HPCG: ONE YEAR LATER

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

LINPACK code is based on “right-looking” algorithm:
O(n^3) Flop/s and O(n^3) data movement
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 put together our lists into the “TOP500”.
- The first TOP500 list was in June 1993.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>System</th>
<th>Cores</th>
<th>Rmax (GFlop/s)</th>
<th>Rpeak (GFlop/s)</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Los Alamos National Laboratory United States</td>
<td>CM-5/1024 Thinking Machines Corporation</td>
<td>1,024</td>
<td>59.7</td>
<td>131.0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Minnesota Supercomputer Center United States</td>
<td>CM-5/544 Thinking Machines Corporation</td>
<td>544</td>
<td>30.4</td>
<td>69.6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>National Security Agency United States</td>
<td>CM-5/512 Thinking Machines Corporation</td>
<td>512</td>
<td>30.4</td>
<td>65.5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>NCSA United States</td>
<td>CM-5/512 Thinking Machines Corporation</td>
<td>512</td>
<td>30.4</td>
<td>65.5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>NEC Japan</td>
<td>SX-3/44 NEC</td>
<td>4</td>
<td>23.2</td>
<td>25.5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Atmospheric Environment Service (AES)</td>
<td>SX-3/44</td>
<td>4</td>
<td>20.0</td>
<td>22.0</td>
<td></td>
</tr>
</tbody>
</table>
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 37 years old
  - TOP500 (HPL) is 21.5 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
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 500-174
- Green/Graph
- Sustained Petascale Performance
- HPC Challenge
- Perfect
- ParkBench
- SPEC-hpc

- Livermore Loops
- EuroBen
- NAS Parallel Benchmarks
- Genesis
- RAPS
- SHOC
- LAMMPS
- Dhrystone
- Whetstone
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.
  • Multi-scale execution of kernels via MG (truncated) V cycle.
  • Data-driven parallelism (unstructured sparse triangular solves).
• Strong verification and validation properties (via spectral properties of PCG).
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.
Example

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

- Results in:
  
  \[ n_x = 70, \quad n_y = 80, \quad n_z = 90 \]
  
  \[ np_x = 4, \quad np_y = 4, \quad np_z = 6 \]

- 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
PCG ALGORITHM

◆ $p_0 := x_0$, $r_0 := b - Ap_0$
◆ Loop $i = 1, 2, \ldots$
  o $z_i := M^{-1}r_{i-1}$
  o if $i = 1$
    ▪ $p_i := z_i$
    ▪ $a_i := \text{dot\_product}(r_{i-1}, z)$
  o else
    ▪ $a_i := \text{dot\_product}(r_{i-1}, z)$
    ▪ $b_i := a_i / a_{i-1}$
    ▪ $p_i := b_i * p_{i-1} + z_i$
  o end if
  o $a_i := \text{dot\_product}(r_{i-1}, z_i) / \text{dot\_product}(p_i, A*p_i)$
  o $x_{i+1} := x_i + a_i * p_i$
  o $r_i := r_{i-1} - a_i * A*p_i$
  o if $\|r_i\|_2 < \text{tolerance}$ then Stop
◆ end Loop
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().
    • If no coarse grids, has identical behavior as HPCG 1.X.

• Symmetric Gauss-Seidel preconditioner
  • In Matlab that might look like:
    
    LA = tril(A); UA = triu(A); DA = diag(diag(A));

    x = LA\y;
    x1 = y - LA*x + DA*x; % Subtract off extra diagonal contribution
    x = UA\x1;
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.

Reference Sparse MV and Gauss-Seidel kernel timing.
- Time calls to the reference versions of sparse MV and MG for inclusion in output report.

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.

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.
Example

• Reference PCG: 50 iterations, residual drop of 1e-6.
• Optimized PCG: Run one set of iterations
  • Multicolor ordering for Symmetric Gauss-Seidel:
    • Better vectorization, threading.
    • But: Takes 55 iterations to reach residual drop of 1e-6.
  • Overhead:
    • Extra 5 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).
HPCG Parameters

- Iterations per set: 50.
- Total benchmark time for official result:
  - 3600 seconds.
  - Anything less is reported as a “tuning” result.
  - Default time 60 seconds.
- 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.
- Setup time: Amortized over 500 iterations.
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.
    • Enables leverage of new parallel patterns, e.g., futures.
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
  - $n_{local} = 1,124,864; 140,608; 17,576; 2,197$
  - 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.
User tuning options

• **MPI ranks vs. threads:**
  - MPI-only: Strong algorithmic incentive to use.
  - MPI+X: Strong resource management incentive to use.

• **Data structures:**
  - Sparse and dense.
  - May not use knowledge of special sparse structure.
  - May not exploit regularity in data structures (x or y must be accessed indirectly when computing y = Ax).
  - Overhead of analysis/ transformation is counted against time for ten 50 iteration sets (500 iterations).
User tuning options

• Permutations:
  • Can permute matrix for ComputeSpMV or ComputeMG or both.
  • Overhead is counted as with data structure transformations.

• Not permitted:
  • Algorithm changes to CG or MG that change behavior beyond permutations or FP arithmetic.
  • Change in FP precision.
  • Almost anything else not mentioned.
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.
  - Or maybe top 50 for now.
HPCG 3.X Features

• Truer C++ design:
  • Have gradually moved in that direction.
  • No one has complained.

• Request permutation vectors:
  • Permits explicit check against reference kernel results.

• Kernels will remain the same:
  • No disruption of vendor investments.
On Going Discussion and Feedback

- June 2013
  - Discussed at ISC
- November 2013
  - Discussed at SC13 in Denver during Top500 BoF
- January 2014
  - Discussed at DOE workshop
- March 2014
  - Discussed in DC at workshop
- June 2014
  - ISC talk at session
Signs of Uptake

- Discussions with and results from every vendor.
- Major, deep technical discussions with several.
- Same with most LCFs.
- SC’14 BOF on Optimizing HPCG.
- One ISC’14 and two SC’14 papers submitted.
  - Nvidia and Intel. 2/3 accepted.
- Optimized results for x86, MIC-based, Nvidia GPU-based systems.
HPL vs. HPCG: Bookends

- Some see HPL and HPCG as “bookends” of a spectrum.
  - Applications teams know where their codes lie on the spectrum.
  - Can gauge performance on a system using both HPL and HPCG numbers.

- Problem of HPL execution time still an issue:
  - Need a lower cost option. End-to-end HPL runs are too expensive.
  - Work in progress.
<table>
<thead>
<tr>
<th>Site</th>
<th>Computer</th>
<th>Cores</th>
<th>HPL Rmax (Pflops)</th>
<th>HPL Rank</th>
<th>HPCG (Pflops)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSCC / Guangzhou</td>
<td>Tianhe-2 NUDT, Xeon 12C 2.2GHz + Intel Xeon Phi 57C + Custom</td>
<td>3,120,000</td>
<td>33.9</td>
<td>1</td>
<td>.580</td>
</tr>
<tr>
<td>RIKEN Advanced Inst for Comp Sci</td>
<td>K computer Fujitsu SPARC64 VIIIfx 8C + Custom</td>
<td>705,024</td>
<td>10.5</td>
<td>4</td>
<td>.427</td>
</tr>
<tr>
<td>DOE/OS Oak Ridge Nat Lab</td>
<td>Titan, Cray XK7 AMD 16C + Nvidia Kepler GPU 14C + Custom</td>
<td>560,640</td>
<td>17.6</td>
<td>2</td>
<td>.322</td>
</tr>
<tr>
<td>DOE/OS Argonne Nat Lab</td>
<td>Mira BlueGene/Q, Power BQC 16C 1.60GHz + Custom</td>
<td>786,432</td>
<td>8.59</td>
<td>5</td>
<td>.101#</td>
</tr>
<tr>
<td>Swiss CSFS</td>
<td>Piz Daint, Cray XC30, Xeon 8C + Nvidia Kepler 14C + Custom</td>
<td>115,984</td>
<td>6.27</td>
<td>6</td>
<td>.099</td>
</tr>
<tr>
<td>Leibniz Rechenzentrum</td>
<td>SuperMUC, Intel 8C + IB</td>
<td>147,456</td>
<td>2.90</td>
<td>12</td>
<td>.0833</td>
</tr>
<tr>
<td>CEA/TGCC-GENCI</td>
<td>Curie tine nodes Bullx B510 Intel Xeon 8C 2.7 GHz + IB</td>
<td>79,504</td>
<td>1.36</td>
<td>26</td>
<td>.0491</td>
</tr>
<tr>
<td>Exploration and Production</td>
<td>HPC2, Intel Xeon 10C 2.8 GHz + Nvidia Kepler 14C + IB</td>
<td>62,640</td>
<td>3.00</td>
<td>11</td>
<td>.0489</td>
</tr>
<tr>
<td>Eni S.p.A.</td>
<td>Edison Cray XC30, Intel Xeon 12C 2.4GHz + Custom</td>
<td>132,840</td>
<td>1.65</td>
<td>18</td>
<td>.0439 #</td>
</tr>
<tr>
<td>Texas Advanced Computing</td>
<td>Stampede, Dell Intel (8c) + Intel Xeon Phi (61c) + IB</td>
<td>78,848</td>
<td>.881*</td>
<td>7</td>
<td>.0161</td>
</tr>
<tr>
<td>Center</td>
<td>Beaufix Bullx B710 Intel Xeon 12C 2.7 GHz + IB</td>
<td>24,192</td>
<td>.469 (.467*)</td>
<td>79</td>
<td>.0110</td>
</tr>
<tr>
<td>Meteo France</td>
<td>Prolix Bullx B710 Intel Xeon 2.7 GHz 12C + IB</td>
<td>23,760</td>
<td>.464 (.415*)</td>
<td>80</td>
<td>.00998</td>
</tr>
<tr>
<td>Meteo France</td>
<td>CALMIP Bullx DLC Intel Xeon 10C 2.8 GHz + IB</td>
<td>12,240</td>
<td>.255</td>
<td>184</td>
<td>.00725</td>
</tr>
<tr>
<td>U of Toulouse</td>
<td>Wilkes, Intel Xeon 6C 2.6 GHz + Nvidia Kepler 14C + IB</td>
<td>3584</td>
<td>.240</td>
<td>201</td>
<td>.00385</td>
</tr>
<tr>
<td>Cambridge U</td>
<td>TUSBAME-KFC Intel Xeon 6C 2.1 GHz + IB</td>
<td>2720</td>
<td>.150</td>
<td>436</td>
<td>.00370</td>
</tr>
</tbody>
</table>

* scaled to reflect the same number of cores
# unoptimized implementation
<table>
<thead>
<tr>
<th>Site</th>
<th>Computer</th>
<th>Cores</th>
<th>HPL Max (Pflops)</th>
<th>HPL Rank</th>
<th>HPCG (Pflops)</th>
<th>HPCG/HPL</th>
<th>HPL Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSCC / Guangzhou</td>
<td>Tianhe-2 NUDT, Xeon 12C 2.2GHz + Intel Xeon Phi 57C + Custom</td>
<td>3,120,000</td>
<td>33.9</td>
<td>1</td>
<td>.580</td>
<td>1.7%</td>
<td>1</td>
</tr>
<tr>
<td>RIKEN Advanced Inst for Comp Sci</td>
<td>K computer Fujitsu SPARC64 VIIIfx 8C + Custom</td>
<td>705,024</td>
<td>10.5</td>
<td>4</td>
<td>.427</td>
<td>4.1%</td>
<td>2</td>
</tr>
<tr>
<td>DOE/OS Oak Ridge Nat Lab</td>
<td>Titan, Cray XK7 AMD 16C + Nvidia Kepler GPU 14C + Custom</td>
<td>560,640</td>
<td>17.6</td>
<td>2</td>
<td>.322</td>
<td>1.8%</td>
<td>5</td>
</tr>
<tr>
<td>DOE/OS Argonne Nat Lab</td>
<td>Mira BlueGene/Q, Power BQC 16C 1.60GHz + Custom</td>
<td>786,432</td>
<td>8.59</td>
<td>5</td>
<td>.101#</td>
<td>1.2%</td>
<td></td>
</tr>
<tr>
<td>Swiss CSCS</td>
<td>Piz Daint, Cray XC30, Xeon 8C + Nvidia Kepler 14C + Custom</td>
<td>115,984</td>
<td>6.27</td>
<td>6</td>
<td>.099</td>
<td>1.6%</td>
<td></td>
</tr>
<tr>
<td>Leibniz Rechenzentrum</td>
<td>SuperMUC, Intel 8C + IB</td>
<td>147,456</td>
<td>2.90</td>
<td>12</td>
<td>.0833</td>
<td>2.9%</td>
<td></td>
</tr>
<tr>
<td>CEA/TGCC-GENCI</td>
<td>Curie tine nodes Bullx B510 Intel Xeon 8C 2.7 GHz + IB</td>
<td>79,504</td>
<td>1.36</td>
<td>26</td>
<td>.0491</td>
<td>3.6%</td>
<td></td>
</tr>
<tr>
<td>Exploration and Production Eni S.p.A.</td>
<td>HPC2, Intel Xeon 10C 2.8 GHz + Nvidia Kepler 14C + IB</td>
<td>62,640</td>
<td>3.00</td>
<td>11</td>
<td>.0489</td>
<td>1.6%</td>
<td></td>
</tr>
<tr>
<td>DOE/OS L Berkeley Nat Lab</td>
<td>Edison Cray XC30, Intel Xeon 12C 2.4GHz + Custom</td>
<td>132,840</td>
<td>1.65</td>
<td>18</td>
<td>.0439*</td>
<td>2.7%</td>
<td></td>
</tr>
<tr>
<td>Texas Advanced Computing Center</td>
<td>Stampede, Dell Intel (8c) + Intel Xeon Phi (61c) + IB</td>
<td>78,848</td>
<td>.881*</td>
<td>7</td>
<td>.0161</td>
<td>1.8%</td>
<td></td>
</tr>
<tr>
<td>Meteo France</td>
<td>Beaufix Bullx B710 Intel Xeon 12C 2.7 GHz + IB</td>
<td>24,192</td>
<td>.469 (.467*)</td>
<td>79</td>
<td>.0110</td>
<td>2.4%</td>
<td></td>
</tr>
<tr>
<td>Meteo France</td>
<td>Prolix Bullx B710 Intel Xeon 2.7 GHz 12C + IB</td>
<td>23,760</td>
<td>.464 (.415*)</td>
<td>80</td>
<td>.00998</td>
<td>2.4%</td>
<td></td>
</tr>
<tr>
<td>U of Toulouse</td>
<td>CALMIP Bullx DLC Intel Xeon 10C 2.8 GHz + IB</td>
<td>12,240</td>
<td>.255</td>
<td>184</td>
<td>.00725</td>
<td>2.8%</td>
<td></td>
</tr>
<tr>
<td>Cambridge U</td>
<td>Wilkes, Intel Xeon 6C 2.6 GHz + Nvidia Kepler 14C + IB</td>
<td>3584</td>
<td>.240</td>
<td>201</td>
<td>.00385</td>
<td>1.6%</td>
<td></td>
</tr>
<tr>
<td>TiTech</td>
<td>TUSBAME-KFC Intel Xeon 6C 2.1 GHz + IB</td>
<td>2720</td>
<td>.150</td>
<td>436</td>
<td>.00370</td>
<td>2.5%</td>
<td></td>
</tr>
</tbody>
</table>

* scaled to reflect the same number of cores
# unoptimized implementation
Optimized Versions of HPCG

```
```

---

### Intel

- MKL has packaged CPU version of HPCG
- In the process of packaging Xeon Phi version to be released soon.

### Nvidia

- Massimiliano Fatica and Everett Phillips
- Binary available
  - Contact Massimiliano mfatica@nvidia.com

### Bull

- Developed by CEA requesting the release
Nvidia has it on their ARM64+K20

GPUs provide ARM64 server vendors with the muscle to tackle HPC workloads, enabling them to build high-performance systems that maximize the ARM architecture’s power efficiency and system configurability.

The first GPU-accelerated ARM64 development platforms will be available in July from Cirrascale Corp. and E4 Computer Engineering, with production systems expected to ship later this year. The Eurotech Group also plans to ship production systems later this year.

### HPCG Benchmark

<table>
<thead>
<tr>
<th>Architecture</th>
<th>GFlops</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Gene + K20</td>
<td>15.8</td>
</tr>
<tr>
<td>X86 + K20</td>
<td>15.9</td>
</tr>
</tbody>
</table>

Recompiled to ARM64

- (No optimizations)
- ARM: Applied Micro
- X-Gene CPU (X3-208)
- X86: Xeon E5-2697

### TCO: Driven by Server Cost + Power

**Customer Considerations:**

- Server Cost + Power drives nearly 90% of TCO in the data center
- Power is critical constraint for expansion/build-out of new data centers
- Viable alternative to x86 architecture is needed

8 ARM V8 64b CUSTOM CORES @ 2.4GHZ

+ TESLA K20 GPU ACCELERATOR 1TFLOPS + SYSTEM

STRONG SINGLE THREAD PERFORMANCE

HIGH BANDWIDTH MEMORY SUBSYSTEM
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