MHPC
Master in High Performance Computing

P1.2 course: Introduction to benchmarking and tools

Stefano Cozzini

CNR/IOM and eXact-lab srl

SISSA
Scuola Internazionale Superiore di Studi Avanzati

ICTP
The Abdus Salam International Centre for Theoretical Physics

MHPC
Master in High Performance Computing

Agenda/ Aims

- Give you the feeling how much is important to know how your system/ application/computational experiment is performing..
- Name a few standard benchmarks that can help you in making/taking a decision
- Show you some tricks and tips how to make your own benchmarking procedure

benchmark: a definition

a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it

from wikipedia

Master in High Performance Computing

A few statements

- no single number can reflect overall performance
- the only benchmark that matters is the intended workload.
- The purpose of benchmarking is not to get the best results, but to get consistent repeatable accurate results that are also the best results.
- absolutely essential
- best to be done by people who know the application, the hardware, and the operating system
- needs several representative benchmarks
- careful with artificial benchmarks or marketing myths

An important note

- Measuring and reporting performance is the basis for scientific advancement in HPC.
- Not always scientific papers/reports guarantee reproducibility
- A lack of standards/rule is actually present in benchmarking arena.

challenges in benchmarking:

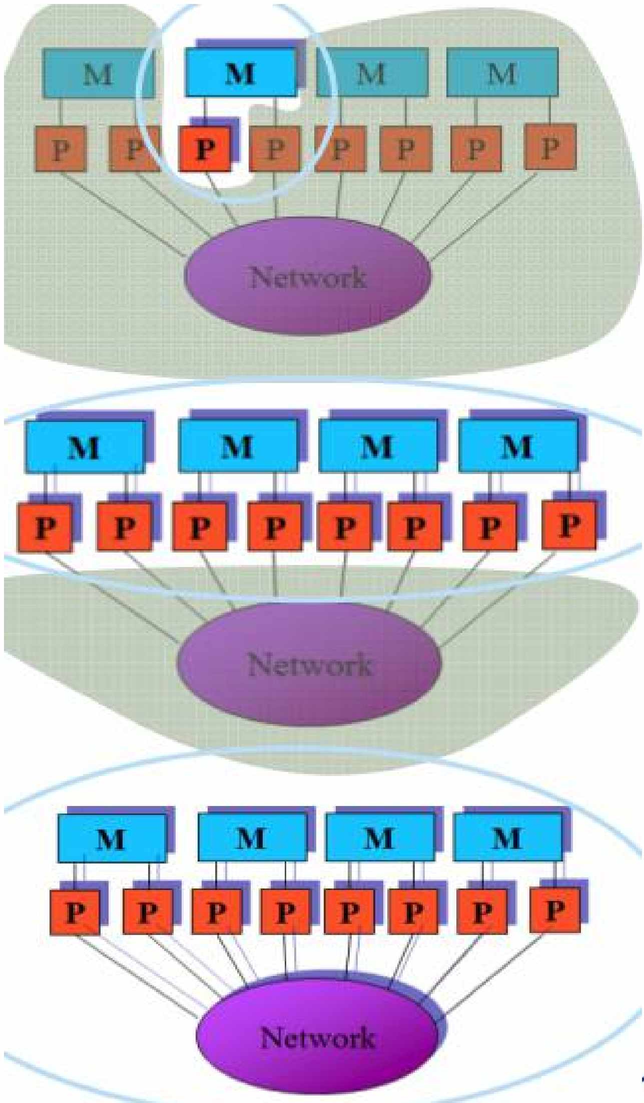
- Benchmarking is not easy and often involves several iterative rounds in order to arrive at predictable, useful conclusions.
- Interpretation of benchmarking data is also extraordinarily difficult:
  - Vendors tend to tune their products specifically for industry-standard benchmarks. Use extreme caution in interpreting their results.
  - Many benchmarks focus entirely on the speed of computational performance, neglecting other important features of a computer system.
  - Benchmarks seldom measure real world performance of mixed workloads — running multiple applications concurrently in a full, multi-department environment

What we need to benchmark on a modern system

Local: only a single processor (core) is performing computations.

Embarrassingly Parallel each processor (core) in the entire system is performing computations but they do no communicate with each other explicitly.

Global -all processors in the system are performing computations and they explicitly communicate with each other.



Type of code for benchmark

- Synthetic codes
  - Basic hardware and system performance tests
  - Meant to determine expected future performance and serve as surrogate for workload not represented by application codes
  - useful for performance modeling
- Application codes
  - Actual application codes as determined by requirements and usage
  - Meant to indicate current performance
  - Each application code should have more than one real test case

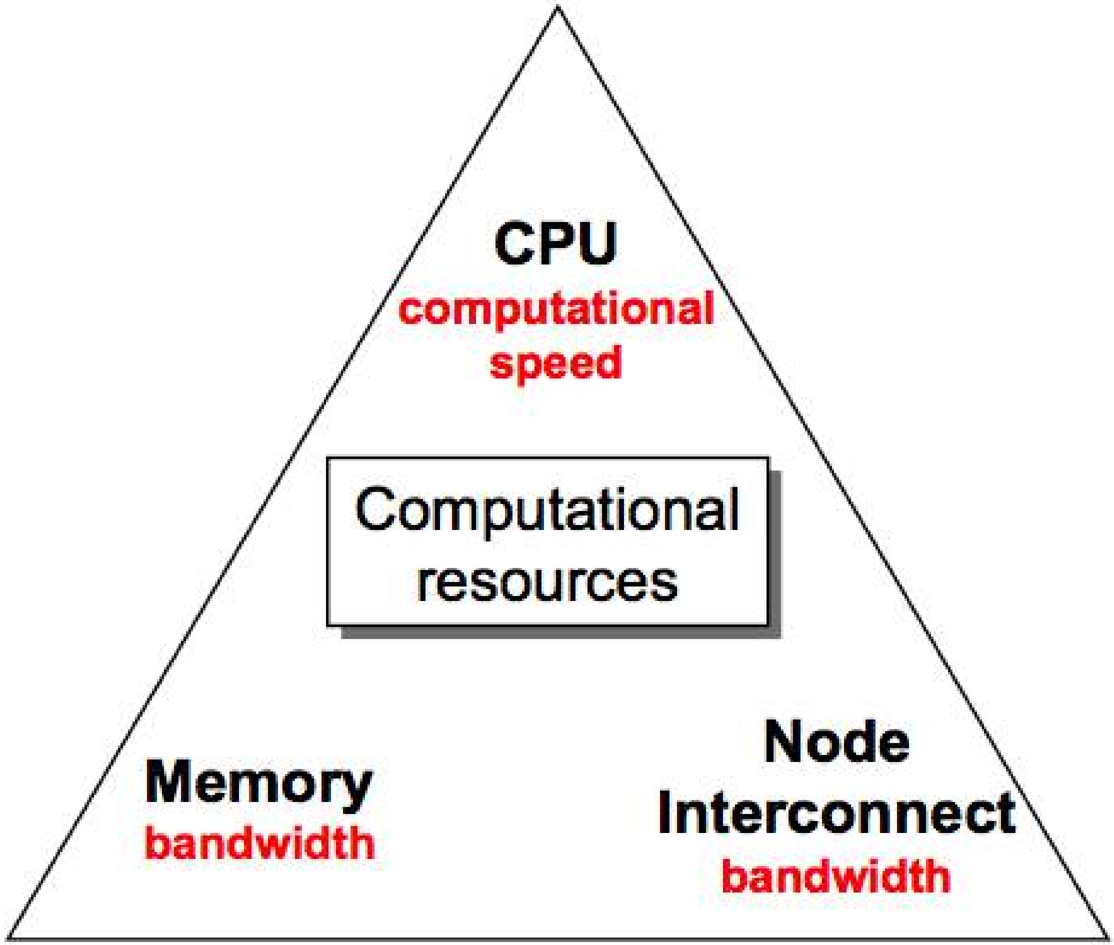
Some freely available benchmark (1)

- General benchmark:
  - HPL Linpack (for Top500)
  - HPC Challenge Benchmark:
    - a collection of basic benchmark beyond HPL
  - NAS benchmark suite
    - math kernel implemented both in MPI and openMP
    - (http://www.nas.nasa.gov/publications/npb.html)
  - HPCG
    - A recently introduced benchmark to “fix” the HPL one
  - Stream
    - Memory benchmark

Some freely available benchmarks (2)

- Network benchmark:
  - Netpipe /Netperf (http://www.netperf.org/netperf/)
    - tcp/ip protocol and more
  - IMB-4.0 (now IMB2017) (INTEL MPI benchmark)
    - MPI protocol ()
    - https://software.intel.com/en-us/articles/intel-mpi-benchmarks
  - OSU benchmarks: http://mvapich.cse.ohio-state.edu/benchmarks/
- I/O benchmarks:
  - Iozone/b_eff_io/IOR/ Mdbench/

resources to benchmark



Master in High Performance Computing

HPCC benchmark

7 tests:

1. HPL - the Linpack TPP benchmark which measures the floating point rate of execution for solving a linear system of equations.

2. DGEMM - measures the floating point rate of execution of double precision real matrix-matrix multiplication.

3. STREAM - a simple synthetic benchmark program that measures sustainable memory bandwidth (in GB/s) and the corresponding computation rate for simple vector kernel.

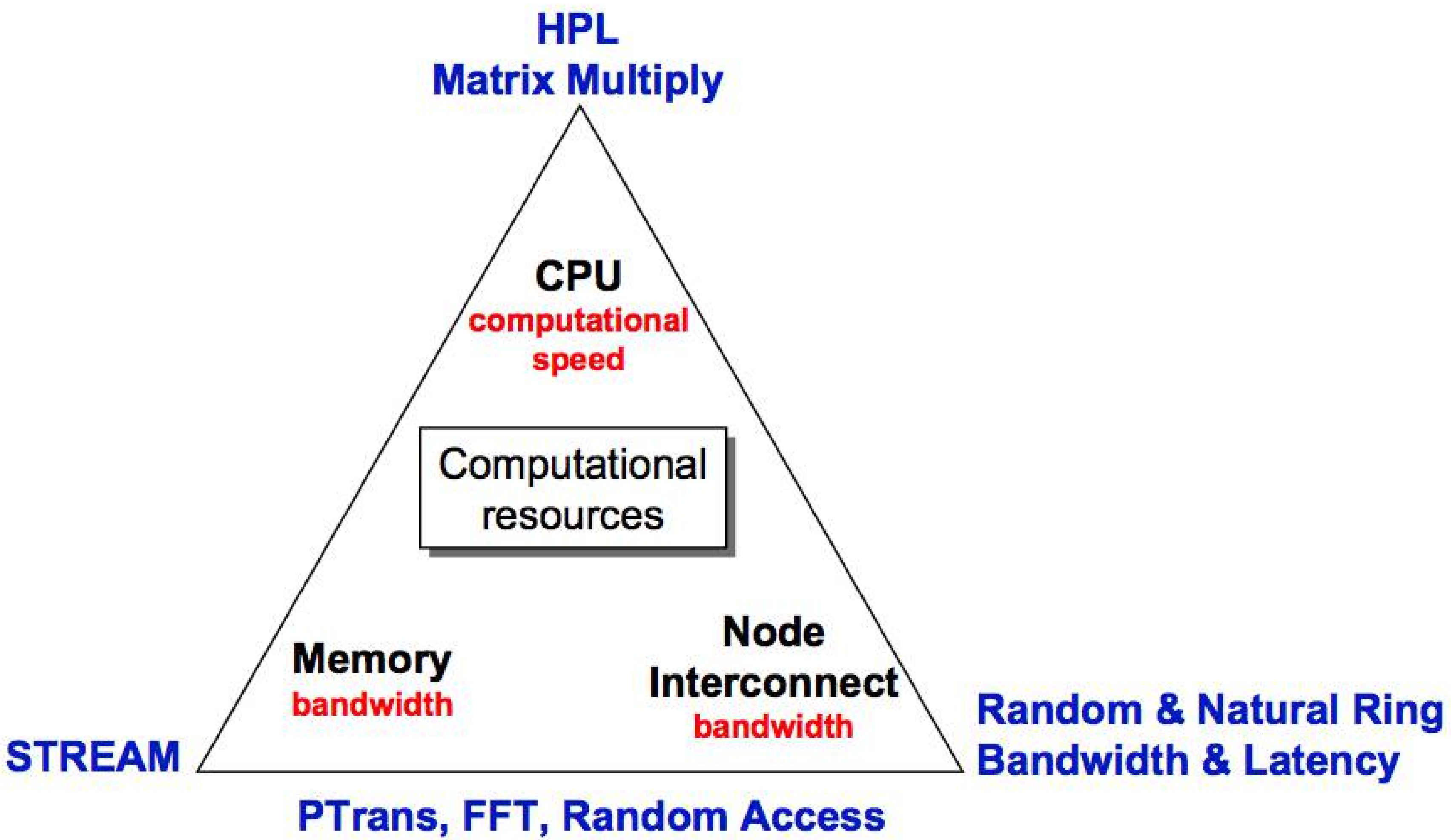
4. PTRANS (parallel matrix transpose) - exercises the communications where pairs of processors communicate with each other simultaneously. It is a useful test of the total communications capacity of the network.

5. RandomAccess - measures the rate of integer random updates of memory (GUPS).

6. FFT - measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform (DFT).

7. Communication bandwidth and latency - a set of tests to measure latency and bandwidth of a number of simultaneous communication patterns; based on b_eff (effective bandwidth benchmark).

HPCC components



http://icl.cs.utk.edu/hpcc/

HPL

- From http://icl.cs.utk.edu/hpl/index.html:

  The code solves a uniformly random system of linear equations and reports time and floating-point execution rate using a standard formula for operation count.

  - Number_of_floating_point_operations = $2/3n^3 + 2n^2$ (n=size of the system)

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR03R2L2       86000  1024     2     1             191.06              2.219e+03
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0043644 ...... PASSED
```

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 a think about a different metric

Master in High Performance Computing

HPCG benchmark

- 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)

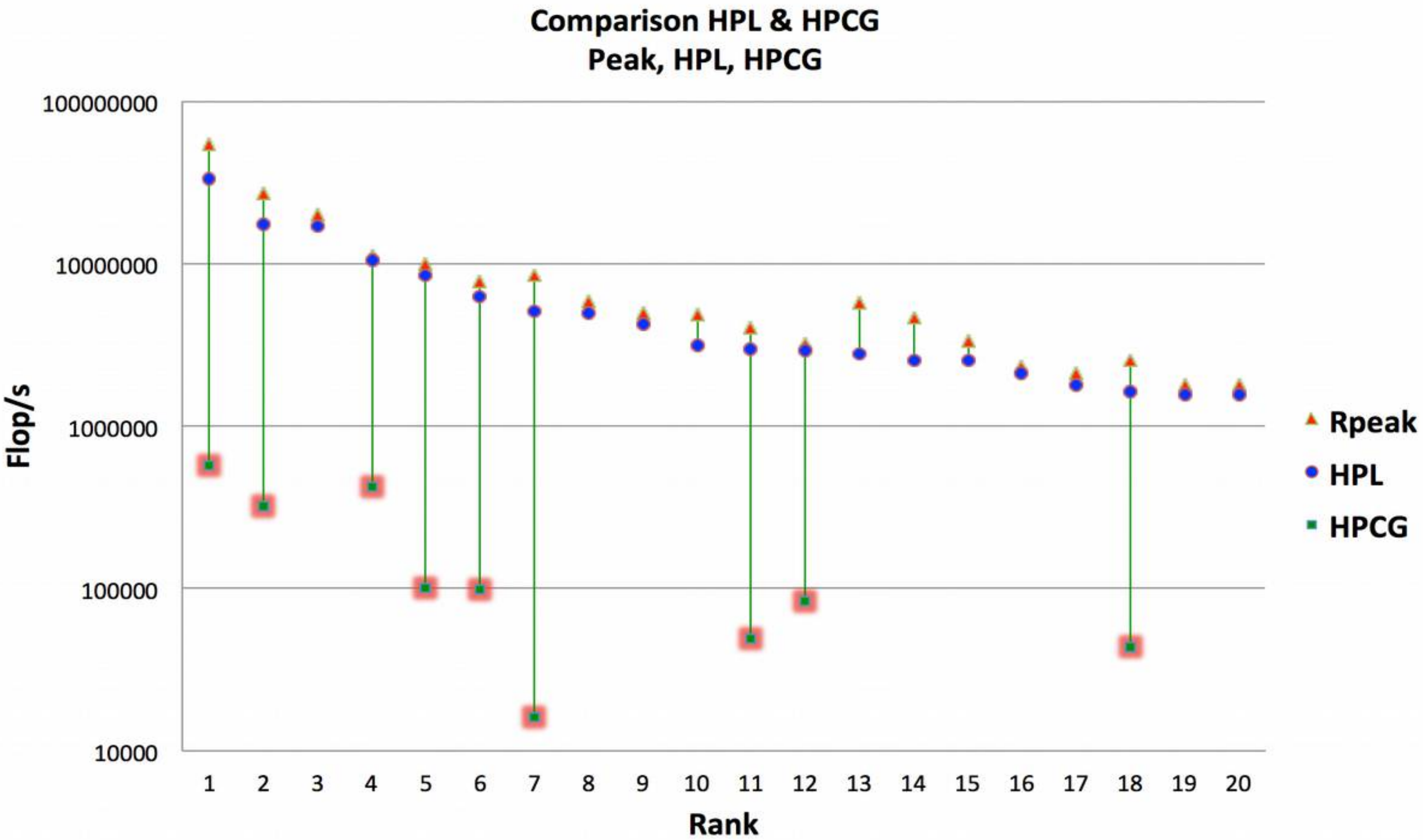
http://www.hpcg-benchmark.org/index.html

ma per seguir virtute e canoscenza SISSA ICTP

HPCG benchmark

- Balanced BW and compute:
- HPCG is memory BW bound in modern processors
  - 6 Byte/FLOP
  - HPCG can utilize at most x/6 of peak FLOP
- Scalable collectives: HPCG uses all-reduce
- Efficient parallelization of Gauss-Seidel: HPCG spends 2/3 of time in GS

Comparison HPL vs HPCG



Remember:

THERE IS NO BENCHMARK THAT SUBSTITUTES your own code on your dataset

Measurement should be done by you on your code !

Benchmarking your scientific package

- Generally (well done) common scientific package provide a set of standard input to help benchmarking them and evaluate the system/compiler/libraries etc..
- When you have to benchmark them please adopt the same rules (if they are stated)
- Otherwise set your own set of rules
  - Am I allowed to turn on unsafe math option ?
  - Am I allowed to change some part of the code / basic libraries/ compiler ?
  - Must the output match “exactly” or is some tolerance allowed?

Tips to benchmark

- use /usr/bin/time and take note of all times

  wall time/ user time /sys time

- repeat the same run at least a few time to estimate the fluctuations of the numbers (this should be generally within a few percent)
- be sure to be alone on the system you are using and with no major perturbation on your cluster
- execution runs should be at least in the order of tens of minutes
- always check the correctness of your scientific output

Performance Evaluation process

- Monitoring your System:
  - Use monitoring tools to better understand your machine's limits and usage
    - is the system limit well suited to run my application ?
  - Observe both overall system performance and single-program execution characteristics. Monitoring your own code
    - Is the system doing well ? Is my program running in a pathological situation ?
- Monitoring your own code:
  - Timing the code:
    - timing a whole program (time command :/usr/bin/time)
    - timing a portion (all portions) of the program
  - Profiling the program (already seen)

Tools to monitor your system

- atop - Advanced System & Process Monitor
- iotop - simple top-like I/O monitor
- iftop - display bandwidth usage on an interface by host
- dstat - versatile tool for generating system resource statistics
- vmstat – to check memory

Running HPL on C3HPC (precompiled version)

- Connect to the machine
  - `ssh mhpc0X@hpc.c3e.cosint.it`
- Submit interactive job:
  - `qsub -I -l nodes=$NODE:ppn=24 -l walltime=6:0:0 -N $GROUP`
- Identify the right executable in /lustre/mhpc/eas/hpl/bin
  - `xhpl.plasma-gnu`
  - `xhpl.openblas`
  - `xhpl.netlib`
  - `xhpl.mkl-gnu`
  - `xhpl.atlas`
  - `hpl.plasma-mkl`
- Execute it:
  - Load appropriate modules
  - Run it:
  - `mpirun -np XX /lustre/mhpc/eas/hpl/bin/xhpl.mkl-gnu`

A few notes:

- Standard input file should be present
- Beware of threads
  - How to control them ?
- Help is provided by courtesy of M.Baricevic
  - Check out wrap.sh and run.sh

What about N ?

- N should be large enough to take ~75% of RAM..
  - N = sqrt (0.75 * Number of Nodes * Minimum memory of any node / 8)
- You can compute it via:
  - http://www.advancedclustering.com/act-kb/tune-hpl-dat-file/

HPL benchmark input file HPL.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
50000 Ns
1             # of NBs
768         NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
4 1 2 1        Ps
4 2 2 4        Qs
16.0         threshold
1            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
2 8          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0 2          BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1 0          DEPTHs (>=0)
1            SWAP (0=bin-exch,1=long,2=mix)
192          swapping threshold
1            L1 in (0=transposed,1=no-transposed) form
1            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

Master in High Performance Computing

Parameters for HPL.dat input file

N	Problem size	**Pmap**	Process mapping
NB	Blocking factor	**threshold**	for matrix validity test
P	Rows in process grid	**Ndiv**	Panels in recursion
Q	Columns in process grid	**Nbmin**	Recursion stopping criteria
Depth	Lookahead depth	**Swap**	Swap algorithm
Bcasts	Panel broadcasting method	**L1, U**	to store triangle of panel
Pfacts	Panel factorization method	**Align**	Memory alignment
Rfacts	Recursive factorization method	**Equilibration**	

MHPC
Master in High Performance Computing

Tips to get performance..

- Figure out a good block size (NB) for the matrix multiply routine. The best method is to try a few out. If you happen to know the block size used by the matrix-matrix multiply routine, a small multiple of that block size will do fine. This particular topic is discussed in the FAQs section.
- The process mapping should not matter if the nodes of your platform are single processor computers. If these nodes are multi-processors, a row-major mapping is recommended.
- HPL likes "square" or slightly flat process grids. Unless you are using a very small process grid, stay away from the 1-by-Q and P-by-1 process grids.

What your are supposed to do:

- Test a few N and Nb to identify the most performing one..
- Tip:
  - Write a small script and submit it before lunch
- Compute the ratio between theoretical peak performance and sustained hpl performance
  - If the result is less than 75% leave the master
  - Otherwise go on with exercise as specified on the github account

Master in High Performance Computing

Some interesting documents/links

- http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/twelve-ways.pdf

  Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers

  David H. Bailey June 11, 1991

- http://www.hpcwire.com/2011/12/13/ten_ways_to_fool_the_masses_when_giving_performance_results_on_gpus/
- http://htor.inf.ethz.ch/publications/img/hoefler-scientific-benchmarking.pdf

  Scientific Benchmarking of Parallel Computing Systems)Twelve ways to tell the masses when reporting performance results)

- http://www.sandia.gov/~maherou/docs/HPCG-Benchmark.pdf