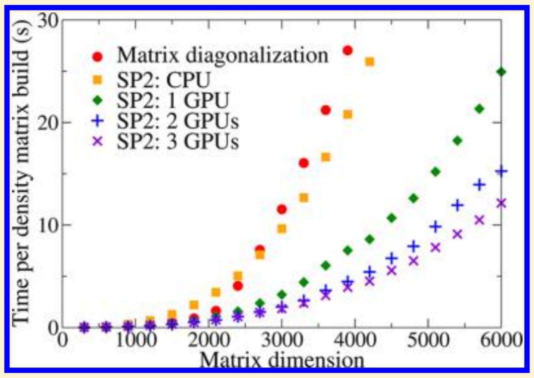
Computation of the Density Matrix in Electronic Structure Theory in Parallel on Multiple Graphics Processing Units

M. J. Cawkwell,*,[†] M. A. Wood,[†,‡] Anders M. N. Niklasson,[†] and S. M. Mniszewski[¶]

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

[‡]School of Materials Engineering, Purdue University, 701 West Stadium Avenue, West Lafayette, Indiana 47907, United States

ABSTRACT: The algorithm developed in Cawkwell, M. J. et al. *J. Chem. Theory Comput.* **2012**, *8*, 4094 for the computation of the density matrix in electronic structure theory on a graphics processing unit (GPU) using the second-order spectral projection (SP2) method [Niklasson, A. M. N. *Phys. Rev. B* **2002**, *66*, 155115] has been efficiently parallelized over multiple GPUs on a single compute node. The parallel implementation provides significant speed-ups with respect to the single GPU version with no loss of accuracy. The performance and accuracy of the parallel GPU-based algorithm is compared with the performance of the SP2 algorithm and traditional matrix diagonalization methods on a multicore central processing unit (CPU).



1. INTRODUCTION

The calculation of the density matrix, **P**, from the Kohn−Sham Hamiltonian, Fockian, or a semiempirical Hamiltonian in electronic structure theory is often a significant burden on overall computational performance, especially for large systems.[1−3] The density matrix has traditionally been computed from the eigenvalues and eigenvectors of the Hamiltonian or Fockian. While algorithms based on matrix diagonalization can be applied to metals and insulators equally well and provide a straighforward route to the introduction of a finite electronic temperature, their complexity leads to an undesirable cubic scaling, $\mathcal{O}(N^3)$, of the computation time on the number of atoms, N. Alternatives to matrix diagonalization have been developed that lead to $\mathcal{O}(N)$ performance if matrix sparsity is present and utilized.[3−6] The second-order spectral projection (SP2) method,[7] which is based on a recursive expansion of the Fermi-operator in a series of generalized matrix−matrix multiplications, is one such method. While we have demonstrated fast, low-prefactor $\mathcal{O}(N)$ applications of the SP2 algorithm in both static and molecular dynamics simulations,[4] its performance in limit of $\mathcal{O}(N^3)$ dense matrix algebra was also found to be better than that of traditional $\mathcal{O}(N^3)$ algorithms.[8]

The performance of the SP2 algorithm is controlled by the cost of the $\mathcal{O}(N^3)$ generalized matrix−matrix multiplication, $\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}$, where **A**, **B**, and **C** are matrices, and α and β are scalars. Highly optimized implementations of the BLAS level 3 DGEMM subroutine[9,10] are available for performing generalized matrix−matrix multiplications on multicore central processing units (CPUs) and many-core general purpose graphics processing units (GPUs).[11] We demonstrated that CPU and GPU implementations of the SP2 algorithm can outperform algorithms based on matrix diagonalization (with highly optimized diagonalization subroutines).[8] At the same time, the errors in the density matrices computed using the SP2 algorithm were as small or smaller than those measured in density matrices computed via diagonalization. The GPU-implementation provided by far the best performance, even in double precision arithmetic, owing to the very high floating point operation rates (FLOPs) and memory bandwidth available on these devices with respect to contemporary multicore CPUs.

Our original GPU implementation of the SP2 algorithm used the cuBLAS[12] DGEMM on a Fermi-architecture Nvidia GPU.[8] We obtained the best overall performance by porting the entire SP2 algorithm to the GPU to avoid the transfer of large arrays between the GPU and CPU at each step in the recursive expansion. Nevertheless, we found that the data transfer constitutes only a small fraction of the total compute time such that very good performance gains could be obtained with the simplest algorithm where each call to the DGEMM subroutine on the CPU is replaced by a call to the cuBLAS version on the GPU.

Compute nodes on parallel supercomputers and high-end desktop workstations can accept multiple general purpose GPUs. Hence, we have parallelized our GPU implementation of the SP2 algorithm such that the expensive generalized matrix−matrix multiplication is partitioned between multiple GPUs. The new algorithm is based on a simple blocking algorithm for the generalized matrix−matrix multiplication that still uses the highly efficient cuBLAS DGEMM as the underlying multi-

Received: September 10, 2014
Published: October 31, 2014

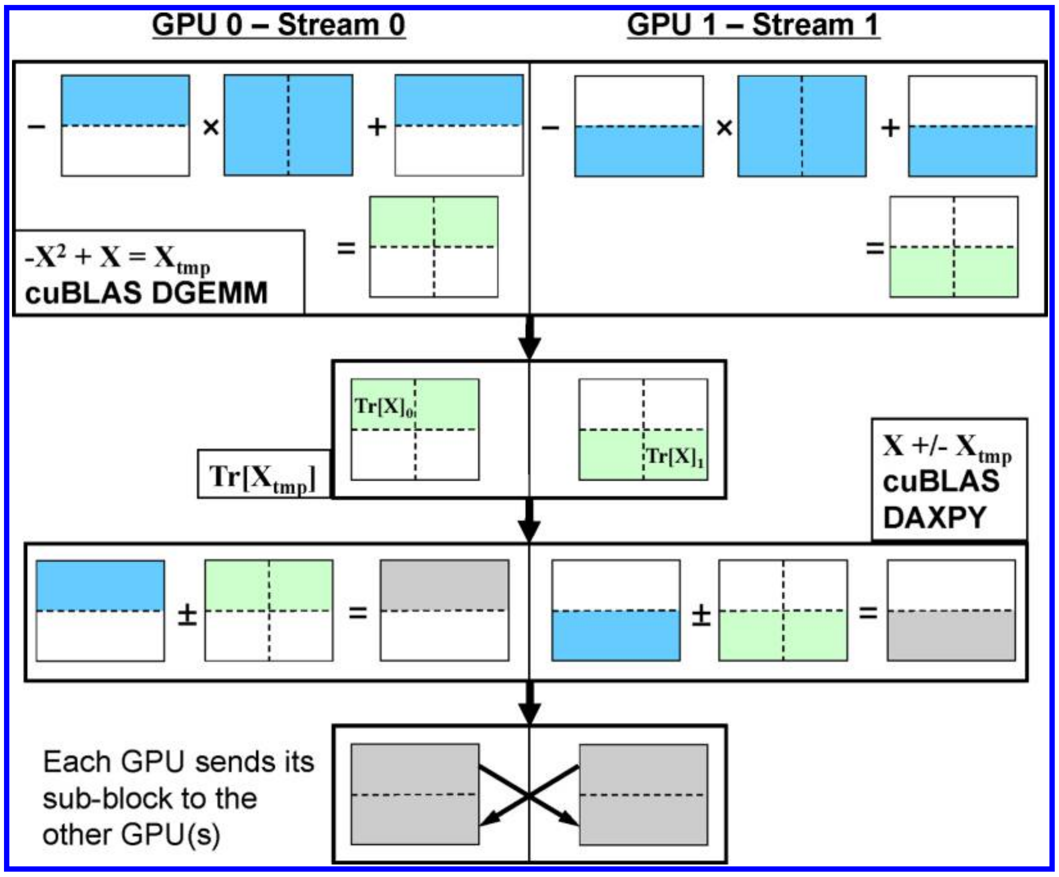


Figure 1. Schematic illustration of the parallel GPU implementation of the SP2 algorithm.

plication kernel. Although nowhere near as computationally expensive as the generalized matrix–matrix multiplication, we also compute matrix traces and matrix–matrix additions using the cuBLAS DAXPY subroutine at the sub-block level across the GPUs. This approach minimizes data transfer between the devices. We found that execution on one GPU gave the best performace for density matrices of dimensions less than about 900 × 900. Running the algorithm in parallel on two GPUs gives the best performance for density matrices greater than 900 × 900 up to about 2000 × 2000. For density matrices greater than approximately 2000 × 2000, parallel execution on three GPUs was optimal. We again found that the SP2 algorithm on both GPUs and CPUs generated density matrices with errors as small or smaller than when matrix diagonalization is applied.

In the next Section we provide a brief overview of the SP2 algorithm. Our simple blocking algorithm for the parallelization of the generalized matrix–matrix multiplication across multiple GPUs on a single compute node is described in Section 4. Timings and errors analyses from the CPU and parallel GPU implementations of the SP2 algorithm and traditional matrix diagonalization are provided in Section 5, followed by conclusions in Section 6.

2. SECOND-ORDER SPECTRAL PROJECTION METHOD

At zero electronic temperature, the Fermi–Dirac distribution becomes equivalent to the Heaviside step function, $\theta[x]$, with the step formed at the chemical potential, μ.[13] The density matrix can then be written using the matrix form of the Heaviside step function as

$$\mathbf{P} = \theta[\mu\mathbf{I} - \mathbf{H}] \tag{1}$$

where **I** is the identity matrix, and **H** is the Hamiltonian matrix obtained from density functional theory, Hartree–Fock theory, or a semiempirical method. The SP2 algorithm[7] is based on a recursive expansion of eq 1

$$\theta[\mu\mathbf{I} - \mathbf{H}] = \lim_{i\to\infty} f_i\{f_{i-1}\{...f_0\{\mathbf{X}_0\}...\}\} \tag{2}$$

where

$$\mathbf{X}_0 = \frac{\epsilon_{\max}\mathbf{I} - \mathbf{H}}{\epsilon_{\max} - \epsilon_{\min}} \tag{3}$$

and $\epsilon_{\max}$ and $\epsilon_{\min}$ are estimates for the maximum and minimum eigenvalues of the Hamiltonian, respectively. In practice, estimates for $\epsilon_{\max}$ and $\epsilon_{\min}$ can be obtained very efficiently through the Gershgorin circle theorem.[14]

At convergence, the trace of the density matrix is equal to the number of occupied states or equivalently

$$2\mathrm{Tr}[\mathbf{P}] = N_e \tag{4}$$

where Tr[**x**] denotes the trace of matrix **x**, and N_e is the total number of electrons. The recursive expansion in eq 2 uses the following projection polynomials that guarantee that eq 4 is satisfied at convergence

$$f_i[\mathbf{X}_i] = \begin{cases} \mathbf{X}_i^2 & \text{if } |2\mathrm{Tr}[X_i] - 2\mathrm{Tr}[-\mathbf{X}_i^2 + \mathbf{X}_i] - N_e| \\ & \quad \le |2\mathrm{Tr}[X_i] + 2\mathrm{Tr}[-\mathbf{X}_i^2 + \mathbf{X}_i] - N_e| \\ 2\mathbf{X}_i - \mathbf{X}_i^2 & \text{otherwise} \end{cases} \tag{5}$$

The application of the projection polynomials in eq 5 pushes the unoccupied and occupied states to 0 and 1, respectively, depending on target occupation of the density matrix. The projection polynomial at each iteration is selected to give the best occupancy in the next iteration.[8] Both of the projection polynomials take the form of a generalized matrix−matrix multiplication. Prior knowledge of the chemical potential is not required, and the number of recursive iterations scales as $\ln(1/\Delta\epsilon)$, where $\Delta\epsilon$ is the difference in energy between the highest occupied and lowest unoccupied molecular orbitals, or HOMO−LUMO gap.[7] Hence, the SP2 algorithm is efficient only when applied to insulators.

2.1. Convergence Criteria.

The recursive expansion in eq 2 is terminated when we detect that further iterations will not improve the idempotency of the density matrix, that is $\mathbf{P}^2 = \mathbf{P}$. This condition can be identified very reliably using

$$|\mathrm{Tr}[\mathbf{X}_{i-1}] - \mathrm{Tr}[\mathbf{X}_{i-2}]| \le |\mathrm{Tr}[\mathbf{X}_{i+1}] - \mathrm{Tr}[\mathbf{X}_i]| \tag{6}$$

Although a number of alternative termination criteria exist, eq 6 was employed in the implementations of the SP2 algorithm discussed in this work.

3. ALGORITHMS FOR CENTRAL PROCESSING UNITS

3.1. Matrix Diagonalization.

Optimized LAPACK and BLAS algorithms were used to the greatest possible extent in the calculation of the density matrix. The Hamiltonian matrix is diagonalized using the LAPACK routine DSYEV. Better performance can be achieved by using the divide-and-conquer version of the algorithm, DSYEVD, but we have found this to be unreliable on occasion. The outer products between the eigenvectors of the occupied states are computed using the BLAS subroutine DGER. Replacing our own OpenMP-threaded code with the DGER subroutine for computing the outer products led to significant improvements in performance with respect to our previous results presented in ref 8.

3.2. SP2 Method.

Pseudocode for the CPU implementation of the SP2 algorithm is given in Algorithm 1. The timings we report in Section 5 include the computation of $\epsilon_{\max}$ and $\epsilon_{\min}$.

Algorithm 1 Pseudo code for the CPU implementation of the SP2 algorithm

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

4. IMPLEMENTATION OF THE SP2 ALGORITHM FOR MULTIPLE GRAPHICS PROCESSING UNITS

Our parallel GPU implementation of the SP2 algorithm is based on Algorithm 3 from ref 8. This algorithm gave the best overall performance on one GPU as it minimizes data transfer between the CPU and GPU. The Hamiltonian matrix, $\epsilon_{\max}$, $\epsilon_{\min}$, and N_e are passed to each GPU at the start of the algorithm, and the converged density matrix is returned to the CPU upon completion. Only a single float, $\mathrm{Tr}[-\mathbf{X}_i^2 + \mathbf{X}_i]$, is sent to the CPU at each step in the recursive expansion in order to select which projection polynomial to apply in the subsequent step. As illustrated in Figure 1, the workload was partitioned between the multiple GPUs using CUDA streams.

We employed the simplest possible blocking scheme to partition the generalized matrix−matrix multiplication across N_{GPU} GPUs. This is illustrated schematically in Figure 1. The dimensions of the sub-blocks are kept as large as possible since GPUs yield better FLOP rates for larger problems,[8,15] that is, it is best to keep the GPUs fully occupied. Generalized matrix−matrix multiplications are performed on each GPU between the sub-block assigned to that device with each of the N_{GPU} sub-blocks of the whole array using the cuBLAS DGEMM. The sub-blocks do not need to be copied to or from temporary working arrays to perform the DGEMM since the indices for the first elements of the sub-blocks in the input and output arrays as well as their dimensions can be passed to the cuBLAS DGEMM directly.

The subsequent steps of the SP2 algorithm do not require the N_{GPU} sub-blocks of the $\mathbf{X}_{\mathrm{tmp}}$ to be reassembled. As depicted in Figure 1, we compute the trace of $\mathbf{X}_{\mathrm{tmp}}$ in parallel across the GPUs. The partial traces are summed across the GPUs and sent to the CPU. The application of the projection polynomial, $\mathbf{X}_{i+1} = \mathbf{X}_i \pm \mathbf{X}_{\mathrm{tmp}}$, can also be performed in parallel across the GPUs on each of the distributed sub-blocks. Finally, each GPU sends its sub-block of $\mathbf{X}_{i+1}$ to the other GPUs so that every device has a copy of the whole array at the start of the next iteration. The recursive expansion is terminated when the inequality in eq 6 is satisfied after which one GPU returns the density matrix to the CPU.

4.1. Padding Arrays on the GPU.

Padding the dimensions of the arrays passed to the cuBLAS DGEMM can significantly improve the performance of the calculation.[8,15] The optimal padding for the arrays depends on the array dimensions, whether single or double precision arithmetic is used, and appears to be dependent on the model of the GPU. For Nvidia C2090 GPUs we found that the best performance is obtained if the dimensions of the arrays in the serial algorithm are integer multiples of 16. The $M \times M$ arrays are allocated with dimensions $M_{\mathrm{pad}} \times M_{\mathrm{pad}}$ on the GPU where

$$M_{\mathrm{pad}} = 16N_{\mathrm{GPU}}\left(\left\lfloor \frac{M-1}{16N_{\mathrm{GPU}}} \right\rfloor + 1\right) \tag{7}$$

where $\lfloor x \rfloor$ denotes the closest integer with value $\le x$. By increasing the dimensions of the arrays in increments of $16N_{\mathrm{GPU}}$ we ensure that both dimensions of the sub-blocks in the parallel GPU implementation are also integer multiples of 16. We have also assessed the effect of padding arrays on the GPU such that M_{pad} is the smallest value that allows the workload to be partitioned equally between the N_{GPU} devices, that is

$$M_{\mathrm{pad}} = N_{\mathrm{GPU}}\left(\left\lfloor \frac{M-1}{N_{GPU}} \right\rfloor + 1\right) \tag{8}$$

The effect the padding schemes in eqs 7 and 8 have on the wall clock time for the generalized matrix−matrix multiplication on one, two, and three GPUs is depicted in Figures 2(a), (b), and (c), respectively. The timings reported in Figure 2 are an

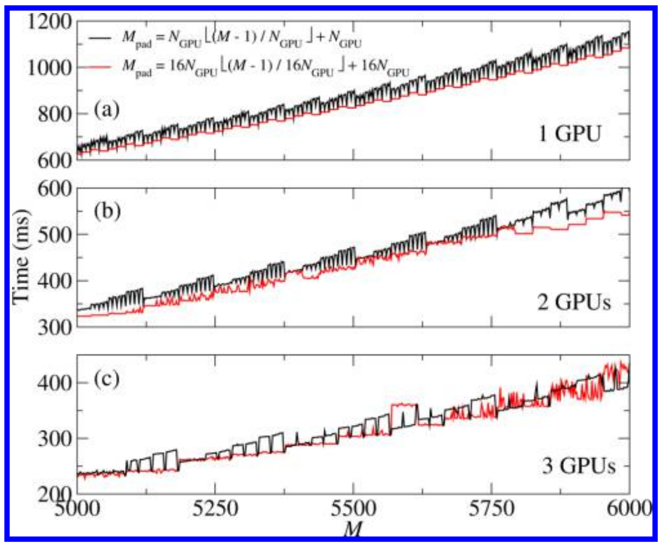


Figure 2. Effect of array padding on the wall clock time per generalized matrix–matrix multiplication. (a) 1 GPUs, (b) 2 GPUs, and (c) 3 GPUs.

average of ten generalized matrix–matrix multiplications for each M. Figures 2(a) and (b) show clear improvements in the wall time per generalized matrix–matrix multiplication on one and two GPUs using the padding scheme in eq 7. Figure 2(c) indicates that the padding scheme in eq 7 generally improves the wall time per generalized matrix–matrix multiplication although the simpler padding scheme in eq 8 that only ensures load balancing can occasionally lead to better performance.

5. ANALYSIS OF PERFORMANCE AND ERRORS

5.1. Test Systems. The performance and accuracy of the three algorithms, (i) matrix diagonalization on a CPU, (ii) the SP2 algorithm on a multicore CPU, (ii) and the parallel GPU implementation of the SP2 algorithm, were assessed with Hamiltonian matrices computed using self-consistent charge transfer tight binding theory (SC-TB), also known as density functional tight binding theory.[4,16−18] The Hamiltonian in SC-TB theory is a sum of a Slater-Koster tight binding Hamiltonian and an electrostatic potential arising from Mulliken partial charges centered on each atom. Since the Hamiltonian depends on Mulliken partial charges that are derived from the density matrix, the SC-TB equations are solved self-consistently.

Our test systems comprised boxes of 50 to 1000 water molecules at normal liquid density under periodic boundary conditions. Each oxygen atom was described by one s and 3 p orbitals, while each hydrogen atom had one s orbital. The systems from which timings were computed were snapshots from molecular dynamics simulations run at a temperature of 300 K. Since the radial distribution function, $g(R)$, for water obtained from our SC-TB parametrization corresponds well to experimental measurements,[19] as depicted in Figure 3, our test systems present a real-world challenge to the three algorithms. The HOMO−LUMO gap of liquid water from our SC-TB model is about $\Delta\epsilon$ = 5.54 eV. This value lies within the spectrum of those obtained from other methods.[20]

5.2. Timings. Timings for the matrix diagonalization and CPU implementation of the SP2 algorithm were gathered on

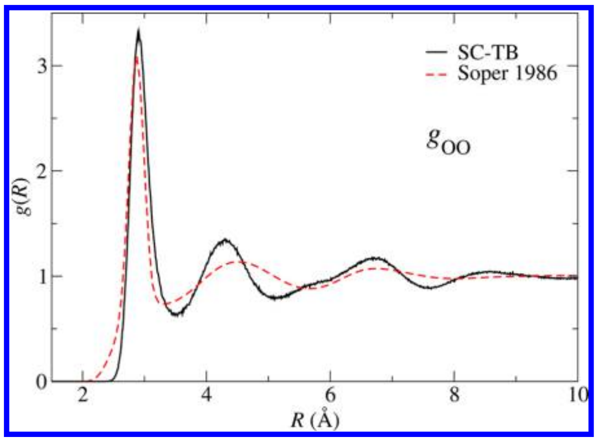


Figure 3. O−O radial distribution function for liquid water at a temperature of 300 K from a SC-TB molecular dynamics simulation and experiment.[19]

one node of the Moonlight cluster at Los Alamos National Laboratory. Each node of Moonlight is comprised of two eight-core 2.6 GHz Intel Xeon E5-2670 CPUs. The CPU code was compiled with the pgf90 compiler version 13.7 with the -fast optimization flag and the threaded ACML implementation of the LAPACK and BLAS libraries. All calculations were performed using 16 OpenMP threads. The timings for the parallel GPU algorithm, which include all data transfers between devices, were gathered on one node of the Carter supercomputer at Purdue University. Each node contained three Nvidia C2090 GPUs, and our CUDA code was compiled using the compilers and CuBLAS library distributed with the Nvidia toolkit version 5.5. A threadblock size of 1024 was used for all GPU calculations. We used two different machines to obtain timings only because the performance of the CPU-based algorithms was better on Moonlight than Carter. Similarly, Moonlight has only two Nvidia M2090 GPUs per node rather than the three available on Carter.

The time per density matrix build as a function of the matrix dimension, M, is presented on linear and logarithmic scales in Figures 4 and 5, respectively. Figure 4 shows clearly that the performance of the SP2 algorithm on a multicore CPU can exceed that of traditional matrix diagonalization even in the limit of $O(M^3)$ dense matrix algebra. The superior performance of the CuBLAS DGEMM on general purpose GPUs leads to

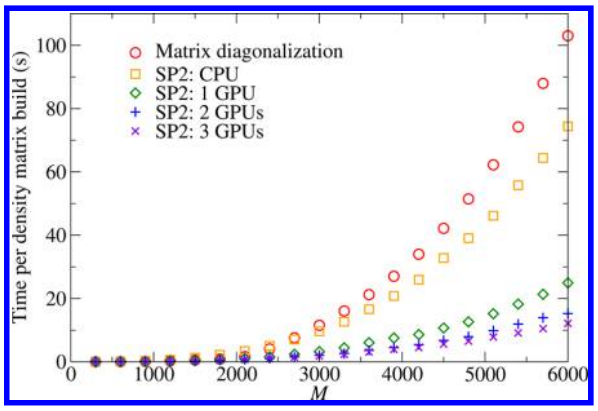


Figure 4. Time per $M \times M$ density matrix build via matrix diagonalization and the CPU and parallel GPU implementations of the SP2 algorithm.

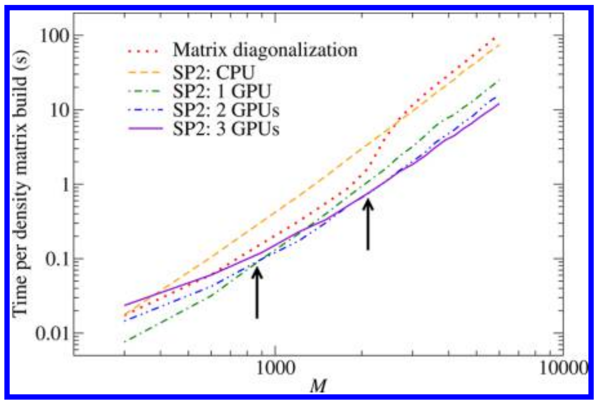


Figure 5. Time per $M \times M$ density matrix build via matrix diagonalization and the CPU and parallel GPU implementations of the SP2 algorithm on logarithmic axes. The arrows denote the matrix dimensions for which the timings for execution on one and two GPUs and two and three GPUs are approximately equal.

very significant improvements in the computational time required for each density matrix build if the problem size is large. For the 1000 water molecule test system we find that executing the SP2 algorithm in parallel on 3 GPUs yields a ×6.2 speed-up with respect to the multicore CPU implementation and a ×8.6 speed-up with respect to matrix diagonalization.

Figure 5 is informative since it depicts clearly how the performance of each algorithm depends on the dimensions of the matrices. For instance, matrix diagonalization is faster than the SP2 method on the CPU up to a matrix dimension of about $M = 2650$. However, owing to the better scaling of the DGEMM with respect to the more complex DSYEV matrix diagonalization subroutine, the SP2 method is fastest for the largest systems. Surprisingly, Figure 5 reveals that the performance of the GPU implementation of the SP2 method is superior to both matrix diagonalization and the CPU implementation of the SP2 algorithm for all problem sizes. Hence, the outstanding performance of the CuBLAS DGEMM even for the relatively small 50 molecule test case more than compensates for overhead arising from data transfers between the CPU and GPU.

Comparing the performance of the GPU implementation on one, two, and three GPUs reveals that one GPU is fastest for matrices smaller than about 900 × 900, two GPUs are fastest for intermediate problem sizes from 900 × 900 to about 2000 × 2000, whereafter execution on 3 GPUs provides the smallest wall clock time per density matrix build. The dependence of the wall clock time on the number of GPUs arises both from the communication overheads and the strong dependence of the available FLOPs on the matrix dimensions. It is important to expose enough parallelism to populate the available GPUs and allow multithreading to keep the cores busy. By parallelizing a small calculation across multiple GPUs each GPU becomes relatively inefficient and underutilized. Therefore, there exists a trade-off between the performance gains that can be achieved through parallelization and the inefficiencies that derive from giving each GPU a smaller task.

5.3. Analysis of Errors. While Figures 4 and 5 show that the SP2 algorithm enables a significant reduction in the wall clock time per density matrix build with respect to matrix diagonalization, it is important to know whether this comes at the expense of a loss of accuracy. As in ref 8 we have estimated the errors in the density matrices using measures of the idempotency

$$\varepsilon_{\text{idem}} = \| \mathbf{P}^2 - \mathbf{P} \|_2 \tag{9}$$

where $\|\mathbf{X}\|_2$ denotes the spectral norm of matrix $\mathbf{X}$, and the commutation between the density matrix and Hamiltonian

$$\varepsilon_{\text{comm}} = \|[\mathbf{H}, \mathbf{P}]\|_2 \tag{10}$$

Both $\varepsilon_{\text{idem}}$ and $\varepsilon_{\text{comm}}$ equal zero if $\mathbf{P}$ is exact.

The errors in the commutation between the $M \times M$ Hamiltonian and density matrices, $\varepsilon_{\text{comm}}$ (eq 10), are presented in Figure 6. The values of $\varepsilon_{\text{comm}}$ for density matrices computed

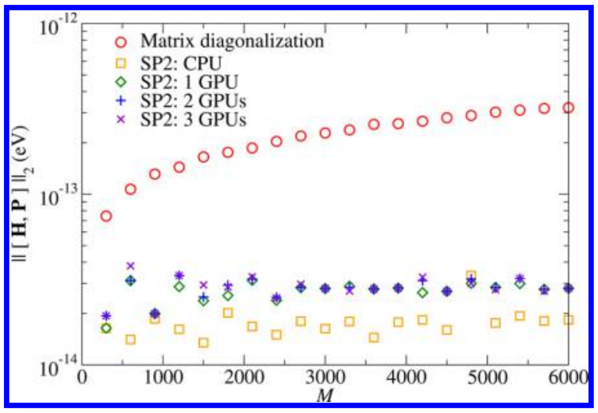


Figure 6. Error in the commutation between the Hamiltonian and density matrices computed using matrix diagonalization and the CPU and parallel GPU implementations of the SP2 method.

via matrix diagonalization are about 1 order of magnitude larger than for those computed using the SP2 method. Furthermore, the matrix diagonalization algorithm leads to errors that increase with system size, whereas for the SP2 method the errors do not depend on M. We attribute the dependence of the commutation errors obtained from the diagonalization-based algorithm on the matrix dimension to a tolerance in the LAPACK DSYEV subroutine on the orthogonality of the eigenvectors. The commutation error for the density matrices computed using the CPU implementation of the SP2 algorithm is smaller than those obtained via the parallel GPU implementation by a factor of about 1.5. The magnitude of the errors arising from the parallel GPU implementation does not depend on the number of GPUs.

The idempotency errors, $\varepsilon_{\text{idem}}$ (eq 9), for the water test systems are presented in Figure 7. Again we observe that the errors in the density matrices obtained using the CPU or parallel GPU implementations of the SP2 method are generally smaller than those obtained via matrix diagonalization, albeit with slightly greater scatter. The scatter in the idempotency errors derived from the SP2 algorithm could not be reduced by increasing the minimum number of iterations of eq 5 during the computation of $\mathbf{P}$. Hence, we attribute the scatter in $\varepsilon_{\text{idem}}$ to the somewhat stochastic nature of the systems of liquid water used in the calculations that were taken from snapshots from molecular dynamics simulations. The idempotency errors reported in ref 8 did not exhibit any scatter and were computed from idealized, fully relaxed geometries rather than from molecular dynamics snapshots. As with $\varepsilon_{\text{comm}}$, the idempotency errors measured in the density matrices computed by matrix diagonalization increase with system size, whereas the errors measured in the density matrices computed using the

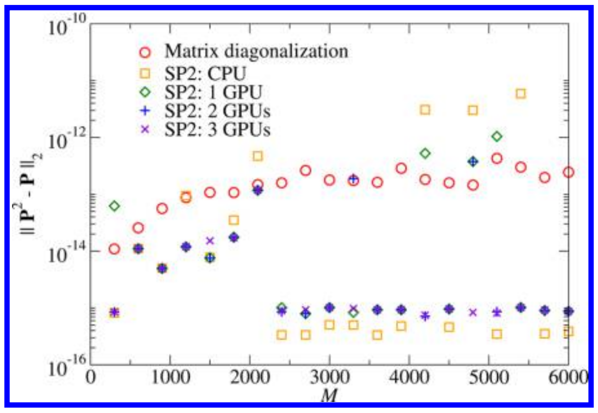


Figure 7. Idempotency error, $\varepsilon_{\mathrm{idem}}$ for the $M \times M$ density matrices computed using matrix diagonalization and the CPU and parallel GPU implementations of the SP2 algorithm.

SP2 method do not exhibit any systematic dependence on M. Taking Figures 6 and 7 together, it is evident that one does not have to compromise accuracy for performance when adopting the SP2 method.

6. CONCLUSIONS

The SP2 method for computing the density matrix in electronic structure theory at zero electronic temperature was originally envisioned for use with linear scaling solvers. The performance of the method in the $O(N^3)$ limit in dense matrix algebra can clearly exceed that of traditional algorithms based on matrix diagonalization. The outstanding performance of the SP2 method can be attributed to the highly optimized implementations of the level 3 BLAS DGEMM generalized matrix−matrix multiplication subroutines that are available for both CPUs and general purpose GPUs.

Compute nodes containing multiple GPUs are nowadays neither uncommon nor prohibitively expensive. We have developed a simple and effective blocking scheme to parallelize the DGEMM, matrix trace, and DAXPY vector−vector addition operations across multiple GPUs on a single compute node. Our parallel GPU implementations of the SP2 algorithm yield significant speed-ups with respect to execution on a single GPU for large problems and are almost ×9 faster than matrix diagonalization on a contemporary multicore CPU. Owing to overheads arising from data transfer between the CPU and GPU(s) and the dependence of the FLOP rate on problem size seen on GPUs, the best performance for small systems ($M <$ 900) is achieved by running on one GPU. Intermediate sized problems with $900 < M < 2000$ run fastest on two GPUs, and problems with $M > 2000$ run fastest on three GPUs.

The SP2 method yields density matrices with errors that are as small or smaller than those obtained by matrix diagonalization. Hence, the application of the SP2 alogorithm in electronic structure calculations can provide both significant improvements in wall clock time and accuracy, provided the system has a gap at the chemical potential.

AUTHOR INFORMATION

Corresponding Author

*E-mail: cawkwell@lanl.gov.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

We thank Ed Sanville, Nick Bock, Kyle Spafford, and Jeffrey Vetter for their contributions and valuable discussions. This work was supported by the Laboratory Directed Research and Development program of Los Alamos National Laboratory. Computing resources were provided in part by the Los Alamos National Laboratory Institutional Computing Program, which is supported by the U.S. Department of Energy National Nuclear Security Administration.

REFERENCES

(1) Goedecker, S. *Rev. Mod. Phys.* **1999**, *71*, 1085−1123.
(2) Finnis, M. *Interatomic Forces in Condensed Matter*; Oxford University Press: 2003.
(3) Bowler, D. R.; Miyazaki, T. *Rep. Prog. Phys.* **2012**, *75*, 036503.
(4) Cawkwell, M. J.; Niklasson, A. M. N. *J. Chem. Phys.* **2012**, *137*, 134105.
(5) Millam, J. M.; Scuseria, G. E. *J. Chem. Phys.* **1997**, *106*, 5569−5577.
(6) Daniels, A. D.; Scuseria, G. E. *J. Chem. Phys.* **1999**, *110*, 1321−1328.
(7) Niklasson, A. M. N. *Phys. Rev. B* **2002**, *66*, 155115.
(8) Cawkwell, M. J.; Sanville, E. J.; Mniszewski, S. M.; Niklasson, A. M. N. *J. Chem. Theory Comput.* **2012**, *8*, 4094.
(9) http://www.netlib.org/blas (accessed Oct 11, 2014).
(10) Dongarra, J.; Du Croz, J.; Hammarling, S.; Duff, I. *ACM Trans. Math. Software* **1990**, *16*, 1−17.
(11) Leang, S. S.; Rendell, A. P.; Gordon, M. S. *J. Chem. Theory Comput.* **2014**, *10*, 908.
(12) http://docs.nvidia.com/cuda/cublas (accessed Oct 11, 2014).
(13) McWeeny, R. *Proc. R. Soc. London, Ser. A* **1956**, *235*, 496.
(14) Golub, G.; van Loan, C. F. *Matrix Computations*; Johns Hopkins University Press: Baltimore, 1996; p 200.
(15) 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; 2011.
(16) Finnis, M. W.; Paxton, A. T.; Methfessel, M.; van Schilfgaarde, M. *Phys. Rev. Lett.* **1998**, *81*, 5149.
(17) Elstner, M.; Porezag, D.; Jungnickel, G.; Elsner, J.; Haugk, M.; Frauenheim, T.; Suhai, S.; Seifert, G. *Phys. Rev. B* **1998**, *58*, 7260.
(18) Frauenheim, T.; Seifert, G.; Elstner, M.; Hajnal, Z.; Jungnickel, G.; Porezag, D.; Suhai, S.; Scholz, R. *Phys. Status Solidi B* **2000**, *217*, 41.
(19) Soper, A. K.; Phillips, M. G. *Chem. Phys.* **1986**, *107*, 47.
(20) Cabral do Couto, P.; Estácio, S. G.; Costa Cabral, B. J. *J. Chem. Phys.* **2005**, *123*, 054510.