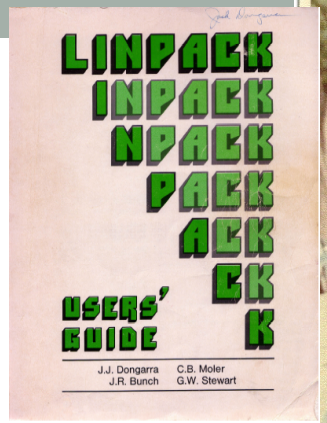
ACCIDENTAL BENCHMARKER

Jack Dongarra & Piotr Luszczek
University of Tennessee/ORNL

Michael Heroux
Sandia National Labs

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

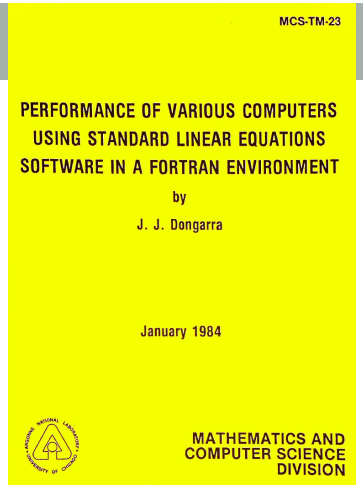
UNIT = 10**6 TIME/(1/3 100**3 + 100**2)

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

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

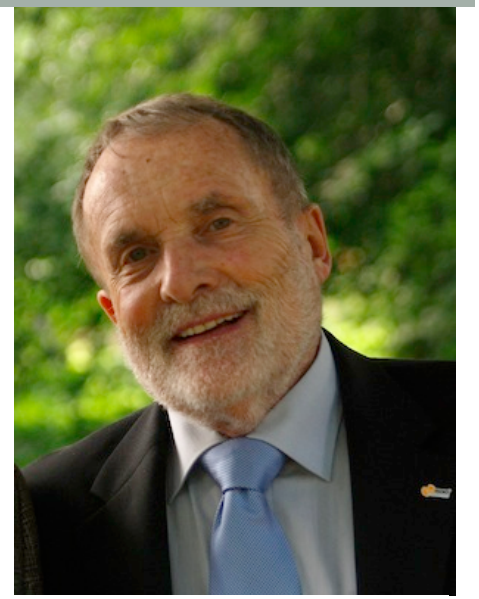
- LINPACK code is based on "right-looking" algorithm:
  - $O(n^3)$ Flop/s and $O(n^3)$ data movement

LINPACK to HPL to TOP500 Changes over time



- LINPACK Benchmark report, ANL TM-23, 1984
  - **Performance of Various Computers Using Standard Equations Software**, listed about 70 systems.
- Over time the LINPACK Benchmark when 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_{max}, N_{max}, R_{peak}
- In 2000 we put together an optimized implementation of the benchmark, called High Performance LINPACK or HPL.
  - Sets the problem up and checks the results
  - 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.

TOP500® SUPERCOMPUTER SITES

Rank	Site	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	Los Alamos National Laboratory United States	CM-5/1024 Thinking Machines Corporation	1,024	59.7	131.0	
2	Minnesota Supercomputer Center United States	CM-5/544 Thinking Machines Corporation	544	30.4	69.6	
3	National Security Agency United States	CM-5/512 Thinking Machines Corporation	512	30.4	65.5	
4	NCSA United States	CM-5/512 Thinking Machines Corporation	512	30.4	65.5	
5	NEC Japan	SX-3/44R NEC	4	23.2	25.6	
6	Atmospheric Environment Service (AES)	SX-3/44	4	20.0	22.0	

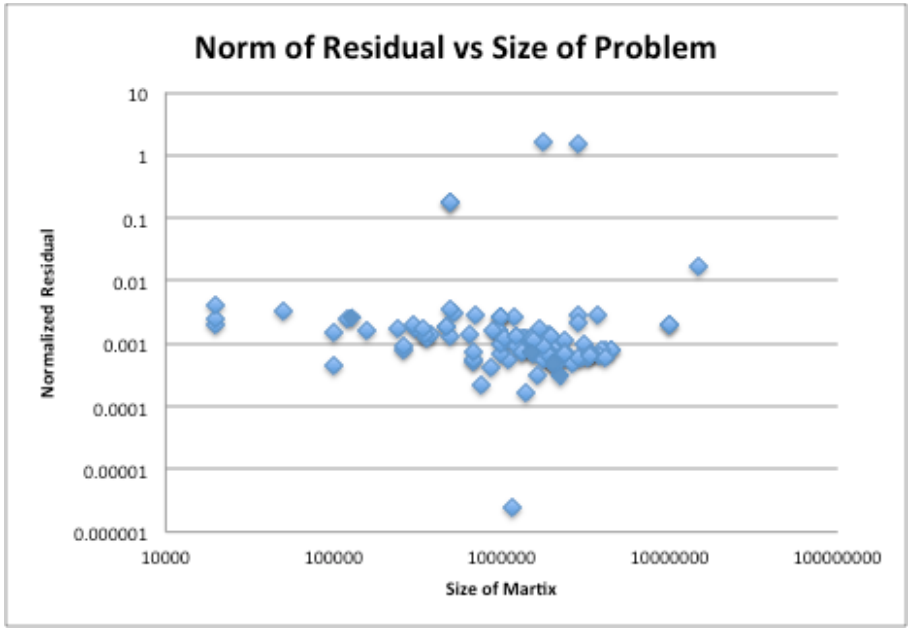
High Performance LINPACK (HPL)

- 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.

LINPACK Benchmark – Still Learning Things

- We use a backwards error residual to check the "correctness" of the solution.

$$\frac{\|b - Ax\|_\infty}{\varepsilon N \left(\|b\|_\infty + \|A\|_\infty \|x\|_\infty\right)}$$

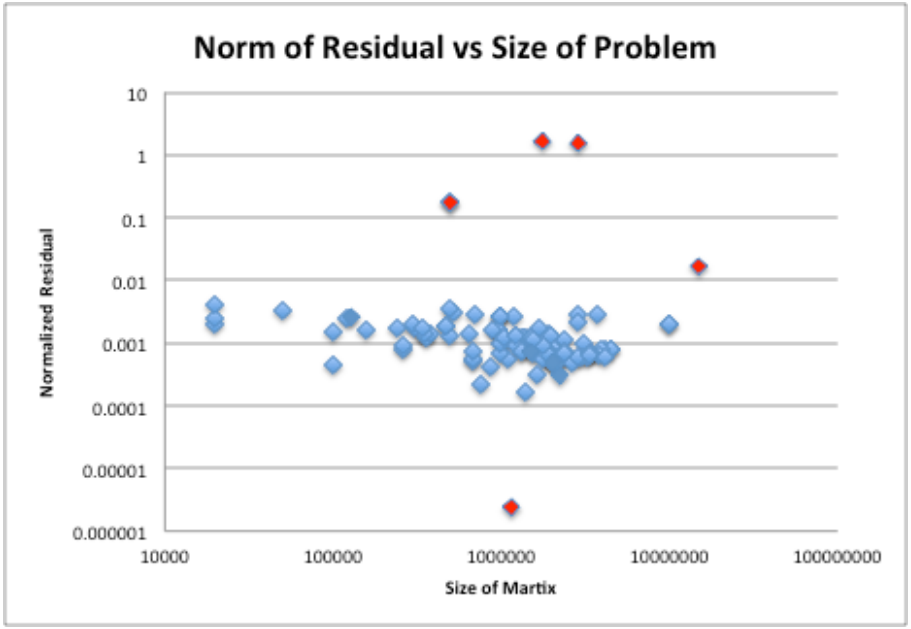


- This is the classical Wilkinson error bound.
  - If the residual is small O(1) then the software is doing the best it can independent of the conditioning of the matrix.
- We say O(1) is OK, the code allows the residual to be less than O(10).
- For large problems we noticed the residual was getting smaller.

LINPACK Benchmark – Still Learning Things

- We use a backwards error residual to check the "correctness" of the solution.

$$\frac{\|b - Ax\|_\infty}{\varepsilon N \left(\|b\|_\infty + \|A\|_\infty \|x\|_\infty\right)}$$

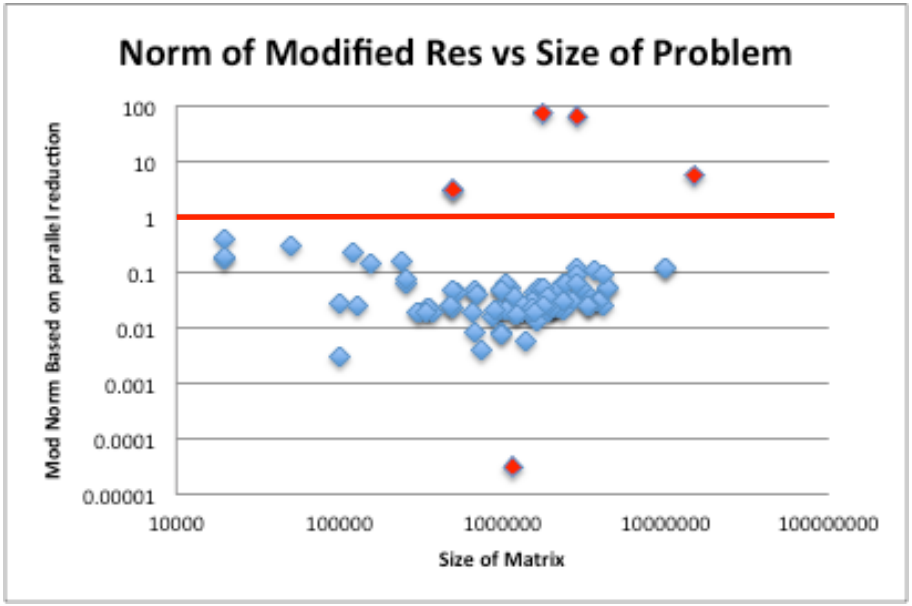


- This is the classical Wilkinson error bound.
  - If the residual is small O(1) then the software is doing the best it can independent of the conditioning of the matrix.
- We say O(1) is OK, the code allows the residual to be less than O(10).
- For large problems we noticed the residual was getting smaller $O(10^{-3})$.
  - Allowing 4 decimal digits of potential error

LINPACK Benchmark – Still Learning Things

- The current criteria might be about O(10^3) too lax which allows error for the last 10-12 bits of the mantissa to go undetected.
- We believe this has to do with the rounding errors for collective ops when done in parallel, i.e. MatVec and norms
- A better formulation of the residual might be:

$$\frac{\| b - Ax \|_\infty}{\varepsilon (N/Q)\log(Q)(\| b \|_\infty + \| A \|_\infty \| x \|_\infty)}$$



Where $(N/Q)\log(Q)$

comes from the way the sums are done in Q chucks and then reduced in a log fashion.

Concerns

- 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.
- 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 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!!

- 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

#1 System on the TOP500 Over the Past 20 Years (16 machines in that club)

9  6  2 

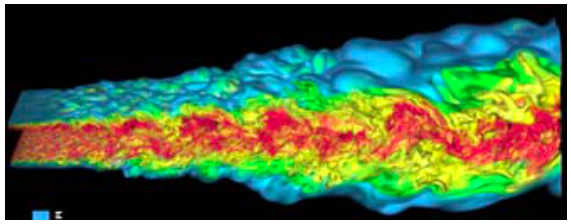
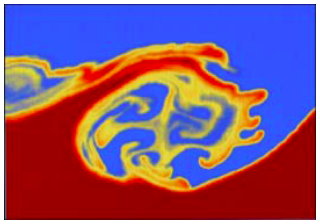
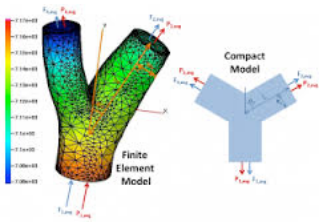
TOP500 List	Computer	r_max (Tflop/s)	n_max	Hours	MW
6/93 (1)	TMC CM-5/1024	.060	52224	0.4	
11/93 (1)	Fujitsu Numerical Wind Tunnel	.124	31920	0.1	1.
6/94 (1)	Intel XP/S140	.143	55700	0.2	
11/94 - 11/95 (3)	Fujitsu Numerical Wind Tunnel	.170	42000	0.1	1.
6/96 (1)	Hitachi SR2201/1024	.220	138,240	2.2	
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 (2)	NUDT Tianhe-2 Intel IvyBridge & Xeon Phi	33,862.	9,960,000	5.4	17.8

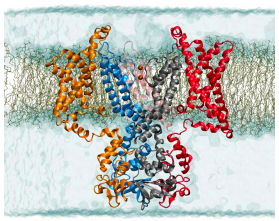
Many Other Benchmarks

- 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 (via spectral properties of CG).