TOWARD A NEW (ANOTHER) METRIC FOR RANKING HIGH PERFORMANCE COMPUTING SYSTEMS

Jack Dongarra & Piotr Luszczek
University of Tennessee/ORNL

Michael Heroux
Sandia National Labs

See: http://tiny.cc/hpcg
Confessions of an Accidental Benchmarker

- Appendix B of the Linpack Users’ Guide
  - Designed to help users extrapolate execution time for Linpack software package
- First benchmark report from 1977;
  - Cray 1 to DEC PDP-10

http://tiny.cc/hpcg
Started 36 Years Ago
Have seen a Factor of $10^9$ - From 14 Mflop/s to 34 Pflop/s

- In the late 70’s the fastest computer ran LINPACK at 14 Mflop/s
- Today with HPL we are at 34 Pflop/s
  - Nine orders of magnitude
doubling every 14 months
  - About 6 orders of magnitude increase in the number of processors
- Plus algorithmic improvements

Began in late 70’s
time when floating point operations were expensive compared to other operations and data movement
High Performance Linpack (HPL)

- Is a widely recognized and discussed metric for ranking high performance computing systems.
- When HPL gained prominence as a performance metric in the early 1990s there was a strong correlation between its predictions of system rankings and the ranking that full-scale applications would realize.
- Computer system vendors pursued designs that would increase their HPL performance, which would in turn improve overall application performance.
- Today HPL remains valuable as a measure of historical trends, and as a stress test, especially for leadership class systems that are pushing the boundaries of current technology.
The Problem

- HPL performance of computer systems are no longer so strongly correlated to real application performance, especially for the broad set of HPC applications governed by partial differential equations.

- Designing a system for good HPL performance can actually lead to design choices that are wrong for the real application mix, or add unnecessary components or complexity to the system.
Concerns

• The gap between HPL predictions and real application performance will increase in the future.
• A computer system with the potential to run HPL at 1 Exaflops is a design that may be very unattractive for real applications.
• Future architectures targeted toward good HPL performance will not be a good match for most applications.
• This leads us to think about a different metric
HPL - Good Things

- Easy to run
- Easy to understand
- Easy to check results
- Stresses certain parts of the system
- Historical database of performance information
- Good community outreach tool
- “Understandable” to the outside world

If your computer doesn’t perform well on the LINPACK Benchmark, you will probably be disappointed with the performance of your application on the computer.
HPL - Bad Things

- LINPACK Benchmark is 36 years old
  - Top500 (HPL) is 20.5 years old
- Floating point-intensive performs $O(n^3)$ floating point operations and moves $O(n^2)$ data.
- No longer so strongly correlated to real apps.
- Reports Peak Flops (although hybrid systems see only 1/2 to 2/3 of Peak)
- Encourages poor choices in architectural features
- Overall usability of a system is not measured
- Used as a marketing tool
- Decisions on acquisition made on one number
- Benchmarking for days wastes a valuable resource
Running HPL

- In the beginning to run HPL on the number 1 system was under an hour.
- On Livermore’s Sequoia IBM BG/Q the HPL run took about a day to run.
  - They ran a size of \( n = 12.7 \times 10^6 \) (1.28 PB)
  - 16.3 PFlop/s requires about 23 hours to run!!
  - 23 hours at 7.8 MW that the equivalent of 100 barrels of oil or about $8600 for that one run.

- The longest run was 60.5 hours
  - JAXA machine
    - Fujitsu FX1, Quadcore SPARC64 VII 2.52 GHz
  - A matrix of size \( n = 3.3 \times 10^6 \)
  - .11 Pflop/s #160 today
Run Times for HPL on Top500 Systems

http://tiny.cc/hpcg
#1 System on the Top500 Over the Past 20 Years
(16 machines in that club)

<table>
<thead>
<tr>
<th>Top500 List</th>
<th>Computer</th>
<th>(r_{\text{max}}) (Tflop/s)</th>
<th>(n_{\text{max}})</th>
<th>Hours</th>
<th>MW</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/93 (1)</td>
<td>TMC CM-5/1024</td>
<td>.060</td>
<td>52224</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>11/93 (1)</td>
<td>Fujitsu Numerical Wind Tunnel</td>
<td>.124</td>
<td>31920</td>
<td>0.1</td>
<td>1.</td>
</tr>
<tr>
<td>6/94 (1)</td>
<td>Intel XP/S140</td>
<td>.143</td>
<td>55700</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>11/94 - 11/95 (3)</td>
<td>Fujitsu Numerical Wind Tunnel</td>
<td>.170</td>
<td>42000</td>
<td>0.1</td>
<td>1.</td>
</tr>
<tr>
<td>6/96 (1)</td>
<td>Hitachi SR2201/1024</td>
<td>.220</td>
<td>138,240</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>11/96 (1)</td>
<td>Hitachi CP-PACS/2048</td>
<td>.368</td>
<td>103,680</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>6/97 - 6/00 (7)</td>
<td>Intel ASCI Red</td>
<td>2.38</td>
<td>362,880</td>
<td>3.7</td>
<td>.85</td>
</tr>
<tr>
<td>11/00 - 11/01 (3)</td>
<td>IBM ASCI White, SP Power3 375 MHz</td>
<td>7.23</td>
<td>518,096</td>
<td>3.6</td>
<td></td>
</tr>
<tr>
<td>6/02 - 6/04 (5)</td>
<td>NEC Earth-Simulator</td>
<td>35.9</td>
<td>1,000,000</td>
<td>5.2</td>
<td>6.4</td>
</tr>
<tr>
<td>11/04 - 11/07 (7)</td>
<td>IBM BlueGene/L</td>
<td>478</td>
<td>1,000,000</td>
<td>0.4</td>
<td>1.4</td>
</tr>
<tr>
<td>6/08 - 6/09 (3)</td>
<td>IBM Roadrunner -PowerXCell 8i 3.2 Ghz</td>
<td>1,105</td>
<td>2,329,599</td>
<td>2.1</td>
<td>2.3</td>
</tr>
<tr>
<td>11/09 - 6/10 (2)</td>
<td>Cray Jaguar - XT5-HE 2.6 GHz</td>
<td>1,759.</td>
<td>5,474,272</td>
<td>17.3</td>
<td>6.9</td>
</tr>
<tr>
<td>11/10 (1)</td>
<td>NUDT Tianhe-1A, X5670 2.93Ghz NVIDIA</td>
<td>2,566</td>
<td>3,600,000</td>
<td>3.4</td>
<td>4.0</td>
</tr>
<tr>
<td>6/11 - 11/11 (2)</td>
<td>Fujitsu K computer, SPARC64 VIlIfx</td>
<td>10,510</td>
<td>11,870,208</td>
<td>29.5</td>
<td>9.9</td>
</tr>
<tr>
<td>6/12 (1)</td>
<td>IBM Sequoia BlueGene/Q</td>
<td>16,324</td>
<td>12,681,215</td>
<td>23.1</td>
<td>7.9</td>
</tr>
<tr>
<td>11/12 (1)</td>
<td>Cray XK7 Titan AMD + NVIDIA Kepler</td>
<td>17,590</td>
<td>4,423,680</td>
<td>0.9</td>
<td>8.2</td>
</tr>
<tr>
<td>6/13 - 11/13(?)</td>
<td>NUDT Tianhe-2 Intel IvyBridge &amp; Xeon Phi</td>
<td>33,862</td>
<td>9,960,000</td>
<td>5.4</td>
<td>17.8</td>
</tr>
</tbody>
</table>
Ugly Things about HPL

• Doesn’t probe the architecture; only one data point
• Constrains the technology and architecture options for HPC system designers.
  • Skews system design.
• Floating point benchmarks are not quite as valuable to some as data-intensive system measurements
Many Other Benchmarks

- Top 500
- Green 500
- Graph 500-161
- Sustained Petascale Performance
- HPC Challenge
- Perfect
- ParkBench
- SPEC-hpc
- Livermore Loops
- EuroBen
- NAS Parallel Benchmarks
- Genesis
- RAPS
- SHOC
- LAMMPS
- Dhrystone
- Whetstone
Proposal: HPCG

• High Performance Conjugate Gradient (HPCG).
• Solves $Ax=b$, $A$ large, sparse, $b$ known, $x$ computed.
• An optimized implementation of PCG contains essential computational and communication patterns that are prevalent in a variety of methods for discretization and numerical solution of PDEs

• Patterns:
  • Dense and sparse computations.
  • Dense and sparse collective.
  • Data-driven parallelism (unstructured sparse triangular solves).
• Strong verification and validation properties (via spectral properties of CG).
Model Problem Description

- Synthetic discretized 3D PDE (FEM, FVM, FDM).
- Single DOF heat diffusion model.
- Zero Dirichlet BCs, Synthetic RHS s.t. solution = 1.
- Local domain: \((n_x \times n_y \times n_z)\)
- Process layout: \((np_x \times np_y \times np_z)\)
- Global domain: \((n_x \times np_x) \times (n_y \times np_y) \times (n_z \times np_z)\)
- Sparse matrix:
  - 27 nonzeros/row interior.
  - 7 – 18 on boundary.
  - Symmetric positive definite.
Example

• Build HPCG with default MPI and OpenMP modes enabled.
  
  export OMP_NUM_THREADS=1
  
  mpiexec –n 96 ./xhpcg 70 80 90

• Results in:
  
  \( n_x = 70, \ n_y = 80, \ n_z = 90 \)

  \( np_x = 4, \ np_y = 4, \ np_z = 6 \)

• Global domain dimensions: 280-by-320-by-540

• Number of equations per MPI process: 504,000

• Global number of equations: 48,384,000

• Global number of nonzeros: 1,298,936,872

• Note: Changing OMP_NUM_THREADS does not change any of these values.

http://tiny.cc/hpcg
CG ALGORITHM

- \( p_0 := x_0, r_0 := b - Ap_0 \)
- Loop \( i = 1, 2, \ldots \)
  - \( z_i := M^{-1}r_{i-1} \)
  - \( \text{if } i = 1 \)
    - \( p_i := z_i \)
    - \( a_i := \text{dot_product}(r_{i-1}, z) \)
  - \( \text{else} \)
    - \( a_i := \text{dot_product}(r_{i-1}, z) \)
    - \( b_i := a_i / a_{i-1} \)
    - \( p_i := b_i * p_{i-1} + z_i \)
  - \( \text{end if} \)
  - \( a_i := \text{dot_product}(r_{i-1}, z_i) / \text{dot_product}(p_i, A*p_i) \)
  - \( x_{i+1} := x_i + a_i * p_i \)
  - \( r_i := r_{i-1} - a_i * A*p_i \)
  - \( \text{if } ||r_i||_2 < \text{tolerance} \) then Stop
- end Loop
Problem Setup

- Construct Geometry.
- Generate Problem.
- Setup Halo Exchange.
- Initialize Sparse Meta-data.
- Call user-defined OptimizeProblem function. This function permits the user to change data structures and perform permutation that can improve execution.

Validation Testing

- Perform spectral properties CG Tests:
  - Convergence for 10 distinct eigenvalues:
    - No preconditioning.
    - With Preconditioning
  - Symmetry tests:
    - Sparse MV kernel.
    - Symmetric Gauss-Seidel kernel.

Reference Sparse MV and Gauss-Seidel kernel timing.

- Time calls to the reference versions of sparse MV and symmetric Gauss-Seidel for inclusion in output report.

Reference CG timing and residual reduction.

- Time the execution of 50 iterations of the reference CG implementation.
- Record reduction of residual using the reference implementation. The optimized code must attain the same residual reduction, even if more iterations are required.

Optimized CG Setup.

- Run one set of Optimized CG solver to determine number of iterations required to reach residual reduction of reference CG.
- Record iteration count as numberOfOptCgIters.
- Detect failure to converge.
- Compute how many sets of Optimized CG Solver are required to fill benchmark timespan. Record as numberOfCgSets

Optimized CG timing and analysis.

- Run **numberOfCgSets** calls to optimized CG solver with **numberOfOptCgIters** iterations.
  - For each set, record residual norm.
  - Record total time.
  - Compute mean and variance of residual values.

Report results

- Write a log file for diagnostics and debugging.
- Write a benchmark results file for reporting official information.

Execution: 7 Phases

- Write a log file for diagnostics and debugging.
- Write a benchmark results file for reporting official information.
Problem Setup

- Construct Geometry.
- Generate Problem.
- Setup Halo Exchange.
  - Use symmetry to eliminate communication in this phase.
  - C++ STL containers/algorithms: Simple code, force use of C++.
- Initialize Sparse Meta-data.
- Call user-defined OptimizeProblem function.
  - Permits the user to change data structures and perform permutation that can improve execution.

http://tiny.cc/hpcg
• Temporarily modify matrix diagonals:
  • (2.0e6, 3.0e6, … 9.0e6, 1.0e6, …1.0e6).
  • Offdiagonal still -1.0.
  • Matrix looks diagonal with 10 distinct eigenvalues.
• Perform spectral properties CG Tests:
  • Convergence for 10 distinct eigenvalues:
    • No preconditioning: About 10 iters.
    • With Preconditioning: About 1 iter.
• Symmetry tests:
  • Matrix, preconditioner are symmetric.
  • Sparse MV kernel. \( x^T Ay = y^T Ax \)
  • Symmetric Gauss-Seidel kernel. \( x^T M^{-1} y = y^T M^{-1} x \)
Reference Sparse MV and Gauss-Seidel kernel timing.

- Time calls to the reference versions of sparse MV and symmetric Gauss-Seidel for inclusion in output report.
Reference CG timing and residual reduction.

- Time the execution of 50 iterations of the reference CG implementation.
- Record reduction of residual using the reference implementation.
- The optimized code must attain the same residual reduction, even if more iterations are required.
- Most graph coloring algorithms improve parallel execution at the expense of increasing iteration counts.
Optimized CG Setup.

- Run one set of Optimized CG solver to determine number of iterations required to reach residual reduction of reference CG.
- Record iteration count as `numberOfOptCgIters`.
- Detect failure to converge.
- Compute how many sets of Optimized CG Solver are required to fill benchmark timespan. Record as `numberOfCgSets`
Optimized CG timing and analysis.

- Run `numberOfCgSets` calls to optimized CG solver with `numberOfOptCglIters` iterations.
- For each set, record residual norm.
- Record total time.
- Compute mean and variance of residual values.
Report results

- Write a log file for diagnostics and debugging.
- Write a benchmark results file for reporting official information.
Example

• Reference CG: 50 iterations, residual drop of 1e-6.

• Optimized CG: Run one set of iterations
  • Multicolor ordering for Symmetric Gauss-Seidel:
    • Better vectorization, threading.
    • But: Takes 65 iterations to reach residual drop of 1e-6.

• Overhead:
  • Extra 15 iterations.
  • Computing of multicolor ordering.

• Compute number of sets we must run to fill entire execution time:
  • 5h/time-to-compute-1-set.
  • Results in thousands of CG set runs.

• Run and record residual for each set.
  • Report mean and variance (accounts for non-associativity of FP addition).
Preconditioner

- Symmetric Gauss-Seidel preconditioner
  - (Non-overlapping additive Schwarz)
  - Differentiate latency vs. throughput optimize core sets.

- From Matlab reference code:
  Setup:
  \[
  LA = \text{tril}(A); \ UA = \text{triu}(A); \ DA = \text{diag}(\text{diag}(A));
  \]
  Solve:
  \[
  x = LA\backslash y;
  x1 = y - LA*x + DA*x; \text{ % Subtract off extra diagonal contribution}
  x = UA\backslash x1;
  \]
Key Computation Data Patterns

- **Domain decomposition:**
  - SPMD (MPI): Across domains.
  - Thread/vector (OpenMP, compiler): Within domains.

- **Vector ops:**
  - AXPY: Simple streaming memory ops.
  - DOT/NRM2 : Blocking Collectives.

- **Matrix ops:**
  - SpMV: Classic sparse kernel (option to reformat).
  - Symmetric Gauss-Seidel: sparse triangular sweep.
    - Exposes real application tradeoffs:
      - threading & convergence vs. SPMD and scaling.
Merits of HPCG

• Includes major communication/computational patterns.
  • Represents a minimal collection of the major patterns.

• Rewards investment in:
  • High-performance collective ops.
  • Local memory system performance.
  • Low latency cooperative threading.

• Detects and measures variances from bitwise identical computations.
COMPUTATIONAL RESULTS
GFLOPS/s “Shock”

Results for Cielo
Dual Socket AMD (8 core) Magny Cour
Each node is 2*8 Cores 2.4 GHz = Total 153.6 Gflops/

<table>
<thead>
<tr>
<th>Mira Partition Size</th>
<th>Peak Gflops</th>
<th>Sustained Gflops</th>
<th>% of peak</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 nodes</td>
<td>13107.2</td>
<td>73.4</td>
<td>0.56%</td>
</tr>
<tr>
<td>128 nodes</td>
<td>26214.4</td>
<td>147.43</td>
<td>0.56%</td>
</tr>
<tr>
<td>256 nodes</td>
<td>52428.8</td>
<td>293.8</td>
<td>0.56%</td>
</tr>
<tr>
<td>512 nodes</td>
<td>104857.6</td>
<td>587.97</td>
<td>0.56%</td>
</tr>
<tr>
<td>1024 nodes</td>
<td>209715.2</td>
<td>1176.69</td>
<td>0.56%</td>
</tr>
<tr>
<td>49152 nodes</td>
<td>10066329.6</td>
<td>55177.6</td>
<td>0.55%</td>
</tr>
</tbody>
</table>

Courtesy Kalyan Kumaran, Argonne

http://tiny.cc/hpcg

512 MPI Processes

Courtesy Mahesh Rajan, Sandia
Cielo, Red Sky, Edison, SID

Results courtesy of Ludovic Saugé, Bull

http://tiny.cc/hpcg

Results courtesy of M. Rajan, D. Doerfler, Sandia
HPCG GFLOP/s on Sequoia: MPI x OpenMP
6.29M total threads, 1.57T equations

Sequoia Results

Results courtesy of Ian Karlin, Scott Futral, LLNL
Tuning result on the K computer

- Parallel scalability shouldn’t be an obstacle for large scale problem
- We are focusing on single CPU performance improvement

Summary of “as is” code on the K

- Total x10 speed up now
  - Continuous memory for matrix
  - Multi-coloring for SYMGS multi-threading
- Under Studying
  - Node re-ordering for SPMV
  - Advanced matrix storage way
  - And so on

8 Processes, 8 Threads/Process (Peak 128x8 GFLOPS)

- Slide courtesy Naoya Maruyama, RIKEN AICS, and Fujitsu
Next Steps

• Validate against real apps on real machines.
  • Validate ranking and driver potential.
  • Modify code as needed.
  • Considering multi-level preconditioner.
    • Discussions with LBL show potential to enrich design tradeoff space
  • Repeat as necessary.

• Introduce to broader community.
  • HPCG 1.0 released today.

• Notes:
  • Simple is best.
  • First version need not be last version (HPL evolved).

Graph courtesy Future Technology Group, LBL
HPCG and HPL

- We are NOT proposing to eliminate HPL as a metric.

- The historical importance and community outreach value is too important to abandon.

- HPCG will serve as an alternate ranking of the Top500.
  - Similar perhaps to the Green500 listing.
HPCG Tech Reports

Toward a New Metric for Ranking High Performance Computing Systems
  • Jack Dongarra and Michael Heroux

HPCG Technical Specification
  • Jack Dongarra, Michael Heroux, Piotr Luszczek

• http://tiny.cc/hpcg