



# Dual-level parallelism for ab initio molecular dynamics: Reaching teraflop performance with the CPMD code

Jürg Hutter<sup>a</sup>, Alessandro Curioni<sup>b,\*</sup>

<sup>a</sup> *Physical Chemistry Institute, University of Zurich, Winterthurerstrasse 190, 8057 Zurich, Switzerland*

<sup>b</sup> *IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland*

Received 15 December 2003; revised 3 November 2004; accepted 28 December 2004

---

## Abstract

We show teraflop performance of the fully featured ab initio molecular dynamics code CPMD on an IBM pSeries 690 cluster. A mixed distributed-memory, coarse-grained parallel approach using the MPI library and shared-memory, fine-grained parallelism using OpenMP directives is used to optimally map the algorithms on the available hardware. The top performance achieved is  $\approx 20\%$  of the peak performance and an estimated parallel efficiency of  $\approx 45\%$  on 1024 processors for a system of 1000 atoms. The main limiting factor of parallel efficiency was found to be the latency of the interconnect.

© 2005 Elsevier B.V. All rights reserved.

PACS: 71.15Pd

Keywords: Car–Parrinello molecular dynamics; Parallelization; OpenMP; MPI

---

---

\* Corresponding author.

E-mail addresses: [hutter@pci.unizh.ch](mailto:hutter@pci.unizh.ch) (J. Hutter), [cur@zurich.ibm.com](mailto:cur@zurich.ibm.com) (A. Curioni).

URLs: <http://www.unizh.ch/pci> (J. Hutter), <http://www.zurich.ibm.com/deepcomputing> (A. Curioni).

## 1. Introduction

Ab initio molecular dynamics is the combination of first-principles electronic structure methods with molecular dynamics based on Newton's equation of motion. The use of electronic structure methods to calculate the interaction potential between atoms overcomes the main shortcomings of the otherwise highly successful pair potential approach. Many-body effects are included, it is parameter-free, and is able to adjust to new chemical situations that may be encountered during a simulation, for example when chemical reactions or structural phase transitions occur.

In their seminal paper [1], Car and Parrinello introduced a new method that allows the efficient propagation of the electronic wave function together with the atomic cores. Although the method is very general, it is primarily used together with the Kohn–Sham approach to density-functional theory. The method has proved to be valuable in many fields. Recent applications include topics in traditional solid-state physics, surface science, interfaces, glasses and amorphous systems, liquids and solutions, catalysis and other chemical reactions, as well as problems from biophysics and biochemistry. For overviews of applications, see recent review papers [2–4].

The combination of a computationally demanding electronic structure method with molecular dynamics requiring thousands of force evaluations make ab initio molecular dynamics simulations highly dependent on high-performance computing resources. Many parallel implementations following various strategies have been reported in the literature [2,5–9] over the past decade. To be able to push the applications from originally a few atoms to now routinely several hundreds of atoms, it was instrumental to adapt algorithms and implementations to modern massively parallel architectures. We report here on the latest developments incorporated into the Car–Parrinello molecular dynamics code CPMD [10] to achieve teraflop performance on clustered SMP servers as the IBM pSeries 690 cluster; this is to our knowledge the first demonstration of teraflop performance for ab initio molecular dynamics. The same developments could also be useful to increase the efficiency of the code on loosely coupled clusters such those available in computational GRIDs.

The CPMD code is based on the original code by Car and Parrinello [1]. It is a production code with many unique features written in Fortran 77 (currently about 150,000 lines of code). Besides the standard Car–Parrinello method, the code also provides an option to compute many different types of properties [11–14], the inclusion of quantum effects on nuclei with the path integral method [15], and interfaces for QM/MM calculations [16]. Since January of 2002, the source code is freely available for noncommercial use. Several thousand registered users in more than 50 countries have compiled and run the code on platforms as diverse as notebooks and computers at the top of the TOP500 list.

## 2. Car–Parrinello molecular dynamics

The Car–Parrinello method [1] starts with an extended Lagrangian

$$\begin{aligned} \mathcal{L}_{\text{CP}} = & \sum_I \frac{1}{2} M_I \dot{\mathbf{R}}_I^2 + \sum_i \frac{1}{2} \mu_i \langle \dot{\psi}_i | \dot{\psi}_i \rangle - E_{\text{KS}}[\{\psi_i\}, \{\mathbf{R}_I\}] \\ & + \sum_{ij} A_{ij} (\langle \psi_i | \psi_j \rangle - \delta_{ij}), \end{aligned} \quad (1)$$

where  $M_I$  and  $\mathbf{R}_I$  are the mass and position of nuclei  $I$ ,  $\mu$  is a fictitious electron mass,  $\psi_i$  are the Kohn–Sham orbitals, and  $E_{\text{KS}}$  is the Kohn–Sham energy. The last term in Eq. (1) is a holonomic constraint to ensure orthogonality of the orbitals. From the Lagrangian, the equations of motion can be derived

$$M_I \ddot{\mathbf{R}}_I(t) = - \frac{\partial}{\partial \mathbf{R}_I} E_{\text{KS}}[\{\psi_i\}, \{\mathbf{R}_I\}], \quad (2)$$

$$\mu_i \ddot{\psi}_i(t) = - \frac{\delta}{\delta \psi_i^*} E_{\text{KS}}[\{\psi_i\}, \{\mathbf{R}_I\}] + \sum_j A_{ij} \psi_j. \quad (3)$$

The equations of motion are integrated using a velocity Verlet scheme. This makes it necessary to compute the forces on the ions  $-\frac{\partial}{\partial \mathbf{R}_I} E_{\text{KS}}$  at each time step during the simulation and the orbitals  $-\frac{\delta}{\delta \psi_i^*} E_{\text{KS}}$ , as well as to determine the Lagrange multipliers  $A_{ij}$  [17].

The Kohn–Sham energy is defined as

$$\begin{aligned} E_{\text{KS}}[\{\psi_i\}, \{\mathbf{R}_I\}] = & \sum_i^{\text{occ}} f_i \int d\mathbf{r} \psi_i^*(\mathbf{r}) \left( -\frac{1}{2} \nabla^2 \psi_i(\mathbf{r}) + \int d\mathbf{r}' V_{\text{ext}}(\mathbf{r}) \rho(\mathbf{r}') \right) \\ & + \frac{1}{2} \int \int d\mathbf{r} d\mathbf{r}' \frac{\rho(\mathbf{r}) \rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} + E_{\text{XC}}[\rho] + E_{\text{I}}[\{\mathbf{R}_I\}], \end{aligned} \quad (4)$$

where  $V_{\text{ext}}$  is the ionic potential,

$$\rho(\mathbf{r}) = \sum_i f_i |\psi_i(\mathbf{r})|^2, \quad (5)$$

the electronic charge density,  $f_i$  the occupation numbers,  $E_{\text{XC}}$  the exchange–correlation energy, and  $E_{\text{I}}$  the electrostatic ion–ion interaction. Periodic boundary conditions allow the expansion of the wave functions  $\psi_i$  in plane waves. Considering the  $\Gamma$ -point representation of the Brillouin zone, the Kohn–Sham orbitals are written as

$$\psi_i(\mathbf{r}) = \frac{1}{\sqrt{\Omega}} \sum_{\mathbf{G}} c_i(\mathbf{G}) e^{i\mathbf{G}\cdot\mathbf{r}}, \quad (6)$$

where  $\Omega$  is the volume of the simulation cell,  $\mathbf{G}$  the reciprocal lattice vectors, and  $c_i(\mathbf{G})$  the Fourier coefficients of orbital  $i$ . The basis set for the expansion in Eq. (6) is reduced to a finite set by truncating the sum over  $\mathbf{G}$  to include only those plane waves with a kinetic energy  $\frac{1}{2}\mathbf{G}^2$  less than a given energy cutoff  $E_c$ . To keep the energy cutoff as low as possible, pseudopotentials are used to remove core electrons and fast oscillating parts of the valence wave functions. Norm-conserving pseudopotentials in the Kleinman–Bylander form result in an external potential of the form

$$V_{\text{ext}}(\mathbf{r}, \mathbf{r}') = V_{\text{loc}}(\mathbf{r}) + V_{\text{nl}}(\mathbf{r}, \mathbf{r}'). \quad (7)$$

Some of the operators in Eq. (4) are diagonal in reciprocal space, whereas others are diagonal in real space. The optimum algorithm is therefore to use fast Fourier transforms to transform quantities between the two spaces. Substituting the plane-wave expansion into the energy expression, one obtains

$$E_{\text{KS}}[\{\psi_{ij}\}, \{\mathbf{R}_I\}] = E_{\text{kin}} + E_{\text{XC}} + E_{\text{loc}} + E_{\text{nl}} + E_{\text{H}} + E_{\text{pair}}, \quad (8)$$

with

$$\begin{aligned} E_{\text{kin}} &= \frac{1}{2} \sum_i f_i \sum_{\mathbf{G}} \mathbf{G}^2 |c_i(\mathbf{G})|^2; & E_{\text{XC}} &= \int d\mathbf{r} F_{\text{XC}}[\rho]; \\ E_{\text{loc}} &= \Omega \sum_{\mathbf{G}} \rho^*(\mathbf{G}) S(\mathbf{G}) V_{\text{loc}}(\mathbf{G}); & E_{\text{nl}} &= \sum_I \sum_i f_i w |F_i^I|^2; \\ E_{\text{H}} &= 2\pi\Omega \sum_{\mathbf{G}} \rho_{\text{tot}}^* \frac{1}{\mathbf{G}^2} \rho_{\text{tot}}, \end{aligned}$$

where  $F_{\text{XC}}$  is the exchange-correlation functional,  $S(\mathbf{G}) = \sum_l e^{-i\mathbf{G}\cdot\mathbf{R}_l}$  the structure factor, and

$$F_i^I = \sum_{\mathbf{G}} c_i(\mathbf{G}) e^{i\mathbf{G}\cdot\mathbf{R}_I} V_{\text{nl}}(\mathbf{G}). \quad (9)$$

We have assumed only one type of atoms with Fourier components of the local potential  $V_{\text{loc}}(\mathbf{G})$  and a single nonlocal contribution  $V_{\text{nl}}(\mathbf{G})$ . The total charge distribution  $\rho_{\text{tot}}$  is calculated from

$$\rho_{\text{tot}}(\mathbf{G}) = \rho(\mathbf{G}) + \sum_I \rho_{\text{ion}}(\mathbf{G}) e^{-i\mathbf{G}\cdot\mathbf{R}_I}, \quad (10)$$

where  $\rho_{\text{ion}}$  is a Gaussian charge centered at the ionic positions.  $E_{\text{pair}}$  is a trivial pair potential defined by the ionic interaction and the Gaussian charge  $\rho_{\text{ion}}$ .

The forces on the wave functions and ions are easily derived from the energy expression. For the wave function forces we find

$$\frac{\partial E_{\text{KS}}}{\partial c_i^*(\mathbf{G})} = \frac{1}{2} \mathbf{G}^2 f_i c_i(\mathbf{G}) + \sum_{\mathbf{G}'} V_{\text{KS}}^*(\mathbf{G} - \mathbf{G}') c_i(\mathbf{G}') + \sum_I F_i^{I*} w e^{-i\mathbf{G}\cdot\mathbf{R}_I}, \quad (11)$$

with the Kohn–Sham potential  $V_{\text{KS}}$  defined by

$$V_{\text{KS}}(\mathbf{G}) = V_{\text{loc}}(\mathbf{G}) S(\mathbf{G}) + V_{\text{XC}}(\mathbf{G}) + 4\pi \frac{\rho_{\text{tot}}(\mathbf{G})}{\mathbf{G}^2}. \quad (12)$$

The forces on the ions

$$\frac{\partial E_{\text{KS}}}{\partial \mathbf{R}_{I,s}} = \frac{\partial E_{\text{loc}}}{\partial \mathbf{R}_{I,s}} + \frac{\partial E_{\text{nl}}}{\partial \mathbf{R}_{I,s}} + \frac{\partial E_{\text{H}}}{\partial \mathbf{R}_{I,s}} + \frac{\partial E_{\text{pair}}}{\partial \mathbf{R}_{I,s}}, \quad (13)$$

where

$$\frac{\partial E_{\text{loc}}}{\partial \mathbf{R}_{I,s}} = -\Omega \sum_{\mathbf{G}} i\mathbf{G}_s V_{\text{loc}}(\mathbf{G}) e^{-i\mathbf{G}\cdot\mathbf{R}_I} \rho^{\star}(\mathbf{G}), \quad (14)$$

$$\frac{\partial E_{\text{nl}}}{\partial \mathbf{R}_{I,s}} = \sum_i f_i \left( F_i^{\dagger\star} w \frac{\partial F_i^I}{\partial \mathbf{R}_{I,s}} + \frac{\partial F_i^{\dagger\star}}{\partial \mathbf{R}_{I,s}} w F_i^I \right), \quad (15)$$

$$\frac{\partial E_{\text{H}}}{\partial \mathbf{R}_{I,s}} = -\Omega \sum_{\mathbf{G}} i\mathbf{G}_s \frac{\rho_{\text{tot}}^{\star}(\mathbf{G})}{\mathbf{G}^2} \rho_{\text{ion}}(\mathbf{G}) e^{-i\mathbf{G}\cdot\mathbf{R}_I}, \quad (16)$$

are calculated most efficiently together with the energy.

The velocity Verlet integrator for the Car–Parrinello equations of motion [17] requires the calculation of two rotation matrices ( $X, Y$ ) for the SHAKE/RATTLE algorithm. They can be obtained from

$$XX^{\dagger} + XB + B^{\dagger}X^{\dagger} = I - A \quad (17)$$

and

$$Y = -\frac{1}{2}(Q + Q^{\dagger}), \quad (18)$$

where the matrices  $A, B$ , and  $Q$  are overlap matrices of different sets of wave function and wave-function velocity coefficients. Whereas matrix  $Y$  can be calculated directly, an iterative scheme has to be employed to solve the nonlinear equation for  $X$ .

### 3. Single CPU optimization

At the  $\Gamma$ -point the wave functions can be chosen to be real, introducing the symmetry

$$c_i(\mathbf{G}) = c_i^{\star}(-\mathbf{G}). \quad (19)$$

This symmetry is used in CPMD to reduce memory requirements and operation count. Owing to this symmetry many basic operations can be performed using real arithmetics although the coefficients  $c_i(\mathbf{G})$  are complex numbers. The prototype of such a basic operation is the inner product of two coefficient vectors

$$\begin{aligned} \sum_{\mathbf{G}} A^{\star}(\mathbf{G})B(\mathbf{G}) &= A(\mathbf{0}) * B(\mathbf{0}) + 2 \sum_{\mathbf{G}_{x \geq 0}} \Re e(A(\mathbf{G}))\Re e(B(\mathbf{G})) \\ &\quad + \Im m(A(\mathbf{G}))\Im m(B(\mathbf{G})). \end{aligned} \quad (20)$$

Note that the  $\mathbf{G} = 0$  has to be treated specially. These inner products are usually encountered within loops over all states in order to calculate overlap matrices

$$S_{ij} = \sum_{\mathbf{G}} A_i^{\star}(\mathbf{G})B_j(\mathbf{G}). \quad (21)$$

We have implemented this operation using BLAS3 routines, enabling both minimal operation count and maximal speed. This is made possible by the special arrangement of COMPLEX\*16 numbers as two contiguous REAL\*8 numbers in FORTRAN

```
CALL DGEMM (T, N, N, N, 2, 2*M, A, 2*M, B, 2*M, O, S, N)
CALL DGER (N, N, -1, A, 2*M, B, 2*M, S, N)
```

( $M$  is the number of plane waves,  $N$  the number of states).

For symmetric matrices  $S = A^\dagger A$  this can be further optimized.

```
CALL DSYRK (U, T, tt N, 2*M, 2, A, 2*M, O, S, N)
CALL DGER (N, N, -1, A, 2*M, A, 2*M, S, N)
```

The rank-1 update (DGER) is used to correct for the double counting of the  $\mathbf{G} = 0$  terms in the matrix multiplies.

Another basic kernel operation is the rotation of a set of vectors by a real matrix

$$B_i(\mathbf{G}) = \sum_j A_j(\mathbf{G}) S_{ji}. \quad (22)$$

This can easily be translated into a BLAS3 call.

```
CALL DGEMM (N, N, 2*M, N, N, 1, A, 2*M, S, N, O, B, 2*M)
```

#### 4. Fast Fourier Transforms in CPMD

Fast Fourier Transforms (FFT) are an essential part of all plane-wave calculations. In fact, it is the near-linear scaling of FFTs that make large-scale density-functional theory (DFT) calculations with plane-wave basis sets possible. FFTs are needed to transform the charge density and local potential between real space grid and reciprocal space. The number of these transforms is fixed and does not depend on the system size. On the other hand the transforms of wave functions from reciprocal space to real space and back (needed in the force calculation) has to be done for each state and therefore scales with the number of electrons. The overall scaling is  $NM \log M$  and dominates the execution time for small and medium-sized systems. Only for large systems (number of atoms greater than 1000) do the cubic scaling inner products and orbital rotations discussed in the preceding section become predominant.

The real-space grid has to be of sufficient size to accommodate the charge density and local potential. These quantities have to be expanded in plane waves with a cutoff four times greater than the one used for the wave functions. Therefore, only a fraction of the grid points in reciprocal space are nonzero for a wave-function array. We can make use of this by limiting the one-dimensional (1D) FFTs needed in the first (last) step of the full three-dimensional FFT to the nonzero columns. This introduces

additional logic in the 3D FFT routine but reduces the operation count by about half (depending on the real-space grid and the symmetry of the simulation cell).

Using the sparsity of the wave-function arrays in the full grid representation makes it impossible to use 3D FFT routines from libraries. We use multiple 1D FFT routines as the basic building block of our optimized 3D FFT kernels. This kernel exploits the rotation algorithm introduced by Goedecker [18] to optimize memory access and vector length in the 1D FFT routines.

Finally, it is also possible to make use of the symmetry of the wave functions in the FFT routines. As the wave functions are real on the real-space grid, we can transform two functions at the same time by defining a new function  $F = A + iB$ . We then obtain for  $\bar{X} = \text{FFT}\{X\}$

$$\bar{A} = \mathcal{R}e\bar{F}; \quad \bar{B} = \mathcal{I}m\bar{F}$$

and

$$A = \frac{1}{2}\mathcal{R}e[F(\mathbf{G}) + F(-\mathbf{G})] + \frac{i}{2}\mathcal{I}m[F(\mathbf{G}) - F(-\mathbf{G})],$$

$$B = \frac{1}{2}\mathcal{I}m[F(\mathbf{G}) + F(-\mathbf{G})] - \frac{i}{2}\mathcal{R}e[F(\mathbf{G}) - F(-\mathbf{G})].$$

This allows a single complex-to-complex FFT kernel to be used for all transforms and at the same time reduces the operation count by a factor of almost 2.

## 5. Parallelization of CPMD

Various strategies were followed in parallel implementations of plane-wave/pseudopotential codes [2,5–7,9]. Parallelization of the CPMD code was done on different levels. The central parallelization is based on a distributed-memory coarse-grain algorithm, which is a compromise between load balancing, memory distribution and parallel efficiency. This scheme achieves good performance on computers with up to about 200 CPUs, depending on system size and communication speed [9]. To overcome to this limited scalability (which is common to all the previous cited parallelization strategies), on top of the basic scheme a fine-grain, shared-memory parallelization was implemented. The two parallelization methods are independent and can be mixed. This allows us to achieve good performance on distributed computers with shared-memory nodes and several thousands of CPUs, and also to extend the size of the systems that can be studied completely ab initio to several thousand atoms.

Some methods implemented in CPMD allow a further level of parallelization. These methods, such as path-integral molecular dynamics or linear response theory are embarrassingly parallel on the level of the energy calculation. Typically two to sixteen copies of the energy and force calculation can be run in parallel. For these methods, an efficient use of computers with tens of thousands of CPUs can be envisaged. However, in this work only the main Car–Parrinello molecular dynamics part of the code has been used.

### 5.1. Distributed-memory parallelization

Our coarse-grain, distributed-memory parallelization is driven by the distribution of wave-function coefficients for all states to all CPUs. Real-space grids are also distributed, whereas all matrices that do not include a plane-wave index are replicated (especially overlap matrices). All other arrays are only distributed if this does not cause additional communications. With this scheme, all loops over plane waves, especially the ones having a  $N^2M$  scaling (overlap matrices, orbital rotations), are parallel. It also requires a parallel 3D FFT. Further requirements to optimize the Fourier transforms are used to find the optimal data distribution. The 3D FFT can be seen as performing the following steps:

- (1) Scatter of data  $C(x, y, z) \leftarrow c(\mathbf{G})$ .
- (2) 1D transforms along direction  $x$ .
- (3) 1D transforms along direction  $y$ .
- (4) 1D transforms along direction  $z$ .

For a general data distribution in both spaces, each of the above steps would include communication between all processors. The data distribution in CPMD tries to minimize the number of communication steps while maintaining optimum load balancing in both spaces. To achieve this goal we try to fulfill the following requirements:

- Each processor has the same number of plane waves.
- All plane waves with common  $y$  and  $z$  components are on the same processor.
- The number of different  $(y, z)$  pairs of plane-wave components is the same on each processor.
- A processor hosts full planes of real-space grid points.
- The number of real-space planes is the same on each processor.

This scheme requires only a single data communication step after the first (or before the last) 1D transform. In addition, we can make use of the sparsity of the wave-function representation still present after the first transform and only communicate nonzero elements. The various load-balancing requirements are interrelated, and we use an heuristic algorithm to achieve near-optimum results. Our experience is that for all cases good load balancing is achieved for the reciprocal space. The restriction to full-plane distributions in real space, however, introduces severe problems in the case of a large number of processors. The number of planes available is typically about 50 for small systems and 200–300 for large systems. This restricts the maximum number of processors that can be used efficiently. The coarse granularity of this approach is also responsible for the appearance of magic numbers of processors where especially good performance can be achieved. This is no major problem because the appearance of these numbers is fully transparent.

The efficiency of the scheme described above is limited due to the following problems:

- Global summation of overlap matrices and broadcast of matrices scales as  $N_{\text{pe}} \log N_{\text{pe}}$  and will become predominant for large numbers of processors ( $N_{\text{pe}}$  is the number of processors).
- The calculation of the rotation matrix in the SHAKE/RATTLE algorithm is not parallel and limits the maximal speedup that can be achieved.
- Replicated overlap matrices might become a memory bottleneck for large systems on many processors with small memory.
- The number of grid points in  $x$  direction limits the maximum number of processors that can be used efficiently in the 3D FFT.
- The time required for the all-to-all communications scale as  $N_{\text{pe}} * \text{Latency}$ , downgrading the performance scaling in the case of communication adapters with relatively high latency.

Distributed-memory parallelization is implemented in CPMD using the MPI library.

### 5.2. Shared-memory parallelization

Shared-memory parallelization on the loop level is achieved by using OpenMP compiler directives and multithreaded libraries (BLAS and FFT) if available. In our test, the multithreaded library ESSL has been used [19]. Compiler directives have been used to ensure parallelization of all longer loops (those that depend on the number of plane waves or the number of grid points in real space), and to avoid parallelization on the shorter ones. This type of parallelization is independent of the MPI parallelization and can be used alone or combined with the distributed-memory approach. Tests on various shared-memory computers have shown that efficient parallelization up to 16 processors can be achieved. It is not surprising that loop-level parallelism is very effective in CPMD. The code has a vectorization degree of more than 99% and routinely reaches more than 75% efficiency on vector processors.

The combined approach is especially interesting for the following reasons:

- The shared-memory parallelization is also effective in the serial parts of the distributed-memory scheme, e.g. the rotation matrix in the RATTLE/SHAKE algorithm and overlap matrixes. This allows calculations on systems of several thousands of atoms.
- For a given total number of processors  $N_{\text{pe}}$ , the number of tasks involved in the distributed-memory parallelization can easily be decreased by one order of magnitude, reducing drastically the impact of the latency in the all-to-all communications, and obtaining good scaling behavior for up to thousands of processors or enhancing the performance on loosely coupled clusters.

## 6. Benchmarks

### 6.1. Hardware used

All calculations have been performed on two different pSeries 690 (Regatta) clusters; the top performance calculations and the demonstration of teraflop performance have been performed on the HPCx system [20] in Daresbury (UK) that consists of 40 Regatta frames, each of which is logically partitioned into four parts to yield a cluster of 160 SMP partitions (8 Power4 1.3 GHz processors per partition) for a total of 1280 processors; these partitions are interconnected with a double-plane Colony switch having a latency of 21  $\mu$ s and a bidirectional bandwidth of 720 MB/s. This supercomputer has a peak performance of  $\approx 6.6$  TFlops and a demonstrated Linpack performance of  $\approx 3.2$  Tflops. It is usually partitioned and, for general use, only a maximum of 128 SMP nodes (1024 processors) are available ( $\approx 5.3$  Tflops of top performance and  $\approx 3.1$  Tflops Linpack performance). The second supercomputer used is the one installed at the IBM Zurich Research Laboratory in Rüschlikon (Switzerland). It consists of eight Regatta frames with a single logical partition to yield eight SMP nodes (32 Power4 1.3 GHz processors per node) for a total of 256 processors; the nodes are interconnected with a double GBit ethernet link. The peak performance of this machine is  $\approx 1.3$  Tflops with a Linpack performance  $\approx 0.6$  TFlops. This computer was used to make the runs on the single Regatta frame.

### 6.2. Performance calculation

As specified in Section 3, a significant effort has been made to minimize the operation count and maximize the speed whenever possible. Therefore any paper-and-pencil estimate of the number of operations during a single ab initio MD step will overestimate the number of operations. We have decided to use the hardware performance monitor `hpmcount` included in the `hpmtoolkit` [21] to read the appropriate hardware counters directly and determine the actual number of floating point operations performed in each test. The overhead associated with the use of `hpmcount` is negligible. The number of operations used to calculate the performance was derived from the runs on a single processor whenever possible, otherwise the actual operation count was used. The maximum overhead introduced by the parallelization in the operation count, i.e. the number of operations with 1024 processors minus the number of operations with one processor, is less than 4%.

### 6.3. Systems investigated

In order to obtain a coherent benchmark of the code, a set of supercell crystalline systems has been considered that are derived from the silicon carbide (SiC) unit cell and contain an increasing number of atoms ranging from 216 to 1000 (see Table 1). Smaller systems run too fast for a reasonable performance evaluation, and larger systems were too time-consuming given the limited time available for benchmarking.

Table 1  
Size of the systems derived from cubic silicon carbide used for the tests

System	Number of atoms	Number of plane waves	FFT grid
<i>I3</i>	216	477,534	$128 \times 128 \times 128$
<i>I4</i>	512	1,131,630	$168 \times 168 \times 168$
<i>I5</i>	1000	2,209,586	$256 \times 256 \times 256$

The systems are called *In*, where *n* can be 3, 4, 5 to indicate the rank of the supercell considered. *I3*, for example, means a system consisting of a  $3 \times 3 \times 3$  supercell of the original simple cubic cell of SiC (which contains four carbon atoms and four silicon atoms). The three systems were chosen such that

- a system (*I3*) fits well on a single Regatta frame, and allows one to check the performance on parallelization-critical regions,
- a system (*I4*) can fit on a single Regatta frame but may exhibit performance bottlenecks only on several Regatta frames,
- a system (*I5*) needs more than one node and exhibits optimum performance on up to 32 Regatta frames.

The number of MPI tasks (and therefore the number of nodes) for the runs on the clustered Regatta frames have been chosen in order to match as closely as possible the number of planes in the FFT grid (see Section 5).

## 7. Performance analysis

### 7.1. System *I3*

System *I3* has been used to check the system performance at the level of a single Regatta frame and to test the scalability of such a relatively small system on the full system.

#### 7.1.1. Single Regatta frame

In Table 2 the results for runs from 1 to 32 processors on a single Regatta frame are reported.

The analysis of these data demonstrate very good efficiency both in the single processor calculation ( $\approx 40\%$  of the top performance) and in the scaling (98%, 95%, 82%, 72%, 57% on 2, 4, 8, 16, 32 processors, respectively). Moreover, the results show clearly that the two parallelization schemes can be mixed efficiently also in runs on relatively small systems and using a relatively small number of processors, i.e. optimal situations for the distributed-memory scheme. Interestingly, the best result on this particular test case is obtained in mixed mode ( $16 \times 2$  vs  $32 \times 1$ ), whereas the average performance loss using mixed-mode (up to 8 SMP thread) is only 8%. The same behavior is observed also when 2 processors are used, where the full

Table 2  
System I3 on a single Regatta

Number of processors	MPI tasks	SMP threads	Time/step (s)	Performance (GFlops)
32	32	1	18.2	34.7
32	16	2	17.8	35.4
32	8	4	18.5	34.1
32	4	8	21.1	29.9
16	16	1	27.8	22.7
16	8	2	27.9	22.6
16	4	4	31.7	19.9
16	2	8	35.2	17.9
8	8	1	48.6	13.0
8	4	2	49.7	12.7
8	2	4	53.3	11.8
8	1	8	65.9	9.6
4	4	1	83.8	7.5
4	2	2	93.0	6.8
4	1	4	100.1	6.3
2	2	1	161.5	3.9
2	1	1	173.2	3.6
1	1	1	313.1	2.0

SMP mode is only  $\sim 8\%$  slower than the full MPI mode. This suggests that our dual-level parallelization scheme could be useful also on dual processor nodes which are commonly used in commodity supercomputers. Note that, however, the performance of our approach is strongly dependent on the characteristics of the hardware used, therefore no straightforward generalization can be made. The worsening of the scaling for the full frame is not due to factors intrinsic to the algorithm, but to memory conflicts due to the overlap of the MPI kernels—which use the shared-memory segments for communication—with the real computation. This results in a reduction of the available memory bandwidth. This becomes clear when the time spent in the various parts of the code is examined in detail. For the two-processor run,  $\approx 52\%$  of the time is spent to calculate the forces acting on both the electronic and ionic degrees of freedom, the orbital rotations involved in the RATTLE/SHAKE algorithm and the building of the overlap matrix (type I) and  $\approx 35\%$  is related to the FFTs (22% real calculation (1D FFT), 11% FFT gather/scatter routines and 2% FFT communication) (type II). For the 32-processor run, we have instead 44% for (I) and 44% for (II) (21% (1D FFT), 15% (FFT gather/scatter), 5% FFT communication). From these numbers, it is clear that in this case the gather/scatter routines are the ones that prevent better scaling and not the communication involved in the FFT. The gather and scatter routines play a fundamental role in reducing the number of operations involved in the calculation (they go from a compressed sparse matrix in reciprocal space to a dense one and vice versa), as explained in Section 4. The indirect addressing used has a nonoptimal memory access and lowers the performance of this part.

On the other hand, the use of full FFTs without data compression would increase the number of operations by a factor of 2, yielding better results in terms of GFlops and scaling but not in terms of real timing.

### 7.1.2. 32 switched Regatta frames

To test the limit of the parallelism in our code, one run has been performed on 32 switched frames; the results are reported in Table 3.

Comparing these results with those in Table 2 we see that one obtains a low parallel efficiency ( $\approx 15\%$ ) going from 1 to 32 frames. However, this can be considered a limiting case. From the analysis of the time spent on the different parts of the code, we see that  $\approx 50\%$  of the total time is spent in the all-to-all communication of the FFT, and that  $\approx 90\%$  of this time is latency-bound. We can therefore estimate that a better communication adapter will improve the performance considerably. Replacing the current double colony switch, having a latency of  $\approx 21 \mu\text{s}$ , with a next-generation federation [22] switch (latency  $\approx 5\text{--}9 \mu\text{s}$ ) will reduce this time by a factor of up to 4 and therefore enhance the global performance by a factor 2 to achieve  $\approx 300$  GFlops.

## 7.2. System I4

System I4 is the largest system that still fits on a single frame (64 GB of memory), and should therefore yield the maximum performance we can achieve on a single frame.

### 7.2.1. Single Regatta frame

In Table 4 the results for runs from 1 to 32 processors on a single Regatta frame are listed.

Again, the analysis of the data demonstrates good efficiency, both in the single-processor calculation ( $\approx 45\%$  of peak performance) and in scaling (96%, 95%, 86%, 82%, 72% on 2, 4, 8, 16, 32 processors, respectively). Also in this case the good mix of the two parallelization schemes is evident, being the average performance loss using mixed-mode (up to 8 SMP thread) only 8% (same result as the I3 test case). This suggests that our dual-level parallelization scheme is weakly influenced by system size. The degradation of the scaling with an increasing number of processors is less pronounced than for I3 because the granularity of the data is improved. An analysis of the time spent in various parts helps explain this result: in the single-processor run  $\approx 75\%$  of the time is spent in routines of the type I (see Section 7.1) and only  $\approx 20\%$  in routines related to the FFT (II) (gather/scatter routine 7%, communication 3%). In the 32-processor run  $\approx 60\%$  is spent in routines of the type I and  $\approx 26\%$  in

Table 3  
System I3 on switched Regatta frames

Number of processors	MPI tasks	SMP threads	Time/step (s)	Performance (GFlops)
1024	128	8	4.28	160

Table 4  
System *I4* on a single Regatta frame

Number of processors	MPI tasks	SMP threads	Time/step (s)	Performance (GFlops)
32	32	1	147.6	51.4
32	16	2	150.9	50.3
32	8	4	153.6	49.4
32	4	8	167.4	45.3
16	16	1	239.9	31.6
16	8	2	247.4	30.7
16	4	4	257.4	29.5
16	2	8	293.1	25.9
8	8	1	430.4	16.5
8	4	2	468.7	16.2
8	2	4	492.5	15.4
4	4	1	741.4	10.2
4	2	2	832.8	9.1
2	2	1	1609.3	4.7
1	1	1	3161.1	2.4

routines related to FFT (II) (gather/scatter routine 10%, communication 3%). The larger size of the system, therefore, prevents the communication and gather/scatter times from becoming predominant, and hence a better scaling is ensured.

### 7.2.2. 40 switched Regatta frames

Also in this case the scaling of the code has been tested on a much larger number of processors: two runs have been performed on 21 and 40 frames, and the results are reported in Table 5.

Comparing the results with those listed in Table 4, we see a parallel efficiency of  $\approx 36\%$  going from 1 to 40 frames and an almost perfect scaling from 21 to 40. The overall parallel efficiency (1–1280 processors) was 20%. The same observations as for system *I3* can be made. More than 50% of the time is spent in the all-to-all communication related to the FFT, and this time is bound by latency. These results are especially very good if compared with any standard distributed-memory-only parallelization, in which the performance is significantly leveled off already with more than 160 processors (number of real-space planes along the *x*-direction) due to the previously described load imbalance.

Table 5  
System *I4* on switched pSeries 690

Number of processors	MPI tasks	SMP threads	Time/step (s)	Performance (GFlops)
672	64	8	19.9	380
1280	160	8	10.7	703

Table 6  
System *I5* on switched Regatta frames

Number of processors	MPI tasks	SMP threads	Time/step (s)	Performance (GFlops)
512	64	8	99.4	563
1024	256	4	71.9	780
1024	128	8	56.3	1017
1232	154	8	52.1	1087

### 7.3. System *I5*

System *I5* cannot fit on a single frame and it is large enough to show the full benefits of a mixed parallelization scheme. The results obtained are listed in Table 6.

In this case we have no reference to determine the global parallel efficiency, but we obtain  $\approx 20\%$  of the peak performance. As the single-processor calculations for the other systems (see *I3* and *I4*) yield a maximum efficiency of  $\approx 40\%$ , we can reasonably expect a parallel efficiency of  $\approx 45\%$ . Moreover, going from 512 to 1024 processors we have a parallel efficiency of close to 90%.

An important point to highlight is the usefulness of the mixed approach. In fact, as discussed above, this helps reduce the number of MPI tasks in all-to-all communications, partly hiding the latency of nonoptimal switches. This is evident when comparing the results of the two runs with 1024 processors. The one with 256 MPI tasks and 4 SMP threads per task is 30% slower than the one with 128 MPI tasks and 8 SMP threads per task. A run with 1024 MPI tasks will yield almost no performance increase over a 256-processor run, owing to load-balancing and communication problems. Following the same lines we can state that if  $N$  is the maximum number of MPI tasks for which the application scale (i.e. the elapsed time decrease increasing the number MPI tasks, usually equal to the number of real-space planes in the  $x$ -direction), our mixed scheme will enhance the scaling up to a number of processors  $N * M$ , where  $M$  is the number of SMP thread per MPI task; the speedup due to the SMP scheme will depend on the size of the systems, being more important on larger systems where an additional speedup come from the parallelization of the linear algebra involved in the orthogonalization. For this reasons, even if our approach could help to boost the scaling also on commodity clusters with dual processors nodes, it is particularly appealing on clustered SMP servers, with many cpus per node and a good memory system.

Last but not least, the teraflop runs resulted in a time-per-step of less than 1 min, which is in the range of values that allows us to perform realistic ab initio simulations (up to 3 min per step).

## 8. Summary

The performance of the ab initio molecular dynamics code CPMD has been tested on an IBM p690 series computer with up to 1280 processors. The maximum

performance achieved exceeded 1 Tflop for a system consisting of 1000 atoms. This constitutes  $\approx 20\%$  of the peak performance (33% of the Linpack-Benchmark performance) and an overall parallel efficiency (calculated with an assumed single processor performance estimated from smaller systems) of 45%. This has to be compared with the maximum performance that can be achieved using the system BLAS3 library calls of about 66% of peak performance on a Power4 chip. Tests with smaller systems reveal that performance bottlenecks are the memory bandwidth on the shared-memory nodes and the latency of the node interconnect. Dual-level parallelism can be used on almost all modern massively parallel computers, from dual CPU node Linux clusters to Constellation-type systems with large SMP nodes and even clustered vector processor machines. The novel mixed parallelization scheme presented here is opening new frontiers for the ab initio study of large molecular systems with several thousands of atoms on modern supercomputers.

### Acknowledgements

We thank the HPCx consortium for access to their IBM pSeries 690-based tera-scale facility and especially Tony Kennedy and Arthur Trew for their cooperation. One of us (A.C.) thanks Wanda Andreoni for her continuous help and support.

### References

- [1] R. Car, M. Parrinello, Unified approach for molecular dynamics and density-functional theory, *Physical Review Letters* 55 (1985) 2471–2474.
- [2] D. Marx, J. Hutter, Ab-initio molecular dynamics: Theory and implementation, in: *Modern Methods and Algorithms of Quantum Chemistry*, in: J. Grotendorst (Ed.), NIC Series, vol. 1, FZ Jülich, Germany, 2000, see also <http://www.fz-juelich.de/nic-series/Volume1>.
- [3] W. Andreoni, A. Curioni, New advances in chemistry and materials science with CPMD and parallel computing, *Parallel Computing* 26 (2000) 819–842.
- [4] J.S. Tse, Ab initio molecular dynamics with density functional theory, *Annual Review of Physical Chemistry* 53 (2002) 249–290.
- [5] K.D. Brommer, B.E. Larson, M. Needels, J.D. Joannopoulos, Implementation of the Car–Parrinello algorithm for ab initio total energy calculations on a massively parallel computer, *Computers in Physics* 7 (3) (1993) 350–362.
- [6] L.J. Clarke, I. Štich, M.C. Payne, Large-scale ab initio total energy calculations on parallel computers, *Computer Physics Communications* 72 (1993) 14–28.
- [7] J. Wiggs, H. Jónsson, A parallel implementation of the Car–Parrinello method by orbital decomposition, *Computer Physics Communications* 81 (1994) 1–18;  
J. Wiggs, H. Jónsson, A hybrid decomposition parallel implementation of the Car–Parrinello method, *Computer Physics Communications* 87 (1995) 319–340.
- [8] C. Cavazzoni, G.L. Chiarotti, A parallel and modular deformable cell Car–Parrinello code, *Computer Physics Communications* 123 (1999) 56–76.
- [9] R. Gruber, P. Volgers, A. De Vita, M. Stengel, T.M. Tran, Parametrization to tailor commodity clusters to applications, *Future Generation Computer Systems* 19 (2003) 111–120.
- [10] CPMD V3.8, Copyright IBM Corp. 1990–2003, Copyright MPI für Festkörperforschung, Stuttgart, 1997–2001. See also <http://www.cpmd.org>.
- [11] A. Putrino, D. Sebastiani, M. Parrinello, Generalized variational density functional perturbation theory, *Journal of Chemical Physics* 113 (2000) 7102–7109.

- [12] P.L. Silvestrelli, M. Parrinello, Water molecule dipole in the gas and in the liquid phase, *Physical Review Letters* 82 (1999) 3308–3311.
- [13] J. Hutter, Excited state nuclear forces from the Tamm–Dancoff approximation to time-dependent density functional theory within the plane wave basis set framework, *Journal of Chemical Physics* 118 (2003) 3928–3934.
- [14] A. Curioni, W. Andreoni, Computer simulations for organic light-emitting diodes, *IBM Journal of Research and Development* 45 (1) (2001) 101–113.
- [15] M.E. Tuckerman, D. Marx, M.L. Klein, M. Parrinello, Efficient and general algorithms for path integral Car–Parrinello molecular dynamics, *Journal of Chemical Physics* 104 (1996) 5579–5588.
- [16] M. Eichinger, P. Tavan, J. Hutter, M. Parrinello, A hybrid method for solutes in complex solvents: Density functional theory combined with empirical force fields, *Journal of Chemical Physics* 110 (1999) 10452–10467;  
A. Laio, J. VandeVondele, U. Rothlisberger, A Hamiltonian electrostatic coupling scheme for hybrid Car–Parrinello molecular dynamics simulations, *Journal of Chemical Physics* 116 (2002) 6941–6947;  
T. Mordasini, A. Curioni, W. Andreoni, Why do divalent metal ions either promote or inhibit enzymatic reactions? The case of Bam HI restriction endonuclease from combined quantum–classical simulations, *Journal of Biological Chemistry* 287 (2003) 4381–4384;  
D. Fischer, A. Curioni, W. Andreoni, Decanethiols on gold: The structure of self-assembled monolayers unravelled with computer simulations, *Langmuir* 19 (2003) 3567–3571.
- [17] M.E. Tuckerman, M. Parrinello, Integrating the Car–Parrinello equations. I. Basic integration techniques, *Journal of Chemical Physics* 101 (1994) 1302–1315.
- [18] S. Goedecker, Rotating a three-dimensional array in an optimal position for vector processing: Case study for a three-dimensional fast Fourier transform, *Computer Physics Communications* 76 (1993) 294–300.
- [19] ESSL(Engineering and Scientific Subroutine Library) Version 3.3, Copyright IBM Corporation.
- [20] <http://www.hpcx.ac.uk> The HPCx Consortium, namely UoE HPCX Ltd, is led by the University of Edinburgh ([www.ed.ac.uk](http://www.ed.ac.uk)), with the Central Laboratory for the Research Councils (CLRC) ([www.clrc.ac.uk](http://www.clrc.ac.uk)) and IBM ([www.ibm.com](http://www.ibm.com)). The project is funded by EPSRC ([www.epsrc.ac.uk](http://www.epsrc.ac.uk)).
- [21] hpmtoolkit, version 2.4, <http://www.alphaworks.ibm.com/tech/hpmtoolkit>.
- [22] P. Williams, HPC computing: Strategy and directions, Presentation at Scicomp7 Meeting, Göttingen, Germany, 4 March 2003.