JCTC Journal of Chemical Theory and Computation

Article

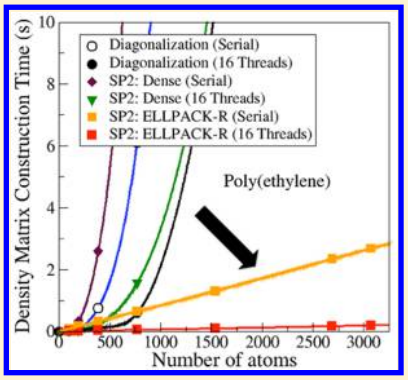
pubs.acs.org/JCTC

Efficient Parallel Linear Scaling Construction of the Density Matrix for Born−Oppenheimer Molecular Dynamics

S. M. Mniszewski,*[,†] M. J. Cawkwell,*[,‡] M. E. Wall,[†] J. Mohd-Yusof,[†] N. Bock,[‡] T. C. Germann,[‡] and A. M. N. Niklasson*[,‡]

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

ABSTRACT: We present an algorithm for the calculation of the density matrix that for insulators scales linearly with system size and parallelizes efficiently on multicore, shared memory platforms with small and controllable numerical errors. The algorithm is based on an implementation of the second-order spectral projection (SP2) algorithm [Niklasson, A. M. N. *Phys. Rev. B* **2002**, *66*, 155115] in sparse matrix algebra with the ELLPACK-R data format. We illustrate the performance of the algorithm within self-consistent tight binding theory by total energy calculations of gas phase poly(ethylene) molecules and periodic liquid water systems containing up to 15,000 atoms on up to 16 CPU cores. We consider algorithm-specific performance aspects, such as local vs nonlocal memory access and the degree of matrix sparsity. Comparisons to sparse matrix algebra implementations using off-the-shelf libraries on multicore CPUs, graphics processing units (GPUs), and the Intel many integrated core (MIC) architecture are also presented. The accuracy and stability of the algorithm are illustrated with long duration Born−Oppenheimer molecular dynamics simulations of 1000 water molecules and a 303 atom Trp cage protein solvated by 2682 water molecules.



1. INTRODUCTION

A major obstacle in Kohn−Sham density functional theory[1,2] is the computational complexity of the quantum mechanical eigenvalue problem whereby the computational effort scales with the cube, $O(N^3)$, of the number of atoms, N. Alternative methods have therefore been developed that avoid matrix diagonalization in the solution of the eigenvalue problem. For example, instead of calculating all of the individual eigenfunctions and eigenvalues, the single particle density matrix or the Green's function is constructed using approximate sparse matrix algebra. With these methods the computational cost of calculating the density matrix can often be reduced to scale linearly, $O(N)$, with system size.[3,4] However, linear scaling techniques often introduce a significant overhead such that for smaller systems, up to a few hundred atoms, it is often faster to use conventional $O(N^3)$ methods. Nevertheless, thanks to linear scaling electronic structure theory it is possible to calculate the electronic ground state of large systems with millions of atoms,[5,6] though such calculations require hours of computation using massive parallelism with thousands of processors. Recent results, however, demonstrate a significant speedup of these methods,[7−9] which looks very promising for practical applications in the near future. In this article we focus on intermediate sized problems containing on the order of 10^3 to 10^4 atoms. These systems are in general too large to treat with conventional $O(N^3)$ methods. On the other hand, because of the computational overhead, they are also expensive to study with existing linear scaling techniques. This is a particular challenge in molecular dynamics simulations, which typically are performed over tens to hundreds of thousands of time steps. Currently electronic structure-based Born−Oppenheimer molecular dynamics simulations are not feasible for this range of system sizes. To overcome this hurdle we present a $O(N)$ algorithm for the construction of the density matrix based on a new implementation of the second-order spectral projection algorithm (SP2) in sparse matrix algebra that exhibits excellent parallel scaling on shared memory, multicore platforms. The performance of the algorithm is such that it enables practical and numerically accurate Born−Oppenheimer molecular dynamics simulations for thousands of atoms running on a single multicore CPU node over tens to hundreds of thousands of time steps within self-consistent density functional tight binding theory.[10−12]

2. SECOND-ORDER SPECTRAL PROJECTION ALGORITHM

The single particle density matrix, $\mathbf{P}$, plays an important role in Kohn−Sham density functional theory,[13,14] Hartree−Fock theory,[15−17] and semiempirical models derived from them. The density matrix is traditionally constructed as an outer product of the eigenvectors, $\mathbf{c}$, of the occupied states of the orthogonalized $M \times M$ Kohn−Sham Hamiltonian or Fockian, that is

$$\mathbf{P} = \sum_{i=1}^{M} f(\epsilon_i - \mu)\mathbf{c}_i\mathbf{c}_i^* \tag{1}$$

Received: June 12, 2015
Published: September 14, 2015

DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654

where ϵ are the corresponding eigenvalues, $f(x)$ is the Fermi−Dirac distribution, and μ is the chemical potential defined such that $2\mathrm{Tr}[\mathbf{P}] = N_e$, where N_e is the total number of electrons. While robust and general, the diagonalization of the effective single particle Hamiltonian gives rise to an undesirable $O(N^3)$ scaling of the computational time on the number of atoms. Hence, the construction of the density matrix via traditional matrix diagonalization can severely limit problem sizes, especially for semiempirical electronic structure methods where the construction of the effective single particle Hamiltonian is cheap relative to *ab initio* methods.

Numerous alternatives to matrix diagonalization exist for the computation of the density matrix in electronic structure theory.[3] Here we focus on the second-order spectral projection (SP2) algorithm of Niklasson[18] that we recently have demonstrated can provide better performance than matrix diagonalization both in $O(N^3)$ dense and $O(N)$ sparse matrix algebra provided the system has a nonvanishing gap at the chemical potential.[19] The SP2 algorithm is based on a recursive expansion of the Fermi operator at zero electronic temperature where it becomes equivalent to the matrix form of the Heaviside step function,

$$\mathbf{P} = \theta[\mu\mathbf{I} - \mathbf{H}] \tag{2}$$

Here $\mathbf{I}$ is the identity matrix and $\mathbf{H}$ is the effective single particle Hamiltonian in an orthogonal matrix representation. The recursive expansion

$$\theta[\mu\mathbf{I} - \mathbf{H}] = \lim_{i\to\infty} f_i(f_{i-1}(\ldots f_0(\mathbf{X}_0)\ldots)) \tag{3}$$

commences from the matrix $\mathbf{X}_0$ which is equal to the Hamiltonian rescaled such that all of its eigenvalues occupy the interval $[0,1]$ in reverse order, that is

$$\mathbf{X}_0 = \frac{\epsilon_{\max}\mathbf{I} - \mathbf{H}}{\epsilon_{\max} - \epsilon_{\min}} \tag{4}$$

Estimates for the maximum and minimum eigenvalues of $\mathbf{H}$, $\epsilon_{\max}$, and $\epsilon_{\min}$, respectively, can be obtained using, for example, the Gershgorin circle theorem.[20] The recursive SP2 expansion uses a sequence of the following projection polynomials.

$$f_i(\mathbf{X}_i) = \begin{cases} \mathbf{X}_i^2 \\ 2\mathbf{X}_i - \mathbf{X}_i^2 \end{cases} \tag{5}$$

The sequence is chosen to ensure that $2\mathrm{Tr}[\mathbf{P}] = N_e$ at convergence by projecting the occupied states toward 1 and the unoccupied states to 0. A simple and efficient criterion to select which polynomial to apply is to choose the function that minimizes the occupation error after each iteration, i.e. $|2\mathrm{Tr}[f_i(\mathbf{X}_i)] - N_e|$.

Each step in the recursive expansion requires one generalized matrix−matrix multiplication, $\mathbf{C} = \alpha\mathbf{A}\mathbf{B} + \beta\mathbf{C}$, where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are matrices and α and β are scalars, and storage for one extra temporary matrix. Pseudocode for the SP2 algorithm is presented in Algorithm 1. The performance of the SP2 algorithm is controlled by the wall-clock time required to compute the generalized matrix−matrix multiplication. This step can often be performed at close to peak performance on multicore architectures. Indeed, it was recently demonstrated that porting the generalized matrix−matrix multiplication to high performance, general purpose graphics processing units (GPUs) can yield almost ×10 speedups with respect to execution on contemporary, multicore CPUs in dense matrix algebra owing to extraction of parallelism and the high memory bandwidth of these devices.[19,21] Apart from the SP2 algorithm used in this study there are a number of alternative recursive Fermi-operator expansion methods[16,22−29] that can be used. These alternative methods are based on higher-order spectral projections that need more intermediate memory storage, require a higher number of matrix−matrix multiplications to reach convergence, or assume prior knowledge of the HOMO−LUMO gap.

Algorithm 1 Second-order spectral projection (SP2)

```
Estimate ε_max and ε_min
X = (ε_max I − H)/(ε_max − ε_min)
TraceX = Tr[X]
for i = 1 : i_max do
    TraceXold = TraceX
    X_tmp = X^2
    TraceXtmp = Tr[X_tmp]
    if |TraceXtmp − N_occ| − |2TraceX − TraceXtmp − N_occ| > IdemTol then
        X = 2X − X_tmp
        TraceX = 2TraceX − TraceXtmp
    else
        X = X_tmp
        TraceX = TraceXtmp
    end if
    IdemErr_i = |TraceX − TraceXold|
    if IdemErr_{i−2} ≤ IdemErr_i and i > i_min then
        break
    end if
end for
P = X
```

Since for nonmetallic materials the real-space representation of the density matrix, $\mathbf{P}(\mathbf{r},\mathbf{r}')$, decays exponentially with increasing $|\mathbf{r}-\mathbf{r}'|$, sufficiently large density matrices are sparse.[4] Hence, using a sparse representation for the $\mathbf{X}_i$ matrices in eq 3 and a sparse matrix−matrix multiplication kernel such as the Gustavson algorithm,[30,31] one can recast the SP2 method as an exact (that is approximation-free) $O(N)$ solver. In practice, each step in the recursive expansion reduces the matrix sparsity (more matrix elements are nonzero) such that the wall-clock time for the sparse matrix−matrix multiplication increases. However, on inspection much of this fill-in of the matrices (loss of sparsity) is seen to arise from the incorporation of matrix elements of very small magnitude. Earlier works showed that by removing matrix elements or entire sub-blocks with an absolute value less than a user-defined tunable tolerance, τ, at each step in the recursive expansion, that high levels of matrix sparsity are maintained throughout the algorithm, leading to greatly improved $O(N)$ performance.[22,24,32,33] The resulting errors in the density matrix are small and controlled by the magnitude of the numerical threshold, τ. Matrix sparsity can also be achieved by assuming a finite range of the electronic overlap, which can be combined with variational formulations of the density matrix.[34−36] Introducing matrix sparsity with a finite overlap radius has then the advantage of providing, at least in principle, a variational formulation of a sparse density matrix but for a constrained functional. Problems, however, may occur for molecular dynamics simulations of inhomogeneous systems where colliding molecules and localizations or dislocalizations of the electronic states makes an interaction radius hard to estimate without prior expert knowledge.

2.1. A Parallel, $O(N)$ Implementation of the SP2 Algorithm. The relative performance of algorithms for the computation of the density matrix as a function of system size via matrix diagonalization, the SP2 algorithm in dense matrix algebra, and a serial (single core) implementation of the SP2 algorithm in thresholded sparse matrix algebra using the compressed sparse row format (CSR) with the scalar Gustavson algorithm are presented in Figure 1. The initial single particle

DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654

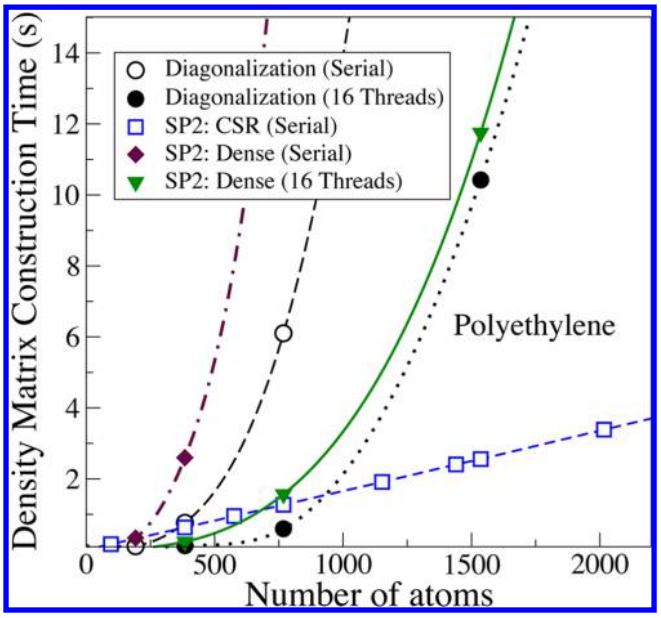


Figure 1. The total wall-clock time for calculating the density matrix using LAPACK diagonalization in the SP2 scheme, Algorithm 1, in comparison to using the compressed sparse row (CSR) format and the Gustavson algorithm for sparse matrix−matrix multiplication as well as using dense matrix−matrix algebra. The CSR calculations were performed in serial without parallelism, whereas the diagonalizations and the dense matrix−matrix multiplications were performed either with or without (shared memory) parallelism. A numerical threshold of $\tau = 10^{-5}$ was used after each matrix operation.

Hamiltonian matrices were generated using a self-consistent tight binding (also known as density functional tight binding) model for hydrocarbons. Figure 1 clearly illustrates that a conventional $O(N^3)$ diagonalization algorithm using the threaded Linear Algebra PACKage (LAPACK)[37] and Basic Linear Algebra Subprograms (BLAS)[38] perform better than the serial CSR-based $O(N)$ algorithm for system sizes up to nearly 1000 atoms. Hence, even if the onset of linear scaling may start already for much smaller systems, the serial CSR-based $O(N)$ implementation of the SP2 algorithm is typically of little practical use in molecular dynamics simulations for systems with less than 1000 atoms owing to its large computational prefactor with respect to optimized, parallel $O(N^3)$ conventional algorithms. The serial CSR-based $O(N)$ implementation of the SP2 algorithm shows good performance compared to conventional $O(N^3)$ diagonalization running in serial on a single CPU core. However, owing to the complexity of the CSR data format and the Gustavson algorithm, improving its performance by parallelization over all cores on a shared-memory node is nontrivial. Several alternative schemes and data formats that are tailored for very large sparse matrix−matrix multiplications have been recently developed for calculations on massively parallel supercomputers with distributed memory.[5,39−44] Here we will instead focus on problems with intermediate sizes (10^3 to 10^4 atoms) that are amenable to single-node, multicore, shared memory architectures.

The parallelization of the arithmetic operations on sparse matrices in the SP2 algorithm was accomplished by first replacing the CSR format with the ELLPACK-R[45] format. The ELLPACK-R (ELL) format is illustrated in Figure 2. While less compact than the CSR format, in particular for inhomogeneous sparsity patterns, it provides better opportunities for parallelization. Its variants have recently been shown to perform well for sparse matrix-vector multiplication on GPUs.[45] The ELLPACK-R format represents a sparse matrix using three arrays: a two-dimensional real array containing the numerical values, a two-dimensional integer array containing the column indices, and an integer vector containing the number of nonzero entries per row. The row-wise data storage makes a parallel implementation of a sparse matrix−matrix multiplication relatively straightforward in comparison to the CSR format, and it gives rise to good parallel efficiency.

A sparse matrix−matrix multiplication in ELLPACK-R format, for $\mathbf{X}^2$, is illustrated in Figure 3. The matrix elements of each row, i, of $\mathbf{X}$ are multiplied by the corresponding column vector elements. However, since the Hamiltonian is symmetric, $\mathbf{X}$ is also symmetric, and we therefore multiply by the corresponding rows. These operations are accumulated into a temporary buffer, denoted RowBuf_i in Figure 3. The temporary row buffer represents a new row, i, of $\mathbf{X}^2$, which is stored in the ELLPACK-R format. Since the computation of each row of $\mathbf{X}^2$ is independent of all of the others, this algorithm can be parallelized over rows easily on a shared memory, multicore node where each thread (or core) operates on a group of one or more rows. Pseudocode for a sparse matrix−matrix multiplication using this approach is presented in the Appendix. The matrix additions and traces in the SP2 algorithm are also computed row-by-row using the ELLPACK-R format.

The parallelization of the sparse matrix operations in the SP2 algorithm over shared memory cores uses a threaded programming model with the Open Multiprocessing (OpenMP)[46] API. OpenMP consists of a set of compiler directives, library

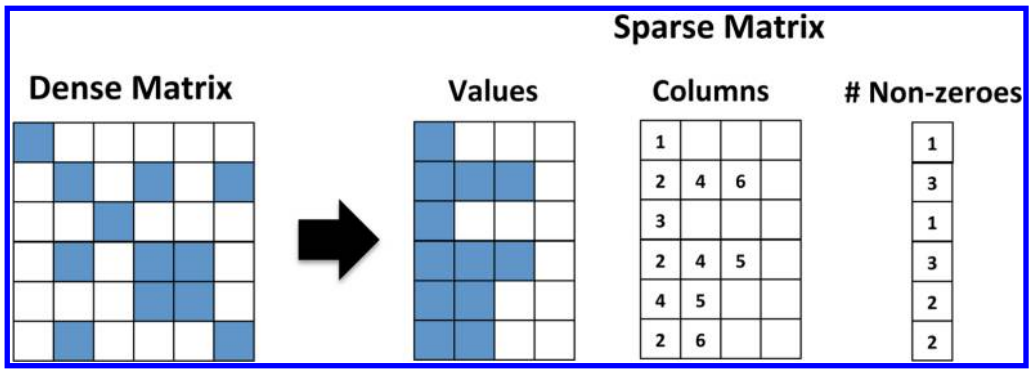


Figure 2. Sparse matrix ELLPACK-R representation consists of a 2-D array of the nonzero values in each row, a 2-D array of the column indices, and a vector of the number of nonzeroes in each row.

DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654

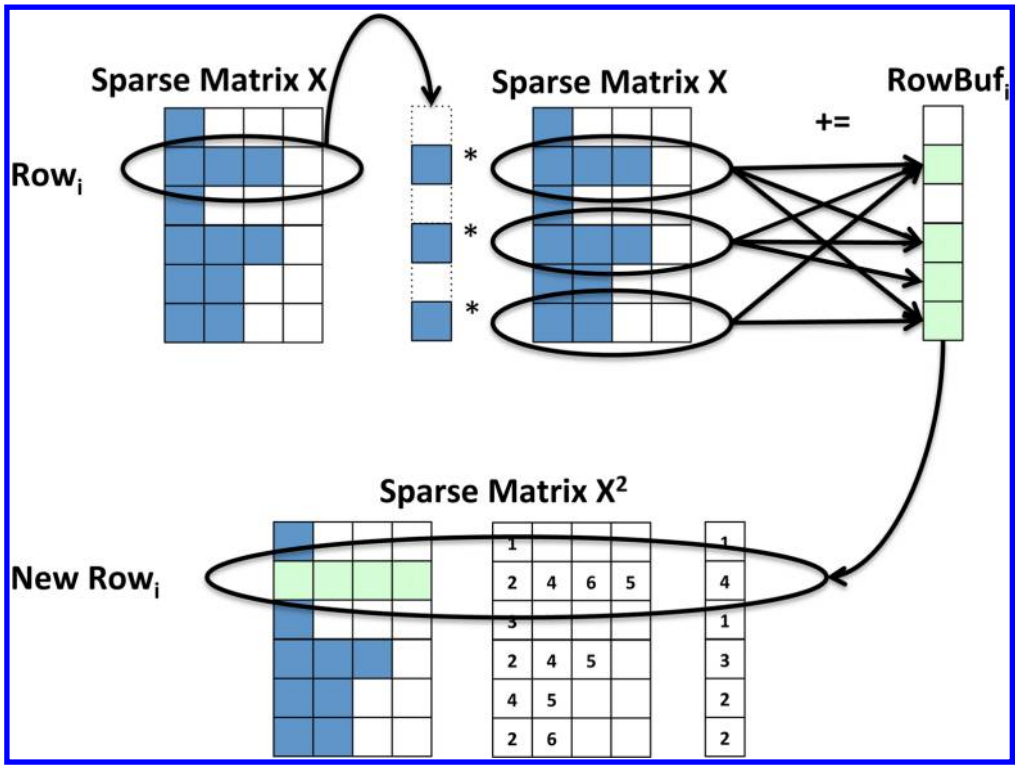


Figure 3. Sparse matrix−matrix multiplication is performed in a row-wise fashion. Each nonzero value in a row is multiplied by the nonzero values corresponding to the column index. These values are summed and stored in a temporary row buffer based on their column indices. The row values are thresholded and added to the resulting X^2 matrix in the ELLPACK-R.

routines, and environment variables that are used to influence run-time behavior. Work sharing constructs are used to divide a task among threads so that each thread executes its allocated part of the code such that both task and data parallelism can be achieved.

3. RESULTS

3.1. Performance Analysis. There are several facets to the performance of the $O(N)$ SP2 algorithm in ELLPACK-R format. We first consider the wall-clock time and speedup for the construction of density matrices for gas-phase poly(ethylene) molecules and periodic cells of liquid water as a function of the total number of atoms. We then explore the effects of local vs nonlocal memory access and the effects of matrix sparsity using the poly(ethylene) systems. We compare the performance of our implementation of the $O(N)$ algorithm (ELLPACK-R, OpenMP) on a multicore CPU with an implementation on Nvidia graphics processing units (GPUs) as well as an Intel many integrated core (MIC) architecture. The implementations of the SP2 algorithm for GPUs and MICs used off-the-shelf sparse matrix algebra libraries in CSR format, the CUDA Sparse Matrix Library (cuSPARSE)[47] for Nvidia GPUs, and the Intel Math Kernel Library (MKL)[48] on CPUs and Intel MICs. We conclude with a comparison of the different implementations.

All calculations were performed starting from Hamiltonian matrices computed using self-consistent tight binding theory[10−12,49,50] as implemented in the electronic structure code LATTE.[51] Since the elements of the Hamiltonian matrix in self-consistent tight binding theory are parametrized, it is 3−4 orders of magnitude faster than density functional theory calculations on identical hardware even in the limit of $O(N^3)$ scaling. Hence, it is ideal for assessing the scaling of our $O(N)$ density matrix build to large systems as well as its application in large-scale, long-duration molecular dynamics simulations where *ab initio* methods would simply be too slow.

3.1.1. Multicore CPU. The timings for the parallel $O(N)$ SP2 algorithm for multicore CPUs were obtained using the Portland Group Fortran90 compiler version 13.10 with the "-fast" optimization flag and its OpenMP implementation. OpenMP static scheduling was used unless specified otherwise whereby chunks are split evenly between the available threads. All timings are taken from a node of the "Moonlight" cluster at LANL that comprises two eight-core Intel Xeon E5-2670 CPUs running at 2.6 GHz.

In Figure 4 we present the wall-clock time per density matrix build for isolated poly(ethylene) molecules containing from 96 to 5472 atoms. The $O(N^3)$ scaling of traditional matrix diagonalization algorithms is clearly evident although the use of the optimized, threaded LAPACK[37] and BLAS[38] libraries does improve performance. The performance of our original, single-core $O(N)$ implementation of the SP2 algorithm in the CSR format is also presented in Figure 4. This earlier implementation used a symbolic premultiplication step in order to (re)allocate the CSR arrays before the real multiplication step. A numerical threshold of $\tau = 10^{-5}$ on small matrix elements was applied before forming $\mathbf{X}_{i+1}$, that is, after the sparse matrix−matrix addition step in Algorithm 1. The performance of the parallel $O(N)$ SP2 algorithm in the ELLPACK-R format is significantly better than that of our original implementation even on one CPU core. The reasons for the improved performance include (i) the replacement of the symbolic premultiplication step with a precalculated (conservative) estimate for the maximum number of nonzero elements per row, and (ii) slightly higher levels of

DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654

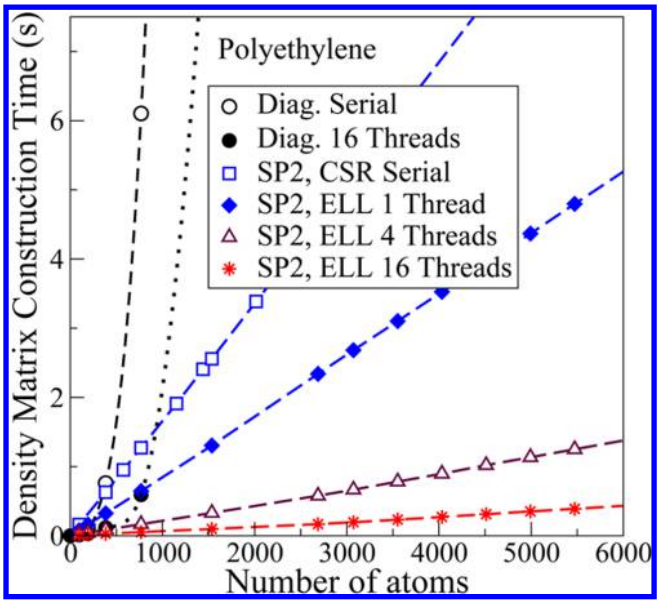


Figure 4. The total wall-clock time for calculating the density matrix for poly(ethylene) chains using regular LAPACK diagonalization or the SP2 scheme, Algorithm 1, with the compressed sparse row format and the Gustavson algorithm for serial sparse matrix−matrix multiplications in comparison to the threaded parallel SP2 scheme, Algorithm 1, using the sparse ELLPACK-R (ELL) matrix format. A numerical threshold of $\tau = 10^{-5}$ was used after each matrix operation. Diagonalization speedup increases by a factor of 4 from 1 to 16 threads. SP2 ELLPACK-R speedup increases by a factor of 4 from 1 to 4 threads and by nearly 14 from 1 to 16 threads.

matrix sparsity are maintained by applying a threshold to small matrix elements after both the sparse matrix−matrix addition and multiplication steps. Finally, Figure 4 illustrates clearly the excellent parallel scaling of the SP2 algorithm in ELLPACK-R as the number of CPU cores (threads) is increased. Increasing the number of cores from 1 to 16 gives a speedup of a nearly ideal factor of 14. We find a speedup of a factor of 24 on going from our original serial CSR implementation to the parallel implementation in ELLPACK-R on 16 cores.

The time per density matrix build using the $O(N)$ SP2 algorithm with the ELLPACK-R format for periodic simulation cells of liquid water containing from 250 to 5000 molecules with $\tau = 10^{-5}$ is presented in Figure 5. We again observe the onset of $O(N)$ scaling even for small systems, but in this case we obtain a speedup of only 6 upon increasing the number of threads from 1 to 16. Nevertheless, the time per density matrix build is smaller for liquid water than for poly(ethylene) molecules containing the same number of atoms (keeping the number of cores and τ fixed) since the density matrices for quasi-zero-dimensional water are more sparse than those for one-dimensional poly(ethylene).

3.1.2. Data Locality and Memory Access. We observed a close-to-ideal speedup of 14 for the $O(N)$ SP2 algorithm for poly(ethylene) as the number of cores was increased from 1 to 16, whereas calculations for liquid water, while faster overall, exhibited less than ideal parallel scaling. While a reduced workload and memory/cache contention for the water system most probably contribute to the reduction in parallel scaling, the origin of this behavior also lies with the distribution of the nonzero matrix elements in the Hamiltonian and density matrices. The density matrix for poly(ethylene) is highly structured and banded about the leading diagonal when atoms are numbered

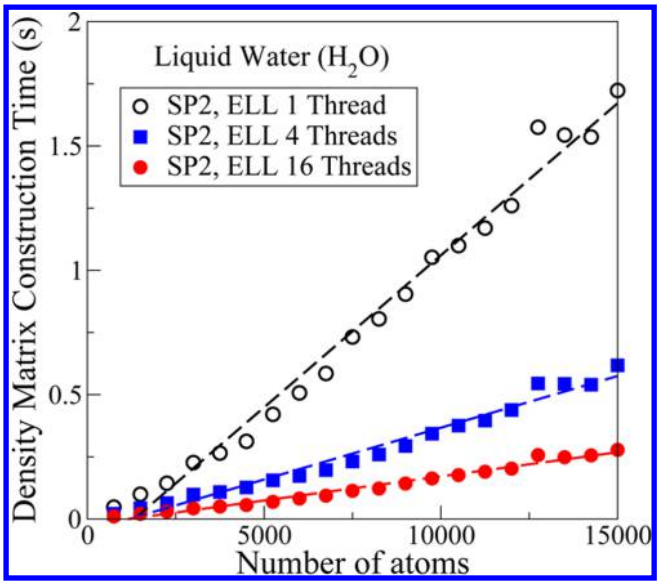


Figure 5. The total wall-clock time for calculating the density matrix for water systems of increasing size using the threaded parallel SP2 scheme, Algorithm 1, and the sparse ELLPACK-R (ELL) matrix format. The dashed lines are guides to the eye that are given from linear interpolations. A numerical threshold of $\tau = 10^{-5}$ was used after each matrix−matrix operation.

contiguously along the chain. On the other hand, for liquid water, the nonzero elements are distributed randomly off the leading diagonal in 6 × 6 dense blocks owing to its intrinsic disorder. Hence, the efficiency and scalability of the ELLPACK-R implementation of the SP2 algorithm depends on both the level of matrix sparsity and the degree of data locality.

The effect of data locality and the deleterious role of nonlocal memory access are illustrated in Figure 6 where the time per density matrix build for poly(ethylene) molecules is presented

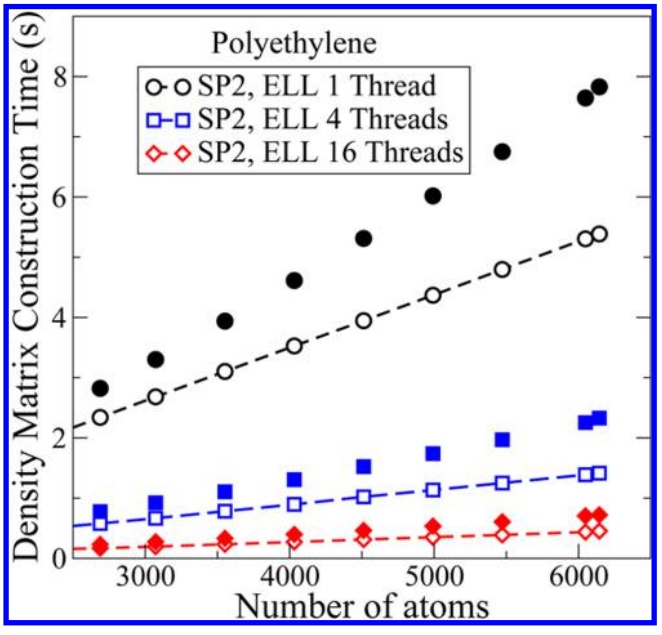


Figure 6. The total wall-clock time for calculating the density matrix for poly(ethylene) chains using the threaded SP2 scheme, Algorithm 1, with the sparse ELLPACK-R (ELL) matrix format. The open symbols correspond to the ordered matrices, whereas the filled unconnected symbols show the timings for randomly disordered matrices.

DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654

for systems where the atoms are ordered either contiguously or randomly along the chain. By randomly reshuffling the atoms the local memory access is reduced and the overall performance decreases by about 40% with respect to ordered, banded matrices. Thus, reordering the matrices of liquids to improve data locality via techniques such as space filling curves and bandwidth minimization may improve performance.

The performance of the parallel, sparse matrix−matrix multiplication in ELLPACK-R format relative to that of an optimized implementation of the BLAS level 3 Double-precision GEneral Matrix−Matrix multiplication (DGEMM) subroutine in dense matrix algebra is depicted in Figure 7. Here we took the converged

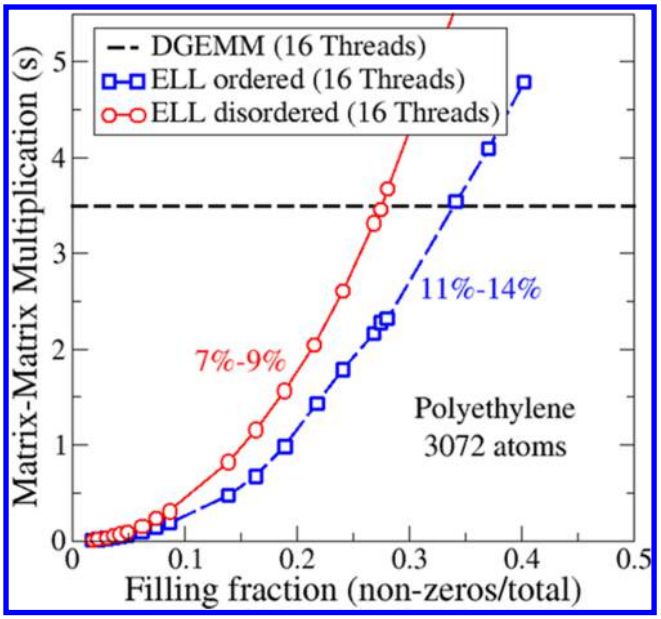


Figure 7. The total wall-clock time for squaring an ordered and disordered matrix as a function of filling fraction for a 3072 atom system of poly(ethylene). The constant dashed line gives the time for the corresponding dense threaded BLAS DGEMM calculation, which is close to peak performance. Compared to a BLAS DGEMM, the ELLPACK-R (ELL) ordered matrix calculation is 11%−14% efficient in GFLOP/s vs 7%−9% for disordered. The numerical tolerances corresponding to the filling fraction for ELL ordered are 1e-05 to 1e-65, while for disordered they are 1e-05 to 1e-46.

density matrix, **P**, of a 3072 atom poly(ethylene) chain and applied a numerical threshold to generate filling fractions (fraction of matrix elements that are nonzero) in the range 0.01 to 0.4. We found that if the matrix is ordered and hence banded about the leading diagonal, our sparse matrix−matrix multiply is faster than the DGEMM subroutine if the filling fraction is less than about 0.34. If the matrices are disordered to increase nonlocal memory access, then our sparse matrix−matrix multiply is faster than the DGEMM for filling fractions less than 0.27. For the higher filling factors shown here the best performance was obtained using OpenMP dynamic scheduling with a chunk size of 1, while static scheduling is sufficient for lower filling fractions. The ratio of floating point operations (FLOPS) per second for the ELLPACK-R sparse matrix−matrix multiplication compared to that of the BLAS DGEMM is the relative FLOP rate. The relative FLOP rate reaches 11%−14% for the ordered case and 7%−9% for the disordered case. While relative FLOP rates for the sparse matrix−matrix multiplications are small for small filling fractions, the effective performance, i.e. the wall-clock time of a matrix−matrix multiplication, is superior.

3.2. GPU-Based Implementation. We have previously reported on the implementation of the SP2 algorithm in dense matrix algebra on GPUs[19] using the CUDA Basic Linear Algebra Subroutines (cuBLAS) library,[52] which is the GPU-accelerated version of BLAS. The cuBLAS DGEMM for dense matrix−matrix multiplication yields excellent performance. The CUDA Sparse Matrix Library, cuSPARSE, provides subroutines for arithmetic operations on sparse matrices on Nvidia GPUs.[47] A sparse matrix−matrix multiplication kernel based on the CSR format was recently added to the cuSPARSE library upon which we have developed an $O(N)$ implementation of the SP2 algorithm for GPUs. The thresholding of the intermediate matrices in the recursive expansion was performed using the Thrust library.[53]

The sparse CSR SP2 GPU version was tested with the polymer systems using both ordered and disordered matrices running on one Nvidia Tesla M2090 GPU. The performance shown in Figure 8 corresponds to the wall-clock time for squaring the

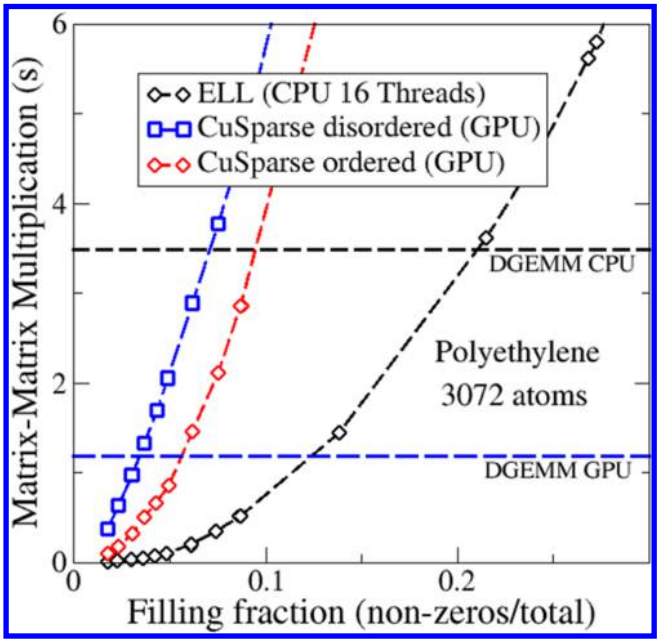


Figure 8. The total wall-clock time for squaring a matrix as a function of filling fraction for a 3072 atom system of poly(ethylene). The ELLPACK-R (ELL) based sparse matrix−matrix multiplication for a disordered matrix is compared to the corresponding cuSPARSE GPU calculations either for an ordered or disordered matrix. The upper constant dashed line (black) gives the time for the corresponding dense threaded BLAS DGEMM CPU calculation, and the lower constant dashed line (blue) shows the corresponding time for the dense cuBLAS DGEMM GPU calculation. The numerical tolerances corresponding to the filling fraction for cuSPARSE ordered are 1e-05 to 1e-16, while for disordered are 1e-05 to 1e-14. The ELL ordered tolerances range from 1e-05 to 1e-46 (using static OpenMP scheduling).

converged density matrix as a function of its filling fraction. The filling fraction was adjusted by applying a numerical threshold to the matrix elements. The threaded ELLPACK-R SP2 algorithm using OpenMP static scheduling for disordered matrices is also shown in Figure 8 as a comparison. The threaded ELLPACK-R SP2 algorithm running on a multicore CPU platform produces by far the best performance. We also find that the GPU implementation is sensitive to the ordering of the

DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654

matrix, which indicates that nonlocal memory access takes its toll. In contrast to the CPU, only part of the working vector in the Gustavson algorithm can be in cache at any time due to the GPU's smaller cache sizes. The sparse CPU and GPU (ordered and disordered) versions perform better than the dense GPU version for filling fractions up to 12%, 5%, and 3%, respectively. They perform better than the dense CPU version for filling fractions up to 22%, 9%, and 6%, respectively.

3.3. MKL Implementation for Multicore CPU and Intel MIC. Sparse matrix−matrix multiplication and addition routines using the CSR format are also distributed with the Intel Math Kernel Library (MKL). A sparse CSR MKL SP2 threaded CPU version was tested with the polymer systems using both ordered and disordered matrices. Timings were obtained using the Intel C Compiler version 15.0 and MKL version 11.1 with OpenMP static scheduling on a single node of the "Wolf" cluster at LANL comprised of two eight-core Intel Xeon E5-2670 CPUs running at 2.6 GHz. Figure 9 shows the performance of CSR MKL SP2

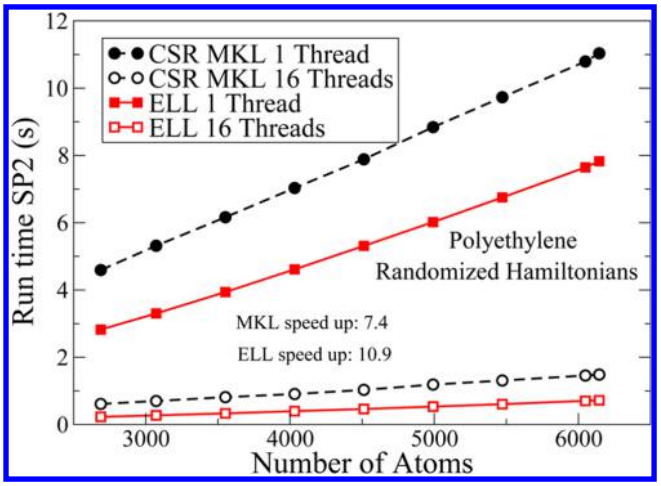


Figure 9. Run time comparison for calculating the density matrix for poly(ethylene) chains using the ELLPACK-R and CSR MKL threaded SP2 versions running on a CPU for disordered matrices. The circles correspond to the CSR MKL results, while the squares represent the ELLPACK-R results.

compared to the ELLPACK-R version. This MKL version is again sensitive to data locality due to matrix ordering. Results for the threaded SP2 on disordered matrices shows a speedup of 10.9 for ELLPACK-R and 7.4 for CSR MKL as the number of threads increase from 1 to 16. The ELLPACK-R version is 2.6 times faster than CSR MKL for ordered matrices and 2 times faster for disordered.

This same threaded sparse CSR MKL SP2 implementation was also tested with the polymer systems on the Intel MIC. In this case, the Intel C Compiler version 15.0 and MKL version 11.2 were used with OpenMP. Timings were obtained from a single node of the "Darwin" cluster at LANL comprised of an Intel Xeon CPU and Xeon Phi 5110P/5120D accelerator. Running on the accelerator, performance of the ELLPACK-R implementation was about twice as fast as CSR MKL for up to 64 threads for ordered and disordered matrices. A speedup of 3 was seen for ELLPACK-R and 2.4 for CSR MKL on disordered matrices from 1 to 16 threads.

ELLPACK-R SP2 was shown to outperform a CSR MKL implementation on a multicore CPU and Intel MIC. We attribute this to our use of the ELLPACK-R format which lends itself to more efficient parallelism coupled with a specialization of the SP2 algorithm, while CSR MKL provides a general, yet slower solution.

3.4. Comparison of SP2 Implementations. We have shown results for SP2 implementations on multicore CPUs, GPUs, and MICs. Table 1 shows a comparison of relative performance for a Hamiltonian from a 3072 atom poly(ethylene) system across the different architectures, libraries, and matrix formats presented in this paper. The MKL CPU (Intel Xeon) version is considered the baseline and assigned a performance of 1.00. The best performance on the maximum number of threads/cores for each implementation is quantified relative to that. Sixteen threads are used on most CPU implementations, while 64 are used on the MIC.

Only the sparse ELLPACK-R CPU implementation outperforms MKL CPU. It is 2.6X faster for an ordered Hamiltonian and 2.0X faster for disordered. The sparse ELLPACK-R MIC version is rated third with a performance over twice that of MKL CPU. The sparse CSR CPU, GPU, and MKL MIC implementations are significantly slower. Using sparse ELLPACK-R provides significant performance improvements over dense SP2 and traditional diagonalization.

Table 1. Comparison of the Performance of the SP2 Algorithm for Different Architecture, Library, and Matrix Format Combinations for a 3072 Atom Poly(ethylene) System[a]

architecture	library	matrix format	ordering	thread count	relative performance
Intel Xeon		Sparse ELLPACK-R	ordered	16	0.38
Intel Xeon		Sparse ELLPACK-R	disordered	16	0.51
Intel Xeon	MKL	Sparse CSR	ordered	16	1.00
Intel Xeon	MKL	Sparse CSR	disordered	16	1.32
Intel MIC		Sparse ELLPACK-R	ordered	64	2.06
Intel MIC		Sparse ELLPACK-R	disordered	64	2.73
Nvidia GPU	CuSparse	Sparse CSR	ordered		3.53
Intel MIC	MKL	Sparse CSR	ordered	64	4.83
Intel MIC	MKL	Sparse CSR	disordered	64	6.04
Intel Xeon		Sparse CSR	ordered	1	9.87
Nvidia GPU	CuSparse	Sparse CSR	disordered		11.81
Intel Xeon	BLAS	Dense	ordered	16	169.81
Intel Xeon	LAPACK	Dense (Diagonalization)	ordered	16	177.36

[a]Performance is relative with MKL on an ordered Hamiltonian matrix considered to be 1.00. The best performance on the maximum number of threads is shown for each.

DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654

4. APPLICATION TO BORN−OPPENHEIMER MOLECULAR DYNAMICS

The ability to achieve efficient, low prefactor linear scaling in the construction of the density matrix for intermediate sized problems including a few thousand atoms is of particular interest in electronic structure-based molecular dynamics simulations. The development of new Born−Oppenheimer molecular dynamics schemes enable stable, time-reversible integration of both the nuclear and electronic degrees of freedom even under approximate solutions of the electronic structure at each time step.[54−58] This next generation quantum-based molecular dynamics is based on an extended Lagrangian formulation of Born−Oppenheimer molecular dynamics and combines the best features of regular Born−Oppenheimer and Car−Parrinello molecular dynamics.[57,59,60] Extended Lagrangian Born−Oppenheimer molecular dynamics recently allowed the combination of linear scaling electronic structure calculation with long-term energy conserving simulations[61,62] often using only a single density matrix construction per time step.[63−65] In combination with the parallel ELLPACK-R based SP2 algorithm presented in this paper we can now significantly extend this capability.

The wall-clock time per time step is a critical quantity in molecular dynamics simulations. Linear scaling methods often enable very large systems (>10^4 atoms) on massively parallel computers, but the time for each force call may span minutes to hours. Since simulations of about 2−4 ps, corresponding to 10,000 to 20,000 time steps for organic materials, are required just to thermalize and equilibrate a system, such expensive force calls do not necessarily give rise to practical molecular dynamics simulations.

We illustrate the application of our low-prefactor, parallel, $O(N)$ implementation of the SP2 algorithm to molecular dynamics simulations of liquid water and a solvated polypeptide. The liquid water system consisted of 1000 molecules. The polypeptide system consisted of a 303-atom Trp cage miniprotein[66] with 2682 water molecules, depicted in Figure 10. Both

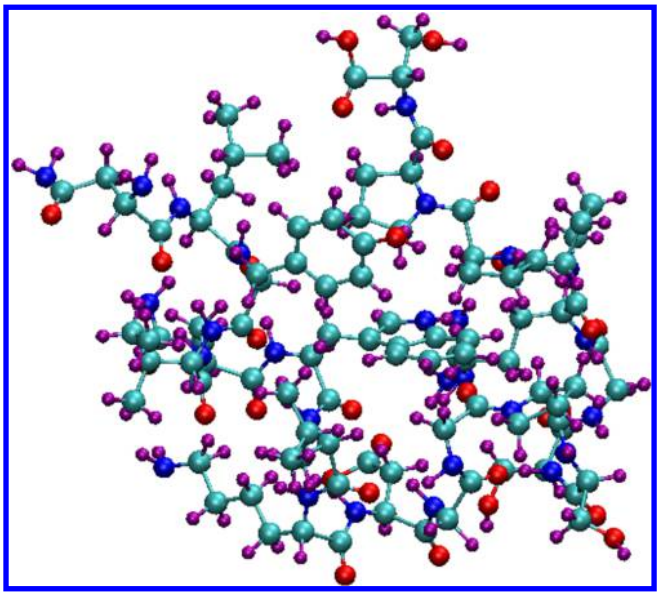


Figure 10. Trp cage protein structure is shown. The spheres correspond to carbon (turquoise), hydrogen (purple), nitrogen (blue), and oxygen (red) atoms.

trajectories were computed in the limit of zero self-consistent field cycles, or one density matrix build per time step, using the

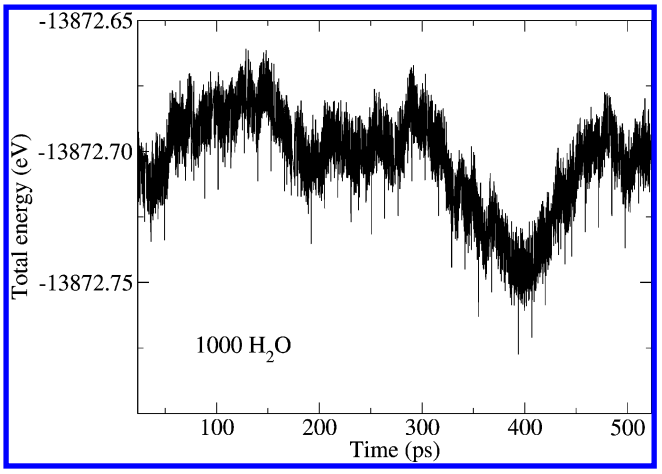


Figure 11. Time history of the total energy from a microcanonical simulation of 1000 H_2O molecules at 1 g/cm^3 at 300 K using the SCF-

generalized extended Lagrangian Born−Oppenheimer molecular dynamics formalism, with a time step of δt = 0.25 fs, and $\tau = 10^{-5}$ in the parallel, $O(N)$ SP2 formalism.

The 1000 molecule liquid water simulation was thermalized to 300 K at its experimental mass density (1 g/cm^3) for 23.25 ps using a Langevin thermostat with a time constant of 10^{-12} s^{-1}.[67] Following thermalization we continued the trajectory in the microcanonical ensemble using the velocity Verlet integrator to propagate the nuclear degrees of freedom. The equation of motion for the auxiliary degrees of freedom, **n**, in the SCF-free, generalized extended Lagrangian Born−Oppenheimer molecular dynamics formalism is

$$\ddot{\mathbf{n}} = c\omega^2(\rho - \mathbf{n}) \tag{6}$$

where c is a tunable local approximation of the inverse Jacobian of the residual function $\rho - \mathbf{n}$,[65] ω is the curvature of the harmonic well in which the auxiliary degrees of freedom evolve, and ρ is the optimized charge density or distribution obtained from an approximate, linearized Kohn−Sham functional. The dynamics are

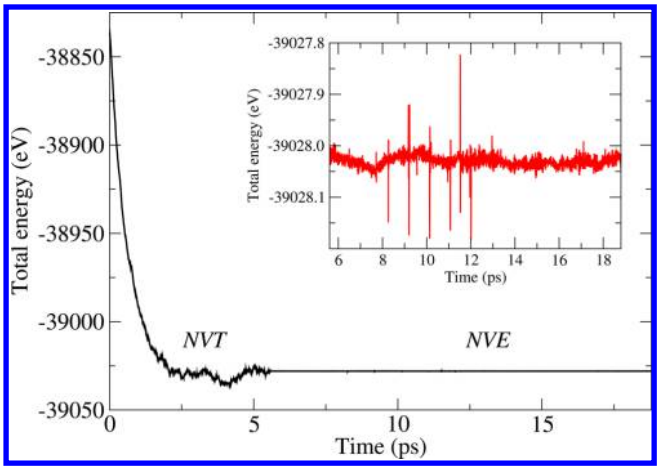


Figure 12. Evolution of the total energy of a 303 atom Trp cage protein solvated by 2682 water molecules during thermalization (*NVT*) and followed by microcanonical dynamics (*NVE*). The time history of the total energy during the microcanonical simulation is shown in the inset. The transients are an artifact of short (10 time step) regular Born−Oppenheimer molecular dynamics trajectories immediately following restarts before applying the generalized extended Lagrangian Born−Oppenheimer molecular dynamics formalism.

DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654

Chart 1. Pseudocode for the ELLPACK-R Sparse Matrix−Matrix Multiplication

```
!! nrows - number of rows in the matrix
!! mcols - maximum number of non-zeroes per row
!!
!! X matrix arays
!! numNonZerosX(nrows) - number of non-zeroes per row for matrix X
!! colmnIndX(mcols, nrows) - column indices for each row for matrix X
!! valueX(mcols, nrows) - non-zero values of matrix X
!!
!! Resulting X^2 matrix arrays
!! numNonZerosX2(nrows) - number of non-zeroes per row for matrix X^2
!! colmnIndX2(mcols, nrows) - column indices for each row for matrix X^2
!! valueX2(mcols, nrows) - non-zero values of matrix X^2
!!
!! workValue(nrows) - working sums used by each thread
!! workInd(nrows) - corresponding indices for workValue
!!
!! traceX - trace of the X matrix (the sum of the diagonal elements)
!! traceX2 - trace of the X^2 matrix
!! Initialize work arrays
workValue = ZERO
workInd = 0
!$OMP PARALLEL DO FIRSTPRIVATE(workInd, workValue)&
!$OMP PRIVATE(irow, jrow, icol, jcol, krow, kcol, lcnt, lcnt2, ax, xtmp)&
!$OMP REDUCTION(+:traceX, traceX2)
!! For each row of the X matrix
do irow = 1, nrows
  lcnt = 0
  !! For each non-zero element of the X matrix irow
  do jcol = 1, numNonZerosX(irow)
    ax = valueX(jcol, irow)
    jrow = columnIndX(jcol, irow)
    !! Calculate trace for X matrix diagonal elements
  if (jrow == irow) traceX = traceX + ax
  !! For each non-zero element of the X matrix jrow
  !! Taking advantage of symmetry, the row is used
  do kcol = 1, numNonZeros(jrow)
    krow = columnIndexX(kcol, jrow)
    if (workInd(krow) == 0) then
      lcnt = lcnt + 1
      columnIndX2(lcnt, irow) = krow
      workInd(krow) = irow
      workValue(krow) = ZERO
    endif
    !! Calculate working sum for X matrix row elements
    !! Multiply current X matrix value by all non-zeroes in
    !! corresponding column as row, since symmetric
    workValue(krow) = workValue(krow) + ax * value(kcol, jrow)
  enddo
enddo
lcnt2 = 0
!! For all potential non-zero values in the X^2 matrix row
do jrow = 1, lcnt
  jcol = columnIndX2(jrow, irow)
  xtmp = workValue(jcol)
  !! Calculate trace for X^2 matrix diagonal elements and
  !! store in X^2 matrix arrays
  if (jcol == irow) then
    traceX2 = traceX2 + xtmp
      lcnt2 = lcnt2 + 1
      valueX2(lcnt2, irow) = xtmp
      columnIndX2(lcnt2, irow) = jcol
    !! Threshold values and store in X^2 matrix arrays
    else if (ABS(xtmp) > NUMTHRESH) then
      lcnt2 = lcnt2 + 1
      valueX2(lcnt2, irow) = xtmp
      columnIndX2(lcnt2, irow) = jcol
    endif
    !! Zero used work elements
    workInd(JP) = 0
    workValue(JP) = ZERO
  enddo
  !! Set number of non-zeroes in row for X^2 array
  numNonZerosX2(irow) = lcnt2
enddo
!$OMP END PARALLEL DO
```


DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654

sensitive to the value of the parameter c in eq 6, and we found that using $c = 0.6$ gave a negligible energy drift (if any) and ensured the long-term stability of the integration. A calculation of the exact nonlocal expression was therefore not necessary. The total energy of the microcanonical trajectory is plotted over 500 ps of simulation time (2×10^6 time steps with 19.2 ps of simulation time per day) in Figure 11. The upper limit to a systematic drift in the total energy is only 21 neV/atom/ps. We showed previously that the application of a numerical threshold to small matrix elements during the SP2 expansion increases the amplitude of the apparent "random walk" of the total energy about its mean value with respect to a calculation performed using exact forces.[61] This small random walk of the local truncation error, as is seen in the total energy fluctuations, occurs instead of a systematic long-term energy drift thanks to the extended Lagrangian formulation. The long-term energy drift is therefore neglible even if the numerical thresholding used in the SP2 expansion leads to nonvariational errors. The test systems chosen in this work are too large for long duration trajectories to be computed using both our approximate, thresholded $O(N)$ method and $O(N^3)$ matrix diagonalization or exact $O(N)$ methods with $\tau = 0$. For instance, applying a threshold of $\tau = 10^{-5}$ in the SP2 expansion in ELLPACK-R format for 1000 water molecules gives a speed-up of ×1375 relative to the same calculation with no numerical threshold, $\tau = 0$, and a speed-up of ×152 with respect to $O(N^3)$ matrix diagonalization.

In Figure 12 we present an 18.8 ps simulation of the Trp cage miniprotein, a designed polypeptide.[66] Trp cage is small (20 residues) and has a limited set of elements (CNOH), making it an ideal system for demonstrating the potential of our methods to simulate protein dynamics. It exhibits two-state folding kinetics[66] which has been studied in detail by molecular dynamics simulations.[68] The Trp cage simulation volume was prepared from model 1 in Protein Data Bank[69] entry 1L2Y.[66] The structure was solvated in a cube 43.923 Å per side using GROMACS version 4.5.5.[70] The polypeptide was neutralized by hand-adjusting the hydrogen atoms. The final structure consisted of 303 covalently linked protein atoms plus 2682 water molecules (8349 atoms).

The Trp cage simulation used the generalized extended Lagrangian Born−Oppenheimer molecular dynamics formalism and the parallel, $O(N)$ SP2 algorithm. In this case, we found good energy conservation and stability using $c = 0.4$ in the propagation of the auxiliary degrees of freedom. We have included in Figure 12 the time history of the total energy during thermalization using a Langevin thermostat over the first 5.6 ps which is followed by 13.2 ps in the microcanonical ensemble. Here it is evident that 3−4 ps of simulation time (10,000−20,000 time steps) are required before thermal equilibration is achieved. Despite the large system size, approaching 10^4 atoms, with our efficient $O(N)$ computation of the density matrix we are able to run 2.1 ps per day of simulation time using relatively modest computational resources (one node of a supercomputer rather than large numbers of distributed memory nodes). The upper limit of the systematic drift in the NVE trajectory is 45 neV/atom/ps.

5. SUMMARY AND CONCLUSIONS

We have proposed a simple and practical approach to a shared memory parallel linear scaling calculation of the density matrix for intermediate sized problems of a few thousand atoms. In combination with the recently developed extended Lagrangian framework for Born−Oppenheimer molecular dynamics, our technique allows subsecond wall-clock time calculations of the density matrix in molecular dynamics simulations of simple hydrocarbons with up to several thousand atoms. Our approach compares favorably to current library implementations for sparse matrix−matrix multiplications on CPUs, GPUs, and Intel MICs.

APPENDIX

Pseudocode for Computing X^2 in ELLPACK-R Format

The pseudocode for the ELLPACK-R sparse matrix−matrix multiplication is listed in Chart 1. It is written in Fortran style for simplicity. Each sparse matrix is represented by 3 arrays, column indices, values, and number of nonzeroes per row. The calculation is done across multiple threads using OpenMP. Each thread uses its own working arrays to accumulate temporary sums. The input X matrix and resulting X^2 matrix are symmetric.

AUTHOR INFORMATION

Corresponding Authors

*E-mail: smm@lanl.gov.
*E-mail: cawkwell@lanl.gov.
*E-mail: amn@lanl.gov.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work was supported by the Laboratory Directed Research and Development (LDRD) program of Los Alamos National Laboratory. This research used resources provided by the Los Alamos National Laboratory Institutional Computing Program, which is supported by the U.S. Department of Energy National Nuclear Security Administration. This research has been supported at Los Alamos National Laboratory under the Department of Energy contract DE-AC52-06NA25396.

REFERENCES

(1) Hohenberg, P.; Kohn, W. *Phys. Rev.* **1964**, *136*, B864−B871.
(2) Kohn, W.; Sham, L. J. *Phys. Rev.* **1965**, *140*, A1133−A1138.
(3) Bowler, D. R.; Miyazaki, T. *Rep. Prog. Phys.* **2012**, *75*, 036503−036546.
(4) Goedecker, S. *Rev. Mod. Phys.* **1999**, *71*, 1085−1123.
(5) Bowler, D. R.; Miyazaki, T. *J. Phys.: Condens. Matter* **2010**, *22*, 074207.
(6) VandeVondele, J.; Borstnik, U.; Hutter, J. *J. Chem. Theory Comput.* **2012**, *8*, 3565.
(7) Weber, V.; Laino, T.; Pozdneev, A.; Feduova, I.; Curioni, A. *J. Chem. Theory Comput.* **2015**, *11*, 3145.
(8) Bowler, D. private communication, CECAM meeting Bremen, July 2015.
(9) Vandevondele, J. private communication, CECAM meeting Bremen, July 2015.
(10) Elstner, M.; Poresag, D.; Jungnickel, G.; Elstner, J.; Haugk, M.; Frauenheim, T.; Suhai, S.; Seifert, G. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1998**, *58*, 7260.
(11) Finnis, M. W.; Paxton, A. T.; Methfessel, M.; van Schilfgarde, M. *Phys. Rev. Lett.* **1998**, *81*, 5149.
(12) Frauenheim, T.; Seifert, G.; Hajnal, Z.; Jungnickel, G.; Porezag, D.; Suhai, S.; Scholz, R.; Elsterner, M. *Phys. Status Solidi B* **2000**, *217*, 41.
(13) Parr, R. G.; Yang, W. *Density-functional theory of atoms and molecules*; Oxford University Press: Oxford, 1989.
(14) Dreizler, R.; Gross, K. *Density-functional theory*; Springer Verlag: Berlin, Heidelberg, 1990.

DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654

(15) Roothaan, C. C. J. *Rev. Mod. Phys.* **1951**, *23*, 69−89.
(16) McWeeny, R. *Proc. R. Soc. London, Ser. A* **1956**, *235*, 496.
(17) McWeeny, R. *Rev. Mod. Phys.* **1960**, *32*, 335.
(18) Niklasson, A. M. N. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2002**, *66*, 155115.
(19) Cawkwell, M. J.; Sanville, E. J.; Mniszewski, S. M.; Niklasson, A. M. N. *J. Chem. Theory Comput.* **2012**, *8*, 4094−4101.
(20) Golub, G.; van Loan, C. F. *Matrix Computations*; Johns Hopkins University Press: Baltimore, 1996.
(21) Chow, E.; Liu, X.; Smelyanskiy, M.; Hammond, J. R. *J. Chem. Phys.* **2015**, *142*, 104103.
(22) Palser, A. H. R.; Manolopoulos, D. E. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1998**, *58*, 12704.
(23) Holas, A. *Chem. Phys. Lett.* **2001**, *340*, 552−558.
(24) Niklasson, A. M. N.; Tymczak, C. J.; Challacombe, M. *J. Chem. Phys.* **2003**, *118*, 8611.
(25) Niklasson, A. M. N. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2003**, *68*, 233104.
(26) Jordan, D. K.; Mazziotti, D. A. *J. Chem. Phys.* **2005**, *122*, 084114.
(27) Rubensson, E. H. *J. Chem. Theory Comput.* **2011**, *7*, 1233.
(28) Suryanarayana, P. *Chem. Phys. Lett.* **2013**, *555*, 291.
(29) Rubensson, E. H.; Niklasson, A. *SIAM J. Sci. Comput.* **2014**, *36*, B147.
(30) Gustavson, F. G. *ACM Trans. Math. Soft.* **1978**, *4*, 250.
(31) Pissanetzky, S. *Sparse Matrix Technology*; Academic Press: London, 1984.
(32) Rubensson, E. H.; Rudberg, E.; Salek, P. *J. Chem. Phys.* **2008**, *128*, 074106.
(33) Rubensson, E. H.; Salek, P. *J. Comput. Chem.* **2005**, *26*, 1628−1637.
(34) Daw, M. S. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1993**, *47*, 10895.
(35) Li, X. P.; Nunes, R. W.; Vanderbilt, D. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1993**, *47*, 10891.
(36) Veeraraghavan, S.; Mazziotti, D. A. *Phys. Rev. A: At., Mol., Opt. Phys.* **2015**, *92*, 022512.
(37) Anderson, E.; Bai, Z.; Bischof, C.; Demmel, J.; Dongarra, J.; Du Croz, J.; Greenbaum, A.; Hammarling, S.; McKenney, A.; Ostrouchov, S.; Sorensen, D. *LAPACK Users' Guide*; SIAM: 1994.
(38) Blackford, L. S.; Demmel, J.; Dongarra, J.; Duff, I.; Hammarling, S.; Henry, G.; Heroux, M.; Kaufman, L.; Lumsdaine, A.; Petitet, A.; Pozo, R.; Remington, K.; Whaley, R. C. *ACM Trans. Math. Soft.* **2002**, *28*, 135−151.
(39) Challacombe, M. *Comput. Phys. Commun.* **2000**, *128*, 93.
(40) Bowler, D. R.; Miyazaki, T.; Gillan, M. *Comput. Phys. Commun.* **2001**, *137*, 255−273.
(41) Hine, N. D. M.; Haynes, P. D.; Mostofi, A. A.; Payne, M. C. *J. Chem. Phys.* **2010**, *133*, 114111.
(42) Buluc, A.; Gilbert, J. *SIAM J. Sci. Comput.* **2012**, *34*, C170.
(43) Borstnik, U.; VandeVondele, J.; Weber, V.; Hutter, J. *Parallel Computing* **2014**, *40*, 47.
(44) Bock, N.; Challacombe, M.; Kale, L. V. *arXiv*; 2013. http://arxiv.org/abs/1403.7458v5 (accessed Sept 1, 2015).
(45) Vazquez, F.; Ortega, G.; Fernandez, J. J.; Garzon, E. *10th IEEE International Conference on Computer and Information Technology* **2010**, 1146−1151.
(46) OpenMP Architecture Review Board, *OpenMP*; 2014. http://openmp.org(accessed Sept 1, 2015).
(47) NVIDIA, *cuSPARSE*; 2014. https://developer.nvidia.com/cusparse (accessed Sept 1, 2015).
(48) Intel, *Intel Math Kernel Library (intel MKL)*; 2015. https://software.intel.com/en-us/intel-mkl (accessed Sept 1, 2015).
(49) Finnis, M. *Interatomic Forces in Condensed Matter*, 1st ed.; Oxford University Press Inc.: New York, 2003.
(50) Cawkwell, M. J.; Niklasson, A. M. N.; Dattelbaum, D. M. *J. Chem. Phys.* **2015**, *142*, 064512.
(51) Sanville, E. J.; Bock, N.; Coe, J. D.; Martinez, E.; Mniszewski, S. M.; Negre, C.; Niklasson, A. M. N.; Cawkwell, M. J. LATTE; Los Alamos National Laboratory: 2010; (LA-CC 10-004).
(52) NVIDIA, *cuBLAS*; 2014. https://developer.nvidia.com/cuBLAS (accessed Sept 1, 2015).
(53) NVIDIA, *Thrust*; 2014. https://developer.nvidia.com/thrust (accessed Sept 1, 2015).
(54) Niklasson, A. M. N. *Phys. Rev. Lett.* **2008**, *100*, 123004.
(55) Steneteg, P.; Abrikosov, I. A.; Weber, V.; Niklasson, A. M. N. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2010**, *82*, 075110.
(56) Zheng, G.; Niklasson, A. M. N.; Karplus, M. *J. Chem. Phys.* **2011**, *135*, 044122.
(57) Hutter, J. *WIREs Comput. Mol. Sci.* **2012**, *2*, 604.
(58) Lin, L.; Lu, J.; Shao, S. *Entropy* **2014**, *16*, 110.
(59) Marx, D.; Hutter, J. *Modern Methods and Algorithms of Quantum Chemistry*, 2nd ed.; Grotendorst, J., Ed.; John von Neumann Institute for Computing: Jülich, Germany, 2000.
(60) Car, R.; Parrinello, M. *Phys. Rev. Lett.* **1985**, *55*, 2471.
(61) Cawkwell, M. J.; Niklasson, A. M. N. *J. Chem. Phys.* **2012**, *137*, 134105.
(62) Arita, M.; Bowler, D. R.; Miyazaki, T. *J. Chem. Theory Comput.* **2014**, *10*, 5419.
(63) Niklasson, A. M. N.; Cawkwell, M. J. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2012**, *86*, 174308.
(64) Souvatzis, P.; Niklasson, A. *J. Chem. Phys.* **2014**, *140*, 044117.
(65) Niklasson, A. M. N.; Cawkwell, M. J. *J. Chem. Phys.* **2014**, *141*, 164123.
(66) Neidigh, J. W.; Fesinmeyer, R. M.; Andersen, N. H. *Nat. Struct. Biol.* **2002**, *9*, 425−430.
(67) Martinez, E.; Cawkwell, M. J.; Voter, A. F.; Niklasson, A. M. N. *J. Chem. Phys.* **2015**, *142*, 154120.
(68) Paschek, D.; Hempel, S.; GarcŠa, A. E. *Proc. Natl. Acad. Sci. U. S. A.* **2008**, *105*, 17754−17759.
(69) Berman, H. M. *Nucleic Acids Res.* **2000**, *28*, 235−242.
(70) Berendsen, H. J. C.; van der Spoel, D.; van Drunen, R. *Comput. Phys. Commun.* **1995**, *91*, 43−56.

DOI: 10.1021/acs.jctc.5b00552
J. Chem. Theory Comput. 2015, 11, 4644−4654