IPACS Benchmarks @ DESY Zeuthen

Valerio Angelini
Università degli Studi di Firenze
amailp@gmail.com

The aim of the project was to run the IPACS Benchmark Suite on the DESY Zeuthen PC-clusters and to be able to evaluate parallel performances about different aspects of parallel computing.

1 Introduction

1.1 IPACS Benchmark Suite

The IPACS-Project's (Integrated Performance Analysis of Computer Systems) [2] objective is to develop methods for measuring system performance on High Performance Computers (HPC) based on low level benchmarks, compute kernels, open-source and commercial application benchmarks.

The IPACS team has selected a complete set of benchmark software, and has developed an user friendly Java client (IPACS-Client) to run the benchmarks and to graph the corresponding results.

The characteristics of the benchmarks are briefly described in Table 1.

For more details check [1] and relative bibliography.

1.2 Cluster equipment at DESY Zeuthen

The benchmarks were tested on two different clusters, which have both two different special networks for HPC (high bandwidth, low latency). Details are described in Table 2.

2 Developed software

Due to the early phase of development of the IPACS-Client and the lack of documentation regarding it, there has been no possibility to use it in the current clusters' environment.

Therefore it has been necessary to develop a set of utilities that makes it possible to execute the benchmarks easily.

In particular four commands have been set:

`run`, `list`, `gather`, `graph`.

The user needs to launch these commands directly from the following *path*:

`/afs/ifh.de/user/a/angelini/benchs`

2.1 `run`: compiling and running benchmarks

`run` is a *shell* script that allows to compile, configure and send a generic benchmark to the *batch system*.

The user can choose, with an uniform interface, all the parameters that each benchmark needs to be run.

The command syntax is:

```
run BENCH_NAME [-ppn NODE_NUM_PROCS]
               [-pa PROC_ARRAY]
               [-cl CLUSTER]
               [-scr SCRATCH_DIR]
               [-h]
```

`BENCH_NAME` is a required argument. It selects the benchmark to be run.

`NODE_NUM_PROCS` allows to choose how many processes have to be sent to each node (also known as `ppn`). The default value is 2.

`PROC_ARRAY` specifies how many total processes have to be created. The default value is 4. An even integer must be used if `NODE_NUM_PROCS=2` is set.

`CLUSTER` is the string that specifies the name of the cluster in which the benchmark has to be run. The values allowed are `Plejade` and `Geminide`.

`SCRATCH_DIR` is the *file system* type to be used by the benchmark. The values allowed are `panfs` and `afs`.

Type	Name	Short description
Low-Level	PRIOmark	file system and disk I/O performance
	CacheBench	performance of the memory hierarchy
	PMB	suite evaluating most of the features provided by MPI-IO
	b_eff	accumulated network bandwidth of parallel/distributed computing systems
Compute Kernels	TauBench	unstructured grid benchmark, kernels are derived from Tau
Open Source	PARPACBench	complex three-dimensional flow problems, generalized lattice Boltzmann method
	Linpack	solve linear equations and linear least-squares problems
	ddfem	parallel numerical linear-elasticity solver
Commercial	Fluent	computational fluid dynamics
	PowerFlow	computational fluid dynamics
	StarCD	computational fluid dynamics

Tab. 1: *Benchmarks in IPACS Benchmak Suite*

	"Plejade" cluster	"Geminide" cluster
CPU manufacturer	AMD	Intel
CPU model	Opteron 250	Xeon P4
CPU frequency	2.4 GHz	1.7 $\sim$ 2.0GHz
CPU cache	1 MB L2	256 kB
CPU register width	64	32
memory size	4 GB	1 GB
memory model	PC2700 ECC DDR SDRAM	RDRAM
host adapter	Mellanox Infiniband HA 4X	Myrinet 2000 M3F-PCI64B-2
switch	Mellanox InfiniScale III 2400, 24 ports	M3-E32 5 slot chassis, 2xM3-SW16 line cards
nodes(processors)	16 (32)	24 (48)

Tab. 2: *Hardware details*

2.2 `list`: getting info about running benchmarks

`list` is a *Ruby* script that checks the status of the user's benchmarks, and shows the main information on the command line.

2.3 `gather`: gathering results from finished benchmarks

`gather` is a *Ruby* script that gets results from all finished benchmarks and puts them in the *result tree*.

This can be seen in the `results` directory. Each result is hierarchically set on the basis of its own properties, the same ones that the user specified with the `run` command.

In addition the *timestamp* of the date and time in which the benchmark was run is also recorded.

2.4 `graph`: displaying the results of the benchmarks

`graph` is an interactive command line program written in *Ruby*.

It allows to display a graph with a selection of results of a previously chosen benchmark.

If results of the same type are selected, an average of them is calculated, and the corresponding *standard deviation* is displayed in the graph.

To run the program, the user has to pass the name of the benchmark to the command line with the following syntax:

```
graph [BENCH_NAME]
```

If a graphicable benchmark has been selected, a command prompt shows up, preceded by the list of all the results successfully recorded for the selected benchmark (Figure 1).

With the command `graph` the user can print the graph of all the results (Figure 2).

It is also possible to show different data for each benchmark with the `data` command (Figure 3).

The user can then refine the results with the `search [regexp]` command (Figures 4 and 5).

For the complete list of commands available, run the `help` command.

3 State of development of each benchmark

Due to very different configuration files, to the various compiling processes, to the different libraries needed, to the different formats of the results, etc., every benchmark is in a different state of development. Here is a brief status list:

3.1 PRIOmark

Executing

It runs well on both Geminide and Plejade.

Displaying results

Full graph support available.

Notes

In Figure 6 it is possible to see that the *Panasas* file system (panfs) always gets better results than afs on both clusters. It would be interesting to better investigate the strange behavior of the benchmark with Plejade on panfs (purple bars).

3.2 CacheBench

Executing

It runs well on both Geminide and Plejade.

Displaying results

It is not integrated in the program to generate graphs, because it is made to run on only one processor. However it has an internal script to generate nice graphs with Gnuplot.

Notes

Figure 7 is the graph generated by CacheBench on the Plejade cluster.

The two levels of cache are clearly visible in this graph: the first big step indicates that the benchmark has reached the first level of cache (64k), the second step indicates the second level of cache (1M).

3.3 PMB

Executing

It runs well on both Geminide and Plejade.

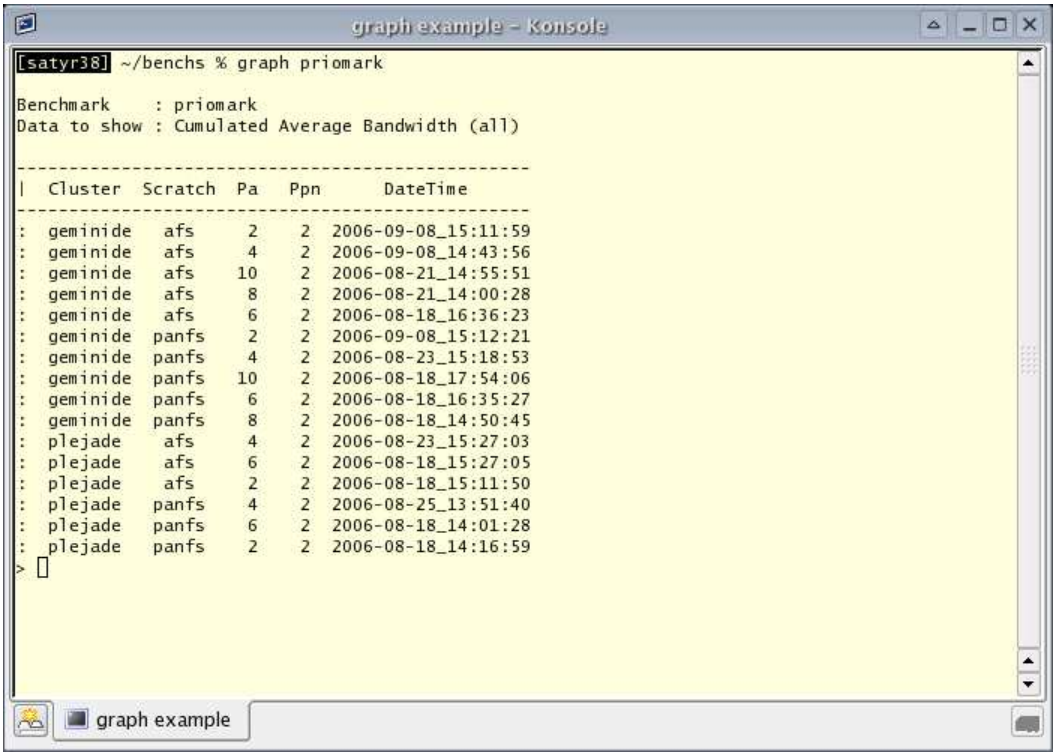

Fig. 1: *At the beginning all the results are shown*

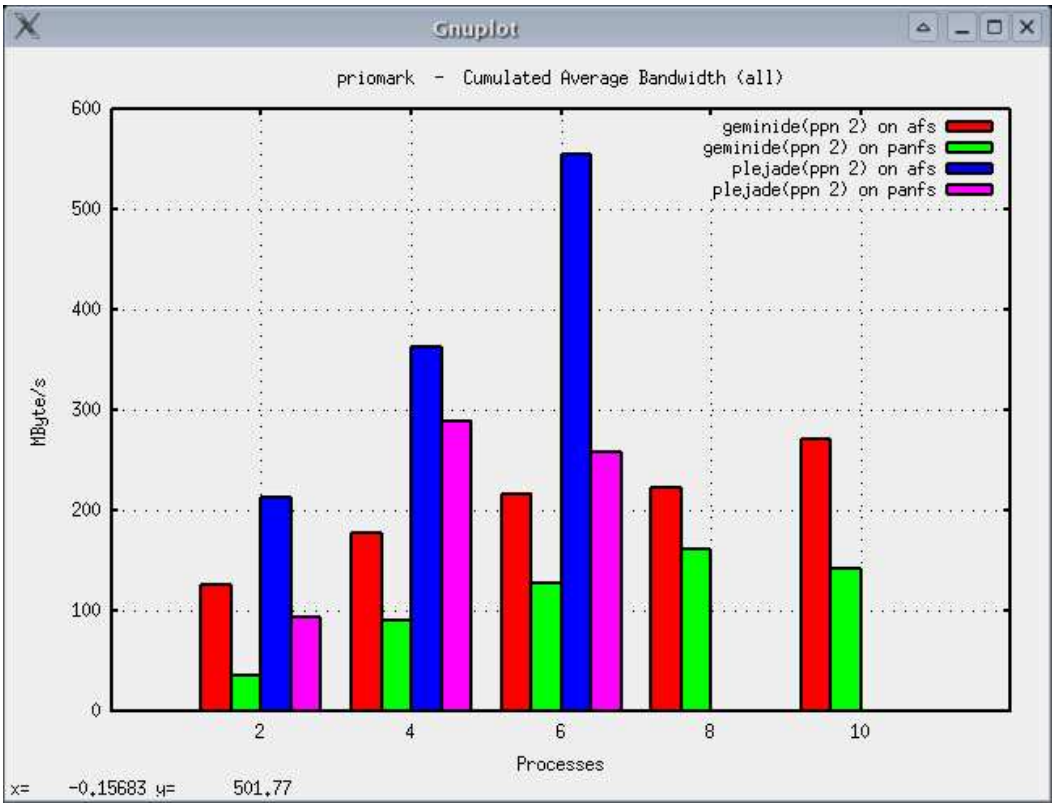

Fig. 2: *The graph of all the results*

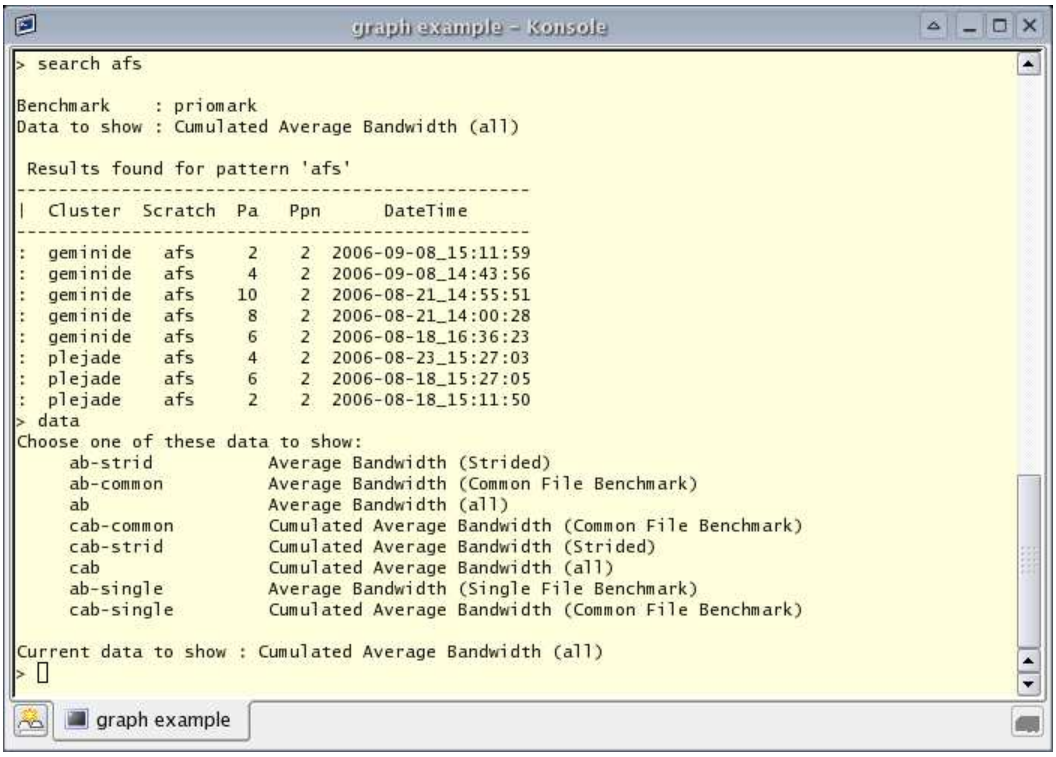

Fig. 3: *The possible data types to be shown*

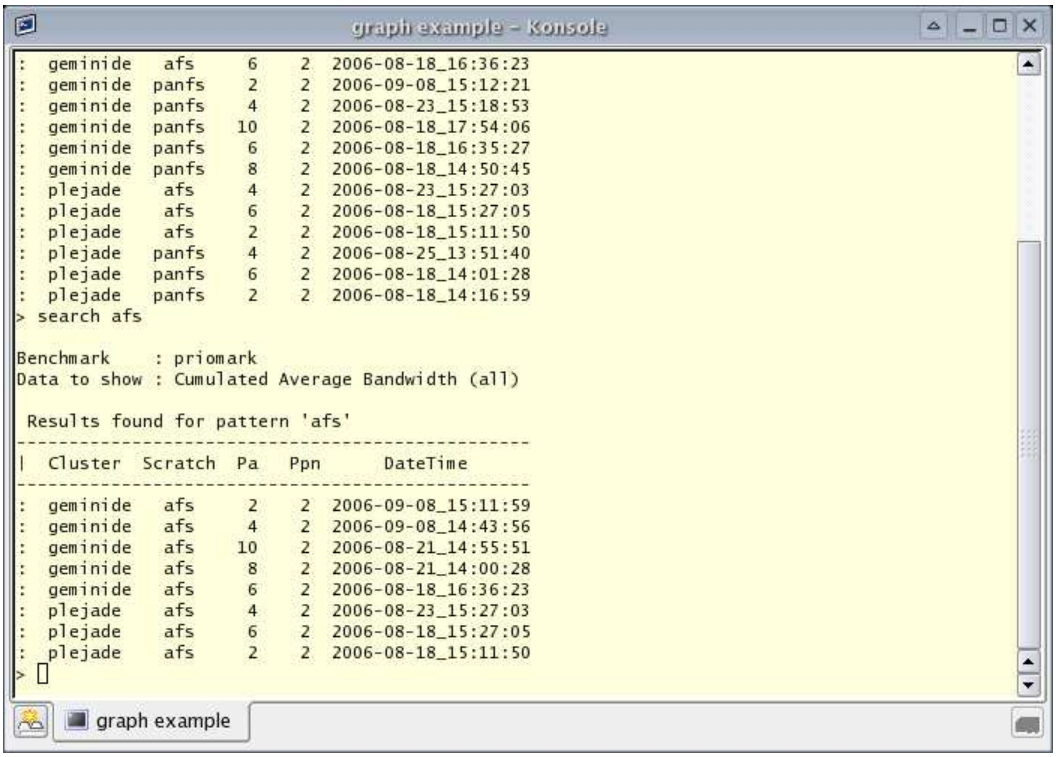

Fig. 4: *Results shown with the command* `search afs`

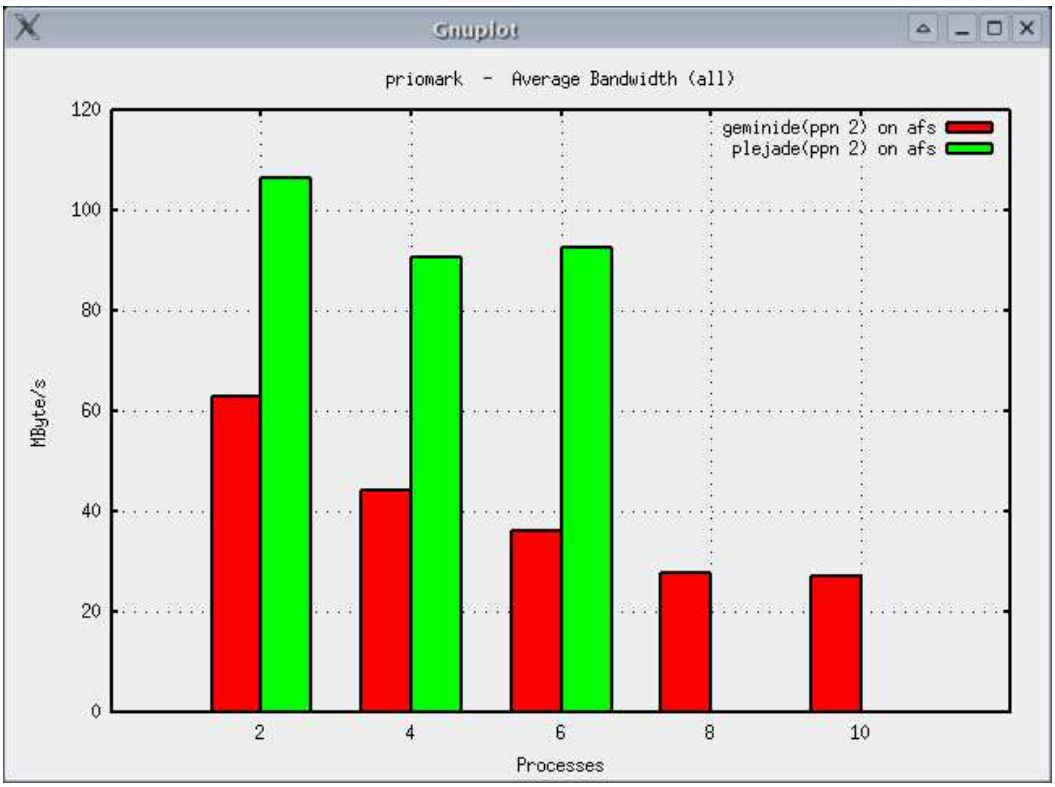

Fig. 5: *Graph of another data with a reduced result set*

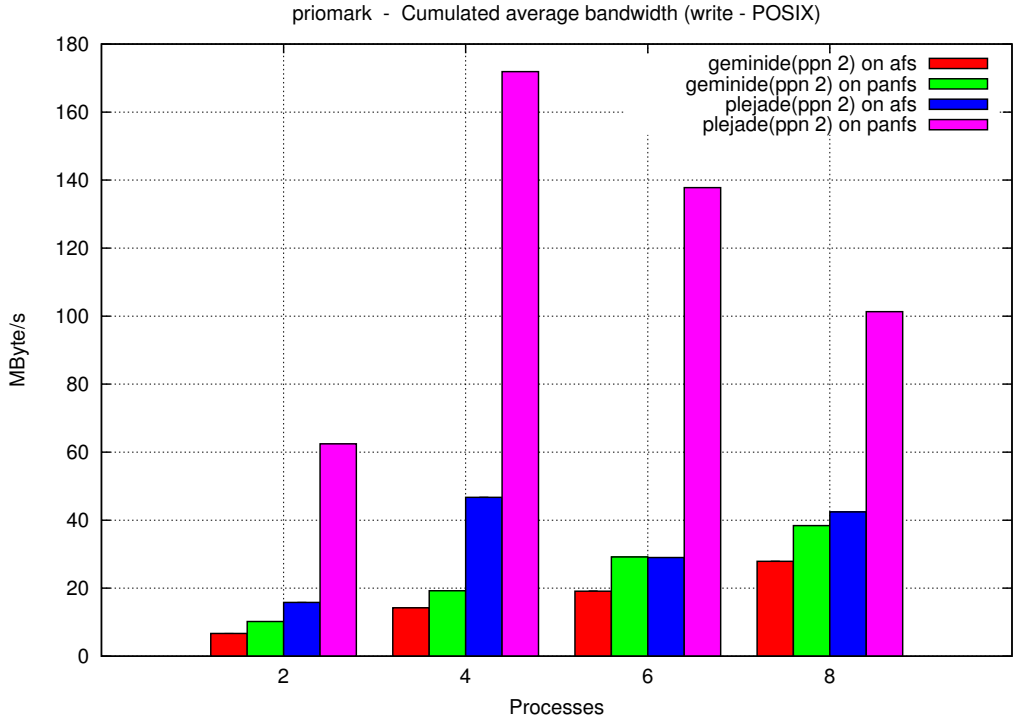

Fig. 6: *PRIOmark on both clusters, with different file systems*

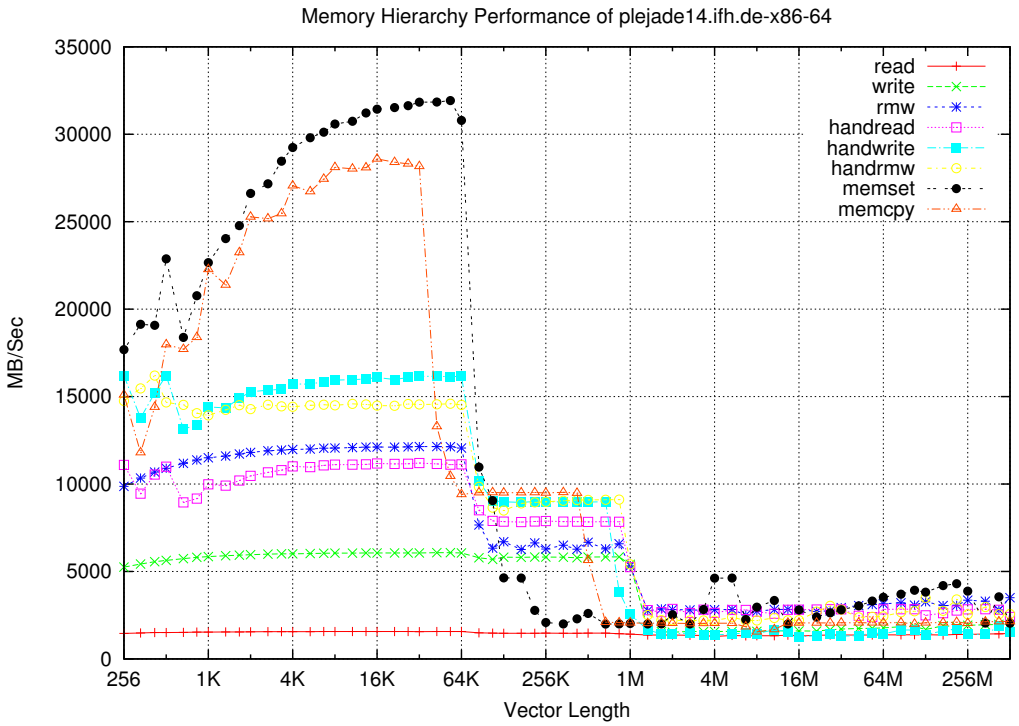

Fig. 7: *CacheBench on Plejade*

Displaying results

There was no time to integrate it in the graph generating system.

3.4 b_eff

Executing

It runs well on Geminide. On Plejade it gets very strange results for obscure reasons.

Displaying results

Full graph support available.

It has also a very good script to auto generate *PDF* reports.

Notes

In Figure 8 it is possible to see that b_eff scales well on Geminide cluster, in fact the bandwidth increases almost linearly with the number of processors. The strange behavior of the first column is due to the fact that the two processes were running on the same node, therefore it is reasonable to observe a greater bandwidth and a lower latency.

Figure 9 shows the trend of the latency increasing the number of processors. The values are almost the same for all the configurations: this means that the network is not the bottleneck, and could stand more load.

3.5 TauBench

Executing

It runs well on both Geminide and Plejade.

Displaying results

Full graph support available.

Notes

In Figure 10 it is possible to see a heavy predominance of the Plejade cluster on the Geminide one in this type of benchmark: special attention should be paid to the logarithmic scale of the y axis.

In particular Plejade is faster by factor of 60, 55, 53 and 55, respectively for 2, 4, 6, and 8 processors. Such a big difference is probably due to an I/O overhead in Geminide. It could be that all the data fit in the second level cache

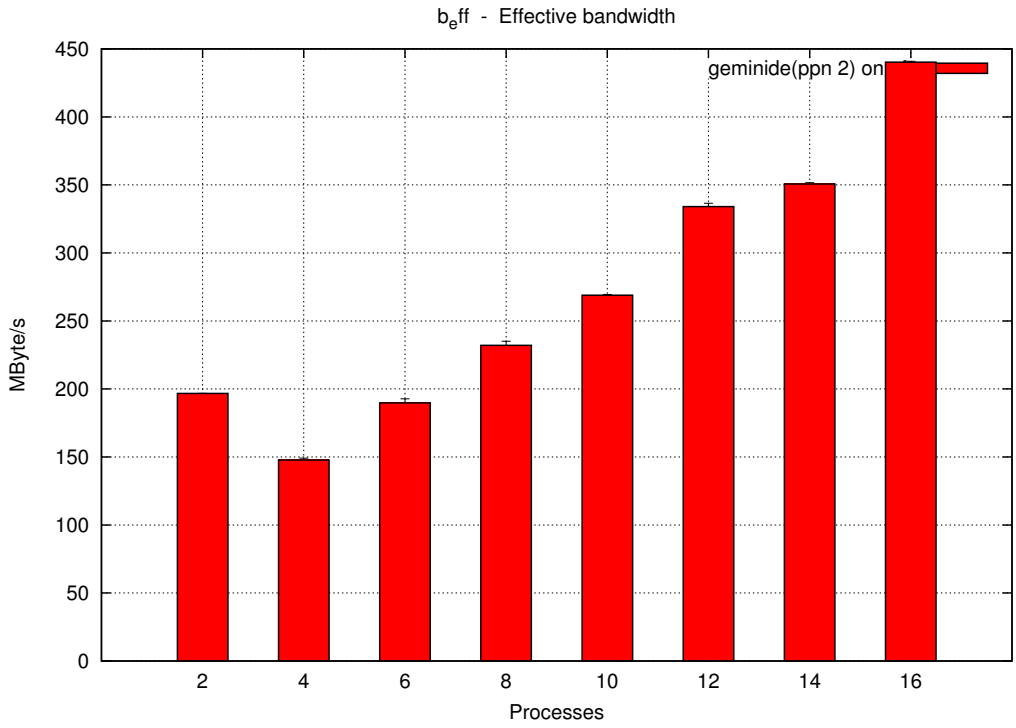

Fig. 8: *b_eff on Geminide, showing the effective bandwidth.*

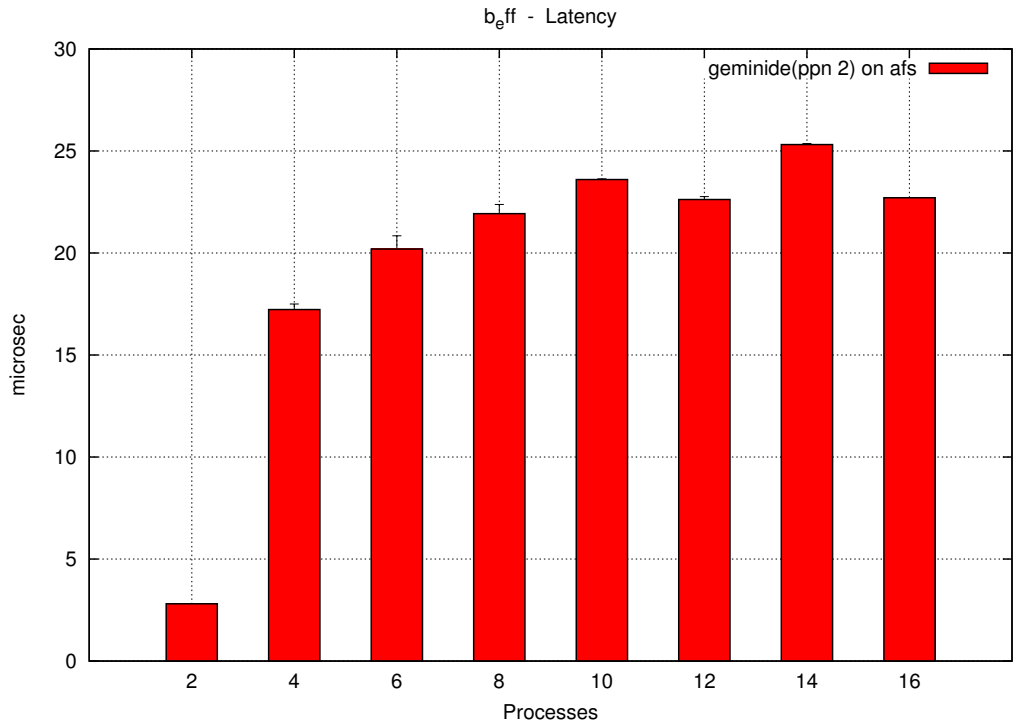

Fig. 9: *b_eff on Geminide, showing the latency.*

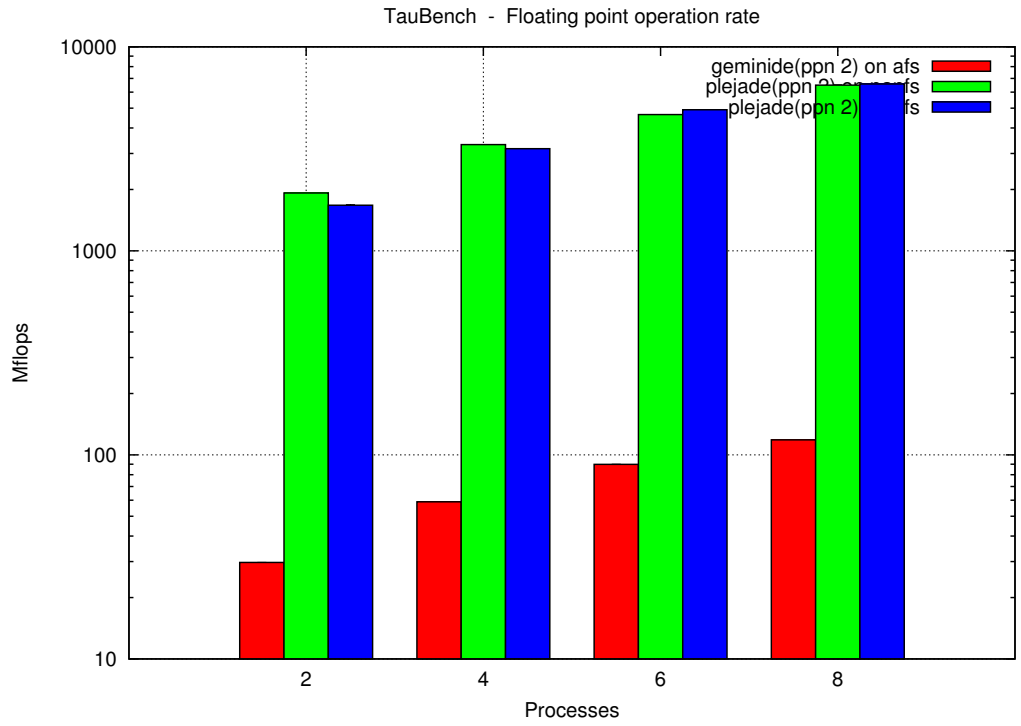

Fig. 10: *TauBench on different clusters (logarithmic scale of the y axis).*

of Plejade and not in Geminide's, that is 4 times smaller (1 MB vs. 256 kB).

3.6 PARPACBench

Executing

It runs well on both Geminide and Plejade.

Displaying results

Full graph support available.

Notes

In Figure 11 it is possible to see a good scalability of both clusters in PARPACBench.

In this example Plejade is faster by an average factor of $2,7$.

3.7 Linpack

Executing

Because of very different installation procedures, there was no time to integrate this benchmark in the `run` command.

3.8 ddfem

Executing

It is not integrated in running environment, because of lack of "petsc" libraries on both clusters.

Unfortunately there was no time to set up the right environment.

3.9 Commercial benchmarks

Executing

None have been tried. They require commercial software to be run.

In particular they generate a test file that has to be evaluated by the corresponding software (Fluent, PowerFlow, StarCD). All these softwares are not open source and not free.

4 General issues

During my work on the clusters I encountered some issues that I would like to report for future studies.

1. It does not seem possible to run an MPI program on more than 4 nodes on Plejade (8

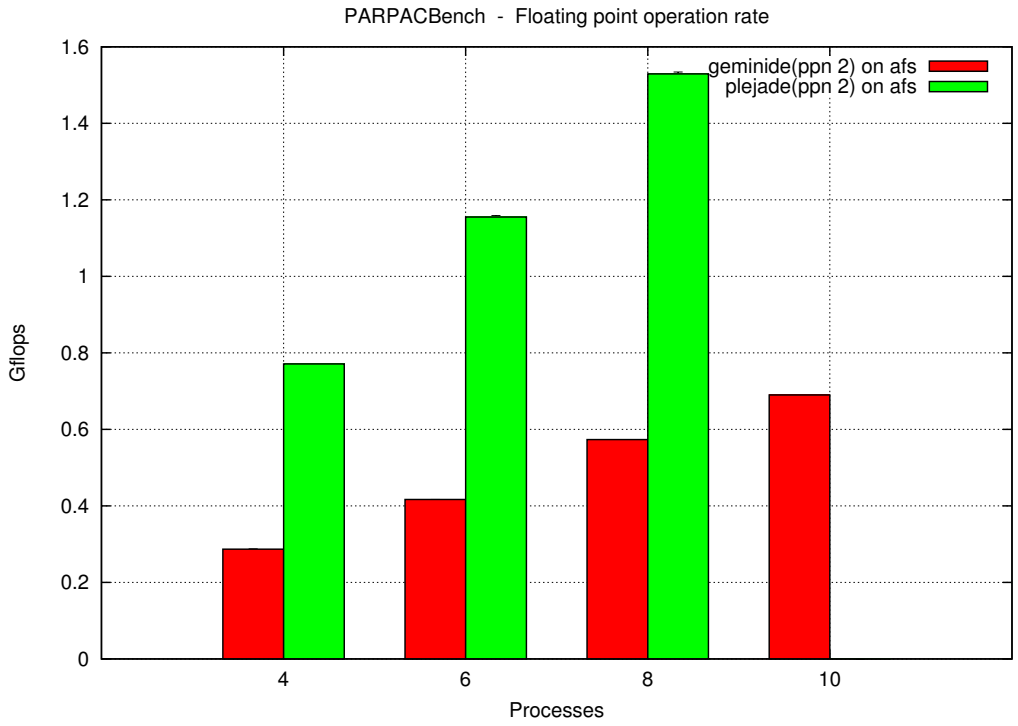

Fig. 11: *PARPACBench on different clusters.*

processors maximum). It seems a problem related to the *batch system*, because I also tried to recompile a very simple example program with different compilers and the behavior was always the same.

2. I found that the IPACS Benchmark Suite uses a different formalism regarding the `GridEngine` files. In particular they use different *machinefile* and *nodefile*. Related to this is the fact that I was not able to run most benchmarks with one process per node.

5 Acknowledgments

I would like to thank Mr Goetz Waschk and my supervisor Mr Peter Wegner.

I'm also very grateful to Mr Karlheinz Hiller, Mrs Sabine Baer and all the Summerstudent Team for the perfect organization, and to Mr Gregorio Landi and Mrs Elisabetta Gallo for making this great experience in DESY Zeuthen possible.

A special thank also to SP!

References

[1] G. Falcone, H. Kredel, M. Krietemeyer, D. Merten, M. Merz, F-J. Pfreundt, C. Simmendinger, D. Versick, "The IPACS-Project at Glance", Germany, September 28, 2005

[2] "IPACS Benchmark" Website, http://www.ipacs-benchmark.org/

[3] I. Hailperin, "High Performance Computing Challenge on small Linux Clusters", DESY Zeuthen, September 7, 2005

[4] M. Snir et alii, "MPI -The Complete Reference", The MIT Press, Cambridge, 1998

[5] T.L. Sterling et alii,"How to Build a Beowulf", The MIT Press, Cambridge, 1999

[6] Gregory R. Andrews,"Foundations of Multithreaded, Parallel, and Distributed Programming", The MIT Press, Cambridge, 1999

[7] Yukihiro "matz" Matsumoto, "Ruby Programming Language" Website, http://www.ruby-lang.org/en/

[8] T. Williams, C. Kelley, "gnuplot -An Interactive Plotting Program", April 2005, http://www.gnuplot.info/docs_4.1

[9] "Online Oxford Dictionaries", Website, http://www.askoxford.com