

Benchmarking production codes on Linux clusters: the Sissa case study

Stefano Cozzini
INFN udr SISSA I-34014 Trieste (Italy)

Abstract

In the Condensed Matter group at Sissa we routinely use a certain number of parallel codes to perform scientific research in computational condensed matter physics. In this work we present and discuss the performance of some of these programs on different kinds of Linux clusters at our disposal. A first machine is an Intel based cluster using Myrinet networks built here at Sissa. Other clusters are based on alpha processors with two different kind of interconnections: Myrinet and QsNet. A careful performance analysis was carried out in terms of the computational characteristics of the benchmarking codes. Performance results on Linux clusters are then compared with data obtained on other parallel machines available. The list includes Origin3000, T3E and SP3. At the light of all the results we collected we are now able to draw some indications for future acquisition of computational power.

1 Introduction

The condensed matter sector at SISSA (International School of Advanced Study) has a long and outstanding experience in the field of numerical simulations. Research groups at Sissa are involved in very large computational projects: we remember here for instance the simulations of biological interest: active sites of proteins are studied by means of classical and quantum methods. This kind of numerical simulation requires huge amount of computer resources in term of RAM and CPU time. For this reason, in the last years, a certain number of parallel codes were developed in house in order to exploit at best the computational power offered by modern massively parallel computers. These codes now implement efficiently all the modern computational techniques and are routinely used as tools for scientific research. At the moment Sissa researchers have at disposal computational time on local parallel platform (a 16 node Origin 2000) and on massively parallel computers hosted at the Cineca Supercomputing Center in Bologna (Italy). There we can share resources on the following platforms: a 256-PE T3E , a 128 PE IBM-SP3 and a 64 PE Origin 3800.

In our continuous effort to find computational resources we could not remain indifferent toward Linux Cluster solutions. The availability of high speed network and increasingly powerful commodity microprocessors are making these platforms an attractive resource for our computational projects. To investigate the potential of this class of system we recently started some experiments assembling and evaluating a small prototype. The goal was to test different interconnect solutions and to evaluate which class of computational tasks this type of machines can solve in a cost-effective way. Performance results on our small cluster were already presented elsewhere [1, 2] and now this platform is a production facility. We are planning to expand and to improve this resource and in this last period we evaluated other cluster implementations. We recently have the chance to test a Compaq Linux cluster based on Alpha processors and an Alpha based Linux machine equipped with QsNet network. On all these machines we run a suite of parallel codes in order to assess the likely delivered performance across the full spectrum of our applications. The goal of this paper is to report and discuss performance results obtained on different Linux clusters.

The paper is organized in the following way: in section 2 are illustrated the technical details of the clusters at our disposal. In section 3 we introduce and discuss the computational characteristics of our parallel codes. A detailed analysis of the performances of two of them is then shown in section 4. A final discussion is presented in section 5.

2 The Linux machines

In this section we describe briefly the Linux platforms we tested in this work. Table 1 compares hardware and software features of the three clusters considered; a presentation of the Sissa cluster is here reported, while the other clusters, which we access as users, are described in a very concise way.

| Systems | Intel Beowulf | QSW Linux | Compaq cluster |
|-----------------|------------------------|----------------------|----------------------|
| processor | Intel PIII 550/933 MHz | Alpha 21264 667Mhz | Alpha 21264 616Mhz |
| N of processors | 8/4 | 16 | 8 |
| DRAM (total) | 2 Gbyte | 16 Gbyte | 4 Gbyte |
| Secondary cache | 512Kbyte | 4 Mbyte (?) | 2 Mbyte |
| Software | | | |
| O.S | Linux 2.2.14smp | Alpha Linux (2.2.17) | Alpha Linux (2.2.12) |
| Compilers | PGI 3.2.3 | Compaq compiler | Compaq compiler |
| Comm. Library | MPICH over GM | Elan MPI | MPICH over GM |
| Math Library | ATLAS 3.1 | CMXL | CMXL |

Table 1: A comparison among the cluster employed in this study

2.1 SISSA Intel Cluster

2.1.1 Hardware configuration

The local Intel Cluster was build at the beginning of 2000 in the framework of a joint SISSA-ICTP (International Center of Theoretical Physics) project granted by Friuli-Venezia Giulia region. A schematic view of the cluster is presented in fig 1. The cluster was originally 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 there are 2

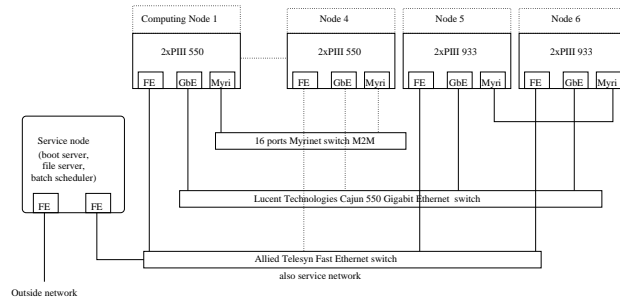


Figure 1: Schematic view of the Beowulf cluster with indicated the network interconnections installed.

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.

Recently (fall 2000) we added two other computing nodes with the goal to test state of the art hardware; the computing nodes are configured as follows: chipset INTEL 840 with two Pentium III (Coppermine) processors running at 933 Mhz. The processor FSB runs at 133Mhz and the memories are RAMBUS. The PCI bus runs at 66Mhz/64 bits. On the PCI buses we mount two Myrinet cards of last generation (Myrinet 2000) cross connected between them (i.e. without the switch). This configuration allows to measure the real performance of the new Myrinet Cards.

2.1.2 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 and we use the Fortran compilers available from the Portland Group (PGI).

2.1.3 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.

As high performance interconnect we installed Myrinet and Gigabit Ethernet. We carefully tested these solutions using different communication protocols [2, 3].

Our results indicate the Myrinet based communication systems is the best solutions in term of performance and stability. So now our cluster uses GM protocol over Myrinet and in the rest of this paper all the results obtained on the Intel cluster refer to this choice.

2.2 The QSW Alpha cluster

The QSW cluster we test in this work was recently installed at the Cineca Center. It is comprised of 8 API UP2000 boards, where each node is made up by 2 Alpha 21264A CPUs (667 Mhz), 1 GB RAM memory and 4 MB L2 Cache. In addition to the 8 nodes, the cluster has also a single CPU controller node, which acts as the system interface with respect to the external world. The nodes are linked by means of 1 Fast Ethernet networks, dedicated to general system services. QSW proprietary interconnection technology, fat-tree QsNet is reserved for Message Passing data exchange in parallel applications. Each node runs Linux Red Hat ver. 6.1 operating system with kernel 2.2.17.

QsNet is a high bandwidth, ultra low latency interconnect for high performance scalable systems. QsNet interfaces to the host computer through the industry standard PCI bus. The network interface is based on Quadrics' third generation "Elan" ASIC. Elan III is designed to offload the entire task of inter-processor communication from the main processor, and to avoid the overhead of system calls for user process to user process messaging. Data links from each node are connected using Quadrics switch: QM-S16 with 16 ports; Internally the switch is a multistage switch, based on the Elite III 8-way switch component. This is connected in a "fat tree" topology.

2.3 The Compaq cluster

The Compaq cluster was made available to us by Compaq Europe in Annecy in December 2000. It was composed by 8 node: the computing nodes were equipped with 1 single Alpha CPU 21264 at 616 Mhz running Linux alpha with kernel 2.2.12. Nodes are connected trough high performance Myrinet network. The communication protocol is MPICH/GM, the same used on our Intel cluster. In this case the BUS PCI where each Myrinet card is installed runs at 66Mhz/64 bits.

2.4 Network Performance results

A comparison of the interconnects on the machines has been performed by means of a standard benchmark suite available from Pallas [4]. The goal is to collect some indications in order to estimate which kind of performances we can expect from our parallel codes.

Table 2.4 reports the different values of latency and bandwidth measured over the MPI protocol for all the parallel machine considered in this study. Latency is expressed in microseconds and it is defined as the time needed to send zero length packets in simple ping pong test. The MPI bandwidth is roughly estimated using the same test; the bandwidth attained in this benchmark is also shown in figure 2 as a function of message size.

We show here results obtained with different network solution on the Linux cluster: in particular we include here, as reference, the data obtained on TCP protocol over fast Ethernet.

The data collected indicates that high performance network solutions on Linux clusters (QsNet and Myrinet) are comparable (if not superior) with network performance obtained on full custom hardware.

| Platform | Latency(ms) | Bandwidth (Mb/sec) |
|-------------------------------|-------------|--------------------|
| MPICH/TCP over 100Mb (Intel) | 172 | 12 |
| MPICH/GM over Myrinet (Intel) | 15 | 114 |
| MPICH/GM over Myrinet (Alpha) | 18 | 116 |
| MPICH/GM over Myrinet2(Intel) | 10 | 147 |
| QsNet (Alpha) | 15 | 161 |
| T3E | 15 | 152 |
| SP3 | 12 | 234 |
| O3K | 10 | 124 |

Table 2: Performance of standard MPI communications on parallel systems available.

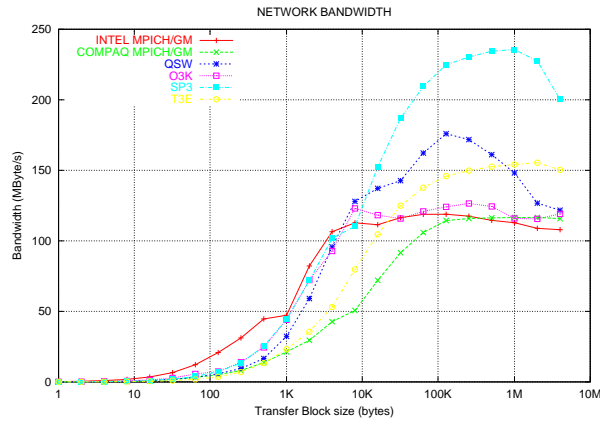


Figure 2: Network bandwidth measured by MPI.

Another network parameter we are interested in is the performance of the reduce operations. These are fundamental operations for some of our codes, and it is therefore important to know which kind of behavior one should expect. Figure 3 show times measured for an All-reduce operation using 8 processors. Performances of QsNet network are again excellent: the QSW data roughly overlap with custom hardware ones. On the contrary the Myrinet network show some limitations: for small and large sizes of data the performance are in the same range of other machines. There is however a range (10K-100K) over which performance decreases dramatically. We observed the bad behavior on either the Intel cluster or the Compaq one. This behavior could limit the performance of our codes as it will be explained later.

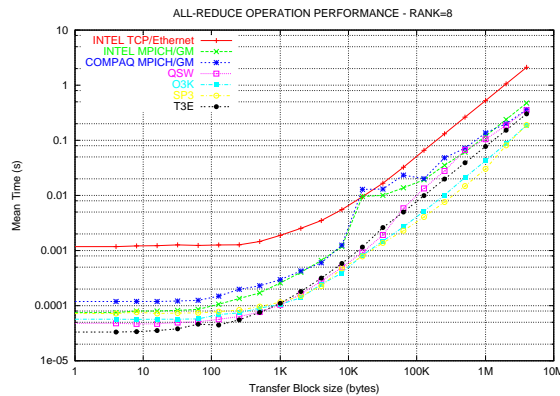


Figure 3: times vs size for MPI reduce operation. 8 processors are used.

3 The parallel codes

In this section we want to discuss the computational requirements we have in using the different classes of parallel codes. These codes can be classified in three main classes:

- First-principles or ab-initio packages 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) packages, which allow to simulate systems composed of several thousand of atoms like for instance molecules of biological interest.
- Quantum Monte Carlo (QMC) programs which are used to study strongly correlated systems.

Codes belonging to different classes are different also from the computational point of view. Table 3 presents a comparison of the three families of codes looking at specific aspects of High Performance Computing (HPC). The columns represents the three classes of codes while each row indicates a peculiar characteristic of the HPC. 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.

Let us briefly comment on this table:

- Ab-initio codes are computationally demanding in term of memory. Parallel versions of electronic structure codes have been developed in the past years using the Message Passing (MP) paradigm to cope with this problem. Due to this effort, highly scalable programs are now available showing good speed-up over a wide range of processors (64/128) These codes make large use of linear algebra kernels and therefore it is fundamental have at disposal highly optimized libraries. The data access is almost always consecutive allowing to exploit at best cache memories hierarchies.
- Classical Molecular Dynamics codes have different computational characteristics: the memory requirements are lower and the scalability is often limited over a small number of processors. Moreover they do not use any specific linear algebra kernel and the data access in the computational core is very scattered and therefore not very cache friendly.
- The codes implementing Quantum Monte Carlo techniques do not need too much dynamical memory. Their Parallelization strategy is embarrassing: each processor works on its own task and communications take place only at the beginning and at the end of the computation and therefore communication does not represent a bottle neck. They use some linear algebra routines but in general this is not the core of the computation.

To help us in deciding which is the best solution for specific computational tasks a suite of representative codes was set-up and by means of them we are currently testing different parallel platforms. In this work we present performances results obtained on two codes of this suite: the first is the PWSCF program, a parallel code belonging to ab-initio class and entirely developed at Sissa; the second one is a representative code of classical MD class: DLPROTEIN_2.1, a MD package to simulate biomolecules. The next two sections give a short description of both codes illustrating the computational load and the parallelization aspects.

3.1 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-SP23,T3E and Origin-3/2000). 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. In the following we briefly outline the computational load of the program. We refer to reference [5] for further technical details about the code.

In the code the wave-functions 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 wave-functions from the reciprocal space to the direct space.

The most time consuming step of the program is the computation of the wave-functions 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.

The CG method requires several applications of FFT procedures and a large number of vector-vector operations. These linear algebra operations are accomplished by means of BLAS1 routines.

In the Davidson method we identify other computational tasks: a large amount of vector-matrix and matrix-matrix operations is required. These operations are performed by BLAS2 and BLAS3 routines. There are then several FFT calls and (quite important) a diagonalization procedure: a matrix of variable dimension is diagonalized several times using appropriate LAPACK routines for generalized eigenvalue problem.

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. 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.

3.2 Classic MD code: DLPROTEIN_2.1

DLPROTEIN_2.1 [6] is a Molecular Dynamics package well suited for biological molecules. It is written by S. Melchionna and S. Cozzini and it is a development of the original general purpose Molecular Dynamics code DL_POLY_2.0 [7] written at Daresbury Lab (UK) by W. Smith and T. R. Forester. DLPROTEIN_2.1 allows to simulate systems in different thermodynamic ensembles 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.

The parallelization strategy adopted by DLPROTEIN_2.1 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 $N_a \times \ln PE$. 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 $PE \times \ln PE$ 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.

4 Results

In this section we present the performance results obtained running our two codes on the three Linux clusters discussed before. Data collected are also compared against the performances obtained on the other parallel machines available: T3E and Origin-3800 and IBM-SP3.

For each of the two codes a certain number of tests was prepared and then run on the 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 5%.

4.1 Ab-initio code: PWSCF

The small configuration of our INTEL cluster does not permit to run the largest production systems currently in production on SP3 and T3E facilities. We are able however to run specific "ad hoc" examples and some medium size systems that are under study. We present here two different set of measurements. The first set is related to typical medium size test: in this case we discuss in details all the performances obtained. The second set refers to an actual system under production; we report here just the times obtained on production runs on the parallel resources available locally.

The test system is a C_{60} molecule and on this example we perform only a single step of the full self-consistency procedure. The test was executed using the two procedures the code offers: Conjugated Gradients and Davidson. The system is described by a small number of plane waves : 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 in the range 2-16 PE where possible.

We start discussing the parallel behavior of the code looking at the performances obtained on the same number of processors (8) on different Linux clusters and on SP3 machine. SP3 machine is here considered because it is in this moment the target machine for large production runs.

Table 4 show the data for both procedures: Conjugate Gradient and Davidson.

In the table the main routines (electrons) are reported together with global times for the FFT procedures and specific sections of the code where Lapack and BLAS routines are used. The FFT times are split in two component: the scalar one (where FFT are performed on single node) and the `fft_scatter` where data are communicated between nodes. Times for the other main communication operation the algorithms perform (a reduce one) are also reported.

| Conjugate gradient [8 Processors] | | | | |
|-----------------------------------|-----------|-----|--------|-----|
| procedure | INTEL 550 | QSW | Compaq | SP3 |
| electrons | 619 | 208 | 180 | 114 |
| fft (scalar) | 214 | 60 | 55 | 56 |
| blas1 (s_1psi) | 76 | 24 | 19 | 11 |
| blas1 (add_1vuspsi) | 58 | 15 | 12 | 7 |
| fft scatter | 109 | 54 | 53 | 21 |
| reduce (20037 calls) | 33 | 21 | 18 | 9 |
| Davidson [8 Processors] | | | | |
| procedure | Intel 550 | QSW | Compaq | SP3 |
| electrons | 659 | 261 | 236 | 158 |
| fft (scalar) | 136 | 41 | 37 | 27 |
| diagonalization | 173 | 52 | 59 | 49 |
| blas3 (overlap) | 68 | 20 | 26 | 14 |
| blas2 (update) | 114 | 79 | 59 | 37 |
| blas3+blas1(last) | 35 | 12 | 13 | 10 |
| fft scatter | 75 | 32 | 35 | 11 |
| reduce (82 calls) | 17 | 26 | 26 | 5 |

Table 4: CPU times for self consistent procedures for C_{60} molecules. Times are in seconds.

Considering the total times we see that RISC architectures are outperforming the INTEL CPU. It is not surprising that Linear Algebra and fft kernels on the almost up to date RISC architecture provide a much higher performance compared with our relatively old Pentium 550Mhz CPU (from to 2.5 to 5 times slower). The RISC processors benefit of larger and faster cache hierarchy than the Intel ones. Moreover RISC processors are using vendor's highly optimized libraries. It is worth to mention here that the gap between INTEL and RISC processor still remains large using the more recent 933 Mhz processors. We report in the table 5 the behavior of some sections of the code on the two different INTEL CPU. The increase in performance varies significantly among the different sections giving an overall gain of about 25%.

| Davidson +CG [4 PE] | | | |
|---------------------|-----------|-----------|-------|
| procedure | Intel 550 | Intel 933 | ratio |
| fft | 240 | 180 | +33% |
| diagonalization | 169 | 113 | +50% |
| blas3 (overlap) | 116 | 133 | -14% |
| blas2 (update) | 168 | 124 | +35% |
| blas3+blas1(last) | 68 | 44 | +54% |
| fft | 380 | 290 | +31% |
| blas1 (add_1vuspsi) | 107 | 83 | +28% |

Table 5: CPU times for specific sections on different INTEL CPU. Times are in seconds.

We can compare now these kernels (linear algebra and fft) on Linux alpha clusters and on the SP3. The performances on the Alpha clusters are roughly the same for almost all sections where fft and linear algebra kernels are heavily used. Significant differences are only reported in the "update" routine where BLAS2 kernel are performed; in particular Alpha processors mounted on QSW machine deliver almost a 25 percent less in performance than the same processors mounted on a Compaq machine: in this case anyway both solutions are outperformed by IBM Power 3.

We can turn now on network performances looking at the two fundamental communication operations the code performs: `fft_scatter` and a collective reduce.

The `fft_scatter` routine performs point to point communication in the fft algorithm and therefore the performance of this operation should be aligned with the bandwidth figures we report before (see figure 2). One should expect that Myrinet network will provide the almost the same

performances on both the Intel and Compaq clusters. This is not true: the performance on the Alpha cluster is much higher than on the Intel one. Our guess to explain this behavior is that on the Intel cluster we are saturating the PCI bandwidth while on the Alpha one the higher PCI bandwidth avoid this bottleneck. It is worth to note that on the Alpha machine there are no significant differences between Myrinet and QsNet networks, but SP3 network show excellent performances outperforming both the Linux solutions.

Let us discuss the reduce operation. There is a very large number of reduce operations in CG procedure, each of them transferring few data. We therefore have to refer at the first range of figure 3 in the case of CG algorithms. The performance obtained reflects roughly the features we got using standard benchmarks for all the platforms.

The situation is quite different for Davidson procedure: in this case there is limited number reduce operations, each of them transferring a large amount of data. In the case of the C60 molecule the data sizes are in 10-100Kbyte range where for both the Myrinet clusters the reduce operations show some problems (see figure 3). The bad performances obtained on Myrinet on Davidson procedure are therefore clear. It is on the contrary not clear the bad performance we got on the QsNet: in this case one should expect higher performances than Myrinet but here the performance are even worst that the Intel cluster.

| Architecture | FFT (CG/DAV) | Reduce DAV | Reduce CG |
|--------------|--------------|------------|-----------|
| Intel | 1 | 1. | 1 |
| QSW | 2.3 | 0.7 | 2.3 |
| Compaq | 2.1 | 0.7 | 2.8 |
| SP3 | 5.6 | 3.4 | 5.6 |

Table 6: Normalized figure for communication operations for C_{60} molecules

Table 6 summarize the comparison of the the network performance obtained on the architecture considered. Times are all normalized to the Intel ones. It is evident that SP3 outperforms clearly all the other solutions.

We can look now at the scalability of this test. Table 7 gives the global time to run this example over all the machines we tested it.

| Platform | 2 | 4 | 8 | 16 |
|--------------------|-----|-----|-----|-----|
| Davidson | | | | |
| T3E | - | - | 317 | 177 |
| INTEL | - | 937 | 665 | - |
| Compaq | - | 388 | 236 | - |
| QSW | 812 | 430 | 261 | - |
| SP3 | 505 | 271 | 154 | 112 |
| Conjugate Gradient | | | | |
| T3E | - | 687 | 346 | 187 |
| INTEL (550) | - | 997 | 619 | - |
| Compaq | - | 381 | 181 | - |
| QSW | 841 | 404 | 208 | - |
| SP3 | 550 | 259 | 111 | 58 |

Table 7: Scalability of PWSCF for C_{60} molecules. Times are in seconds.

Scalability is excellent for all modern RISC platforms in the case of the CG algorithm: as a matter of fact we observed super-linear speed-up in some ranges. For Intel and T3E architecture scalability is still good but not excellent. The reason of the super-linear speed up is due to a cache effect: it is well known that the increase in the number of processors allows to exploit better the cache hierarchy. On modern RISC processors with large secondary cache, this effect is quite spectacular. On the contrary on relatively old processors (T3E and Intel) with very small secondary caches the effect is practically absent. The super-linear effect on the SP3 machine

(where the L2 cache is 8Mb) is more pronounced than on the Alpha platforms (equipped with a 4Mb L2).

The scalability of the Davidson procedure is limited by a factor common to all the architecture apart from the T3E: the diagonalization procedure is not performed in parallel. Moreover for the Alpha clusters another strong limitation comes from the bad performance on the reduce operations as already discussed.

As second example we discuss here the behavior of one system under study. The system (a faylite mineral) is a medium size system and it is currently simulated on our local Origin 2000 (R10000 processors). We want to check how feasible is to move the production runs from that overloaded resource to to our dedicated INTEL cluster resource. We run a complete self consistent procedure with the Davidson procedure while for the CG method we just stop the simulation after 3 iterations. Table 8 summarize the execution times we measured.

| Architecture | Davidson (full SCF) | Conjugate Gradient (3 steps) |
|----------------------|---------------------|------------------------------|
| Origin 2000 (R10000) | 39323 | 22390 |
| Intel Cluster | 42016 | 14579 |

Table 8: Execution times for faylite system. Times are in seconds

Results show that, despite the limited performances we got compared on state of the art custom hardware (i.e. SP3), our small, relatively old Intel cluster can deliver interesting performances when compared with a local parallel resource at our disposal.

4.1.1 Classical MD: DLPROTEIN_2.1

For the DLPROTEIN_2.1 program we show performances on three different tests that are actual systems under study. The comparison among computational platforms is therefore complete and significant for the computational requirements of this class of codes. Test 1 refers to a system composed by two identical proteins (1347 atoms each) solvated with 5494 water molecules. The second test is a system containing 45 Micelle (each of one formed by 80 atoms) solvated with 8513 water molecules. The third test is comprised of two proteins solvated with a 17824 water molecules. All the systems have an high degree of connectivity with a large number of constraints. The values of the significant parameters of the tests are collected in table 9.

| Parameter | Test 1 | Test 2 | Test 3 |
|----------------------|---------------|--------------|---------------|
| N_{atoms} | 19176 | 32829 | 58701 |
| N of waters | 5494 | 8513 | 17824 |
| averaged N_{neigh} | ≈ 200 | ≈ 50 | ≈ 260 |

Table 9: Input parameters for test cases of the DLPROTEIN_2.1 code.

The tree tests are executed in the 1-8 range: this allow a detailed analysis of scalar performance on different CPUs and these results are presented in the next section.

4.1.2 Single Processor performances

Table 10 collects scalar performances for the three tests obtained on different architectures. We do not have any data for the Compaq machine because the present release of the code was not available when we tested that platform. We report for each test the global time and time spent in the the two most consuming tasks of the code: short range and long range computations. Times are referred to single iteration and are averaged over 100 timesteps. The short range computation is split here in the two sub-tasks: building the neighbor list and the actual computation of short ranges using the list. The two sub-tasks 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, i.e. in not a “cache friendly” way.

There are several observations that can be made:

- Intel 933Mhz processor is delivering global performances in the range of the RISC processors; RISC processors are still faster than INTEL one but the gap reduces greatly compared with ab-initio codes.
- the improvement in performance between 550Mhz processor and 933Mhz one is impressive: the increase is about 80%; it is worth to note that the increase is greater on the short range section where computation does not benefit from cache: in this case fast RAMBUS memory plays a crucial role.
- building lists is an operation where INTEL processors show excellent performance: times, practically the same obtained by alpha, are much better than R12000; this is probably due to the excellent performances of the INTEL architecture on integers.
- in the short range computations, where data are accessed in a very scattered way, RISC processors with large L2 caches show higher performances for test 1 and test 2 when compared with 933Mhz Intel. The gap in performance disappear in the third examples: this is likely due to the fact that data structure are too large in this case even for the large L2 lines and therefore the number of cache misses should be roughly the same on all the platforms.
- the SPME algorithm based on FFT procedures shows on the contrary the same trend on all the tests: RISC processors are delivering much higher performances than INTEL. The RISC architectures fit well with the type of computational load the task needs: optimized FFT routines and consecutive access to data.

| Section | INTEL 550 | Intel 933 | EV 667(QSW) | R12000 (O3K) |
|-------------------|-----------|-----------|-------------|--------------|
| Test 1 | | | | |
| Built Link List | 0.55 | 0.31 | 0.30 | 0.53 |
| Short Range | 4.67 | 2.53 | 1.82 | 2.33 |
| Long Range (SPME) | 3.19 | 1.83 | 0.86 | 1.13 |
| Total | 9.50 | 5.22 | 3.43 | 4.59 |
| Test 2 | | | | |
| Built Link List | 0.90 | 0.49 | 0.48 | 0.95 |
| Short Range | 8.69 | 4.78 | 3.26 | 3.56 |
| Long Range (SPME) | 8.38 | 4.94 | 2.25 | 3.70 |
| Total | 20.17 | 11.29 | 6.69 | 9.36 |
| Test 3 | | | | |
| Built Link List | 1.12 | 0.61 | 0.59 | 1.24 |
| Short Range | 18.10 | 9.21 | 10.56 | 9.24 |
| Long Range (SPME) | 10.60 | 6.12 | 3.63 | 4.68 |
| Total | 33.86 | 17.87 | 16.40 | 17.10 |

Table 10: Single processors performance of the test cases for the DLPROTEIN_2.1 code: times are in seconds and refer to one iteration.

4.1.3 Parallel performances

We now discuss parallel performances; as already explained the RD algorithm adopted by this code does not scale: moreover systems tested here posses complex topologies and constraints that require an high communication overhead.

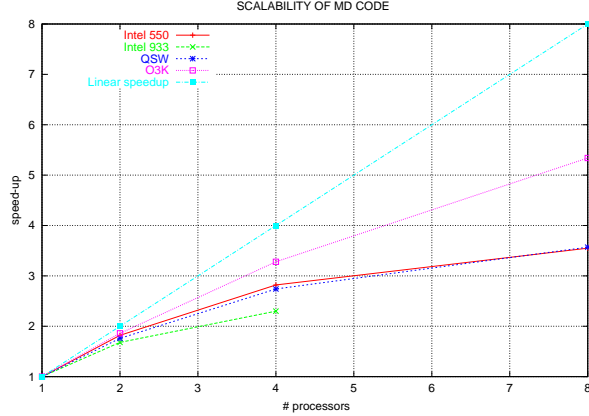


Figure 4: The speed-up for the test 2.

Figure 4 spots out the overall scalability for Test 2 in the range 1-8 processors; parallel behavior is practically the same for Linux platforms with a significant deviations from ideal behavior. We note however that for O3K platform the situation is better.

This reason of this poor scalability is easily found by looking at ratio between communication time and total computational time.

| PE | O3K | | | QSW | | | INTEL | | |
|----|-------|-------|----|-------|-------|----|-------|------|----|
| | Total | Comm. | % | Total | Comm. | % | Total | Comm | % |
| 2 | 250 | 8 | 3 | 200 | 12 | 6 | 545 | 21 | 4 |
| 4 | 143 | 15 | 10 | 126 | 28 | 22 | 320 | 45 | 14 |
| 8 | 88 | 19 | 22 | 95 | 42 | 44 | 242 | 81 | 33 |

Table 11: Total communication time and the percentage of the communication time over the total time for Test1. Times are in seconds

This is reported in table 11 for Test 1: the percentage of time spent in communication grows rapidly with the number of processors on the various platforms. This reach even 44% in the case of QSW cluster, indicating that global network performance of the QsNet are quite bad compared with the O3K ones.

The communication time reported in the previous table is given by the two main global operations already discussed (see 3.2): the reduce operation (gdsum) that reduces the force and the velocities values at each time-step and all_gather (merge) where new positions are updated. A detailed comparison on these operations again for Test 1 are reported in table 12.

| PE | merge | | | reduce | | |
|----|----------------|-----|-----|----------------|------|------|
| | Intel 550(933) | QSW | O3K | Intel 550(933) | QSW | O3K |
| 2 | 6.9 (4.7) | 2.0 | 2.9 | 12.8(10.2) | 8.6 | 4.4 |
| 4 | 13.1(11.7) | 4.0 | 4.1 | 28.2(25.1) | 22.7 | 9.6 |
| 8 | 27.0 | 4.2 | 5.7 | 44.7 | 22.5 | 11.9 |

Table 12: Times for reduce and merge operations. Times are in seconds

The table allows to compare closely the O3K and the QsNet performances: O3K network is delivering a much higher performance on reduce operations than QsNet while on the merge operation both networks perform at same level. The Myrinet solution on the Intel shows some strong limitation on both the operations. We are again hitting here the limit already seen in the PWSCF code: PCI bus saturation.

A surprising fact emerges however from the data; in the QSW case the sum of the merge and

the reduce operations performed using 8 PE is much less than the global communication time reported in table 11. This means that some other communication operations take a significant amount of time on that platform when we run at 8 processors. This feature is also present for the other two tests; Looking at the data we indeed spot out that the synchronization routines are responsible of this: why this happens is not clear for us.

A final comparison can be done between Myrinet cards installed on the 550Mhz nodes and the more recent ones installed on 933Mhz nodes. We can see from table 12 that network performance improves but the gain is much slower than scalar one. Therefore an increasing of computational power of the scalar INTEL processor causes a decreasing of the overall scalability because the communication time weights more and more. This is of course a limiting factor for the cluster that can be hopefully overcome with the new Myrinet switch.

As final point we remark that this analysis show clearly the limits of the RD algorithm on Linux cluster (not so evident on SMP machine like O3K for instance). This stimulated us to start developing a new parallel version of the code using a better parallel algorithms to improve performances on Linux Cluster.

5 Discussion

In this final section we want to summarize the results shown before and discuss about the use of such platforms in our computational environment.

Firstly we discuss about the ab-initio class of codes. In this case the results from previous sections can be summarized as follows:

- Intel platforms are inadequate: the gap in performance between almost state of the art processor and custom solutions we have at disposal is too large (3/4 times). Moreover the Intel system shows strong limitation in the interconnects, indicating that increasing the number of nodes in order to simulate larger systems is not possible.
- Alpha system overcome the INTEL limitations. The processor itself is delivering excellent performance. Still we noted some limitations on both the networks interconnection we tested. In particular the QsNet solution seems (at least on the platform we tested) not to deliver its full potential on our applications. This makes the SP3 solution outperforms the Alpha solution.

At the light of these statements we consider the Linux cluster solution not able in this moment to fulfill all our computational requirements for the ab-initio class of codes. As already pointed out the small Intel resource can treat efficiently or medium size problems at good performance/price ratio but a large Intel facility should not be a good choice. QSW clusters, in our humble opinion, seems to be still an immature platform, showing some limitations in stability and overall performances. Moreover network cost makes this configuration not really cost effective.

Let us discuss the second class of codes. The situation is quite different from the previous case. Classical MD codes like the ones we use (DLPROTEIN and Amber) are well suited to run on SMP platforms and therefore small INTEL cluster seem to be a good choice. A severe limitation for this class of codes is represented by the network performances; the parallel algorithm implemented requires high performance communication hardware to be able to deliver useful speed-ups on this clusters. This is indeed confirmed by the results we got. Anyway, despite the limited scalability (common to all the platforms) we show that state of the art commodity processors (like INTEL and maybe AMD processors) deliver excellent performance and make PC clusters cost-effective platforms. Moreover we hope that new promising Intel/AMD processors (Pentium IV, Athlon...) and new high performances devices will overcome some of the limit we run into. More expensive solution like the QSW one are not outperforming the INTEL ones and therefore make them not feasible from the point of performance/price ratio.

Finally let us discuss the third class of codes, currently used at Sissa: QMC codes. We do not present any performances about QMC codes in this work. This is because the embarrassing parallelism of the codes itself should not have revealed any interesting features of the parallel

behavior of the platform. For such a family of codes the only parameter to look at is the scalar performance on the CPU. As we already spot out [1] the INTEL CPUs are in this case the most appropriate in term of performance/price: this is because an increasing the frequency of the processor our codes increases by almost the same factor their performances. It is well known that the best platform for such class of code is a real commodity PC clusters with commodity communication hardware and with no limitation in the number of nodes.

What can we learn from this ? Firstly it is clear that a single Linux cluster solution won't fit all our computational requirements. To fit at some extent the computational need of each single class of codes one should plan and build at least three different Linux clusters. This is clearly not possible for our institute. Nevertheless we see from the everyday use of our small INTEL cluster that the cost-effectiveness of such a solution make this resource a fundamental one which deal efficiently with a relatively large number of different computational requirements.

6 Acknowledgments

I want to thanks Compaq Europe and A. Galli and G. Bortolami to give us the chance to test its Linux cluster and for the pleasant stay at Annecy. Carlo Cavazzoni and Stefano Martinelli are warmly acknowledged for their precious help in exploiting at best all the Cineca platforms and for interesting discussions. This work benefits from a grant for applied research from FVG region. Finally I thank my daughter Marta to allow me to write this paper before the deadline.

References

- [1] Cozzini, S., Innocente, R. & Corbatto, M. Comparing scientific code on different parallel platforms, Proc. of 6th SGI-MPP workshop: Manchester 2000
- [2] Cozzini, S., Innocente, R. & Corbatto, M. A PC cluster with high speed network interconnects. Proc. of CAPI conference: Milan 2000
- [3] For a detailed discussion of all the results see: <http://hpc.sissa.it>
- [4] <http://www.pallas.com/pages/pmbd.htm>
- [5] Cozzini, S., & Dal Corso, A. Optimization of a plane waves self consistent code (PWSCF) on SGI platforms. Proc. of the 5th SGI-MMP workshop, Bologna 1999
- [6] Melchionna, S. & Cozzini, S., DLPROTEIN_2.1 User Guide (<http://www.sissa.it/cm/DLPROTEIN>)
- [7] Smith, W., & Forester, T.R. , "The DiPoly 2.0 User Manual", T.R.Forester and W.Smith, CCLRC, Daresbury Laboratory, Daresbury, Warrington WA4 4AD, England (1995).