JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

Semiempirical Molecular Dynamics (SEMD) I: Midpoint-Based Parallel Sparse Matrix−Matrix Multiplication Algorithm for Matrices with Decay

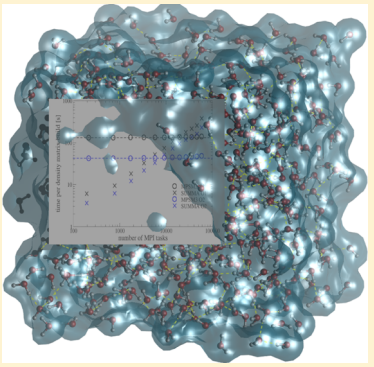
Valéry Weber,* Teodoro Laino,[†] Alexander Pozdneev,[‡] Irina Fedulova,[¶] and Alessandro Curioni[†]

[†]IBM Research−Zurich, Säumerstrasse 4, 8803 Rüschlikon, Switzerland

[‡]IBM Systems Lab Services Russia/CIS, Presnenskaya Nab., 10, 123317, Moscow, Russia

[¶]IBM Science and Technology Center, Presnenskaya Nab., 10, 123317, Moscow, Russia

ABSTRACT: In this paper, we present a novel, highly efficient, and massively parallel implementation of the sparse matrix−matrix multiplication algorithm inspired by the midpoint method that is suitable for matrices with decay. Compared with the state of the art in sparse matrix−matrix multiplications, the new algorithm heavily exploits data locality, yielding better performance and scalability, approaching a perfect linear scaling up to a process box size equal to a characteristic length that is intrinsic to the matrices. Moreover, the method is able to scale linearly with system size reaching constant time with proportional resources, also regarding memory consumption. We demonstrate how the proposed method can be effectively used for the construction of the density matrix in electronic structure theory, such as Hartree−Fock, density functional theory, and semiempirical Hamiltonians. We present the details of the implementation together with a performance analysis up to 185 193 processes, employing a Hamiltonian matrix generated from a semiempirical NDDO scheme.



1. INTRODUCTION

Linear scaling calculations are of paramount importance for realistic modeling of large-scale calculations, such as biological and material science systems. A key aspect of any linear scaling electronic structure theory is the circumvention of the bottleneck of the diagonalization's cubic scaling through purification[1−3] or density matrix minimization methods,[4,5] in which the computationally expensive kernel is represented by the sparse matrix−matrix multiplication. As acknowledged by several authors,[6,7] computer scientists so far have mainly explored theoretical aspects of the parallel sparse matrix−matrix multiplication,[8,9] with relatively little research on novel algorithms. This was mainly justified with the diverging interests between the fields of numerical analysis and discrete algorithm design, because the multiplication between sparse matrices and dense vectors were considered to be of much wider applicability for designing efficient iterative linear solvers. Only recently has the role of sparse matrices in quantum chemistry applications received increasing attention, because of the implementation and performance challenges that are intrinsic to linear scaling electronic structure methods. At the same time, because of the ever-growing increase of parallel computational resources, the design of scalable algorithms for sparse matrix multiplication is becoming a task of paramount importance for computational scientists.

Four decades ago, Gustavson[10] introduced one of the first algorithms to multiply, serially, two sparse matrices which, compared with the McNamee[11] general-purpose sparse matrix−matrix multiplication algorithm, sensibly decreased the complexity of the multiply operation. Gustavson introduced, for the first time, the concept of the compressed sparse row (CSR) data structure to represent sparse matrices. Since then, we have seen a steadily growing interest in the multiplication of sparse matrices, starting with the contribution by Challacombe,[6] who developed a general approach to the parallel sparse-blocked matrix−matrix multiplication. The method was designed in the context of linear scaling self-consistent-field (SCF) equations, and a well-designed non-blocking communication in the data parallel message passing was crucial to overlap computation and communication to increase the overall efficiency.

Recently, Buluç and Gilbert[12,13] focused their attention on slightly slower, when run on a single node, but easy to parallelize kernels for general sparse matrix−matrix multiplication (SpGEMM). They demonstrated that their SpGEMM method, which uses two-dimensional (2D) block data distribution with serial hyperspace kernels, is highly flexible and scalable, enabling an acceleration to thousands of processes. Their parallelization strategy is based on the scalable universal matrix multiplication (SUMMA), designed by Van de Geijn and Watts[14,15] for the multiplication of dense matrices. In the past decade, several libraries for sparse matrix−matrix multiplications were developed,[7,13] and the sparse SUMMA is probably the most popular scheme; it is used in all of the libraries. Sparse SUMMA is characterized by a $O(P^{1/2})$ scaling

Received: April 23, 2015

DOI: 10.1021/acs.jctc.5b00382
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

(where P is the number of processes), in which the communication costs and the logistic of the computation dominate. Because of the $O(P^{1/2})$ regime, the method does not scale to a large number of processes.

Among the schemes that have a better communication pattern than the SUMMA algorithm, it is worth mentioning those by Bowler et al.,[16] Hine et al.,[17] and Rubensson and Rudberg,[18] all of which use schemes based on data locality or locality-aware approaches.

For performing large-scale calculations, an obvious choice is the use of semiempirical Hamiltonians. Semiempirical methods were originally developed to reduce the cost of the Hartree−Fock method and to include some of the missing correlation effects through parametrization. The semiempirical framework has also been used successfully to describe a large class of different chemical problems, such as photochemistry[19] and nonadiabatic molecular dynamics in complex environments.[20,21] The modern version of the semiempirical approximation is based on the neglect of the diatomic differential overlap (NDDO),[22,23] for which numerous parametrizations have been proposed.[24−29] NDDO still requires some degree of tuning to describe experimental or *ab initio* data accurately, similarly to the semiempirical corrections developed in the past decades for purely *ab initio* methods. As of today, no purely *ab initio* methods exist for an on-the-fly electronic structure description that are both efficient and accurate and can scale to millions of atoms.

In this manuscript, we present a novel computer program to perform semiempirical molecular dynamics (SEMD) simulations, resulting from deep insights into hardware solutions, the mathematical expertise to design novel algorithms that fully scale to and benefit from modern machine architectures, in conjunction with optimal quantum-chemical approaches based on semiempirical Hamiltonians, to advance computational biophysics beyond the fixed-charge approximation.

The kernel for the evaluation of the density matrix is a novel scheme based on a recently introduced method[30] for parallelizing range-limited particle interactions, known as the midpoint method. The midpoint method is part of a large class of methods called "neutral territory methods",[31] in which, unlike in traditional spatial decomposition methods, each process computes the interaction between a pair of particles, neither of which necessarily must reside on it. By exploiting similar concepts as in the midpoint method, we devised a novel algorithm for the multiplication of sparse matrices and applied it to the computation of the density matrix in electronic structure theory, such Hartree−Fock, density functional theory, and semiempirical Hamiltonians. The new algorithm will be referred to as midpoint-based parallel sparse matrix−matrix multiplication (or MPSM3).

Thanks to the superior scalability, when compared to other similar codes, SEMD opens the doors to long molecular dynamics studies (in the range of tens of nanoseconds) on systems of biological interests (∼1 million atoms) to be performed at the semiempirical level of theory on large-scale computational resources.

2. METHODS

2.1. NDDO Semiempirical Hamiltonians. Before disclosing the details of the MPSM3 algorithm, we briefly discuss the theory of NDDO semiempirical methods, extensively presented in several publications.[32,33] For this reason, we only review the general structure of the semiempirical Hamiltonian, focusing on the terms that have been specifically modified in the present work. For simplicity of notation and without loss of generality, we consider systems with a closed-shell configuration. The generalization to open-shell systems is straightforward. The variational treatment of the molecular orbital coefficients $C_{\mu i}$ (where μ labels the atomic orbitals and i the molecular orbitals) leads to the Roothan equations:[34]

$$\sum_{\nu=1}^{N} F_{\mu\nu}C_{\nu i} = C_{\mu i}\epsilon_i$$

in which ϵ_i is the energy of the molecular orbital i, $F_{\mu\nu}$ are the matrix elements of the Fockian matrix, and N is the number of atomic basis sets. We can define the various components in the following way:

$$F = H + G(D) \quad \text{nd} \quad D_{\lambda\sigma} = 2\sum_{i=1}^{N_s} C_{\lambda i}C_{\sigma i}^{\dagger}$$

where H is the one-electron Hamiltonian matrix, including the kinetic energy and the potential energy in the electrostatic field of the nuclei, G is the matrix of the potential due to the valence electrons, and N_s represents the number of occupied states. G depends on the molecular orbitals via the density matrix D.

The total electronic energy of the valence electrons can be cast in the following form:

$$E_{\text{elec}} = \frac{1}{2}\text{Tr}[D(H + F)]$$

and the total energy of the system is obtained by adding the repulsion energy between nucleus to the total electronic energy. Of the various approximations leading to the NDDO Hamiltonian, here, we focus only on the expression of the off-diagonal core matrix elements $H_{\mu\nu}$. For those cases where μ and ν are on different atoms, the corresponding matrix element is computed as

$$H_{\mu\nu} = \beta_{\text{AB}}S_{\mu\nu} \tag{1}$$

where $S_{\mu\nu}$ is an overlap integral, evaluated analytically between Slater orbitals, and β_{AB} is a parameter that is dependent only on the nature of atoms A and B.

2.1.1. Tapering of the Overlap. NDDO methods are generally much less costly than first-principles methods. The reason for this is not only the reduced cost of computing the Hamiltonian matrix, but also the fact that the matrices are much more sparse than the corresponding first-principle ones. Generally, the sparsity of the Hamiltonian depends on the corresponding parametrization and is affected by the overlap matrix used in the evaluation of the core part (see eq 1). To increase sparsity in the Hamiltonian matrix, we used a tapering function to cut the long tails of the overlap integrals and thus improve the poor sparsity induced by the standard PM6 parametrization.[28] The function used for tapering is

$$T(r) = \frac{1}{2}\left[1 - \tanh\left(\frac{r - r_0}{r_c}\right)\right] \tag{2}$$

in which we used values of r_c = 0.5 Å and r_0 = 2.0 Å. A thorough validation of the new reparameterization will be presented in a forthcoming paper.

2.1.2. Treatment of the Electrostatic. Similar to the overlap, all electrostatic interactions in the condensed-phase systems

DOI: 10.1021/acs.jctc.5b00382
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

have been treated with a pairwise neighbor list scheme, tapering the Coulomb behavior at large distances. In this specific case, the interactions were cut off using values of r_c = 12.0 Å and r_0 = 0.5 Å (see eq 2). Although this approximation may sound crude, it is justified with the possibility that, in future developments, the corresponding truncated potential can be replaced with more sophisticated expressions, including a mean-field potential for the treatment of long-range interactions within the pair-based potential.[35,36]

2.2. Sparse Matrix Library. We use a distributed blocked compress sparse row (BCSR) format to store the sparse matrices. Note that also the list of rows is compressed. Each block represents the interaction between a pair of atoms. The dimensions of the block are dependent on the type of atoms involved in the interaction and the level of theory used. In the case of semiempirical, the different block sizes of a system containing only first- and second-row atoms are 1 × 1, 1 × 4, 4 × 1, and 4 × 4.

The local multiplication $C \leftarrow \alpha AB + \beta C$ is performed as follows:

(a) Scale βC; for $\beta = 0$, the blocks are zeroed, but not removed.
(b) Hash columns of each row of matrix C (use BCSR to hash tables).
(c) Multiply the local matrices A and B with an ikj loop order to obtain αAB. The row hashes are used in the innermost loop to quickly access or create C_{ij} blocks.
(d) Finalize matrix C, i.e., for each row, the columns contained in the hash table are sorted and repacked into the BCSR format.

Steps a, b, and c are performed only once per parallel multiplication. In Algorithm 1, we present a pseudo-code that describes a basic implementation of the multiplication.

Algorithm 1 Pseudo code for the multiply

```
C ← βC
Hash columns of each row of C
for Communication steps (SUMMA or MPSM3) do
    SEND/RECV A and B matrices
    Perform local multiply C ← C + αAB
end for
Finalize C
```

2.3. SUMMA. We implemented the SUMMA algorithm[14] with different ring broadcasts, i.e., naive broadcast, nonblocking, and one-sided pipelining. A random distribution is used to load balance the atoms across the processes. The same distribution of the atoms is used for the rows and columns to avoid unnecessary operations. The main drawback of the SUMMA algorithm for a sparse matrix is the $P^{1/2}$ steps, leading to excessive logistic and communication operations.[13]

2.4. MPSM3. To avoid excessive communications and logistic operations, we designed a sparse matrix−matrix multiplication algorithm that exploits the physical properties of the problem, namely, the locality principle. In a local basis, quantum effects are short-ranged for nonmetallic systems. Locality is manifested in an approximate exponential decay of the density matrix $|D_{ij}|$ with an atom−atom separation $|\mathbf{R}_i - \mathbf{R}_j|$. The locality of D may be exploited to achieve $O(N)$ algorithms for SCF theory and beyond.

The simulation cell is divided into a regular grid of small boxes. Each box is assigned to a specific process in a three-dimensional (3D) Cartesian topology. Rather than distributing *atoms* of the system among the processes, we use the midpoint approach[30] to distribute the *blocks* of the matrix. Each block A_{ij} is located on the process that owns the corresponding midpoint between particle i and j.

We define the *interaction length* of a matrix A as

$$R_A = \max\{|\mathbf{R}_i - \mathbf{R}_k| \quad \forall\, ik \quad \text{such that } A_{ik} \neq 0\}$$

where $\mathbf{R}_i$ and $\mathbf{R}_k$ are atom positions. The interaction length of the AB matrix product is $R_{AB} \leq R_A + R_B$. Regardless of the initial interaction length of matrix C, the condition $R_C \geq R_{AB}$ holds after the matrix multiplication operation. As the AB matrix product contributes to matrix C only within the AB sparsity pattern, we assume $R_C = R_{AB}$ without any loss of generality.

In order to perform the $C \leftarrow AB + C$ operation, let us consider a term $A_{ik}B_{kj}$ that contributes to block C_{ij}. In our algorithm, the processes that own A_{ik} and B_{kj} send these blocks to the process that holds the C_{ij} block. The latter performs the $C_{ij} \leftarrow A_{ik}B_{kj} + C_{ij}$ operation (see Figure 1). In Algorithm 2, we present a pseudo-code that describes a basic implementation of the midpoint sparse matrix multiplication.

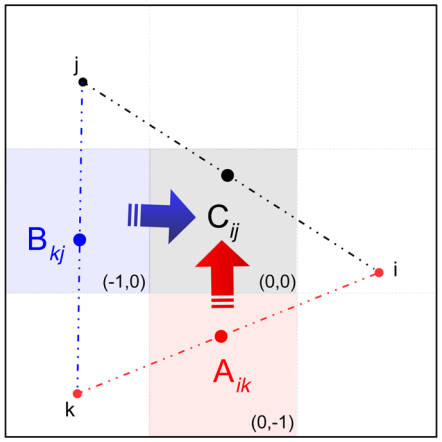


Figure 1. A matrix block A_{ik}, owned by a process that is responsible for the interaction between atoms i and k, is needed by some process that owns the midpoint between atoms i and j. A_{ik} and B_{kj} (owned by the processes indicated by shaded boxes red and blue, respectively) are sent to the process box shaded in black, which performs the product.

Algorithm 2 Pseudo code for the MPSM3 algorithm

```
procedure MPSM3(A, B, C)
    for dim ← x, y, z do
        for push ← 1, ⌈max(R_A,R_B) / 2l_dim⌉ do
            Prefiltering: A and B matrices before communication
            Non-blocking SEND/RECV: A and B matrices
            Multiply and Postfiltering: local with received matrices
            Wait for completion of SEND/RECV
        end for
    end for
    Multiply and Postfiltering: local with received matrices
end procedure
```

2.4.1. Communication. Let us assume that l_x, l_y, and l_z are the lengths of a process box along the corresponding spatial dimensions. Given an interaction length of R, the corresponding matrix block needs to be sent at most $\lceil R/(2l_x)\rceil$, $\lceil R/(2l_y)\rceil$, and $\lceil R/(2l_z)\rceil$ process boxes away from the process that owns the block, and we use the term *push* to denote this number. All the boxes first send messages in the $+x$- and $-x$-directions, along with all other data accumulated in previous pushes. A

DOI: 10.1021/acs.jctc.5b00382
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

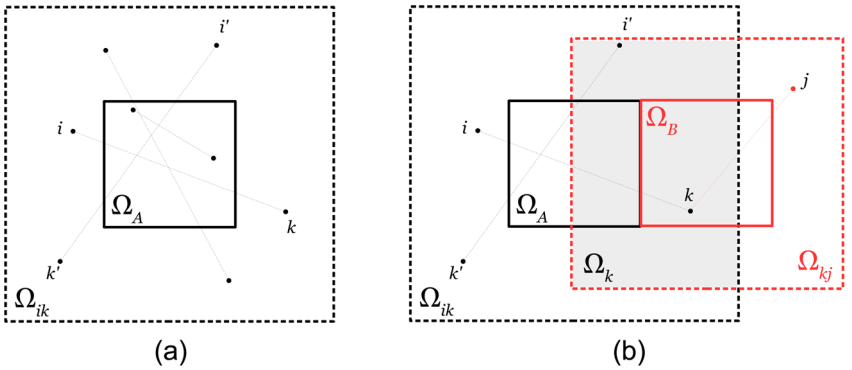


Figure 2. (a) Definition of the allowed domain. The solid line circumscribes the midpoint associated with a given process. Dashed lines indicate the upper bound domain that contains all the atoms, whose midpoints fall inside the process box Ω_A. A few pair of atoms (dots) are also shown connected by lines. (b) Definition of the allowed domain Ω_k for atoms k, shown in gray shading. While the k atom of the A_{ik} block falls within Ω_k, the k' atom of the $A_{i'k'}$ block does not and can be safely filtered out.

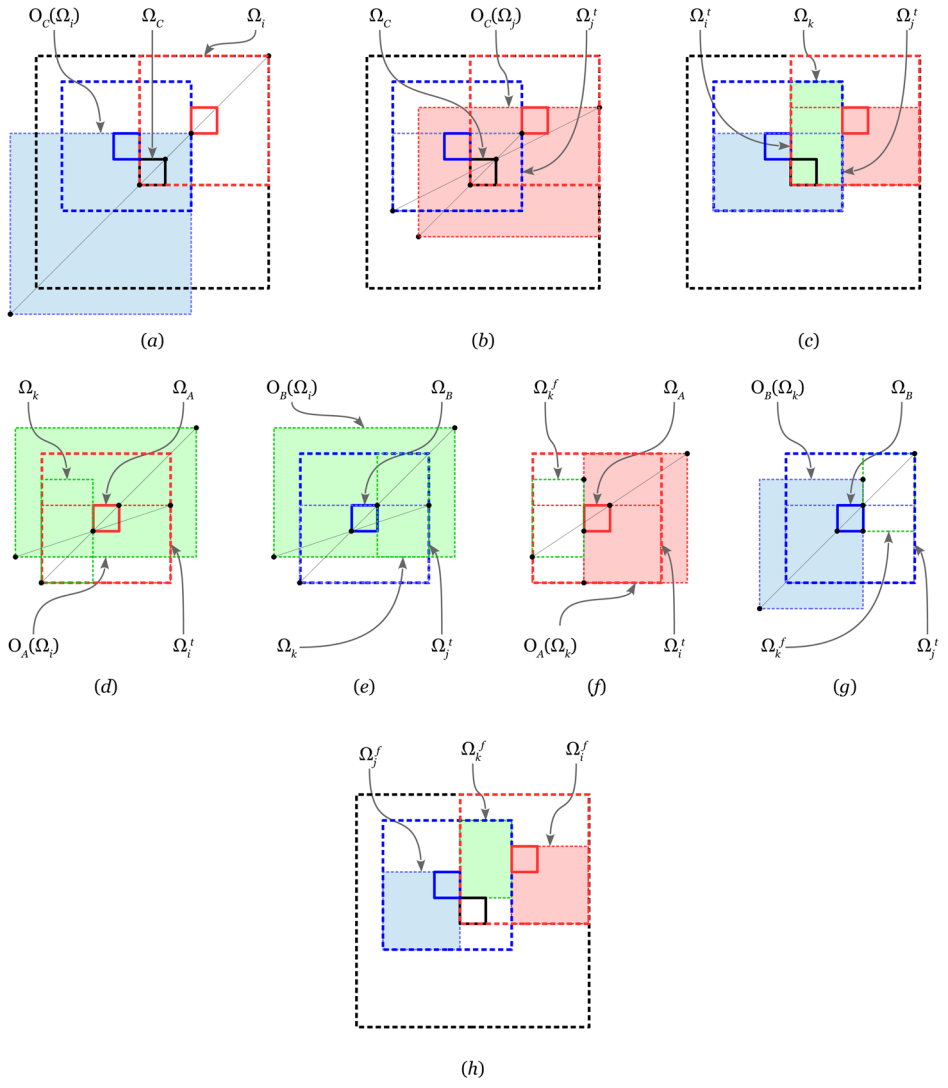


Figure 3. Shrinking allowed domains for atoms in a 2D postfiltering stage with two pushes per spatial dimension: (a−c) updating Ω_i and Ω_j based on Ω_C domain; (d, e) updating Ω_k based on Ω_i^t and Ω_j^t domains; (f, g) updating Ω_i^t and Ω_j^t, based on Ω_k^f domains; and (h) the resulting Ω_i^f, Ω_j^f, and Ω_k^f.

similar data transfer is performed sequentially for the y- and z-directions. This leads to accumulating $(2n + 1)^3$ matrices for n pushes per direction. The local matrix multiplication and the communications can be overlapped by using nonblocking communication routines (see Algorithm 2).

2.4.2. Filtering. The computationally most intensive part of Algorithm 2 is the local multiplication. To reduce the number of midpoint tests required in the local multiplication, we discard as many blocks of matrices that can by no means contribute to

DOI: 10.1021/acs.jctc.5b00382
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

the product. We call this procedure *filtering*. We implemented two types of filtering:

- prefiltering — this task is performed prior to sending data
- postfiltering — this task is performed on a particular pair of matrices prior to multiplication

Each matrix that is received by a process is filtered out, depending on the data that is needed to be sent further along the current spatial dimension and depending on the data that are needed for local multiplications.

While prefiltering reduces the number of pairs of each matrix A and B independently, postfiltering performs the reduction of pairs based on the corresponding domains for each index of the product $A_{ik}B_{kj}$. To postfilter matrix blocks, we must define the *allowed domain* of atoms that could potentially hold the atoms needed for multiplications. The allowed domain is calculated beforehand, and it is dependent only on the spatial position of the process boxes and on the interaction length of the matrices.

Specifically, let us consider a two-dimensional case. In Figure 2a, we use an Ω_A symbol and a solid line to show the domain covered by the process box. The atom domain Ω_{ik} that contains all the atoms that contribute with midpoints to the domain of Ω_A is shown with a dashed line. Having this notation, we can estimate the upper bound on the allowed domain for atoms k as the intersection $\Omega_k = \Omega_{ik} \cap \Omega_{kj}$ (shaded domain in Figure 2b). Similar domains can be evaluated for indexes i and j.

2.4.3. Shrinking Domains. The evaluation of the allowed domains can be further reduced by shrinking the atom domain Ω_x ($x = i, j, k$). The corresponding operation consists of applying a reflection to the atom domain, which creates a second atom domain. The transformation is defined such that the midpoints between the initial and the transformed domain fall inside the midpoint domain used for the reflection.

Specifically, in Figure 3, we consider a 2D case applied to the postfiltering for two pushes per spatial dimension. As one can see in Figure 3a, initially, the allowed domains Ω_i and Ω_j for indexes i and j coincide with the domains Ω_A and Ω_B.

The shrinking concept can be schematically described with the following steps:

$$\Omega_j^t \leftarrow \Omega_j \cap O_C(\Omega_i) \tag{3}$$

$$\Omega_i^t \leftarrow \Omega_i \cap O_C(\Omega_j^t) \tag{4}$$

$$\Omega_k^f \leftarrow \Omega_k \cap O_A(\Omega_i^t) \cap O_B(\Omega_j^t) \tag{5}$$

$$\Omega_i^f \leftarrow \Omega_i^t \cap O_A(\Omega_k^f) \tag{6}$$

$$\Omega_j^f \leftarrow \Omega_j^t \cap O_B(\Omega_k^f) \tag{7}$$

where O_X is a reflection operator that generates a domain containing atoms that, together with the atoms from the operator's argument, produce midpoints falling within Ω_X.

In the first step, shown in Figure 3a and described by eq 3, we evaluate the upper-bound domain $O_C(\Omega_i)$ containing atoms j that, together with atoms i from the allowed domain Ω_i, could produce midpoints that fall within Ω_C. The updated allowed domain Ω_j^t for atoms j is then evaluated in eq 3.

In the next step (Figure 3b), we use the updated Ω_j^t to evaluate

$$O_C(\Omega_j^t)$$

and compute the allowed domain Ω_i^t for atoms i as shown in eq 4. Figure 3c shows the temporary domains Ω_i^t and Ω_j^t, together with Ω_k.

Next, we consider the midpoint domains Ω_A and Ω_B, and independently evaluate

$$O_A(\Omega_i^t)$$

and

$$O_B(\Omega_j^t)$$

based on the recently computed Ω_i^t and Ω_j^t (see Figures 3d and 3e). As a result, we can evaluate the final Ω_k^f, as shown in eq 5.

At the end, we compute the final allowed domains Ω_i^f and Ω_j^f, as depicted in Figures 3f and 3g and described by eqs 6 and 7.

The different atom domains are shown in Figure 3h. It is possible to show that no further reduction of the domains Ω_i^f, Ω_j^f, and Ω_k^f is possible when using the present shrinking scheme.

2.5. Second-Order Spectral Projection Method. At zero electronic temperature, the density matrix can be written using the matrix form of the Heaviside step function as

$$D = \theta[\mu I - F] \tag{8}$$

where μ is the chemical potential, I is the identity matrix, and F is the Fockian obtained from density functional theory (DFT), Hartree–Fock theory, or a semiempirical method.

The second-order spectral projection (SP2) method[3] is based on a recursive expansion of eq 8 in a series of generalized matrix–matrix multiplications as

$$D = \theta[\mu I - F] = \lim_{i\to\infty} f_i[f_{i-1}[\dots f_0[X_0]]]$$

with

$$X_0 = \frac{\varepsilon_{\max} I - F}{\varepsilon_{\max} - \varepsilon_{\min}}$$

where $\varepsilon_{\min}$ and $\varepsilon_{\max}$ are estimations of the extremal eigenvalues of the Hamiltonian. In practice, the Gershgorin circle theorem[37] has proved to be very inexpensive and efficient at bounding the extremal eigenvalues.

The SP2 algorithm uses the following polynomials in the recursive expansion:

$$f_i[X_i] = \begin{cases} X_i^2 & \mathrm{Tr}[X_i] > N_s \\ 2X_i - X_i^2 & \text{otherwise} \end{cases}$$

The recursive application of these polynomials "purifies" the density matrix by bringing the unoccupied and occupied eigenvalues toward the fixed points at 0 and 1, respectively, and ensures that the correct occupation is reached at convergence, i.e., $\mathrm{Tr}[D] = N_s$. No prior knowledge of the chemical potential is required. The iterations are stopped when the sufficient idempotency convergence criteria $|\mathrm{Tr}[X_i^2 - X_i]|/N = |\mathrm{Tr}[X_{i+1}] - \mathrm{Tr}[X_i]|/N < \epsilon_{\mathrm{idem}}$ is fulfilled. Algorithm 3 contains the pseudo-code for the implementation of the SP2 method.

At each iteration, Algorithm 3 requires a copy, a multiplication, a trace, and a matrix filter. The filter procedure removes those blocks from the matrix that have a Frobenius norm smaller than a given threshold, ϵ_{mat}. To increase performance, the subtraction $2X_i - X_i^2$ is performed at the same time as the multiplication. Note also that the filtering and the trace can be performed at almost no extra cost while finalizing the matrix after just the multiplication.

DOI: 10.1021/acs.jctc.5b00382
J. Chem. Theory Comput. XXXX, XXX, XXX–XXX

Algorithm 3 Pseudo code for the implementation of SP2 algorithm

```
Estimate ε_max and ε_min
X ← (ε_max I − F)/(ε_max − ε_min)
trX_1 ← Tr[X]
for i ← 1, MaxIter do
    if trX_i > N_s then
        α ← 1 and β ← 0
    else
        α ← −1 and β ← 2
    end if
    X_tmp ← X
    X ← αX_tmp^2 + βX
    X ← Filter[X]
    trX_{i+1} ← Tr[X]
    if |trX_{i+1} − trX_i|/N < ε_idem break
end for
```

3. RESULTS AND DISCUSSIONS

All developments were implemented in the SEMD program for linear scaling electronic structure theory and semiempirical molecular dynamics, which will be released to the scientific community in the near future. The code was compiled using the IBM XL Fortran compiler for BlueGene/Q (BG/Q) 14.1.10 with the `-O2` optimization flags, and all calculations were carried out on a BlueGene/Q supercomputer. [Note: IBM and Blue Gene are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies.]

Each oxygen atom is described by one *s* and three *p* orbitals, whereas for each hydrogen, one *s* orbital was used.

To demonstrate the capabilities of the MPSM3 algorithm, we present the weak and strong scaling of the density matrix build for boxes of water molecules at normal liquid density and under periodic boundary conditions in this section. The density matrix is constructed with the SP2 (see Algorithm 3) with a matrix threshold of $\epsilon_{mat} = 10^{-6}$ and an idempotency convergence criterion of $\epsilon_{idem} = 10^{-8}$. The numerical accuracy of the SP2 algorithm has been already reported by Cawkwell and Niklasson,[38] in which the authors presented extensive molecular dynamics simulations with different matrix thresholds ranging from 10^{-7} to 10^{-5}.

The timing for this construction is measured for the first SCF iteration, starting with a diagonal guess of the density matrix.

All runs were performed on the Mira supercomputer at the Argonne Leadership Computing Facility, Argonne National Laboratory (Argonne, IL). Mira consists of 48 IBM BG/Q racks, reaching a theoretical peak performance of 10 PFlop/s. A rack consists of 1024 computational nodes, each hosting a 18-core A2 chip that runs at 1.6 GHz. Sixteen of these cores are devoted to computation, while the 17th is used by the lightweight O/S kernel and the 18th core is designated for redundancy. Every core supports 4 H/W threads. Each core has a 16 kB L1 cache, while the 16 cores share 32 MB of L2 cache. In total, Mira has 786 432 cores and can support up to 3 145 728 H/W threads. All the results presented in the following sections were performed with 4 MPI tasks (4 cores) per BG/Q node.

3.1. Performance Analysis. *3.1.1. Weak Scaling.* For the weak scaling benchmarks, we used boxes of $\sim 4096/6^3$ water molecules per MPI task (exact for MPSM3). We ran MPSM3 with $(3i + 3)^3$ and SUMMA with $(q(i) + \mathrm{mod}(q(i), 2))^2$ MPI tasks, with $q(i) = \mathrm{floor}((3i + 3)^{3/2})$ and $i = 1, ..., 12$.

The time per density matrix build as a function of the number of MPI tasks is presented with logarithmic scales for SUMMA and MPSM3 in Figure 4, for two different occupations of the density matrix, O1 and O2. The number of nonzero elements of the converged density matrix is $\sim 1.6 \times 10^3$/water for O1 and 1.0×10^3/water for O2.

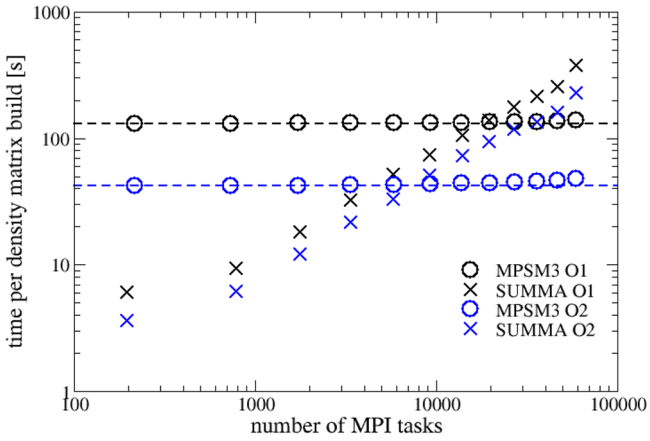


Figure 4. Weak scaling. Time per density matrix build, with respect to the number of MPI tasks, with $\sim 4096/6^3$ waters per task. Scaling results are shown for two different relative occupations: O1 and O2.

Because of logistics and unfavorable communications, the SUMMA algorithm should scale, at best, as the square root of the number of MPI tasks. A detailed analysis of the communication and computation costs for the sparse SUMMA can be found in the work of Buluç and Gilbert.[13] We observed that, for large numbers of MPI tasks, our implementation of the sparse SUMMA scales approximately like $O(P^{0.8})$, which is more than the theoretical value of $O(P^{1/2})$. Currently, we are not able to explain this behavior; however, this deviation does not affect the general conclusions of the new algorithm.

The MPSM3 algorithm scales much more favorably, and only a weak deviation from ideal scaling can be observed (horizontal dashed lines in Figure 4). We have localized the origin of this deviation in two distinct parts of the code, namely, (a) the computation of the extremal eigenvalues for SP2 (Gershgorin circle theorem), and (b) the addition of a scalar to the diagonal of a sparse matrix. Those operations use, unnecessarily, replicated global arrays or unparallelized loops. A better implementation avoiding replicated computations and data is rather straightforward.

For the particular number of water molecules per MPI task in this experiment, the SUMMA algorithm shows significantly better timings than MPSM3 when a small number of MPI tasks is used. The MPSM3 becomes faster than SUMMA from ~19 683 (373 248 water molecules) and ~9261 (175 616 water molecules) MPI tasks for O1 and O2 onward, respectively. For the largest system (1 124 864 waters), MPSM3 builds the density matrix 2.7 and 4.7 times faster than SUMMA for O1 and O2, respectively.

3.1.2. Strong Scaling. For the strong scaling benchmarks, we used three different boxes, with 110 592, 373 248, and 1 124 864 water molecules, respectively. The three systems are referred to as S1, S2, and S3, from the smallest to the largest. The time per density matrix build as a function of the number of MPI tasks is presented in Figure 5 on logarithmic scales for the SUMMA and MPSM3.

The SUMMA shows, as in the weak scaling case, good scaling for small numbers of MPI tasks. The MPSM3 algorithm shows good strong scaling as long as only one push is

DOI: 10.1021/acs.jctc.5b00382
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

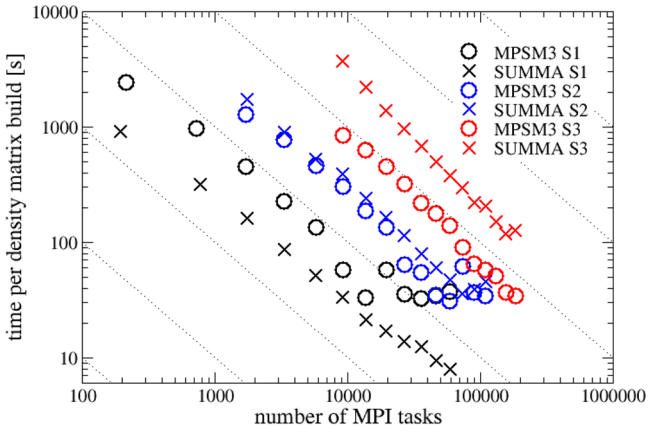


Figure 5. Strong scaling. Time per density matrix build, with respect to the number of MPI tasks for three different system sizes (S1, S2, and S3).

performed in each direction. This is an important limitation of the method for many MPI tasks and/or denser matrices. The ratio of the best time for MPSM3 and SUMMA are 0.5, 1.7, and 3.7 for the increasing system sizes. The crossover point between the two algorithms is slightly smaller than S2 (i.e., 373 248 waters). For both algorithms, we observe some superlinear acceleration that can be assigned to memory cache effects.

4. CONCLUSIONS

In this paper, we present a novel, highly efficient and massively parallel implementation of the sparse matrix−matrix multiplication algorithm inspired by the midpoint method suitable for matrices with decay. Our so-called MPSM3 algorithm relies on the midpoint method recently proposed by Bowers et al. for the parallelization of particle simulations. We demonstrate how the proposed method can be effectively used for the construction of the density matrix in electronic structure theory, such Hartree−Fock, density functional theory, and semiempirical Hamiltonians. Compared with the state of the art in sparse matrix−matrix multiplications, the MPSM3 algorithm heavily exploits data locality, yielding better performance and scalability, and approaching a perfect linear scaling up to a process box size equal to a characteristic length, that is intrinsic to the matrices. Moreover, the method is able to scale linearly with system size, reaching constant time with proportional resources, also regarding memory consumption. The method has several advantages over the standard SUMMA algorithm, such as (a) reduced communication volume, (b) more effective load balancing, and (c) communication latency (communication with nearby processes only, optimal usage of the Blue Gene/Q torus).

The scalability of the algorithm comes from the reduced volume of interprocess communications and logistic operations, compared to general dense matrix multiplication algorithms. The matrix distribution over processes and the communication pattern of the algorithm follow the interaction structure of the atomic system. It is the locality of interactions that enables this limitation of the total quantity of data sent between processes. On the negative side, the MPSM3 algorithm shows a very good scalability as long as only one push is performed in each direction. Compared to SUMMA, this may constitute an important limitation, specifically for denser matrices or/and for small system sizes.

For further developments, improved variants of the midpoint method introduced by Bowers[30] can also be used in conjunction with the MPSM3 algorithm to balance the computational load across the processes even more evenly. Moreover, the product of symmetric matrices can easily be handled with the MPSM3 with a more-dedicated implementation. Because the SP2 algorithm used to build the (symmetric) density matrix requires the operation $C_{sym} \leftarrow \alpha A_{sym}^2 + \beta C_{sym}$ and the use of only symmetric matrices, it would be advantageous to design a specific matrix multiply for this particular case that has (a) reduced memory usage, (b) reduced communication load, and (c) a reduced number of arithmetic operations.

AUTHOR INFORMATION

Corresponding Author

*Tel.: +41 (0) 44 724 8286. Fax: +41 (0) 44 724 8958. E-mail: vwe@zurich.ibm.com.

Notes

The authors declare no competing financial interest.

REFERENCES

(1) McWeeny, R. *Rev. Mod. Phys.* **1960**, *32*, 335−369.
(2) Palser, A. H. R.; Manolopoulos, D. E. *Phys. Rev. B* **1998**, *58*, 12704−12711.
(3) Niklasson, A. M. N. *Phys. Rev. B* **2002**, *66*, 155115.
(4) Li, X.-P.; Nunes, R. W.; Vanderbilt, D. *Phys. Rev. B* **1993**, *47*, 10891−10894.
(5) Daw, M. S. *Phys. Rev. B* **1993**, *47*, 10895−10898.
(6) Challacombe, M. *Comput. Phys. Commun.* **2000**, *128*, 93−107.
(7) Borštnik, U.; VandeVondele, J.; Weber, V.; Hutter, J. *Parallel Comput.* **2014**, *40*, 47−58.
(8) Kruskal, C. P.; Rudolph, L.; Snir, M. *Theor. Comput. Sci.* **1989**, *64*, 135−157.
(9) Manzini, G. *J. Parallel Distrib. Comput.* **1994**, *21*, 169−183.
(10) Gustavson, F. G. *ACM Trans. Math. Software* **1978**, *4*, 250−269.
(11) McNamee, J. M. *Commun. ACM* **1971**, *14*, 265−273.
(12) Buluç, A.; Gilbert, J. On the representation and multiplication of hypersparse matrices. In *IEEE International Parallel and Distributed Processing Symposium, 2008 (IPDPS 2008)*; IEEE: New York,2008; p 1.
(13) Buluç, A.; Gilbert, J. *SIAM J. Sci. Comput.* **2012**, *34*, C170−C191.
(14) Van De Geijn, R. A.; Watts, J. *Concurr. Comput.* **1997**, *9*, 255−274.
(15) Cannon, L. E. *A Cellular Computer to Implement the Kalman Filter Algorithm*, Ph.D. Thesis No. AAI7010025, Montana State University, Bozeman, MT, 1969.
(16) Bowler, D.; Miyazaki, T.; Gillan, M. *Comput. Phys. Commun.* **2001**, *137*, 255−273.
(17) Hine, N. D. M.; Haynes, P. D.; Mostofi, A. A.; Payne, M. C. *J. Chem. Phys.* **2010**, *133*, 114111.
(18) Rubensson, E. H., Rudberg, E. arXiv, 1501.07800v2.
(19) Persico, M.; Granucci, G.; Inglese, S.; Laino, T.; Toniolo, A. *J. Mol. Struct. THEOCHEM* **2003**, *621*, 119−126.
(20) Toniolo, A.; Ciminelli, C.; Granucci, G.; Laino, T.; Persico, M. *Theor. Chem. Acc.* **2004**, *111*, 270−279.
(21) Inglese, S.; Granucci, G.; Laino, T.; Persico, M. *J. Phys. Chem. B* **2005**, *109*, 7941−7947.
(22) Pople, J. A.; Santry, D. P.; Segal, G. A. *J. Chem. Phys.* **1965**, *43*, S129−S135.
(23) Sustmann, R.; Williams, J. E.; Dewar, M. J. S.; Allen, L. C.; Schleyer, P. v. R. *J. Am. Chem. Soc.* **1969**, *91*, 5350−5357.
(24) Dewar, M. J. S.; Thiel, W. *J. Am. Chem. Soc.* **1977**, *99*, 4899−4907.
(25) Dewar, M. J. S.; Zoebisch, E. G.; Healy, E. F.; Stewart, J. J. P. *J. Am. Chem. Soc.* **1985**, *107*, 3902−3909.

DOI: 10.1021/acs.jctc.5b00382
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

(26) Stewart, J. J. P. *J. Comput. Chem.* **1989**, *10*, 209−220.
(27) Stewart, J. *J. Mol. Model.* **2004**, *10*, 155−164.
(28) Stewart, J. *J. Mol. Model.* **2007**, *13*, 1173−1213.
(29) Stewart, J. *J. Mol. Model.* **2013**, *19*, 1−32.
(30) Bowers, K.; Dror, R.; Shaw, D. *J. Chem. Phys.* **2006**, *124*, 184109.
(31) Bowers, K. J.; Dror, R. O.; Shaw, D. E. *J. Comput. Phys.* **2007**, *221*, 303−329.
(32) Thiel, W. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2014**, *4*, 145−157.
(33) Thiel, W. In *Theory and Applications of Computational Chemistry*; Dykstra, C. E., Frenking, G., Kim, K., Scuseria, G. E., Eds.; Elsevier: Amsterdam, 2005; pp 559−580.
(34) Roothaan, C. C. J. *Rev. Mod. Phys.* **1949**, *23*, 68−89.
(35) Fennell, C. J.; Gezelter, J. D. *J. Chem. Phys.* **2006**, *124*, 234104.
(36) McCann, B. W.; Acevedo, O. *J. Chem. Theory Comput.* **2013**, *9*, 944−950.
(37) Golub, G. H.; Van Loan, C. F. *Matrix Computations*, 3rd Ed.; Johns Hopkins University Press: Baltimore, MD, USA, 1996.
(38) Cawkwell, M. J.; Niklasson, A. M. N. *J. Chem. Phys.* **2012**, *137*, 134105.

DOI: 10.1021/acs.jctc.5b00382
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX