Computing the Density Matrix in Electronic Structure Theory on Graphics Processing Units

M. J. Cawkwell,*,[†] E. J. Sanville,[†] S. M. Mniszewski,[‡] and Anders M. N. Niklasson[†]

[†]Theoretical Division and [‡]Computer, Computational, and Statistical Sciences Division, Los Alamos National Laboratory, Los Alamos, New Mexico 87545, United States

ABSTRACT: The self-consistent solution of a Schrödinger-like equation for the density matrix is a critical and computationally demanding step in quantum-based models of interatomic bonding. This step was tackled historically via the diagonalization of the Hamiltonian. We have investigated the performance and accuracy of the second-order spectral projection (SP2) algorithm for the computation of the density matrix via a recursive expansion of the Fermi operator in a series of generalized matrix−matrix multiplications. We demonstrate that owing to its simplicity, the SP2 algorithm [Niklasson, A. M. N. *Phys. Rev. B* **2002**, *66*, 155115] is exceptionally well suited to implementation on graphics processing units (GPUs). The performance in double and single precision arithmetic of a hybrid GPU/central processing unit (CPU) and full GPU implementation of the SP2 algorithm exceed those of a CPU-only implementation of the SP2 algorithm and traditional matrix diagonalization when the dimensions of the matrices exceed about 2000 × 2000. Padding schemes for arrays allocated in the GPU memory that optimize the performance of the CUBLAS implementations of the level 3 BLAS DGEMM and SGEMM subroutines for generalized matrix−matrix multiplications are described in detail. The analysis of the relative performance of the hybrid CPU/GPU and full GPU implementations indicate that the transfer of arrays between the GPU and CPU constitutes only a small fraction of the total computation time. The errors measured in the self-consistent density matrices computed using the SP2 algorithm are generally smaller than those measured in matrices computed via diagonalization. Furthermore, the errors in the density matrices computed using the SP2 algorithm do not exhibit any dependence of system size, whereas the errors increase linearly with the number of orbitals when diagonalization is employed.

1. INTRODUCTION

Atomistic simulation is employed heavily in materials science, chemistry, and biology in the study of structures, defects, and equilibrium and nonequilibrium phenomena at the atomic scale. Popular simulation methodologies include molecular statics, where the geometry of an ensemble of atoms is optimized by minimizing the total energy, the Monte Carlo techniques where time independent thermodynamic properties can be evaluated through statistical sampling, and molecular dynamics, where time dependent trajectories for ensembles of particles are computed over a sequence of finite time steps.[1] A unifying aspect of all of these simulation methodologies is their dependence on a suitable interatomic potential, $U = \mathcal{V}(\{\mathbf{R}_k\})$, that gives the potential energy of the system, U, as a function of the coordinates of all of the atoms, $\{\mathbf{R}_k\}$, from which the forces acting on each atom can be derived, $\mathbf{f}_k = -\partial U/\partial \mathbf{R}_k$.[1,2] The ability of an atomistic simulation to capture a given system with high accuracy is determined almost entirely by the physical accuracy of the interatomic potential that is employed.

A variety of interatomic potentials have been employed in atomistic simulation. These range from simple pairwise models, such as the Lennard-Jones potential,[3] that depend only on the separation of two atoms, to the fully quantum mechanical formalisms based on the Hartree−Fock[4] or density functional theories[5,6] and their extensions. It is well established that quantum-based interatomic potentials, where the electronic structure of atoms and molecules is modeled explicitly, provide the most accurate and reliable descriptions of interatomic bonding. Unfortunately the high computational cost associated with these methods, in terms of both a high prefactor and a poor scaling with the number of atoms, N, has limited severely their application to large-scale simulations. Thus, there is a constant requirement for the development of better algorithms and computational methods to accelerate the computation of quantum-based interatomic potentials such that increasingly challenging atomistic simulations can be tackled.

Modern general purpose graphics processing units (GPUs) are attractive candidates for the acceleration of many compute-intensive applications on account of their very high memory bandwidth and peak number of floating point operations per second (FLOPs) compared with contemporary central processing units (CPUs). GPUs are currently employed in several leading high performance computing platforms since their performance, especially on a per Watt or per unit cost basis, exceed significantly those of CPUs. However, many of the algorithms for quantum-based atomistic simulations are not ideally suited to implementation on GPUs owing to difficulties in extracting thread-level parallelism, avoiding branching within warps, and minimizing communication between the CPU and GPU, although significant progress has been made in some areas.[7−11] The algorithms that will utilize GPUs most effectively will be those that tolerate best or avoid their inherent limitations. Furthermore, the vast majority, if not all of the quantum-based electronic structure codes employed by the community are implemented in double precision arithmetic. Until recently

Received: May 30, 2012
Published: September 13, 2012

GPUs either did not support double precision arithmetic or enabled it with a peak FLOP rate that was about an order of magnitude smaller than that achieved in single precision. However, the "Fermi" GPU architecture developed by Nvidia yields a FLOP rate in double precision arithmetic that is only half of that in single precision. This is in accord with relative performance of double and single precision arithmetic on contemporary CPUs.

The density matrix, which is described in detail in section 2, is a key quantity in the computation of the total energy and interatomic forces in many quantum-based potentials (see, for example, refs 2 and 12). In dense matrix algebra, the time required for the computation of the $M \times M$ density matrix scales with the cube of the matrix dimension, $O(M^3)$. Thus, for large systems the calculation of the density matrix may consume a significant fraction of the total computational time. The density matrix has historically been computed from the eigenvalues and eigenvectors of the Hamiltonian matrix, requiring a matrix diagonalization. We have pursued in the quantum-based molecular dynamics code LATTE[13] an alternative approach where the density matrix is computed directly from the Hamiltonian through a recursive expansion of the Fermi operator with the second order spectral projection (SP2) algorithm.[14] The SP2 algorithm is based on a recursive series of generalized matrix–matrix multiplications, $\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}$, where **A**, **B**, and **C** are matrices, α and β are scalars, and $X \leftarrow Y$ denotes the assignment of the value of Y to X. Hence, its performance is controlled by the performance of the Level 3 BLAS DGEMM and SGEMM subroutines, in double and single precision arithmetic, respectively.

The SP2 algorithm was designed originally to facilitate linear scaling, $O(M)$, electronic structure theory in sparse matrix algebra (see, for example, refs 15–18). As such, its performance in the dense matrix limit has not been studied intensively although it has been shown to be efficient in the sparse limit.[19] We have found that the algorithm in dense matrix algebra maps very well onto the capabilities of Nvidia's Fermi-based GPUs. In section 3 we describe two approaches to GPU implementations of the SP2 algorithm. The first is a hybrid CPU/GPU approach that is based on the optimal CPU implementation with the generalized matrix–matrix multiplications transferred to the GPU using the DGEMM or SGEMM subroutines implemented in Nvidia's CUBLAS library. We have also ported the entire SP2 algorithm to the GPU. This implementation exhibits slightly better performance than the hybrid CPU/GPU version since the communication overhead associated with the transfer of arrays between the GPU and CPU is minimized.

In the next section we provide a brief outline of quantum-based models of interatomic bonding, the density matrix, and how the density matrix may be computed from the eigenvalues and eigenvectors of the Hamiltonian or through a recursive expansion of the Fermi-Dirac operator with the SP2 algorithm. In section 3 we describe in detail optimal CPU, hybrid CPU/GPU, and full GPU implementations of the SP2 algorithm. Timings and an analysis of errors in the density matrices computed using all of the algorithms in double and single precision arithmetic are provided in section 4.

2. QUANTUM-BASED MODELS OF INTERATOMIC BONDING

2.1. Theoretical Background. The aim in many quantum-based models of interatomic bonding is the solution of a Schrödinger-like equation for an effective single particle Hamiltonian, $\hat{H}$,

$$\hat{H}|n\rangle = \varepsilon_n|n\rangle \tag{1}$$

where ε_n are the eigenvalues and $|n\rangle$ are the corresponding eigenstates. $\hat{H}$ can be, for example, the Kohn–Sham Hamiltonian,[6] the Fockian,[4] or a parametrized tight-binding Hamiltonian.[2,20] If the eigenstates are expanded in a finite basis, $\{|\phi^{(n)}\rangle\}$, which we assume for simplicity is orthonormal, then

$$|n\rangle = \sum_{i=1}^{M} c_i^{(n)}|\phi_i\rangle \tag{2}$$

where M is the number of basis functions and $\mathbf{c}^{(n)} \equiv \{c_i^{(n)}\}$ are the expansion coefficients. This reduces the quantum eigenvalue problem in eq 1 to the regular matrix eigenvalue equation,

$$\mathbf{H}\mathbf{c}^{(n)} = \varepsilon_n\mathbf{c}^{(n)} \tag{3}$$

where the elements of the Hamiltonian matrix, **H**, are $H_{i,j} = \langle\phi_i|\hat{H}|\phi_j\rangle$. The eigenvalues can be written in terms of the eigenvectors and the Hamiltonian:

$$\varepsilon_n = \mathbf{c}^{(n)\mathrm{T}}\mathbf{H}\mathbf{c}^{(n)} \tag{4}$$

where T denotes the transpose. An important quantity in quantum-based descriptions of interatomic bonding is the band structure energy, E_{band}, which is a sum over the eigenvalues of the occupied eigenstates,

$$E_{\mathrm{band}} = 2\sum_{n=1}^{M} f(\varepsilon_n)\varepsilon_n \tag{5}$$

where we include a factor of 2 to account for spin degeneracy. Here $f(\varepsilon_n)$ is the Fermi-Dirac distribution,

$$f(\varepsilon_n) = 1/[1 + \exp((\varepsilon_n - \mu)/k_{\mathrm{B}}T_{\mathrm{e}})] \tag{6}$$

where μ is the chemical potential, k_{B} is Boltzmann's constant, and T_{e} is the temperature of the electronic subsystem. The chemical potential is determined by the requirement that $2\sum_n f(\varepsilon_n)$ is equal to the total number of electrons, N_{e}. Combining eqs 4 and 5, we may write E_{band} as

$$E_{\mathrm{band}} = 2\mathrm{Tr}[\rho\mathbf{H}] = 2\sum_j\sum_i \rho_{j,i}H_{i,j} \tag{7}$$

Here Tr[**X**] denotes the trace of matrix **X** and we identify the density matrix, ρ, as,

$$\rho = \sum_n f(\varepsilon_n)\mathbf{c}^{(n)}\mathbf{c}^{(n)\mathrm{T}} \tag{8}$$

2.2. Computation of the Density Matrix at Zero Electronic Temperature. At zero electronic temperature, $T_{\mathrm{e}} = 0$ K, the Fermi–Dirac distribution, eq 6, reduces to the Heaviside step function, $f(\varepsilon_n) = \theta[\mu - \varepsilon_n]$, where

$$\theta[x] = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{9}$$

The density matrix can then be computed from the eigenvalues and eigenvectors of the Hamiltonian using

$$\rho = \sum_{n=1}^{M} \theta[\mu - \varepsilon_n]\mathbf{c}^{(n)}\mathbf{c}^{(n)\mathrm{T}} \tag{10}$$

where μ is determined by the condition

$$2\sum_{n=1}^{M}\theta[\mu-\varepsilon_n]=N_e \tag{11}$$

Since the Hamiltonian $\mathbf{H}=\sum_n \varepsilon_n \mathbf{c}^{(n)}\mathbf{c}^{(n)\mathrm{T}}$, the density matrix at zero electronic temperature can also be defined in matrix form using the Heaviside step function,

$$\rho=\theta[\mu\mathbf{I}-\mathbf{H}] \tag{12}$$

where $\mathbf{I}$ is the identity matrix. The SP2 algorithm[14] is based on a recursive expansion matrix form of the density matrix,

$$\theta[\mu\mathbf{I}-\mathbf{H}]=\lim_{i\to\infty} f_i[f_{i-1}[...f_0[\mathbf{X}_0]]...]] \tag{13}$$

where $\mathbf{X}_0$ is the Hamiltonian rescaled such that its eigenvalues occupy in reverse order the interval $[0, 1]$, that is,

$$\mathbf{X}_0=(\varepsilon_{\max}\mathbf{I}-\mathbf{H})/(\varepsilon_{\max}-\varepsilon_{\min}) \tag{14}$$

where $\varepsilon_{\max}$ and $\varepsilon_{\min}$ are the maximum and minimum eigenvalues of $\mathbf{H}$, respectively. Estimates for $\varepsilon_{\max}$ and $\varepsilon_{\min}$ can be obtained rapidly and without the diagonalization of $\mathbf{H}$ from, for example, the Gershgorin circle theorem.[21] The SP2 algorithm employs the following projection polynomials in the recursive expansion:

$$f_i[\mathbf{X}_i]=\begin{cases}\mathbf{X}_i^2 & \text{if } 2\mathrm{Tr}[\mathbf{X}_i]\geq N_e\\ 2\mathbf{X}_i-\mathbf{X}_i^2 & \text{if } 2\mathrm{Tr}[\mathbf{X}_i]<N_e\end{cases} \tag{15}$$

The recursive application of these polynomials "purifies" the density matrix by pushing the unoccupied and occupied eigenvalues toward the fixed points at 0 and 1, respectively, and ensures that the correct occupation is achieved at convergence, that is, $2\mathrm{Tr}[\rho]=N_e$. Prior knowledge of the chemical potential is not required. The SP2 algorithm has been shown to be very efficient since the number of recursive steps that are needed for convergence scales as $\ln(\Delta\varepsilon^{-1})$ with a low prefactor, where $\Delta\varepsilon$ is the gap at the chemical potential.[14] Furthermore, in dense matrix algebra only one $O(M^3)$ matrix–matrix multiplication is required per iteration in eq 15 and interim storage is required for only one $M\times M$ matrix. This is an advantage compared to alternative schemes based on density matrix minimization techniques[22] or higher order polynomial purification.[16,19,23] The SP2 algorithm may be cast as an $O(N)$ scheme by using sparse matrix algebra in conjunction with a numerical threshold on the matrix elements or blocks to maintain high levels of matrix sparsity during the recursive expansion in eqs 13 and 15. Implementations of the SP2 algorithm in sparse matrix algebra are not considered in this work.

3. IMPLEMENTATION OF THE SP2 ALGORITHM ON CENTRAL AND GRAPHICS PROCESSING UNITS

3.1. Convergence Criteria. At zero electronic temperature the density matrix is idempotent; that is, its eigenvalues are either 0 or 1, and

$$\rho=\rho^2 \tag{16}$$

Since the matrix $\mathbf{X}_i$ is not idempotent before the recursive expansion in eq 13 has converged, we may use this property to construct a criterion that identifies at which iteration, i, sufficient idempotency is achieved, for example,

$$|\mathrm{Tr}[\mathbf{X}_i^2-\mathbf{X}_i]|=|\mathrm{Tr}[\mathbf{X}_{i+1}]-\mathrm{Tr}[\mathbf{X}_i]|\leq\chi \tag{17}$$

where χ is a user-defined small tolerance. However, we find that the criterion in eq 17 may lead to undesirable behavior particularly when single precision arithmetic is employed or during molecular dynamics simulations. Owing to the finite precision of floating pointing arithmetic, eq 17 may not be satisfied if the value of χ is of the order of the machine precision, or exactly zero. Alternatively, if the value of χ is set too large, the recursive expansion may stop prematurely such that eq 16 is not satisfied. We instead employ a convergence criterion that is based on a relative measure of idempotency that compares the idempotency measured in iteration i, $|\mathrm{Tr}[\mathbf{X}_{i+1}]-\mathrm{Tr}[\mathbf{X}_i]|$ with that computed in iteration $i-2$, $|\mathrm{Tr}[\mathbf{X}_{i-1}]-\mathrm{Tr}[\mathbf{X}_{i-2}]|$ rather than an absolute measure as in eq 17. Hence, in this approach the recursive expansion is halted when the following condition is satisfied;

$$|\mathrm{Tr}[\mathbf{X}_{i-1}]-\mathrm{Tr}[\mathbf{X}_{i-2}]|\leq|\mathrm{Tr}[\mathbf{X}_{i+1}]-\mathrm{Tr}[\mathbf{X}_i]| \tag{18}$$

that is, the idempotency does not improve over two consecutive iterations. We have found empirically that eq 18 is accurate and extremely reliable during long duration molecular dynamics simulations in both double and single precision arithmetic.

3.2. CPU Implementation. The SP2 algorithm is surprisingly simple to implement since it involves mainly the recursive application of generalized matrix–matrix multiplications. However, it can be implemented in several ways that differ slightly in performance. We found that the optimal schemes were those that allowed us to employ to the greatest extent the full functionality of the BLAS subroutines implemented in highly optimized numerical linear algebra libraries. To improve the convergence of the algorithm we have also modified the criteria for choosing the projection polynomials in eq 15. Instead of choosing f_i to improve the occupation from the previous iteration, we select the f_i that gives the best occupations after the projection,[24]

$$f_i[\mathbf{X}_i]=\begin{cases}\mathbf{X}_i^2 & \text{if } |2\mathrm{Tr}[\mathbf{X}_i]-2\mathrm{Tr}[-\mathbf{X}_i^2+\mathbf{X}_i]-N_e|\\ & \quad\leq|2\mathrm{Tr}[\mathbf{X}_i]+2\mathrm{Tr}[-\mathbf{X}_i^2+\mathbf{X}_i]-N_e|\\ 2\mathbf{X}_i-\mathbf{X}_i^2 & \text{otherwise}\end{cases} \tag{19}$$

The pseudocode for the optimal CPU implementation of the SP2 algorithm is presented in Algorithm 1.

3.3. GPU Implementations. Algorithm 1 is the foundation for the hybrid CPU/GPU and full GPU implementations. The CPU version was implemented in LATTE[13] in Fortran 90 but in the implementations for GPUs we entirely rewrote the Fortran 90 code in CUDA rather than using the Fortran bindings or "thunking" wrappers provided by CUBLAS.[25]

In the pseudocode for the hybrid and full GPU implementations of the SP2 algorithm we denote transfers of data between the CPU and GPU by $\Leftarrow$ and $\Rightarrow$, and those variables and arrays that are allocated in the GPU memory by a tilde.

3.3.1. Hybrid CPU/GPU Implementaiton. Our hybrid CPU/GPU approach to utilizing the high FLOP rate of a GPU in the SP2 algorithm can be considered to be an evolutionary advance on the CPU implementation. In the hybrid version all of the calls to the DGEMM or SGEMM subroutines on the CPU are replaced with calls to the corresponding CUBLAS[25] versions on the GPU. This requires the creation of the arrays $\tilde{\mathbf{X}}_A$ and $\tilde{\mathbf{X}}_B$ on the GPU for the generalized matrix–matrix multiplications. Each purification step requires the transfer of an $M\times M$ matrix to and from the CPU and a memory copy of the contents of $\tilde{\mathbf{X}}_A$

Algorithm 1: Computation of the Density Matrix Using the SP2 Algorithm: CPU Implementation

```
Estimate $\varepsilon_{\max}$ and $\varepsilon_{\min}$
$\mathbf{X} \leftarrow (\varepsilon_{\max}\mathbf{I} - \mathbf{H})/(\varepsilon_{\max} - \varepsilon_{\min})$
TraceX $\leftarrow$ Tr[$\mathbf{X}$]
BreakLoop $\leftarrow 0$
$i \leftarrow 0$
while BreakLoop = 0 do
    $i \leftarrow i+1$
    $\mathbf{X}_{\text{tmp}} \leftarrow \mathbf{X}$
    $\mathbf{X}_{\text{tmp}} \leftarrow -\mathbf{X}^2 + \mathbf{X}_{\text{tmp}}$ /* xGEMM */
    TraceXtmp $\leftarrow$ Tr[$\mathbf{X}_{\text{tmp}}$]
    if $|2\text{TraceX} - 2\text{TraceXtmp} - N_e| > |2\text{TraceX} + 2\text{TraceXtmp} - N_e|$ then
        $\mathbf{X} \leftarrow \mathbf{X} + \mathbf{X}_{\text{tmp}}$
        TraceX $\leftarrow$ TraceX + TraceXtmp
    else
        $\mathbf{X} \leftarrow \mathbf{X} - \mathbf{X}_{\text{tmp}}$
        TraceX $\leftarrow$ TraceX − TraceXtmp
    end if
    IdemErr$_i$ $\leftarrow$ |TraceXtmp|
    if IdemErr$_{i-2} \le$ IdemErr$_i$ then
        BreakLoop $\leftarrow 1$
    end if
end while
$\rho \leftarrow \mathbf{X}$
```

into $\tilde{\mathbf{X}}_B$ on the GPU. The matrix traces and the matrix additions to update the density matrix are computed on the CPU. The pseudocode for this implementation is given in Algorithm 2.

Algorithm 2: Computation of the Density Matrix Using The SP2 Algorithm: Hybrid CPU/GPU Implementation

```
Estimate $\varepsilon_{\max}$ and $\varepsilon_{\min}$
$\mathbf{X} \leftarrow (\varepsilon_{\max}\mathbf{I} - \mathbf{H})/(\varepsilon_{\max} - \varepsilon_{\min})$
TraceX $\leftarrow$ Tr[$\mathbf{X}$]
BreakLoop $\leftarrow 0$
$i \leftarrow 0$
Create on GPU $\tilde{\mathbf{X}}_A$ and $\tilde{\mathbf{X}}_B$
while BreakLoop = 0 do
    $i \leftarrow i+1$
    $\tilde{\mathbf{X}}_A \Leftarrow \mathbf{X}$
    $\tilde{\mathbf{X}}_B \leftarrow \tilde{\mathbf{X}}_A$
    $\tilde{\mathbf{X}}_B \leftarrow -\tilde{\mathbf{X}}_A^2 + \tilde{\mathbf{X}}_B$ /*CUBLAS xGEMM */
    $\tilde{\mathbf{X}}_B \Rightarrow \mathbf{X}_{\text{tmp}}$
    TraceXtmp $\leftarrow$ Tr[$\mathbf{X}_{\text{tmp}}$]
    if $|2\text{TraceX} - 2\text{TraceXtmp} - N_e| > |2\text{TraceX} + 2\text{TraceXtmp} - N_e|$ then
        $\mathbf{X} \leftarrow \mathbf{X} + \mathbf{X}_{\text{tmp}}$
        TraceX $\leftarrow$ TraceX + TraceXtmp
    else
        $\mathbf{X} \leftarrow \mathbf{X} - \mathbf{X}_{\text{tmp}}$
        TraceX $\leftarrow$ TraceX − TraceXtmp
    end if
    IdemErr$_i$ $\leftarrow$ |TraceXtmp|
    if IdemErr$_{i-2} \le$ IdemErr$_i$ then
        BreakLoop $\leftarrow 1$
    end if
end while
$\rho \leftarrow \mathbf{X}$
```

3.3.2. Full GPU Implementation. The full GPU implementation minimizes the transfer of data between the GPU and CPU and enables all of the operations on the matrices to be performed on the GPU. Instead of transferring matrices to and from the GPU and CPU during each iteration, only the rescaled Hamiltonian, $\mathbf{X}_0$ in eq 14, is transferred to the GPU before the recursive application of the projection polynomials in eq 15. The density matrix is transferred back to the CPU once convergence is reached. The selection of which projection polynomial to apply at each iteration (eq 19) takes place on the CPU and requires only the value of the trace of the matrix $\mathbf{X}_{\text{tmp}}$, a single floating point number, to be transferred from the GPU to the CPU at each iteration. The pseudocode for this implementation is given in Algorithm 3.

Algorithm 3: Computation of the Density Matrix Using the SP2 Algorithm: Full GPU Implementation

```
Estimate $\varepsilon_{\max}$ and $\varepsilon_{\min}$
$\mathbf{X} \leftarrow (\varepsilon_{\max}\mathbf{I} - \mathbf{H})/(\varepsilon_{\max} - \varepsilon_{\min})$
BreakLoop $\leftarrow 0$
$i \leftarrow 0$
Create on GPU $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{X}}_{\text{tmp}}$
$\mathbf{X} \Rightarrow \tilde{\mathbf{X}}$
TraceX $\Leftarrow$ Tr[$\tilde{\mathbf{X}}$]
while BreakLoop = 0 do
    $i \leftarrow i+1$
    $\tilde{\mathbf{X}}_{\text{tmp}} \leftarrow \tilde{\mathbf{X}}$
    $\tilde{\mathbf{X}}_{\text{tmp}} \leftarrow -\tilde{\mathbf{X}}^2 + \tilde{\mathbf{X}}_{\text{tmp}}$ /* CUBLAS xGEMM */
    TraceXtmp $\Leftarrow$ Tr[$\tilde{\mathbf{X}}_{\text{tmp}}$]
    if $|2\text{TraceX} - 2\text{TraceXtmp} - N_e| > |2\text{TraceX} + 2\text{TraceXtmp} - N_e|$ then
        $\tilde{\mathbf{X}} \leftarrow \tilde{\mathbf{X}} + \tilde{\mathbf{X}}_{\text{tmp}}$ /* CUBLAS xAXPY*/
        TraceX $\leftarrow$ TraceX + TraceXtmp
    else
        $\tilde{\mathbf{X}} \leftarrow \tilde{\mathbf{X}} - \tilde{\mathbf{X}}_{\text{tmp}}$ /* CUBLAS xAXPY*/
        TraceX $\leftarrow$ TraceX − TraceXtmp
    end if
    IdemErr$_i$ $\leftarrow$ |TraceXtmp|
    if IdemErr$_{i-2} \le$ IdemErr$_i$ then
        BreakLoop $\leftarrow 1$
    end if
end while
$\tilde{\mathbf{X}} \Rightarrow \rho$
```

In both of the GPU implementations of the SP2 algorithm, the $M \times M$ matrices are stored on the GPU as $1 \times M^2$ vectors. This allows us to use the optimized CUBLAS implementations of the Level 1 BLAS DAXPY and SAXPY subroutines to perform the "matrix−matrix" additions on the GPU as vector−vector additions. The matrix traces were computed on the GPU by a reduction across threads using our own kernel.

3.4. Implementation in LATTE. Algorithms 1, 2, and 3 were implemented in the quantum-based molecular dynamics code LATTE. LATTE employs a semiempirical, self-consistent tight-binding model of interatomic bonding that is based heavily on the formalisms derived from density functional theory by Elstner et al.[26] and Finnis et al.[27] Self-consistent tight-binding models are the simplest quantum-based schemes that capture explicitly the making and breaking of covalent bonds, charge transfer between species of differing electronegativities, and long-range electrostatic interactions. The effective single particle Hamiltonian is a sum of two terms,

$$H_{i\alpha,j\beta} = H^{(0)}_{i\alpha,j\beta} + H^{(1)}_{i\alpha,j\beta} \tag{20}$$

where the first term is the Slater−Koster Hamiltonian[28] that represents the overlap between orbital α on atom i with orbital β on atom j. The second term represents an electrostatic potential arising from the set of atom centered Mulliken partial charges, $\{q_i\}$,

$$H^{(1)}_{i\alpha,j\beta} = \sum_k \gamma_{i,k} q_k \delta_{i,j} \delta_{\alpha,\beta} \tag{21}$$

where γ is a screened Coulomb potential[26] and $\delta_{x,y}$ is the Kronecker delta. The Mulliken charges are computed from the density matrix via,

$$q_i = 2\sum_{\alpha \in i} (\rho_{i\alpha,i\alpha} - \rho^0_{i\alpha,i\alpha}) \tag{22}$$

where ρ^0 is the density matrix for isolated atoms. Since the Hamiltonian depends on the density matrix via the Mulliken partial charges, eq 12 must be solved self-consistently.

LATTE represents the eigenstates of the system with a minimal, orthogonal basis of atomic-like orbitals, eq 2. We employed in our analysis of the relative performance and accuracy of the three implementations of the SP2 algorithm our parametrization of a self-consistent tight-binding model for hydrocarbons.[29] This model uses a minimal basis of one s orbital per hydrogen atom, $|s\rangle$, and one s and three p orbitals per carbon atom, $|s\rangle$, $|p_x\rangle$, $|p_y\rangle$, and $|p_z\rangle$.

4. ANALYSIS OF PERFORMANCE AND ERRORS

All of the calculations reported in this section were performed on a single node of the Keeneland cluster at the National Institute for Computational Sciences.[30] Each node comprises two hex-core X5660 Intel Xeon CPUs with 24 GB of memory and three Nvidia M2070 GPUs each with 6 GB of memory. The CPU code was compiled with the Intel Fortran Compiler version 12 with the -fast flag and the Intel MKL version 10 implementation of the LAPACK and BLAS libraries. All of the computationally intensive nested loops in the CPU code were threaded using OpenMP and we used a threaded version of the Intel MKL. All of the timings we report were computed with an optimal number of CPU threads up to a maximum of 12. Matrix diagonalizations in double and single precision were performed using the LAPACK subroutines DSYEV and SSYEV, respectively. The hybrid CPU/GPU and full GPU implementations of the SP2 algorithm used only one of the three GPUs available on each node. The CUDA subroutines for the Nvidia GPU were compiled with the software and CUBLAS libraries provided with the CUDA Toolkit version 4.0,[25,31] and with a thread block size of 1024.

4.1. Padding Arrays on the GPU. It was shown in ref 32 that the performance of the CUBLAS DGEMM and SGEMM subroutines depends sensitively on the size of the matrix, M. On the Fermi architecture, the CUBLAS DGEMM and SGEMM reach 95% of their maximum FLOP rates for matrices with $M \geq 1088$ and $M \geq 1344$, respectively. While the general trend is that the FLOP rates increase with increasing M, the improvements in performance are not monotonic. Instead, the FLOP rate exhibits distinct, sharp "spikes" as a function of M, where a difference in the FLOP rate of about 25% may separate adjacent maxima and minima. Song et al. report that the adjacent peaks are separated by $\Delta M = 64$ and 96 in double and single precision, respectively, for large M.[32] These results suggest that matrices on the GPU should be padded by allocating arrays whose dimensions, M_{pad}, are only integer multiples of either 64 or 96, depending on the precision of the arithmetic, that is,

$$M_{\text{pad}} = \Delta M\left(\left\lfloor \frac{M-1}{\Delta M}\right\rfloor + 1\right) \tag{23}$$

where $\lfloor x \rfloor$ denotes the floor of x that is equal to the nearest integer $\leq x$. When padding is used the $M \times M$ matrix forms a block within the $M_{\text{pad}} \times M_{\text{pad}}$ matrix and the remaining $M_{\text{pad}}{}^2 - M^2$ elements are set equal to zero. The use of padded arrays does not effect the results of the matrix−matrix multiplication, addition, or scalar multiplication on the $M \times M$ blocks.

We have investigated further the effects of padding by computing the average time to execute the DGEMM and SGEMM CUBLAS subroutines as a function of matrix size in the interval $1 \leq M \leq 4000$. The average execution times were computed over 10 iterations of the generalized matrix−matrix multiplication $\mathbf{C} \leftarrow \mathbf{AB} + \mathbf{C}$, and do not include memory transfers between the CPU and GPU. The elements of the $M \times M$ matrices $\mathbf{A}$ and $\mathbf{B}$ were assigned random numbers in the interval [0, 1]. These timings showed that the simple padding scheme given by eq 23 is not optimal, particularly for small M. We find that the padding schemes that provide the smallest wall clock timings for the execution of the CUBLAS DGEMM and SGEMM subroutines are

$$M_{\text{pad}} = \begin{cases} M & \text{if } M \leq 720 \\ 832 & \text{if } 720 < M \leq 832 \\ 64\lfloor M/64 \rfloor & \text{if } M > 832 \text{ and } M = 64\lfloor M/64 \rfloor \\ 64\lfloor M/64 \rfloor + 16 & \text{if } M > 832 \text{ and } 64\lfloor M/64 \rfloor < M \leq 64\lfloor M/64 \rfloor + 16 \\ 64\lfloor M/64 \rfloor + 32 & \text{if } M > 832 \text{ and } 64\lfloor M/64 \rfloor + 16 < M \leq 64\lfloor M/64 \rfloor + 32 \\ 64(\lfloor M/64 \rfloor + 1) & \text{if } M > 832 \text{ and } M > 64\lfloor M/64 \rfloor + 32 \end{cases} \tag{24}$$

in double precision, and,

$$M_{\text{pad}} = \begin{cases} 64(\lfloor M - 1/64 \rfloor + 1) & \text{if } M \leq 704 \\ 96\lfloor M/96 \rfloor & \text{if } M > 704 \text{ and } M = 96\lfloor M/96 \rfloor \\ 96\lfloor M/96 \rfloor + 16 & \text{if } M > 704 \text{ and } 96\lfloor M/96 \rfloor < M \leq 96\lfloor M/96 \rfloor + 16 \\ 96\lfloor M/96 \rfloor + 32 & \text{if } M > 704 \text{ and } 96\lfloor M/96 \rfloor + 16 < M \leq 96\lfloor M/96 \rfloor + 32 \\ 96(\lfloor M/96 \rfloor + 1) & \text{if } M > 704 \text{ and } M > 96\lfloor M/96 \rfloor + 32 \end{cases} \tag{25}$$

in single precision, respectively. We present in Figure 1 examples of the effects of the padding schemes in eqs 24 and 25

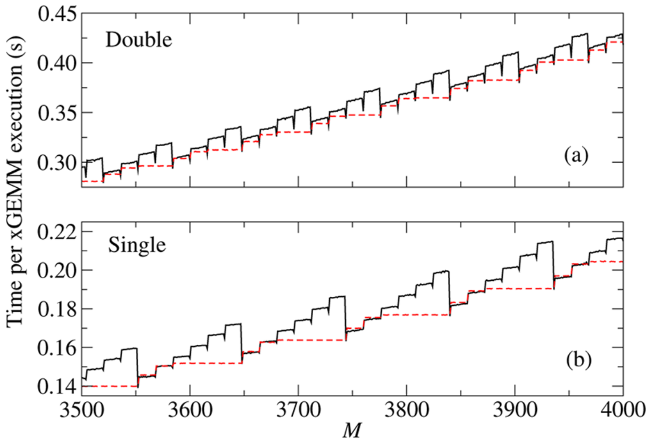

Figure 1. Average time to execute the CUBLAS xGEMM generalized matrix−matrix multiplication for $M \times M$ matrices. (a) DGEMM, (b) SGEMM. The broken and solid lines correspond to computations performed with and without the padding of the arrays, respectively.

on the performance of the CUBLAS DGEMM and SGEMM. The padding of the arrays on the GPU provides improvements in the wall clock time for the execution of a generalized matrix−matrix multiplication of up to 7% and 12% in double and single precision arithmetic, respectively.

4.2. Measures of Error in Density Matrices. In addition to assessing the performance of the CPU and GPU implementations of the SP2 algorithm with respect to standard algorithms based on diagonalization, we also evaluate and compare the accuracy of the computed density matrices. The errors in the density matrices are quantified using measures of the idempotency,

$$\varepsilon_{\text{idem}} = \|\rho^2 - \rho\|_2 \tag{26}$$

the commutation of the density matrix with the Hamiltonian,

$$\varepsilon_{\text{comm}} = \|[\mathbf{H}, \rho]\|_2 \tag{27}$$

and the normalized error in the occupancy,

$$\varepsilon_{\text{occ}} = |2\text{Tr}[\rho] - N_e|/N_{\text{orb}} \tag{28}$$

where $\|\mathbf{X}\|_2$ denotes the spectral norm of $\mathbf{X}$ and N_{orb} is the total number of electronic orbitals. For the exact density matrix, $\varepsilon_{\text{idem}}$, $\varepsilon_{\text{comm}}$, and ε_{occ} all equal zero.

4.3. Application to Liquid Methane. The relative performance and accuracy of the three implementations of the SP2 algorithm and a diagonalization-based algorithm, eq 10, were assessed in single and double precision arithmetic using a series of static, total energy calculations on liquid methane. Liquid methane is one of the simplest systems where covalent bonding and charge transfer take place, and because of the finite overlap between orbitals on neighboring molecules it is an extended quantum mechanical system. We employed a series of periodic cells containing from 10 to 1000 CH_4 molecules ($80 \leq N_{\text{orb}} \leq 8000$) at the experimental value of the density of the liquid. Since CUBLAS does not take into account any sparsity in the matrices, the timings we report depend mainly on the energy gap at the chemical potential rather than the chemical species.

In Figures 2 and 3 we plot the average over all self-consistent field cycles of the wall clock time to compute the density matrix via diagonalization or with the SP2 method using Algorithms

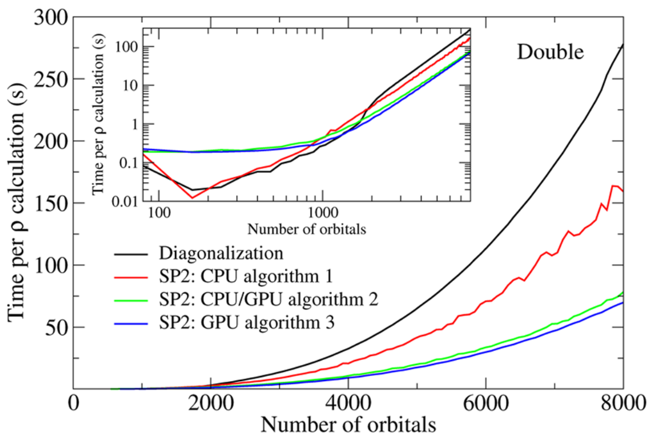

Figure 2. Time per density matrix calculation using diagonalization and SP2 purification with Algorithms 1, 2, and 3 in double precision arithmetic.

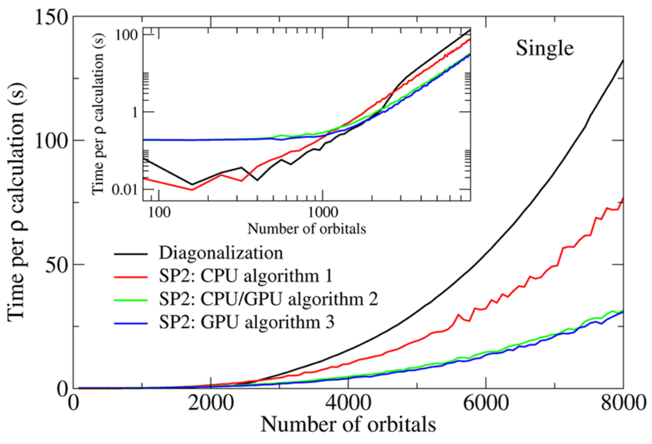

Figure 3. Time per density matrix calculation using diagonalization and SP2 purification with Algorithms 1, 2, and 3 in single precision arithmetic.

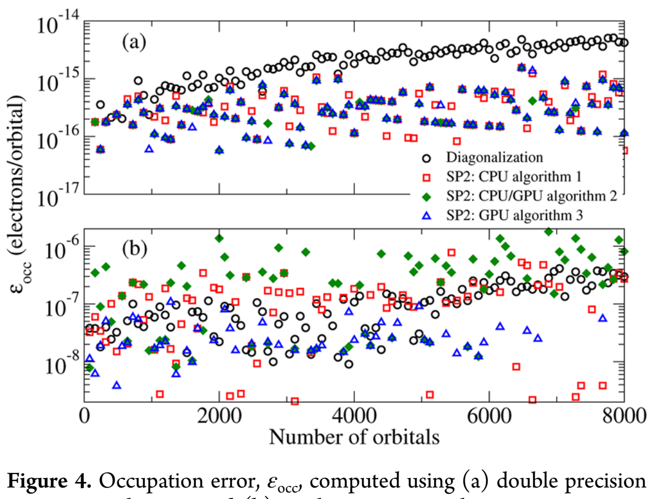

Figure 4. Occupation error, ε_{occ}, computed using (a) double precision precision arithmetic, and (b) single precision arithmetic.

1, 2, and 3, in double and single precision arithmetic, respectively. The insets in each figure show that for small systems ($N_{\text{orb}} \lesssim 2000$) the performance of the diagonalization algorithm and the CPU implementation of SP2 algorithm exceed those of the hybrid and full GPU implementations in double and single precision arithmetic. We attribute these results to the overheads associated with data transfer between the CPU and GPU and the low FLOP rates measured on GPUs for small matrices. For larger systems (N_{orb} > 2000) we find that the hybrid and full GPU implementations of the SP2 algorithm are faster than diagonalization and the CPU implementation of the SP2 algorithm in double and single precision arithmetic. A comparison between the performance of hybrid CPU/GPU implementation in Algorithm 2, where an $N_{orb} \times N_{orb}$ matrix is transferred from the CPU to the GPU and back again at each recursive iteration in eq 15 and the full GPU implementation in Algorithm 3 where a $N_{orb} \times N_{orb}$ matrix is transferred from the CPU to the GPU at the start of the algorithm and returned to the CPU at completion, show clearly that data transfer between the CPU and GPU comprises a relatively small part of the overall computational time. Indeed, the speed-ups that arise from the minimization of data transfer in Algorithm 3 as compared with Algorithm 2 are modest for all values of N_{orb}. Moreover, as N_{orb} becomes large, $O(N_{orb}^2)$ data transfer becomes an increasingly small fraction of the overall $O(N_{orb}^3)$ generalized matrix−matrix multiplication procedure such that for the largest systems the performance of hybrid CPU/GPU and full GPU implementations are almost indistinguishable.

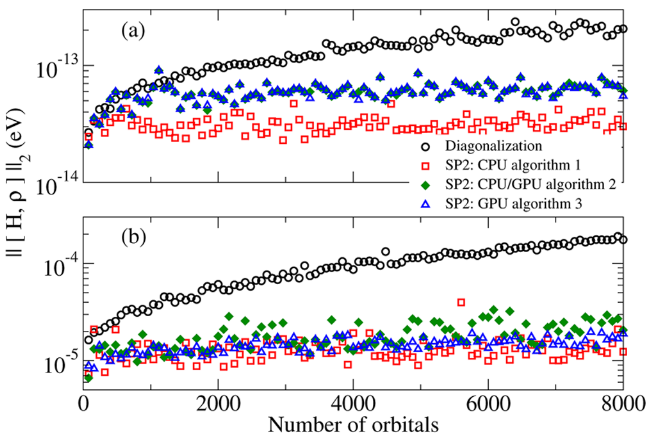

Figure 5. Commutation error, ε_{comm}, computed using (a) double precision precision arithmetic and (b) single precision arithmetic.

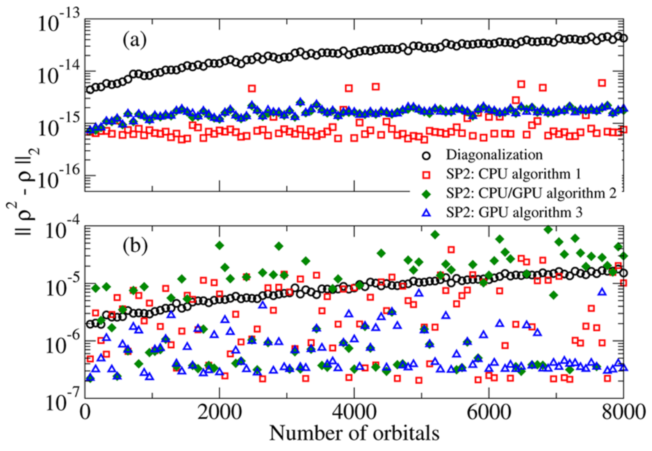

Figure 6. Idempotency error, ε_{idem}, computed using (a) double precision arithmetic and (b) single precision arithmetic.

The dependence of the occupation, commutation, and idempotency errors in the density matrix on the number of orbitals and the method of calculation are presented in Figures 4, 5, and 6, respectively. The surprising result of these analyses is that the three algorithms based on the SP2 expansion yield density matrices with errors that are as good or smaller than those computed via traditional matrix diagonalization.

In double precision arithmetic, the values of ε_{occ} computed with the SP2 algorithm are about an order of magnitude smaller than those measured in the density matrices computed using diagonalization. As shown in Figure 4a, in double precision arithmetic, Figure 4b, we find that the values of ε_{occ} computed using all four methods are comparable and about 8 orders of magnitude greater than the errors measured from identical calculations performed in double precision arithmetic. The commutation errors, ε_{comm}, shown in Figure 5 panels a and b measured using density matrices computed with diagonalization in double and single precision arithmetic are larger than those measured in the corresponding matrices computed using the SP2 algorithm. Furthermore, the values of ε_{comm} for density matrices computed in double precision arithmetic with the

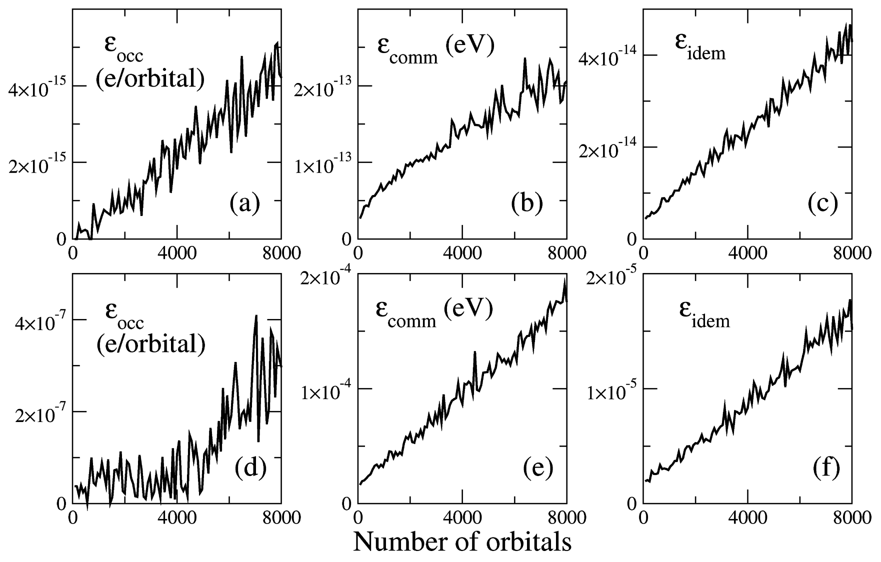

Figure 7. Dependence of the occupation, commutation, and idempotency errors in the density matrices computed via diagonalization on the number of orbitals: (a, b, c), double precision, (d, e, f), single precision.

CPU implementation of the SP2 algorithm are marginally smaller than those computed with the hybrid CPU/GPU and full GPU implementations. However, as shown in Figure 5b, in single precision all three implementations of the SP2 algorithm yield similar values of $\varepsilon_{\text{comm}}$. Figure 6a shows that the idempotency errors, $\varepsilon_{\text{idem}}$, in the density matrices computed using double precision arithmetic with the CPU, hybrid CPU/GPU, and full GPU implementations of the SP2 algorithm are about an order of magnitude smaller than those measured in density matrices computed using diagonalization. Figure 6b shows that the idempotency errors arising from the three implementations of the SP2 algorithm in single precision are generally smaller, but exhibit considerably greater scatter, than those obtained from the diagonalization technique.

Figures 4, 5, and 6 show that the occupation, commutation, and idempotency errors that arise from the computation of the density matrix with the SP2 algorithm do not depend on the size of the system. However, as shown in Figure 7, the errors in the density matrices computed via diagonalization increase linearly with the system size. The observed size dependence of the errors is likely to be related to thresholds on orthogonality of the eigenvectors in the implementation of the DSYEV and SSYEV subroutines.

5. CONCLUSIONS

We have demonstrated how graphical processing units may be used effectively in the computation of the density matrix in quantum mechanical models for interatomic bonding. Owing to its simplicity, the recursive SP2 algorithm is eminently suited to implementation on GPUs despite it being conceived originally for reduced complexity, linear scaling calculations in sparse matrix algebra. Our timings showed that the two GPU-based implementations of the SP2 algorithm outperform a highly optimized implementation on a multicore CPU in double and single precision arithmetic when the dimensions of the density matrix exceeds about 2000 × 2000. Our results indicate that the penalties associated with the transfer of arrays between the CPU and GPU are a minor burden on the overall performance compared with those encountered if the problem size is small compared with those required for the GPU to approach its maximum FLOP rate. We have also demonstrated the beneficial effect of padding the arrays in the GPU memory on the performance of the CUBLAS implementations of the level 3 BLAS DGEMM and SGEMM subroutines. Our results reveal that the performance of a hybrid CPU/GPU implementation, where only the most computationally intensive, $O(N_{\text{orb}}^3)$, operations are transferred to the GPU approaches very closely the performance of the optimized, fully integrated GPU code. The analysis of the errors in the self-consistent density matrices show that the SP2 algorithm provides as good or better accuracy compared with traditional approaches based on diagonalization. Furthermore, the accuracy of the density matrices computed on the GPU with the SP2 algorithm are very similar to those computed only on the CPU in both double and single precision arithmetic. We have also demonstrated that the errors in the density matrices computed using the SP2 algorithm do not depend on the size of the system, whereas the size of the errors increase linearly with the number of atoms when diagonalization is employed.

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: cawkwell@lanl.gov.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

We thank Nick Bock, Kyle Spafford, and Jeffrey Vetter for stimulating discussions. This work was supported by the LDRD program at Los Alamos National Laboratory.

■ REFERENCES

(1) Allen, M. P.; Tildesley, D. J. *Computer Simulation of Liquids*; Oxford University Press: Oxford, 1989.
(2) Finnis, M. W. *Interatomic Forces in Condensed Matter*; Oxford University Press: Oxford, 2003.
(3) Lennard-Jones, J. E. *Proc. R. Soc. London A.* **1924**, *106*, 463−477.
(4) Roothaan, C. C. J. *Rev. Mod. Phys.* **1951**, *23*, 69−89.
(5) Hohenberg, P.; Kohn, W. *Phys. Rev.* **1964**, *136* (B), 864−B871.
(6) Kohn, W.; Sham, L. J. *Phys. Rev.* **1965**, *140*, 1133.
(7) Ufimtsev, I. S.; Martinez, T. J. *Comp. Sci. Eng.* **2008**, *10*, 26−34.
(8) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2008**, *4*, 222−231.
(9) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 1004−1015.
(10) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 2619−2628.
(11) Genovese, L.; Ospici, M.; Deutsch, T.; Mehaut, J.-F.; Neelov, A.; Goedecker, S. *J. Chem. Phys.* **2009**, *131*, 034103.
(12) McWeeny, R. *Proc. R. Soc. London, Ser. A* **1956**, *235*, 496.
(13) Sanville, E. J.; Bock, N.; Coe, J.; Mniszewski, S. M.; Niklasson, A. M. N.; Cawkwell, M. J. LATTE. Los Alamos National Laboratory (LA-CC 10-004), 2010; http://savannah.nongnu.org/projects/latte (accessed September 2012).
(14) Niklasson, A. M. N. *Phys. Rev. B* **2002**, *66*, 155115.
(15) Millam, J. M.; Scuseria, G. E. *J. Chem. Phys.* **1997**, *106*, 5569.
(16) Daniels, A. D.; Scuseria, G. E. *J. Chem. Phys.* **1999**, *110*, 1321−1328.
(17) Bowler, D. R.; Miyazaki, T. *Rep. Prog. Phys.* **2012**, *75*, 036503.
(18) Goedecker, S. *Rev. Mod. Phys.* **1999**, *71*, 1085−1123.
(19) Rudberg, E.; Rubensson, E. H. *J. Phys.: Condens. Matter* **2011**, *23*, 075502.
(20) Sutton, A. P.; Finnis, M. W.; Pettifor, D. G.; Ohta, Y. *J. Phys. C (Solid State)* **1988**, *21*, 35−66.
(21) Golub, G.; van Loan, C. F. *Matrix Computations*; Johns Hopkins University Press: Baltimore MD, 1996; p 200.
(22) Li, X. P.; Nunes, R. W.; Vanderbilt, D. *Phys. Rev. B* **1993**, *47*, 10891.
(23) Palser, A. H. R.; Manolopoulos, D. E. *Phys. Rev. B* **1998**, *58*, 12704.
(24) Niklasson, A. M. N.; Weber, V.; Challacombe, M. *J. Chem. Phys.* **2005**, *123*, 44107.
(25) CUDA Toolkit 4.0 CUBLAS Library. 2011.
(26) Elstner, M.; Poresag, D.; Jungnickel, G.; Elsner, J.; Haugk, M.; Frauenheim, T.; Suhai, S.; Seifert, G. *Phys. Rev. B* **1998**, *58*, 7260.
(27) Finnis, M. W.; Paxton, A. T.; Methfessel, M.; van Schilfgaarde, M. *Phys. Rev. Lett.* **1998**, *81*, 5149.
(28) Slater, J. C.; Koster, G. F. *Phys. Rev.* **1954**, *94*, 1498.
(29) Sanville, E.; Bock, N.; Niklasson, A. M. N.; Cawkwell, M. J.; Dattelbaum, D. M.; Sheffield, S. In *Proceedings of the 14th International Detonation Symposium*; Office of Naval Research, 2010; p 91.
(30) Vetter, J. S.; Glassbrook, R.; Dongarra, J.; Schwan, K.; Loftis, B.; McNally, S.; Meredith, J.; Rogers, J.; Roth, P.; Spafford, K.; Yalamanchili, S. *Comput. Sci. Eng.* **2011**, *13*, 90−95.
(31) *CUDA API Reference Manual*, version 4.0; Nvidia, Santa Clara, CA, 2011.
(32) Song, F.; Tomov, S.; Dongarra, J. *Efficient Support for Matrix Computations on Heterogeneous Multi-core and Multi-GPU Architectures*; University of Tennessee Computer Science Technical Report, UT-CS-11-668; http://www.netlib.org/lapack/lawnspdf/lawn250.pdf, 2011.