

Benchmark Analysis of 64-bit Servers for Linux Clusters for Application in Molecular Modeling and Atomistic Simulations.

Stefano Cozzini¹, Axel Kohlmeyer², and Roger Rousseau³

¹ Democritos,

2-4 Via Beirut, 34014 Trieste, Italy

² Center for Molecular Modeling, University of Pennsylvania,
213 South 34th Street, Philadelphia, PA 19104, USA

³ International School For Advanced Studies (ISAS/SISSA),
2-4 Via Beirut, 34014 Trieste, Italy

Abstract. We present a detailed comparison of the performance of synthetic test programs such as DGEMM and STREAM as well as, typical atomistic simulation codes DLPROTEIN and CPMD which are extensively used in computational physics, chemistry and biology on a large number of high performance computing platforms. With an eye toward maximizing the price / performance ratio for applications in atomistic simulations, we examine a wide class of commonly used 64-bit machines and discuss our results in terms of the various aspects of the machine architecture such as CPU speed, SMP memory performance and network. We find that although the INTEL EM64T machines show superior performance for applications which extensively exploit the MKL library the OPTERON based machines show superior performance with less optimized codes. Moreover, for large memory applications such as electronic structure codes, the SMP performance of the OPTERON is superior. An overview of which architecture is suitable for which applications and a comparison of AMD dual core CPU technology to INTEL hyper-threading are also discussed.

1 Introduction

Large scale computer modeling of molecules, liquids and solids, denoted atomistic simulations, is an active area of research in physics, chemistry, biology, pharmacy, engineering and materials science. As such, there is a tremendous demand for computational resources to perform these simulations as evidences by the fact that these types of calculations consume in excess of 50% of the CPU time on the majority of publicly available supercomputers. The Democritos National Simulation Center [1], and the Center for Molecular Modeling (U. of PA) [2], are both actively involved in the development and deployment of a large variety of codes in this software class with particular emphasis on massively parallel, high performance computing applications. A critical component of this research endeavor is to maintain an up-to-date knowledge of both hardware and software advances in order to allow us to maximally exploit the available computer power.

In this context, it is important to note that in addition to the large use of national computing facilities, atomistic simulation software are also extensively deployed on in-house mostly GNU/Linux based computing clusters. Hence, we are constantly evaluating state of the art hardware / software solutions currently on the market and we are particularly interested in which are the best building blocks for in-house *ad hoc* Linux Cluster solutions. In this report we will present a specific validation study performed recently on 64-bit servers available for cluster computing.

This study is performed using two classes of codes, that are commonly used for applications ranging from materials science to pharmaceutical research: (i) First-principles or *ab initio* electronic structure packages in particular the CPMD code [3]. (ii) Classical Molecular Dynamics (MD), specifically DLPROTEIN [4]. Both packages are extensively deployed on a variety of platforms and are representative of the types of codes employed by this large and diverse user community. Although often used by the same researchers, the two types of code have fundamentally different computational requirements ranging from large memory usage to efficient scalability over a given number of CPUs. It is important to stress that a common scenario encountered is that expansion of in-house computational resources is often *entirely* determined by funding of the projects of the research groups that normally use these applications. Hence, in addition to standard benchmarks such as LINPACK it is absolutely necessary to measure a given computational platform in terms of how it performs on these (and other) codes.

The paper is organized as follows: (i) In Sect. 2 we introduce and discuss the computational characteristics of the parallel codes considered in this study. (ii) In Sect. 3 we give a short overview of the several hardware/software solutions we used in this work and we characterize them in term of standard benchmarks. (iii) A detailed analysis of the performances of the codes presented in Sect. 2 is then presented in Sect. 4. (iv) A final discussion is presented in Sect. 5.

2 Computational Codes

Ab initio electronic structure codes solve from first principles the quantum mechanical equations of electronic states to obtain energies, forces and properties of a given system. In general, these codes are computationally demanding in term of memory and CPU usage. For most of these codes parallel versions exist with the better ones exhibiting good scalability over a wide range of processors of up to a few hundred in number, depending on the job size and network speed. These codes extensively employ linear algebra kernels and therefore it is fundamental to have highly optimized, hardware specific performance libraries available. Memory usage can be on the order of several GB where data access is frequently consecutive, allowing to exploit the best cache memories hierarchies. On the other hand, classical molecular dynamics codes, which use simplified descriptor of the energy and forces, have very different computational characteristics. Generally, the memory requirements are lower (10's of MB) and the

scalability is often limited over a small number of processors (ca. several tens of CPUs). Moreover, they do not use any specific linear algebra kernel and the data access in the computational core is often quite scattered and therefore not very “cache friendly”. Thus, machines with the largest number of general purpose registers and cache are often preferred for this type of application.

Our choice of the CPMD code as our representative *ab initio* code is based on the fact that it is highly optimized by making extensive use of external math libraries, and exhibits almost 99% CPU usage in serial implementation on most CPU architectures. We note that many *ab initio* codes also produce enormous quantities of *I/O*, reading/writing on the order of a few GB of scratch files; which tends to dominate the code performance as well as limiting the parallel scalability. In this respect, CPMD is ideal for testing memory and CPU performance as it does not produce scratch files hence, *I/O* does not play an important role in the performance. Finally, CPMD is based upon plane wave basis sets, which while being easily Fourier transformed to reciprocal space and back, need to be very large to be able to represent the electronic structure properly. Thus for the largest currently (2005) manageable problems as much as several 10s of GB of memory are required, which can only be efficiently handled with a distributed memory parallelization approach and a specialized interface to the available Fourier transform libraries (as well as a highly efficient and adapted generic FFT implementation [5]), which in turn leads to extensive amounts of MPI all-to-all communication and thus demands low-latency and high bandwidth networks for good parallel scaling.

The DLPROTEIN_2.1 [4] Molecular Dynamics package is an ideal benchmarking tool for classical MD simulation techniques applied to biological system as all typical algorithms employed in this class of codes are present. From the computational point of view the decisive section of the code is the force computation, in particular the so-called non-bonded interactions between all pairs of atoms present in the system, which takes 80-90% of the total calculation. There are different techniques to treat the short and long range contributions to these interactions. The short range part is computed by means of the neighbor list technique, which is an array of pointers to the position arrays. This means that the data are loaded in a very scattered way leading to bad performance of the cache mechanism. Long Range forces are treated by means of the Smooth Particle Mesh Ewald (SPME) algorithm that requires FFT kernels on a 3D grid. The parallelization strategy adopted by DLPROTEIN_2.1 is based on the Replicated Data (RD) approach and implemented using MPI message passing. RD itself is not very scalable: in practice RD works most effectively in the range 1–8 processors and for benchmarking purposes this algorithm offers a precise evaluation of the interconnect, both in terms of latency and bandwidth. Moreover it is also an excellent test case for small SMP machines.

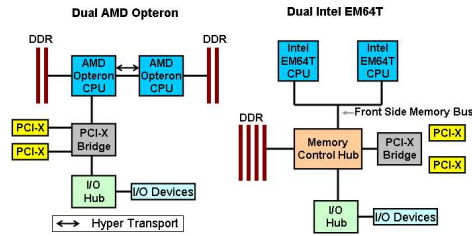


Fig. 1. Schematic representation of relevant architectural features of dual (a) AMD OPTERON and INTEL EM64T.

3 Hardware/Software Solutions

For atomistic simulations the large memory requirement of *ab initio* codes makes the 64-bit addressing technology very attractive, since even very large data sets can be managed with a simple flat memory model. In the case of distributed memory parallelization, this allows for much larger jobs to be done on fewer CPUs thus the maximum computational problem size is less limited by parallel scalability. This is especially true for codes which produce scratch files where the *I/O* is best done on local scratch space and scalability is often limited. The 64-bit requirement is not that necessary for classical MD codes, however, the superior cache and register structure of many 64-bit machines radically improves their performance. Hence our discussion will focus primarily on novel 64-bit architecture hardware solutions as well as software tools (compilers/libraries) installed in each platform benchmarked.

With these observations in mind we consider the available commodity CPUs on the market namely the AMD OPTERON, INTEL ITANIUM, INTEL XEON PIV and the extended memory 64-bit technology variant the INTEL XEON EM64T (NOCONA). We note that since our main concern is price/performance, we put only a small amount of attention on the INTEL ITANIUM CPU where the superior performance is offset by the substantial increase in cost. To frame the following discussion the relevant architectural features of the dual OPTERON and XEON/EM64T are presented in Fig. 1. A critical difference lies in the mechanism for accessing the main memory in the case of symmetric multi-processing (SMP). For the OPTERON each CPU is connected directly to its own banks of memory, with the memory controller directly integrated on the chip, and relies on a hyper-transport mechanism for tasks accessing memory outside of what is directly connected to a given CPU. Hence memory contention for the dual OPTERON will only play a role with the recently introduced dual core CPUs, i.e. when two physical cores share the same memory banks. Conversely, the XEON/EM64T provides shared memory access via an external memory controller and both CPUs use the same memory banks. As a result, memory contention for the XEON/EM64T, especially for *ab initio* codes, should become a serious performance issue and must be critically evaluated. INTEL and AMD also differ significantly in the way they provide extra CPUs on the same dual or quad moth-

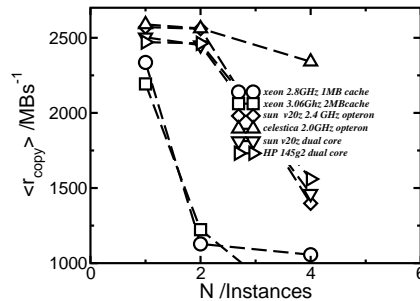


Fig. 2. Average memory transfer rate, $\langle r \rangle$, as a function of the number of concurrent instances, N , as measured by the COPY routine of the STREAM program for an array of length 8000000 double precision words. See text for details.

erboard and therefore enhance the SMP capabilities within the same server. AMD is now offering the new dual core CPUs: two real CPUs sharing the same die and hence the same hyper-transport link [6]. This approach, even if convenient is far more expensive than the INTEL hyper-threading technology present on all recent Xeon CPUs. This latter technology enables a single physical processor to appear like two logical processors to the operating system and multi-threaded applications.

To accomplish this we have employed the STREAM program [7] to measure the performance speed of the memory on several platforms as a function of the number of concurrent instances. This program performs the BLAS level 1 functions copy ($A=B$), scale ($A=\alpha B$), add ($A=B+C$), and triad ($A=B+\alpha C$) where A , B and C are vectors and α is a scalar. Hence STREAM performs operations where memory accessing events significantly dominate over floating point operations and thus essentially measures the bandwidth between memory and CPUs. We employ a double precision array of length 8000000 ($\approx 183\text{MB}$) [8] and 1–4 concurrent instances; we report here only the results of COPY and note that similar trends are obtained with all four BLAS functions. The results of these tests as measured by the average memory transfer rate, $\langle r \rangle$ over all N instances, are shown graphically in Fig. 2.

For sample architectures we chose the Xeon 2.8GHz 1MB cache with hyper-threading enabled (front-side bus at 533MHz and memory at 400MHz), a Xeon 3.06GHz 2MB cache hyper-threading disabled (front-side bus at 800MHz and standard memory at 400MHz), as typical Xeon based architecture compute nodes, and the SUN V20Z and CELESTICA A2210 based dual OPTERON machines both with 400MHz Kingston memory. The by far worst performance was seen for the Xeon machines, where the memory contention for 2 concurrent processes induces a drop of about 50% in the memory transfer rate. Neither changing CPU speed, front side bus frequency, cache size nor dis-/enabling hyper-threading has a significant effect on this type of behavior. The STREAM results are essentially the same for single and dual core OPTERON SUN V20Z both of which exhibit essentially identical performance with the dual core HP DL145G2 server. Here

the memory transfer rate does not degrade on going from 1 to 2 instances due to the 2 independent memory interfaces of the two processors. However, on increasing to 4 instances there is an approximate 40-50% drop in performance, i.e. similar to that observed on the Xeon with 2 instances. Most surprising is the performance of the CELESTICA based machine which shows 0-40 % memory transfer rate degradation for up to 4 simultaneous instances, depending on size of the array. This demonstrates that mainboard architecture is an important

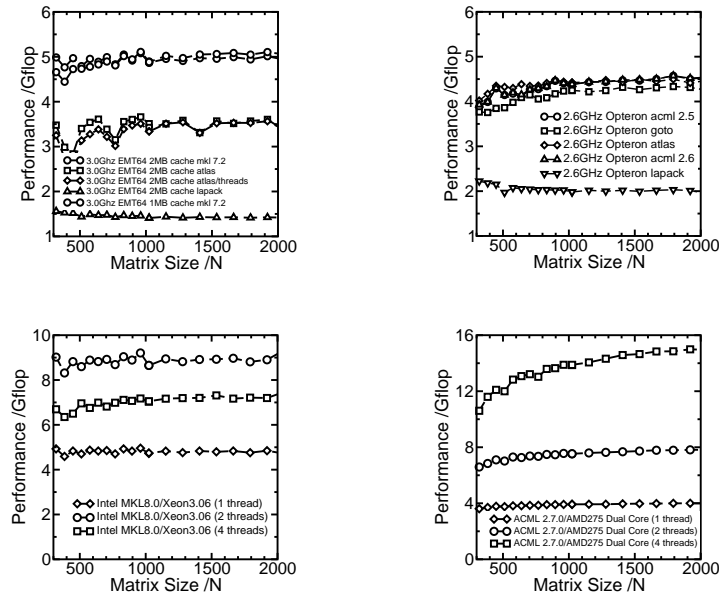


Fig. 3. Performance in Gflops as a function of Matrix size N for the standard level 3 BLAS DGEMM kernel for selected libraries on (a) EM64T (b) OPTERON, (c) multi-threaded MKL/EM64T, (d) multi-threaded ACML/OPTERON See text for discussion

(and often under-appreciated) variable in machine performance profiles which should not be neglected when evaluating hardware solutions.

The second issue to evaluate is the performance of the CPU and the available mathematical libraries for the respective machines. In this context we consider the ubiquitously used double precision generic matrix-matrix multiplication (DGEMM) BLAS level 3 kernel as implemented in various performance libraries on the EM64T and OPTERON platforms, see Fig. 3. For a 3.06 GHz EM64T (with either 1 or 2MB of cache) the best performance for $N \times N$ matrix multiplication is found for the MKL 7.2 library which shows a performance of 4.9-5.0 Gflops for $N > 300$ with the ATLAS library showing only 3.5 Gflops and 1.5 Gflops for the standard fortran77 LAPACK reference implementation for the same matrix sizes.

In comparison a 2.6GHz OPTERON yields values of 4.0-4.5 Gflops for ACML 2.5, ACML 2.6 and ATLAS. and 2.0 for standard LAPACK. For the same libraries ie LAPACK and ATLAS the opteron shows superior throughput however, the very well tuned MKL 7.2 allows for far better throughput on the EM64T.

Table 1. Mflops as function of threads (size=4096)

Machine Description	1 instance	2 instances	4 instances
Sun v20Z SC 2.6 GHz	4660	4687	2345
Sun v20Z DC 2.2 GHz	4003	4006	3960
HPdl145 DC 2.4 GHz	4397	4399	4394
Xeon EM64T 3.06GHz	4890	4730	1950

It is now interesting to see if there is any overhead in concurrent execution of the DGEMM kernel. We therefore run at the same time 1, 2, and 4 DGEMM instances on large matrices ($N=4096$, 8192 square matrices) in order to almost saturate the available memory on machine, see Table I. In the case of 2GB memory, as on Xeon machines, four instances of $N = 4096$ matrix occupy around 80% of the available physical memory. It is quite disappointing that, even if the Intel Xeon platform is delivering the best performance for single and dual CPU usage, on 4 instances the performance drops to less than 50%. Similar trends are obtained when considering the performance of multi-threaded libraries, see Fig. 3(c)-(d). On a dual 3.0GHz Xeon machine with hyper-threading MKL 8.0 shows perfect performance increase of 2 when run with 2 threads but actually *decreases* in performance upon increasing from 2 to 4. On the dual core 2.2GHz OPTERON the multi-threaded ACML library shows lower performances for 1 and 2 instances than the Xeon/MKL equivalent but shows perfect scalability up to 4 instances for larger matrix sizes. Hence, although the XEON/MKL combination gives better performance for one or 2 instances it does not appear to benefit from the hyper-threading and thus for multi-threaded applications the dual core OPTERON shows superior scalability.

Overall, for single CPU applications using codes that extensively draw on the math libraries EM64T/INTEL MKL is the platform of choice. However, for the less optimized codes the OPTERON architecture is superior and given the above observation of degradation of memory bandwidth for multiple concurrent processes this performance would be greatly reduced on dual or hyper-threaded Xeon machines for large memory applications. This last point is a serious de-traction of the EM64T for cluster computing involving massively parallel codes due to the fact that in order to minimize the cost of the high speed network needed, it is much better to use dual CPU platforms instead of single CPU ones. Hence, the best scenario is for small parallelization of single EM64T using Gigabit Ethernet as a cost effective solution if the applications at hand do not require massive parallelization. Whereas a dual OPTERON platform would be the

preferred choice for clustering in the case of less optimized codes and the need of a high speed network for parallelization.

4 Case Studies for Atomistic Simulation Codes

4.1 *Ab initio* code: CPMD

In this section we discuss benchmarks for the CPMD program [3]. To begin we consider a simple test calculation of an electronic wavefunction optimization for a 63 atom Si cell with a 50 Ry plane wave basis set [9]. This calculation

Table 2. CPU time for a wavefunction optimization of 63 Si atoms with plane wave cutoff of 50 Ry (requiring 1GB of memory) on selected platforms.

Machine Description	time (s)
AMD Athlon XP2500+, 1.83GHz, PC333	5196
AMD Athlon XP2500+, 1.83GHz, dual-channel PC333	3848
Intel Pentium 4 Xeon, 2.4GHz, dual-channel PC200	3401
Intel Itanium2, 900MHz, HP zx6000	3145
AMD Opteron, 1.6GHz, PC266, 64-bit	3632
AMD Athlon64 3200+, 2.0GHz, PC333, 32-bit	3143
AMD Athlon64 3200+, 2.0GHz, PC333, 64-bit	3134
IBM Power4+ 1.7 GHz, Regatta H+	2259

requires 1GB of memory and performs a large number of both level 1 and 3 BLAS operations as well as extensive use of FFT routines. This job extensively employs the standard optimized libraries maximizing cache/CPU usage with minimal *I/O* but with heavy memory access. The time required per optimization step on selected single CPU architectures is reported in Table II.

This group of benchmarks underscores the fact that that CPMD performance is mainly influenced by two factors: floating-point performance and memory bandwidth. Lack of the latter can severely hurt the performance of an otherwise capable CPU—see the two results of Athlon XP2500+ CPUs, where the same machine with a dual-channel memory interface is about a third faster. The AMD Opteron machine stands out due to the fact, that it runs 32-bit code about as fast as 64-bit “native” code. For single-CPU performance the IBM Power4 CPU is the fastest and the 900 MHz Itanium2 and the 2GHz Athlon64 are comparable, the former being impressive given its comparatively low clock speed.

We next consider the shared memory, symmetric multiprocessor (SMP) performance by running concurrently 1 and 2 instances of this same job, see Table III. The dual Pentium 4 machine (which does not have a fast front-side bus), gives a truly abysmal SMP performance. This observation combined with the fact that the dual Athlon results are only marginally better, prove that the low memory bandwidth in that kind of PC hardware badly hurts the SMP performance. The Itanium2 machine gives the typical performance of most workstation

Table 3. Average CPU time/step for concurrent instances of a wavefunction optimization of 63 Si atoms with plane wave cutoff of 50 Ry (requiring 1GB of memory) on selected platforms. SMP speed is calculated by the ratio of the time for a single process divided by average of 2 processes.

Machine Description	1 instance [s]	2 instances [s]	SMP speed
Dual Pentium 4 Xeon, 2.4GHz, PC200 ¹	1275	2090	61%
Dual Athlon MP1800+, 1.53GHz, PC266-ECC	2167	3134	69%
Dual Compaq Alpha EV68AL, 833MHz, DS20	1519	1719	88%
Dual Intel Itanium2, 900MHz, HP zx6000	3120	3507	89%
AMD Opteron, 1.6GHz, PC266, 64-bit	3582	3617	99%

(¹ due to limited main memory, this run had to be done with a reduced 30Ry plane wave cutoff.)

class SMP machines in that the scaling is better than standard PCs but there is still a 10-15 % overhead due to sharing the memory bus. The optimal timings on the OPTERON machines document the benefits of integrating a memory controller into the CPU. Similar observations have been obtained with other test examples on several different OPTERON and ITANIUM2 machines including 4-way machines with the distinct note that SMP performance can be improved using faster memory on the same machine.

We now discuss the scalability of the CPMD code as illustrated by the same benchmark but employing a larger plane wave basis set of 70 Ry. This increases the memory requirement to 1.8GB as well as leads to an increase in parallel communication. It is noted that CPMD has the most favorable scaling for jobs employing a large plane wave basis set. To examine the scalability we compare single CPU Athlon machines with 100 and 1000 Mbit Ethernet and Dolphin-SCI in Fig. 4 in order to factor out effects from SMP overhead and focus primarily on the network. In this graph we calculate the time/optimization step multiplied by the number of CPUs; in the ideal case of perfect scaling this function should be a flat line. Not surprising the high latency of Ethernet leads to terrible scaling due to the heavy MPI communications of the FFT library. It is noted however, that for a small number (2-4) of nodes, Gigabit Ethernet performs acceptably and would thus be the most cost effective solution for clusters where small parallel jobs are to be run. Optimizing the Ethernet interconnect by increasing the backplane communication capacity, larger ethernet frames, or using a different MPI implementation, has only marginal effects, since it does not change the latencies. The Dolphin-SCI card which has approximately 3 Gbit bandwidth and latency of only a few microseconds shows excellent scalability up to 30 nodes employing a single CPU/node and maintains this performance when 2 CPU/node are employed, see Fig. 4 where the different timing between the 2 curves is a result solely of the SMP overhead of using 2 CPU on the same machine. That kind of scaling has been achieved with both, the vendor supplied MPI package, as well as with using a standard Open Source implementation in combination with a library that diverts the normally TCP/IP encoded communication

channels to the SCI distributed shared memory communication [10]. This same behavior is also seen with a 2GHz OPTERON/INFINIBAND cluster employing the INFINIPATH driver [11] where the timings are significantly faster than the Athlon and the scaling remains good up to about 50 nodes.

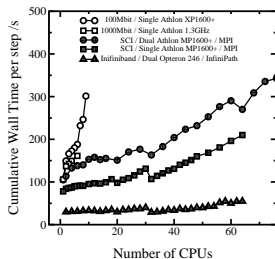


Fig. 4. (a) Cumulative time per MD step for a CPMD system of 250 Electrons with 70Ry cutoff requiring 1.8GB of memory on selected PC cluster platforms.

We next turn to a slightly different scenario for running CPMD where we again consider a job which requires on the order of 2GB of memory; a Car-Parrinello simulation of Au(111) surface with 8 SCH₃ molecules with a low plane wave cutoff. In this case we use fewer plane waves than the previous example but have approximately 4 times the number of electrons. This job thus stresses different parts of the code in the serial version the majority of the time is spent calculating forces from the non-local parts of the pseudo-potential (which draws extensively from level 3 BLAS) as well as in the parallel implementation where an increasing proportion of the time goes toward the all-to-all communication of the FFT interface routines as the number of CPUs increases. We performed benchmarks of scalability for the above test job on the dual 3GHz Xeon/Myrinet cluster at Cineca supercomputing center [12] and the dual 2.0GHz dual OPTERON/Lanai 9 Myrinet [13] cluster at SISSA, see Fig. 5. The SMP penalty on single core OPTERON machines is essentially negligible for machines with 333 and 400MHz memory and only 15 % with 266MHz memory. Comparing this with the dual 3GHz Dual PIV 533 MHz front side bus [12] the SMP penalty for CPMD compiled with MKL is only about 5-10 % for jobs of this type. The test job run on this latter platform is about 10-20 % slower than the 2.0GHz dual OPTERON machine ie largely due to the SMP overhead on the Xeon. Scalability on the OPTERON is impressive in that there is no difference in timing between running with 1 or 2 CPUs per node, see Fig. 5 compare with the Athlon results in Fig. 4. However, the older Myrinet network on the OPTERON cluster degrades at about 7-10 nodes for this job whereas the newer network on the Cineca machine scales well up to 15 nodes for the same job thus timings are concurrent on the 2 machines between 15-20 CPUs. This example highlights a few important points: (i) for code extensively using math libraries Pentium based machines with currently available fast front

side bus frequencies can give performance comparable to that of the OPTERON. (ii) for applications, where massive inter-node communication occurs, network performance is as critical a consideration as the compute node itself.

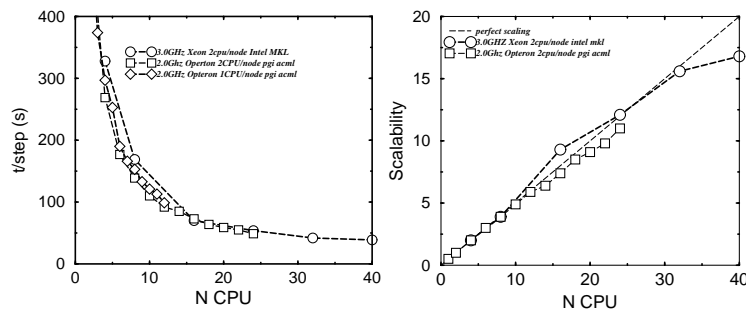


Fig. 5. (a) Time per MD step for a large CPMD system of 1000 Electrons requiring 1.8GB of memory on 3GHz dual Xeon/Myrinet cluster and 2GHz OPTERON/Myrinet. (b) MPI Scalability for the same simulations. See text for discussion

We now consider performance of CPMD on the recently available dual core OPTERON CPU: with example test results on a 2.2GHz sun v20z. We first consider a conjugate gradient wavefunction optimization (using both level 1 and 3 BLAS routines) as a function of plane wave basis set size (ie memory), See Fig. 6. All the calculations scale perfectly for 1 and 2 CPUs in accordance with our above analysis on SMP penalty on single core machines. For 3 CPU there is a dramatic drop in scaling in part due to asymmetrical balancing of the load between CPUs but at 4 CPUs the scaling is still lower than 4 and is proportional to the memory requirement of the job. This can be understood in terms of the STREAM benchmarks in Fig. 2 which show that on the dual core OPTERON for level 1 BLAS routines there is memory access contention for multiple concurrent instances. We note, in general, the scalability is very much dependent on the application and its use of the optimized libraries, however, most plane wave based codes we have tested seem to obtain scaling between 3-3.7/4 CPU (after suitable optimization of the compilation of the executable) even for applications requiring 4GB of RAM. However, since the cost increase is only on the order of 25 % relative to a single core but the performance increase is on the order of 50-90 % (even for these big memory applications) then this technology definitely improves the price/performance ratio.

4.2 Classical MD: DLPROTEIN

The DLPROTEIN_2.1 program has a standard set of four benchmarks for which we have collected many performance numbers over the last few years [14]. These benchmarks stress different computational kernels within the code yielding quite

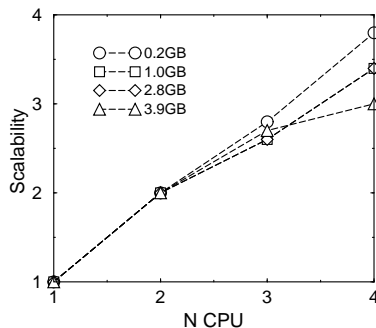


Fig. 6. MPI scalability of CPMD compiled with ACML 2.7.0/IFORT 9.0 wavefunction optimization requiring 0.2GB, 1GB, 2.8GB and 3.9GB peak memory usage on a 2.2 GHz dual core sun v20z with 400MHz memory. See text for discussion

different results from the various parts of the code. In the course of this section we point out, when needed, the differences among them.

Table 4. Single processor performance of the DHFR cases for the DLPROTEIN_2.1 code: times are in seconds

Section	Link Cell	VdW	SPME	total time
Xeon 3.06 32-bit	85	201	77	437
Xeon 3.06 64-bit	80	180	69	392
AMD 2.0SC(celestica)	71	172	75	374
AMD 2.2DC(sun v20z)	59	153	78	342
AMD 2.4DC(HP)	54	140	71	310
AMD 2.6SC(sun v20z)	54	133	61	292

Table 4 collects scalar performances obtained on different 64 bit architecture using DHFR protein solvated with TIP3P water, in a periodic box. There are 23,558 total atoms, and SPME is used with a direct space cutoff of 9 Ang: this a standard benchmark for biological simulations. Other MD codes reports benchmark data for this system as well [15]. We report for this test the global time and time spent in the two most consuming tasks of the code: short range interactions and long range electrostatics by Smoothed Particle Mesh Ewald SPME computations. The short range computation is split here in the two sub-tasks: building the neighbor list (Link Cell) and the actual computation of short ranges Van der Waals forces using the list (VdW). The two sub-tasks have different computational characteristics: the first is based on integer operations and data are accessed in a direct way, whilst for the latter floating point operations are dominant and data are accessed through the list, i.e. in a “cache unfriendly” way. In the SPME part the 3d-grid employed is generally accessed in a direct way, al-

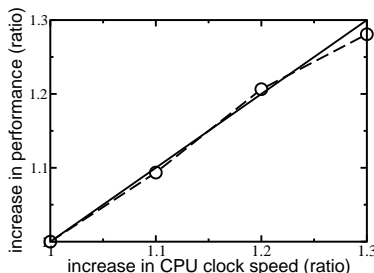


Fig. 7. Increase of relative performance of DLPROTEIN_2.1 vs CPU frequency on AMD OPTERON for the DHFR example

lowing better exploitation of caching heuristics. Measurements are all performed employing the INTEL V9.0 compiler.

We note that this example stresses a couple of important observations which we have noted with many different pieces of software.

- on Intel Xeon the 32-bit version is less efficient than the 64-bit version and this is more evident in the VdW section: which is the least optimized part of the code and therefore a 64-bit approach can be more efficient.
- AMD OPTERON CPUs are better than Intel Xeon even if the latter have an up to 30% faster clock rate: since DLPROTEIN_2.1 is essentially not well optimized and very “cache unfriendly”, and thus requires an extremely high memory bandwidth for optimal performance.

This observation seems to be generally true for most poorly optimized software and is one of the bigger selling points of the AMD 64-bit architecture. Moreover from Fig. 7 it is clear that the increase in performance is almost linear with the increasing the CPU frequency, meaning that no big role is played by memory architecture issues for this kind of codes.

It is now interesting to see SMP performance running concurrent copies of the same DLPROTEIN examples. We focus our analysis here on Xeon machines with and without hyper-threading enabled versus AMD single/ dual core machines. Table 5 allows us to compare efficiency of hyper-threading vs dual core technology. As a consequence of the small memory requirement, which allows us to avoid SMP memory access contention, for this job the performance on OPTERON machines is almost perfect up to all 2 and 4 CPU for single and dual core CPUs respectively. The results on the EM64T machine are likewise equally good up to 2 CPU for the same reason. However, the hyper-threading technology provides only a modest 8 % improvement in performance when the number of simultaneous instances is increased to 4, i.e. it is virtually ineffective. So even for these non-memory intensive applications hyper-threading may not be a reasonable alternative to the dual core CPUs of available for OPTERON.

We now discuss parallel performances; as noted in Sect. 2, the RD algorithm adopted by this code does not show good scaling for large numbers of CPU.

Table 5. SMP performance for 1/2/4 instances of DLPROTEIN_2.1 for the DHFR example : SMP efficiency, in parentheses, is defined as serial time divided by the average of 2 and 4 instances respectively

Platform	Serial	2 instances	4 instances
Xeon 3.06 64-bit	397	401 (99%)	792 (50%)
Xeon 3.06 64-bit HT	378	383 (98%)	645 (58%)
AMD 2.2DC(sun v20z)	332	334 (99%)	337 (98%)
AMD 2.6SC(sun v20z)	293	306 (96%)	607 (48%)

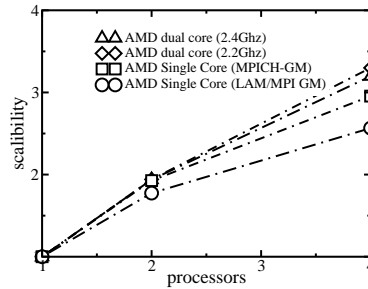


Fig. 8. MPI scalability on Dual core vs Single Core AMD CPUs

Moreover the systems tested here possess complex topologies and constraints that require a high communication overhead. For these reasons it would be most convenient to execute these types of simulations on medium sized (4–8 CPU) SMP machines. Note that memory access contention, is not a significant factor in this type of simulation, however MPI communication is and thus remaining within the same compute node is the most efficient way to execute this type of simulation. Figure 8 shows the overall scalability for a simulation of liquid water in the range 1–4 processors. Parallel behavior is rather poor for 4 CPU on two different dual single core AMD CPUs, even though they are connected by Myrinet cards [13]. We note in this case MPICH-gm shows slightly better scaling than LAM (rpi=gm) which is due to the larger bandwidth made available by MPICH in the all-to-all communications. On the other hand, a single server equipped with two AMD dual core CPUs delivers, for this type of job, much better performance and increases the overall speed-up to an acceptable value.

Overall, for clustering purposes this type of code runs well on either EM64T or OPTERON. However, due to the limited scalability of these types of simulations they are best performed with as many CPUs within the same machine. In this respect the dual core technology of the AMD OPTERON shows a clear advantage over the hyper-threading (on Xeon) which may not be that effective for many applications.

5 Discussion and Conclusions

In general we observe that XEON PIV/EM64T machines are very good for codes that extensively use math libraries, especially MKL. The critical architecture feature on these machines for *ab initio* codes (and all big memory applications) is the front side bus speed which is currently at 800MHz for good compute nodes however, the memory contention between concurrent instances will always induce an overhead due to the shared memory bus and thus only well optimized code will truly be able to exploit this architecture. The OPTERON shows a superior performance for less optimized codes due to its integrated memory controller, and is excellent for cluster applications as it shows a negligible SMP penalty even for poorly optimized software. The dual core version of the OPTERON also shows reasonable scalability for both large memory applications and less optimized codes and definitely improves the price performance ratio.

In closing we stress the importance of knowing the computational requirements of the target applications of the users. To maximize price/performance it is useful to “spot out” where the codes perform the worst and for what reasons. A second beneficial tool is to compile a library of well-thought out test examples that stress the various parts of the codes and run them on a wide variety of test machines. Although standard benchmarks with libraries do provide fundamental insight into what a machine *can* deliver, the more common scenario is that the applications are far from ideal in their optimization and performance profiles. In short, the best price / performance can be obtained by knowing why a code runs slow on a given architecture so one can more accurately choose the machine which best overcomes these weaknesses.

6 Acknowledgments

We would like to thank P. Calucci, M. Baricevic, C. Onime R. Gebauer and A. Nobile for insightful discussion and D. Marx for technical support. We are extremely grateful to M. Davini, SUN MICROSYSTEMS ITALIA, AMD ITALIA, EXADRON, OXIRYA, MYCROFT/HP ITALIA for generously providing us with many test machines. This work was partially supported grants from MIUR and CNR-INFN.

References

1. To see more about the Democritos center and our software visit <http://www.democritos.it>
2. See <http://www.cmm.upenn.edu> for more details.
3. CPMD, J. Hutter *et al.*, IBM Corp and MPI für Festkörperforschung 1997–2001; Based upon: R. Car and M. Parrinello, Phys. Rev. Lett. **55**, 2471 (1985); For a description of the code see: D. Marx and J. Hutter in *Modern Methods and Algorithms of Quantum Chemistry*, pp. 301–449, Editor: J. Grotendorst (NIC, FZ Jülich 2000).

4. S. Melchionna and S. Cozzini, DLPROTEIN_2.1 User Guide (<http://www.sissa.it/cm/DLPROTEIN>).
This code is based on DLPOLY, see: W. Smith & T. R. Forester, "The DIPoly 2.0 User Manual", CCLRC, Daresbury Laboratory, Daresbury, Warrington WA4 4AD, England (1995).
5. S. Goedecker, *Comp. Phys. Commun.* **76**, 294 (1993).
6. At the time of preparing this document, dual-core CPUs were also introduced by Intel, but not available at the time of running the benchmarks on them.
7. The STREAM benchmark program measures the sustainable memory bandwidth and computation rate for simple vector kernels. See: <http://www.cs.virginia.edu/stream/ref.html>
8. We repeated almost all the measurements with larger data sets (411Mb and 869Mb) with no significant changes in the numbers and trends obtained for the smallest data set.
9. more CPMD benchmarks are discussed at <http://www.cmm.upenn.edu/~akohlmeier/cpmd-bench.html>
10. SCI Socket Library, <http://www.dolphinics.no/support/downloads.html>
11. Numbers for the Si example on OPTERON/INFINIBAND provided courtesy of Greg Lindahl of Pathscale, Inc.
12. These calculations were run on the older nodes of the cl-x cluster at the Cineca supercomputing center. See <http://www.cineca.it> for details of the machine.
13. Although the Myricom Lanai9.0 card is not supported for the x86_64 Linux platform it was found that a stable network could be establish for MPI communications employing the GM 2.0.x driver running under a SUSE linux 9.0 (or greater). For LAM MPI it was found that a robust package could be created by disabling the memory de-pinning using the `--disable-rpi-gm-ptmalloc` option in LAM 7.0.x though use of the parallel feature in LAM 7.1.1 did not lead to a stable library. Employing MPICHGM-1.2.6..14b, which does not have this feature, also leads to a stable package with an increase in the overall bandwidth, relative to LAM, hence leading to improved MPI all-to-all communication.
14. See <http://hpc.sissa.it/paper02/paper02.ps>
15. See <http://amber.scripps.edu/amber8.bench1.html>