Comparing scientific codes on different parallel platforms

Stefano Cozzini[1,3], Roberto Innocente[2,3] and Marco Corbatto[2,3]

[1] *INFM, SISSA Trieste Unit*

[2] *SISSA, Via Beirut 2/4, 34014 Trieste (Italy)*

[3] *ICTP,Strada Costiera 11, 340100 Trieste (Italy)*

August 23, 2000

Abstract

In the Condensed Matter group at Sissa we developed a certain number of parallel codes that are now routinely used to perform scientific research in computational condensed matter physics. The target machines of these scientific packages have been T3E and SGI Origin-2000 for almost three years. Recently we started some experiments with a Beowulf class machine locally developed. In this paper we briefly describe our machine and we show the behavior of a representative suite of these scientific parallel codes on our prototype. We compare the performances achieved on our Beowulf Cluster against results obtained on the SGI parallel machines used so far. The differences in performances are discussed with respect to the computational characteristics of the codes. We will show that none of the architectures considered fits all the requirements of our suite of codes, but specific machines can address specific problems.

1 Introduction

Computer Simulation is a fundamental tool for scientific research and the condensed matter sector in SISSA has a long and outstanding experience in the field of numerical simulations. The rapid growth of computer power, including the new parallel architectures, allows to deal nowadays with computational problems that were judged not even affordable only a few years ago. The computational techniques currently used can be classified in three main classes:

- First-principles or ab-initio methods where the quantum description of the electronic properties of matter is taken into account; the size of complex systems simulated is limited to, at most, hundreds of atoms: examples includes metal surfaces or active sites of biological systems.
- Classical Molecular Dynamics (MD) techniques, which allow to simulate systems composed of thousand of atoms like for instance molecules of biological interest.
- Quantum Monte Carlo (QMC) techniques which are used to study strongly correlated systems.

Codes belonging to different classes are different also from the computational point of view. It is therefore fundamental to have at our disposal computational resources that fit with the different kind of computational needs we have. Computer resources on parallel and massively parallel computer for the last three years were obtained by means of MURST grants which allowed Sissa to be equipped with an SGI Origin-2000 machine with 16 processor R10000 and with a 32 CRAY T3E nodes (out of 256) present at Cineca Supercomputing Center.

The relative obsolescence of our computer resources recently motivated us to test different parallel platforms in order to understand which is the hardware solutions that can fit at best our computational requirements. For this reason a set of representative codes was set-up and by means of them we are currently benchmarking several parallel architectures. Along this line we started to build a parallel machine based on SMP Intel nodes connected by high-speed networks. A small prototype (4 node with 2 processor each) is now currently under exhaustive test of our scientific computational requirements. The goal is to define which class of computational tasks this type of machines can solve with the best performance/price ratio. The aim of this paper is to report the work done in this direction.

The paper is organized in the following way: section 2 describes the set of programs we used to test our parallel platforms. In section 3 we present some technical details of our cluster implementation mainly discussing the network interconnections and their performances. A detailed analysis of the performances of the codes is presented and commented in section 4. Some conclusions will be drawn in Section 5.

2 The suite of codes

In this section we describe the representative suite of parallel codes employed in this study. As already pointed out the three main classes of applications used by our researchers have distinct characteristics from the computational point of view.

Ab-initio codes are computationally demanding in term of both memory and CPU time. Parallel versions of electronic structure codes have been already developed in the past years using the Message Passing (MP) paradigm. In this way parallel codes could exploit at maximum the modern massively parallel computers (MPP). Due to this effort, highly scalable programs, showing an almost linear speed-up over a wide range of processors (64/128) are now routinely used. We include in our suite two parallel codes belonging to this class, both of them developed at Sissa: PWSCF and FPMD.

Classical Molecular Dynamics codes has different computational characteristics: the memory requirements are lower and the scalability is limited over a small number of processors. For this reason such codes work better on shared memory machines which are the target machines of the locally developed codes. The benchmark suite includes two parallel codes of this type: DLPROTEIN_2.0 and Amber-5.0

The codes implementing Quantum Monte Carlo techniques do not need too much dynamical memory and can be run efficiently on relatively small machines where each processor works on its own task and communications do not represent a bottle neck. The parallel code gf3z is the representative QMC code included in the suite.

In the following a short description of each code is given together with some details about the computational load and the parallelization aspects.

2.1 Ab-initio codes: PWSCF

The PWSCF (Plane Wave Self Consistent Field) code is the main computational tool used in the Material Properties group at Sissa. The program is completely parallel and it has been already successfully ported to all the parallel platforms available (IBM-SP2/3,T3E and Origin-2000). The program PWSCF is maintained and continuously improved by S. Baroni, A. Dal Corso, S. de Gironcoli, and P. Giannozzi. It has recently received parallel support by S. Cozzini. The code computes the electronic band structure, the electronic charge density and the total energy of a periodic crystal allowing the study of its electronic, structural and dynamical properties.

The main computational platform is the Cineca T3E and for this reason a careful analysis of the performances and subsequent optimizations for that platform has been carried out [1].

In the following we briefly outline the computational load of the program with respect with the size of the inputs. We refer to reference [1] for further technical details about the code.

In the code the wavefunctions are represented in a Plane Waves (PW) basis in the reciprocal space and the number of PW's N_{pw} is one of the main parameters which define the computational load of the code. The code makes large use of Fast Fourier-Transform (FFT) algorithms to transform wavefunctions from the reciprocal space to the direct space. Other quantities defined on a grid with dimensions strictly related to N_{pw} are FFT transformed in the same way. The number of operations related with N_{pw} is then proportional to $N_{pw} \times lnN_{pw}$

The most time consuming step of the program is the computations of the wavefunctions by means of an iterative procedure (the Self Consistent Field: SCF). This task is accomplished by two methods: a Conjugate Gradient (CG) routine, or a Davidson algorithm. There are two other parameters, N_b, number of bands of the system and $N_{projector}$, number of projectors needed to describe the nonlocal pseudopotential, which contribute to the computational load of SCF procedure.

The CG method requires several applications of FFT procedures and a large number of vector-vector operations. The size of vectors is proportional to N_b x N_{pw}. These linear algebra operations are accomplished by means of BLAS1 routines.

The Davidson method [5] is generally more efficient and is the most often used. The computational aspects can be summarized as follows:

- Linear Algebra Kernels : several vector-matrix and matrix-matrix operations are required. The size of the data is $N_b^2 \times N_{pw}$ for vector matrix operations and $N_{projector} \times N_{pw}$ for matrix-matrix operations. These tasks are performed by BLAS2 and BLAS3 routines.
- FFT algorithms: the order of operations is proportional to $N_{pw} \times lnN_{pw}$
- Diagonalization procedure: a matrix of dimension proportional to N_b is diagonalized using appropriate LAPACK routines for generalized eigenvalue problem: the number of operation required is therefore proportional to N_b^3.

The parallel implementation is achieved with an even distribution of the PW among the processors (PE). Each PE owns an effective number of plane waves approximately equal to the others. The FFT grid is also split among them. Both the CG algorithm and the Davidson procedure exploit the PW distribution and almost all the tasks required by these procedures are done in parallel. The linear algebra kernels scale linear because the data size is split among processors but a certain communication overhead is introduced to collect data. These communication operations are mainly global operations (reduce gather/scatter) on specific pools of processors. Concerning FFT the code implements "ad hoc" parallel FFT algorithms to exploit at the best the underlying structure of the distributed data. The matrix diagonalization is done in parallel only if the size of the matrix involved is larger than a threshold value. The code scales well over a wide range of processors (16-64) on T3E. Systems currently under study require a very large amount of memory and the number of T3E processors needed for these runs is in the 32-64 range.

2.2 Ab-initio code: FPMD

FPMD [2] is a modular parallel code implementing the Car-Parrinello algorithm including the variable cell dynamics. The code is developed by C.Cavazzoni and G.L.Chiarotti: it is written in F90 and makes use of some new programming concepts like encapsulation, data abstraction and data hiding. The code has been recently used to simulate water and ammonia at giant planet conditions. The computational

cost and the parallelization strategy of the code are very similar in many aspects to the previous PWSCF program and here we limit to report in table 1 the main computational tasks and the communication algorithms with the relative cost. We refer to reference [2] for a full description of the code.

Tasks	Computational Load
Diagonalization	N_b^3
Matrix-vector operations	$N_b^2 \times N_{pw}$
FFT transforms	$N_{pw} \times lnN_{pw}$
All to All	$N_b \times N_{pw} \times lnN_{pw}$
Global reduce	$N_b^2 \times lnN_{PE} + N_{at} * N_b \times ln(N_{PE})$
Gather and scatter	$N_{pw} \times N_b$

Table 1: Computational load and communication algorithms for the FPMD code. Here N_{pw} is the number of plane waves, N_b is the number of bands, N_{at} the number of atoms and N_{PE} is the number of processors

2.3 Classic MD code: DLPROTEIN_2.0

DLPROTEIN_2.0 [3] is a Molecular Dynamics package written by S. Melchionna and S. Cozzini and it is a development of the original general purpose Molecular Dynamics code DL_POLY_2.0 [6] written at Daresbury Lab (UK) by W. Smith and T. R. Forester. The motivation underlying the development of DLPROTEIN_2.0 has been to create a simulation package well suited for biological molecules, with particular focus on proteins, with high efficiency and with a programming language and style suitable for complex molecular topologies. DLPROTEIN_2.0 is a a package consisting of two major parts, a topology builder for (bio)molecules and a molecular dynamics (MD) engine. DLPROTEIN_2.0 allows to simulate systems in different thermodynamic ensembles (NVT,NPT,NHT) by means of time reversible algorithms and implements modern methods to reduce the computational cost of the simulation.

Classic MD conceptually is quite simple: it is an iterative process where at each step the sum of all forces on each atom is calculated and then applied, updating the position and the velocity values. The forces originate from bonded and non-bonded forces between atoms. A single atom has bonded-related force with a limited number of other atoms, while the non-bonded forces exist between all pairs of atoms yielding a $o(N_a^2)$ interactions (N_a being the number of atoms). The non-bonded forces are comprised of electrostatic forces (long-range) forces and Lennard-Jones forces (short range forces) and there are different tricks to deal with them. Short range forces are computed by means of the neighbor list technique. Each particle just interacts with particles within a certain cutoff distance (the neighbors) and these atoms are stored in a list. The list is nothing but an array of pointers to the position arrays. This means that the data are loaded in a very scattered way with bad performance of the cache mechanism. The amount of computation required is proportional to N_a (with a large proportionality constant: the number of neighbors: N_{neigh}). Long Range forces are treated by means of the Smooth Particle Mesh Ewald (SPME) algorithm that requires FFT kernels on a 3D grid. The size of the grid is defined by the N_a, number of atoms in the system. The non bonded computations (short range + long range) constitute between the 80 and 95 percent of the overall computations, depending on the systems. Table 2 summarizes the main tasks of an MD program and the algorithms implemented in DLPROTEIN_2.0.

Tasks	Methods	Computational cost
Short Range Forces	Link-cell + neighbor lists	$N_{neigh} \times N_a$
Long Range Forces	Smooth Particle-Mesh Ewald (SPME)	$N_a \times lnN_a$
Bonded Forces comp.:	2-body,3-body,4-body potential	negligible
Updating Atoms	velocity verlet algorithms	negligible
Constraint procedures	Shake/Rattle	$N_{constraints}$

Table 2: Computational tasks,algorithms and cost for DLPROTEIN_2.0 code. Here N_a is the number of atoms, N_{neigh} is average number of neighbors, and $N_{constraints}$ is the number of constrained atoms.

The parallelization strategy adopted by DLPROTEIN_2.0 is based on the Replicated Data (RD) approach implemented using Message Passing technique. All data (arrays containing attributes, coordinates and forces of atoms) are replicated on each processor. The force computations can then be evenly distributed among processors at will, as any processor is capable of carrying out any particular force computation. If there are N_a atoms and PE processors the $O(N_a/PE)$ forces accumulated by each processor must be added up across all processors. This requires a reduce operation with a communication time proportional to $N_a \times lnPE$. A similar operation is required to update positions among processors as each of them just moves N_a/PE particles. The ratio among communication time and computation time is therefore given by $PE \times lnPE$ and it is independent of N_a. So if we want to simulate a system twice as large as the current one, we cannot hope to double the number of processors to retain the same efficiency, because the time spent in communication will be larger. Thus RD is non scalable.

It is important to note however that in practice RD works effectively the range 1-8 and there is no real need of large MPP platforms to run this kind of code; memory requirements, even for large systems are limited and can be easily satisfied by small SMP machines.

2.4 classical MD: Amber-5.0

The Amber-5.0 package is another MD code heavily used for simulation of proteins here at Sissa. Several users are currently running very large systems with the MD engine of the package and for this reason we include this program in our benchmark suite.

The computational characteristics of the code are practically the same of DLPROTEIN_2.0; also the parallelization strategy is the same as before,i.e. Replicated Data. There is however a significant difference in its implementation for the SGI Origin-2000 version of the code: on this platform the code is parallelized using a Shared Memory approach. This means that the overhead due to communication in Message Passing implementation could be greatly reduced and the scalability of the code improved. The parallelization has been done using specific Origin-2000 directives and not using a standard like openMP. The Shared Memory version is therefore specific for that architecture while for other platform a standard MPI version is available. We compiled the code on the Beowulf cluster using the MPI version.

2.5 Quantum Monte Carlo code: gf3z

The QMC code included in the suite has been developed by S. Fantoni and coworkers and applies a particular formulation of the QMC techniques to a systems of Nucleons.

This formulation allows to sample statistically the ground state of a many-body Hamiltonian by a set of "walkers". By statistical iterations of a very large number of walkers the ground state of the system could be estimated.

The computational load is related to the number of particles which compose the system: this number defines the size of the "walkers" vectors and the size of Hamiltonian matrix diagonalized at each iteration. The sizes of the quantum systems numerically investigated are not very large and therefore the diagonalization procedure is not computationally dominant.

The code can easily be parallelized because an even number of walker can be assigned at each processor at the beginning and the simulation. Each processor can then run independently. It is necessary however after several iteration to check the number of walkers on each processor and balance the load among the processors because the number of walkers does not remain constant as the simulation proceeds. The load of this operation is anyway negligible for this specific case and the code scales linearly over the all range (1-256) on the T3E machines.

2.6 A global comparison

As final points of this section we want to compare the three families of codes looking at specific aspects of high performance computing (HPC). Table 3 presents such a comparison. The three columns represents the three classes of codes we discussed above while each row indicates a peculiar characteristic of the HPC. An adjective defines the behavior of each class on that specific subject. From the table emerges clearly that the computational requirements of the three classes are quite different.

aspect	Ab-initio	Classical MD	Quantum MC
`Communications`	high	high	negligible
`Memory requirements`	high	moderate	moderate
`Parallelization strategy`	MPI	MPI/Shared Memory	MPI
`Scalability`	high	not scalable	linear
`Use of Linear Algebra kernel`	high	almost null	moderate
`Accessing Cache`	good	bad	moderate

Table 3: Comparison between codes on different aspects of HPC.

3 The Beowulf machine

3.1 Cluster configuration

The cluster is composed of 4 dual processors computing nodes interconnected by 3 different network technologies: Fast Ethernet, Gigabit Ethernet and Myrinet. A service node is connected only to the Fast Ethernet network and performs as dhcp/boot/file server and batch scheduler. On the computing nodes

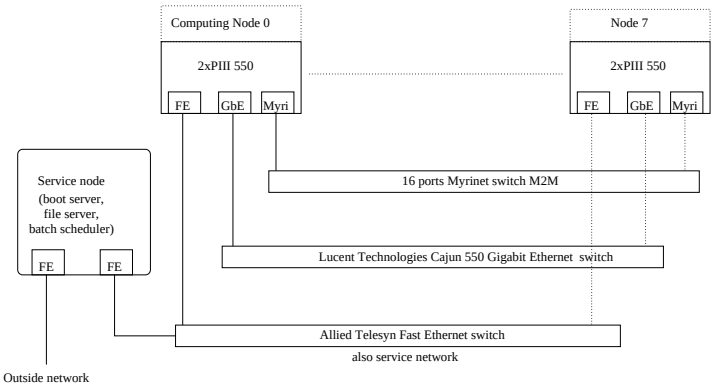

Figure 1: Schematic view of the Beowulf cluster with indicated the network interconnections installed.

there are 2 Pentium III (Katmai) processors running at 550 Mhz. These processors have an on chip 16 KB + 16 KB separate instruction and data level 1 cache and an off chip (but in-package) discrete 512 KB level 2 cache on a separate bus (Back Side Bus) at f/2 Mhz.

The processor bus or Front Side Bus runs at 100 Mhz. On the Front Side Bus, the Intel 440 chipset manages the memory and the peripheral buses(a 32 bits @33 Mhz PCI). Memories are commodity 100 Mhz SDRAM. Each computing node has 256 MB of memory. A schematic view of the machine is presented in fig 1.

3.2 Software setup

Because of the requirement to frequently switch kernels, drivers and software setup, since the beginning we planned to reduce at a minimum the system management overhead. For this reason we installed netboot eproms on the Fast Ethernet cards and all the computing nodes access their root and usr partition on the service node via NFS. The nodes are now running a linux 2.2.14 smp-enabled kernel. Computing nodes have a local disk that is mainly used as a temporary scratch area and swap disk. We chose to install PBS (Portable Batch System) as our batch system. We mainly use the fortran compilers available from the Portland Group (PGI).

3.3 Network hardware

The service network - a Fast Ethernet - supports the remote boot, common file systems through NFS, remote logins, etc. We have used for the service network 3COM 3C905B cards and an Allied Telesyn

switch. For this network the most strict requirement was the possibility to have netboot eproms on the cards. We used it in some comparison as a reference.

The aim of the cluster was the characterization of the performance for multiple interconnects and software combinations. As high performance interconnect we installed Myrinet and Gigabit Ethernet.

- Gigabit Ethernet: this technology promises to make available during the next years NICs working at 1 Gbit/s as commodity hardware. Nevertheless many of the devices available in the present, with prices comparable to those of other Gigabit/s technologies, have serious difficulties in obtaining results near what can be expected from the nominal performance of this technology. Two techniques have been developed to alleviate the problem : *interrupt coalescing* and *jumbo frames*. *Interrupt coalescing* refers to the possibility for the NIC to group together multiple interrupts issuing one host interrupt for multiple events. *Jumbo frames* refers to the possibility to use "big" frames up to 9000 bytes (this frames are not compatible with the Ethernet/802.3 standards and therefore could'nt be generally employed outside a restricted environment of switches and hosts). The cost of this solution is about 500$ for each NIC plus at least 7000$ for a small switch. As a representative of this technology we chose the 3com 3c985 cards. These cards are made out of an ASIC chip with 2 on board 32 bits 88 Mhz RISCs and 2 independent DMA engines. They have 1 MByte of memory on board. These cards support receive hardware TCP/UDP/IP checksumming, jumbo frames and receive/transmit interrupts coalescing. The cards can also support Ethernet flow control in hardware.

- Myrinet: it is a proprietary net that evolved from USC/ISI ATOMIC LAN. At the moment the speed is 1.28 Gigabit/s. Crossbar switches up to 16 gates are available; for an higher number of nodes multi-stage nets can be used. The net card is equipped with a RISC chip of good performance which can be programmed to reduce the central unit (processor and memory) load. The specifications of the card and of the RISC chip are available as well as various software like optimized drivers and communication libraries (GM, mpich over myrinet). The actual cost is about 1000$ per card plus 4000$ for a 16 port switch.

3.4 The GM Protocol

The quality of communication software has a great weight in the overall performance of a cluster. Various protocols are available (Link-Layer, VIA, TCP/IP, GM, BIP) and we are currently testing all of them. The most stable among them is the GM protocol and this protocol is the one we used to test the machine against our suite of code. GM is a lightweight communication system for Myrinet that supports reliable ordered delivery and protected access, provided directly by Myricom. It requires the use of DMAable memory (registered with the system or allocated through the system). It supports messages up to 2^{31}-1 bytes if the OS allows such amount of DMAable memory. It has 2 levels of message priority to help avoiding deadlocks in an efficient way . In the GM programming model a reliable connection is established between hosts, while communication endpoints do not need any communication establishment to communicate(connectionless). It supports up to 10000 nodes. Sends and receives are regulated by implicit tokens representing space allocated to the client by the system in its queues.

3.5 Performance results

We report the performances of the various network protocols indicated before We used BIP 0.99, GM 1.2 and MPICH 1.2.0 both over TCP and over GM. During the tests a considerable difference in the TCP latency time has been observed enabling/disabling the SMP capability of the Linux kernel: this is due to the overhead in locking/unlocking of the internal structures to assure mutually exclusive access in case of multiple CPU.

Table 4 reports the different values of latency measured. Latency time over Myrinet is comprehensive of the switch delay while in the case of fast Ethernet both values were measured. The high value for gigabit Ethernet (using a direct link) is due to the use of the *interrupt coalescing* feature of the NIC to assure an high b/w.

The best bandwidth obtained on the cluster are presented in figure 2.

At the MPI level we measures performances by means of the PALLAS benchmark suite MPI2. The figures 3 and 4 show the bandwidth obtained on a ping-pong and times measured for a reduce operation

LATENCY TIME (us)	SMP Kernel	UP kernel
TCP loop-back	53	29
TCP 100Mb direct link	98	53
TCP 100Mb switched	105	61
TCP 1Gb direct link	490	490
TCP over Myrinet	103	n.a.
GM over Myrinet	15	15
BIP over Myrinet	5	n.a.
MPICH/TCP over 100Mb	172	n.a.
MPICH/TCP over 1 Gb	490	n.a.
MPICH/GM over Myrinet	15	n.a.

Table 4: Latency's measurements for different protocols on different networks.

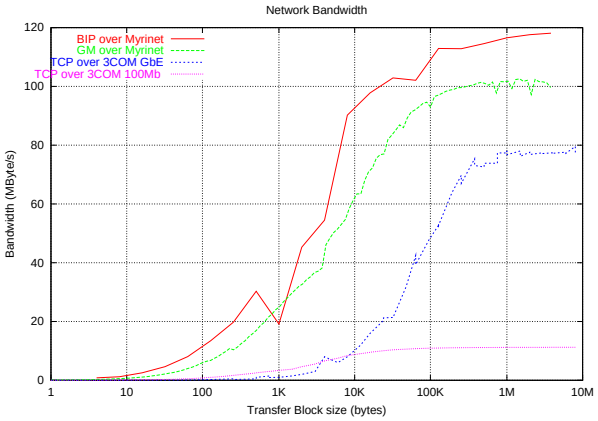

Figure 2: Best Bandwidths on Bewoulf : BIP/GM/TCP protocols.

using 8 processors. If available we inserted data for T3E and ORIGIN 2000. It has to be noted in fig 4 the bad performances in the range 10K-100K on the myrinet network.

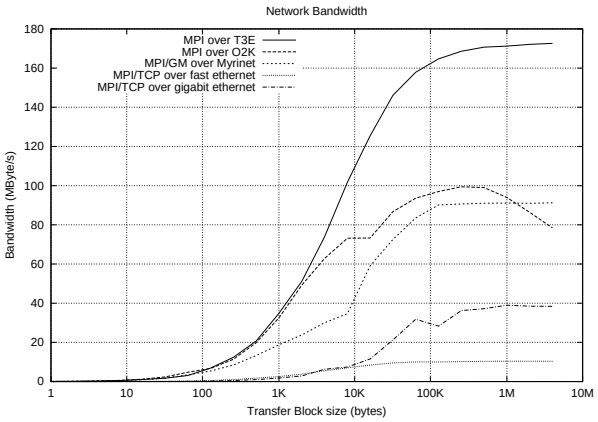

Figure 3: Network bandwidth measured by MPI.

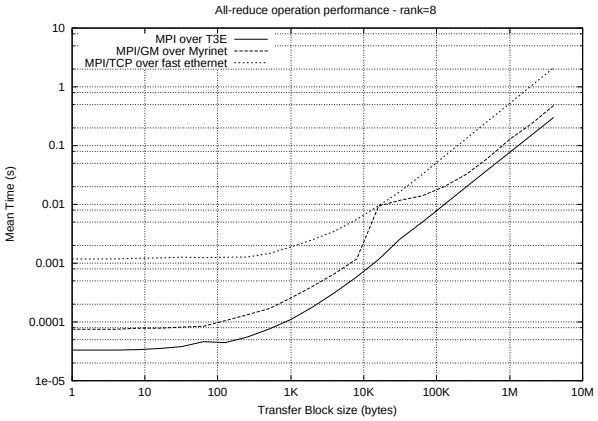

Figure 4: times vs size for MPI reduce operation. 8 processors are used.

4 Results

In this section we present the results obtained running our codes on three parallel machines: T3E and Origin-2000 installed at Cineca and our local Beowulf cluster. Table 5 provides a short comparison of three platforms. A rough evaluation of prices is included: we report the cost of our 32 PE quota on Cineca T3E and the cost of our local 16-node ORIGIN-2000.

For each code a certain number of tests was prepared and then run on all the three parallel machines. Execution times were collected using the internal timing system of each code. Tests are repeated several times in order to estimate the error that is generally less than 2% on dedicated machines like T3E and the Beowulf clusters. Origin-2000 machine is a shared machine and therefore times could be quite different when we run in parallel. In order to minimize this "sharing effect" we monitor the load of the machine and we tried to run our tests when at least the double of the requested processors were free. This precaution seems to work for 2/4 processors runs (error bar is $\approx$ 2-3 %) while for 8 processors the differences between runs remain still large in some cases ($\approx$ 10-15%).

For QMC and classic MD codes we measure performances on tests that are actual systems under study. In this way the comparison among parallel platforms is complete and significant for our scientific computational requirements. In the case of ab-initio codes this was not possible: the small configuration of our cluster does not permit to run large systems but just to test specific "ad hoc" examples and some medium size examples.

System &processor	Beowulf	CrayT3E	Origin 2000 (O2K)
processor	Intel PIII 550 MHz	Alpha 21164 600Mhz	R12000 300Mhz
N of processors	8	256	64
DRAM (total)	1Gbyte	49 Gbyte	32Gbyte
Primary cache(data)	16Kbyte	8 Kbyte	512 Kbyte
Secondary cache	512 Kbyte	96 Kbyte	8 Mbytes
Network			
Kind	Myrinet switch	Torus 3-D	Fat Bristled Cube
Bandwidth(Pallas)	90 Mbyte/sec	180 Mbyte/sec	100 Mbyte/sec
Latency(Pallas)	15 microS	13 microS	12
Software			
O.S	Linux 2.2.14smp	Unicos/Mk 2.0.4	IRIX 6.5.8f
Compilers	PGI 3.0.3	Cray Compilers 3.0.0.3	MIPS Pro 7.30
Comm. Library	MPICH over GM	Cray libraries	SGI libraries
Costs	$\approx$ 25,000$	$\approx 20\times$	$\approx 10\times$

Table 5: A comparison among the parallel platforms employed in this study. Values of latency and bandwidth are estimated by means of the Pallas MPI benchmark.

4.1 Ab-initio codes: PWSCF

We consider three different tests for this program. The first two examples are similar: a full relaxation toward the minimum energy configuration of two small molecules (CO and O_2) is performed. The self-consistency procedure (SCF) is computed several time using the Davidson procedure for both the examples. These two tests are very small examples that can fit on single processor allowing to estimate the computational efficiency of the code on different kind of processors.

The third test is a C_{60} molecule. For this case we perform only a single step of the self-consistency procedure. The step was done using the two procedures the code offers: Conjugated Gradients and Davidson. This test is a medium size example and it is the largest one that can be successfully run on our small Beowulf machine. The system is described by a small number of plane waves but with a relatively large number of bands and projectors: it follows that the linear algebra kernels in the Davidson procedure (Diagonalization and BLAS2/3 routines) are dominant compared to FFT kernels. We run the test using 8 PE on all the architectures.

The sizes of parameters for all the tests are reported in table 6.

Parameter	CO molecule	O_2 molecule	C_{60} molecule
N_{pw}	3407	3887	20199
N_b	20	40	120
$N_{projector}$	16	16	60

Table 6: Input parameters for test cases of the PWSCF code.

First we discuss the behavior of the small two tests with the help of table 7 where the total time and the time spent by the code in doing FFTs are reported. Linear algebra load is marginal compared to the FFT cost in SCF procedure and it is therefore not reported.

The R12000 processor is the fast processor but the behavior of the code on the Pentium III machine is quite promising with respect to the RISC 12000. The relatively old Pentium 550Mhz is just twice slower than the RISC 12000. Our hope is that this factor can easily reduced using Pentium III processors available now (733 /800 MHz). The other positive aspect is the good efficiency of the FFT procedures on the INTEL architecture: the public FFTW library we currently use is the code works fine and compares well with highly optimized vendor libraries. Again we hope that with other powerful INTEL processors the gap in performances will eventually disappears.

We now discuss the parallel behavior by means of the third example; data here are reported just for T3E and Beowulf cluster since on the ORIGIN-2000 at Cineca, this large example cannot be correctly

	CO			O_2		
sections	Beowulf	T3E	O2K	Beowulf	T3E	O2K
global time	409.2	321.42	210.48	1086.57	773.27	564.78
FFT procedures	97.96	94.49	53.38	296.88	267.87	166.83

Table 7: CPU times for the main tasks of the two test systems for PWSCF code. Times are in seconds.

timed due to the fact the "sharing effect" was quite pronounced. We use LAPACK/BLAS library generated by means of the ATLAS project on the Beowulf cluster because the precompiled pgi versions of these libraries are very inefficient.

Table 8 refers to the Conjugate Gradient algorithm. There is an overall factor of less than two in performances among the two architectures; we report global times for FFT procedure and specific sections of the code where BLAS1 routines are used. The ratio in performances on these different parts of the code ranges from 1.65 to 2.35. This means that on these main sections of the code the gap in performances between the two architectures is similar.

It has to be noted however that on the INTEL processors the efficiency of the BLAS1 routines in a particular section (add_1vupsi) is 2.35 times slower than on the Alpha. This indicates that for BLAS1 routines there is still a large gap among INTEL and RISC processors.

procedure	Beowulf	T3E	ratio
electrons	639.93	345.94	1.85
fft	342.86	207.05	1.65
blas1 (s_1psi)	75.60	38.67	1.97
blas1 (add_1vuspsi)	57.93	26.08	2.35

Table 8: CPU times for self consistent procedure for C_{60} molecules using Conjugate gradients. Times are in seconds.

We discuss now the Davidson procedure (table 9) where diagonalization procedure and BLAS2/3 routines play a major role. The overall factor in this case is 2.16. Looking at specific sections of the code the situation is similar to the previous case with just an exception: the diagonalization procedure; this bad results is because this procedure is done in scalar on the Beowulf Machine: the parallel diagonalizer used on the T3E is still to be ported on this architecture.

procedure	Beowulf	T3E	ratio
electrons	685.61	317.42	2.16
fft	224.77	138.50	1.62
diagonalization	175.44	34.54	5.14
blas3 (overlap)	68.11	34.46	1.98
blas2 (update)	112.43	76.92	1.45
blas3+blas1(last)	35.36	17.70	1.99

Table 9: CPU times for self consistent procedure for C_{60} molecules using Davidson iteration procedure. Times are in seconds.

We can turn now on network performances for the parallel runs. It is interesting to note the behavior of the FFT routines. Performances obtained by the scalar version of the FFT procedure are practically the same on Beowulf and T3E. The situation changes for the parallel case: on the Beowulf cluster parallel FFTs run approx. 1.6 slower than on T3E. The communication overhead of the Beowulf is therefore 1.6 times than on T3E: comparing this result with the bandwidth ratio measured by the Pallas benchmarks (≈ 2) (see table 5) one could infer that in the case of FFT point-to point communication the actual performances of the Myrinet network are excellent.

A final note concerns the network performances of the reduce operations used on C_{60} simulations. Data are collected in the following table:

system	Beowulf	T3E	ratio	N of calls
CG	33.40	12.40	2.69	20037
DAV	19.81	5.14	3.85	82

Table 10: CPU times for reduce operations for C_{60} tests.

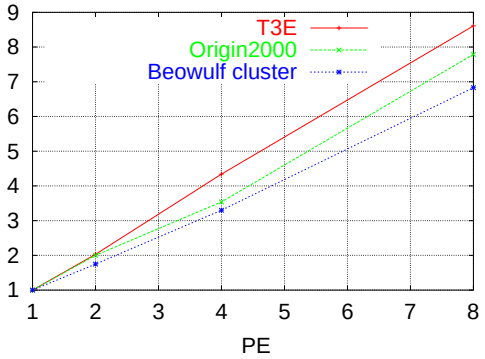

Figure 5: Speed-up of FPMD code.

There is a very large number of reduce operations in CG procedure, each of them costing little, and from this point of view this could measure a kind of "code reduce" latency over the network. On the contrary the limited number of the time consuming reduce operation in the Davidson procedure can measure the "reduce" bandwidth available to our code. The ratio between platforms of the "code reduce" latency (2.69) is quite similar to the same quantity estimated using Pallas MPI benchmark results. In the case of the "code reduce bandwidth" the high ratio between values obtained on the two platforms seems to indicate that the average size of the data involved in the operations is in a range where performances are quite bad for myrinet (see fig 4).

4.2 ab-initio: FPMD

FPMD code comes with a benchmarking test composed by a small system of 32 water molecules. We therefore run this test on our cluster and the data obtained are then compared with performance data provided with the code . The size of the parameter set is the following: $N_b = 128$, $N_{pw} = 14000$ giving a FFT of 72x72x48 and $N_{at} = 96$. This small test fit an a single processor and can be used to test parallel performances over small number of processors. The following tables compares the performances obtained on the three architecture over the range of processors we can use.

	Beowulf		T3E		O2K	
processors	time	speed-up	time	speed-up	time	speed-up
1	119	1	56.0	1	42.1	1
2	68	1.75	27.5	2.03	21.0	2.00
4	36	3.30	12.9	4.34	11.9	3.54
8	17.4	6.83	6.5	8.61	5.4	7.79

Table 11: CPU times and scalability for FPMD code. Times are in seconds.

The comparison among processors shows that the code is more than 2 times slower than T3E on single processor. This result is in agreement with parallel performances data collected for PWSCF code. Figure 5 gives a graphical idea of the overall scalability of the code on the three different architectures.

The super-linear scalability on the T3E is mainly due to cache effect: the reduction in size of local arrays could reduce also the cache miss events, exploiting at best the cache mechanism. The code scales well also on the ORIGIN2000 machines with a bad speed-up just for the 4 processor case.

The behavior on the Beowulf cluster is not so good; speed-up is disappointing for 2 and 4 processors while is slightly better for 8 processors. The communication overhead is larger than the expected one. We guess that reduce operations are working on data sizes not favorable for myrinet as pointed out in the previous section.

4.3 classical MD: DLPROTEIN_2.0

DLPROTEIN_2.0 performances are measured on three tests. The first test is a standard benchmark of the code: a system composed by 4096 water molecules that allows to test specific algorithms to deal with water molecules. Test 2 refers to a system composed by two identical proteins (1347 atoms each) solvated with 5494 water molecules. The system is currently under study at the University of Roma. The third test is a system containing 45 Micelle (each of one formed by 80 atoms) solvated with 8513 water molecules. System 2 and 3 have an high degree of connectivity with a large number of constraints. Test 1 one is different from this point of view: it only has water constraints. The values of the significant parameters of the tests are collected in table 4.3.

Parameter	Test 1	Test 2	Test 3
N_{atoms}	12288	19176	32829
N of waters	4096	5494	8513
$N_{constraints}$	12288	$1117\times2 + 5494 \times 3$	$45\times 105 + 8513\times 3$
averaged N_{neigh}	≈ 47	≈ 200	≈ 50

Table 12: input parameters for test cases of the DLPROTEIN_2.0 code.

We present and discuss performance data in two separate section: the single processor performances and the parallel performances.

4.3.1 Single Processor performances

Table 13 collects scalar performances for the three tests. We report for each test the global time and time spent in the most consuming tasks of the code. Times are referred to single iteration. The short range computation is split here in the two subtasks: building the neighbor list and the actual computation of short ranges using the list. The two subtasks have different computational characteristics: the first is based on integer operations and data are accessed in direct way, while in the second one floating point operations are dominant and data are accessed through the list.

There are several observations that can be made:

- Intel processor outperforms the alpha processor. R12000 is still faster than INTEL but the gap reduces compared with ab-initio codes.
- Building lists is an operation where INTEL processors dominates even over R12000 for all the tests; it this probably due to the excellent performances of the INTEL architecture on integers.
- R12000 shows the effect of its large secondary cache in the short range computations where data are accessed in a very scattered way. The cache miss effect is minimized only if processor has large cache. The bad performances on T3E are due to the small size of its cache.
- The SPME algorithm based on FFT procedures show roughly the same behavior of the ab-initio codes: same performance between INTEL and T3E with a gap around 1.65- 2.0 compared to R12000 performances.
- Performances obtained on the constraints procedures are practically the same between T3E and INTEL for test 2 and 3 and are significantly slower than R12000. Performances are again limited by small caches on T3E and on the INTEL processor; test 1 shows a different pattern: there are very bad performances on T3E while the gap between R12000 and INTEL greatly reduces: this is due to different kind of constraints between systems.

Section	Beowulf	T3E	O2K
Test 1			
Total	5.35	12.54	4.64
Built Link List	0.57	2.72	0.94
Short Range	3.44	6.55	2.50
Long Range (SPME)	0.33	0.42	0.24
Constraints	0.78	2.41	0.62
Test 2			
Total	11.54	18.45	6.89
Built Link List	0.55	1.43	0.72
Short Range	5.89	12.01	3.43
Long Range (SPME)	2.99	2.87	1.50
Constraints	1.48	1.39	0.75
Test 3			
Total	22.84	33.00	12.56
Built Link List	0.90	2.50	1.23
Short Range	9.58	17.90	5.32
Long Range (SPME)	7.98	8.02	4.40
Constraints	3.08	2.92	1.44

Table 13: Single processors performance of the test cases for the DLPROTEIN_2.0 code: times are in seconds and refer to one iteration.

4.3.2 Parallel performances

Parallel performances of DLPROTEIN_2.0 are reported for all the three parallel machines. Figure 6 spots out the overall scalability for the three test cases in the range 1-8 processors; parallel behavior is practically the same for Origin-2000 and Beowulf Cluster: the better scalability of the T3E with respect to the other machines is due to a combined effect of two opposite factors: the lower performances of the T3E processor and the excellent behavior of network.

This observation is easily confirmed by looking at ratio between communication time and total computational time as reported in table 14; for the three examples total time, total communication time and percentage between the two are reported.

PE	T3E			O2K			Beowulf		
	Total	Comm.	%	Total	Comm.	%	Total	Comm	%
TEST 1									
2	598.33	7.49	1	244.23	7.17	3	300.76	11.95	4
4	320.10	14.58	5	138.34	15.36	11	175.49	29.03	17
8	181.09	16.42	9	86.61	20.81	24	111.19	33.22	30
TEST 2									
2	1042.25	18.33	2	379.46	18.34	5	672.61	29.66	5
4	570.36	22.70	4	230.72	33.98	15	403.07	70.63	18
8	326.20	25.22	8	166.48	50.83	30	311.43	121.38	38
TEST 3									
2	926.24	5.57	2	376.97	15.40	4	666.78	26.51	4
4	511.09	19.38	4	223.19	24.00	10	392.86	50.21	13
8	293.74	20.70	7	159.63	38.23	24	269.59	76.74	28

Table 14: Total computational time, communication time and The percentage of the communication time over the total time for three examples. Times are in seconds

Communication/computation ratio is always lower than 10% on T3E machine while on the other two platforms the percentage of time spent in communication can reach almost 40%. It has to be noted

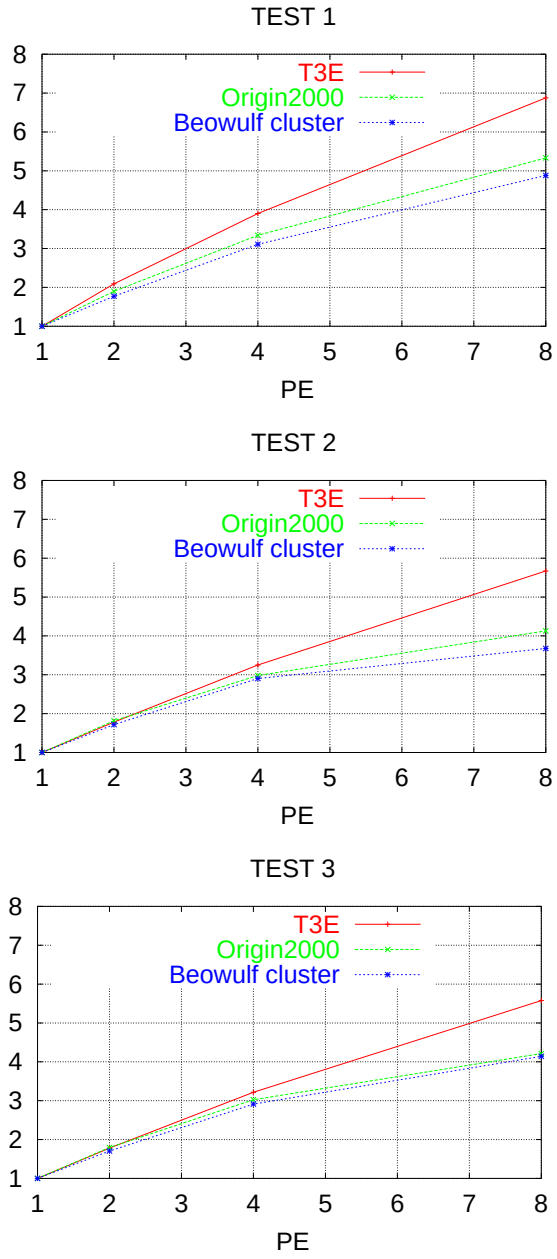

Figure 6: The speed-up for the three test cases considered.

however that parallel global times obtained on T3E are still greater than Beowulf times, indicating that the excellent network performances are not enough to counter balance the bad scalar performance of the T3E processor.

The communication time is given by the sum of all the different communication operations in DLPROTEIN_2.0 during the run. The most time consuming ones are the reduce operations (gdsum) that reduces the force and the velocities values at each time-step and all_gather (merge) where new positions are updated. Performances of these two operations obtained on Test 3 are reported in the following table:

PE	reduce			merge		
	T3E	O2K	Beowulf	T3E	O2K	Beowulf
2	9.09	6.85	12.59	4.85	8.04	12.84
4	8.48	11.46	23.94	4.88	11.73	19.99
8	11.99	19.14	37.54	4.79	18.12	33.64

Table 15: Times for reduce and merge operations for Test 3. Times are in seconds

The T3E network shows excellent performances: by increasing the number of processors the time slightly increases for the reduce operation but practically remains constant for the merge one. For the other two networks the time grows as the number of processor increases. Reduce operation follows quite close the expected logarithmic law while for the merge operation this behavior is not so evident either for O2K or for the Beowulf network. A comparison can be done between myrinet network and Origin-2000 network: the ratio between times for reduce operation is around 2, while for the merge operation is something less. These ratios are similar to the ratios obtained for computation on single node. As a consequence of this we can image that an increasing of computational power of the scalar INTEL processor could cause a decreasing of the overall scalability for this particular kind of codes because the communication time remains constant and weights more and more. This could be a limiting factor for the Beowulf platform.

4.4 classical MD: Amber-5.0

The system we tested using Amber packages is real system currently under investigation here at Sissa formed by a protein (the NGF-TrkA ligand-receptor complex) solvated with 13025 waters for a total of 45717 atoms. Performances are measured just for Origin-2000 and Beowulf Cluster: our goal is to compare performances of codes which use different parallel strategy on the two architectures. Results obtained on Beowulf machine are stable: the error bar is less than 2% while for Origin-2000 the situation is again under the "sharing effects" already explained. Using the shared memory version of the code we noted that "sharing effect" is maximum for 8 processors runs showing differences in time between tests of 15 percent. Data collected for Origin-2000 are not fully feasible in terms of absolute values but still deserve to be compared with Beowulf data. This comparison is done on a system that is currently simulated on the Origin-2000 resource in those precise conditions: a shared resource with a high computational load. Data are presented in the following table:

	Beowulf		O2K		ratio
processors	time	speed-up	time	speed-up	
1	309	1	168	1	1.83
2	191	1.61	102	1.64	1.87
4	133	2.32	70	2.40	1.90
8	99	3.12	61	2.75	1.62

Table 16: CPU times and scalability for AMBER.Times are in seconds.

Scalability results resemble those of DLPROTEIN_2.0 even for the Shared Memory version of the code. The ratio in performances among Beowulf cluster and R12000 is practically constant over the 1-4 range. The 8 processors datum is affected by the problem discussed but still is not far from the other values.

The negative aspect of the results is the limited scalability of both the platforms. This is quite surprising in the case on the Origin-2000 where the different Shared memory parallel implementation seems not to outperform the MPI one as one could expect. To check better this aspect we run a specific benchmark of Amber (4096 H20 water) on the Sissa Origin-2000; this machine is equipped with R10000 processors so performance on single node are lower but we could run the tests in dedicated way. Table 17 summarized data for this test obtained on both Origin-2000 and Beowulf cluster.

	Beowulf		O2K (R10000)		ratio
processors	time	speed-up	time	speed-up	
1	273	1	174	1	1.56
2	150	1.82	96	1.81	1.56
4	90	3.03	56	3.10	1.60
8	60	4.55	37	4.70	1.62

Table 17: CPU times and scalability for 4096 water AMBER benchmark Times are in seconds.

The overall scalability of this benchmark is slightly better than that of the previous case but still the behavior of two implementations looks quite similar with no significant differences.

4.5 QMC

The gf3z code is tested here just for T3E and the Beowulf cluster because these two platforms are in this moment the production platforms. The test we run is just a slice of the large production runs: the code is run in parallel on the full Beowulf machine and on 32/16 T3E processors; the scalability is linear on both machines: communication overhead is practically negligible in both cases. Here we report just the relative performances of two platforms on a single processor to compare the behavior of the code: the Pentium processor is $\approx 5\%$ slower than the Alpha processor. We performed a parallel run using 8 processors on both machines obtaining practically the same ratio in performances.

5 Conclusions

To summarize all our results discussed before we report in table 18 a short overview of the performances obtained by all the codes on the different machines; for each code we define two benchmark values: the first, named Scalar Benchmark (SB) is given by summing times over all the serial runs. The second, named Parallel Benchmark (PB), is similar but the summed value refers to 8 processors runs.

code	SCALAR (SB)			PARALLEL (PB)		
	T3E	O2K	Beowulf	T3E	O2K	Beowulf
PWSCF	1094	775	1485	663	-	1325
FPMD	56	42.1	119	6.5	5.4	17.4
DLPROTEIN_2.0	4249	1781	2831	801	412	693
AMBER-5.0	-	168	309	-	61	99
GF3Z	67	-	71	537	-	569

Table 18: scalar and parallel benchmark values for all the codes. See text for definition.

These benchmark values help to draw some observations about the overall behavior of different classes of codes on the parallel platform we tested.

We consider first ab-initio codes; for this group the T3E machine is the actual production machine and we look at performances of the Beowulf machine with respect to this one. Scalar performances between T3E and Beowulf cluster ($SB_{beowulf}/SB_{t3e} = 1.35 - 2.12$) are better than the parallel ones ($SB_{beowulf}/SB_{t3e} = 1.99 - 2.67$). We can not test at this moment parallel performances on larger configurations (16-32 nodes) that could allow to run real size systems but it seems that the Beowulf network will be unable to cope with large number of nodes. Therefore it could be rather difficult to run

real simulation with the network technology at our disposal in this moment. It has noted however that increasing the scalar performances by means of more recent INTEL CPU, could reduce the need of large number of processors allowing to run at least medium size system in a range where the network behavior is still acceptable. This makes the Beowulf cluster a valid support platform where to run small/medium size applications.

Concerning Classical MD codes the target machine is Origin-2000: the ratio in performances between Beowulf cluster and Origin-2000 is 1.65 -1.83 and maintains almost the same values also for the parallel case; this is a good result that makes the Beowulf machine a interesting computational resource for this kind of code with a excellent performance/price ratio.

Scalar and parallel performances obtained with the QMC code are the same due to the embarrassing parallelism of the code itself. The overall performances obtained confirm that this family of codes is perfect to be run on a Beowulf cluster.

Beside this observation there are two other issues we want to discuss:

- *mathematical libraries* : this is the weakest point of the Intel architecture: the lack of high performance mathematical libraries for the linux environment is a severe limitation for some kind of codes which could not be run efficiently. Our investigation is still uncompleted from this point of view: interesting projects like the ATLAS one can help to overcome this serious limitation but still we are far from what one can obtain on RISC processors.

- *Network behavior* The relatively high cost network we choose for our Beowulf cluster gives us good performances and makes our parallel system available for parallel tasks otherwise impossible with other solutions. It was recently announced a new myrinet board that can significantly improve network performances. This could overcome network limitations discussed above.

6 Acknowledgements

We thank C. Cavazzoni to help us testing his FPMD code on our cluster and G. Settanni for allowing us to test his system simulated using Amber 5.0. S. Fantoni is aknowledged for providing us his QMC code. This work benefits from a grant for applied research from FVG region.

References

[1] S. Cozzini and A. Dal Corso, Proceedings of the 5th SGI-MMP workshop, Bologna 1999

[2] C. Cavazzoni and G .L. Chiarotti, Comp. Physics Reports, **123** (1999) 56-76

[3] Melchionna, S. and Cozzini, S., DLPROTEIN_2.0 User Guide

[4] D.A. Case, D.A. Pearlman, J.W. Caldwell, T.E. Cheatham III, W.S. Ross, C.L. Simmerling, T.A. Darden, K.M. Merz, R.V. Stanton, A.L. Cheng, J.J. Vincent, M. Crowley, D.M. Ferguson, R.J. Radmer, G.L. Seibel, U.C. Singh, P.K. Weiner and P.A. Kollman (1997), AMBER 5, University of California, San Francisco.

[5] E. R. Davidson, Journal of Comput. Phys. **17**, 87 (1975).

[6] Smith, W., Forester, T.R. , "The DlPoly 2.0 User Manual", T.R.Forester and W.Smith, CCLRC, Daresbury Laboratory, Daresbury, Warrington WA4 4AD, England (1995).