Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

Journal of Computational Physics 188 (2003) 1–15

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

Massively parallel linear-scaling algorithm in an ab initio local-orbital total-energy method

Spencer D. Shellman [a], James P. Lewis [b,*], Kurt R. Glaesemann [c], Krzysztof Sikorski [a], Gregory A. Voth [b]

[a] *School of Computing, University of Utah, Salt Lake City, UT 84112-0850, USA*

[b] *Henry Eyring Center for Theoretical Chemistry, and Department of Chemistry, University of Utah, Salt Lake City, UT 84112-0850, USA*

[c] *Lawrence Livermore National Laboratory, Livermore, CA 94550, USA*

Received 19 October 2002; received in revised form 13 November 2002; accepted 31 January 2003

Abstract

Similar to the manner of S. Itoh et al. [Comp. Phys. Commun. 88 (1995) 173], we report implementation of a massively parallel linear-scaling algorithm into an ab initio tight-binding method called Fireball [Phys. Rev. B (2001)]. The use of local-orbitals yields a very sparse Hamiltonian matrix which facilitates using a linear-scaling algorithm to obtain the electronic band-structure energy. The general functional form of Kim et al. [Phys. Rev. B 52 (1995) 1640], which minimizes a functional to obtain the electronic band-structure energy, has been parallelized utilizing the conjugate gradient method. The results of this approach are reported here. In addition, the use of "fireball" wavefunctions, where the wavefunctions are explicitly zero beyond some cutoff, allows for pre-generating all integrals describing two- and three-center interactions. The computation of these integrals is then an easily parallelizable problem for which the results are reported. Both integral generation and the linear-scaling optimization procedures are parallelized using the standard MPI message passing interface mixed with an OpenMP strategy.

1. Introduction

As each new generation of computers is introduced, the scope of computational endeavors becomes more promising. Larger and longer simulations can be run, producing better quantitative results, answering a variety of scientific questions. A major factor in producing more efficient and feasible computational tools is the utilization of parallel computers. The most recently produced supercomputers boast of terascale computing through the development of parallel algorithms. The advances of computational hardware has

[*] Corresponding author. Present address: Department of Physics and Astronomy, Brigham Young University, N233 ESC, P.O. Box 24658 Provo, UT 84206-4658, USA.

E-mail address: james_lewis@byu.edu (J.P. Lewis).

doi:10.1016/S0021-9991(03)00069-X

also significantly improved the computational software produced in the electronic-structure community. Several methods even incorporate some degree of parallelization to ideally reduce the overall computation time by the number of processors utilized (for example, see [1–6]). However, with a few exceptions, most have not demonstrated efficient parallelization on hundreds or thousands of processors. The need to apply theoretical methods to increasingly computationally demanding chemical systems is the driving force behind the parallelization efforts reported here.

Many properties of a molecular system depend upon the environment in which they occur. To fully understand many chemical reactions of molecular systems it is necessary to study condensed-phase reactions rather than gas-phase reactions. For example, performing a simulation of a single octahydro-1,3,5,7-tetranitro-1,3,5,7-tetrazocine (HMX) molecule, a highly energetic (explosive) material shown in Fig. 1, decomposing in a condensed-phase environment would require that the image of the decomposing molecules do not strongly interact with each other. Therefore, to properly treat the environment in condensed-phase molecular systems, thousands of atoms would be needed to increase the size of the periodic supercell. For example, a $4 \times 4 \times 4$ supercell of HMX would contain 1792 atoms (128 molecules). In addition, a sufficiently accurate method must be used to obtain at least the correct qualitative features of the reaction, if not the quantitative accuracy.

Empirical potentials will fail to produce the correct chemistry without bias because bond-breaking mechanisms require a many-body effect due to the quantum nature of the chemical bond. Semi-empirical, or tight-binding (TB), methods provide a quantum mechanical picture; however, the primary computational "bottleneck" of such electronic-structure methods is the adverse power law scaling, $O(N^l)$, where N is the total number of basis set orbitals and l is generally 3 or greater. In TB methods or density functional theory (DFT) methods, where a linear combination of atomic orbitals (LCAO) are used, the power law scaling inherently is $l = 3$ because the band-structure energy is evaluated solely from a generalized eigenvalue equation. Several approaches for reducing the adverse power law scaling in these methods (for non-metals) from $l = 3$ to $l = 1$ (linear-scaling methods) have been the topic of much discussion in the literature (see [7,8] for reviews). The primary criterion for using using these linear-scaling methods is usually satisfied within a tight-binding approach, namely, the Hamiltonian and overlap matrices are quite sparse. Little effort has been made to parallelize these linear-scaling algorithms.

Similar to the typical TB approach, we have introduced an ab initio TB approach, called Fireball, which is an improvement to the original Sankey–Niklewski method [9]. The Fireball method avoids the stringent approximations present in semi-empirical methods, because the approach relies upon no experimental parameterization. The use of local-orbitals in the Fireball method yields a very sparse Hamiltonian matrix, which facilitates using a linear-scaling algorithm to obtain the electronic band-structure energy. The general functional form of Kim et al. [10], which optimizes a defined functional to obtain the

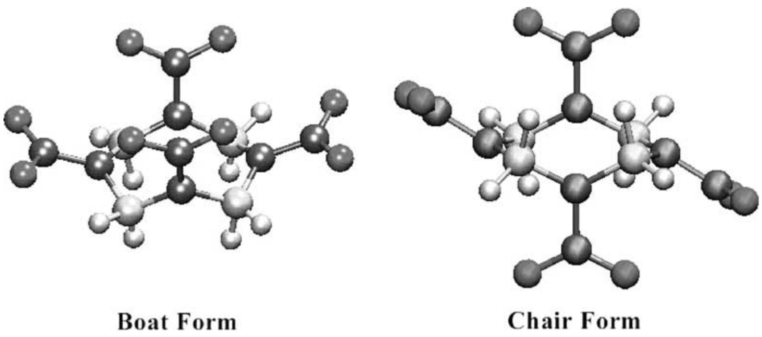


Fig. 1. The boat and chair gas-phase conformers of HMX.

electronic band-structure energy, is our functional of choice. The optimization of this functional has been parallelized similar to the method of Itoh et al. [2] developed for TB methods; however, we have made significant improvements. In particular, we have developed a method for determining the ideal stepsize in the conjugate gradient optimization algorithm and we use the standard MPI and OpenMP libraries to perform the parallel tasks. We report here the implementation of a massively parallel linear-scaling algorithm into our ab initio TB FIREBALL method [11]. Generalization to similar TB or other semi-empirical methods can be achieved with little effort.

This paper is organized as follows. Section 2 describes the theoretical background of the FIREBALL method. Section 3 discusses the results for parallelization of the integral routines evaluated in the CREATE module. As an initial benchmark comparison, implementation of parallel diagonalization libraries (ScaLAPACK) into FIREBALL was considered; our results are discussed in Section 4. From the results of the ScaLAPACK implementation, it is evident that a more efficient parallel strategy is needed. This strategy is discussed in Section 5. The results for the massively parallel linear-scaling with system size implementation are discussed in Section 6. More specific details about the algorithm for our parallel linear-scaling strategy are given in Appendix A.

2. Theoretical background

A summary of the main features of the FIREBALL method is discussed here; for a more detailed discussion, we refer the reader to [11] and references therein.

The theoretical basis of the FIREBALL method is the use of DFT with a non-local pseudopotential scheme. At the core of the method is the replacement of the Kohn–Sham energy functional by the approximate Harris–Foulkes functional [12,13]

$$E_{\mathrm{tot}}^{\mathrm{Harris}} = E^{\mathrm{BS}} + \left\{U^{\text{ion–ion}} - U^{ee}[\rho_{in}(\mathbf{r})]\right\} + \{U^{xc}[\rho_{in}(\mathbf{r})] - V^{xc}[\rho_{in}(\mathbf{r})]\}. \tag{1}$$

In Eq. (1), E^{BS} is the band structure energy ($2\sum_{i\epsilon occ}\epsilon_i$), where ϵ_i are the eigenvalues of the one-electron Schrödinger equation; the second term is the "short-range" repulsive interaction which is the ion–ion interaction offset by the overcounting of the Hartree interactions; and the last term is a correction to the exchange-correlation.

In evaluating the total energy of the system (Eq. (1)), the input density is a sum of confined spherical atomic-like densities, $\rho_{in}(\mathbf{r}) = \sum_i n_i|\phi_i(\mathbf{r}-\mathbf{R}_i)|^2$. The orbitals $\phi_i(\mathbf{r}-\mathbf{R}_i)$ are the slightly excited "fireball" pseudoatomic wavefunctions which are used as basis functions for solving the one-electron Schrödinger equation. The occupation numbers n_i determine the number of electrons occupying each spherically confined atomic-like densities. Although the Harris–Foulkes functional is inherently a non-self-consistent formalism, a self-consistent generalization has been implemented. This implementation is necessary to deal with systems in which charge transfer contributes significantly to the underlying chemistry and requires a self-consistent determination of the occupation numbers, n_i, i.e., now $n_i = n_i^0 + \delta n_i$. Thus, the total energy is a *function* of the occupation numbers, $E_{\mathrm{tot}}[\rho_{in}(\mathbf{r})] \equiv E_{\mathrm{tot}}[n_i]$ $(n_i \neq n_i^0)$, and a self-consistent procedure on the occupation numbers, n_i, is introduced (DOGS) [14,15]. With the given form of the total energy, the forces acting on an atom at position R_l are determined by taking the derivative of the total energy with respect to R_l. The band-structure force is evaluated using a variation of the Hellmann–Feynman theorem [9].

In our calculations, the generalized norm-conserving separable pseudopotentials of the Hamann [16] or Troullier–Martins type [17] are used, employing for their construction the scheme of Fuchs and Scheffler [18]. The pseudopotentials are transformed into the fully separable form suggested by Kleinman and Bylander [19]. For the exchange-correlation energy within the pseudopotential calculation, and respectively for the exchange-correlation potential, various parameterizations of the local-density approximation (LDA) and of the generalized gradient approximation (GGA) are available [20–27].

For calculating the exchange-correlation (XC) interactions, we use the approach proposed by Horsfield [28], which represents these interactions as a many-center expansion based on an expansion of the density a site at a time. The Horsfield approximation can be used with gradient corrected functionals. The XC potential matrix elements are calculated using up to a three-center approximation, splitting the matrix elements into one, two, and three center integrals, similar to the electrostatic integrals. However, the on-site terms and the double counting correction term are calculated using only a two-center approximation. This approach for determining the exchange-correlation interactions is independent of the type of functional used. Therefore, a variety of LDA and GGA exchange-correlation interactions can be implemented within the method. Currently, however, only two types of exchange-correlation density functionals are available within Fireball–LDA and Becke exchange (B88) [23] with Lee–Yang–Parr (LYP) correlation [22].

In solving the one-electron Schrödinger equation a set of slightly excited pseudoatomic "fireball" wavefunctions are used. These orbitals are computed within DFT and a norm-conserving separable pseudopotential [16] and are chosen such that they vanish at some radius r_c ($\psi^{\text{atomic}}_{\text{fireball}}|_{r \geqslant r_c} = 0$). This boundary condition is equivalent to an "atom in the box" and has the effect of raising the electronic energy levels ($\epsilon_s, \epsilon_p, \epsilon_d, \ldots$ atomic eigenvalues) due to confinement. The radial cutoffs (r_c) are chosen such that these electronic eigenvalues remain negative and are mildly perturbed from the free atom. A flexible choice of basis set of double numerical (DN) or additional polarization orbitals are permitted within the Fireball method.

The "fireball" boundary condition yields promising features. First, the slight excitation of the atoms somewhat accounts for Fermi compression in solids which apparently gives a better representation of solid-state charge densities [29]. Second, the range of hopping matrix elements between orbitals on different atoms is limited; therefore, very sparse matrices are created for large systems. This inherent sparseness allows one to more readily implement linear-scaling algorithms to obtain the band structure energy, which will be the main discussion of this paper and Section 5. Also as a result of the finite interaction range, the two- and three-center interactions can be evaluated for all possible geometries, which will be the discussion of the next section.

3. Parallelization of the integral package – Create

Given any two atomic orbitals, i and j, beyond some cutoff interaction distance $(r_{ci} + r_{cj})$ the matrix elements H_{ij} and S_{ij} become exactly zero. As an example, the interaction range of the overlap matrix of two s-orbitals is demonstrated in Fig. 2. As a result of the cutoff, there is only a prescribed interaction range over which the integrals need be evaluated.

Within the Fireball approach, all integrals may be pre-computed on a numerical grid and the specific values needed gleaned from the tabulated values via interpolation. With this approach, interactions cal-

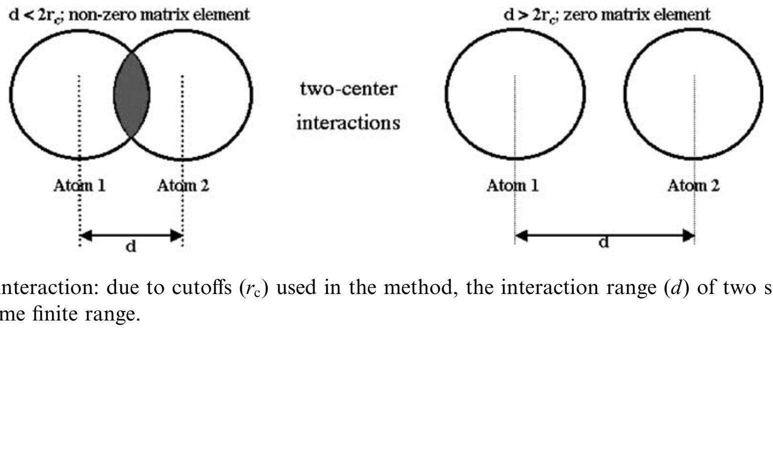

d < 2r$_c$: non-zero matrix element
d > 2r$_c$: zero matrix element
two-center
interactions
Atom 1
Atom 2
d
Atom 1
Atom 2
d

Fig. 2. The two-center interaction: due to cutoffs (r_c) used in the method, the interaction range (d) of two s-orbitals, shown here, is non-zero only within some finite range.

culated within the Create module will be readily available. For two-center interactions, a cubic spline function is chosen as the interpolation method, so as to maximize numerical stability. Because these integral tables depend only on the atom type, their r_c values, and the type of DFT exchange-correlation functional, the integral tables only need to be generated once, for a given set of atomic species. As long as the criteria chosen for generating the data files does not change, the data files may be used over and over again, saving overall computation time. The generation process for the atomic interactions lends itself to easy parallelization by spreading the data file creation over multiple processors based on differing integral types.

Within the Create module, integrals are distributed to different processors based upon atomic number and based upon integral type (kinetic, overlap, exchange-correlation, etc.). The simple loop based approach is used for distributing work to processors and has been found to be very effective. This approach minimizes communication while distributing work evenly across the processors. This parallelization scheme is particularly important, since the number of data files needed grows as $O(M^3)$ with the number of different atomic species M.

The other two traditional approaches to integral evaluation were rejected for efficiency reasons. Within the "conventional" approach employed in gas-phase electronic-structure codes [30], integrals are evaluated once per geometry and stored to disk for use during the SCF procedure. This approach would result in extensive disk I/O and computation, given the shear number of MD time steps generally needed. The "direct" approach [31] in which integrals are calculated as needed is similar to the approach taken within Fireball. Given the large computational cost of calculating each individual integral (because the integrals are evaluated numerically), the evaluation of different matrix element components are performed via interpolation within Fireball, rather than via explicit evaluation of the integral as done in the gas-phase direct approach.

The scalability of the Create module is demonstrated for both 4 and 5 atomic species in Fig. 3. On an SGI Origin 2000 system (Los Alamos National Laboratory), increased performance is achieved up to and including 1024 processors. The Origin 2000 computer is designed with efficient parallel computation as the goal, containing 128 processors per shared memory box. The version of MPI is also vendor optimized to use the hardware interconnect present on the Origin 2000. This approach to parallelism for the Create module works very well on both shared memory machines and on distributed memory machines. Because

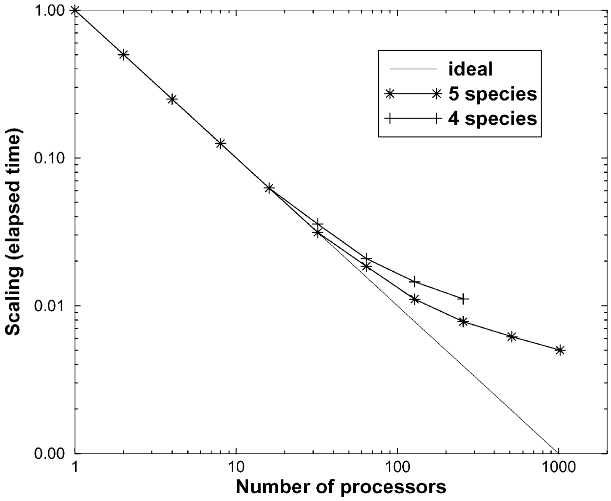


Fig. 3. Scalability of the Create module. The larger calculation (five species) produces better scaling, as expected.

the code scales better with more species, the code scales as well as is needed. The efficiency advantage of this parallelization is evident in that calculations previously taking several weeks now take only a few hours.

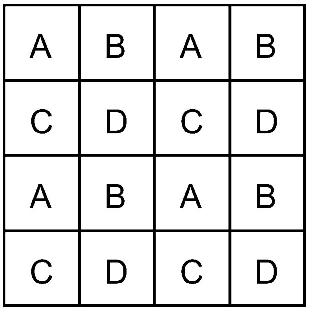
4. Fireball parallelization – ScaLAPACK implementation

Within most DFT and TB methods, as well as the Fireball method, the most time consuming step are the $O(N^3)$ matrix manipulations, such as matrix multiplication and diagonalization, where N is the number of orbitals (i.e., matrix diagonalization and matrix multiplication). The adverse scaling of these matrix manipulations produces a "bottleneck" which prohibits performing calculations of systems with more than a few hundred atoms. One approach for reducing this "bottleneck" is to implement a parallel set of routines for performing matrix diagonalization and matrix multiplication found in the BLACS (Basic Linear Algebra Communication Subprograms), PBLAS (Parallel Basic Linear Algebra Subprograms), and ScaLAPACK (Scalable Linear Algebra PACKage) packages. The BLACS package provides a standard basis for parallel linear algebra. The PBLAS package is a parallel version of the standard BLAS library, and performs matrix arithmetic on matrices distributed using BLACS. A variety of data types (both real and complex) and operations are supported (including dot products, addition and multiplication). The ScaLAPACK package is a parallel version of the standard LAPACK library, which solves a variety of linear algebra problems, including the generalized eigenvalue problem.

The BLACS library manages the distribution of matrices across multiple processors using a block cyclic decomposition. A block cyclic approach ensures load balancing and thus scalability. An example of the global view of the matrix distributed to four processors A, B, C, D is presented in Fig. 4. In this example, each processor has one-fourth of the matrix stored locally.

All computational work within fireball that is inherently $O(N)$ is done on processor zero only. Processors 1 through $p-1$ go to sleep and do computation only when contacted by processor zero. This is similar to the fork and rejoin approach used in multi-threaded programming, but in this case processes 1 through $p-1$ wait rather than die and then reincarnate as needed. Initially, processor zero uses MPI to distribute the matrices to the appropriate processors in a BLACS block cyclic distribution. The PBLAS routines are called whenever matrices must be multiplied and ScaLAPACK routines are called to diagonalize matrices as needed. Finally, MPI is used to accumulate the final answers to processor zero. The Fireball code is written in Fortran90 to take advantage of reduced memory requirements provided by parallelism.

Once the matrix blocks get too small, the performance degrades as is shown in Fig. 5. As the system becomes larger (i.e., more orbitals implies larger N), the scalability improves because the $O(N^3)$ step dominates the calculation, thus ensuring good load balancing. As N becomes larger, the size and number of



A	B	A	B
C	D	C	D
A	B	A	B
C	D	C	D

Fig. 4. Global view of a block cyclic distributed matrix.

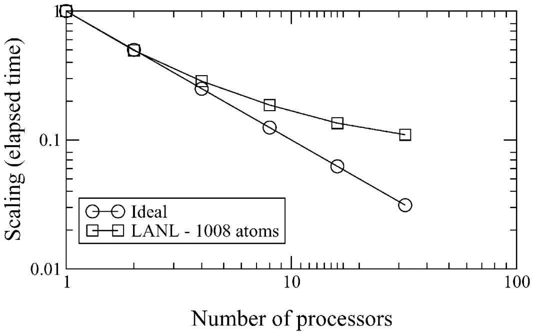


Fig. 5. Scalability of the fireball module with the ScaLAPACK (parallel diagonalization) implementation. These calculations were performed on the SGI-Nirvana machine located at Los Alamos National Laboratory (LANL).

blocks distributed becomes larger, and the thus the PBLAS and ScaLAPACK algorithms become more efficient. Better performance is obtained if the interconnect between processors is fast, because of the high levels of inter-processor communication. This method provides a precursor to other approaches, such as linear-scaling methods by providing a standard with which to compare. In general, parallel diagonalization exhibits very poor scaling due to the heavy communication overhead (as seen by Fig. 5).

5. Fireball parallelization – linear-scaling implementation

As a result of the conclusion from parallel diagonalization (Section 4), to achieve more efficient scaling a better algorithm for obtaining the band-structure energy must be used. Similar to the report of Itoh et al. [2], we choose to implement a linear-scaling algorithm into the ab initio TB Fireball method [11]. Our goal is to develop a parallel implemention using OpenMP/MPI languages for minimizing the general functional form of Kim et al. [10], which when optimized determines the electronic band-structure energy. We have developed and tested an efficient parallel optimization algorithm utilizing the conjugate gradient method. The algorithm exhibits a near linear speedup in the problem size n and number of processors p, as implemented on up to 1024 processors. We illustrate the key principles of the algorithm here, and describe it in detail in Appendix A. Since we introduce variations from the original work of Itoh et al. [2], we include full details here.

5.1. Energy functional and gradient matrix

We define h and s to be the symmetric $n \times n$ Hamiltonian and overlap matrices, respectively, and m, $1 \leqslant m \leqslant n$, to be the number of occupied bands (typically, $m \geqslant n/2$). The vectors C_i, $i = 1..m$, are the length-n wavefunction coefficient vectors, and C is the $n \times m$ matrix of which C_i is the ith column. We define the matrices $F \equiv hC$, $F^s \equiv sC$, $H \equiv C^{\mathrm{T}}F = C^{\mathrm{T}}hC$, and $S \equiv C^{\mathrm{T}}F^s = C^{\mathrm{T}}sC$; clearly H and S are symmetric. The energy functional is given in [10] as

$$\tilde{E} \equiv 2\left[2\mathrm{Tr}(H - \eta S) + \sum_{i,j=1}^{m}(\eta S_{ij}^2 - H_{ij}S_{ij})\right] + \eta N, \tag{2}$$

where N is the number of electrons in the system, η is a parameter related to the chemical potential, and $\mathrm{Tr}(A) \equiv \sum_{i=1}^{n} A_{ii}$ is the trace of an $n \times n$ matrix A.

We define the $n \times m$ gradient matrix

$$G \equiv \frac{\partial \tilde{E}}{\partial C} \equiv \left[\frac{\partial \tilde{E}}{\partial C_{ij}} \right]. \tag{3}$$

To find an expression for G we apply the chain rule to the matrix factors of $\tilde{E}$. Defining the $n \times m$ matrix $\Delta_{kl} \equiv [\delta_{ik}\delta_{jl}]$, where δ_{ij} is the Kronecker delta function, we observe that

$$\frac{\partial H}{\partial C_{kl}} = \Delta_{kl}^{\mathrm{T}} hC + C^{\mathrm{T}} h\Delta_{kl} = \Delta_{kl}^{\mathrm{T}} F + \left(\Delta_{kl}^{\mathrm{T}} F\right)^{\mathrm{T}}, \tag{4}$$

$$\frac{\partial S}{\partial C_{kl}} = \Delta_{kl}^{\mathrm{T}} sC + C^{\mathrm{T}} s\Delta_{kl} = \Delta_{kl}^{\mathrm{T}} F^s + \left(\Delta_{kl}^{\mathrm{T}} F^s\right)^{\mathrm{T}} \tag{5}$$

and hence

$$\begin{aligned}\frac{\partial \tilde{E}}{\partial C_{kl}} &= 2\left\{ 2\mathrm{Tr}\left(\frac{\partial H}{\partial C_{kl}} - \eta \frac{\partial S}{\partial C_{kl}} \right) - \sum_{i,j=1} \left[\left(H_{ij} - 2\eta S_{ij}\right) \frac{\partial S_{ij}}{\partial C_{kl}} + \frac{\partial H_{ij}}{\partial C_{kl}} S_{ij} \right] \right\} \\ &= 8\left(F_{kl} - \eta F_{kl}^s\right) - 4\sum_{i=1}^{m} F_{ki}^s (H_{li} - 2\eta S_{li}) - 4\sum_{i=1}^{m} F_{ki} S_{li}.\end{aligned} \tag{6}$$

It follows that

$$G = 8F - 8\eta F^s + 8\eta F^s S - 4F^s H - 4FS. \tag{7}$$

As described in [10], the closeness of η to the electronic chemical potential determines whether the ground-state energy E_0 is a minimum of $\tilde{E}$. For the purposes of simplification, we currently work with a fixed $\eta = 0$, with $m = n/2$ (the occupied subset – i.e., the functional of Ordejón et al.) [32]. This avoids the need for computing η at each iteration of the algorithm, but requires careful selection of the initial coefficients and slows down convergence in many cases (see Section 7). The algorithm developed here can be readily applied to the general case ($\eta \neq 0$) with only slight variation. With $\eta = 0$, Eqs. (2) and (7) become

$$\tilde{E} \equiv 2\left(2\ \mathrm{Tr}(H) - \sum_{i,j=1}^{m} (H_{ij} S_{ij}) \right), \tag{8}$$

$$G = 8F - 4F^s H - 4FS. \tag{9}$$

5.2. Minimization method

The iterative method described in Appendix A approximates the coefficient matrix $\tilde{C}$ at which $\tilde{E}$ achieves its ground state, E_0. At each iteration the gradient matrix G is computed and used by the conjugate gradient method to determine a search direction along which to vary the coefficient matrix C so that C converges to $\tilde{C}$. The approximation of the optimal step length $\tilde{\gamma}$ which locally minimizes $\tilde{E}$ along the search direction requires only the minimization of a fourth-degree polynomial. We let i denote the number of the iteration with $i = 1$ at the first iteration, let C^i denote the value assigned to C at the beginning of iteration i (with C^1 equal to the initial guess), and let $\tilde{E}_i$ denote the energy value computed at the end of iteration i. The algorithm terminates when the relative change in $\tilde{E}$ falls within a given threshold $\delta > 0$, that is,

$$|\tilde{E}_i - \tilde{E}_{i-1}| \leqslant \delta |\tilde{E}_i| \tag{10}$$

at step $i > 1$, and when the RMS of G falls within an absolute threshold $\zeta > 0$.

5.3. Communication method

We describe here the communication method we use to perform distributed matrix multiplications. The method attempts to ensure that the communications overhead has order $\mathrm{O}(p)$, where $p > 0$ is the number of processors, by ensuring that a processor only sends its matrix section to those other processors that require it to perform the multiplication.

We define $m' \equiv m/p$ (resp. $n' \equiv n/p$) to be the number of coefficient vectors (resp. rows of h and s) distributed to each processor. We assume for simplicity that p exactly divides m and n. Thus processor k, $k = 0, \ldots, p-1$, locally stores rows $kn' + 1 \ldots (k+1)n'$ of C, h, and s.

We introduce a notation to describe the distribution of matrices over p processors. For a matrix X with $m = m'p$ rows (resp. columns), we let $X_{[km]}$ (resp. $X^{[km]}$) denote the submatrix made up of rows (resp. columns) $km' + 1 \ldots (k+1)m'$ of X. For a matrix X with $n = n'p$ rows (resp. columns), we let $X_{[kn]}$ (resp. $X^{[kn]}$) denote the submatrix made up of rows (resp. columns) $kn' + 1 \cdots (k+1)n'$ of X. We let $X_{[km]}^{[ln]}$ represent $(X^{[ln]})_{[km]}$ or, equivalently, $(X_{[km]})^{[ln]}$. Following this notation, each processor k locally stores $C_{[kn]}$, $h_{[kn]}$, and $s_{[kn]}$.

Suppose we want to multiply a $l \times m$ matrix A by a $m \times n$ matrix B, where p is the number of processors, $l = l'p$, $m = m'p$, and $n = n'p$. Suppose also that for $k = 0, \ldots, p-1$, processor k locally stores $A_{[kl]}$ and $B_{[kl]}$. Processor k can compute $(A \times B)_{[kl]}$ using the equality

$$(A \times B)_{[kl]} = \sum_{i=0}^{p-1} A_{[kl]}^{[im]} B_{[im]}. \tag{11}$$

Let us now define a $p \times p$ block information matrix A' associated with A, such that $A'_{i+1,j+1} = 0$ if $A_{[il]}^{[jm]} = 0$ and $A'_{i+1,j+1} = 1$ if $A_{[il]}^{[jm]} \neq 0$. Row k of A' contains a 1 in position $i+1$ if processor k needs to receive $B_{[im]}$ from processor i to complete the computation of (11). It naturally follows that column k of A' contains a 1 in position $i+1$ if processor k needs to send $B_{[km]}$ to processor i. We always assume that the diagonal of a block information matrix is zero.

A block information matrix associated with a distributed matrix may be easily computed by examining matrix elements on each processor to fill in the rows, then gathering the rows across all processors. The block structure of h, s and the initial C ensure that blocks which are zero in the initial values of H, S, and C remain zero throughout the algorithm, so that the associated block information matrices need be computed only once.

Given distributed matrices A and B and a block information matrix A' stored on every processor, we use the following method to compute and distribute $A \times B$. Here we call the sequence of processors $k+1, \ldots, p-1, 0, \ldots, k-1$ the *successor chain* of processor k.

1. Initialize the Boolean array $recv[0 \cdots p-1]$ to false. This array indicates which processors will be targets of sends.
2. Each processor k executes the following loop for $i = 0, \ldots, p-1$. Processor i examines row $i+1$ of A' to find the first processor in i's successor chain that needs to receive the local section from processor i, and whose entry in *recv* is false. If such a processor exists we call it j, otherwise go to step$(A'_{i+1,j+1})$ to zero, set $recv[j]$ true, send the local section of k to j if $i = k$, and receive the local section of i if $j = k$.
3. If $A' = 0$ then terminate, otherwise go to step 2.

5.4. Use of parallel tools

In the parallel implementation presented here, the MPI and OpenMP libraries are used to manage communication between processes. The eight processors on each node are divided into two four-processor shared memory groups. The groups of processors communicate with each other using MPI library calls as if each group were a single processor. Processors within the same group behave according to the OpenMP standard; one processor executes the application code and communicates with other groups, while the other processors in the group remain idle except when a shared loop is encountered. A shared loop is identified by OpenMP directives placed within the code. The iterations of a shared loop are distributed among the processors in the group, while any variables that must be available to the entire group are flagged in the code and held in shared memory. In this manner we achieve limited parallelism without the need for MPI communications between processors in a group. We declare the application's most compute-intense loops to be shared; these include matrix multiplication and addition, and computation of the energy functional.

An additional decrease in the communication overhead is obtained by using the MPI function called MPI_GraphCreate. This function takes a graph whose vertices indicate paths between pairs of processors along which data must be sent, and creates a communicator in which the processors are ordered in such a manner that the graph is almost optimally matched to the underlying multiprocessor architecture. This improves the chances that processors that need to communicate with each other will be physically "close" to one another. Our current method is to perform a Boolean OR of all the block information matrices associated with the matrices in the algorithm, then create a graph containing a vertex for every pair of processors whose entry in the resulting block matrix is nonzero. This graph is passed to MPI_GraphCreate, and then each processor sends its local parameters and matrix segments to the processor that will have its rank under the new communicator. In essence, each processor assumes the identity of another processor. Using the new communicator in place of the original communicator makes this substitution transparent. This process needs to be done only once, after the computation of the block information matrices.

6. Results

The scaling results from running the optimization algorithm for systems of 1512, 3024, and 6048 atoms (this corresponds to 54, 108, and 216 HMX molecules, respectively) on Blue Horizon (IBM SP architecture at San Diego Supercomputer Center), with up to 1024 processors, are presented in Figs. 6 and 7. As seen from Fig. 6, the linear scaling in the number of processors becomes stronger and more consistent as the number of atoms increases. For 1512 atoms, the ratio of the time per iteration for $2n$ processors to the time per iteration for n processors ranges between 58 and 71% (with 140% at $2n = 1024$), with an average of 75% (50% indicates ideal linear scaling). For 6048 atoms, the same ratio ranges between 59 and 68% (with 69% at $2n = 1024$), with an average of 64%. The improved scaling performance is due to the fact that the proportion of computation time (matrix multiplication, etc.) to communication time increases with the number of atoms. Additionally, as seen in Fig. 7, the linear scaling in the number of atoms becomes stronger and more consistent as the number of processors increases. For eight processors, the ratio of the time per iteration for $2n$ atoms to the time per iteration for n atoms ranges from 194 to 255%. For 512 processors, the same ratio ranges from 180 to 233%. This reflects slight superlinearity of scaling as the dimensions of the matrix segments distributed to the processors increase.

We performed preliminary tests to measure the rate of convergence to the minimum energy value using the 1512 atom system. By changing the relative energy tolerance, δ, gradient RMS tolerance, ζ, and cutoff

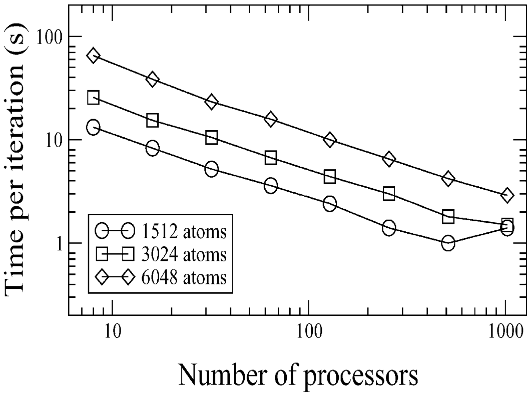


Fig. 6. Scaling results – time to perform one iteration of the optimization algorithm as the number of processors are increased with fixed number of atoms (on NPACI Bluehorizon IBM-SP3 architecture).

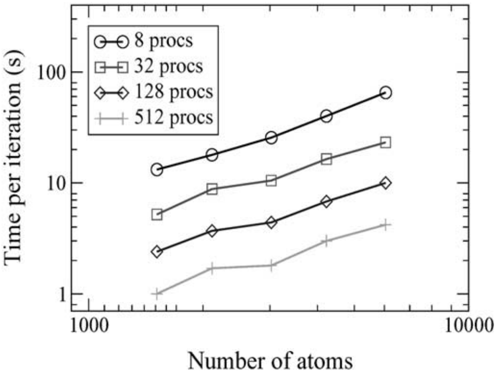


Fig. 7. Scaling results – time to perform one iteration of the optimization algorithm as the number of atoms are increased with fixed number of processors (on NPACI Bluehorizon IBM-SP3 architecture).

radius, r, no obvious series of patterns regarding the number of iterations until convergence was determined. The lack of a certain pattern in our preliminary results may be largely due to the variable randomness in initializing the wavefunction coefficients that occurs from one job to the next (different sets of processors from one job submission to the other). However, one exception was that as the relative energy tolerance, δ, was decreased from $\delta = 10^{-5}$ to $\delta = 10^{-6}$ the number of iterations required increased additionally by about an order of magnitude.

7. Summary

Similar to the manner of Itoh et al. [2], we report implementation of a massively parallel linear-scaling algorithm into an ab initio tight-binding method called Fireball [11]. Our approach uses the standard MPI message passing interface mixed with an OpenMP strategy. Our results demonstrate that we have achieved linear-scaling parallelization both with the number of processors (up to 1024) and with system size (up to 6048 atoms).

The use of local-orbitals in the Fireball method yields a very sparse Hamiltonian matrix, which facilitates using a linear-scaling algorithm to obtain the electronic band-structure energy. Evaluation of forces is also written in a massively parallel procedure so that linear-scaling is achieved for molecular-dynamics simulations. Implementation of our parallel algorithms can be generalized to other TB or other semi-empirical methods with little effort.

Plans are underway to implement an improved energy functional having a positive definite Hessian matrix. Described in Section 5, this functional requires that η be computed at each step to approximate the electronic chemical potential. Although with η fixed at 0 the careful selection of initial coefficient vectors seems to prevent the functional value from spiraling ever downward, the nonexistence of a global minimizer of P in step 6 of the algorithm may be responsible for the slow convergence. Additionally, plans are underway to explore methods of computing search directions that may have better performance than conjugate gradient, and methods of computing search lengths that may overcome the sensitivity of the current root-finding method without requiring a full line search.

Acknowledgements

The authors thank P. Ordejón and S. Itoh for useful discussions in the initial stages of this project. This research is funded by The University of Utah Center for the Simulation of Accidental Fires and Explosions (C-SAFE), funded by the Department of Energy, Lawrence Livermore National Laboratory, under subcontract B341493. Within this DOE funding, allocations of computer time on the SGI Origin 2000 (Nirvana) located at Los Alamos National Laboratory were used in this work; an allocation of computer time from the NPACI–SDSC (National Partnership for Advanced Computational Infrastructure–San Diego Supercomputer Center) was also used in this work.

Appendix A. Description of the algorithm

The steps involved in an iteration of the algorithm are described in detail below, and graphically in the associated figures. The total communication cost of an iteration consists of five order $\leqslant \mathrm{O}(p)$ operations (see Section 5.3) and four $\mathrm{O}(\log p)$global gathers.

A.1. Steps of the algorithm

1. Initialization. Each processor k, $k = 0, \ldots, p-1$, initializes and locally stores $C_{[kn]}$, $h_{[kn]}$, and $s_{[kn]}$.
2. Computation of $(C^{\mathrm{T}})_{[km]}$, $F_{[kn]}$, and $(F^s)_{[kn]}$.

This step is based on the equalities

$$F_{[kn]} = h_{[kn]}C = \sum_{i=0}^{p-1} h_{[kn]}^{[in]} C_{[in]}, \quad (F^s)_{[kn]} = s_{[kn]}C = \sum_{i=0}^{p-1} s_{[kn]}^{[in]} C_{[in]}. \tag{A.1}$$

Set $C = C^i$, where i is the iteration number. Each processor k copies $(C_{[kn]}^{[km]})^{\mathrm{T}}$ into $(C^{\mathrm{T}})_{[km]}^{[kn]}$, multiplies $h_{[kn]}^{[kn]}$ and $s_{[kn]}^{[kn]}$ by $C_{[kn]}$, and copies these products into $F_{[kn]}$ and $(F^s)_{[kn]}$, respectively. It then sends $C_{[kn]}$ to all other processors that require it, an order $\leqslant \mathrm{O}(p)$ global operation described in Section 5.3. As each processor k receives $C_{[ln]}$ from processor l, $l \neq k$, it copies $(C_{[ln]}^{[km]})^{\mathrm{T}}$ into $(C^{\mathrm{T}})_{[km]}^{[ln]}$, multiplies $h_{[kn]}^{[ln]}$ and $s_{[kn]}^{[ln]}$ by $C_{[ln]}$, and adds these products to $F_{[kn]}$ and $(F^s)_{[kn]}$, respectively. At the end of this step each processor k has computed and locally stored $(C^{\mathrm{T}})_{[km]}$, $F_{[kn]}$, and $(F^s)_{[kn]}$. See Fig. 8 for an illustration of the computation of $F_{[kn]}$(the computation of $(F^s)_{[kn]}$ is similar).

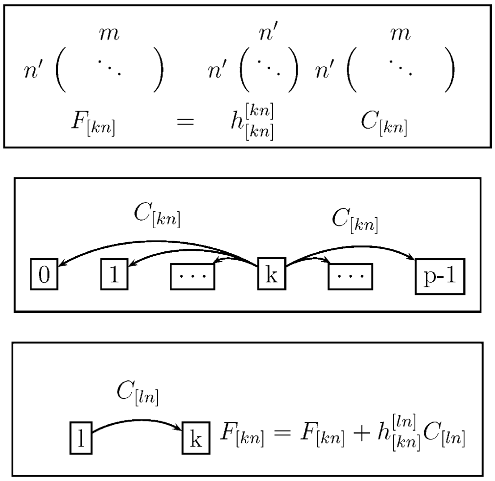

m
n'
m
n'
n'
n'
$F_{[kn]} = h^{[kn]}_{[kn]} C_{[kn]}$
$C_{[kn]}$
$C_{[kn]}$
0
1
···
k
···
p-1
$C_{[ln]}$
l
k
$F_{[kn]} = F_{[kn]} + h^{[ln]}_{[kn]} C_{[ln]}$

Fig. 8. Computation of $F_{[kn]}$ in step 2.

3. Computation of $H_{[km]}$ and $S_{[km]}$.
This step is based on the equalities

$$\begin{aligned} H_{[km]} &= (C^{\mathrm{T}})_{[km]}F = \sum_{i=0}^{p-1}(C^{\mathrm{T}})^{[in]}_{[km]}F_{[in]}, \\ S_{[km]} &= (C^{\mathrm{T}})_{[km]}F^{s} = \sum_{i=0}^{p-1}(C^{\mathrm{T}})^{[in]}_{[km]}(F^{s})_{[in]}. \end{aligned} \tag{A.2}$$

Each processor k multiplies $(C^{\mathrm{T}})^{[kn]}_{[km]}$ by $F_{[kn]}$ and $(F^{s})_{[kn]}$, and copies these products into $H_{[km]}$ and $S_{[km]}$, respectively. It then sends $F_{[kn]}$ and $(F^{s})_{[kn]}$ to all other processors that require it, an order $\leqslant \mathrm{O}(p)$ global operation described in Section 5.3. As each processor k receives $F_{[ln]}$ and $(F^{s})_{[ln]}$ from processor l, $l \neq k$, it multiplies $(C^{\mathrm{T}})^{[ln]}_{[km]}$ by $F_{[ln]}$ and $(F^{s})_{[ln]}$ and adds these products to $H_{[km]}$ and $S_{[km]}$, respectively. At the end of

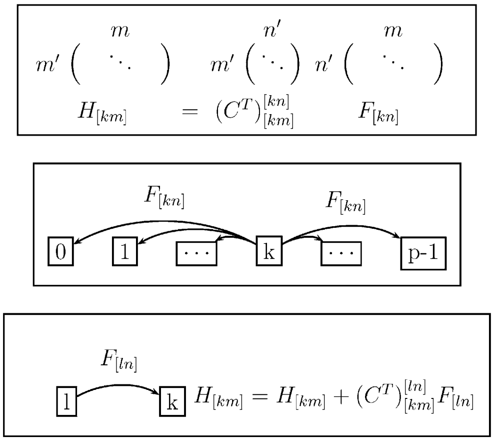

m
n'
m
m'
m'
n'
$H_{[km]} = (C^{T})^{[kn]}_{[km]} F_{[kn]}$
$F_{[kn]}$
$F_{[kn]}$
0
1
···
k
···
p-1
$F_{[ln]}$
l
k
$H_{[km]} = H_{[km]} + (C^{T})^{[ln]}_{[km]} F_{[ln]}$

Fig. 9. Computation of $H_{[km]}$ in step 3.

this step each processor k has computed and locally stored $H_{[km]}$ and $S_{[km]}$. See Fig. 9 for an illustration of the computation of $H_{[km]}$ (the computation of $S_{[km]}$ is similar).

4. Computation of the local gradient $G_{[kn]}$ and energy value $\tilde{E}$.

This step is based on the equality

$$G_{[kn]} = 8F_{[kn]} - 4(F^s)_{[kn]}H - 4F_{[kn]}S = 8F_{[kn]} - 4\sum_{i=0}^{p-1}(F^s)^{[im]}_{[kn]}H_{[im]} - 4\sum_{i=0}^{p-1}F^{[im]}_{[kn]}S_{[im]}. \tag{A.3}$$

Each processor k sets $G_{[kn]}$ to $8F_{[kn]} - 4(F^s)^{[km]}_{[kn]}H_{[km]} - 4F^{[km]}_{[kn]}S_{[km]}$. It then sends $H_{[km]}$ and $S_{[km]}$ to all other processors that require them, an order $\leqslant \mathrm{O}(p)$ global operation described in Section 5.3. As each processor k receives $H_{[lm]}$ and $S_{[lm]}$ from processor l, $l \neq k$, it subtracts $4(F^s)^{[lm]}_{[kn]}H_{[lm]} + 4F^{[lm]}_{[kn]}S_{[lm]}$ from $G_{[kn]}$. At the end of this step each processor k has computed and locally stored $G_{[kn]}$. See Fig. 10 $\{\tilde{E}\} \equiv \{\tilde{E}\}_i$ using 8. Since H and S are symmetric, this may be done by summing the matrix entries and products on the separate processors and using a global gather to sum the contributions of the processors.

5. Computation of conjugate gradient search direction.

To compute the search direction R we use the Polak–Ribiere conjugate gradient method ([33, p. 120]). If $i = 1$ or $\tilde{E}_i > \tilde{E}_{i-1}$ (i.e., the energy has increased from the last iteration) then reset the method by setting $R = G$ (the steepest descent search direction). Otherwise set $R = G + \beta R_{i-1}$, where R_{i-1} and G_{i-1} are the search direction and gradient from the previous iteration, $\bar{A}$ denotes a vectorization of the elements of the matrix A, and $\beta = \bar{G}^{\mathrm{T}}(\bar{G} - \bar{G}_{i-1})/\|\bar{G}_{i-1}\|_2^2$. The vector products are computed using global gather operations.

6. Approximation γ of optimal step length $\tilde{\gamma}$.

To approximate the step length $\tilde{\gamma}$ that minimizes $\tilde{E}$ along the search direction R, we need a trial step length γ_0. We set $\gamma_0 = \min(-\zeta, \{\tilde{\gamma}\}_{i-1})$, where $\tilde{\gamma}_{\{i-1\}}$ is the step length from the previous iteration and ζ is a small positive constant which ensures that the step length has a minimum magnitude (we currently use $\zeta = 0.001$). Use steps 2–4 to compute the energy values $\tilde{E}^1, \tilde{E}^2, \tilde{E}^3, \tilde{E}^4$ at the points $C - 2\gamma_0 R, C - \gamma_0 R, C + \gamma_0 R, C + \gamma_0 R$, respectively. These computations are independent of each other and can be done simultaneously, requiring an order $\leqslant \mathrm{O}(p)$ global communication operation. Each processor computes the coefficients of $P(\gamma)$, the unique fourth-degree polynomial such that $P(-2\gamma_0) = \tilde{E}^1$, $P(-\gamma_0) = \tilde{E}^2$, $P(\gamma_0) = \tilde{E}^3$, $P(2\gamma_0) = \tilde{E}^4$, and $P(0) = \tilde{E}$. Each processor then computes the roots $\{z_1, z_2, z_3\}$ of $\mathrm{d}P/\mathrm{d}\gamma$. Out of these r is the real root which yields the smallest value of the polynomial. To ensure progress and avoid problems with

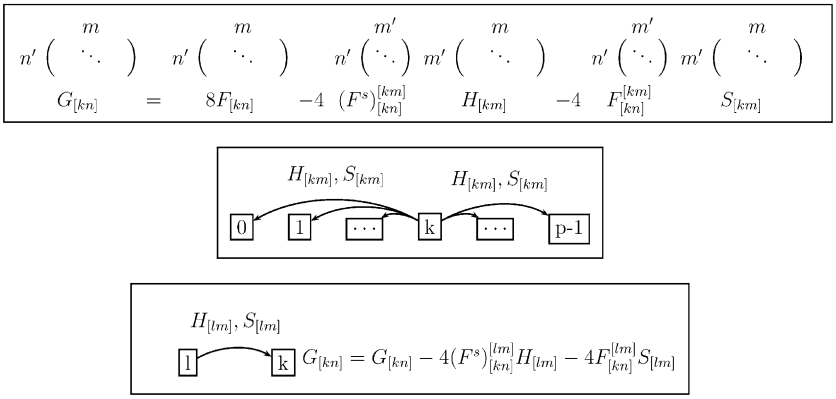


Fig. 10. Computation of $G_{[kn]}$ in step 4.

instability in root finding, we do not allow the step length to be non-negative or too small or large in magnitude. Hence $\gamma = \max(-2\gamma_0, \min(-\zeta, -\mathrm{abs}(r)))$.

7. Update C and test for termination.

Set $C^{i+1} = C + \gamma R$. If the termination conditions of Section 5.2 are satisfied then terminate with $\{\tilde{E}\} = \{\tilde{E}\}_i$, otherwise return to step 2. Observe that on termination, C (not C^{i+1}) is the coefficient matrix whose energy value is $\tilde{E}$.

References

[1] D.E. Bernholdt, E. Apra, J. Nieplocha, Int. J. Quantum Chem. 29 (1995) 475.

[2] S. Itoh, P. Ordejón, R.M. Martin, Comp. Phys. Commun. 88 (1995) 173.

[3] M.J. Frisch et al., Gaussian 98, Revision A.6, Gaussian, Inc., Pittsburgh, PA, 1998.

[4] J. Hutter, P. Ballone, M. Bernasconi, P. Focher, E. Fois, S. Goedecker, D. Marx, M. Parinello, M. Tuckerman, CPMD Version 3.0f, MPI für Festkörperforschung and IBM Research Laboratory, Zürich, 1997.

[5] G. Kresse, J. Furthmüller, Comput. Mater. Sci. 6 (1996) 15.

[6] W. Pan, T.-S. Lee, W. Yang, J. Comput. Chem. 19 (1998) 1101.

[7] P. Ordejón, Comput. Mater. Sci. 12 (1998) 157.

[8] S. Goedecker, Rev. Mod. Phys. 4 (1999) 1085.

[9] O.F. Sankey, D.J. Nikleswki, Phys. Rev. B 40 (1989) 3979.

[10] J. Kim, F. Mauri, G. Galli, Phys. Rev. B 52 (1995) 1640.

[11] J.P. Lewis, K.R. Glaesemann, G.A. Voth, J. Fritsch, A.A. Demkov, J. Ortega, O.F. Sankey, Phys. Rev. B 64 (2001) 195103.

[12] J. Harris, Phys. Rev. B 31 (1985) 1770.

[13] W.M.C. Foulkes, R. Haydock, Phys. Rev. B 39 (1989) 12520.

[14] A.A. Demkov, J. Ortega, O.F. Sankey, M.P. Grumbach, Phys. Rev. B 52 (1995) 1618.

[15] O.F. Sankey, A.A. Demkov, W. Windl, J.H. Fritsch, J.P. Lewis, M. Fuentes-Cabrera, Int. J. Quantum Chem. 69 (1998) 327.

[16] D.R. Hamann, Phys. Rev. B 40 (1989) 2980.

[17] N. Trouiller, J.L. Martins, Phys. Rev. B 43 (1991) 1993.

[18] M. Fuchs, M. Scheffler, Comp. Phys. Commun. 119 (1999) 67.

[19] L. Kleinman, D.M. Bylander, Phys. Rev. Lett. 48 (1982) 1425.

[20] J.P. Perdew, A. Zunger, Phys. Rev. B 23 (1981) 5048.

[21] J.P. Perdew, Phys. Rev. B 33 (1986) 7406.

[22] C. Lee, W. Yang, R.G. Parr, Phys. Rev. B 37 (1988) 785.

[23] A.D. Becke, Phys. Rev. A 38 (1988) 3098.

[24] J.P. Perdew, K.A. Jackson, M.R. Pederson, D.J. Singh, C. Fiolhais, Phys. Rev. B 46 (1992) 6671.

[25] J.P. Perdew, Y. Wang, Phys. Rev. B 45 (1992) 13244.

[26] J.P. Perdew, K. Burke, M. Ernzerhof, Phys. Rev. Lett. 77 (1996) 3865.

[27] J.P. Perdew, K. Burke, Y. Wang, Phys. Rev. B 54 (1996) 16533.

[28] A. Horsfield, Phys. Rev. B 56 (1997) 6594.

[29] M. Finnis, J. Phys.: Condens. Mater. 2 (1990) 331.

[30] M.W. Schmidt, K.K. Baldridge, J.A. Boatz, S.T. Elbert, M.S. Gordon, J.H. Jensen, S. Koseki, N. Matsunaga, K.A. Nguyen, S. Su, T.L. Windus, M. Dupois, J.J.A. Montgomery, J. Comput. Chem. 14 (1993) 1347.

[31] J. Almlöf, in: D.R. Yarkony (Ed.), Modern Electronic Structure Theory, Part I, World Scientific, New Jersey, 1995, p. 110.

[32] P. Ordejón, D.A. Drabold, R.M. Martin, M.P. Grumbach, Phys. Rev. B 51 (1995) 1456.

[33] J. Nocedal, S.J. Wright, Numerical Optimization, Springer, New York, 1999.