PHYSICAL REVIEW B VOLUME 47, NUMBER 4 15 JANUARY 1993-II

Plane-wave electronic-structure calculations on a parallel supercomputer

J. S. Nelson
Semiconductor Physics Division, 1112, Sandia National Laboratories, Albuquerque, New Mexico 87185

S. J. Plimpton and M. P. Sears
Parallel Computational Science Division, 1421, Sandia National Laboratories, Albuquerque, New Mexico 87185

(Received 13 December 1991; revised manuscript received 24 August 1992)

We present a detailed description of the implementation on a parallel supercomputer (hypercube) of the first-order equation-of-motion solution to Schrödinger's equation, using plane-wave basis functions and *ab initio* separable pseudopotentials. By distributing the plane waves across the processors of the hypercube many of the computations can be performed in parallel, resulting in decreases in the overall computation time relative to conventional vector supercomputers. This partitioning also provides ample memory for large fast-Fourier-transform (FFT) meshes and the storage of plane-wave coefficients for many hundreds of energy bands. The usefulness of the parallel techniques is demonstrated by benchmark timings for both the FFT's and iterations of the self-consistent solution of Schrödinger's equation for different sized Si unit cells of up to 512 atoms.

I. INTRODUCTION

The development of iterative solutions of Schrödinger's equation in a plane-wave (PW) basis[1-12] over the past several years has coincided with great advances in the computational power available for performing the calculations. These dual developments have enabled many interesting condensed-matter phenomena to be studied from a first-principles approach. Static calculations (time and temperature independent) that predict the ground-state properties of surfaces or unit cells containing 10–20 atoms can now be performed on desktop workstations. By contrast, dynamical calculations and larger unit cells still require "mainframe" supercomputing. The largest such static calculation that we are aware of has been on a unit cell containing about 100 atoms.[12] A dynamical computation of 100 atoms often requires 50–100 h of central processing time (CPU) and 100 megabytes (MB) of random access memory (RAM) on a vector supercomputer such as the CRAY-YMP. As the size of the unit cell increases to 100 or more atoms (or transition-metal atoms are included which require a larger PW basis set), the CPU and memory demands become nearly insurmountable for even the largest of conventional supercomputers.

However, iterative PW solutions have the advantage of being parallel in nature, meaning that much of the computation can be distributed among cooperating processors. In particular, key components of the computation include fast Fourier transform (FFT's), Gram-Schmidt (GS) orthogonalization of a large number of wave functions, and calculation of the operation of the Hamiltonian on the wave function. As we shall see, each of these calculations has a natural decomposition onto a parallel set of processors.

The current generation of parallel supercomputers are capable of multigigaflop computation rates (1×10^9 floating point operations per second) and, equally important for this application, have gigabytes of internal memory. This additional compute power and memory increases the size of electronic-structure calculations that can be solved in a few hours of CPU time via these iterative techniques.

In this paper, we present the details of the parallel implementation of the first-order equation-of-motion solution to Schrödinger's equation on a 1024-processor nCUBE 2 supercomputer.[13] This processor is a multiple-instruction, multiple-data (MIMD) computer, meaning that each of its processors (nodes) contains its own copy of the program and executes instructions independently on its own local data. The nodes communicate data with each other (e.g., during a FFT computation) by passing messages. Each of the nodes in the nCUBE 2 has 4 MB of memory for a total of 4 gigabytes on the entire machine. An important feature of the nCUBE 2 is that the connectivity of the nodes is that of a ten-dimensional hypercube (1024=2**10), so that each node is connected to ten "neighbor" nodes in the hypercube. This high connectivity is particularly useful for exchanging data quickly during various parts of the computation.

It is useful to note that many of the parallel strategies we will describe here are appropriate for any parallel computer with a high degree of connectivity. In particular, they could be implemented on the single-instruction, multiple-data (SIMD) Connection Machine.[14] This processor differs from a MIMD machine in that only one copy of a program executes synchronously (one instruction at a time) on each processor's local data. However, because the nodes have the underlying connectivity of a hypercube, the parallel decomposition described herein for the MIMD nCUBE 2 should also be an effective choice for the SIMD Connection Machine.

The remainder of the paper is organized as follows: In the next section a formal discussion of the equation-of-motion method for solving Schrödinger's equation and its associated computations is presented. In Sec. III, the partitioning of the computations and data onto the parallel computer is described. In Sec. IV, the FFT routine is presented in more detail and benchmark timings for different sized FFT meshes are given. Finally, in Sec. V, self-consistent iterations for Si unit cells up to 512 atoms are performed and analyzed. These larger calculations demonstrate the increased computational capability that parallel supercomputing provides.

II. EQUATION OF MOTION METHOD: FORMALISM

A. Time integration

There are several iterative approaches to the solution of the Schödinger equation in a PW basis.[1-12] Of these, we focus on the first-order equation of motion approach proposed by Williams and Soler.[3] We begin with a brief review of the formalism (Woodward *et al.*[3]). One approach to the minimization of a function subject to some constraints is the method of steepest descents. Here, the function is minimized by moving in the direction of the negative gradient. Williams and Soler improved on this approach by introducing a first-order equation of motion. In this approach, more information concerning the local gradient is contained over a standard steepest-descent description. The first-order equation is written as

$$\frac{\partial \Psi_{nk}}{\partial t} = -H\Psi_{nk} + \sum_{n'k'} \Lambda_{nk,n'k'}\Psi_{n'k'} , \tag{1}$$

where $\partial/\partial t$ is the fictitious time derivative of the wave function, Ψ_{nk}, H is the Hamiltonian operator, and n and k are the band index and reciprocal lattice points in the irreducible Brillouin zone, respectively. The second term on the right provides the constraint of orthnormality, using the Lagrange multipliers $\Lambda_{nk,n'k'}$. For convenience, in the following we let n represent both the band and the k-point index. Note that Eq. (1) represents the imaginary-time Schrödinger equation for band n. In a PW basis, the wave function Ψ_n is expanded in a Fourier series of reciprocal-lattice vectors (**g**):

$$\Psi_n(\mathbf{r},t) = (1/\sqrt{V}) \sum_{\mathbf{g}} a_n(\mathbf{g},t) \exp[i(\mathbf{k}+\mathbf{g})\cdot\mathbf{r}] , \tag{2}$$

where V is the volume of the crystal and $a_n(\mathbf{g},t)$ is the PW Fourier expansion coefficient. Substituting Eq. (2) into (1), multiplying on the left by $(1/\sqrt{V})\exp[-i(\mathbf{k}+\mathbf{g}')\cdot\mathbf{r}]$, and integrating gives the equation of motion for each wave-function expansion coefficient $a_n(\mathbf{g},t)$:

$$\frac{\partial a_n(\mathbf{g},t)}{\partial t} = -(H_{\mathbf{g},\mathbf{g}} - \Lambda_{n,n})a_n(\mathbf{g},t) + \beta_n(\mathbf{g},t) , \tag{3}$$

where

$$\beta_n(\mathbf{g},t) = \sum_{\mathbf{g}'}{}' H_{\mathbf{g},\mathbf{g}'} a_n(\mathbf{g}',t) + \sum_{n'} \Lambda_{n,n'} a_{n'}(\mathbf{g},t) . \tag{4}$$

The primes on the summations in (4) indicate the diagonal terms, $H_{\mathbf{g},\mathbf{g}}$ and $\Lambda_{n,n}$, are omitted. To integrate Eqs. (3) and (4) forward in time using the scheme of Williams and Soler,[3] we assume that $H_{\mathbf{g},\mathbf{g}}$, $\Lambda_{n,n}$, and β_n are constants and let the Lagrange multipliers $\Lambda_{n,n'}$ be zero (the orthogonalization is taken care of later using the Gram-Schmidt procedure). The result of the integration for a small time step Δt is

$$a_n(\mathbf{g},t+\Delta t) = a_n(\mathbf{g},t) + F_n(\mathbf{g},t) f_n(\mathbf{g},t,\Delta t)\Delta t , \tag{5}$$

where

$$F_n(\mathbf{g},t) = -\sum_{\mathbf{g}'} (H_{\mathbf{g},\mathbf{g}'} - \Lambda_{n,n}\delta_{\mathbf{g},\mathbf{g}'}) a_n(\mathbf{g}',t) , \tag{6}$$

and

$$f_n(\mathbf{g},t,\Delta t) = [\exp(x) - 1]/x, \quad x = -(H_{\mathbf{g},\mathbf{g}} - \Lambda_{n,n})\Delta t . \tag{7}$$

Equations (5)–(7) are iterated until $a_n(\mathbf{g},t+\Delta t) = a_n(\mathbf{g},t)$. After each iteration, the wave functions $\Psi_n(\mathbf{r},t+\Delta t)$ are orthogonalized using the GS procedure. When the solution is converged [i.e., $a_n(\mathbf{g}t+\Delta t)$ is equal to $a_n(\mathbf{g},t)$], the residual vector, $F_n(\mathbf{g},t)$, is equal to zero in Eqs. (5) and (6), and the Schrödinger equation is satisfied.

It is instructive to compare Eq. (5) to the method of steepest descents,[15] where a function is minimized by moving parallel to the negative gradient. If $f_n = 1$ in Eq. (5), then the change in the wave function at a given iteration is parallel to the residual vector $F_n(\mathbf{g},t)$. This is precisely the steepest-descent prescription. Its modification when $f_n = [\exp(x) - 1]/x$, contains information related to the curvature of the steepest-descent path. It therefore leads to faster convergence. Equations (5)–(7) are guaranteed to give $E(t+\Delta t) < E(t)$, as long as the time step is small enough for the approximation that $H_{\mathbf{g},\mathbf{g}'}$, $\Lambda_{n,n}$, and β_n are constant to remain valid.

B. Evaluation of $H\Psi$

The most time-consuming part of the solution to Eqs. (5)–(7) is evaluating the operation of the Hamiltonian on the wave function appearing in the expression for $F_n(\mathbf{g},t)$. In the pseudopotential approach, the Hamiltonian is given by

$$H = E_k + V_{\text{ps}} + V_H + V_{\text{exc}} , \tag{8}$$

where E_k is the kinetic energy, V_{ps} is the pseudopotential describing the electron-ionic-core interaction, V_H is the Hartree interaction between the electrons, and V_{exc} is the exchange-correlation interaction of the electrons. More explicitly, the individual terms are given by the following expressions (units are in rydbergs):

$$E_k = -\nabla^2 , \tag{9}$$

$$V_{\text{ps}}(\mathbf{r}) = \sum_{\tau,\mathbf{R}} V_{\text{ps}}(\mathbf{r} - \mathbf{R} - \tau) , \tag{10}$$

$$\nabla^2 V_H(\mathbf{r}) = -8\pi\rho(\mathbf{r}) , \tag{11}$$

and

$$V_{\text{exc}}(\mathbf{r}) = V_{\text{exc}}(\rho(\mathbf{r})) . \tag{12}$$

The summation in Eq. (10) is over all the atoms τ, in the unit cell, and lattice vectors **R**, in the crystal. The Hartree potential, V_H, is constructed from the Poisson equation [$\rho(\mathbf{r})$ is the pseudovalence charge density], and the exchange-correlation potential, $V_{\rm exc}$ [a functional of $\rho(\mathbf{r})$], through the local-density approximation.[16]

Before the usefulness of separable pseudopotentials[17] was recognized, norm-conserving pseudopotentials of the form[18]

$$V_{\rm ps}(\mathbf{r})=\sum_{\tau,\mathbf{R}}\Big[V_L(\mathbf{r}-\mathbf{R}-\tau)+\sum_{lm}|Y_{lm}\rangle V_{\rm NL}^{l}(\mathbf{r}-\mathbf{R}-\tau)\langle Y_{lm}|\Big]\,, \tag{13}$$

were used, where Y_{lm} are the spherical harmonics with angular momentum lm, and $V_{\rm NL}^l$ and V_L are the nonlocal and local parts of the pseudopotential. In a PW basis, with a potential of this form, one must evaluate $N(N+1)/2$ nonlocal matrix elements for each reciprocal-lattice point k, where N is the size of the PW expansion. The memory required to store these matrix elements, even for a small PW set (~ 1000 PW), is quite large. In addition, efficient coding of the iterative equation of motion method is difficult when the nonlocal component of the pseudopotential is given by Eq. (13). The separable form of the pseudopotential introduced by Kleinman and Bylander[17] has thus been extremely important for the development of the iterative solutions to Schrödinger's equation. A Kleinman and Bylander pseudopotential is separable in both the radial and angular components. It is of the form

$$V'_{\rm ps}(\mathbf{r})=\sum_{\tau,\mathbf{R}}\left[\sum_{lm}[|\delta V_l(\mathbf{r}-\mathbf{R}-\tau)\psi_{lm}^0\rangle\langle\delta V_l(\mathbf{r}-\mathbf{R}-\tau)\psi_{lm}^0|]/N_{lm}+V'_L(\mathbf{r}-\mathbf{R}-\tau)\right]\,, \tag{14}$$

where $N_{lm}=[\langle\psi_{lm}^0|\delta V_l(\mathbf{r}-\mathbf{R}-\tau)|\psi_{lm}^0\rangle]$, $V'_L(\mathbf{r})$ is a completely arbitrary local potential (usually taken to be the highest angular momentum of the valence states), ψ_{lm}^0 is the pseudoatomic wave function from which $V_{\rm ps}$ was constructed, $\delta V_l=V_{\rm ps}^l-V'_L$, and $V_{\rm ps}^l$ is the l component of $V_{\rm ps}$ defined in Eq. (13). By construction, $V'_{\rm ps}\psi_{lm}^0=V_{\rm ps}\psi_{lm}^0$, but $V'_{\rm ps}\psi\sim V_{\rm ps}\psi$, where ψ is an arbitrary wave function. The separable form given in Eq. (14) necessitates the evaluation of only N matrix elements in a PW basis. For details see Refs. 17 and 18.

In the momentum space formalism of the self-consistent pseudopotential method,[19–21] all local operators are represented in Fourier space:

$$\rho(\mathbf{r})=\sum_{\mathbf{g}}\rho(\mathbf{g})\exp[i(\mathbf{g}\cdot\mathbf{r})]\,, \tag{15}$$

$$V_H(\mathbf{r})=\sum_{\mathbf{g}}V_H(\mathbf{g})\exp[i(\mathbf{g}\cdot\mathbf{r})]\,, \tag{16}$$

and

$$V_{\rm exc}(\mathbf{r})=\sum_{\mathbf{g}}V_{\rm exc}(\mathbf{g})\exp[i(\mathbf{g}\cdot\mathbf{r})]\,. \tag{17}$$

The Fourier components of the Hartree, $V_H(\mathbf{g})$, and the exchange-correlation potential, $V_{\rm exc}(\mathbf{g})$, can be easily evaluated by first constructing the real space, $\rho(\mathbf{r})$, and reciprocal space, $\rho(\mathbf{g})$, charge densities. This is most easily accomplished by transforming (FFT) the reciprocal-space representation of the wave function, $\Psi_n(\mathbf{g},t)$, to real space, to obtain $\Psi_n(\mathbf{r},t)$. The real-space density is then simply

$$\rho(\mathbf{r})=\sum_n\Psi_n(\mathbf{r},t)^*\Psi_n(\mathbf{r},t)w_n\,, \tag{18}$$

where w_n is the band and k-point weighting factor. In Eq. (18) we have implicitly assumed a summation over all k points in the irreducible Brillouin zone (IBZ). The summation over the IBZ is usually accomplished via special k-point sampling schemes.[22] The reciprocal-space charge density, $\rho(\mathbf{g})$, is obtained by a FFT on $\rho(\mathbf{r})$. The Hartree and exchange-correlation Fourier components are given by

$$V_H(\mathbf{g})=8\pi\rho(\mathbf{g})/g^2 \tag{19}$$

and

$$V_{\rm exc}(\mathbf{g})=\int V_{\rm exc}(\mathbf{r})\exp[-i(\mathbf{g}\cdot\mathbf{r})]d^3\mathbf{r}\,. \tag{20}$$

The form of $V_{\rm exc}(\mathbf{r})$ depends on the particular exchange-correlation potential used.[23–25] In general, $V_{\rm exc}(\mathbf{r})=V_{\rm exc}[\rho(\mathbf{r})]$, and can be easily evaluated by operating on $\rho(\mathbf{r})$ to obtain $V_{\rm exc}(\mathbf{r})$, and then performing a FFT to get $V_{\rm exc}(\mathbf{g})$.

The evaluation of $F_n(g,t)$ in Eq. (6) is performed in the diagonal representation of each operator (either real or reciprocal space). The kinetic-energy operator [Eq. (9)] is diagonal in reciprocal space; the potential-energy operators, $V_L(\mathbf{r})$ [Eq. (14)], $V_H(\mathbf{r})$ [Eq. (16)], and $V_{\rm exc}(\mathbf{r})$ [Eq. (17)] are diagonal in real space; and the nonlocal potential-energy operator, $\delta V_l(\mathbf{r})$ [Eq. (14)], is diagonal in reciprocal space. The kinetic-energy operator is just $|(\mathbf{k}+\mathbf{g})|^2$. Let us first discuss the nonlocal contribution to $F_n(\mathbf{g},t)$. For each wave-function coefficient, $a_n(\mathbf{g},t+\Delta t)$, and band n (actually band and k point), the expression of $F_n(\mathbf{g},t)$ must be evaluated. The second term in Eq. (6) is straightforward and involves multiplication of the Lagrange multiplier [or expectation value for band n $\varepsilon_n=\langle\Psi_n(\mathbf{r},t)|H|\Psi_n(\mathbf{r},t)\rangle$] and the wave function. Consider the nonlocal contribution to the first term:

$$H\Psi(\mathbf{g})_n^{\rm NL}=\sum_{\mathbf{g}'}H_{\mathbf{g},\mathbf{g}'}^{\rm NL}a_n(\mathbf{g}',t)\,, \tag{21}$$

where we have defined a nonlocal function, $H\Psi(\mathbf{g})_n^{\rm NL}$ ($H\Psi^{\rm NL}$ represents the nonlocal function and does not imply H operating on Ψ). The off-diagonal nonlocal matrix element, $H_{\mathbf{g},\mathbf{g}'}^{\rm NL}$, is given by

$$H^{\mathrm{NL}}_{\mathbf{g},\mathbf{g}'}=\frac{1}{V}\int \exp[-i(\mathbf{k}+\mathbf{g})\cdot\mathbf{r}]\left[\sum_{i,\tau_i,\mathbf{R}}\left[\sum_{lm}|\delta V_l(\mathbf{r}-\mathbf{R}-\tau_i)\psi^0_{lm}\rangle\langle\delta V_l(\mathbf{r}-\mathbf{R}-\tau_i)\psi^0_{lm}|\right]\right](1/N_{lm})\exp[i(\mathbf{k}+\mathbf{g}')\cdot\mathbf{r}]d^3\mathbf{r} \tag{22}$$

or

$$H^{\mathrm{NL}}_{\mathbf{g},\mathbf{g}'}=\sum_i\sum_{\tau_i}\sum_l[4\pi(2l+1)/\Omega]V^i_l(|\mathbf{k}+\mathbf{g}|)F_S(\mathbf{g},\tau_i)V^i_l(|\mathbf{k}+\mathbf{g}'|)F^*_S(\mathbf{g}',\tau_i)P_l(\mathbf{k}+\mathbf{g};\mathbf{k}+\mathbf{g}') , \tag{23}$$

where Ω is the unit-cell volume, i is a summation over all types of atoms in the unit cell, τ_i is over all atoms of type i in the unit cell, and l is over all angular momentum. Respectively, the quantities $V^i_l(|\mathbf{k}+\mathbf{g}|)$, $F_S(\mathbf{g},\tau_i)$, and $P_l(\mathbf{k}+\mathbf{g};\mathbf{k}+\mathbf{g}')$ are the nonlocal potential form factor for atoms of type i and angular momentum l, the atomic structure factor for τ_i, and the lth Legendre polynomial. They are given by the following expressions:

$$V^i_l(|\mathbf{k}+\mathbf{g}|)=\left[\int dr\, r^2 j_l(|\mathbf{k}+\mathbf{g}|r)\delta V_l(r)\psi^0_l(r)\right]\Big/\left[\int dr\, r^2[\psi^0_l(r)]^2\delta V_l(r)\right]^{1/2} , \tag{24}$$

$$F_S(\mathbf{g},\tau_i)=\exp[-i(\mathbf{g}\cdot\tau_i)] , \tag{25}$$

and

$$P_l(\mathbf{k}+\mathbf{g};\mathbf{k}+\mathbf{g}')=\begin{cases}1 & \text{for } l=0\\ (\mathbf{k}+\mathbf{g})\cdot(\mathbf{k}+\mathbf{g}')/|\mathbf{k}+\mathbf{g}||\mathbf{k}+\mathbf{g}'| & \text{for } l=1 .\end{cases} \tag{26}$$

The expression for P_l has been defined for $(l=0,1)$, since in most cases the d pseudopotential is taken as local, whereby only s and p nonlocal corrections will be necessary. Using Eqs. (24)–(26) we can rewrite Eq. (21) as follows:

$$H\Psi(\mathbf{g})^{\mathrm{NL}}_n=\sum_i\sum_{\tau_i}F_S(\mathbf{g},\tau_i)\sum_l\sum_j 4\pi(2l+1)V^i_l(|\mathbf{k}+\mathbf{g}|)p_{lj}(\mathbf{g})\sum_{\mathbf{g}'}[V^i_l(|\mathbf{k}+\mathbf{g}'|)p_{lj}(\mathbf{g}')F^*_S(\mathbf{g}',\tau_i)a_n(\mathbf{g}',t)] , \tag{27}$$

where we have separated the Legendre polynomial, $P_l(\mathbf{k}+\mathbf{g},\mathbf{k}+\mathbf{g}')$, into a product of functions $p_{lj}(\mathbf{g})$ and $p_{lj}(\mathbf{g}')$ by

$$\begin{aligned}&p_{lj}(\mathbf{g})=1, \quad \text{for } l=0 ,\\ &p_{lj}(\mathbf{g})=\mathbf{g}_j/|\mathbf{g}|, \quad j=x,y,z \text{ components of } \mathbf{g}, \quad \text{for } l=1 .\end{aligned} \tag{28}$$

The summation over $\mathbf{g}'$ in Eq. (27) is the same for all the $\mathbf{g}$ components, so that this summation can be performed and all the $\mathbf{g}$ components of $H\Psi(\mathbf{g})^{\mathrm{NL}}_n$ update at once.

We can similarly define a local function, $H\Psi(\mathbf{g})^L_n$ (again, $H\Psi^L$ represents the local function and does not imply H operating on Ψ) describing the operation of the local potential operators on the wave functions.

$$H\Psi(\mathbf{g})^L_n=\sum_{\mathbf{g}'}H^L_{\mathbf{g},\mathbf{g}'}\,a_n(\mathbf{g}',t) . \tag{29}$$

In order to see how to evaluate this expression, we can write it out more explicitly.

$$H\Psi(\mathbf{g})^L_n=\sum_{\mathbf{g}'}\int \exp\{-i((\mathbf{k}+\mathbf{g})\cdot\mathbf{r})\}[V_H(\mathbf{r})+V_{\mathrm{exc}}(\mathbf{r})+V'_L(\mathbf{r})]a_n(\mathbf{g}',t)\exp\{i((\mathbf{k}+\mathbf{g}')\cdot\mathbf{r})\}d^3\mathbf{r} . \tag{30}$$

This expression is the convolution of the total local potential with the wave function. By transforming both the local potential and the wave function to real space, multiplying them on the real-space mesh, then back transforming the product to reciprocal space, all of the $H\Psi(\mathbf{g})^L_n$ can be computed with three FFT's. In practice, the total local potential in square brackets in Eq. (30) is stored on the real-space mesh, so that only one FFT is done on the total local potential for all the bands.

With the updated wave functions from Eq. (5), using the expressions given above, the new charge density can be constructed from Eq. (18), and the new self-consistent potential from Eqs. (19) and (20). The new self-consistent potential and wave functions are used to reevaluate the expression in Eq. (5), and the process is repeated until $a_n(\mathbf{g},t+\Delta t)=a_n(\mathbf{g},t)$. With the converged wave functions, charge density, self-consistent potential, and energy bands other physical properties such as total energies, atomic forces, density of states, etc., can be computed. For a discussion of the total energies and atomic forces see Yin and Cohen.[26] To start the iterative solution, initial wave functions can be obtained from either direct diagonalization of a small wave-function basis or from a random number generator. Typically, the wave functions generated from the diagonalization converge faster.

C. Scaling relations for the computation

The overall scaling of the computational work involved in the iterative equation-of-motion method can be determined from the above expressions. There are two scaling properties that should be distinguished. The first is for a fixed unit-cell size (or equivalently a fixed number of bands) and an increase in the PW's, N, included in the wave-function expansion. This will also increase the size of the three-dimensional (3D) FFT mesh used in the problem since if M is the number of real-space lattice points used in each FFT, then $M \propto N$. Thus when N is increased the overall scaling is proportional to N^2, since Eq. (30) scales as $M \log_2 M$, Eq. (27) as N^2, and the GS orthogonalization as N. The second scaling of interest is for an increase in the unit-cell size for a fixed PW energy cutoff. In this case, the number of bands (proportional to the number of atoms), the number of PW coefficients used to represent each wave function, and the size of the FFT mesh all scale linearly with the volume of the unit cell. Thus, the overall scaling is N^3, since Eqs. (27) and (30) and the GS orthogonalization now scale as N^3.

III. PARTITIONING THE PROBLEM ON THE HYPERCUBE

When performing a large calculation on a parallel machine with distributed memory such as a hypercube, a significant issue is how the data structures of the computation are partitioned across the processors. The goal is to maximize the computation that can be performed simultaneously on the data by all the processors while minimizing the amount of data that must be exchanged between the processors. A typical approach to this issue is to look for a simple idea which subdivides the problem in a convenient way and then to seek possible improvements.

In computing the solution to Eqs. (1)–(30), there are two basic data arrays operated on, one for FFT calculations [in real space and reciprocal space (or g space)] and one for the PW coefficients of the basis vectors. As we shall see in the next section, for fast computation of the FFT's, it is desirable to have an equal number of mesh points stored in the memory of each processor, and to have at least one of the three dimensions of the mesh stored entirely on a particular processor. This is accomplished by splitting the mesh into a two-dimensional array (yz dimensions) of one-dimensional columns (x dimension). The hypercube processors are mapped onto the two-dimensional (yz) grid so that each stores a small subset of columns [Fig. 1(a)].

The second data set (PW coefficients) is a subset of the FFT mesh, since the set of active coefficients is enclosed in a cutoff sphere embedded in the three-dimensional "box" used for the FFT mesh. Since the vector difference between any two g vectors must be contained in the box, the sphere of active g vectors has a diameter at most $\frac{1}{2}$ the shortest dimension of the box [Fig. 1(b)]. If the coefficients were allocated to the processors the same way as the FFT mesh is (several columns of the "box" on each processor) then $\frac{3}{4}$ of the processors would have no coefficients to compute on [see Fig. 1(b)]. However, if the vector of PW coefficients were divided evenly among the processors, then each processor would have to communicate with many others to fill its portion of the FFT mesh with coefficient values computed by the other processors. The partitioning we implement is a compromise between these two extreme cases and is illustrated in Fig. 2. A two-dimensional (yz) slice of the 3D FFT mesh partitioned onto 16 processors is shown with active g-vector coefficients enclosed in the circle. Each processor stores mesh points within one square of the diagram and each square in turn may contain many columns of the FFT mesh. As partitioned in the figure, only four of the processors would store any active coefficients; the other 12 processors would be idle during wave-vector computations. To alleviate this imbalance consider the upper left quadrant of the diagram; the others are handled in a symmetrically similar fashion. Processor 0 breaks its domain into four subdomains (a, b, c, and d) and lets idle processors 1, 2, and 3 each store one of the subdomains. It keeps subdomain 0 for itself. At times in the computa-

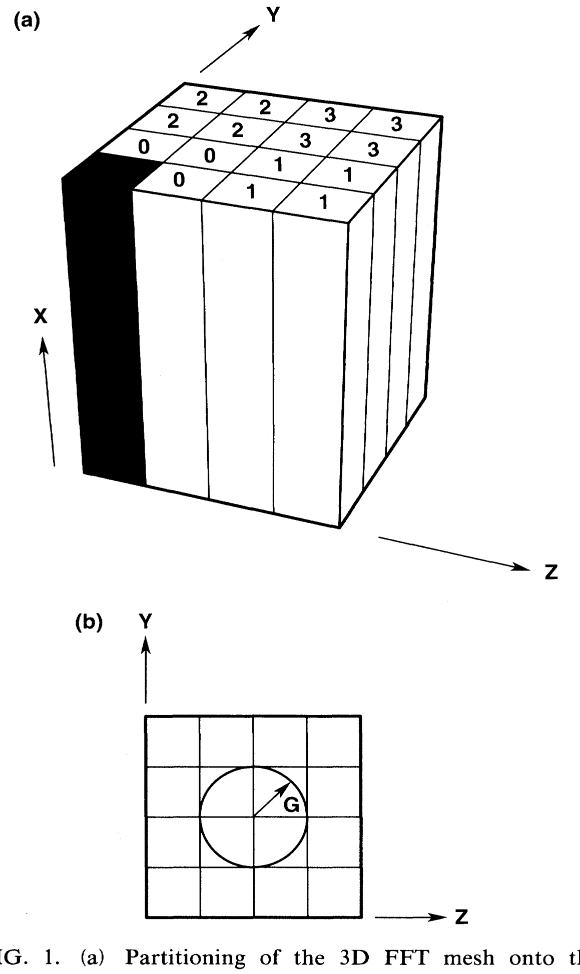


FIG. 1. (a) Partitioning of the 3D FFT mesh onto the memory of the parallel processors. The processors are arranged as a 2D mesh in the yz dimensions. Each stores a few columns of the x direction of the data. In this example each of the four processors (0, 1, 2, and 3) stores four x columns of the $4\times4\times n$ 3D mesh. (b) A projection of the 3D FFT mesh and sphere of active g vectors into two dimensions. FFT's must be computed on all the mesh points, but PW coefficients calculations only need to be performed on mesh values within the cutoff sphere.

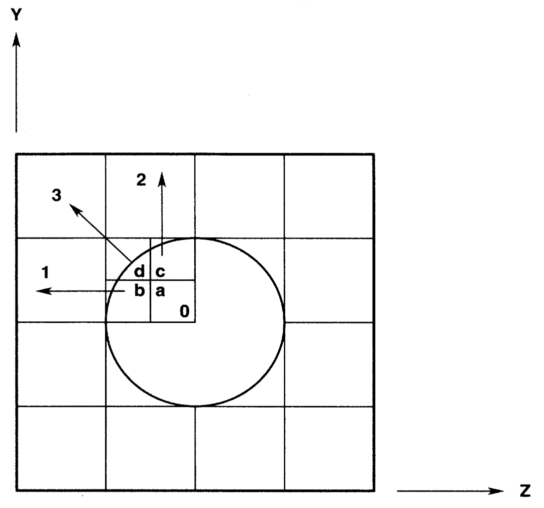


FIG. 2. An x plane of the 3D FFT mesh and the sphere of g-vector coefficients (shaded circle). If the problem is partitioned onto 16 processors, each has one of the square blocks of columns. Processor zero shares its work load with processors 1, 2, and 3 as described in the text.

tion when coefficients must be mapped onto the FFT mesh or vice versa, the three partner processors (1, 2, and 3) communicate their data with processor 0. This can be done very quickly on a hypercube, since each similar set of four partner processors forms a subcube with communication wires directly connecting the four processors to each other. The net result of this subpartitioning is to ensure that every processor has some g vectors to compute on during wave-vector computations and is fully utilized during FFT computations. However, the work load is still not perfectly balanced since some processors have a larger piece of the embedded sphere than others.

With this partitioning, each processor performs its portion of the real- and reciprocal-space lattice summations appearing in Eqs. (5)–(30), then communication routines are used to obtain the global summation or total lattice sum when necessary. Useful hypercube communication routines can be found in the book by Fox *et al.*[27]

IV. FAST-FOURIER-TRANSFORM ROUTINE

Three-dimensional 64-bit complex FFT's are a significant part of the computational task for this problem. In each iteration of the self-consistent solution, $\sim 3n$ FFT's are required, where n is the number of bands for all k points. Two FFT's (forward and reverse) are done for each band in the calculation of Eq. (30) and one FFT is performed for each band to calculate the charge density in Eq. (18). More details can be found in the Appendix.

Timings for our FFT's on various mesh sizes on the nCUBE 2 hypercube are listed in Table I. Timings are given for several powers-of-2 numbers of processors since the nCUBE 2 allows subcubes of the full 1024-processor machine to be used on a problem. The larger FFT's on 1024 processors spend about 65% of their time in communication (transpose), while the computation portion (one-dimensional FFT's) run at a speed of ~ 1.5 megaflops on each processor. Thus the average speed of the entire FFT on the full 1024-processor hypercube is about 0.5 gigaflop. This compares to 290 Mflops for a single YMP processor.[28]

We only list timings for FFT meshes that are power-of-2 in all three dimensions since they are used in the benchmark calculations in this paper. However, we also have a mixed-radix FFT routine[29] (powers of 2, 3, and 5) that we can use on problems that allow intermediate-size meshes. The algorithm for computing the mixed-radix FFT's is similar to that described in this section. The timings are about a factor of 2 slower (for a given total number of mesh points) than those in Table I due to the mixed-radix one-dimensional FFT's being written in a high-level language rather than assembler. Nonetheless, for some problems, the advantage of a smaller FFT mesh (e.g., $40\times40\times40$ instead of $64\times64\times64$) in terms of memory and total computational time required can offset this speed difference.

V. BENCHMARK CALCULATIONS

In this section, benchmark calculations are performed to assess the performance of the parallel code versus a similar version of the code running on the CRAY-YMP. Benchmarks for the nCUBE 2 will be given for 1024 processors. The problems that we will consider consist of bulk Si unit cells containing from 8 to 512 atoms. Timings on a single processor for the CRAY-YMP are given for the 8- and 64-atom unit cells; larger unit cells could not be computed on the CRAY-YMP because of memory limitations. The 512-atom unit-cell computation demon-

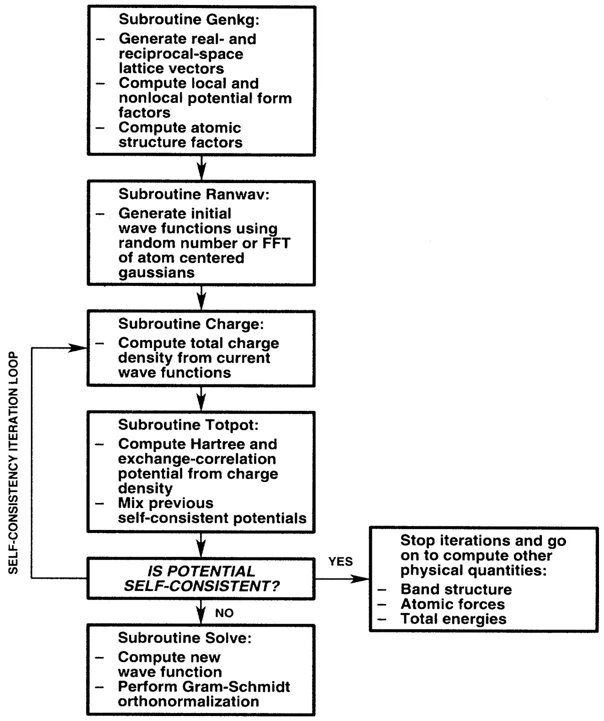
TABLE I. Benchmark timings (in seconds) for the parallel fast Fourier transform (FFT) on the hypercube (nCUBE 2) for various numbers of processors, p. All the FFT's are 3D double-precision complex and restore the transform data to their original locations on the mesh (no bit reversal). Single-processor CRAY-YMP timings are all given for the current CRAY library routines (Ref. 28).

	$Np=128$	$Np=256$	$Np=512$	$Np=1024$	CRAY-YMP
(32,32,32)	0.046 09	0.0270	0.0170	0.011 88	0.0134
(32,32,64)	0.091 6	0.0502	0.0294	0.018 7	
(32,64,64)	0.178 1	0.0975	0.0547	0.031 2	
(64,64,64)	0.360 2	0.1938	0.1063	0.058 9	0.0846
(64,64,128)	0.730 5	0.3891	0.2102	0.113 8	
(64,128,128)	1.480	0.7883	0.4211	0.225 8	0.762
(128,128,128)	2.998	1.591	0.8523	0.453 0	

strates the additional computational power that parallel supercomputers bring to bear to first-principles calculations. It requires virtually all of the 4 Gbytes of nCUBE 2 memory to store the PW coefficients for the 1024 energy bands. The calculations employ the Wigner exchange-correlation potential,[20] a PW cutoff of 10.0 Ry, s and p nonlocal corrections, and one k point in the irreducible Brillouin zone. For each unit cell, the FFT mesh size and the total number of PW's and energy bands are given in Table II.

In Fig. 3 a simple flow chart of the program execution is given. Here we are only considering the self-consistency iterations. The subroutine *genkg* generates the real- and reciprocal-space lattice vectors, and computes all the local and nonlocal pseudopotential form factors. Since this routine is only performed once for a given unit cell, the total CPU timings are an insignificant portion of the overall computation. The initial wave functions (subroutine *ranwav*) can be obtained by several approaches: (1) diagonalization of a small matrix; (2) random number generation; or (3) by FFT Gaussian-based localized atomic wave functions. The self-consistency iterations can be broken up into three main sections. The first section (subroutine *charge*) involves constructing the real- and reciprocal-space charge densities from the updated orthonormalized wave functions. In the second section (subroutine *totpot*), the local Hartree and exchange-correlation potential form factors are computed from the reciprocal- real-space charge densities. This routine also mixes previous self-consistent potential cycles. The third section (subroutine *solve*) solves the equation of motion for all the bands and reciprocal-lattice vectors, and orthonormalizes the updated wave functions using the GS procedure. Timings for each of the three subroutines and the total CPU time for one self-consistent iteration are given in Table II.

The benchmark timings in Table II indicate that the dominant portion of the computation is the solution of the equation of motion and the GS orthonogonalization (subroutine *solve*). The total CPU time spent in the *solve* subroutine can be further decomposed into the amount of



Subroutine Genkg:
– Generate real- and reciprocal-space lattice vectors
– Compute local and nonlocal potential form factors
– Compute atomic structure factors
Subroutine Ranwav:
– Generate initial wave functions using random number or FFT of atom centered gaussians
Subroutine Charge:
– Compute total charge density from current wave functions
Subroutine Totpot:
– Compute Hartree and exchange-correlation potential from charge density
– Mix previous self-consistent potentials
IS POTENTIAL SELF-CONSISTENT?
YES
NO
Stop iterations and go on to compute other physical quantities:
– Band structure
– Atomic forces
– Total energies
Subroutine Solve:
– Compute new wave function
– Perform Gram-Schmidt orthonormalization
SELF-CONSISTENCY ITERATION LOOP

FIG. 3. Schematic diagram of the self-consistent calculation. See text for descriptions of the subroutines given in the figure.

time for the computation of the operation of the total local potential on the wave function [Eq. (29)], the computation of the operation of the nonlocal potential on the wave function [Eq. (27)], and the time spent in the GS orthogonalization. We find that about 30% of the time is spent in the local part, 60% of the time in the nonlocal part, and about 10% in the GS orthogonalization. The other two subroutines make up about 15% of the total CPU time for one iteration.

The "total" timings in Table II also show how the computation is scaling when the number atoms in the unit cell is increased by a factor of 2. The scaling is less

TABLE II. Benchmark timings (in CPU seconds) for one self-consistent iteration of various sized Si unit cells on the 1024-processor *n*CUBE 2. Timings for the CRAY-YMP (single processor) are also given for the 8- and 64-atom unit cells. All the calculations are performed for one k point and a 10-Ry plane-wave energy cutoff. The 512 atom was done with an 8-Ry plane-wave energy cutoff and a more computationally costly *solve* routine to conserve memory. The total entry is the total time for a self-consistent iteration on each machine. The values in parentheses are the scaling ratio on the nCUBE 2 for a factor of 2 increase in unit-cell size. The scaling is found to be less than N^3, but increases towards N^3 with increasing unit-cell size.

Atoms	8	16	32	64	128	256	512
No. PW	587	1173	2335	4625	9261	185 33	310 00
No. bands	17	33	65	129	257	513	1025
No. FFT pts	(32,32,32)	(32,32,64)	(32,64,64)	(64,64,64)	(64,64,128)	(64,128,128)	(128,128,128)
totpot	0.03	0.06	0.10	0.17	0.33	0.64	1.30
solve	1.06	3.02	10.28	41.88	190.55	1019.98	11 211.0
charge	0.28	0.76	2.50	9.19	35.34	139.13	553.9
total (nCUBE)	1.37	3.84	12.89	51.3	226.2	1160.75	117 66
		(2.8)	(3.4)	(3.9)	(4.4)	(5.1)	
total (CRAY)	4.77			226.1			

than N^2 for unit cells less than 64 atoms and approaches N^3 thereafter. The N^3 scaling for the larger unit cells is due to the increasing dominance of the nonlocal part of the computation.

Comparing to a similar version of the code running on the CRAY-YMP, the nCUBE 2 is about a factor of 5 faster when running with all 1024 processors. Although more optimization could likely be done on both versions of the code, these timings show that significant performance enhancements can be obtained with the current generations of massively parallel supercomputors. More importantly, the partitioning of the PW set over the parallel set of processors has allowed a large increase in the size of problems which can be addressed from a first-principles approach.

VI. SUMMARY

We have found that a parallel supercomputer, the 1024-processor nCUBE 2 hypercube, is a factor of 5 times faster than a single processor of the CRAY-YMP for large electronic-structure calculations. This performance factor, along with the ability to store (in memory) large numbers of PW coefficients and large FFT meshes, allows for a significant increase in the unit-cell sizes that can be investigated from a first-principles approach for both static and dynamic calculations. We have considered only one approach to the iterative solution of the Schrödinger equation in a PW basis. Other iterative solutions (e.g., the preconditioned conjugate gradient method of Teter, Payne, and Allan[30]) or programming strategies may be equally good candidates for improvements by parallelization. The next generation of massively parallel supercomputers and new algorithms for electronic-structure calculations should make the calculation from first principles of much larger atomic systems possible. This is an exciting time for the computational electronic-structure theorist.

ACKNOWLEDGMENTS

We would like to thank P. J. Feibelman, W. J. Camp, S. P. Dosanjh, P. S. Peercy, and P. Gourley for many useful discussions and encouragement during this project. This work was supported by the U.S. Department of Energy, under Contract No. DE-AC04-76DP00789.

APPENDIX: PARALLEL IMPLEMENTATION OF FFT'S AND GRAM-SCHMIDT PROCEDURE

In this section we discuss some of the implementation details for the three-dimensional FFT and the parallel Gram-Schmidt algorithms. Both of these have in common that we attempt to do as much work as possible using standard serial programs with a rather small amount of parallel programming. This is very typical of the style of programming needed for parallel message-passing machines like the nCUBE.

1. Parallel FFT's

In order to perform a three-dimensional FFT efficiently, we first partition the processors into a two-dimensional grid, and then partition the three-dimensional data array into a corresponding two-dimensional array of columns. After this partitioning each processor contains a block of columns. Other possible decompositions are into a one-dimensional array of planes or a complete three-dimensional decomposition. The first of these has the great disadvantage that it does not provide sufficient opportunity for parallelism except for extremely large data arrays. For example, if the data array was 32 by 32 by 32 then we could at most partition onto 32 processors, or only about 3% of our 1024-node nCUBE. The three-dimensional decomposition has the disadvantage that it requires the implementation of truly parallel FFT algorithms. Such algorithms have been written, but they require the same data communications as the serial FFT-parallel transpose method, and are much harder to write. In addition, our approach allows the implementation of non-power-of-2 multidimensional FFT's. We are not aware of any other multidimensional FFT implementation for non-power-of-2 lengths on message-passing parallel computers.

The serial FFT algorithm may be a program written and tuned for the particular architecture, or it may be a portable routine. We have tested and used both. For the nCUBE, a tuned assembly language FFT is available for power-of-2 lengths and we wrote a portable FFT for the case of lengths given as products of powers of small primes. The tuned routine performs at about 1.5 Mflops per processor and the portable routine at about 1.0 Mflops per processor on the nCUBE.

After the partitioning, the FFT along the first (i.e., x) axis can be performed using the serial algorithm, since each column of data is stored entirely within memory of a single processor. We now reformat the data from $[x\,y\,z]$ order to $[y\,x\,z]$ order using an $(x\,y)$ parallel transpose algorithm. Then each column along the y axis is now local to a processor and we can use the same serial FFT algorithm to perform the y FFT's. Finally we need an $(x\,z)$ transpose to put the data into $[z\,x\,y]$ order, leaving the x axis where it is. Then the z FFT's can again be performed locally. The data array is then transposed back to the original format with two more transposes. These two additional transposes could be omitted if the forward and backward FFT's are modified to be slightly different and if the user's program is written to handle the different ordering of transformed versus untransformed data. This would double the parallel efficiency of the FFT and improve its absolute performance by about 30% but was not done for the program at hand.

We now turn to the parallel transposes, which is where all of the data communication occurs. We examine only a simple two-dimensional transpose of an n by n matrix on n processors, where each column of the matrix is stored on a single processor. On a hypercube n is a power of 2: $n=2^D$, where D is the dimension of the hypercube. Considering the first row of the matrix we see that each of its elements is stored in a different processor. Therefore the time required for a naive approach, where each processor writes each of its elements to the processor which will contain that element, will be dominated by the overhead for sending a single short message. Since this time is several hundred times longer than the time for the actual data transfer the resulting algorithm will be very slow.

A better approach is to pack and send as much data as possible in a multistage algorithm. The following algorithm is based on a recursive decomposition of the transpose algorithm. We note that a matrix transpose can be accomplished by exchanging the two off-diagonal quadrants, followed by matrix transposes on each of the four quadrants in turn (see Fig. 4). The resulting recursive algorithm is easily adapted to the hypercube architecture, where each processor in the lower half has a direct connection to each processor in the upper half, each quadrant of processors is similarly connected, and so forth. The algorithm will take D steps. At the first step, each processor takes half of its column and exchanges with the equivalent processor in the upper half. The processor in the lower half takes the second half of its column while the processor in the upper half takes the first half of its columns. At the second stage, each processor takes two quarters of its columns, packs them together into a buffer, and exchanges with a processor which is one-quarter of the hypercube away. This goes on, until at the last stage even and odd processors pack every other element of the column and exchange, then unpack the buffer back into the matrix. At each stage the amount of data

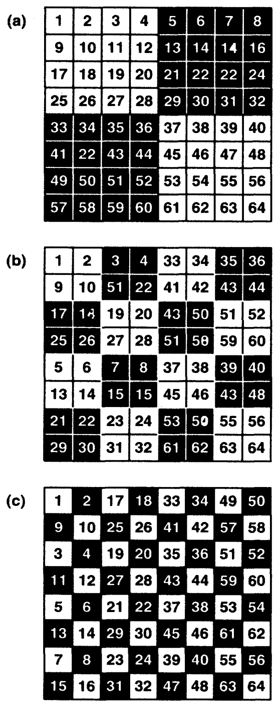


FIG. 4. Recursive matrix transpose for an 8×8 matrix. At the first step of the transpose (a), the upper right quadrant is exchanged with the lower left quadrant. The next step (b), the same algorithm is applied to all four quadrants. In each quadrant, the upper right and lower left subquadrants are exchanged. In the final step (c), the upper right and lower left elements of all 16 sub-sub-quadrants are exchanged. This will leave the matrix in transposed form.

sent and received is the same: one-half of the column length, and each communication is to a nearest neighbor in the hypercube. Although much more complex, this algorithm is about 40 times faster than the naive algorithm on the full 1024-processor nCUBE. We should also point out that there exist algorithms which are intermediate between the naive algorithm and the recursive one which may perform better than both.

Both the $(x\, y)$ and $(x\, z)$ transposes are implemented with this method. Generalizations in the actual code include allowing for multiple blocks and different possible block sizes for the different axes, as well as for the use of array dimensions which are not powers of 2.

2. Further optimizations of the FFT algorithm

Perhaps the most obvious improvement that can be made to our program is to note that most of the data in the 3D FFT are zero, since it lies outside the reduced mesh. It is easy to see that we only need to do the starting (x) FFT's on the core mesh. Then if we transpose x and z, we again only need to do FFT's that intersect the core mesh, and only in the last z step of the 3D FFT do we actually need to use the full mesh. By taking advantage of this savings, the amount of FFT work is reduced by a factor of 4, and a similar savings is available for the reverse FFT.

3. Linear algebra operations: Gram-Schmidt orthogonalization

We now turn our attention to the Gram-Schmidt orthogonalization of the wave function. For each k point we need to orthogonalize all of the wave functions for all the bands. Our column decomposition for this orthogonalization is excellent in one sense, since each processor has identical pieces of all wave functions. However, these pieces vary considerably in length on different processors and therefore different processors have greatly different computational loads.

The Gram-Schmidt (actually the modified GS) algorithm is quite simple. For each unorthogonalized vector ϕ_m we subtract off the overlap with each of the previously orthogonalized vectors ϕ'_n, and when this is done we normalize the vector and add it to the set of orthogonalized vectors. This requires two kinds of parallel operations, a parallel saxpy (i.e., $\alpha x + y \rightarrow y$, where x, y are vectors) and a parallel inner product. The parallel saxpy is trivial, since no communication is needed each processor can perform it independently on its own wave-vector coefficients. The parallel inner product is computed by computing an inner product on each processor and then accumulating the global sum of the local inner products using an exchange and collapse algorithm. Both of these operations can be computed at rates 1.5–2 Mflops/sec per processor on the nCUBE 2. However, due to the load imbalance of each processor not storing the same number of active wave-vector coefficients (described in Sec. III), the net aggregate computational rate for the GS orthogonalization is about 400 Mflops/sec.

[1]R. Car and M. Parinello, Phys. Rev. Lett. **55**, 2471 (1985).

[2]M. C. Payne, J. D. Joannopolous, D. C. Allan, M. P. Teter, and D. H. Vanderbilt, Phys. Rev. Lett. **56**, 2656 (1986).

[3]A. R. Williams and Soler, Bull. Am. Phys. Soc. **32**, 562 (1987); see also C. Woodward, B. I. Min, R. Benedek, and J. Garner, Phys. Rev. B **39**, 4853 (1989).

[4]M. C. Payne, D. C. Allan, and M. P. Teter, Phys. Rev. B **40**, 12 255 (1989).

[5]I. Stich, R. Car, M. Parinello, and S. Baroni, Phys. Rev. B **39**, 4997 (1989).

[6]P. Bendt and A. Zunger, Phys. Rev. Lett. **50**, 1684 (1983).

[7]J. L. Martins and M. L. Cohen, Phys. Rev. B **37**, 6134 (1988).

[8]G. W. Fernando, G. X. Qian, M. Weinert, and J. W. Davenport, Phys. Rev. B **40**, 7985 (1989).

[9]M. Needels, M. C. Payne, and J. D. Joannopoulos, Phys. Rev. Lett. **58**, 1765 (1987).

[10]X. P. Li, P. B. Allen, R. Car, M. Parinello, and J. Q. Broughton, Phys. Rev. B **41**, 3260 (1990).

[11]G. Galli, R. M. Martin, R. Car, and M. Parinello, Phys. Rev. Lett. **62**, 555 (1989).

[12]G. Brocks, P. J. Kelly, and R. Car, Phys. Rev. Lett. **67**, 1728 (1991); recently two large-scale electronic-structure calculations using massively parallel SIMD (single instruction multiple data) computers have appeared: K. D. Brommer, M. Needels, B. E. Larson, and J. D. Joannopoulos, Phys. Rev. Lett. **68**, 1355 (1992); and I. Stich, M. C. Payne, R. D. King-Smith, and J. S. Lin, *ibid.* **68**, 1531 (1992).

[13]nCUBE Corp., 1825 NW 167th Place, Beaverton, OR 97006.

[14]Thinking Machine Corp., 245 First Street, Cambridge, MA 02142-1214.

[15]W. H. Press, B. P. Flannery, S. T. Teukolsky, and W. T. Vetterling, *Numerical Recipes* (Cambridge University Press, Cambridge, 1986).

[16]P. Hohenberg and W. Kohn, Phys. Rev. **136**, B864 (1964); see also W. Kohn and L. J. Sham, *ibid.* **140**, A1133 (1965); in addition, see *Theory of the Inhomogeneous Electron Gas,* edited by S. Lundqvist and N. M. March (Plenum, New York, 1983).

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

[18]D. R. Hamann, Phys. Rev. B **40**, 2980 (1989); D. R. Hamann, M. Schluter, and C. Chiang, Phys. Rev. Lett. **43**, 1494 (1979); G. B. Bachelet, D. R. Hamann, and M. Schluter, Phys. Rev. B **24**, 4190 (1982). A program to generate the pseudopotentials is available by writing to D. R. Hamann.

[19]M. Schluter, J. R. Chelikowsky, S. G. Louie, and M. L. Cohen, Phys. Rev. B **12**, 4200 (1975); J. Ihm, A. Zunger, and M. L. Cohen, J. Phys. C **12**, 4409 (1979); K. C. Pandey, Phys. Rev. Lett. **49**, 223 (1982); I. P. Batra and S. Ciraci, Phys. Rev. B **33**, 4312 (1986).

[20]W. E. Pickett, Comput. Phys. Rep. **9**, 115 (1989).

[21]G. P. Srivastava and D. Weaire, Adv. Phys. **36**, 463 (1987).

[22]D. J. Chadi and M. L. Cohen, Phys. Rev. B **8**, 5747 (1973).

[23]E. Wigner, Phys. Rev. **46**, 1002 (1934).

[24]D. M. Ceperley and B. J. Alder, Phys. Rev. Lett. **45**, 566 (1980); see parametrization by J. Perdew and A. Zunger, Phys. Rev. B **23**, 5048 (1981).

[25]L. Hedin and B. I. Lundqvist, J. Phys. C **4**, 2064 (1971).

[26]M. T. Yin and M. L. Cohen, Phys. Rev. B **26**, 3259 (1982).

[27]G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors* (Prentice-Hall, Englewood Cliffs, NJ, 1988).

[28]M. Merchant (private communication).

[29]M. P. Sears (unpublished).

[30]M. P. Teter, M. C. Payne, and D . C. Allan, Phys. Rev. B **40**, 12 255 (1989). We are currently implementing a parallel version of this approach, which has been shown to converge faster for larger systems than the above equation-of-motion approach.