

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

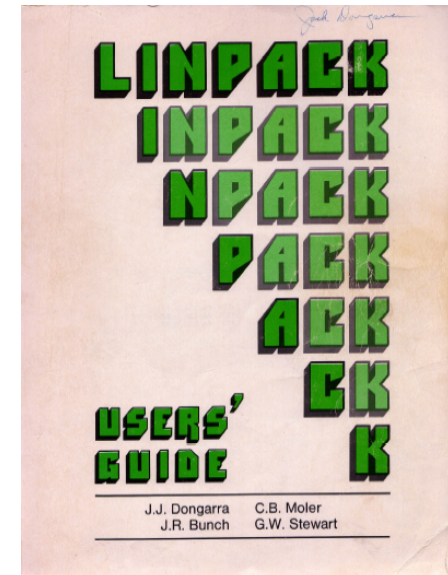
Jack Dongarra
University of Tennessee/ORNL

Michael Heroux
Sandia National Labs

See: <http://bit.ly/hpcg-benchmark>

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

Began in late 70's

time when floating point operations were expensive compared to other operations and data movement

$\frac{2}{3} N^3$ 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	3.54 .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	1.34 .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	Intel AS/5 mod3	D	H
U. Ill. Chicago	.144 4.10	11.9	IBM 370/158	D	G1
Purdue	.124 5.69	16.6	CDC 6500	S	FUN
U. C. San Diego	.062 13.1	38.2	Burroughs 6700	S	H
Yale	.040 17.1*	49.9	DEC KA-10	S	F40

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

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.

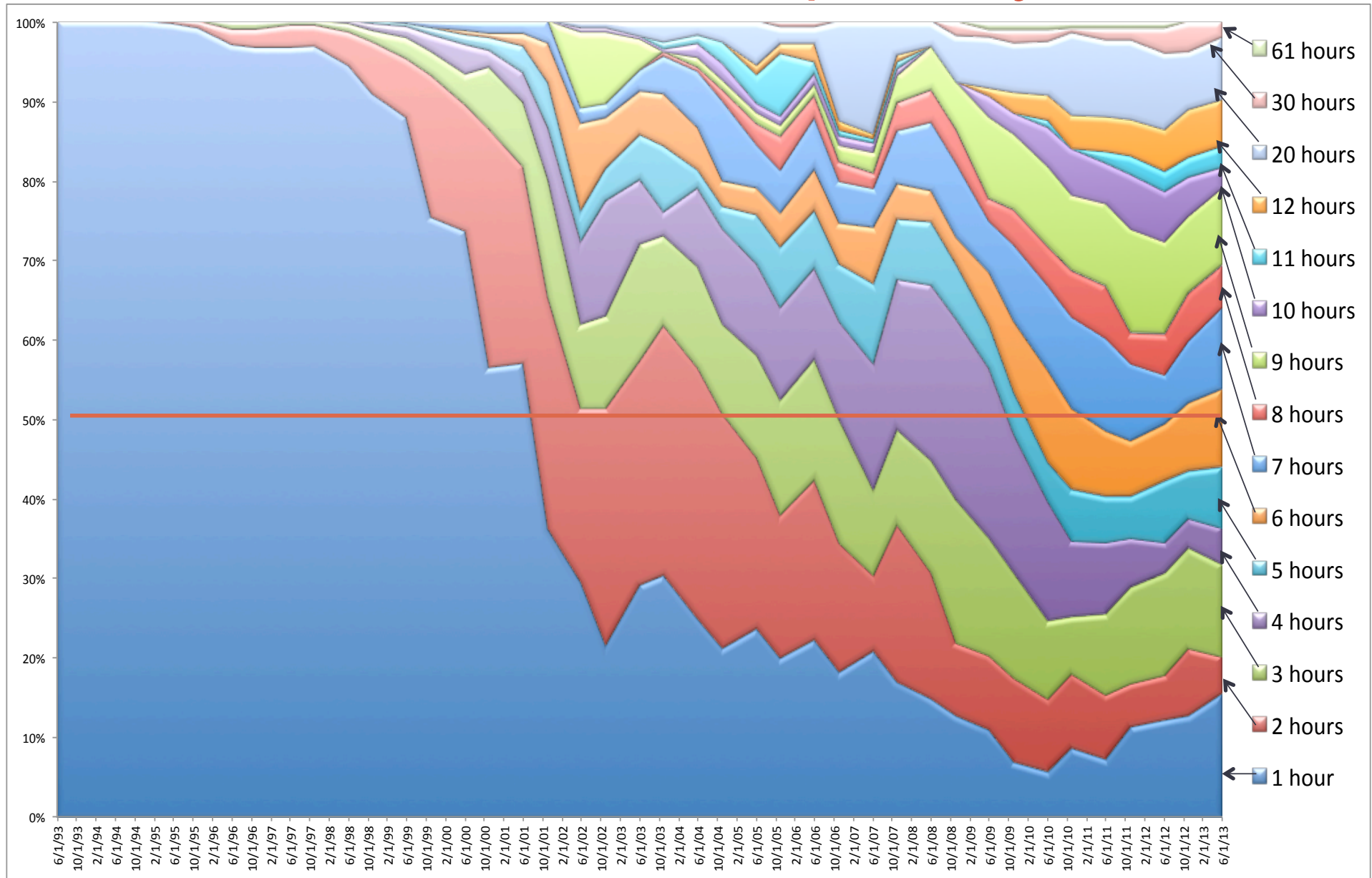
HPL - Bad Things

- 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



#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 (?)	NUDT Tianhe-2 Intel IvyBridge & Xeon Phi	33,862.	9,960,000	5.4	17.8

Assumptions

- Leadership class system:
 - Cost: \$200M
 - Lifetime: 4 years
 - Power consumption: 10MW
- Cost of one MW-year is \$1M
- Linpack measurement requires system for a week
 - To achieve a high fraction of peak requires a large problem size so a typical MP Linpack run takes a day
 - Multiple runs are made as initial tests are run with “small” problems
 - Successive tests use larger and larger problem sizes, some of these tests will “fail” – requiring re-runs

From: Jim Ang, SNL; What's the True Cost of LINPACK, Salishan 2013

Cost Estimates

- Electricity Cost
 - One week of usage $\approx [1/50 \text{ year}] \times 10\text{MW} = 0.20 \text{ MW-year} = \0.2M
- Amortized CapEx Cost
 - Opportunity cost associated with one week of usage
 - One week of dedicated system time is 1/200th of the life of the machine
 - That week represents 1/200 of the cost of the system or \$1M
 - The cost for one week of time on a new system is $> \$1\text{M}$
- Staff Cost
 - One week of how many peoples' loaded salaries?
 - How many are working around the clock?
 - Pizzas, Fried Chicken, Breakfast Burritos, Beer, Ice Cream, etc.

From: Jim Ang, SNL; What's the True Cost of LINPACK, Salishan 2013

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 500-142
- 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).

What about the NAS Parallel CG Benchmark?

- NAS CG is **flawed** from the perspective of modeling the design choices of real science and engineering codes.
- The **matrix truly random** and make the **placement of entries** random means that, for distributed memory machines, a 2-dimensional matrix decomposition is most effective, which is fundamentally different that the 1D processor decomposition that spatial locality in PDEs needs.
- Random also meant that the natural **spatial and temporal locality properties of real sparse matrices were not present**, so caches were much less useful in the benchmark than in real life.
- Finally, NAS CG has **no preconditioner**, so it is essentially a **fast sparse MV benchmark for an atypical sparse matrix.**

Problem Setup

- Synthetic symmetric positive definite problem
 - Matrix, rhs, and initial guess
 - Perhaps with several sparsity patterns using compressed row storage
 - User can change the matrix format and cost will be reported.
 - Matrix pattern may be regular but user cannot use this information in solution

Preconditioner Setup

- Symmetric Gauss-Seidel preconditioner
 - (Non-additive Schwarz)
- 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;
```

Iteration

- We will perform some number of iterations, repeated k times, using the same initial guess each time, where k is sufficiently large to test system uptime, at least **X** hours.
 - By doing this we can compare the numerical results for “correctness/reproducibility” at the end of each iteration phase.
- If the result is not bit-wise identical across successive iteration phases, we can report the deviation.
- Cache will be flushed between each of the k times the iterations are performed to report fair timing data for averaging.

Post-processing and reporting

- Collect numbers and provide an alternate to listing Top500
- V&V numbers are reported
- Timing and execution rates are reported
- Also reported will be the number of nodes, total storage, processors, accelerators, precision used, compiler version, optimization level, compiler directives used, flop count, power used, cache effects, loads and stores, etc.

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

- Provides coverage for major communication and computational patterns.
 - Represents a minimal collection of the major patterns.
- Rewards investment in high-performance collective ops.
- Rewards investment in local memory system performance.
- Detects and measures variances from bitwise identical computations.

Next Steps

- Validate against real apps on real machines.
 - Validate ranking and driver potential.
 - Modify code as needed.
 - Repeat as necessary.
- Introduce to broader community.
 - Buy-in.
 - Permutation of Top500 list.
- Notes:
 - Simple is best.
 - First version need not be last version (HPL evolved).

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.

See:

Toward a New Metric for Ranking High Performance Computing Systems

- Michael Heroux and Jack Dongarra

<http://bit.ly/hpcg-benchmark>