Parallel ab-initio molecular dynamics *

B. Hammer[1], and Ole H. Nielsen[1,2]

[1] Center for Atomic-scale Materials Physics (CAMP), Physics Dept., Technical University of Denmark, Bldg. 307, DK-2800 Lyngby, Denmark,
[2] UNI•C, Technical University of Denmark, Bldg. 304, DK-2800 Lyngby, Denmark.

Abstract. The Car-Parrinello ab-initio molecular dynamics method is heavily used in studies of the properties of materials, molecules etc. Our Car-Parrinello code, which is being continuously developed at CAMP, runs on several computer architectures. A parallel version of the program has been developed at CAMP based on message passing, currently using the PVM library. The parallel algorithm is based upon dividing the "special **k**-points" among processors. The number of processors used is typically 6-10. The code was run at the UNI•C 40–node SP2 with the IBM PVMe enhanced PVM message passing library. Satisfactory speedup of the parallel code as a function of the number of processors is achieved, the speedup being bound by the SP2 communications bandwidth.

1 Car-Parrinello ab-initio molecular dynamics

The term *ab-initio* molecular dynamics is used to refer to a class of methods for studying the dynamical motion of atoms, where a huge amount of computational work is spent in solving, as exactly as is required, the entire quantum mechanical electronic structure problem. When the electronic wavefunctions are reliably known, it will be possible to derive the forces on the atomic nuclei using the Hellmann-Feynman theorem[1]. The forces may then be used to move the atoms, as in standard molecular dynamics.

The most widely used theory for studying the quantum mechanical electronic structure problem of solids and larger molecular systems is the density-functional theory of Hohenberg and Kohn[2] in the *local-density approximation*[2] (LDA). The selfconsistent Schrödinger equation (or more precisely, the Kohn-Sham equations[2]) for single-electron states is solved for the solid-state or molecular system, usually in a finite basis-set of analytical functions. The electronic ground state and its total energy is thus obtained. One widely used basis set is "plane waves", or simply the Fourier components of the numerical wavefunction with a kinetic energy less than some cutoff value. Such basis sets can only be used reliably for atomic potentials whose bound states aren't too localized, and hence plane waves are almost always used in conjunction with *pseudo-potentials*[3] that

* To appear in proceedings of *Workshop on Applied Parallel Computing in Physics, Chemistry and Engineering Science* (PARA95), August 21-24, 1995, ed. J. Wasniewski, Springer Lecture Notes in Computer Science.

effectively represent the atomic cores as relatively smooth static effective potentials in which the valence electrons are treated.

Car and Parrinello's method[4] is based upon the LDA, and uses pseudopotentials and plane wave basis sets, but they added the concept of updating iteratively the electronic wavefunctions simultaneously with the motion of atomic nuclei (electron and nucleus dynamics are coupled). This is implemented in a standard molecular dynamics paradigm, associating dynamical degrees of freedom with each electronic Fourier component (with a small but finite mass). The efficiency of this iteration scheme has opened up not only for the mentioned pseudopotential based molecular dynamics studies, but also for static calculations for far larger systems than had previously been accessible. Part of this improvement is due to the fact that some terms of the Kohn-Sham Hamiltonian can be efficiently represented in real-space, other terms in Fourier space, and that Fast Fourier Transforms (FFT) can be used to quickly transform from one representation to the other.

Since the original paper by Car and Parrinello[4], a number of modifications[5, 6] have been presented that improve significantly on the efficiency of the iterative solution of the Kohn-Sham equations. The modifications include the introduction of the conjugate gradients method[5, 6, 7] and a direct minimization of the total energy[6].

The present work is based upon the solution of the Kohn-Sham equations using the conjugate gradients method. We use Gillan's all-bands minimization method[7] for simultaneously updating all eigenstates, which is important when treating metallic systems with a Fermi-surface.

The Car-Parrinello code (written in Fortran-77) employed by us has been used for a number of years, and has been optimized for vector supercomputers and workstations. On a single CPU of a Cray C90 the code performs at about 350-400 MFLOPS (out of 952 MFLOPS peak), mainly bound by the performance of Cray's complex 3D FFT library routine. On a single node of a Fujitsu VPP-500/32 at the JRCAT computer center in Tsukuba, Japan the code achieves about 500 MFLOPS (out of 1600 MFLOPS peak).

2 A parallel Car-Parrinello algorithm

Given the virtually unlimited need for computational resources required for studying large systems using *ab-initio* molecular dynamics, it is obvious that parallel supercomputers must be employed as vehicles for performing larger calculations. Parallel Car-Parrinello implementations were pioneered by Joannopoulos et al.[8] and Payne et al.[9], and by now several parallel codes are being used[10, 11].

The parallelization approaches for the Car-Parrinello algorithm focus on the distribution of several types of data[10]: 1) the Fourier components making up the plane wave eigenstates of the system, and 2) the individual eigenstates ("bands") for each **k**-point of the calculation. The first approach requires the availability of an efficiently parallelized 3D complex FFT, whereas the second

one does the entire FFT in local memory, but needs to communicate for eigenvector orthogonalization.

The selfconsistent iterations require that a sum over the electronic states in the system's Brillouin-zone be carried out. For very large systems, and especially for semiconductors with fully occupied bands, it may be a good approximation to use only a single **k**-point (usually $\mathbf{k} = \mathbf{0}$) in the Brillouin-zone, and this is done in several of the current implementations.

However, it is our goal to treat systems that consist only of a few dozen atoms, and which typically consist of transition metals with partially occupied d-electron states. This requires rather large plane wave basis sets, as well as a detailed integration over the **k**-points in the Brillouin-zone, in order to define reliably the band occupation numbers of states near the metal's Fermi surface. Hence we typically need to use about a dozen **k**-points.

With such a number of **k**-points, and given that many parallel supercomputers consist of relatively few processors with rather much memory (128 MB or more), it becomes attractive to pursue a parallelization strategy based upon farming out **k**-points to processors in the parallel supercomputer. Since traditional electronic structure algorithms have always contained a serial loop over **k**-points, each iteration being in principle independent of other iterations, this is a much simpler task than the other two approaches referred to above. This approach is not any better than the other approaches, except that it is well suited for the problems that we are studying, and it could eventually be combined with the other approaches in an ultimate parallel code.

The parallelization over **k**-points is in principle straightforward. If we for simplicity assume that our problem contains N **k**-points and we have N processors available to perform the task, each processor will contain only the wavefunctions (one vector of Fourier components for each of the bands) for its own single **k**-point. A very significant memory saving results from each processor only having to allocate memory for its own **k**-point's wavefunctions (in general, $1/N$-th of the **k**-points). The wavefunction memory size is usually the limiting factor in Car-Parrinello calculations. A number of tasks are not inherently parallel: a) Input and output of wavefunctions and other data from a standard data file, b) accumulation of **k**-point-dependent quantities such as charge densities, eigenvalues, etc., c) the calculation of total energy and forces, and the update of atomic positions, and d) analysis of data depending only upon the charge density. These tasks should be done by a "master" task.

We have chosen to implement parallelization over **k**-points by modest modifications of our existing serial Fortran-77 code. Using conditional compilation, the code may be translated into a master task, a slave task, or simply a non-parallel task to be used on serial and vector machines. The master-slave communication is implemented by message-passing calls (send/receives and broadcasts).

The **k**-point parallelization is not as trivial as it might seem at first sight. Even though each iteration of the serial loop over **k**-points is algorithmically independent of other iterations, the wavefunction data actually depend crucially on the result of previous iterations, when the standard Car-Parrinello iterative

update of wavefunctions takes place. When the **k**-points are updated in parallel with the same initial potential, one may experience slowly converging or even unstable calculations for systems with a significant density-of-states at the Fermi level.

One can understand this behavior by considering the screening effects that take place during an update of the electronic wavefunction: The electrons will tend to screen out any non-selfconsistency in the potential. When a standard algorithm loops serially through **k**-points, the first **k**-point will screen the potential as well as possible, the second one will screen the remainder, and so on, leading to an iterative improvement in the selfconsistency. However, when all **k**-points are calculated from the same initial potential and in parallel, they will all try to screen the same parts of the potential, leading to an over-screening that gets worse with increasing numbers of processors, possibly giving rise to an instability.

Obviously, some kind of "damping" of the over-screening is needed in order to achieve a stable algorithm. We have selected the following approach: Each **k**-point contains a number of electronic bands (eigenstates), which are updated band by band using the conjugate gradients method. The screening of the potential takes place through the Coulomb and LDA exchange-correlation potentials derived from the total charge density. We construct the total charge density and the screening potentials after the update of each band (performed for all **k**-points in parallel): When all processors have updated band no. 1, the charge density and potentials are updated before proceeding to band no. 2, etc. This damping turns out to very effectively stabilize the selfconsistency process, even for difficult systems such as Ni with many states near the Fermi level.

The algorithmic difference may be summarized by the following pseudo-code. The standard serial algorithm is:

```
DO k-point = 1, No_of_points
  DO band = 1, No_of bands
    Update wavefunction(k-point,band)
    Calculate new charge density and screening potentials
  END DO
END DO
```

whereas our parallel algorithm is:

```
DO band = 1, No_of bands
  DO (in parallel) k-point = 1, No_of_points
    Update wavefunction(k-point,band)
  END DO
  Calculate new charge density and screening potentials
END DO
```

It is understood that an outer loop controls the iterations towards a self-consistent solution. The conjugate gradient algoritm actually requires the calculation of an intermediate "trial" step in the wavefunction update, so that the

work inside the outer loop is actually twice that indicated in the pseudo-code. In addition, the subspace rotations (not shown here) also require updates of the charge density.

3 Results on IBM SP2

The parallel algorithm described above is implemented in our code using the Parallel Virtual Machine (PVM) message-passing library, specifically the IBM PVMe implementation on the IBM SP2. At the time this work was carried out, PVMe was the most efficient message-passing library available from IBM, but we envisage other libraries to be substituted easily for PVM with time, or when porting to other parallel supercomputers such as the Fujitsu VPP-500.

In order to show the performance of the **k**-point-parallel algorithm, we choose a problem similar to a typical production problem of current interest to us. We perform fully selfconsistent calculations of a slab of a NiAl crystal with a (110) surface and a vacuum region, with 4 atoms in the unit cell. The plane wave energy cutoff was 50 Ry. We choose a Brillouin-zone integration with $N_k = 24$ **k**-points so that an even distribution of **k**-points over processors means that the job can be run on $N_{proc} = 1, 2, 3, 4, 6, 8, 12$ and 24 processors, respectively. At each **k**-point $N_{bands} = 18$ electronic bands were calculated, and the charge density array had a size of (16,24,96), or $N_{CD} = 0.295$ Mbytes. Only a single conjugate gradient step ($N_{CG} = 1$) is used. The starting point was chosen to be a selfconsistent NiAl system, where one of the Al atoms was subsequently moved so that a new selfconsistent solution must be found.

In our parallel algorithm, after each band has been updated by the slaves, the charge density array needs to be summed up from slaves to the master task, and subsequently broadcast to all slaves. Since the charge density array is typically 0.5 to 5 Mbytes to be communicated in a single message, our algorithm requires the parallel computer to have a high communication bandwidth and preferably support for global sum operations. Communications latency is unimportant, at the level provided on the IBM SP2.

Using the IBM PVMe optimized PVM version 3.2 library, a few minor differences between PVMe and PVM 3.3 are easily coded around. Unfortunately, the present PVMe release (1.3.1) lacks some crucial functions of PVM 3.3: The `pvm_psend` pack-and-send optimization, and the `pvm_reduce` group reduction operation, which could be implemented as a binary tree operation similar to the reduction operations available on the Connection Machines. Both would be very important for optimizing the accumulation of the charge density array from all slave tasks onto the master task; fortunately, they are included in a forthcoming release of PVMe.

The estimated number of bytes exchanged between the master and all of the slaves per iteration is $2N_{CD}(N_{proc} - 1)(2N_{bands}N_{CG} + 2)N_k/N_{proc}$, or $538 \times (N_{proc} - 1)/N_{proc}$ Mbytes total for the present problem, ignoring the communication of smaller data arrays. Since the IBM PVMe library doesn't implement reduction operations, the charge density has to be accumulated sequentially

from the slaves. IBM doesn't document whether the PVMe broadcast operation is done sequentially or using a binary tree, so we assume that the data is sent sequentially to each of the slaves. If we take the maximum communication bandwidth of an SP2 node with TB2 adapters to be $B = 35$ Mbytes/sec, we have an estimate of the communication time as $538 \times (N_{proc} - 1)/(N_{proc} B)$ seconds per iteration.

The runs were done at the UNI•C 40–node SP2, where for practical reasons we limited ourselves to up to 12 processors. The timings for a single selfconsistent iteration over all **k**-points is shown in Table 1, including the speedup relative to a single processor with no message-passing. Since the number of subspace rotations[7] varies depending on the wavefunction data, the calculation of **k**-points may take different amounts of time so that some processors will finish before others. The resulting load-imbalance is typically of the order of 10%. We show the average iteration timings in the table.

Number of processors	Iteration time (sec)	Speedup
1	361	1.00
2	221	1.64
3	157	2.30
4	119	3.04
6	87	4.13
8	74	4.85
12	56	6.48

Table 1. Time for a single selfconsistent iteration, and the speedup as a function of the number of processors.

The speedup data is displayed in Fig. 1 together with the "ideal" speedup assuming an infinitely fast communication subsystem. More realistically, we include the above estimate of the communication time in the parallel speedup for the two cases of $B = 20$ and $B = 35$ Mbytes/sec, respectively. We see that the general shape of the theoretical estimates agree well with the measured speedups, and that a value of the order of $B = 20$ Mbytes/sec for the IBM SP2 communication bandwidth seems to fit our data well. This agrees well with other measurements[12] of the IBM SP2's performance using PVMe.

From the above discussion it is evident that any algorithmic changes which would reduce the number of times that the charge density needs to be communicated, while maintaining a stable algorithm, would be most useful. We intend to pursue such a line of investigation. Better message passing performance could be achieved by optimizing carefully the operations that are used to communicate, mainly, the electronic charge density. Efficient implementations of the global summation as well as the broadcast of the charge density (for example, using a "butterfly"-like communication pattern) should be made available in the message passing library, or could be hand-coded into our application if unavailable in the library.

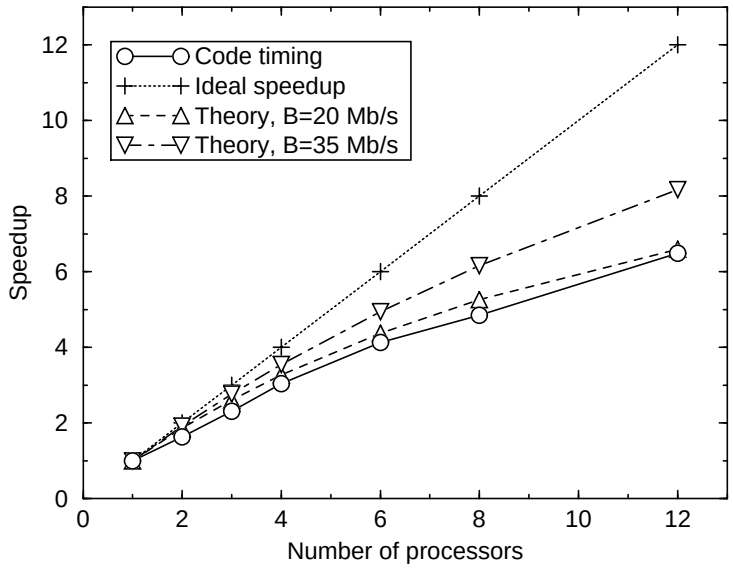

Figure 1. Speedup of the parallel code relative to the CPU time on a single processor (cf. Table 1). Besides the measured code timings, the linear "ideal" speedup is shown, along with the model estimate of the message passing time (discussed in the text) for the two bandwidths of 20 and 35 Mbytes/sec.

4 Conclusions

A Car-Parrinello ab-initio molecular dynamics method used hitherto on workstations and vector-computers has been parallelized using a master-slave model with message-passing. A fairly simple parallel algorithm based on farming a modest number of **k**-points out to slave processors has been used, complementary to other parallel Car-Parrinello algorithms. The memory savings are significant in our algorithm, since each processor only holds the wavefunction array for a single (or a few) **k**-points.

We find that **k**-point-parallel algorithms are non-trivial because of the changed convergence properties, owing to changes in the way the potential is screened. Updating the charge density after each band has been treated (in parallel) makes the algorithm stable.

The speedups measured for a test problem show satisfactory results for up to about 6 processors (depending on the problem at hand), which is more than adequate for our large-scale production jobs.

The timings obtained on an IBM SP2 show that the present parallel algorithm is bound by the communication bandwidth between the master processor and its slaves. Two options are identified to alleviate this bottleneck: 1) the investigation of less communication intensive algorithms, and 2) the efficient implementation

of global reduction and broadcast operations within the message passing library.

5 Acknowledgments

We are grateful to Richard M. Martin for discussions of **k**-point parallelization. CAMP is sponsored by the Danish National Research Foundation. The computer resources of the UNI•C IBM SP2 were provided through a grant from the Danish Natural Science Research Council.

References

1. The so-called Hellman-Feynman theorem of quantum mechanical forces was originally proven by P. Ehrenfest, Z. Phys. **45**, 455 (1927), and later discussed by Hellman (1937) and independently rediscovered by Feynman (1939).
2. W. Kohn and P. Vashishta, *General Density Functional Theory*, in *Theory of the Inhomogeneous Electron Gas*, eds. Lundqvist and March (Plenum, 1983).
3. G. B. Bachelet, D. R. Hamann and M. Schlüter, Phys. Rev. B **26**, 4199 (1982).
4. R. Car and M. Parrinello, Phys. Rev. Lett. **55**, 2471 (1985).
5. I. Stich, R. Car, M. Parrinello and S. Baroni, Phys. Rev. B **39**, 4997 (1989).
6. M. P. Teter, M. C. Payne, and D. C. Allan, Phys. Rev. B **40**, 12255 (1989).
7. M. J. Gillan, J. Phys.: Condens. Matter **1**, 689 (1989).
8. K. D. Brommer, M. Needels, B. E. Larson, and J. D. Joannopoulos, Comput. Phys. **7**, 350 (1992).
9. I. Stich, M. C. Payne, R. D. King-Smith, J. S. Lin and L. J. Clarke, Phys. Rev. Lett. **68**, 1359 (1992).
10. J. Wiggs and H. Jónsson, Comput. Phys. Commun. **87**, 319 (1995), and their refs. 13-17.
11. T. Yamasaki, in these proceedings.
12. Oak Ridge performance measurements available on World Wide Web at the `<URL:http://www.epm.ornl.gov/pvm/perf-graph.html>`.

This article was processed using the LaTeX macro package with LLNCS style