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

Jack Dongarra & Piotr Luszczek
University of Tennessee/ORNL

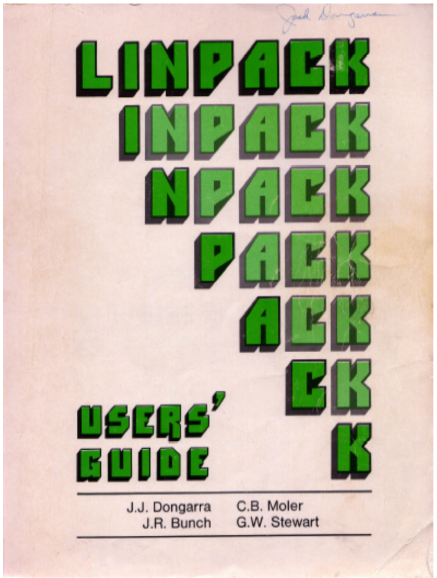
Michael Heroux
Sandia National Labs

See: http//tiny.cc/hpcg

http://tiny.cc/hpcg

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

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

$\frac{2}{3}n^3 + 2n^2$ ops / time

UNIT = 10**6 TIME/(1/3 100**3 + 100**2)

Facility		TIME N=100 secs.	UNIT micro-secs.	Computer	Type	Compiler
NCAR	14.0	.049	0.14	CRAY-1	S	CFT, Assembly BLAS
LASL	4.64	.148	0.43	CDC 7600	S	FTN, Assembly BLAS
NCAR	3.58	.192	0.56	CRAY-1	S	CFT
LASL	3.27	.210	0.61	CDC 7600	S	FTN
Argonne	2.31	.297	0.86	IBM 370/195	D	H
NCAR	1.91	.359	1.05	CDC 7600	S	Local
Argonne	1.77	.388	1.33	IBM 3033	D	H
NASA Langley	1.40	.489	1.42	CDC Cyber 175	S	FTN
U. Ill. Urbana	1.36	.506	1.47	CDC Cyber 175	S	Ext. 4.6
LLL	1.24	.554	1.61	CDC 7600	S	CHAT, No optimize
SLAC	1.19	.579	1.69	IBM 370/168	D	H Ext., Fast mult.
Michigan	1.09	.631	1.84	Amdahl 470/V6	D	H
Toronto	.772	.890	2.59	IBM 370/165	D	H Ext., Fast mult.
Northwestern	.477	1.44	4.20	CDC 6600	S	FTN
Texas	.356	1.93*	5.63	CDC 6600	S	RUN
China Lake	.352	1.95*	5.69	Univac 1110	S	V
Yale	.265	2.59	7.53	DEC KL-20	S	F20
Bell Labs	.197	3.46	10.1	Honeywell 6080	S	Y
Wisconsin	.197	3.49	10.1	Univac 1110	S	V
Iowa State	.194	3.54	10.2	Itel AS/5 mod3	D	H
U. Ill. Chicago	[illegible]	4.10	11.9	IBM 370/158	D	G1
Purdue	[illegible]	5.69	16.6	CDC 6500	S	FUN
U. C. San Diego	[illegible]	13.1	38.2	Burroughs 6700	S	H
Yale	[illegible]	17.1*	49.9	DEC KA-10	S	F40

* TIME(100) = (100/75)**3 SGEFA(75) + (100/75)**2 SGESL(75)

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 a 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.

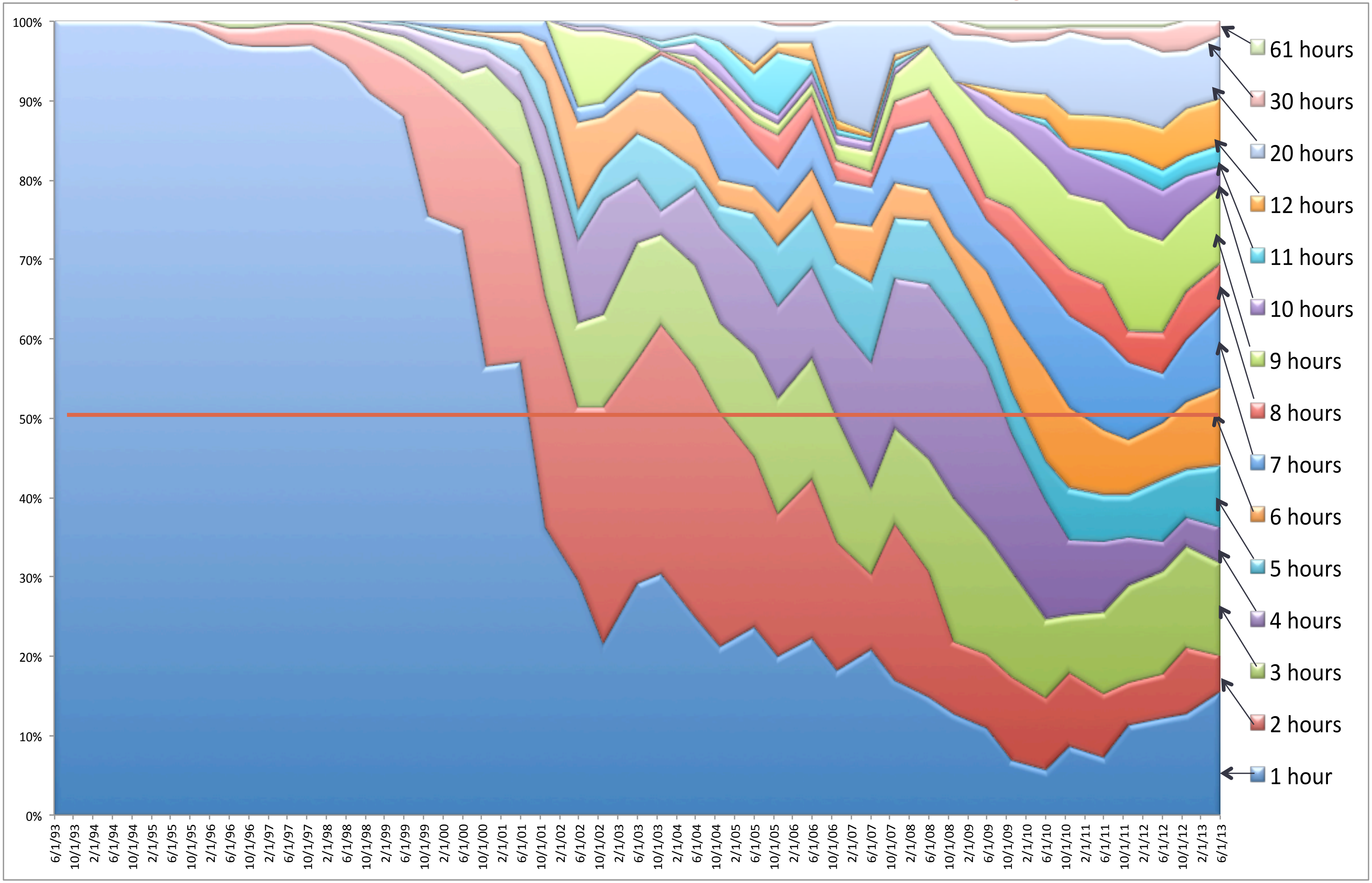
HPL - Bad Things

- 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 x 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 x 10^6
  - .11 Pflop/s #160 today

Run Times for HPL on Top500 Systems



#1 System on the Top500 Over the Past 20 Years (16 machines in that club)

9 🇺🇸 6 🇯🇵 2 🇨🇳

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

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

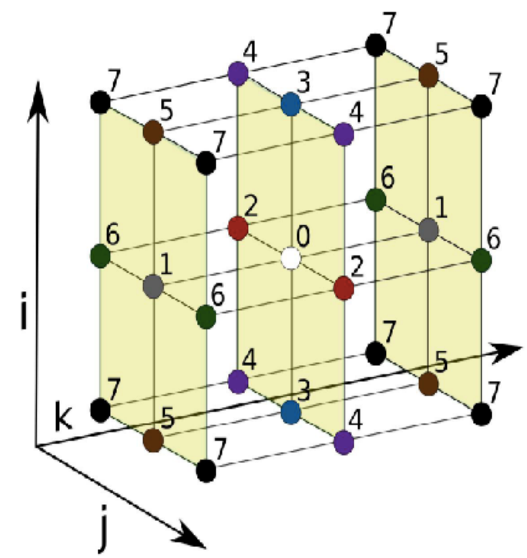
Proposal: HPCG

- 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: $(n_x * np_x) \times (n_y * np_y) \times (n_z * np_z)$
- Sparse matrix:
  - 27 nonzeros/row interior.
  - 7 – 18 on boundary.
  - Symmetric positive definite.



27-point stencil operator

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.

CG ALGORITHM

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

Execution: 7 Phases

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.

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.

Validation Testing

- 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. $x^T A y = y^T A x$
  - 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 **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 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 = tril(A); UA = triu(A); DA = diag(diag(A));
```

Solve:

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

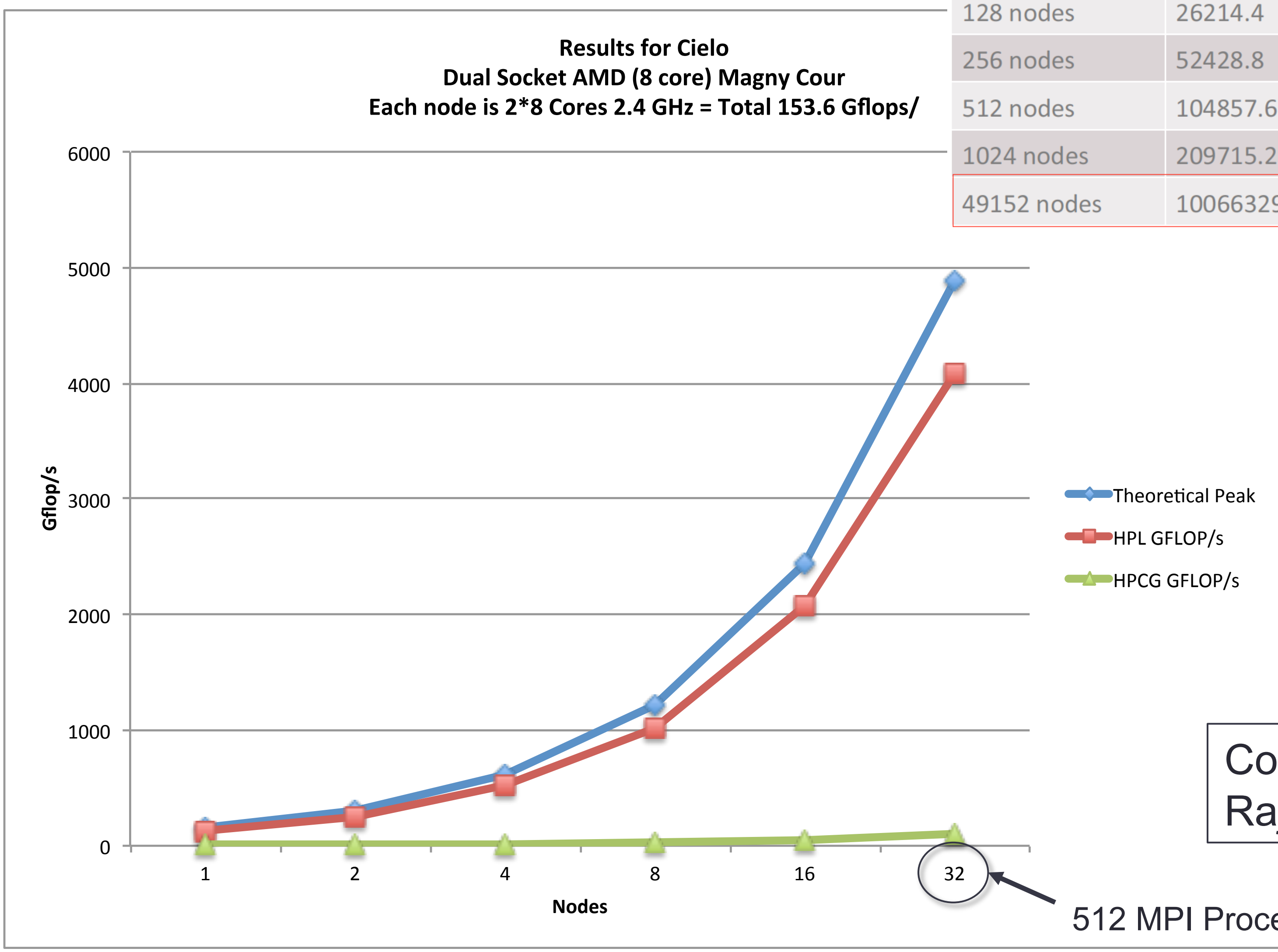
- Detects and measures variances from bitwise identical computations.

COMPUTATIONAL RESULTS

GFLOPS/s "Shock"

Mira Partition Size	Peak Gflops	Sustained Gflops	% of peak
64 nodes	13107.2	73.4	0.56%
128 nodes	26214.4	147.43	0.56%
256 nodes	52428.8	293.8	0.56%
512 nodes	104857.6	587.97	0.56%
1024 nodes	209715.2	1176.69	0.56%
49152 nodes	10066329.6	55177.6	0.55%

Courtesy Kalyan Kumaran, Argonne



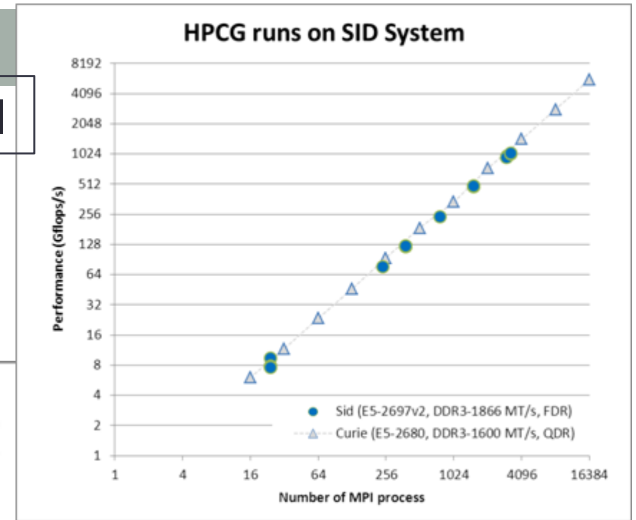
512 MPI Processes

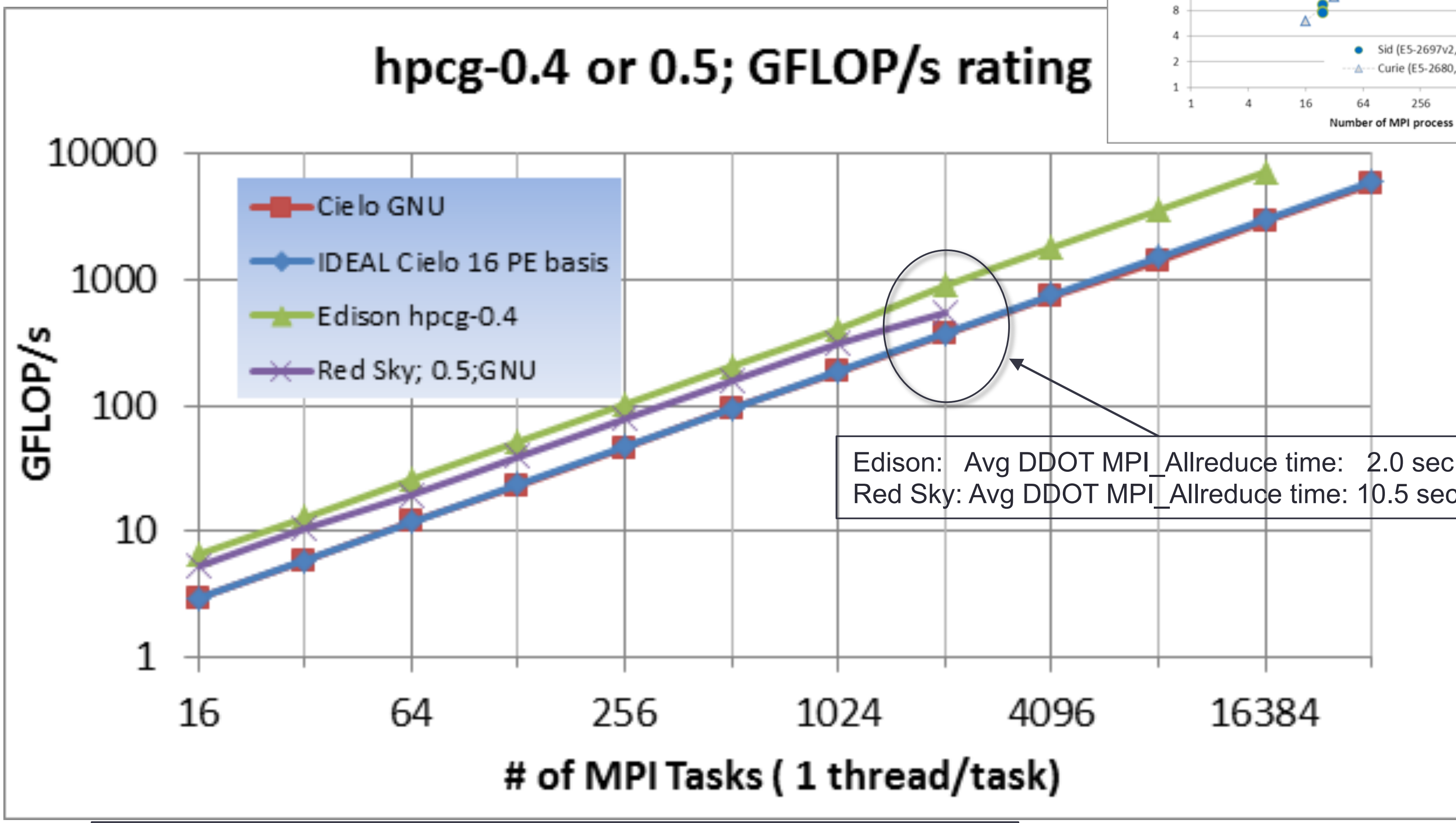
Courtesy Mahesh Rajan, Sandia

http://tiny.cc/hpcg

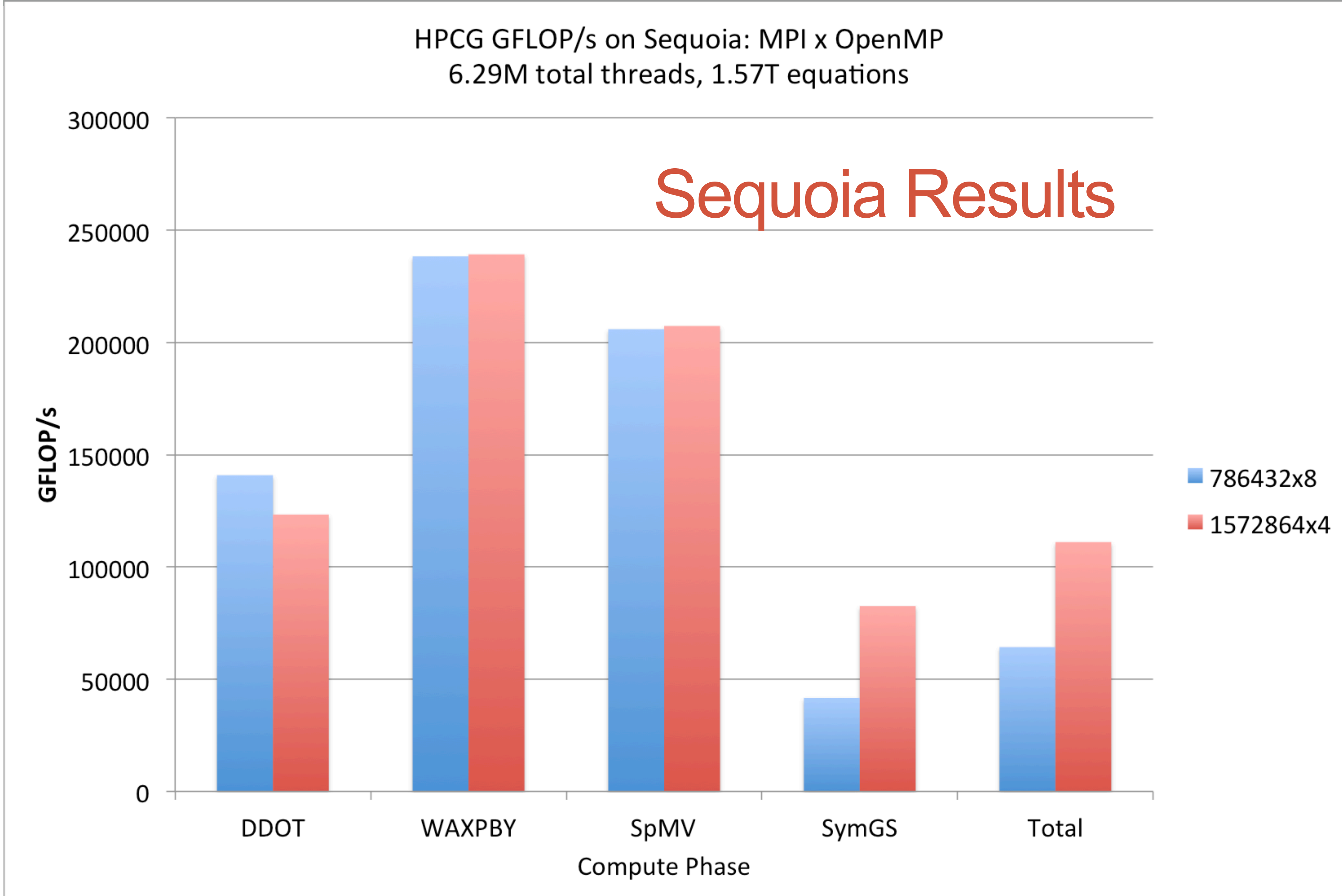
Results courtesy of Ludovic Saugé, Bull

Cielo, Red Sky, Edison, SID



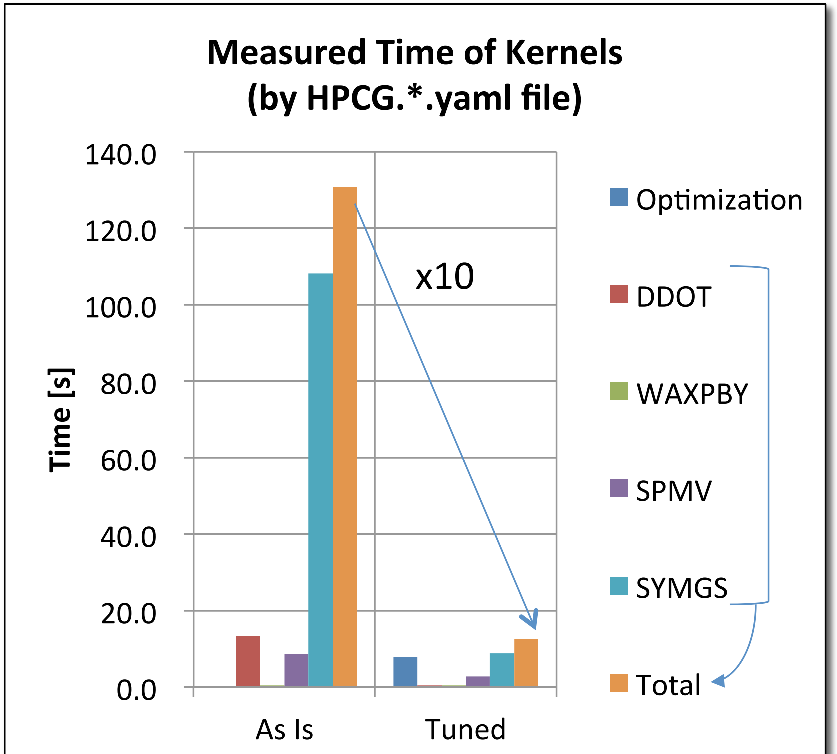


Results courtesy of M. Rajan, D. Doerfler, Sandia



Results courtesy of Ian Karlin, Scott Futral, LLNL

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

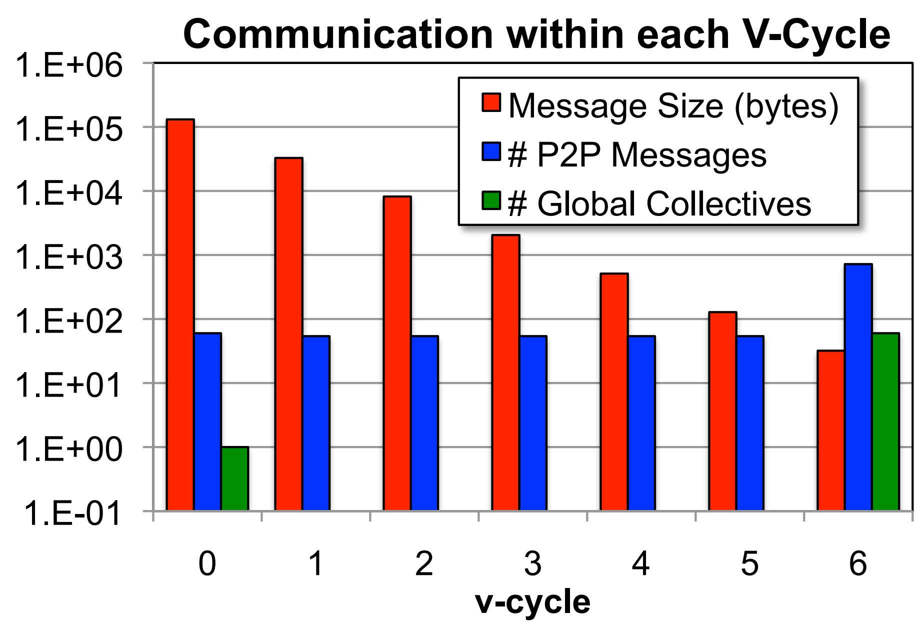
- Total x10 speed up now
  - Continuous memory for matrix
  - Multi-coloring for SYMGS multi-threading
- Under Studying
  - Node re-ordering for SPMV
  - Advanced matrix storage way
  - And so on

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

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

SANDIA REPORT
SAND2013- 8752
Unlimited Release
Printed October 2013

HPCG Technical Specification

Michael A. Heroux, Sandia National Laboratories[1]
Jack Dongarra and Piotr Luszczek, University of Tennessee

Prepared by

SANDIA REPORT
SAND2013-4744
Unlimited Release
Printed June 2013

Toward a New Metric for Ranking High Performance Computing Systems

Jack Dongarra, University of Tennessee
Michael A. Heroux, Sandia National Laboratories[1]

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

[1] Corresponding Author, maherou@sandia.gov