
</tr>
<tr>
<td>5</td>
<td>4156</td>
<td>6876</td>
<td>15583</td>
<td>3.5</td>
<td>0.4</td>
<td>1.6</td>
<td>2.6</td>
</tr>
</tbody>
</table>

- Highly optimized
- Even running in turbo boost!
- Very good locality
- Steps 3+4 decent
- Step 5 close to peak!
Some Early Conclusions

• Average Effective Frequency: 2.40 GHz
  • Anticipated frequency: 2.45 GHz
• Average FLOP rate: 1.48 GF (min: 398 GF (WRF), max: 6.876 GF (NWCHEM))
  • 15% of peak ☺
  • Standard deviation: 1.37 GF (!!!)
• But what does that mean?
  • Are we hitting any limits/bottlenecks?
Serial Performance - The Roofline Model

The poster child - NWChem

Floating Point Performance (GF)

Operational Intensity (F/B)

CCSD

DGEMM

TRI

Thanks to Victor Anisimov
MIMD Lattice Computation - MILC

Floating Point Performance (GF)

Operational Intensity (F/B)

Thanks to Greg Bauer
MIMD Lattice Computation - MILC

Cache-aware programming

Operational Intensity (F/B)

Floating Point Performance (GF)

10x10x10x10

6x6x6x6

GF

FF

LL

CG

FL

Thanks to Greg Bauer
MIMD Lattice Computation - MILC

Floating Point Performance (GF)

Operational Intensity (F/B)

Thanks to Greg Bauer
Lessons learned and Discussion

- Performance modeling is a powerful tool
  - To detect bottlenecks and bounds
- The Roofline Model
  - PAPI/CrayPAT may fool you (uncore events)
  - Bandwidth read or write?
  - Shows most important characteristics of serial codes
    - Room for improvement and interpretation
- Roofline may depend on critical parameters
- Could use a tool to handle all of this!

Hoefler: “Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC”
Conclusions & Future Work

- We modeled the performance of several SPP applications
  - Gained insight on limits/bounds
- Kernel classification through IPC works well
  - Not automatic yet
- Kernel profiling works in early stages
  - Need better tools
- Extending modeling towards communication
  - “MPI counters”, congestion, etc.

Hoefler, Gropp, Snir, Kramer: Performance Modeling for Systematic Performance Tuning, SC11
Acknowledgments

• Thanks to
  • Gregory Bauer (pulling together the data)
  • Victor Anisimov, Eric Bohm, Robert Brunner, Ryan Mokos, Craig Steffen, Mark Straka (SPP PoCs)
  • Bill Kramer, Bill Gropp, Marc Snir (general modeling ideas/discussions)
  • The Cray performance group (Joe Glenski et al.)

• The National Science Foundation
Blue Waters in a Nutshell

• XE6 with AMD Interlagos 2.3-2.6 (3.0?) GHz
  • ~390k BD modules, ~780k INT cores
• XK6 with Kepler GPUs
  • ~3k
• Gemini Torus
  • Very large (23x24x24), BB-challenged, torus
• How do we make sure the (heterogeneous) system is ready to fulfill it’s mission?
  • Well, confirm a certain SPP number (> 1PF!)
Performance Counter Sanity Checks

Table 1: Performance characteristics for simple kernels

<table>
<thead>
<tr>
<th>kernel</th>
<th>MIPS</th>
<th>MFLOPS/s</th>
<th>MiBPS</th>
<th>CI</th>
<th>AI</th>
<th>IPC</th>
<th>effGHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>triad s</td>
<td>300</td>
<td>407</td>
<td><strong>3958</strong></td>
<td>1.1</td>
<td>0.1</td>
<td>0.1</td>
<td>2.3</td>
</tr>
<tr>
<td>triad l</td>
<td>241</td>
<td>156</td>
<td><strong>1574</strong></td>
<td>1.0</td>
<td>0.1</td>
<td>0.1</td>
<td>2.6</td>
</tr>
<tr>
<td>stencil s</td>
<td>1089</td>
<td>2508</td>
<td><strong>9172</strong></td>
<td>1.4</td>
<td>0.3</td>
<td>0.5</td>
<td>2.3</td>
</tr>
<tr>
<td>stencil l</td>
<td>181</td>
<td>458</td>
<td>1684</td>
<td>1.4</td>
<td>0.3</td>
<td>0.1</td>
<td>2.6</td>
</tr>
<tr>
<td>dgemm l</td>
<td>3690</td>
<td><strong>7940</strong></td>
<td>3297</td>
<td>5.0</td>
<td>2.4</td>
<td>1.6</td>
<td>2.3</td>
</tr>
<tr>
<td>reg int</td>
<td>2000</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.8</td>
<td>2.6</td>
</tr>
</tbody>
</table>

- Running small test kernels to check counters
- s=small, l=large
- Stream: 2 GB/s per integer core
- LL_CACHE_MISSES are L2 misses!?
  - Still a proxy metric (use with caution!)
Upping my FLOPS (if I was a vendor)

- Algorithms may have different FLOP counts
  - Slow time to solution but high FLOPS (dense LA)
  - Same time to solution, more FLOPS
  - Single of half FLOPS (esp. GPUs)
  - Redundant FLOPS for parallel codes
- Performance counters are thus not reliable!
  - Just count the observed, not the necessary FLOPS
Reference FLOP Counts

• We establish “reference FLOP count”
  • Specific to an input problem
  • Ideally established analytically
  • Or (if necessary) on reference code on x86
    • Single-core run (or several parallel runs)
• Input problem needs to be clearly defined
  • Set the right expectations
  • Real, complete science run vs. maximum FLOPS