

## openMosix vs Beowulf: a case study

Moshe Bar[1,2], Stefano Cozzini[1], Maurizio Davini[3] and Alberto Marmodoro[1]

1: INFN Democritos I-34014 Trieste (Italy)

2: openMosix Project

3: Department of Physics University of Pisa( Italy)

### Abstract

A comparison between the openMosix and Beowulf clustering paradigms is presented. The evaluation work of these clustering paradigms was performed in a specific scientific environment: the INFN Democritos simulation center in Trieste (Italy.) The heterogeneous nature of the computational requirements of scientific software programs used suggests that different cluster solutions should be considered for different computational problems. Our benchmarking work aims at illustrating which are the cluster solutions that fit some of our specific computational precise requirements. The standard Beowulf paradigm still emerges as the best solution for highly parallel codes, whereas the openMosix approach allows to easily deal with embarrassingly parallel problems. Moreover this technology has proven to be superior in term of global throughput when compared with standard queueing system on small/medium cluster sizes.

# 1 Introduction

Today two main clustering paradigms exist for the Linux environment: openMosix and Beowulf (MPI based). Whereas in MPI-based solutions the parallelization needs to be explicitly coded through special library directives, in openMosix [1] <sup>1</sup> the usual Unix serial programming API can be followed, because all clustering functionalities are transparently provided from within the operating system's kernel. openMosix is a fully implemented single-system-image clustering technology which offers dynamic and adaptive load-balancing through process migration: processes requiring resources not available on the local node migrate to remote nodes where the required resources are still plentiful. openMosix also features a cache-coherent, unified namespace clustering file system as well as a series of management tools. Since all of openMosix' technology resides within the Linux kernel the cluster is completely transparent to users and applications. The process migration per se is implemented in the classic Linux demand paging mode[2]. MPI is one of the core technologies of Beowulf, similar in purposes to PVM. MPI (Message Passing Interface) is an explicit message passing paradigm where tasks communicate with each other by sending messages[3]. MPI is highly portable and implements an inherent shared memory model.

In this paper we will present our experience in the high performance computing field, and how this experience led us to adopt both approaches (Beowulf and openMosix) for our production environment, i.e. the Democritos center for atomic simulation within the INFN institute<sup>2</sup>. Democritos is hosted at the SISSA<sup>3</sup> and involving about 30 researchers belonging to different scientific institutes (among them ICTP<sup>4</sup> and the CINECA Supercomputing Center). The mission of the Democritos center is to carry on coordinated research at leading edge of computational material science. The achievement of this goal pose a strong demand of allocated computational resources able to cope with the variable and evolving researchers' demands.

These computational requirements can be defined as the facilities needed to perform a "computer experiment". This term can involve many different phases: developing from scratch the codes in an ad-hoc fashion, or finding appropriate ready-made applications; evaluating the best platform for execution, optimizing/parallelizing the softwares as well as collecting statistically meaningful results through repeated executions.

In a high performance production environment the following issues are of primary importance: (i)throughput, in order to guarantee efficient resource sharing among users and maximum exploitation of the computational resources; (ii)the so called "time to production", to reduce at minimum the time needed to implement a specific experiment: i.e. going from a serial application to a scalable and efficient parallel one.

The required applications used for our research are of a surprisingly heterogeneous nature; here we will present some of them, focusing on the very common goal of finding the best cluster paradigm to perform a specific, real computational experiment.

In the final analysis our benchmarking study suggests that different requirements can be better met adopting various kinds of cluster technologies, which happen to be related better to different classes of problems.

The paper is organized as follows: the next section offers a detailed overview of the currently available computational environment. Section 2 presents the details of the benchmarking study we performed on some particularly representative test cases. Section 3 reports and examines the results. Our conclusions are proposed in section 4.

## 2 Computational Environment

In this section we describe in some detail our scientific computational environment, considering two different aspects: the platforms (hardware and system software) and the scientific applications running on them.

---

<sup>1</sup>openMosix is an Open Source project adding extensions to the Linux kernel and creating a single-system-image cluster out of individual, network-connected boxes.

<sup>2</sup>INFN (Istituto Nazionale Fisica della Materia) is the Italian National Institute for condensed matter Physics.

<sup>3</sup>SISSA (Scuola Internazionale di Studi Superiori Avanzati) is an International school of advanced studies located at Trieste (Italy)

<sup>4</sup>ICTP is the Unesco International Center for Theoretical Physics located at Trieste

## 2.1 Cluster Specifications

The Democritos hardware computing resources have two different locations: SISSA (Trieste, Italy) and the CINECA (Bologna, Italy).

CINECA offers to our researchers a large fraction of the computational power of their massively parallel platforms (IBM SP3, IBM SP4, and SGI Origin3000). These resources are dedicated to very large parallel applications.

The on site production facility consists of two mid-range Linux clusters, available for serial and medium-size parallel applications. Both the machines have the same configuration; the only difference from an hardware point of view is the choice of two specific network interconnection technologies, consistent with the two different clustering paradigms adopted as described later.

For the purpose of this paper we also used a third production cluster installed at the department of physics at University of Pisa, Italy (referred as Hator). Beside these production clusters we also tested a prototype installation kindly made available by ICTP in Trieste.

This prototype cluster presented an interesting opportunity for a side by side comparison of two kinds of high end interconnect solutions especially designed to meet the HPC requirements for commodity hardware nodes: Myrinet and Dolphin. They both exploit advanced main memory mapping technologies (RDMA or Remote Direct Memory Access) which do not require CPU and OS intervention. Both technologies also feature optimized MPI interfaces to deliver real performances in the range of 2+2 Gbit/s for Myrinet 2000 cards, whereas Dolphin D33X series can achieve up to 1333MBytes/s full duplex <sup>5</sup>. The main difference from an advanced user up to system integrator point of view seems to derive from the larger market share and greater maturity enjoyed by the Myricom alternative, resulting in simpler and stabler drivers, bigger choice of integrated solutions vendors, and consequently lower end prices.

The technical aspects of all machines used are reported in the following table 1

Cluster	Briareo	Hokule	Hator	ICTP proto
<b>HARDWARE</b>				
processor	dual PIII 1.4GHz	dual PIII 1.4GHz	dual PIV 2.2GHz Xeon	dual PIV 2.4GHz Xeon
CPUS (total)	32	16	16	4
Cache (L2)	512 KB	512 KB	512 KB	512 KB
northbridge	ServerWorks CNB20LE	ServerWorks CNB20LE	Intel e7500	Intel e7500
DRAM (total)	32 GB	16 GB	8 GB	2 GB
PCI bus	33 MHz 64bit	33 MHz 64 bit	66 MHz 64 bit	66 MHz 64 bit
network	Myricom M3F-PCI64B	Intel Ethernet Pro 100	Myricom M3F-PCI64B	Myricom M3F-PCI64B /Dolphin D335 /Intel e1000
<b>SOFTWARE</b>				
Linux kernel	2.4.2	2.4.18	2.4.18	2.4.18
Patch/specific drivers	GPFS 1.2, GM 1.5.1	openMosix 2.4.18-2	openMosix 2.4.18-3, GM 1.5.1	Scali2.4.10beta6/ - /GM 1.5.1
Distribution	RedHat 7.1	RedHat 7.1	RedHat 7.3	RedHat 7.3
Comm. Library	MPICH 1.2.1-7b over GM	MPICH 1.2.4 over FE	MPICH 1.2.1-7b over GM	MPICH 1.2.1-7b over GM/ gigabit / scali

Table 1: Technical characteristics of the Linux clusters used

<sup>5</sup>cfr. <http://www.dolphinics.com/products/hardware/pci64.html>

## 2.2 Scientific simulation softwares

This section discusses the computational requirements posed by different classes of scientific codes.

We refer to “packages” describing a well structured set of programs, adopting standard interfaces and common libraries, usually resulting in a bigger, more complex system which can be readily used on different platforms and by scientists different from the original developers.

On the other hand a “program” is a quick, simple answer to a well defined, temporary scientific requirement. The programmer in this case is usually a single scientist, and development as well as production time remains very short.

IT aspects	Packages	Programs
Development time	long (years)	short (weeks/months)
Production life	long (years)	short
Number of users	large	small
Maintenance	difficult	absent
Cross site support	yes	local only
Portability	good	complex
Optimization	high	low

Table 2: A short comparison between packages and programs (see text)

Table 2 points out some of the IT aspects that characterize packages and programs used in our center. Locally developed packages implement team work, cross site cooperation and state-of-the-art techniques resulting in highly optimized, portable codes working on a large variety of scalar and parallel platforms. Some of them are also being freely distributed ( i.e. PWSCF [4] and DLPROTEIN [5]). Although this kind of software constitutes an invaluable asset, it still suffers from severe limitations: documentation is very poor, and its maintenance and training are difficult and time consuming.

“Programs” on the other hand must be seen only as temporary tools, created to provide a quick answer to very specific needs. The clear advantage is flexibility, whereas many other important aspects (portability/optimization) are not usually taken into account.

From the scientific point of view our simulation software can be briefly classified into three main classes:

- First-principles or ab-initio simulations where the quantum description of the electronic properties of matter is taken into account resulting in quite complex and computationally demanding simulations. To cope with this need, recent years have seen the development of parallel versions of electronic structure codes adopting the Message Passing (MP) paradigm. Due to this effort highly scalable packages are now available, offering a good speed-up over a wide range of architectures.
- Classical Molecular Dynamics (MD) simulation, which allows to simulate systems composed of several thousand of atoms treated using classical mechanic laws. The required techniques can be easily implemented as long as the simulated systems remain simple. Simulating complex system like for instance biological molecules (i.e. proteins) poses the need for structured parallel packages even if scalability is often limited to a small number of processors.
- Quantum Monte Carlo (QMC) simulations to study strongly correlated systems. This field is characterized by many different implementations of the basic theoretical idea; since an embarrassing parallelization strategy is also possible many different programs keep flourishing and evolving without the real need of a structured packages.

These different areas pose quite different computational requirements. Table 3 presents a comparison of the three families of codes focusing on High Performance Computing (HPC) specific

aspect	Ab-initio	Classical MD	Quantum MC
Communications	high	high	negligible
Memory requirements	high	moderate	moderate
Parallelization system	MPI	MPI/Shared Memory	MPI
Scalability	high	not scalable	linear
Linear Algebra kernel usage	high	almost null	moderate
Cache exploitation efficiency	good	bad	moderate

Table 3: Comparison between software codes under HPC loads.

aspects. The columns represent the three classes of software codes and each row identifies a peculiar HPC aspect. The need for different resources emerges quite clearly.

We finally note that local computational activities roughly involve two different classes of researchers: people only interested in scientific research can be defined as “users” of the computational facility. Planners, developers and maintainers of the information technology structure constitute the second class; this our belonging category as this paper authors.

This is of course only a convenience distinction: many so called “users” are literate in many sectors of the IT field and provide constant support and valuable help, and official IT responsible perform pure scientific research as well.

The driving idea in designing and administrating our computational resources is to provide the best environment for both packages and programs, taking advantage and further developing the knowledge, experience, and know-how inherited from previous developers and users.

### 3 The Various “Computer Experiments”

In this section we present in detail the specific benchmarking activity, describing which set of measurements were performed on the different clusters available to collect quantitative results for every computational problem mentioned earlier.

We decided to focus on well defined cases, selecting a small subset of the above defined “computer experiment” to be performed in the real as well as in the prototypical installations available. We tried to repeat the tests with different usage scenario and on the various cluster architectures / HPC paradigms.

The experiments are briefly defined in the following sections:

#### 3.1 Computer Experiment One: Classical MD Evaluation

This experiment is a typical task actually performed in our site on a daily basis: the evaluation of different solutions (here limiting the comparisons only to clustering architectures / networking interconnects / optimizing compilers), looking for the best platform for DLPROTEIN, a classical MD package dedicated to biophysical simulations. This parallel package is currently developed at Democritos and is also used as production code by the users. The importance of this experiment is twofold: it provides our users with detailed information about expected simulation time on the available platforms, and it also gives indication for future developing work.

The physical sample comes from a production system we already adopted in previous benchmarking works [6]. The sample represents a typical size problem in the area of biological MD simulations: memory requirement is low but it has cpu intensive demands due to the long simulation runs needed to collect significant results. In this experiment we performed just enough steps (100) of the time evolution of the system in order to correctly estimate the mean value for a single step.

#### 3.2 Computer Experiment Two: MD Repeated Simulations

This experiment reproduces a very common research scenario, where we are interested in evaluating how the change of a single parameter affects a set of otherwise identical simulations.

We measure the total time to results from a “user” perspective, defined as the time required to setup the code, to use it on a multi users real world facility, and collect the desired data. We are particularly interested in investigating drawbacks and benefits from different clustering paradigms.

In this specific example we perform a few different molecular dynamics simulations at different temperatures, adopting as sample data a reference system studied at 20 different temperatures simulated for 400 timesteps each. In this experiment we use the Amber package [8] which offers both serial and parallel implementation of the needed molecular dynamic code. The execution of the experiment for the two cluster paradigms is different: on the Beowulf one we had to setup and submit 20 different jobs using the installed PBS job queuing system. On the openMosix cluster we just proceeded to launch interactively the 20 different simulations simultaneously. It will be interesting to see how multiple parallel executions on Beowulf cluster compare with multiple serial run on the openMosix clusters.

### 3.3 Computer Experiment Three: Serial Monte Carlo Calculation

This experiment is performed using a single serial program, developed by a local research group: this code implements some advanced techniques to solve numerically theoretical models in superconductivity field and requires a quite specific Monte Carlo implementation. Scientific results are obtained just exploring a relatively large set of different parameters.

This scenario represents the ideal problem for the “fork and forget” approach available with an openMosix enabled kernel; our measurements aimed at exploring the network interconnect dependency of this clustering paradigm. The serial code is run repeatedly in order to collect significant statistical sampling sets. The independent copies, each one processing a different input data set, are just launched interactively with simple scripts.

### 3.4 Computer Experiment Four: Parallel Monte Carlo Calculation

This experiment uses the MILC software for quantum chromodynamics simulations. Even though this software comes from the high energy physics field and does not belong to our Condensed Matter suite, the Monte Carlo techniques implemented do not differ too much from the ones used in our Monte Carlo packages. We consider useful a direct comparison aimed at observing again the role of different cluster paradigms. In particular we investigate the role of the openMosix paradigm on a PentiumIV cluster.

The MILC package is a set of codes developed by the MIMD Lattice Computation (MILC) for simulations of 4-dimensional  $SU(3)$  lattice gauge theory.

The evaluated Staggered CONGRAD 5 code from MILC is a FPU intensive algorithm here configured to use 4-dimensional lattices of varying sizes spreading the calculation across the cluster nodes via MPI.

## 4 The “Computational Experiments” Results

In this section we present the result obtained for every computer experiment presented in the previous section.

### 4.1 Network Performance Results

Since latency and bandwidth of a particular network interconnect are of fundamental importance for clustering, we conducted a basic evaluation of the impact on performance of each solution. The measurements was done by means of the standard benchmark suite freely available from Pallas[14].

Table 4 reports the different values of latency and bandwidth measured over the MPI protocol. All possible combinations of hardwares are included here.

Latency is expressed in microseconds and it is defined as the time needed to send zero length packets in a simple ping-pong test (half duplex)

The MPI bandwidth is estimated as the average of the last six values of the ping-pong test (message size goes from 128KB to 4MB).

System	Network hardware	MPI library	Latency (us)	Bandwidth (MB/s)
ICTP proto	Dolphin	MPI scal1.13.8.b9	5	235
ICTP proto	Myrinet	MPICH/GM	11	224
ICTP proto	Gigabit	MPICH	91	36
ICTP proto	FastEthernet	MPICH	79	11
Briareo	Myrinet	MPICH/GM	15	155
Hokule	FastEthernet	MPICH	87	10

Table 4: Performance of standard MPI communications on the available clusters. The error is estimated to be around 2%

Dolphin and Myrinet network solutions deliver almost identical bandwidth on the small ICTP prototype, but Dolphin measured latency is half of the Myrinet one. For our purpose it's interesting to note the Myrinet cards increase the bandwidth as the PCI bus frequency is doubled (from 33MHz to 66MHz).

Our experience has shown that the Dolphin interconnect system is not yet completely independent on the software support version: upgrading the Scali drivers from the default shipped release to the last beta available resulted in almost doubling the performances.

## 4.2 Computer Experiment One

In this section we present the detailed results obtained in the previously described benchmarking activity with the DLPROTEIN package. We collected many data for different conditions; table 5 is a summary of all the results. The benchmark runs were repeated at least twice; the deviation is estimated to be less than 5%. In the table the first column lists the various combination of CPU / interconnection hardware / compiler and optimization flags tested during the experiment. The other columns report the total execution times as well as the time spent in communication routines (numbers within brackets).

The results lead to some interesting observations:

- *Compilers* We tested two different compilers: the PGI suite [13] and the Intel one [12]. The PGI compiler (with the suggested standard optimization flag `-fast`) delivers practically the same results of the Intel compiler without any optimization in the serial case. Performances are slightly increased adopting IFC for the parallel version.

On the PentiumIV architecture the speed up is more significant: Intel compiler on this CPU is able to boost performances very significantly when compared to the PGI product, especially for runs with two UP nodes (-20%). Considering that the Portland Group has been recently acquired by Intel, such differences can be expected to disappear over time.

- *Interconnect* As indicated next to the results, we ran some of the tests using only one CPU per computing node in order to simulate UP processing elements, thus leaving the full PCI bandwidth available to CPU<->network card data transfers. On our Briareo cluster this led to nearly halved communication times, with direct benefits for the final benchmark results; we can see this as a consequence of the 33MHz PCI bus being a real bottleneck for high-bandwidth, low-latency interconnects. The problem appears to be less evident using a good optimizing compiler like IFC.

This conclusion is confirmed by results collected on the ICTP prototype cluster: since in the SMP run the MPI communication times on this platform are roughly equal to the UP

<sup>8</sup>We used for all pgi compilations the standard `-fast` flag

<sup>8</sup>Optimization used for PIII: `-pad -O3 -tpp6`

<sup>8</sup>Optimization used for PIV: `-pad -O3 -tpp7`

CPUs	1	2		4		8	
		UP	SMP	UP	SMP	UP	SMP
Briareo (pgi <sup>6</sup> )	699	394 (23)	456 (49)	229 (33)	282 (63)	152 (45)	196 (76)
Briareo (ifc)	700	394 (29)	445 (36)	227 (37)	265 (47)	151 (48)	182 (58)
Briareo (ifc optimized <sup>7</sup> )	674	381 (22)	431 (33)	222 (33)	261 (46)	148 (45)	177 (57)
ICTP proto + Myrinet (ifc optimized <sup>8</sup> )	550	291 (24)	-	-	195 (37)	-	255 (82)
ICTP proto + Dolphin (ifc optimized)	561	318 (17)	-	-	194 (23)	-	267 (63)
ICTP proto + Dolphin (ifc)	620	385 (14)	-	-	229 (22)	-	-
ICTP proto + Dolphin (pgi)	686	367 (17)	-	-	219 (27)	-	296 (76)

Table 5: Scalability of the DLPROTEIN software for each cluster used. Times are in seconds. Numbers in brackets represent communication times.

communication times measured on PIII, we can conclude that the availability of a faster PCI bus solves the above problem.

We also noticed a different communication behavior between the two different interconnect solutions. Going from 2 UP to 2 SMP processing elements the Dolphin cards were able to deliver shorter wait times when compared to Myrinet. (only +35%, whereas the latter showed +54%). We attribute this result to the lower Dolphin latency, since the communication pattern of this code shows frequent synchronization points.[7]. Further investigation with this package is difficult because of its inherent scalability deficiencies beyond the reported number of nodes.

- *CPU* The increase in performance moving from PentiumIII at 1400 MHz to PentiumIV at 2200 MHz is disappointing: required time reduced only by -35%. Basing on results collected with this software we conclude that at the moment such an upgrade is not justified. Even the new advanced features introduced (like SSE2 SIMD instructions or HyperThreading (HT) [9] support) were not helpful in this case. IFC was not able to compile the package when explicitly asked to generate PIV specific code (-axW flag), while experiments executed without HT enabled not surprisingly showed better results.

### 4.3 Computer Experiment Two

We present here the results of the second computer experiment, performed with the Amber software package. We chose a standard Amber benchmark as a simulation system, studying the system behavior under different temperatures. This problem required  $n$  independent instances of the code, each sharing the same initial elements arrangement but with a different temperature setting.

This package offers both serial and parallel versions. The parallel replicated data algorithm of this software is not scalable [10], but previous benchmarking studies [7] clearly suggest that despite the inherently poor scalability of the algorithm a multi CPU platform in the range of 2/8 processors can offer significant productivity.

The main challenge is to find the trade-off between parallel runs and serial executions. Therefore we performed our task twice: using 20 different serial instances on the openMosix cluster and submitting 20 different jobs using the parallel version of the program. We then measured

the total time for the two experiments (defined as the total time elapsed from the first run start to the last execution/job end) and the average time for each of the 20 simulations. We repeated these procedures both for the PentiumIII based clusters (Briareo and Hokule) and for the PentiumIV one (Hator with Myrinet). The following table collects all the results. All the experiments (on both the openMosix and Beowulf machines) were performed with an identical number of dedicated processors (8 nodes, 16 processors): the clusters were otherwise idle.

System: Hokule	Beowulf (4 CPUs)	Beowulf (8 CPUs)	openMosix
Total execution time	1013	1138	894
average simulation time	193.4	142.5	687.5
System: Hator	Beowulf (4 CPU UP)	Beowulf (4 CPU SMP)	openMosix
Total execution time	1432	2082	1066
average simulation time	139.4	206.3	844

Table 6: Results for experiment two; times are in seconds

The disappointing performance of the high frequency (2.2GHz) PentiumIV CPUs are related to the lack of an optimizing compiler beside GNU g77 for this test, while on the PentiumIII clusters we were able to use the clearly superior PGI compiler.

As a final result we notice that openMosix technology is more efficient than the Beowulf one for this kind of tasks. Even though the parallel execution benefits from a superior Myrinet network, the total execution time for all our parallel experiments is still higher and the gap tends to increase as the number of processors becomes larger.

We repeated the openMosix experiment twice on the PentiumIII cluster, in order to have a rough estimate of the overhead introduced by process migration: it turns out to account for about 30% of each simulation average length. In all likelihood, for very long runs this overhead will tend to decrease, since openMosix has an inverted age/priority migrating scheduler which prevents long-running jobs from migrating all too often. Newer versions of openMosix (2.4.19-2 and later) go even further by introducing the concept of “goodness” of a potential migration candidate, and evaluating criteria derived from process age and I/O behavior to compute a heuristic goodness factor

Results collected from benchmarks on Hator require some additional observations due to observed role played by HyperThreading technology. This feature virtually doubles the number of CPUs visible to the Linux kernel<sup>9</sup>. Linux’s support for Hyperthreading is not yet satisfactory; we observed many times the kernel trying to assign our tasks to a virtual CPU leaving a real one idle and thus degrading the overall performance. To avoid this behavior we ran 40 instances of our simulation for the openMosix case, in order to be sure that all the CPUs were in use so that the openMosix migration mechanism was allowed to work under a “fair“ context.

The same problems were observed for parallel SMP benchmarks as well. A similar precaution was taken in those cases: we just used a single (real!) CPU on each SMP machine. Results indicate again the openMosix technology to be superior.

#### 4.4 Computer Experiment Three

Firstly, we interactively launched 13 instances of the program. Then we repeated the runs using a larger number of instances ( 40): our scope was evaluating the behaviour of the openMosix load balancing algorithm under various realistic conditions.

These simple experiments were run on both the PentiumIII and PentiumIV clusters, with various interconnects in order to collect also measurements about the role of the networks. The openMosix kernel patch was temporary installed on the PentiumIII system with Myrinet interconnection (Briareo), and the experiment was executed only once. On the other hand we were able to repeat the simulations several times on the Hokule cluster. Numbers in table refers

<sup>9</sup>In Linux threads have their own distinctive task structure, and are therefore visible from within the kernel. They just lack their own address space, contrary to normal processes

to averaged values over different runs: maximum (minimum) number of migration refers in this case to the maximum (minimum) within the set of identical repetitions.

The mean and the variance of the elapsed time of all the instances (measured by the program) are reported as well, together with number of process migrations, also including the maximum and the minimum number and some descriptive statistic.

Parameters	Briareo	Hokule	Hator (myrinet)	Hator(Ethernet)
<time>	2970	2734	1499	1861
$\sigma(time)$	36	110	139	90
<Nmigs>	26.3	3.7	14.1	1.8
$\sigma(Nmigs)$	2.9	0.5	0.43	0.22
max/min Nmigs	34/23	0/15	11/17	1/4

Table 7: Result for experiment three with low load; times are in seconds

Table 7 reports the measures referring to an underload situation (13 jobs on 16 CPUs). This low-load scenario shows a weakness in openMosix migration cost evaluation on high speed interconnects: the continuous effort to evenly balance the load led to a higher number of migrations, since their cost was probably underestimated due to the low latency and large buffers provided by Myrinet cards. This did not happen using Fast Ethernet NIC, since the higher migration cost was correctly taken into account.

Again we see the PentiumIV platform appears to apparently benefit substantially from openMosix, especially if combined with Myrinet cards. The faster migration plays a crucial role in lowering the bad side effects of Linux' poor implementation of HyperThreading support. Under low load and with HyperThreading enabled the suspiciously high  $\sigma$  does not reflect an resource distribution inefficiency: the mean execution time of the instances is still shorter than with the Ethernet, i.e. 25% faster.

Parameters	Briareo	Hokule	Hator (Myrinet)	Hator (Ethernet)
<time>	3659	3234	2757	2823
$\sigma(time)$	10	49	5	5.5
<Nmigs>	31.9	28.5	9.22	15.63
$\sigma(Nmigs)$	2.33	1.54	0.53	0.54
max/min Nmigs	12/39	18/36	2/15	10/22

Table 8: Results for experiment 3 with high load; times are in seconds

Results from table 8 suggest that heavier load leads to more migrations, especially so for Ethernet network. The system is now able to get a more efficient load balancing as simulation time range narrows. In fact,  $\sigma$  goes from 36 to 10 for Myrinet and from 110 to 49 for Ethernet. Under both load conditions the average run time on Briareo is about 10% higher than on Hokule. The openMosix developers are working towards providing native, kernel-level integration of drivers for these fast interconnects, thus eliminating unnecessary performance degradation.

The same behavior is noticeable on PentiumIV platform. Under heavy load, i.e. 40 instances for 16 real CPUs, more migrations happen with optimal load balancing results; however performance degrades as more instances are added to the virtual processors provided by HyperThreading. Here the role of the Myrinet network in minimizing this problem is greatly reduced: the gap between Ethernet and Myrinet equipped systems shrinks to 2%.

#### 4.5 Computer Experiment Four

This experiment was performed using the MultiNode configuration of the Staggered CONGRAD5 component of the MILC code. The source code was compiled using the GNU gcc 3.1 [11] compiler with some optimizations for the Pentium Xeon platform; the code uses MPI with GM Libraries

over Myrinet. We have varied the lattice dimensions (4x4x4x4, 8x8x8x8, 12x12x12x12) running 20 tests on the Hator cluster with and without openMosix enabled, always using 16 CPUs in a SMP with HT configuration. In table 9 we present the results.

Lattice Parameter	openMosix	Beowulf
4x4x4x4	5.3	5.6
8x8x8x8	17.2	17.6
12x12x12x12	101	126

Table 9: Results for experiment Four: times are in seconds

This parallel program doesn't migrate in the openMosix environment because of the usage of shared-memory: since openMosix does not yet implement a distributed shared memory mechanism these processes cannot be migrated. It appears that for small dimensions of lattices openMosix makes no difference. But as the lattice dimensions grow, we can observe a speed-up in the execution time, which might be attributed to a better load distribution within the node of other competing, migratable system processes, thus lowering the effects of misplaced resources due to the naive view of the HyperThreading mechanism. Further investigations on this subject are still been carried on.

## 5 Discussion

Several issues came up during the benchmarking for this paper:

- Many parallel programs in their actual form cannot migrate on openMosix architecture due to their use of shared memory segments for inter-process communication. However the openMosix dynamic load balancer can sometime still benefit the application, because it will simply migrate away other processes, thus leaving more CPU power and memory for the parallel code.
- The openMosix solution offers great advantages for many serial computational problems. The automatically conversion from serial to parallel implementations of the algorithms through the process migration technology can significantly increase the overall throughput of the whole facility.
- High bandwidth clearly improve MPI-based applications and certain classes of programs. However it is less important for an openMosix platforms dedicated to serial jobs because of its demand-paging address space migration. On the other hand, openMosix is far more dependent on low latency because of the inter-node load balance status gossiping algorithms[15].

It is clear that a single Linux cluster solution cannot fit all our computational requirements. Providing each single experiment with the most productive execution environment requires some preliminary evaluation and planning, often resulting in creating custom solutions each time. This is clearly not possible for our institute; nevertheless we showed that the two clustering paradigms presented allow us to install computational resources able to cope quite well with some medium-size advanced physics computational demands.

## 6 Acknowledgments

We want to thanks Dr. A.Nobile of ICTP for providing us some of the hardware used for the pourpose of this paper.

## References

- [1] The project's home page is at <http://openmosix.sourceforge.net/>
- [2] Linux Internals, chapter 3, Moshe Bar, McGraw-Hill-Hill, New York (2000)
- [3] High Performance Cluster Computing, Volume 2, Rajkumar Buyya, Prentice Hall New Jersey (1999)
- [4] Baroni, A. Dal Corso, S. de Gironcoli, and P. Giannozzi, <http://www.pwscf.org>.
- [5] Melchionna,S.,Cozzini,S. DLPROTEIN\_2.1 User Guide (<http://www.sissa.it/cm/DLPROTEIN>)
- [6] S. Cozzini "Benchmarking production codes on Linux Cluster: the Sissa case study" Proc. of Linux Clusters :the HPC Revolution 2001, Urbana, june 2001
- [7] Cozzini, S., Innocente, R. & Corbato, M. Comparing scientific code on different parallel platforms, Proc. of 6th SGI-MPP workshop: Manchester 2000
- [8] <http://www.amber.ucsf.edu/amber/amber.html>
- [9] Marr, D.; Binns, F.; Hill, D.; Hinton, G.; Koufaty, D.; Miller, J.; Upton, M. "Hyper-Threading Technology Architecture and Microarchitecture: A Hypertext History." Intel Technology Journal. <http://developer.intel.com/technology/itj/2002/volume06issue01/> (Feb 2002).
- [10] S. Plimpton "Fast Parallel Algorithms for Short Range Molecular Dynamics" ,Journal of Comp. Physics, 117(1995)
- [11] <http://www.gnu.org/software/gcc/gcc.html>
- [12] <http://www.intel.com/software/products/compilers/>
- [13] <http://www.pgggroup.com>
- [14] <http://www.pallas.com/pages/pmbd.htm>
- [15] openMosix Internals, paper presented at the Linux Kongress in Germany, September 2002, Moshe Bar