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

Jack Dongarra & Piotr Luszczek
University of Tennessee/ORNL

Michael Heroux
Sandia National Labs
Confessions of an Accidental Benchmarker

- Appendix B of the LINPACK Users’ Guide
  - Designed to help users extrapolate execution LINPACK software package
- First benchmark report from 1977;
  - Cray 1 to DEC PDP-10
Started 36 Years Ago

- In the late 70’s the fastest computer ran LINPACK at 14 Mflop/s
- In the late 70’s floating point operations were expensive compared to other operations and data movement
- Matrix size, n = 100
  - That’s what would fit in memory
- LINPACK code is based on “right-looking” algorithm:
  - O(n³) Flop/s and O(n³) data movement
Benchmarks Evolve: From LINPACK to HPL to TOP500

- LINPACK Benchmark report, ANL TM-23, 1984
  - Performance of Various Computers Using Standard Linear Equations Software, listed about 70 systems.
- Over time the LINPACK Benchmark went through a number of changes.
  - Began with Fortran code, run the code as is, no changes, N = 100 (Table 1)
  - Later N = 1000 introduced, hand coding to allow for optimization and parallelism (Table 2)
  - Timing harness provided to generate matrix, check the solution
  - The basic algorithm, GE/PP, remained the same.
- 1989 started putting together Table 3 (Toward Peak Performance) of the LINPACK benchmark report.
  - N allowed to be any size
  - Timing harness provided to generate matrix, check the solution
  - List $R_{\text{max}}, N_{\text{max}}, R_{\text{peak}}$
- In 2000 we put together an optimized implementation of the benchmark, called High Performance LINPACK or HPL.
  - Just needs optimized version of BLAS and MPI.
TOP500

• In 1986 Hans Meuer started a list of supercomputer around the world, they were ranked by peak performance.
• Hans approached me in 1992 to merge our lists into the “TOP500”.
• The first TOP500 list was in June 1993.
Rules For HPL and TOP500

• Have to compute the solution to a prescribed accuracy.
• Excludes the use of a fast matrix multiply algorithm like "Strassen's Method”
• Algorithms which compute a solution in a precision lower than full precision (64 bit floating point arithmetic) and refine the solution using an iterative approach.
• The authors of the TOP500 reserve the right to independently verify submitted LINPACK results, and exclude computer from the list which are not valid or not general purpose in nature.
• By general purpose computer we mean that the computer system must be able to be used to solve a range of scientific problems.
• Any computer designed specifically to solve the LINPACK benchmark problem or have as its major purpose the goal of a high TOP500 ranking will be disqualified.
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 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.
HPL has a Number of Problems

• 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 an Exaflop 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!!

- 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
#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>0.060</td>
<td>52224</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>11/93 (1)</td>
<td>Fujitsu Numerical Wind Tunnel</td>
<td>0.124</td>
<td>31920</td>
<td>0.1</td>
<td>1.</td>
</tr>
<tr>
<td>6/94 (1)</td>
<td>Intel XP/S140</td>
<td>0.143</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>0.170</td>
<td>42000</td>
<td>0.1</td>
<td>1.</td>
</tr>
<tr>
<td>6/96 (1)</td>
<td>Hitachi SR2201/1024</td>
<td>0.220</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>0.368</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>0.85</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 VIIfx</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 (2)</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

- TOP500
- Green 500
- Graph 500-160
- Sustained Petascale Performance
- HPC Challenge
- Perfect
- ParkBench
- SPEC-hpc
- Big Data Top100
- Livermore Loops
- EuroBen
- NAS Parallel Benchmarks
- Genesis
- RAPS
- SHOC
- LAMMPS
- Dhrystone
- Whetstone
- I/O Benchmarks
Goals for New Benchmark

• Augment the TOP500 listing with a benchmark that correlates with important scientific and technical apps not well represented by HPL

• Encourage vendors to focus on architecture features needed for high performance on those important scientific and technical apps.
  • Stress a balance of floating point and communication bandwidth and latency
  • Reward investment in high performance collective ops
  • Reward investment in high performance point-to-point messages of various sizes
  • Reward investment in local memory system performance
  • Reward investment in parallel runtimes that facilitate intra-node parallelism

• Provide an outreach/communication tool
  • Easy to understand
  • Easy to optimize
  • Easy to implement, run, and check results

• Provide a historical database of performance information
  • The new benchmark should have longevity
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
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.
HPCG Design Philosophy

• Relevance to broad collection of important apps.
• Simple, single number.
• Few user-tunable parameters and algorithms:
  • The system, not benchmarker skill, should be primary factor in result.
  • Algorithmic tricks don’t give us relevant information.
• Algorithm (PCG) is vehicle for organizing:
  • Known set of kernels.
  • Core compute and data patterns.
  • Tunable over time (as was HPL).
• Easy-to-modify:
  • _ref kernels called by benchmark kernels.
  • User can easily replace with custom versions.
  • Clear policy: Only kernels with _ref versions can be modified.
PCG ALGORITHM

◆ $p_0 := x_0$, $r_0 := b - Ap_0$
◆ Loop $i = 1, 2, ...$
  ○ $z_i := M^{-1}r_{i-1}$
  ○ if $i = 1$
    ▪ $p_i := z_i$
    ▪ $\alpha_i := \text{dot_product}(r_{i-1}, z)$
  ○ else
    ▪ $\alpha_i := \text{dot_product}(r_{i-1}, z)$
    ▪ $\beta_i := \alpha_i / \alpha_{i-1}$
    ▪ $p_i := \beta_i * p_{i-1} + z_i$
  ○ end if
  ○ $\alpha_i := \text{dot_product}(r_{i-1}, z) / \text{dot_product}(p_i, A*p_i)$
  ○ $x_{i+1} := x_i + \alpha_i * p_i$
  ○ $r_i := r_{i-1} - \alpha_i * A*p_i$
  ○ 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 PCG Tests:
  - Convergence for 10 distinct eigenvalues:
    - No preconditioning.
    - With Preconditioning
  - Symmetry tests:
    - Sparse MV kernel.
    - MG kernel.
- Time calls to the reference versions of sparse MV and MG for inclusion in output report.

**Reference Sparse MV and Gauss-Seidel kernel timing.**

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

**Execution: 7 Phases**

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

**Optimized CG timing and analysis.**
- Run **numberOfCgSets** calls to optimized PCG 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.
Preconditioner

- Hybrid geometric/algebraic multigrid:
  - Grid operators generated synthetically:
    - Coarsen by 2 in each x, y, z dimension (total of 8 reduction each level).
    - Use same GenerateProblem() function for all levels.
  - Grid transfer operators:
    - Simple injection. Crude but…
    - Requires no new functions, no repeat use of other functions.
    - Cheap.
  - Smoother:
    - Symmetric Gauss-Seidel [ComputeSymGS()].
    - Except, perform halo exchange prior to sweeps.
    - Number of pre/post sweeps is tuning parameter.
  - Bottom solve:
    - Right now just a single call to ComputeSymGS().

Symmetric Gauss-Seidel preconditioner
- In Matlab that might look like:

\[
\begin{align*}
LA &= \text{tril}(A); \quad UA = \text{triu}(A); \quad DA = \text{diag(diag}(A)); \\
x &= LA\backslash y; \\
x1 &= y - LA^*x + DA^*x; \quad \% \text{Subtract off extra diagonal contribution} \\
x &= UA\backslash x1;
\end{align*}
\]

(In 2D, something like this)
HPCG Parameters

- Iterations per set: 50.
- Total benchmark time for official result:
  - Repeated until 3600 seconds (1 hour run).
  - Anything less is reported as a “tuning” result.
- Coarsening: 2x – 2x – 2x (8x total).
- Number of levels:
  - 4 (including finest level).
  - Requires nx, ny, nz divisible by 8.
- Pre/post smoother sweeps: 1 each.
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/measures variances from bitwise reproducibility.

- Executes kernels at several (tunable) granularities:
  - \( nx = ny = nz = 104 \) gives
  - \( n_{\text{local}} = 1,124,864; 140,608; 17,576; 2,197 \)
  - ComputeSymGS with multicoloring adds one more level:
    - 8 colors.
    - Average size of color = 275.
    - Size ratio (largest:smallest): 4096
  - Provide a “natural” incentive to run a big problem.
Performance “Shock”

Results for Cielo
Dual Socket AMD (8 core) Magny Cour
Each node is 2*8 Cores 2.4 GHz = Total 153.6 Gflops/
LANL’s Cray XT3

<table>
<thead>
<tr>
<th>Mira Partition Size</th>
<th>Peak Gflops</th>
<th>Sustained Gflops</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANL’s IBM BG/Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 nodes</td>
<td>13107.2</td>
<td>73.4</td>
</tr>
<tr>
<td>128 nodes</td>
<td>26214.4</td>
<td>147.43</td>
</tr>
<tr>
<td>256 nodes</td>
<td>52428.8</td>
<td>293.8</td>
</tr>
<tr>
<td>512 nodes</td>
<td>104857.6</td>
<td>587.97</td>
</tr>
<tr>
<td>1024 nodes</td>
<td>209715.2</td>
<td>1176.69</td>
</tr>
<tr>
<td>49152 nodes</td>
<td>10066329.6</td>
<td>55177.6</td>
</tr>
</tbody>
</table>

% of peak

 Courtesy Kalyan Kumaran, Argonne

Courtesy Mahesh Rajan, Sandia

512 MPI Processes
Tuning result on the K computer

Summary of “as is” code on the K
- Parallel scalability shouldn’t be obstacle for large scale problem
- We are focusing on single CPU performance improvement

Improvement
- 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)

http://tiny.cc/hpcg

Slide courtesy Naoya Maruyama, RIKEN AICS, and Fujitsu
Multi-node Scaling

Stampede cluster, dual socket of 8-core SNB, 2.7 GHz
2 MPI processes per node (1 MPI process per skt. for NUMA)
160³ input per MPI process
93% parallelization efficiency with 1024 nodes

University of Texas Austin, NSF’s Stampede system
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.
Signs of Interest and Uptake

- Input from various people at DOE Labs
- Discussions with and results from every HPC vendor.
  - Major, deep technical discussions with several.
- Same with most LCFs.
- Intel-sponsored SC’14 Workshop on Optimizing HPCG.
HPCG Tech Reports

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

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

• http://tiny.cc/hpcg