The Impact of virtualization on grid parallel Molecular Dynamics simulations

Lino García Tarrés[1](*), Elvira Guàrdia[2], Joaquim Casulleras[2]

[1] Fundación Centro Tecnolóxico de Supercomputación de Galicia (CESGA). Santiago de Compostela, Spain.
[2] Dep. de Física i Enginyeria Nuclear. Universitat Politècnica de Catalunya. Barcelona, Spain.
(*) To whom the correspondence should be addressed: lino.garcia@upc.edu

Abstract

The influence of the Xen virtualization technology on a parallel Molecular Dynamics simulation algorithm has been studied by the use of the Interactive European Grid infrastructure. Two parallelization strategies have been implemented; the first one is based on the Message Passing Interface (MPI) standard while the second one uses the low level shared memory (SHM) UNIX system V communication capabilities between processors. In order to compare the benefits of these methodologies and technologies, four sets of pure water physical systems have been simulated.

1 Introduction

During the last years a great effort has been carried out improving the Molecular Dynamics simulation algorithms. The recently available computational platforms based on a highly coupled network of processors receive the generic name of "symmetric multi processing" SMP, in which two or more CPUs or cores are connected to a shared main RAM memory. Such architectures offer a good multiprocessor resource that suits particularly well to the needs of these simulations.

In this sense is worthwhile to highlight the following:

- J. Casulleras and E. Guàrdia implemented in 1991 a parallel version of their research team Molecular Dynamics algorithm by using transputer based architectures[1].
- In the year 1994 E. Watson and T. Lybrand[2] had adapted the molecular mechanics AMBER package to a cluster of workstations using "Parallel Virtual Machine" or PVM[3].
- In 1995 H.J.C. Berendsen[4] and collaborators had also implemented their Molecular Dynamics GROMACS application by the use of the "message passing" methodologies PVM and Message Passing Interface[5] (MPI).
- Mark Nelson and collaborators[6] had ported their NAMD package to the Charm++ Parallel Object-Oriented platform.
- In a previous work[7], one of the authors extended the job done by J. Casulleras by the use of PVM and shared memory.

The Molecular Dynamics simulations are particularly well suited applications to run on he grid middleware[8]. Such infrastructure allows the scientist to use remote grid resources[9] in a transparent and powerful way. Actually, the Interactive European Grid[10],[11],[12] project (IntEuGrid) is working on the development of a new MPI infrastructure for grid parallel applications.

The recently available virtualization tools are becoming a useful technique which allows achieving better resource utilization. These new technologies give a virtual hardware interface on top of which multiple operating systems instances can run concurrently. In particular, the Xen Virtual Machine Monitor[13],[14] takes a further step allowing to dynamically hot-plugging processors from one operating system to another. Consequently, on a grid site is convenient to group the working nodes systems on the same management privileged Xen domain0; by doing so, the Xen CPU scheduler could efficiently balance the load between different working nodes by giving proportional fair shares of the CPU to virtual systems.

In this paper, two methodologies have been implemented to improve the parallelization benefits on molecular dynamics simulations in grid computers: the Message Passing Interface[2] (MPI) and the shared memory and semaphores[1], [2] inter process communications[15] tools (SHM). Both strategies have been deployed and tested on native and virtualised grid working node systems.

2 Physical system

The simulated physical systems are set up by a group of water molecules contained in a cubic box subjected to periodic contour conditions at ambient temperature. To analyze the efficiency of the virtualization and parallelization method in front of the size of the system, several Molecular Dynamics simulations have been carried out with N_{mol} = 216, 512, 1000 and 10000 water molecules. The cube side length was adjusted to a density of 1.00 g/cm3, and the reference temperature was T_{ref}= 298 K. The water molecules have been modelled by using the "extended single point charge" SPC/E interaction potential proposed by Berendsen[16]. It's a water rigid model in which the three interaction centres are located in Oxygen and Hydrogen atom positions. The bond distances are r (O-H) = 1Å and the angle H-O-H is fixed to 109.5^o. The charge assignments corresponds to a water dipolar moment μ= 2.35 D.

For such a water model, the interaction energy between two molecules "a" and "b" is determined by summing up the site to site intermolecular interactions according to the expression:

$$V_{ab} = \sum_{i}^{a} \sum_{j}^{b} \left(\frac{A_i A_j}{r_{ij}^{12}} - \frac{C_i C_j}{r_{ij}^{6}} + \frac{q_i q_j}{r_{ij}} \right)$$

The "short range" part coefficients can be expressed in terms of the Lennard-Jones parameters $A_i^2 = 4\varepsilon_i \sigma_i^{12}$ and $C_i^2 = 4\varepsilon_i \sigma_i^6$ where q_i is the electric charge assigned to the "i" site. The Lennard-Jones parameters as well as the model's partial charges are summarized in table 1.

Table 1. Interaction potentials parameters.

Atom	ε (Kcal/mol)	σ (Å)	q (e)
O	0.1553	3.166	-0.8476
H_w	0.0	0.0	0.4238

To carry out the simulations, we use the integration algorithm proposed by Berendsen and co-workers[17]. The time step was 2 fempto seconds and the whole system were placed into a thermal bath with a coupling constant of 0.2 ps. The bond length and angles remain fixed by using the shake[18] method.

In order to compute the long-range coulombic forces, the Ewald[19] summation method was used.

Under these premises, the system's total energy is given by the expression:

$$\mathrm{U} = \frac{1}{2}\sum_{i,j=1}^{N}\left[q_i q_j \frac{erfc(\alpha r_{ij})}{r_{ij}} + \frac{A_{ij}}{r_{ij}^{12}} - \frac{C_{ij}}{r_{ij}^{6}}\right] - \frac{1}{2}\sum_{a=1}^{N_{mol}}\sum_{\substack{i,j=1\\ i\neq j}}^{3}\left[q_i^a q_j^a \frac{erfc(\alpha r_{ij})}{r_{ij}}\right]$$
$$+\frac{1}{2\pi L}\sum_{h\neq 0}\frac{\exp\left(-\frac{\pi^2\left|\overrightarrow{h}\right|^2}{\alpha^2 L^2}\right)}{\left|\overrightarrow{h}\right|^2}\left\{\left[\sum_{i=1}^{N} q_i \cos\left(\frac{2\pi}{L}\overrightarrow{h}\cdot\overrightarrow{r_i}\right)\right]^2 + \left[\sum_{i=1}^{N} q_i \sin\left(\frac{2\pi}{L}\overrightarrow{h}\cdot\overrightarrow{r_i}\right)\right]^2\right\}$$
$$-\frac{\alpha}{\sqrt{\pi}}\sum_{i=1}^{N} q_i^2 - \frac{2\pi N\mu^2}{L^3}$$

3 Parallelization strategy

The main objective is to reduce the time that these simulations last. Habitually it is done by distributing and executing the same program in n_p processors concurrently. The implemented parallelization strategy is the well-known data replication method[20] in which each processor has an identical copy of the working data, but it performs the computation over a smaller and exclusive working area. These areas are subsets of the whole work to carry out, so the total execution time is reduced. In the best case, the total simulation time will drop by an n_p factor.

After each calculation step, and before the processors require the new computed data, the results have to be transferred to all processors. This stage adds an extra cost to the total computation, which hinders reaching the maximum performance.

To carry out an optimal task distribution among processors, is necessary to know the relative cost of all the computational stages involved. A typical Molecular Dynamics simulation run spends 80% of his total computing time evaluating forces and 18% determining the atom's new positions and speeds, so, to get the best performance, the parallelization of both the computational job

related to the determination of the inter atomic force calculations as well as the new positions and speeds will be necessary.

3.1 Forces computation

The total system energy equation states that the force $\overrightarrow{F_i}$ exerted by the whole system over the atom "i" can be expressed as the sum of two contributions:

$$\overrightarrow{F_i} = \overrightarrow{F_i^{rel}} + \overrightarrow{F_i^{abs}}$$

Where $\overrightarrow{F_i^{rel}}$ and $\overrightarrow{F_i^{abs}}$ are the real and reciprocal space contributions respectively:

$$\overrightarrow{F_i^{rel}} = \sum_{\substack{j=1 \\ j \neq i}}^{N} \left[q_i q_j \left\{ \frac{erfc(\alpha r_{ij})}{r_{ij}^2} + \frac{2\alpha}{\sqrt{\pi}} \frac{\exp\left(-\alpha^2 r_{ij}^2\right)}{r_{ij}} \right\} + \frac{12A_{ij}}{r_{ij}^{13}} - \frac{6C_{ij}}{r_{ij}^7} \right] \frac{\overrightarrow{r_{ij}}}{r_{ij}}$$

$$\overrightarrow{F_i^{abs}} =$$

$$\frac{2q_i}{L^2} \sum_{h \neq 0} \frac{\overrightarrow{h}}{\left|\overrightarrow{h}\right|^2} \exp\left(-\frac{\pi^2 \left|\overrightarrow{h}\right|^2}{\alpha^2 L^2} \right) \left\{ c\left(\overrightarrow{h}\right) \sin\left(\frac{2\pi}{L} \overrightarrow{h} \cdot \overrightarrow{r_i}\right) - s\left(\overrightarrow{h}\right) \cos\left(\frac{2\pi}{L} \overrightarrow{h} \cdot \overrightarrow{r_i}\right) \right\}$$

and

$$c\left(\overrightarrow{h}\right) = \sum_{j=1}^{N} q_j \cos\left(\frac{2\pi}{L} \overrightarrow{h} \cdot \overrightarrow{r_j}\right)$$

$$s\left(\overrightarrow{h}\right) = \sum_{j=1}^{N} q_j \sin\left(\frac{2\pi}{L} \overrightarrow{h} \cdot \overrightarrow{r_j}\right)$$

N is the total systems atom number (for pure water $N = 3N_{mol}$). $\overrightarrow{F_i^{rel}}$ Depends on the relative distance between atoms r_{ij}, but, on the other side, $\overrightarrow{F_i^{abs}}$ depends on the atomic absolute position r_i. The real space sum is truncated by applying a spherical "cut-off" of $L/2$.We have taken $\alpha = 5L$ and the sum over the reciprocal space has been restricted until $\left|\overrightarrow{h}\right|_{Max} = 16$, which corresponds to 128 vectors. According to our experience, these values give a satisfactory convergence for both the real and reciprocal space terms.

The $\overrightarrow{F_i^{rel}}$computational cost is proportional to the square of the number of atom pairs N^2. So, the parallelization strategy is carried out by distributing a different atom pair subset to each processor. Figure 1 shows a particular example for a small system to make it easier to understand. The left array represents the N^2 inter atomic interactions, but only those labelled in colour have to be calculated. The processors stores the partial forces results in the vectors showed at the right of the figure; so each processor computes $1/n_p$ of the total *N (N-1)/2* inter atomic interactions.

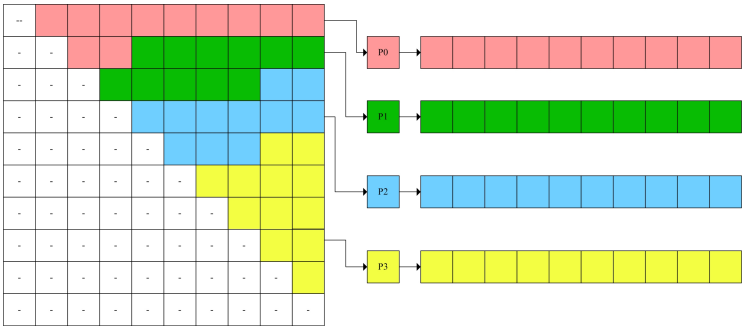

Fig. 1. Inter atomic forces parallelization strategy. N = 10 atoms and n_p= 4 processors labelled P0, P1, P2 y P3.

Regarding $\overrightarrow{F_i^{abs}}$, the computational cost is proportional to the product of the amount of reciprocal space vector used (N_h) by the number of atoms simulated. The parallelization is carried out by distributing subsets of the reciprocal space vectors among processors; each processor computes a subset ofN_h/n_p vectors.

At the end of the forces calculation stage, all the processors communicate their partial atomic forces to each other processor.

3.2 Atomic position and speed computation

The integration algorithm selected is the Verlet "leap-frog" proposed by Berendsen and co-workers[17]. Beginning with the particles positions $\overrightarrow{r_i}(t)$ and velocities $\overrightarrow{v_i}(t - \Delta t/2)$ the algorithm proceeds as follows:

1. Calculation of the new speeds (before bond restrictions are applied):
$$\overrightarrow{v_i}(t + \Delta t/2) = \lambda\left(\overrightarrow{v_i}(t - \Delta t/2) + \overrightarrow{a_i}(t)\,\Delta t\right)$$
The scale factor λ is:
$$\lambda = \left\{1 + \frac{\Delta t}{\tau}\left(\frac{T_{ref}}{T(t-\Delta t/2)} - 1\right)\right\}^{1/2}$$
Where $T_{ref}$is the thermal bath reference temperature and $\tau$is the bath-coupling constant.
2. New positions computation:
$$\overrightarrow{r_i}(t + \Delta t) = \overrightarrow{r_i}(t) + \overrightarrow{v_i}(t + \Delta t/2)/\Delta t$$
3. Application of geometrical restrictions to each molecule by the "shake" algorithm.
4. New atomic speed calculation (after molecular bond restrictions):
$$\overrightarrow{v_i}(t + \Delta t/2) = (\overrightarrow{r_i}(t + \Delta t) - \overrightarrow{r_i}(t))/\Delta t$$

El total cost is proportional to the number of atoms N. The parallelization is carried out distributing subsets of N/n_p atoms to each processor.

At the end of the new positions calculation stage, all the processors communicate their recently evaluated new atomic positions to all other processors.

4 Inter process communication methods

After each computational stage the new values are stored locally to each processor, so they have to be transferred to the other ones. To carry out this communication stage two methodologies have been used, one based on the Message Passing Interface (MPI) standard and the other by using shared memory primitives (SHM).

4.1 MPI

The method based on MPI standard carries out the transfer of the partial results stored on each MPI process by using the "MPI_Allreduce" SUM operation. This operation computes the addition of the vector elements distributed on a group of processes and returns its result into another local vector owned by each processor. Finally each processor has his results added up to the results locally computed by the other processors.

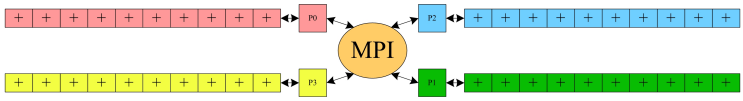

Fig. 2. MPI operation "MPI_Allreduce" for N=10 atoms and n_p=4 processors.

4.2 SHM

To develop an alternative method to the one based on MPI standard, we have used the following UNIX System V[15] inter-process communication mechanisms[21],[22] :

- Shared Memory. This system calls allows the processes to share their own data addresses space with other processes. It is the quickest way of transferring information among them.
- Semaphores. Semaphores are a programming construct designed by E. W. Dijkstra[23]. For the present purpose, these mechanisms allows a set of processes to be synchronized in order to avoid a process reading the shared memory while another is still writing on the same area.

This method offers the following advantages regarding the one based on MPI:

- Avoids the use of "daemon" processes which control the messages transmission between processors.
- Avoids the repeated copy of memory blocks necessary to send messages.

And as drawbacks:

– The MPI based method is more portable.
– SHM is only efficient in computers with shared memory architecture.
– Specific communication routines are necessary to be implemented. MPI standard has high-level build in routines allowing the programmer to operate directly on vectors and arrays.

The SHM method has better access to lower level routines than MPI, so bigger control over the way of adding-up the partial results is available. The SHM method that offers better results consists in the three stages exposed in figure 3. In this figure the square symbols represent processors (with labels P0...P3), and the rectangular ones represent the vectors placed in shared memory. The arrows show the flow of information. Each stage carries out the following operations:

1. The local vector is copied into the shared space. Then, all the processors share n_p vectors. The job is carried out in parallel, and all the processors copy the data concurrently in time. The cost is proportional to the number of atoms N (the force's and position's vector size are proportional to the number of atoms on the system).
2. Each processor adds a subset of elements from each shared vectors and puts the total value into a shared result vector (rectangular symbol located at the right of the processors). The summation is carried out therefore in parallel. The cost is proportional to N/n_p.
3. Finally each processor copies the result vector from the shared space to the local vector. The process is carried out in parallel; all the processors perform the copy process concurrently in time. Cost proportional to N.

The total cost is consequently proportional to the number of atoms and independent of the number of processors. So, there is no penalty because the use of a large amount of processors. The SHM method expends only one extra array of memory, the one which stores the shared results, in relation to the MPI_Allreduce operation.

5 Computational grid resources and Virtualization technology

The IntEuGrid test bed architecture relies on a CrossBroker particularly enhanced to manage parallel and interactive jobs[24], and on MPI-Start, a set of scripts developed[25] to support a wide range of MPI implementations, schedulers and file distribution methods (see figure 4). Its middleware is based on glite 3.0.2.

The virtualization tool is Xen 3.0.3 using the "earliest deadline first" default scheduler. The privileged Xen domain0 kernel is version 2.6.18-1 with SMP and both the native and virtual operating systems are Scientific Linux release 3.0.8 with kernel 2.4.21-20.

All the native and virtual working nodes run on a Dell PowerEdge 1955 system with two QuadCore Intel® Xeon® E5310 processors at 1.60GHz frequency

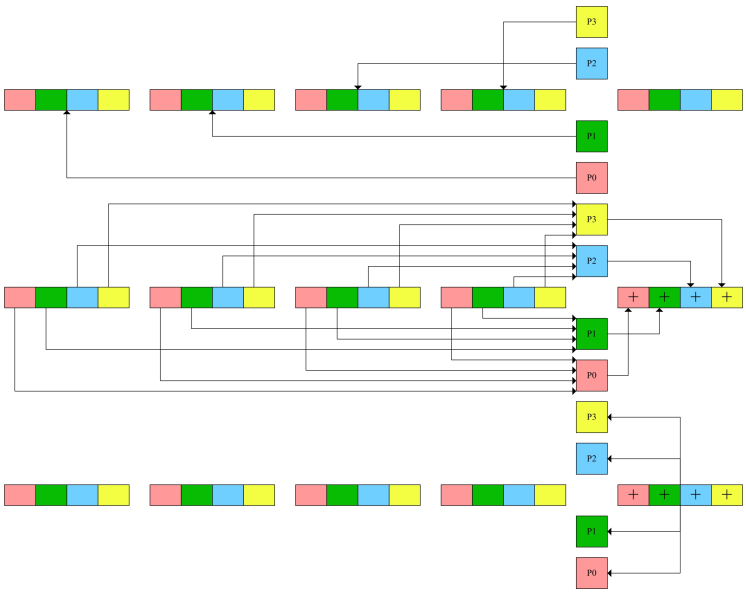

Fig. 3. Global operations on vectors by using shared memory (n_p=4 processors). Each processor places its data in SHM (top). Each processor adds the data from SHM to SHM (centre). Each processor obtains the data added from SHM (bottom).

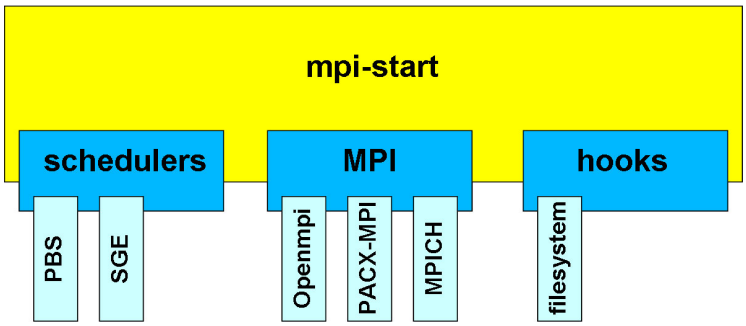

Fig. 4. MPI-Start architecture.

with 4 MB of L2 cache per processor and 4 GB of global main memory. The code is developed in C and FORTRAN and was compiled with gcc/g77 version 3.2.3 with openMPI version 1.1.2 configured to run on top of TCP-IP stack.

Four sets of simulations have been performed:

- For the SHM methodology, the grid jobs where executed in a single working node with SMP kernel. Two set cases where analysed, the native (or real) operating system and the Xen virtual O.S.
- Regarding the MPI methodology, two sets of jobs were submitted, one was running in a single eight core working node with native operating system and SMP kernel, while the other job set was submitted to another eight core working node installed over a Xen virtual O.S. with SMP kernel.

Each one of these four sets include simulations for three physical systems with $N_{mol} = 216$, 512, 1000 water molecules and 1000 time steps and a fourth with $N_{mol} = 10000$ and 100 time steps to give results in the similar time scale as the smaller systems. Each simulation where executed for a number of processors ranging from 1 to 8.

6 Results and discussion

Table 2 summarises the total execution time measured for each simulation run. The values decrease as more processors are used; also, for a given number of processors, all the methods give comparable time values. As an example of such behaviour, figure 5 shows how the total execution time changes as the number of processors increases for the SHM method on a native operating system.

Table 2. Simulation times for 1000 water molecules and 1000 time steps.

CPUs	SHM Native	SHM Virtual	MPI Native	MPI Virtual
1	766.82	784.13	784.64	783.19
2	384.33	397.59	391.00	393.50
3	256.56	261.99	261.00	262.14
4	193.05	199.78	196.54	203.88
5	154.83	158.04	161.12	158.56
6	129.59	132.20	132.73	133.96
7	111.57	113.29	119.37	115.54
8	97.71	98.49	100.46	100.79

The efficiency of a parallel algorithm is given by the expression:

$$Eff(n_p) = \frac{T_1/T_{n_p}}{n_p}$$

Where T_1 and T_{n_p} are the execution times for 1 and n_p processors respectively. The speed-up is then defined as follows:

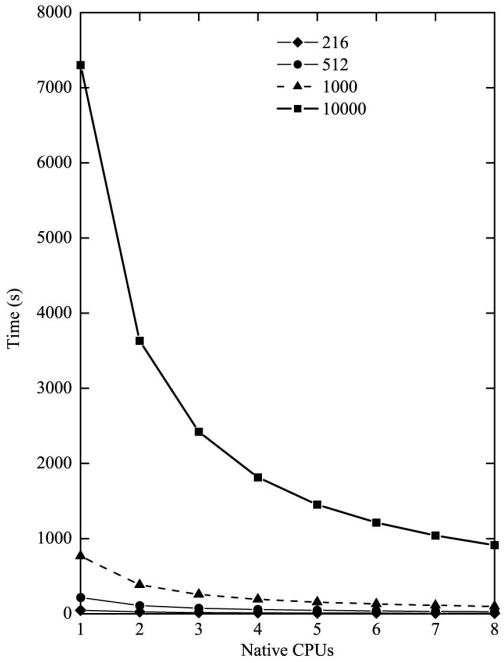

Fig. 5. Computational times as a function of the number of processors used for different number of water molecules using SHM method in native systems.

$$SpeedUp(n_p) = n_p \cdot Eff\,(n_p)$$

Figures 6 and 7 graphically show the achieved speedups. These results show a very good scalability for all underlying algorithms, with values close to the ideal linear case (where $SpeedUp(n_p) = n_p$).

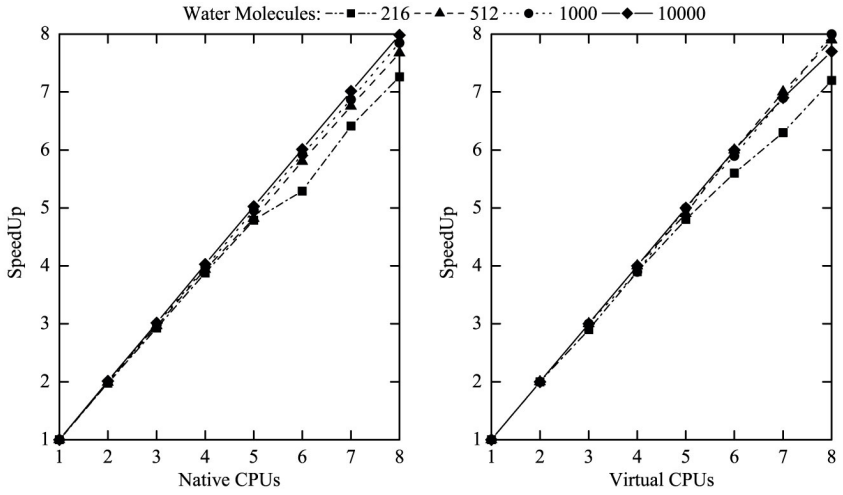

Fig. 6. Speedup as a function of the number of processors for different number of water molecules using SHM method. Left native SMP system; right virtual SMP systems.

The most significant result is the very good scalability of the parallelization algorithms. Both strategies show a linear speed up behaviour close to the ideal case. Only the small physical system with N_{mol}=216 gives slightly worse speedup results with efficiencies of about 92% for n_p=8 processors. For the largest physical systems, efficiencies of about 98% using 8 processors have been achieved.

Regarding Xen virtualization, its impact over the performance and scalability of the codes is almost negligible. No significant penalty has been found for all the studied cases. Neither the virtualized network nor the Xen memory management system shows a negative impact over MPI message operations and shared memory performance respectively.

7 Conclusions

This paper presents two different Molecular Dynamics algorithm's parallelization strategies. The Interactive European Grid test bed has been the selected

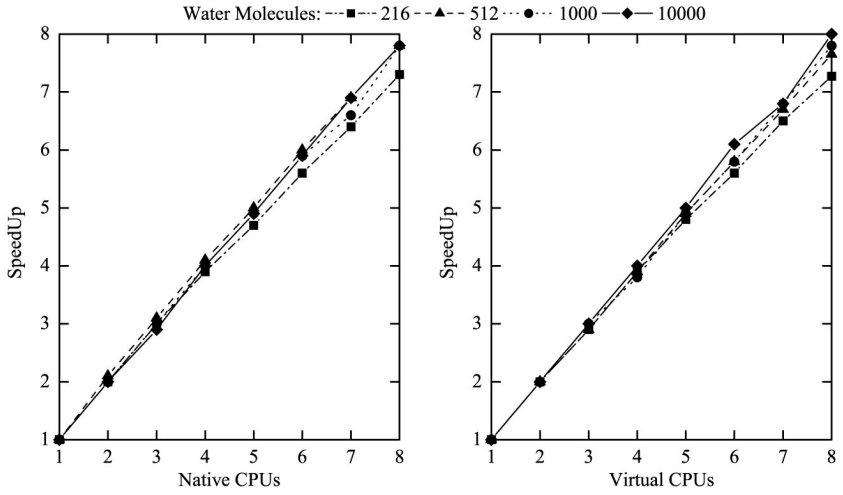

Fig. 7. Speedup as a function of the number of processors for different number of water molecules using MPI standard. Left native eight core SMP system; right virtual single core systems.

infrastructure to evaluate the impact of the Xen virtualization technology over the performance of such simulations.

The achieved speedup and efficiency have always been close to the linear speedup case and no significant differences for both parallelization methods were observed. For all physical system sizes simulated, the use of Xen virtualized operating systems on the different working nodes yields no penalty for any parallelization strategy.

The inter processor communication times were always no more than 0.2% of the total processing time. Such small values indicate that both the new multicore technologies and Xen virtualized systems are a hardware/software combination especially well suited for the highly demanding parallel applications as Molecular Dynamics simulations are.

8 Acknowledgments

The simulations reported in this work were carried out at the Fundación Centro Tecnolóxico de Supercomputación de Galicia (CESGA). L.G. gratefully acknowledges to all CESGA and IntEuGrid teams for their support and collaboration.

References

1. J. Casulleras, E. Guàrdia; *Mol. Sim.*, **7**, 155, (1991). J. Casulleras, E. Guàrdia; *Mol. Sim.*, **7**, 155, (1991).
2. E. Swanson, T. P. Lybrand; *J. Comput. Chem.*, **16**, 9, 1131, (1995).
3. V.S. Sunderam, G.A. Geist, J. Dongarra and R. Mancheck; Parallel Comput. **20**, 531, (1944).
4. H. J. C. Berendsen, D. van der Spoel, R. van Drunen; *Comput. Phys. Comm.*, **91**, 43, (1995).
5. R. Hempel; *Lect Notes Comput. Sci.*, **797**, 247, (1994).
6. M. Nelson, W. Humphrey, A. Gursoy; Intl. J. Supercomput. Applics., **10**, 251-268, (1996).
7. Lino García. PhD Tesis. Molecular Dynamics Simulation of Polyatomic Ions in solution. UPC (2005).
   http://www-fa.upc.es/docencia/tesis_llegides/resumen_garcia_cst.pdf
8. P. V. Coveney, editor. Scientific Grid Computing. Phil. Trans. R. Soc. A, (2005).
9. I. Foster, C. Kesselman, and S. Tuecke. Intl J. Supercomputer Applications, **15**, 3–23, (2001).
10. J. Marco et all. The interactive European Grid: Project objectives and achievements. Computing and Informatics, **27**, 161-171 (2008).
11. http://www.interactive-grid.eu
12. Keller, R., Krammer, B., Müller, M.S., Resch, M.M, Gabriel, E, Journal of Grid Computing, **1** (2), 133-149, (2003).
13. B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer; In Proc. of the ACM Symp. on Operating Systems Principles, Oct. (2003).
14. T.F. Vachon and J.D. Teresco; Conference on Parallel and Distributed Computing and Networks, February 13-15, (2007).
15. J. S. Gray; *Interprocess communications in UNIX*. Prentice Hall, (1997).
16. H. J. C. Berendsen, J. R. Grigera, T. P. Straatsma; *J. Phys. Chem.*, **91**, 6269, (1987).
17. H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. di Nola, J. R. Haak; *J. Chem. Phys.*, **81**, 3684, (1984).
18. G. Ciccotti, M. Ferrario, J. P. Ryckaert; *Mol. Phys.*, **47**, 1253, (1982).
19. M. P. Allen, D. J. Tildesley; *Computer Simulation of Liquids*. Clarendon: Oxford, Chapter 5, (1987).
20. W. Smith; Comp. *Phys. Comm.*, **62**, 229, (1991).
21. F. Manuel Márquez; UNIX, advanced Programming. Ed. Ra-Ma (1996).
22. K.A. Robins; Practical UNIX programming. Prentice Hall (1996).
23. E.W. Dijkstra, Cooperating Sequential Processes, in Programming Languages, ed. F. Genuys, Academic Press, (1968).
24. E. Fernández, A. Cencerrado, E. Heymann, M. A. Senar. Crossbroker: a grid metascheduler for interactive and parallel jobs. Computing and Informatics, **27**, 187-197 (2008).
25. K. Dichev, S. Store, R. Keller, E. Fernández. MPI support on the grid. Computing and Informatics, **27**, 213-222 (2008).