Advanced School in
High Performance
and GRID Computing

The art of benchmarking

Stefano Cozzini

CNR-INFM DEMOCRITOS, Trieste

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

three important notes:

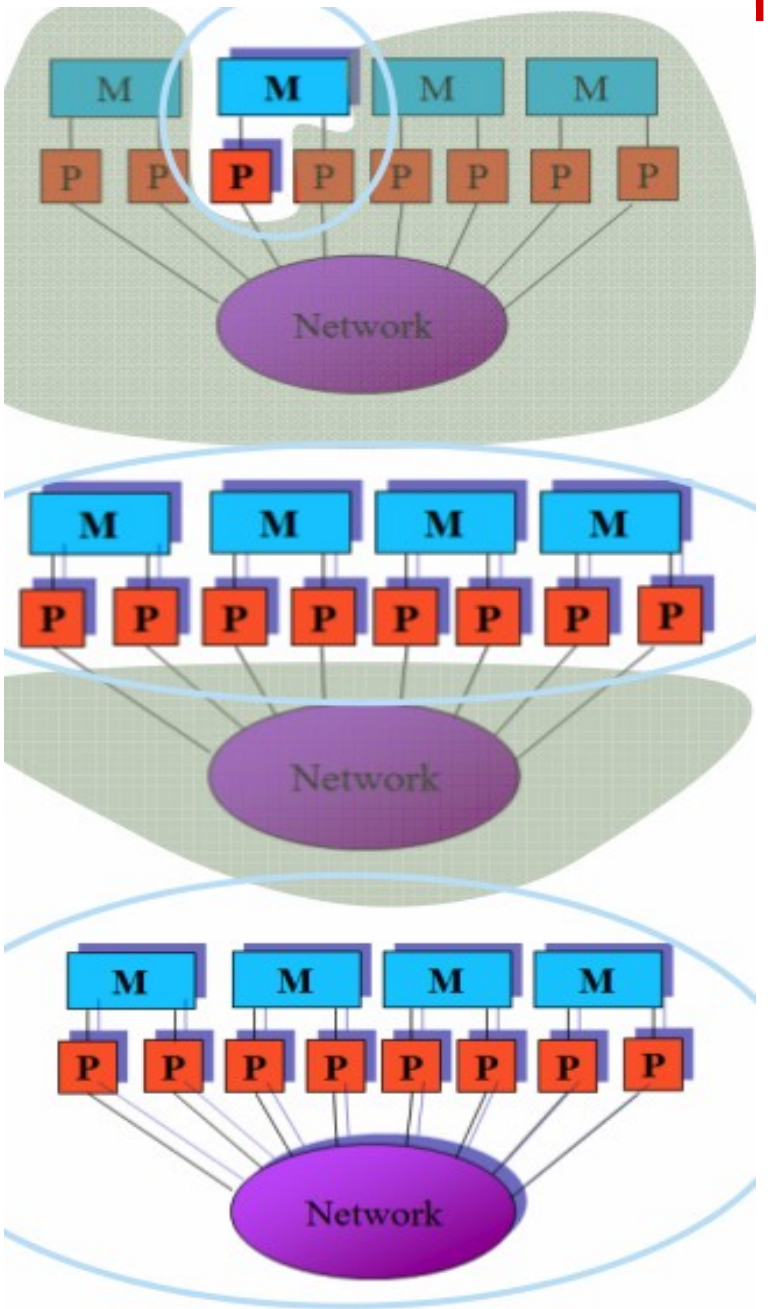
- no single number can reflect overall performance
- the only benchmark that matters is the intended workload.
- The purpose of benchmarking is not to get the best results, but to get consistent repeatable accurate results that are also the best results.

a few 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

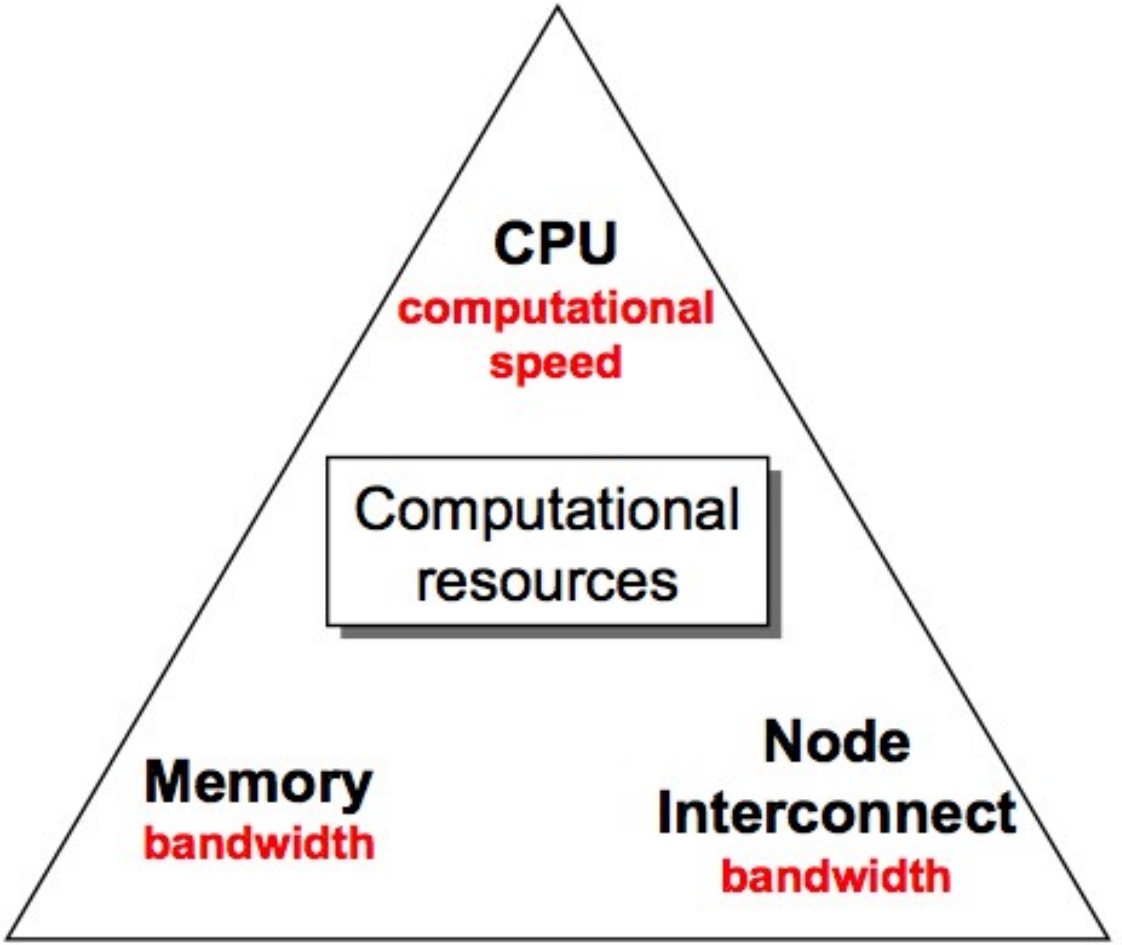
A very incomplete list of freely available benchmark:

- 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
- Network benchmark:
  - Netpipe /Netperf
    - tcp/ip protocol and more
  - IMB
    - MPI protocol
- I/O benchmarks: Iozone /bonnie etc..

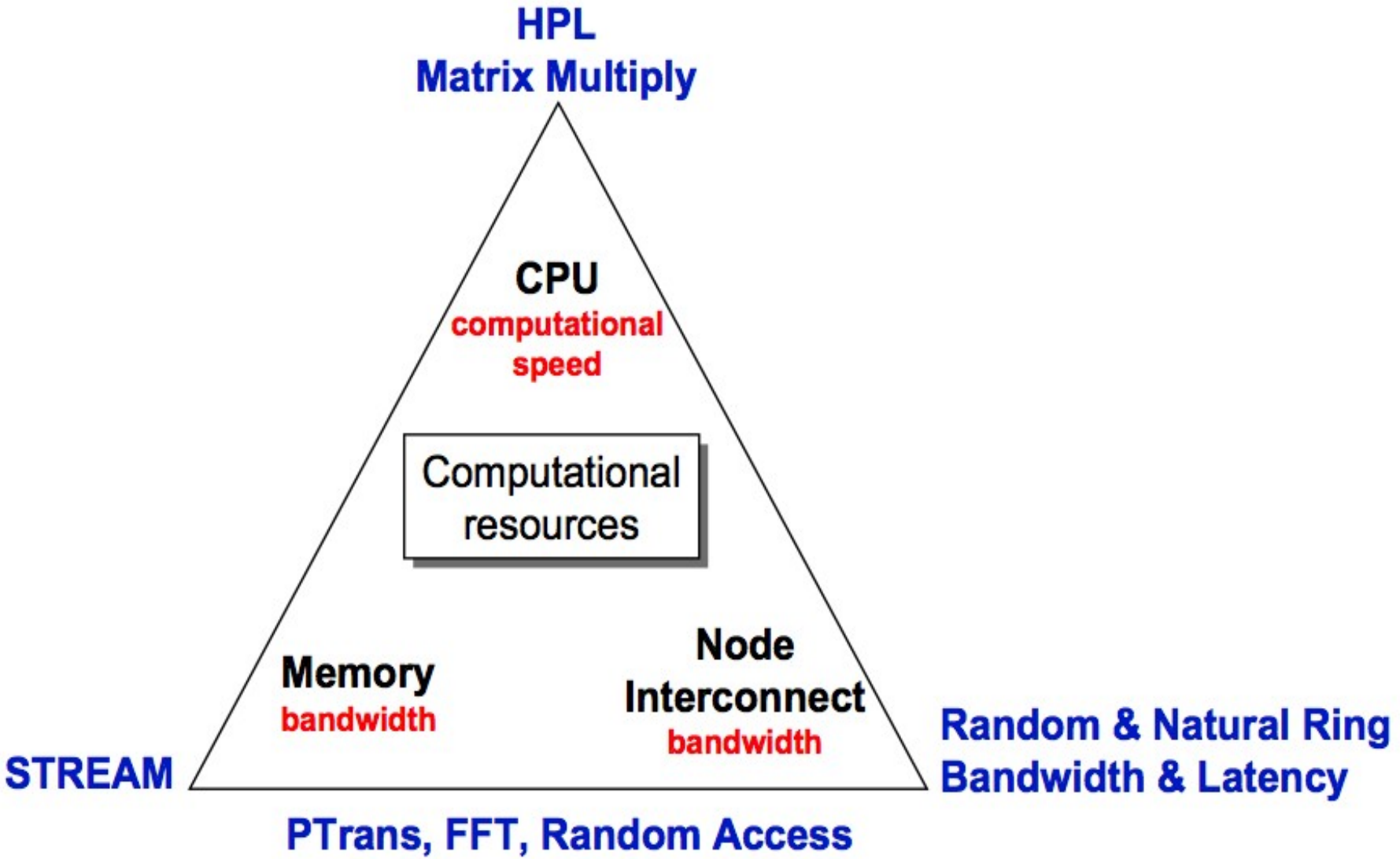
HPCC benchmark

- The HPC Challenge benchmark consists of basically 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).

Computational resources to benchmark



HPCC components



Remember:

- THERE IS NO BENCHMARK THAT SUBSTITUTES your own code on your dataset
- Measurement should be done by you on your code !

a few tips to benchmark your application.

- 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

eLBaS: eLab Benchmark Suite:

- Aim:
  - collect a set of scientific codes with several dataset representing actual workload of our user community
  - run them on all the machine available to us
  - create a database of the results and keep it updated
- Status:
  - work in progress:
    - identified codes and dataset
    - running benchmarks collecting the results
    - databas/web interface to it to be built
- Open effort:
  - people can check results
  - people interested can contribute sending us results

eLBaS : codes and datasets

- Codes are as much as possible heterogeneous:
  - Material/Nano Science (mainly) (Q/E,dlprotein/QMC)
  - Climate modeling
  - CFD
  - Nuclear Physics
  - Other to come
- Datasets
  - Small dataset: to test SMP multicore
  - large/medium data set to test medium size clusters
  - specific workload to test HPC/GRID infrastructure in oder to assses the global throughtput

some results for multicore machine

name	CPU type	cores	cache/chip	FSB
cl-dell	Intel Xeon X7460 2.66GHz	4 × 6	9+16 Mbytes	1066 MHz
neon	Intel Xeon X7350 2.93GHz	4 × 4	8 Mbytes	1066 MHz
zebra	Intel Xeon E5420 2.5GHz	2 × 4	12 Mbytes	1333 MHz
blade	AMD Opteron 280 2.4GHz	2 × 2	1 Mbyte	1000 MHz
myri	AMD Opteron 275 2.2GHz	2 × 2	1 Mbyte	1000 MHz
a40x	AMD Opteron 8218 2.6GHz	4 × 2	1 Mbyte	1000 MHz
clima100	AMD Opteron 885 2.6Ghz	8 × 2	1 Mbyte	1000 MHz

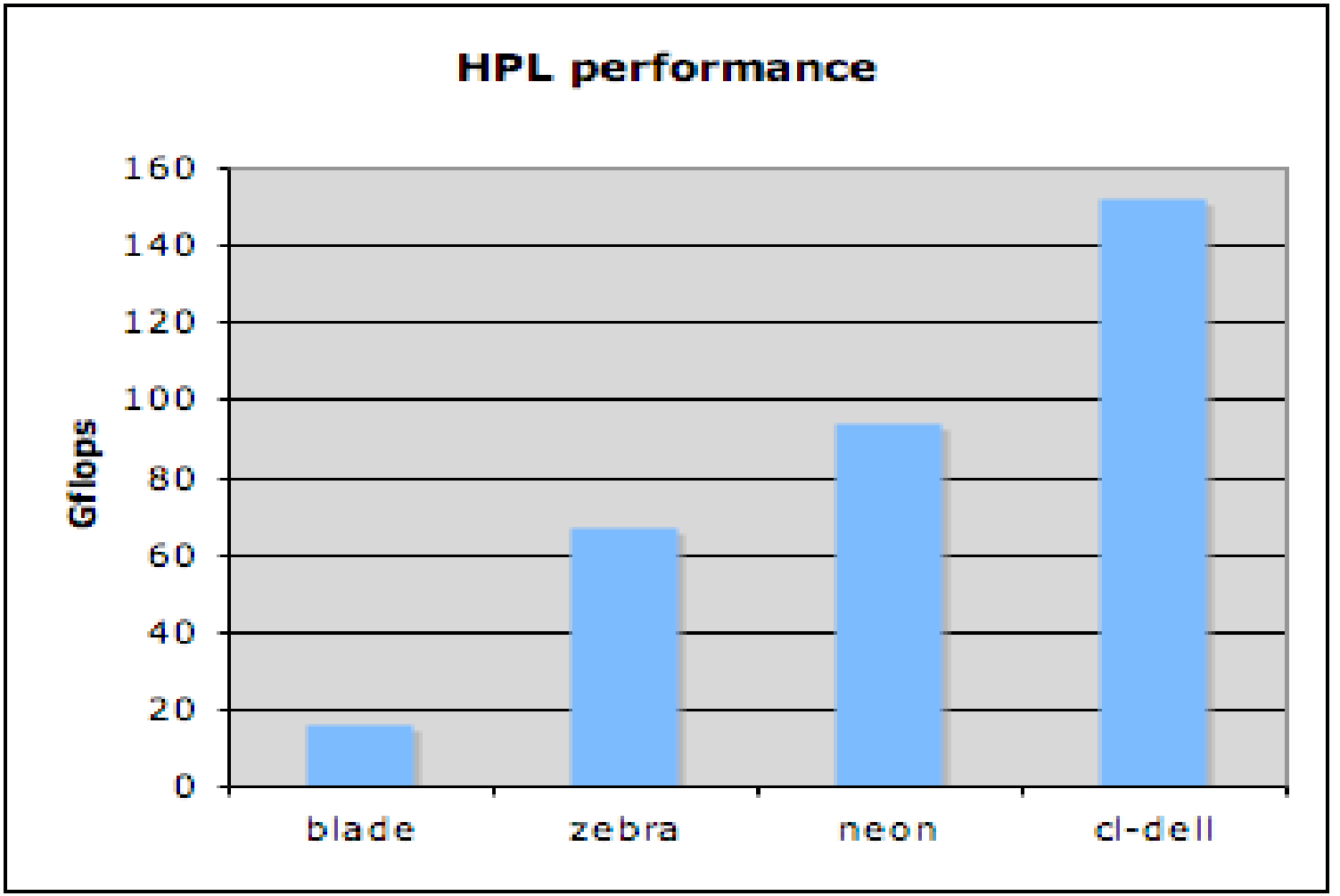
a few information more:

- software stack:
  - Intel Fortran/C/C++ Compiler 10.1.018
  - OpenMPI 1.2.8
  - MKL 10.0.5.025
  - FFTW 3.2 compiled with SSE2 support

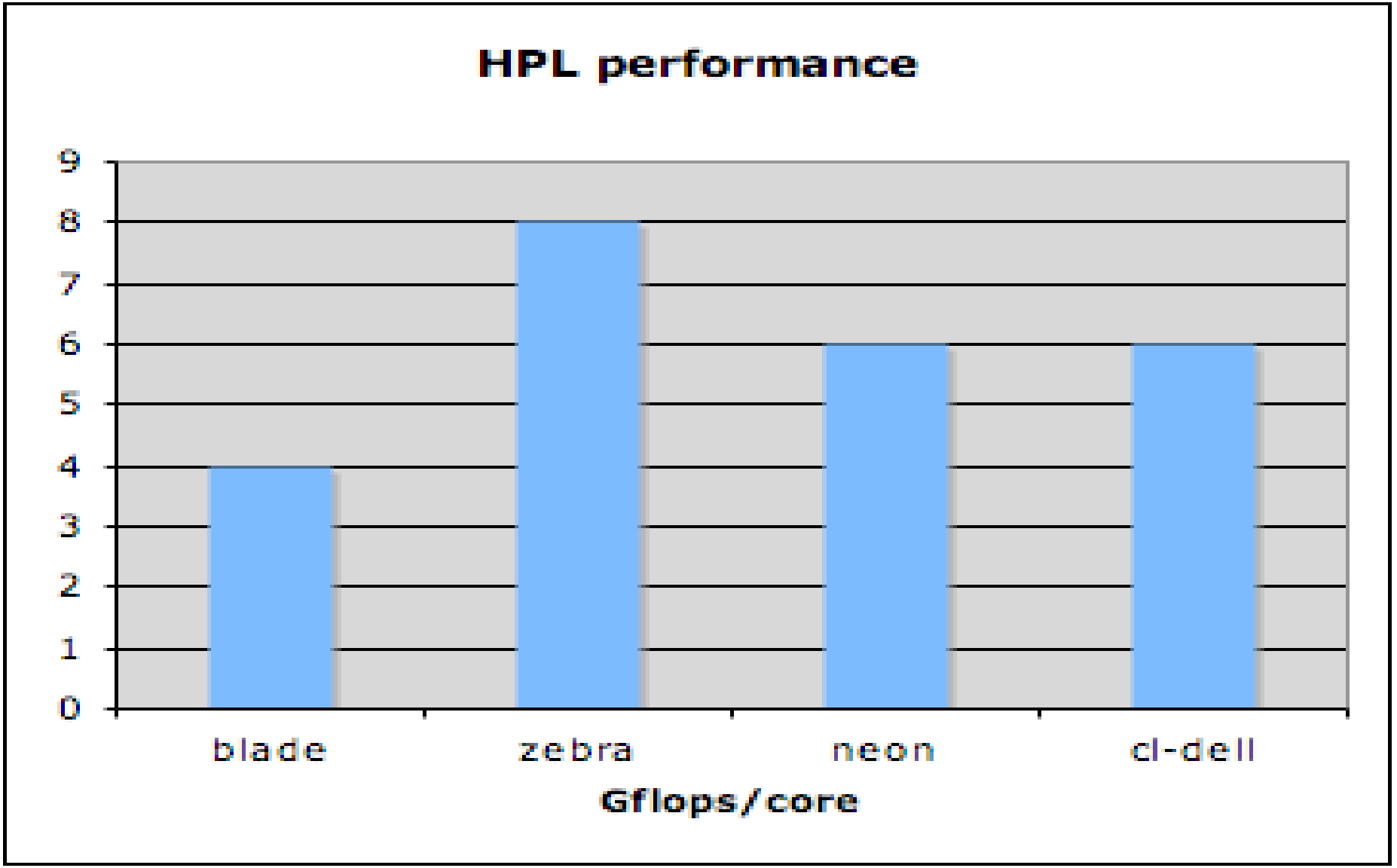
On AMD machines comparative tests were conducted to investigate the performance gain that the ACML library could offer on native architecture with respect to the MKL.

HPCC: hpl

- LU factorization: mainly BLAS level 3 dominated.



HPL: performance for core



HPL: ACML vs MKL

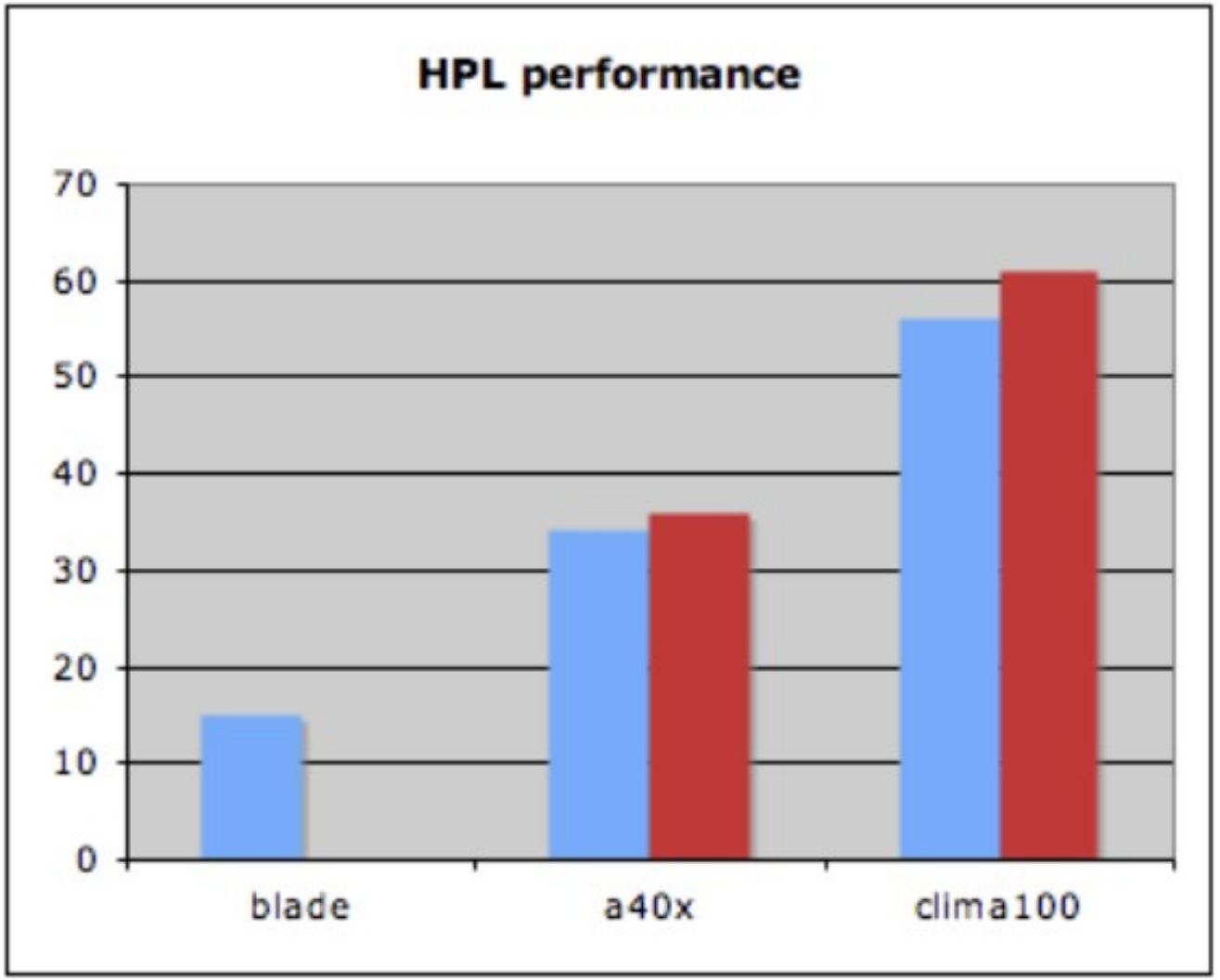


Figure 3: Linpack performance using MKL (blue bars) and ACML (red bars)

HPCC: stream

- Single: one single stream instance
- Star: all the CPUs running stream..

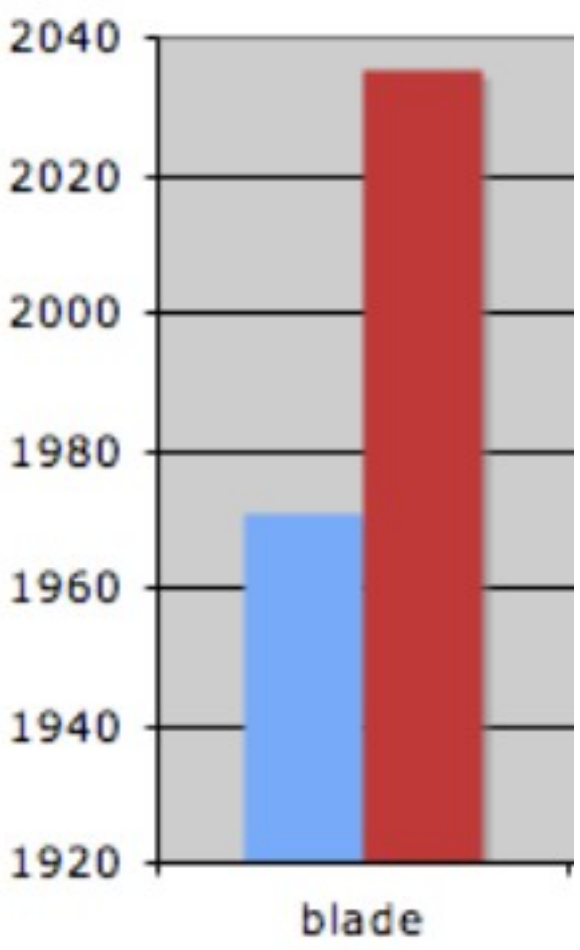
name	Single (Gb/s)	Star (Gb/s)	Used mem. (Gb)
cl-dell	2.265	0.399	1.35
neon	2.363	0.453	0.75
zebra	2.818	0.784	1.49
blade	2.490	1.455	1.46
a40x	2.667	2.070	2.92
clima100	2.204	0.899	1.41

Quantum/Espresso preliminary results

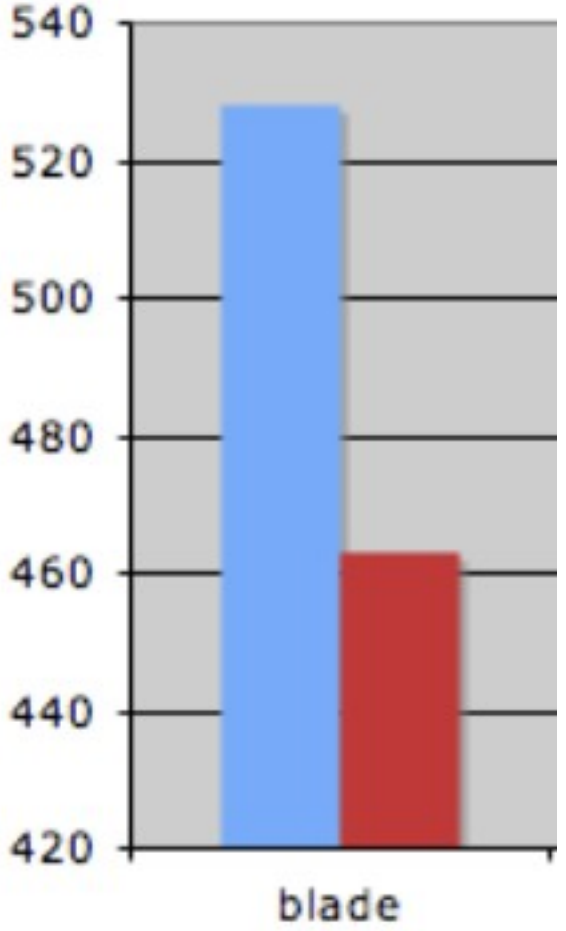
- Pwscf3:
  - small SCF calculation
- CP
  - CP MD test

Role of MKL/ACML on blade for Q/E...

- pwscf3



- cptest

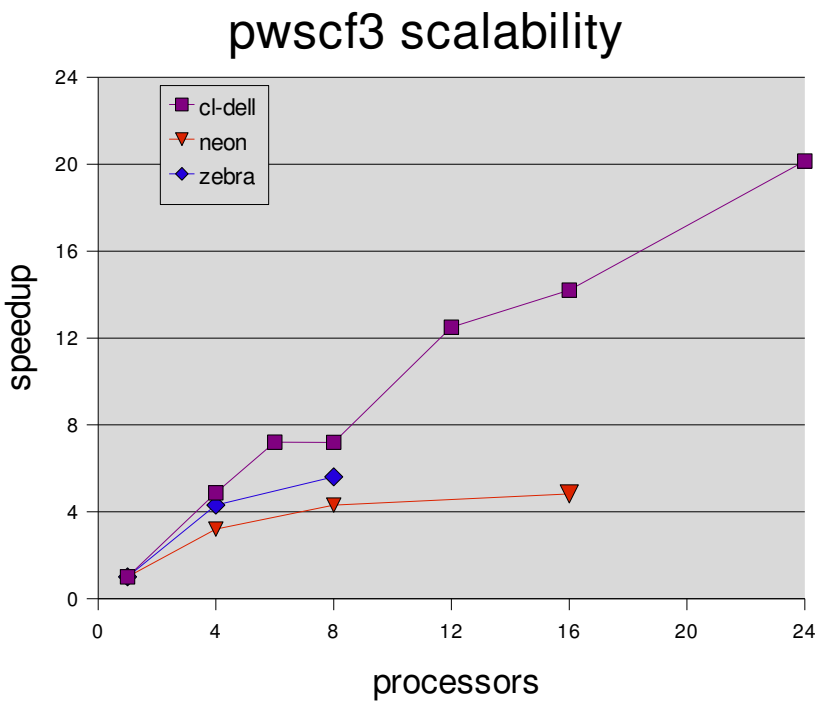


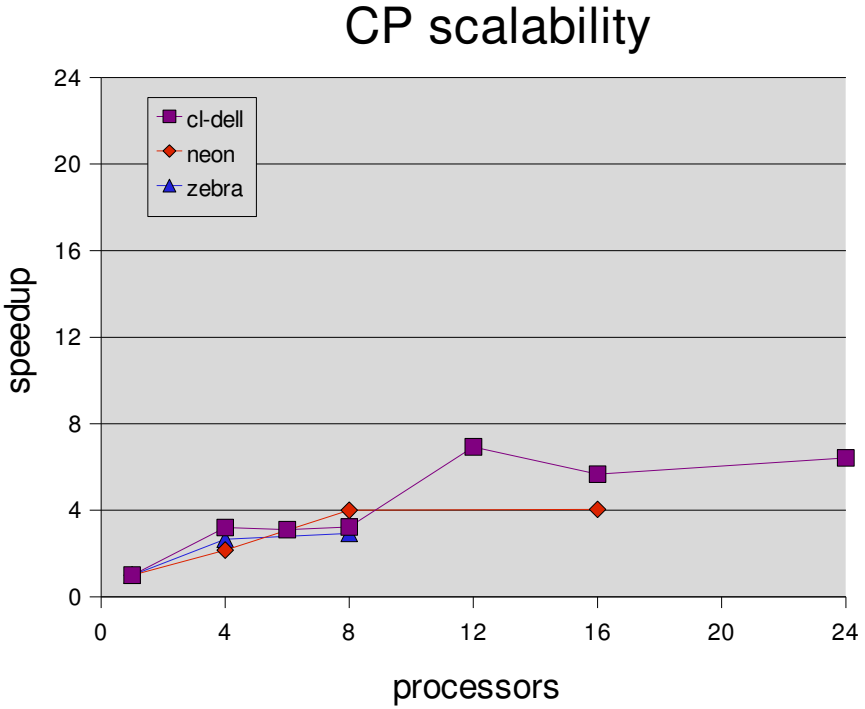
a few results....

test	CPU no.	blade	zebra	neon	cl-dell
pwscf3	1	3898	2786	**2551**	3124
	4	1971	647	798	**641**
	8		497	528	**434**
	16		n.a.	528	**220**
cptest	1	1254	776	**736**	970
	4	709	**292**	340	303
	8	528	265	**184**	300
	16		n.a.	182	**171**

Table 6: Compared results (best performance highlighted)

scalability results





more results...

test	CPU no.	av. wct (sec.)	std. dev.
pwscf3	1	3898	10
	2	2576	287
	4	1971	109
	8		
cptest	1	1254	1
	2	709	294
	4	528	49
	8		0

Table 2: Results overview on *blade*

test	CPU no.	av. wct (sec.)	std. dev.
pwscf3	1	2786	0
	2	1644	402
	4	647	3
	8	497	4
cptest	1	776	1
	2	405	4
	4	292	0
	8	265	0

Table 3: Results overview on *zebra*

Finally..

The more you benchmark

the less you understand..