Quantum-ESPRESSO Performance on Cray Systems

Roberto Ansaloni, *Cray Italy,* **Carlo Cavazzoni** *and* **Giovanni Erbacci**, *(CINECA)*

ABSTRACT: *Quantum-ESPRESSO (op**E**n-**S**ource **P**ackage for **R**esearch in **E**lectronic **S**tructure, **S**imulation, and **O**ptimization) is a Total Energy and Molecular Dynamics simulation package based on Density-Functional Theory, using a Plane-Wave basis set and Pseudopotentials. The Quantum-ESPRESSO package is composed by three main simulation engines: PWscf for Total Energy and Ground State properties, FPMD for Car-Parrinello molecular dynamics with norm-conserving pesudopotentials, and CP for Car-Parrinello molecular dynamics with ultra-soft pseudopotentials.Quantum-ESPRESSO suite has been ported and optimized on Cray vector and MPP systems. This talk will discuss some porting and optimization issues and will report on the performance achieved.*

KEYWORDS: *DFT, Quantum-ESPRESSO, modeling, molecular dynamics, FFT*

Introduction

The Quantum-ESPRESSO package is an open-source collection of state-of-the-art codes for density-functional theory (DFT[1]) calculations of electronic structure, based on plane waves and pseudopotentials. It is used by a large community of material science researchers for computer simulations. Quantum-ESPRESSO includes the following codes:

- **PWscf**, a code for electronic structure, structural optimization, molecular dynamics, vibrational and dielectric properties[2];
- **FPMD**, a Car-Parrinello variable-cell molecular dynamics code for norm-conserving pseudopotentials[3];
- **CP**, a Car-Parrinello variable-cell molecular dynamics code for ultrasoft pseudopotentials[4]. (The two Car-Parrinello codes are based on the original code written by R. Car and M. Parrinello[5].)

Quantum-ESPRESSO also includes these auxiliary codes:

- **Pwgui**, a graphical interface for producing input data files for PWscf[6];
- **atomic**, a program for atomic calculations and generation of pseudopotentials.

Quantum-ESPRESSO codes share a common installation method, input format, data output format, and pseudopotentials format, as well as parts of the basic code. The codes have been the subject of papers in a number of scientific journals.

These codes are also well-suited for parallel supercomputers. This is because DFT allows you to perform simulations with quantum mechanical accuracy, since the full electronic problem is solved for the simulation box. So, the computational cost of DFT is in the determination of inter-atomic potential.

[1] P.C. Hohenberg, W. Kohn, Phys. Rev. 136 (1964) B864.
[2] Developed by S. Baroni, S. de Gironcoli, A. Dal Corso (SISSA, Trieste), P. Giannozzi (Scuola Normale, Pisa) and others.
[3] Developed by C. Cavazzoni (CINECA, Bologna), S. Scandolo (ICTP, Trieste), G. Chiarotti (SISSA, Trieste), P. Focher, G. Ballabio and others.
[4] Developed by A. Pasquarello (IRRMA, Lausanne), K. Laasonen (Oulu), A. Trave (UCBerkeley), R. Car (Princeton), P. Giannozzi, N. Marzari (MIT) and others
[5] R. Car, M. Parrinello, Phys. Rev. Lett. 55 (1985) 2471.
[6] written by Anton Kokalj (IJS, Ljubljana).

Car-Parrinello Algorithm and Code Parallelization

Car-Parrinello molecular dynamics simulations are among the most diffused material science applications run on supercomputers, so it is important to evaluate their performance on Cray systems. To that end, we have performed a set of FPMD benchmarks on Cray systems.

In the Car–Parrinello method, forces that act on atoms are calculated from the full quantum mechanical solution of the electronic problem, based on DFT. Here, a hypothetical dynamic system that represents the physical system is introduced. Its potential energy surface E, derived from the Car-Parrinello lagrangian, is an appropriate function of both ionic and electronic degrees of freedom, with electronic wave functions treated as classical fields. This system is devised so that the ionic trajectories generated by its dynamics closely reproduce those of the physical system in the Born–Oppenheimer (BO) approximation. The CP algorithm scheme is illustrated in Figure 1.

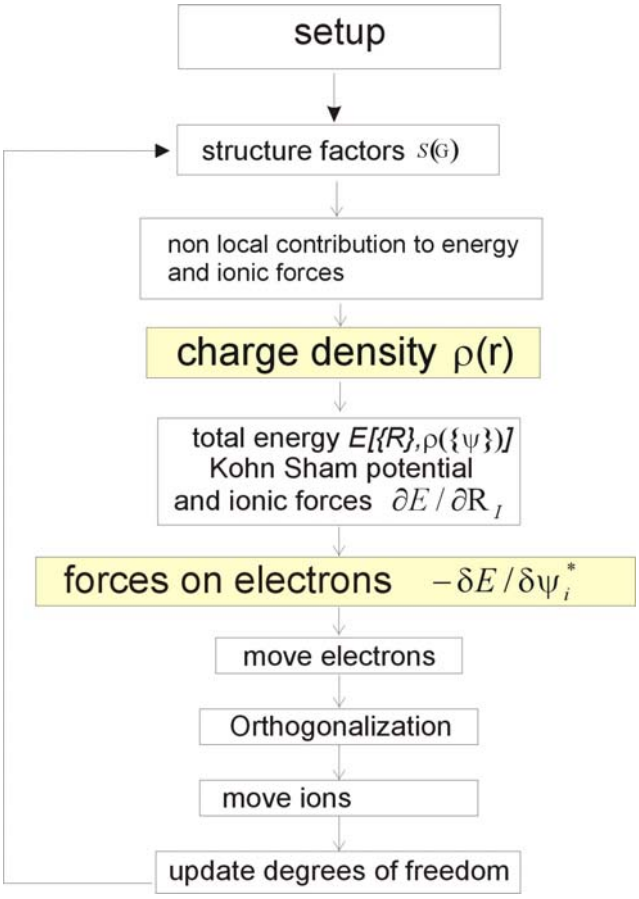

Figure 1. Scheme of the CP loop – a turn is completed each time step. The light yellow operations (computation of charge density, and computation of electronic forces) are the heaviest operations. They involve three series of FFT transformations (two forward, one backward), one FFT for each couple of electrons (one electronic state in LDA approximation).

Ab-initio total energy (***E***), interaction potential (***V***), and ionic and electronic forces ($\partial E/\partial R_I$, $-\delta E/\delta\psi_i^*$) are functions of the ionic positions (R_I) and of the charge density ($\rho(r)$) of the system, which is defined as:

$$\rho(r) = \sum_i |\psi_i(r)|^2$$

where ψ_i are single particle electronic wave functions. Periodic boundary conditions allow the expansion of the electronic wave functions in plane waves. As a consequence, the wave function for the electronic state is expressed as:

$$\psi_i(r) = \sum_G C_i(G)\exp(iGr)$$

where ***G*** represents vectors in reciprocal space. The basis set for this expansion is reduced to a finite set by truncating the sum over ***G*** to include only those plane waves with a kinetic energy $K=1/2|G|^2$, less than a given energy cutoff *E*c. It is clear that the choice of *E*c determines the accuracy of the calculation of the DFT energy. With the atomic positions (R), the fourier coefficients (C) are propagated in the main dynamic loop as classical degrees of freedom. They are then transformed to real space in order to compute the charge density (see Figure 2).

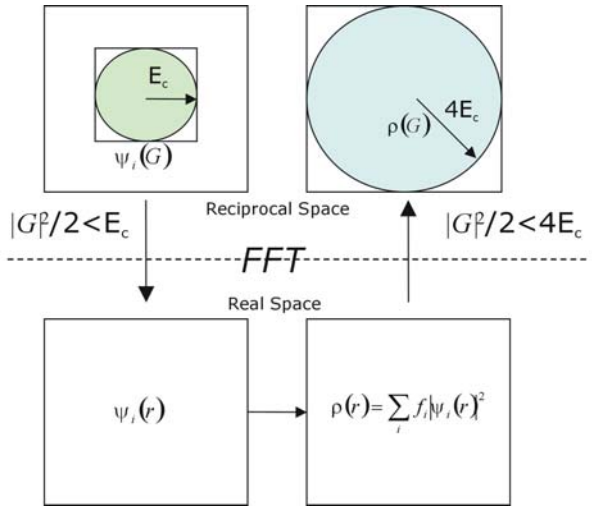

Figure 2. Computation of the charge density. Starting from the wave-functions in reciprocal space (top-left panel) a series of FFT is performed to transform them into real space (bottom-left panel). In real space, the single state wave functions are summed up to give the charge density in real space (bottom-right panel).

Finally, the real space charge density is transformed back to reciprocal space, using a single FFT (top-right panel).

To compute the energy, potentials, and forces, we need to represent physical quantities (charge density and wave functions) in both spaces. This is because some terms of the energy are diagonal (local) in reciprocal space, and other terms are diagonal in real space. This determines the presence of two types of arrays in the code:

- 3D arrays that store quantities represented in real space, and;
- 1D arrays that store quantities represented in reciprocal space (where ***G*** vectors are ordered according to their module).

A fast Fourier algorithm (FFT) is used to transform quantities from real space to reciprocal space. In real space, a unique 3D mesh is used to represent both charge density and potentials; and in reciprocal space, a smaller mesh is used to represent wave functions (**G** vectors up to $|\mathbf{G}|^2/2<Ec$) and a larger mesh represents potentials (**G** vectors up to $|\mathbf{G}|^2/2<4Ec$). Potentials span a larger space since they are functions of the charge density, which in reciprocal space is a convolution of wave functions.

The presence of quantities that span both reciprocal meshes requires a careful data distribution to balance the workload.

In Figure 3, we show an example with 4 PEs (*Np = 4*) to illustrate how reciprocal vectors are distributed in our code. *Ng* and *Ngw* are the number of vectors whose kinetic energies are smaller than *4Ec* and *Ec*, respectively. Vectors up to *Ec* are divided among PEs according to the FFT scheme (see Figure 5), under the criterion that each processor should have *Ngw/Np* G vectors; the remaining (*Ng - Ngw*) vectors up to *4Ec* are then distributed with the same algorithm, but this time the number of *G* vectors on each processor is as close as possible to *(Ng – Ngw)/Np*.

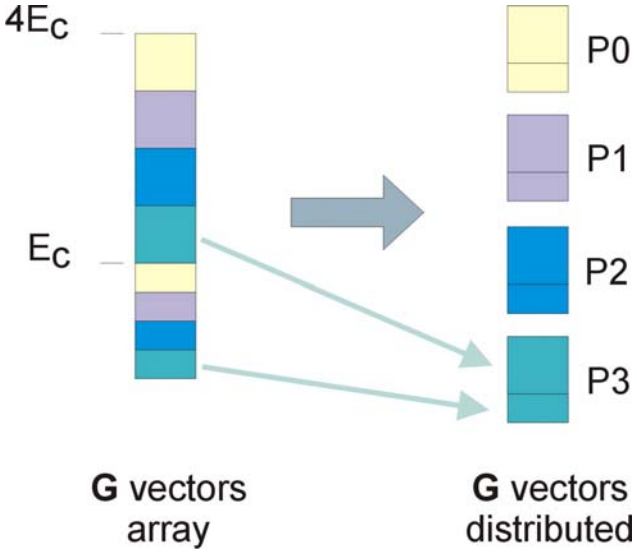

Figure 3. Vectors of the reciprocal space stored in a single processor memory (left) and mapped to processors (right).

Real space 3D arrays are subdivided into blocks across the PEs (see Fig. 4). In particular we have distributed the z-dimension of the 3D real space mesh, while the y- and x-dimensions are not distributed (see Fig. 4). This guarantees good load balance for a wide range of numbers of processors and mesh sizes, yet it still allows the 3D-FFT algorithm (see below) to be based on 1D/2D FFT scalar routines.

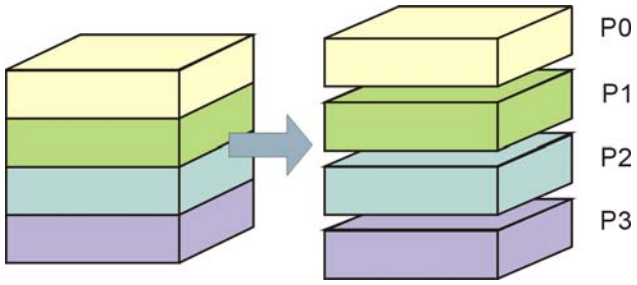

Figure 4. Vectors of the real space stored in a single processor memory (left) and mapped to processors (right).

Given this data distribution, in practice only two algorithms need to be parallelized, 3D FFT, and wave function orthogonalizations. All communication and synchronization are then confined to these two points in the CP algorithm.

The FFT Routine

The FFT routine is called repeatedly to transform wave functions, charge density, and potentials back and forth between reciprocal and real space. (Typical applications spend nearly two-thirds of processing time on this task.) The size of the real space mesh is fixed by the charge density, and corresponds, in reciprocal space, to the mesh that contains the sphere of G vectors with a kinetic energy smaller than 4Ec. Wave functions, whose Fourier components have a kinetic energy smaller than Ec, must be transformed on the same real mesh as the charge density. Since standard FFT algorithms operate between meshes of the same size, the wave functions in our case should be copied into a bigger mesh, and then transformed. It is clear that in this way a large amount of time is wasted in transforming elements with zero value. To optimize this operation, we implement an ad hoc FFT algorithm for wave functions, shown here:

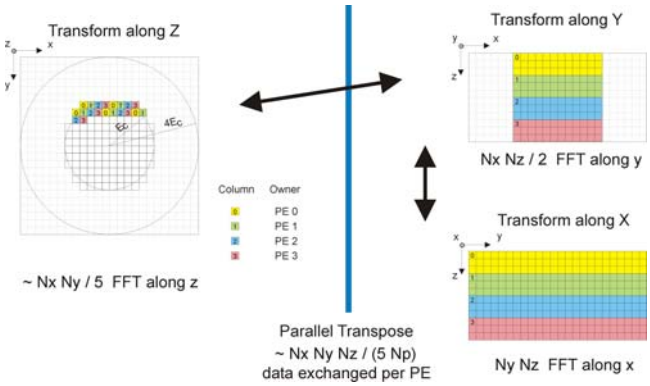

Figure 5. 3D FFT ad-hoc algorithm (see text).

This algorithm is for four PEs (Np = 4), and transformation from reciprocal to real space. It takes advantage of the fact that a 3D FFT is a linear superposition of three subsequent series of 1D FFTs, along the Cartesian coordinates. For each series, only those 1D FFTs containing non-zero elements are evaluated. In the left panel is a top view of the FFT grid (Nx;Ny;Nz) together with the cut-off radius. This example illustrates what happens when the first FFT transformation is performed along the z-direction. Sticks (that is, columns along the z-direction) are allocated by the PEs only if they contain at least one G vector of the inner sphere. Allocated sticks are assigned to the PEs (see color code) in such a way that the number of FFTs per PE differs, at maximum, by one, and only in the case that the number of columns is not a multiple of the number of PEs. The total number of 1D FFTs performed in this step is approximately 1 / 5 the number of a standard 3D FFT algorithm (NxNy). The transformed columns are then transposed to distribute the z-direction across PEs.

The transposition, like the whole FFT, has been implemented with the particular distribution of non-zero elements in mind: it involves local data collection and an all-to-all communication. On the Cray X1 system, thanks to co-array, the transposition has been implemented using just one communication step, without data collection and buffering (see below).

As is the case for z-sticks, only those columns that contain at least one non-zero element are likewise transformed along the y-direction (Figure 5, middle panel). Here the number of 1D FFTs is reduced by 1/2 (NxNz/2) with respect to standard routines.

Finally, the last series of FFT transformations are performed along the x-direction.

Orthogonalization

The orthogonal constraints for the wave functions are satisfied by solving a matrix equation of size Nb^2, where Nb is the number of electronic states. Terms entering this equation are calculated through scalar products between wave functions, giving a complexity of $Ngw\ Nb^2$. The solution of the matrix equation instead has a complexity Nb^3 and involves both matrix multiplications and a matrix diagonalization. In the present implementation of the code, depending on the number of bands and processors, the subroutine solving this equation can use scalar or parallel algorithm. Usually if the relation $Nb/Np > 32$ holds, it is convenient to use the parallel algorithm.

Porting to Cray Systems

The Quantum-ESPRESSO suite has been ported to several Cray systems ranging from MPP vector systems like the Cray X1, to Opteron based systems like the Cray XD1 and the Cray XT3 systems.

Some optimizations have been carried out on the Cray X1 version to exploit specific vectorized FFT routines and co-array based communication routines (CAF) in the matrix transposition section of the 3D FFT algorithm.

The 3D FFT algorithm is implemented using local 1D and 2D multiple FFT routines, and by isolating the communication part into a specific FFT_transpose routine.

Local FFTW routines have been replaced by the optimized and vectorized LibSci FFT routines. The main FFT algorithm is implemented by using 2D (X and Y directions cft2_xy) and 1D (Z direction cft_1z) multiple FFT routines.

LibSci routines CCFFTM (stride=1) and MCFFT (variable stride) are used.

Furthermore, 2D array sections have been packed to increase the number of multiple FFT performed, thus increasing the vector length and improving the performance.

Original FFT_transpose MPI implementation uses a send and a receive buffer to hold array data:

```
    DO ipz = 1, npz
      itag = mype + 1 + npz * ( ipz - 1 )
      call mpi_irecv (
       rcvbuf(1,ipz), nbuf, MPI_DOUBLE_COMPLEX,
    END DO

    DO ipz = 1, npz
      k_start = ( ipz - 1 )  * nz_l + 1
      k_end   = k_start  + nz_l - 1
      offset  = - k_start + 1
      DO is = 1, ns_l
        DO k = k_start , k_end
          sndbuf(k + offset, ipz) =
            zstick( k + (is-1)*ldz )
        END DO
        offset = offset + nz_l
      END DO
      itag = ipz + npz * mype
      CALL mpi_isend(
       sndbuf(1,ipz), nbuf, MPI_DOUBLE_COMPLEX,
    END DO

111 CONTINUE
    DO IPZ = 1, NPZ
      call mpi_test(
        irhand(ipz),rtest(ipz),istatus(1,ipz),
      IF(rtest(ipz).AND..NOT.rdone(ipz)) THEN
        offset = 0
        is_offset = dfft%iss( ipz )
        DO is = 1, ns_lp - 1
          mc1 = stmask( is   + is_offset )
          DO k = 1 , nz_l
            r(+ (k-1)*ldx*ldy) =
              rcvbuf(k + offset, ipz)
          END DO
          offset = offset + nz_l
        END DO
        rdone( ipz ) = .TRUE.
      END IF
    END DO
    IF( .NOT. ALL( rtest ) ) GO TO 111
```

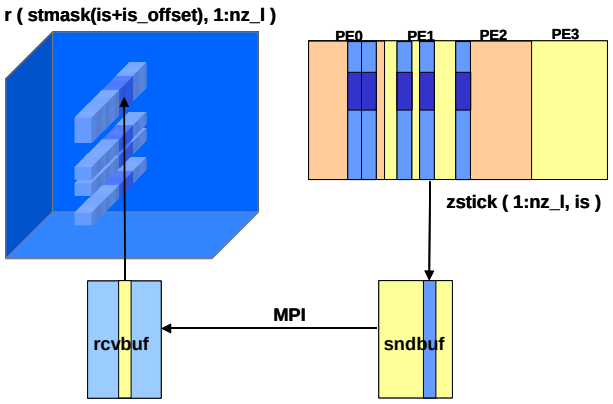

Figure 6. Transposition using buffers and MPI.

By using CAF it is possible to avoid buffering operation and transfer data directly from/to user arrays.

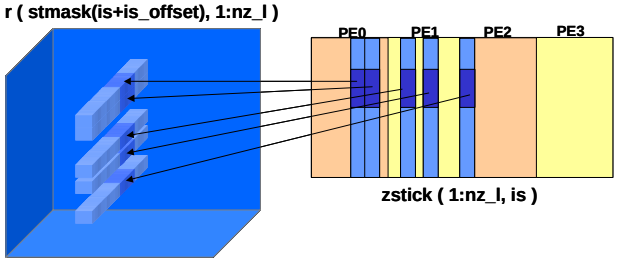

Figure 7. Transposition using CAF.

This can be implemented with a pretty simple coding, once proper pointers are defined:

```
TARGET zstick
TYPE CAFP
  COMPLEX, DIMENSION(:), POINTER :: p
END TYPE CAFP
TYPE (CAFP) pzstick[*]
 …
pzstick%p => zstick(1:n)
call sync_all()
 …
DO ipz = 1, npz
  k_start = ( me - 1 )  * dfft%npp( ipz )
  is_offset = dfft%iss( ipz )
  DO is = 1, ns_lp
    mc1 = stmask( is   + is_offset )
    DO k = 1 , nz_l
      r( mc1 + (k-1)*ldx*ldy ) =
        pzstick[ipz]%p(k_start+k+(is-1)*ldz)
    END DO
  END DO
END DO
```

No modifications or optimizations have been performed so far on the Cray XD1 and XT3 systems. Standard code is supported by the PGI compiler and FFTW routines.

Benchmarks

The physical system considered for our benchmarks is a super-cell (37.33 atomic units) that contains 256 water molecules. The plane waves basis set for wave functions has been truncated with an energy cut-off of 70Rydberg. The effects of the inner shell, are described by the ab initio pseudopotentials of Troullier-Martins[7] that produce valence orbitals smooth within the core regions. Exchange and correlation energy contributions have been evaluated using the Becke-Lee-Yang-Parr (BLYP) functional[8].

For this benchmark system relevant array sizes are reported in Table 1, with these values the overall allocated memory is 32Gbyte.

FFT Grid	Nx, Ny, Nz	220, 220, 220
Electronic states	Nb	1024
Plane waves (wave functions)	Ngw	513171
Plane waves (charge density)	Ng	4105867
Number of atoms	Nat	768

Table 1. Main dimensions of the 256 H2O molecules system used for the benchmark.

Benchmark Results

The H2O – 256 molecules test has been executed on several systems described in Table 2.

System	Processor	Peak cpu Gflop/s	Inter connect	Ncpus
Cray X1	Cray MSP 800 MHz	12.8	Custom Cray	16 – 64
Cray XD1	Opteron 2.4 GHz	4.8	Cray RapidArray	16 – 64
Cray XT3	Opteron 2.4 GHz	4.8	Cray SeaStar	64 -96
IBM p690	IBM Power4 1.3 GHz	5.2	IBM HPS Federation	32 – 128
IBM Blade Center HS20	Intel Xeon 3.06 GHz	6.1	Myrinet LAM C-D	32 – 128

Table 2. Computer systems used for the benchmark.

In Table 3, the performance results obtained on the Cray system (and on other systems available at Cineca) are displayed.

CPUs	Cray X1	Cray XD1	Cray XT3	IBM p690	IBM HS20
16	74	254			
32	45	142		233	293
64	27	68	62	110	153
128				61	93

Table 3. Benchmark results. Values are in seconds, and they are obtained averaging over 10 Car-Parrinello molecular dynamics steps.

Note that the Cray XT3 system is running pre-release software. As soon as an improved Portals communication layer becomes available, system performance is expected to improve significantly.

Figures 8 and 9 show the timings and the performance of the code on the systems considered.

[7] N. Troullier and J.L. Martins, Phys. Rev. B 43, 1993 (1991).

[8] A.D. Becke, Phys. Rev. A 38, 3098 (1988). C. Lee, W. Yang, and R.G. Parr, Phys. Rev. B 37, 785 (1988).

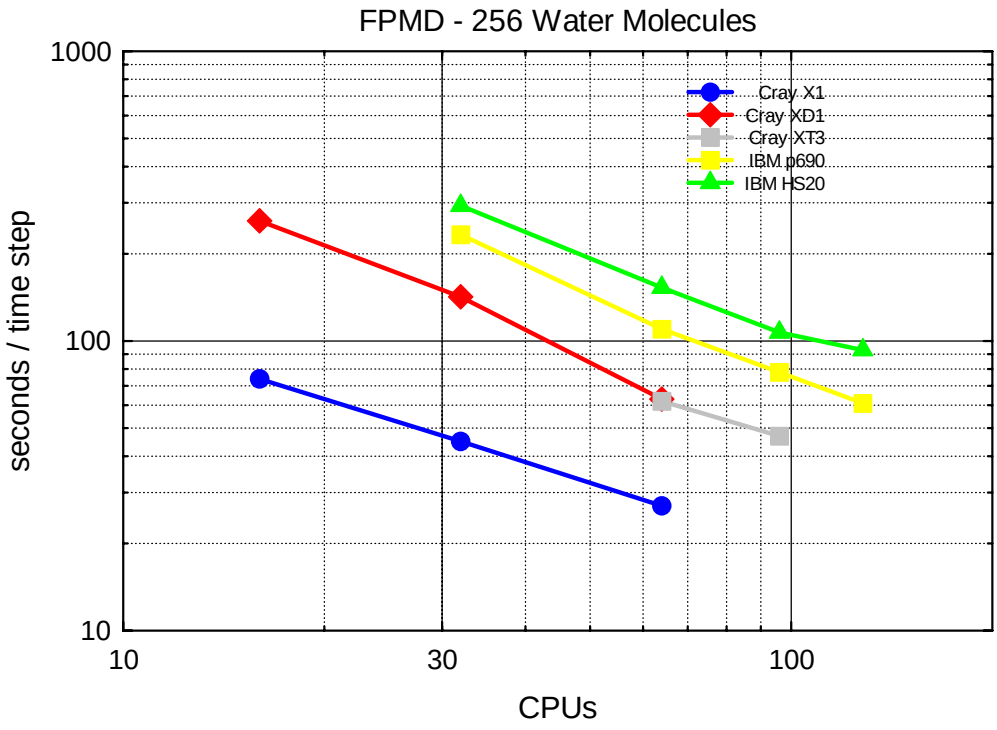

Figure 8. FPMD timings (seconds) on H2O 256 molecules benchmark.

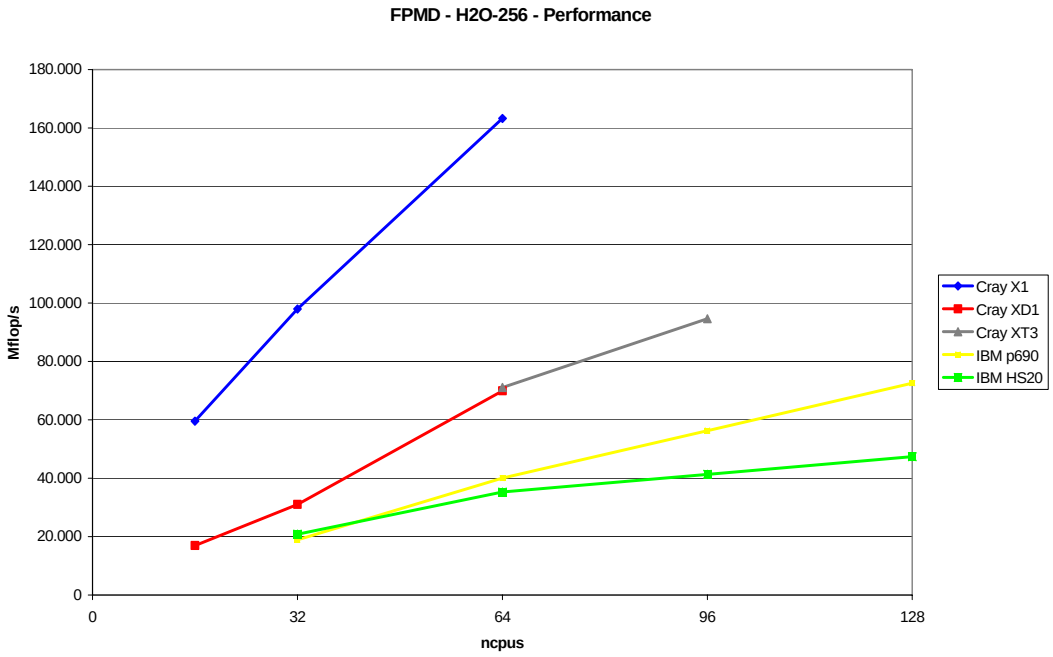

Figure 9. FPMD performance (Mflop/s) on H2O 256 molecules benchmark.

In Table 4, the timings for the FFT section of the algorithms obtained on 64 processors, are displayed.

System	FFT comp	FFT transpose	FFT total
Cray X1	7.90	3.19	11,10
Cray XD1	14.90	12.00	26,90
Cray XT3	15.32	10.57	25,89
IBM p690	19.37	20.13	39,50
IBM HS20	24.05	22.94	46,98

Table 4. FFT timings on 64 processors. Values are in seconds, and they are obtained averaging over 10 Car-Parrinello molecular dynamics steps.

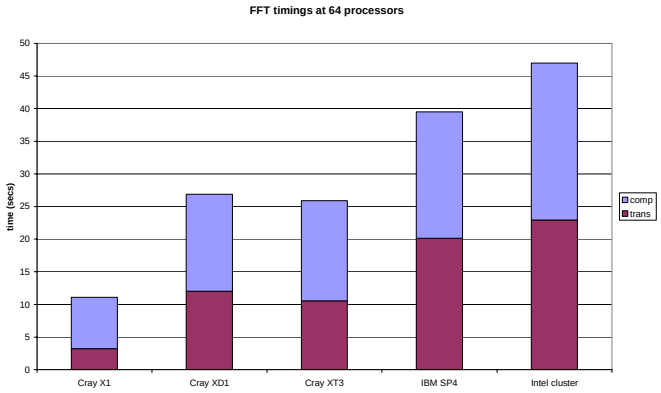

Figure 10. FPMD FFT timings (seconds) on H2O 256 molecules benchmark.

Performance Considerations

We can observe that, in spite of the nominal peak performance (see Table 2), Cray XD1 and Cray XT3 are faster than the IBM p690 and IBM HS20 clusters.
An explanation for that can be found in the different memory architecture and interconnect performance.

While Opteron-based systems can exploit Direct Connect Hypertransport memory connections running at 6.4 Gbyte/s peak, the other systems connect multi-cpu nodes to memory through a memory bus.

In particular, on IBM HS20 Xeon node, the memory and the 512 KB level 2 cache are accessed through a memory bus shared between two processors. This limits the performance, especially for floating point and memory intensive application.

On IBM p690 the Power4 processor hosts a larger 1.44 MBytes level2 cache shared between two processors, but the high processor peak performance is only reachable if both the floating point units are used; since this is not always true for a user application, only when IBM proprietary ESSL library is used it is possible to sustain a number of floating point operations per clock close to 4.

Furthermore, interconnect performance plays an important role in global performance, especially for large processor counts. This is particularly evident in the case of the 3D FFT transform: as shown in Figure 9 the time spent in data movement across the distributed memory is about twice on IBM p690 and HS20 clusters than that on Cray Opteron-based systems. This is due to the superior performance of Cray custom RapidArray and SeaStar interconnects.

Completely different considerations should be drawn about the Cray X1 vector system.

In some cases computation can exploit the vector processors by using highly tuned library routines (for example, LibSci FFT): this implementation has generally required significant code modifications.

On other computational kernels, the Cray X1 is unable to achieve significant fractions of peak performance due to the scalar nature of the algorithms involved (for example, orthogonalization).

Usage of CAF syntax allows the Cray X1 to achieve very good performance on the FFT transpose algorithm. Of course this optimization has required significant code modifications, but this can often be seen as an algorithm simplification producing a more readable code.

Conclusion and future work

FPMD code has been shown to be extremely portable to Cray systems: apart from the Cray X1 vector system, no porting modification has been required to get the code to run efficiently on the systems considered.

In general, FPMD shows good scalability features, in some cases emphasized by high performance network interconnects.

We plan to extend the work performed so far to the other components of the Quantum Espresso suite; and we expect to get a confirmation to its portability and scalability features.