DESIGN AND PERFORMANCE EVALUATION OF A LINUX HPC CLUSTER

DONATO PERA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
Università degli Studi dell'Aquila
via Vetoio (snc), Località Coppito, L'Aquila 67010, Italy

(received: 9 February 2018; revised: 23 February 2018;
accepted: 6 March 2018; published online: 20 March 2018)

Abstract: In this paper after a short theoretical introduction about modern techniques used in parallel computing, we report a case study related to the design and development of the Caliban Linux High Performance Computing cluster, carried out by the author in the High Performance Computing Laboratory of the University of L'Aquila. Finally we report some performance evaluation tests related to the Caliban cluster performed using HPL (High-Performance Linpack) benchmarks.

Keywords: high performance computing, parallel computing, cluster design, HPL
DOI: https://doi.org/10.17466/tq2018/22.2/b

1. Introduction

Computational mathematics is one of the most important fields of research in modern mathematics, because by using computational techniques it is possible to get numerical solutions to describe the behavior of several scientific problems. Numerical solutions related to the mathematical models that describe these scientific problems often require a high spatial resolution to capture details, as a consequence, long computational times are often required when using a serial implementation of a numerical scheme. Parallel computation on cluster computers can improve dramatically the time efficiency and give scientists the possibility to analyze their problems reducing the cost related to real experiments using only software simulations on high performance computing structures.

In this paper we report information related to the design, development and performance evaluation of a Linux High Performance Computing Cluster with reference to CALIBAN (http://caliban.dm.univaq.it), the HPC cluster located at the Laboratory of High Performance Parallel Computing of L'Aquila University. In Sections 2 and 3 we describe the machine hardware features illustrating the choices that were made for the cluster components such as servers, processors, memories

and network devices. In Sections 4, 5 and 6 we report the machine software features illustrating the choices made for the operating system, the job scheduler and the system monitor. Finally, in Sections 7 and 8 we report performance tests related to the Caliban cluster performed using the HPL (High-Performance Linpack) benchmarks.

2. Hardware architecture

The choice of hardware architecture is generally made in combination with the operating system, in view of the close interaction between the two cluster components.

Generally, there are two main kinds of computer architectures based on CPU, namely architectures in shared and distributed memory [1].

In HPC shared memory architectures, CPUs work together with each other having only shared memory where they read and write data. In this kind of architecture we usually have a very short latency memory access time and very efficient systems, however it is very difficult to build single low-cost supercomputers with a large number of processors on the same motherboard. This problem can be avoided creating distributed memory HPC architectures. In this case, processors reside on multiple motherboards and work together using a suitable network system (Ethernet, Infiniband, Optical Fiber).

In this architecture, each processor (or group of processors) residing on a dedicated motherboard has a read-write memory to process data. Over the years several software libraries have been developed to use two different HPC architectures, such as openMP for shared-memory and MPI (Message Passing Interface) for distributed memory systems [2, 3]. We emphasize the fact that the latter kind of computer architectures is limited in terms of performance by a bottle-neck related to the interconnection network between different system components, however, current network technologies based on Infiniband devices with bandwidths of Gbit/s order allow avoiding the above problems and maximize in this way the calculation performance. Nowadays, it is also possible to create mixed calculation systems with an architecture based on CPU/GPU, equipping each supercomputer with one or more GPUs. This technological solution allows maximizing the performance of distributed memory architectures considering the very high computing power of GPUs [4].

During the cluster hardware design, the major issues to consider were the price, performance, power consumption, and operating system compatibility. For instance, Intel CPUs have excellent performance, but they are expensive compared to AMD processors. The first parameter to consider is then the number of cores per CPU, this parameter characterizing the entire machine and relative performance. In general, a good designer should maximize the total number of CPU-cores and, at the same time, reduce the number of cluster-nodes on which those cores are distributed. The statement above is due to the fact that the communication between cluster nodes can be slow and likely to generate performance reduction in the HPC system. In Caliban, we have decided to use

a distributed memory architecture, with mixed CPU/GPU nodes according to the following configuration:

- 18 HP Proliant DL 585 G7;
- 144 AMD Opteron 8 core 2.0 GHz (32 cores/node), 1280 Gbyte RAM, 59 HD 146 Gbyte;
- Front end node configuration (x1): HP Proliant DL 585 G7 with 4 AMD Opteron 8 core 2.0 GHz (32 cores/node), 128 Gbyte RAM, 8 HD 146 Gbyte RAID 5;
- Compute node configuration (x17): HP Proliant DL 585 G7 with 4 AMD Opteron 8 core 2.0 GHz (32 cores/node), 64 Gbyte RAM, 3 HD 146 Gbyte RAID 5;
- 6 GPU NVIDIA Geforce GTX 670 4Gbyte RAM;
- Switch Ethernet D-LINK DGS-3427 1 Gb/s;
- Switch INFINIBAND Mellanox MIS5023Q-1BFR 40Gb/s;
- Storage system QSAN P400Q-D424 60 Tbyte.

In our project, we have chosen an AMD Opteron 8 core 2.0 GHz because of the good performance and the low costs. Regarding the server architecture, we have chosen HP Proliant DL 585 G7 because this kind of a server allows installing up to 4 processors, 8 hard disks and 512 GByte of RAM with low costs.

3. Network

As mentioned in the previous section the choice of network interfaces is very important in the cluster design. In fact, network interfaces have to handle all the data traffic between compute nodes and storage devices. The fundamental parameter to consider in the network design is the bandwidth of the selected interconnection system. This choice must be made considering the kind of applications to which the HPC system is dedicated and also the costs relating to the technology capable of satisfying the design parameters. In our project we have designed the Caliban cluster with two network systems with the target of minimizing the network bottle-necks and maximizing the performance. One network is dedicated to the data traffic related to the computing operations and another is designed for data traffic related to general system services.

In particular a 1 Gb/s Ethernet (Eth0) system ensures the communication for general services and data transfer in the SAN (Storage Area Network). An Inifiband 40 Gb/s network (Ib0) ensures communication between cluster nodes during computation (Figure 1). Through this double network system we can perform service and calculation operations on two different service channels, minimizing data traffic jams on the calculation channel due to storage or service communications.

4. Operating System

After defining the hardware, another important decision in the cluster design is the choice of the operating system. In the HPC world, the most popular

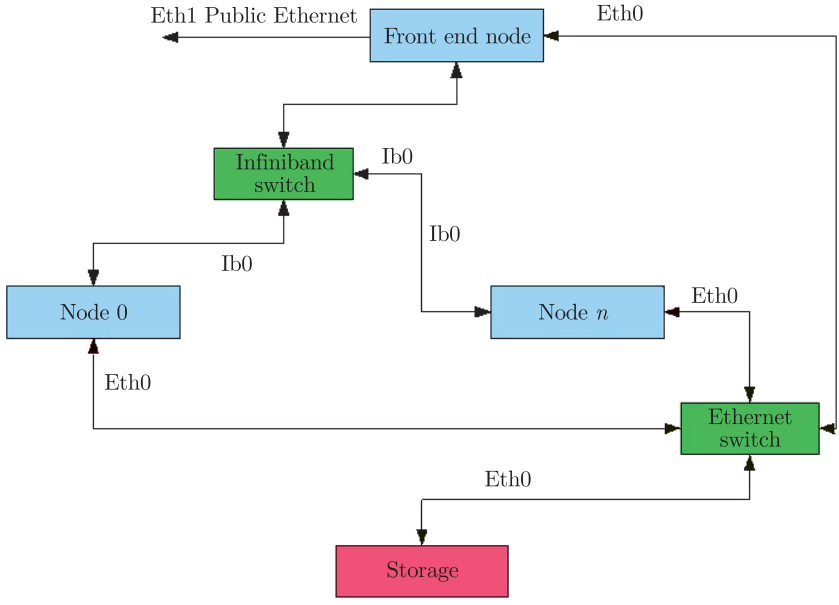

Figure 1. Caliban cluster network architecture

OS (Operating System) is Linux. In fact, nearly 90% of clusters for scientific computation are Linux based, it is possible to compare the spread of this operating system in the HPC business sector to the monopoly that Windows has with home-desktop systems; indeed, a cluster can run any operating system, be it Linux/Unix, Windows or MacOS. However, in recent years, many universities and research institutions have chosen to create Linux-based clusters in view of the great potential of this system and the low costs.

The first thing to consider is the compatibility with the hardware components selected, as well as the presence of a wide range of scientific computing libraries for development of scientific codes. Linux operating systems have excellent libraries dedicated to parallel computing, such as MPI and openMP, and also, thanks to the manufacturers' efforts, the hardware compatibility problems are reduced to a minimum. There are also Linux distributions dedicated to parallel computing that allow easy cluster development. For example, Rocks [5], Oscar [6] or Pelican HPC [7] distributions provide excellent tools for management and administration of calculation structures. For these reasons, we have chosen the distribution Rocks 5.4.3 (Maverick) based on Linux CentOS 5.5 for Caliban HPC.

The cluster is also equipped with the following compilers and software development tools:

- Operating System Rocks Cluster Maverick 5.4.3 based on Linux CentOS 5.5;
- Job scheduler Sun Grid Engine SGE 5.2;
- Software libraries openMP, openMPI, MPICH, BLAS, LAPACK, ScaLAPACK;

- Compilers gcc, gfortran, g++, Intel Fortran, PGI Fortran 14.0-16.7;
- Web page http://caliban.dm.univaq.it;
- NVIDIA CUDA Toolkit and compiler 4.2/5.0/6.5;
- FreeFem++ ver. 3.19.

These tools allow us to develop parallel scientific codes in programming languages C, C++, and Fortran. It is also possible to simulate FEM (Finite Element Method) problems through the use of FreeFem++ libraries.

5. Job scheduling

The use of a job scheduler in a cluster is intended to make better use of computing resources among various users. In fact, one of the major problems in HPC clusters is work management inside the machine. In general, in a system without a job scheduler, we may have to deal with working conditions in which one or more users can monopolize calculation resources preventing all other users from work. Moreover, submission of process groups without any kind of optimization and control may also lead to sub-optimal use of the computational structures. A solution to all these problems lies in the use of job scheduling systems. Over the years different job-manager systems have been developed such as Sun Grid Engine (SGE) [8], Torque [9] or Moab [10]. In the Caliban system we have installed and configured the SGE (Sun Grid Engine) job scheduler version 5.2. This system allows the creation of job queue families defining priorities and duration (wall time) parameters to each group. In general, to use resources in the best way and to avoid bottle-necks due to the execution of computationally complex (and therefore very long) works, it is a good practice to define different job queue families for different jobs.

To face the data-traffic management problems described above in our cluster we define a job-class for *debug codes* with high priority and a very short wall time, a job-class for *ordinary codes* with medium priority and a not too long wall time, and finally a job-class with a very low priority and a very long wall time to execute more computationally expensive codes. Following these guidelines, we have defined the following job queue families on Caliban:

Table 1. Caliban cluster job queue families

	Debug	Parallel	Long	Large	GPU
Wall Time	30 min	24 h	336 h	24 h	24 h
Priority	+10	0	−10	0	0
Max resources	12 nodes	12 nodes	6 nodes	1 node	6 nodes

Particularly, the *Debug* queue is used during the development of codes or during the fine-tuning of existing codes; the *Parallel* queue is used to execute ordinary works that would require the use of computing libraries such as openMP or MPI on parallel CPU based architectures; *Long* queue is used for works that require the same characteristics of the works on *Parallel* class, but for which a very

long calculation time is estimated. For this reason, the wall time of the *Long* class was set as equal to 336 h (2 weeks) and the priority is very low in order to avoid unnecessary traffic-jams in the job management. The *Large* queue is similar to the *Parallel* and *Long* queues, but it is designed to run jobs that require a lot of RAM. For economic and project reasons, on Caliban there is only one "fat node" with 128 GB of RAM instead of 64 Gbytes as other regular nodes, therefore, the use of this queue is limited to one node only. Finally, the *GPU* queue is designed for codes that require the use of GPU computing architectures and have similar characteristics as the *Parallel* queue in terms of priorities and wall time.

6. System Monitoring

In order to facilitate job-monitoring on the Caliban supercomputer, we installed the Ganglia monitoring system (on version 3.1.7). This software being a standard on Rocks cluster allows obtaining information about the status of the front-end node and compute nodes, through this instrument we can monitor also the job-queue status managed by SGE.

7. Performance Evaluation

In this section, we report the performance evaluation tests carried out using the HPL (High Performance Linpack) benchmark [11]. HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers. It can thus be regarded as a portable as well as freely available implementation of the High Performance Computing Linpack Benchmark. The algorithm used by HPL can be summarized by the following keywords:

- Two-dimensional block-cyclic data distribution;
- Right-looking variant of the LU factorization with row partial pivoting featuring multiple look-ahead depths;
- Recursive panel factorization with pivot search and column broadcast combined.

The HPL package provides a testing and timing program to quantify the accuracy of the obtained solution as well as the time it takes to compute it. The best performance achievable by this software on cluster systems depends on a large variety of factors. Nonetheless, with some restrictive assumptions on the interconnection network, the algorithm described here and its implementation are scalable in the sense that their parallel efficiency is maintained constant with respect to the per processor memory usage. The HPL software package requires the availability of an implementation of the Message Passing Interface MPI (1.1 compliant) on the cluster system. An implementation of either the Basic Linear Algebra Subprograms BLAS or the Vector Signal Image Processing Library VSIPL is also needed. Machine-specific as well as generic implementations of MPI, the BLAS and the VSIPL are available for a large variety of systems [11].

In our test, we use the PDGESV subroutine contained on the Scalapack library, which is a parallel version of the DGESV library contained on the Lapack library, this last subroutine computes the solution of a system of linear equations:

$$AX = B \tag{1}$$

where A is an n by n real matrix, X and B are n by r real matrices. This software tool uses the LU decomposition with partial pivoting and row interchanges to factorize A as $A = PLU$, where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factor form of A is then used to solve the system of equations $AX = B$. In a parallel case with PDEGSV we solve the system

$$\text{sub}(A) \times X = \text{sub}(B) \tag{2}$$

where $\text{sub}(A) = A(IA{:}IA+N-1, JA{:}JA+N-1)$ is an N-by-N distributed matrix, X and $\text{sub}(B) = B(IB{:}IB+N-1, JB{:}JB+NRHS-1)$ are N-by-$NRHS$ distributed matrices, and:

- N (global input) INTEGER is the number of rows and columns to be operated, *i.e.* the order of the distributed submatrix $\text{sub}(A)$, $N \geq 0$.
- $NRHS$ (global input) INTEGER is the number of right hand sides, *i.e.* the number of columns of the distributed submatrix $\text{sub}(B)$, $NRHS \geq 0$.
- A (local input/local output) is a DOUBLE PRECISION pointer into the local memory to an array of dimension $(LLD_A, LOCc(JA+N-1))$. On entry, we have the local pieces of the N-by-N distributed matrix $\text{sub}(A)$ to be factored. On exit, this array contains the local pieces of the factors L and U from the factorization $\text{sub}(A) = P \times L \times U$; the unit diagonal elements of L are not stored.
- IA (global input) INTEGER is the row index in the global array A indicating the first row of $\text{sub}(A)$.
- JA (global input) INTEGER is the column index in the global array A indicating the first column of $\text{sub}(A)$.
- B (local input/local output) is a DOUBLE PRECISION pointer into the local memory to an array of dimension $(LLD_B, LOCc(JB+NRHS-1))$. On entry, we have the right hand side distributed matrix $\text{sub}(B)$. On exit, if $INFO = 0$, $\text{sub}(B)$ is overwritten by the solution distributed matrix X.
- IB (global input) INTEGER is the row index in the global array B indicating the first row of $\text{sub}(B)$.
- JB (global input) INTEGER is the column index in the global array B indicating the first column of $\text{sub}(B)$.

The LU decomposition with partial pivoting and row interchanges is used to factorize $\text{sub}(A)$ as $\text{sub}(A) = P \times L \times U$, where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. L and U are stored in $\text{sub}(A)$. The factored form of $\text{sub}(A)$ is then used to solve the system of equations $\text{sub}(A) \times X = \text{sub}(B)$ [12].

In the tests carried out on the Caliban supercomputer, we considered matrices having dimensions equal to 20 000 and 40 000 elements with evaluation sub-blocks equal to 1000 elements, respectively, where:

- Matrix A is randomly generated for each test;
- The following scaled residual check is computed: $||Ax-b||_\infty/(\text{eps}\cdot(||x||_\infty\cdot||A||_\infty+||b||_\infty)\cdot N)$;
- The relative machine precision (eps) is taken to be $1.110223\cdot 10^{-16}$;
- Computational tests pass if scaled residuals are less than 16.0.

All tests were performed with up to 3 nodes, using a number of cores between 1 and 75. With respect to the Caliban cluster structure, characterized by compute-nodes with 32 cores, the tests up to 30 cores were carried out using a single computing node, whereas the tests with 50 and 75 processors were performed using two nodes with 25 cores and 3 nodes with 25 cores, respectively. Tables 2 to 3, reported below, show the performance obtained for tests performed varying the number of processors. We report also graphs relating to the computation time (in seconds), the computing power (in Gflops), the speed-up and the efficiency for the two test problems described above.

Table 2. Caliban cluster (3 nodes) HPL benchmark, performance (a) $N=20\,000$ and (b) $N=40\,000$, $NB=1000$

(a)

N_{procs}	Time (s)	Gflops
1	822.16	6.488
10	154.36	34.55
15	134.47	39.67
20	113.65	46.93
25	113.14	47.14
30	113.11	47.16
50	63.98	83.36
75	47.98	111.2

(b)

N_{procs}	Time (s)	Gflops
1	6512.10	6.552
10	926.90	46.03
15	736.11	57.97
20	613.17	69.59
25	576.31	74.04
30	537.56	79.37
50	315.23	13.54
75	219.79	19.41

Table 3. Caliban cluster (3 nodes) HPL benchmark speed-up and efficiency (a) $N=20\,000$ and (b) $N=40\,000$, $NB=1000$

(a)

N_{procs}	Speed up	Efficiency
1	1	1
10	5.32	0.53
15	6.11	0.40
20	7.23	0.36
25	7.26	0.29
30	7.27	0.24
50	12.85	0.26
75	17.13	0.23

(b)

N_{procs}	Speed up	Efficiency
1	1	1
10	7.02	0.70
15	8.84	0.59
20	10.62	0.53
25	11.30	0.45
30	12.11	0.40
50	20.65	0.41
75	29.62	0.39

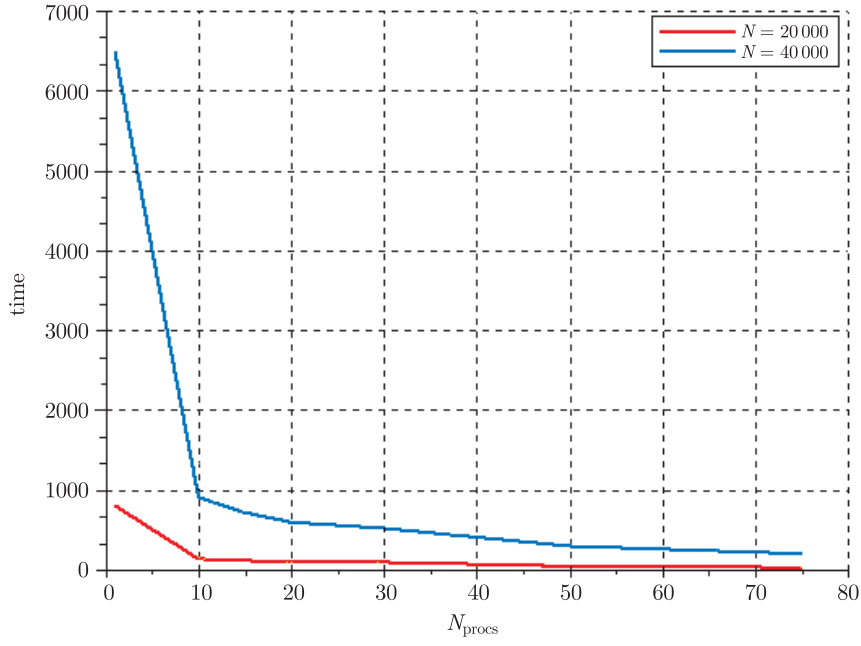

Figure 2. Caliban cluster (3 nodes) HPL benchmark, performance $N = 20\,000$, $NB = 1000$ and $N = 40\,000$, $NB = 1000$ (Time Computing graph)

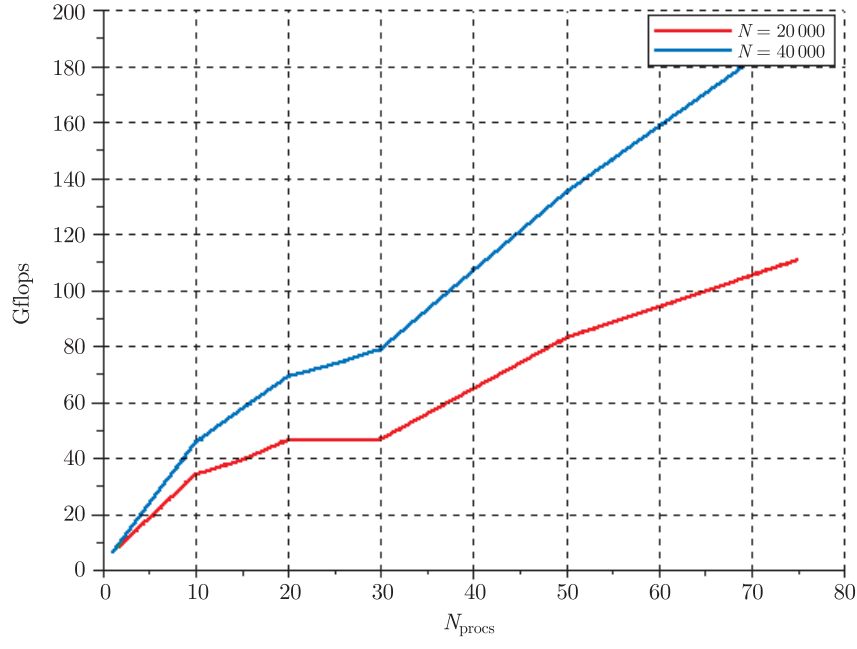

Figure 3. Caliban cluster HPL (3 nodes) benchmark, performance $N = 40\,000$, $NB = 1000$ (Power computing graph)

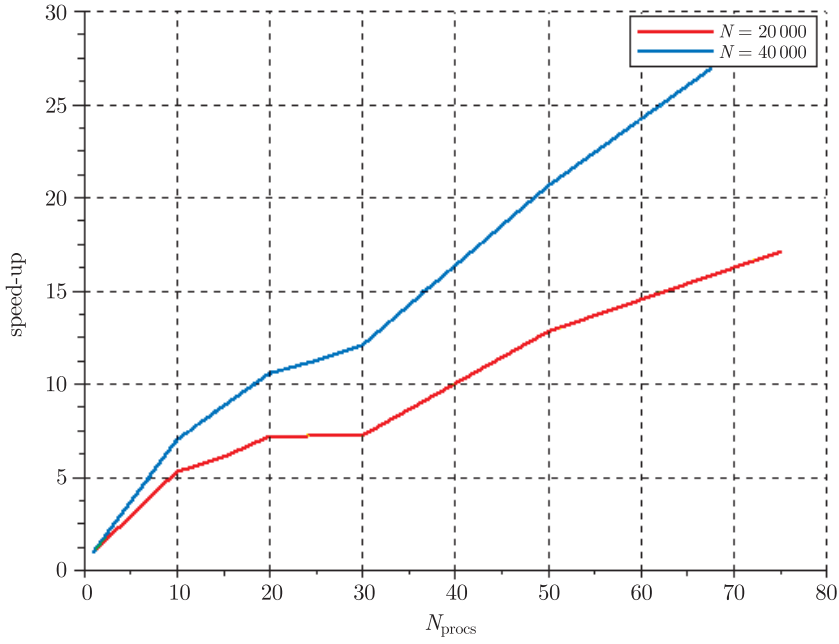

Figure 4. Caliban cluster HPL (3 nodes) benchmark, $N = 20\,000$, $NB = 1000$ and $N = 40\,000$, $NB = 1000$ (Speed-Up)

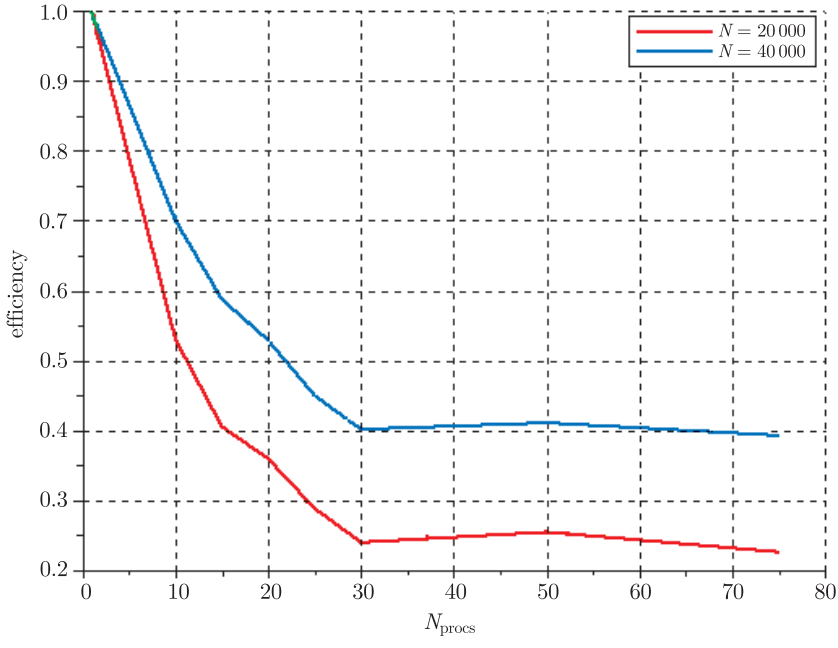

Figure 5. Caliban cluster HPL (3 nodes) benchmark, $N = 20\,000$, $NB = 1000$ and $N = 40\,000$, $NB = 1000$ (Efficiency)

8. Conclusions

Analyzing the speed-up we can see that the performance grows sublinearly in relation to the number of cores used. In our tests we run the codes using one single node ($N_{\text{procs}} < 32$) and multinodes ($N_{\text{procs}} > 32$) in both cases we get a sublinear speed without saturation. The efficiency decreases as the number of cores is increased, in particular the efficiency decreases quickly in the single node case and we obtain saturation after the threshold of 32 cores (multi-node case). Such behavior could be related to the Ware-Amdahl's Law according to a non-zero serial execution part of the algorithm used during the tests.

References

[1] Kupferschmid M 2009 *Classical Fortran: Programming for Engineering and Scientific Applications. Second Edition*, CRC Press
[2] Rauber G 2010 *Runger Parallel Programming*, Springer
[3] Pacheco P 2011 *An introduction to parallel programming*, Morgan Kaufmann
[4] Kirk D B and Hwu W-M W 2010 *Programming Massively Parallel Processors: A Hands-on Approach*, Morgan Kaufmann Publishers Inc., San Francisco
[5] [Online] available at: `http://www.rocksclusters.org/wordpress/`
[6] [Online] available at: `http://www.csm.ornl.gov/oscar/`
[7] [Online] available at: `http://pareto.uab.es/mcreel/PelicanHPC/`
[8] [Online] available at: `http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html`
[9] [Online] available at: `http://www.adaptivecomputing.com/products/open-source/torque/`
[10] [Online] available at: `http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-basic-edition/`
[11] [Online] available at: `http://www.netlib.org/benchmark/hpl/`
[12] [Online] available at: `http://www.netlib.org/scalapack/double/pdgesv.f`