DB-SpGEMM: A Massively Distributed Block-Sparse Matrix-Matrix Multiplication for Linear-Scaling DFT Calculations

Zhong Zheng
University of Science and Technology of China
Hefei, China
zhengz722@mail.ustc.edu.cn

Junshi Chen*
University of Science and Technology of China
Hefei, China
Laoshan Laboratory
Qingdao, China
cjuns@ustc.edu.cn

Yang Zhao
University of Science and Technology of China
Hefei, China
yang_zhao@mail.ustc.edu.cn

Longsheng Song
University of Science and Technology of China
Hefei, China
sls_ustc@mail.ustc.edu.cn

Xinming Qin
University of Science and Technology of China
Hefei, China
xmqin03@ustc.edu.cn

Hong An
University of Science and Technology of China
Hefei, China
Laoshan Laboratory
Qingdao, China
han@ustc.edu.cn

ABSTRACT

Linear-scaling $O(N)$ density functional theory (DFT) represents a significant advancement in the field of computational materials science, especially for simulations of large systems where traditional cubic-scaling methods become computationally prohibitive. The core operation in $O(N)$ methods is sparse general matrix-matrix multiplication (SpGEMM), which is the major performance bottleneck. To enhance the computational efficiency of SpGEMM, it is crucial to consider the inherent sparse pattern of these matrices. Targeting block-sparse matrices with moderate block sizes and regular block shapes, we have developed a distributed block-sparse matrix-matrix multiplication (DB-SpGEMM) algorithm for large-scale DFT calculations. Through deep optimizations in distributed matrix storage, computational task decomposition, asynchronous task scheduling, and load balancing, we have implemented a linear-scaling method based on this algorithm within the discontinuous Galerkin density functional theory (DGDFT). On the new Sunway supercomputer, our approach achieves a 8 ∼ 10x speedup compared to the original version on monolayer phosphorene systems, and demonstrates superior scalability.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**;
• **Applied computing** → *Chemistry*.

KEYWORDS

Density functional theory, block-sparse matrix, matrix-matrix multiplications, new-generation sunway supercomputer

ACM Reference Format:
Zhong Zheng, Junshi Chen, Yang Zhao, Longsheng Song, Xinming Qin, and Hong An. 2024. DB-SpGEMM: A Massively Distributed Block-Sparse Matrix-Matrix Multiplication for Linear-Scaling DFT Calculations. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3673038.3673159

*Corresponding author.

ICPP '24, August 12–15, 2024, Gotland, Sweden

ACM ISBN 979-8-4007-1793-2/24/08
https://doi.org/10.1145/3673038.3673159

1 INTRODUCTION

Electronic structure calculations based on Kohn-Sham (KS) density functional theory (DFT) [17, 24] are widely used in computational chemistry, condensed matter physics, and other fields for the accurate simulation of metallic systems with thousands of atoms. However, as the number of atoms increases from thousands to hundreds of thousands or even millions, the memory requirements and computational time of the traditional cubic-scaling direct diagonalization method become prohibitive, leading to serious performance bottlenecks. Consequently, many DFT codes have adopted low or linear-scaling methods to mitigate the high computational overhead in large-scale simulations.

Among these DFT codes, the DGDFT [18–20] has made a series of innovations in numerical algorithms and data communication. It was nominated for the 2022 Gordon Bell Prize for achieving high-precision electronic structure calculations of 2.5 million atoms on the new Sunway supercomputer. While the PEXSI [26] solver used by DGDFT reduces the computational complexity from $O(N^3)$ to $O(N^{1.5\sim2})$, providing a significant advantage for metallic systems, linear-scaling methods are often preferred for large-scale simulations in both molecular and condensed systems.

Linear-scaling methods are grounded in the locality principle in quantum mechanics [32], confine computations to local regions rather than the entire system. This ensures that the information

required to solve the KS equations scales linearly with the system size. These methods, such as divide-and-conquer [37], density matrix minimization [10], and density matrix purification [30], avoid direct diagonalization by computing the density matrix. Typically, they employ localized basis sets like Gaussian-type orbitals (GTO) or numerical atomic orbitals (NAO) [35]. The use of localized basis sets results in a block-sparse KS matrix, characterized by non-zero elements concentrated in irregularly shaped dense matrix blocks. The computation of density matrix effectively translates to operations on sparse matrices, with the primary performance bottleneck being sparse general matrix-matrix multiplication (SpGEMM). Although the computational complexity is reduced, communication and other issues make it challenging to parallelize SpGEMM and maintain good scalability in large-scale computations.

To enhance computational efficiency, numerous SpGEMM algorithms have been developed, focusing on reducing communication or innovating storage formats. These algorithms are integral to massively parallel libraries such as NTPoly [11], DBCSR [6, 34], and Combinatorial BLAS [7]. Some libraries have optimized data structures for block-sparse matrices, but they face limitations in algorithm design when dealing with irregular dense matrix blocks. Therefore, when dealing with block-sparse matrices that have regularly shaped and relatively large matrix blocks, these algorithms cannot fully exploit this sparsity pattern.

On the other hand, with the rapid development of high-performance computing technologies, an increasing number of DFT codes are being deployed on large-scale heterogeneous computing systems [9, 33]. This trend places greater demands on the design of SpGEMM algorithms. Many libraries exploit parallelism that cannot fully utilize the computational capabilities of modern supercomputer. Thus, achieving efficient linear-scaling DFT calculations faces challenges not only in algorithm design but also in ensuring compatibility with the system architecture. Without addressing these challenges effectively, the advantages of linear-scaling methods cannot be fully realized.

The main contributions of this work can be summarized as follows:

(1) We propose DB-SpGEMM for block-sparse matrix-matrix multiplication, enabling large-scale DFT calculations on the heterogeneous supercomputer.
(2) We design a distributed storage method for block-sparse matrices with regularly shaped blocks and preformed a computational task decomposition based on that.
(3) To hide communication and parallelize computational tasks, we design a runtime on the Sunway many-core architecture capable of handling tasks with dependencies.
(4) Developing optimization techniques to achieve better load balancing and reduce solution time without sacrificing accuracy.
(5) We simulate monolayer phosphorene systems with up to 35,840 atoms, achieving better parallel efficiency on more than 4 million cores and a significant speedup in single SpGEMM.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 provides relevant background knowledge. Section 4 details the main components of DB-SpGEMM, including the matrix storage method, computational task decomposition strategy, runtime design, and other optimization techniques. Section 5 presents the experimental results. Finally, Section 6 summarizes our work and outlines prospects for future research.

2 RELATED WORK

In the computation of density matrix, SpGEMM occupies the majority of the processing time, making suitable parallel algorithms crucial for enhancing the performance of linear-scaling DFT calculations. Early work explored various parallelization methods, such as the 2D decomposition-based SpSUMMA and SpCannon algorithms [7]. However, as parallel scales increased, the escalating communication overhead severely impacted scalability. This challenge led to the development of numerous SpGEMM algorithms with communication-avoid as their objective. Ballard et al. [4] optimized on random sparse matrices, proposing a 3D recursive algorithm. Subsequently, Azad et al. [3] tested this 3D algorithm on real world sparse matrices, demonstrating through analysis that compared to 2D decomposition, it effectively reduces communication overhead in large-scale parallelism, thereby enhancing scalability.

In addition to communication optimization, some work seek to leverage spatial structure in sparse matrices to enhance computational intensity. For instance, the DBCSR library used in quantum chemistry code CP2K [21] stores matrices in blocked compressed sparse row (CSR) format and has designed specialized library for small matrix block multiplications on GPUs. Lazzaro et al. [25] built upon the DBCSR library, replacing MPI point-to-point communication with one-sided communication to propose a 2.5D algorithm. The NTPoly library combines the 3D SpGEMM with DBCSR, introducing non-blocking communication and MPI+OpenMP hybrid parallelism. NTPoly utilize this algorithm as the core matrix multiplication routine, achieving various linear-scaling matrix function calculations and quantum chemistry methods. Chen et al. [8] achieved large-scale DFT calculations in FHI-aims [5] using optimized NTPoly on the new Sunway supercomputer. Additionally, Herault et al. [15] researched block-sparse data types on GPUs, optimizing electron structure simulation applications on the Summit supercomputer using the PaRSEC runtime [14].

3 BACKGROUND

3.1 Kohn–Sham Density Functional Theory

The goal of the DFT calculations is to solve the Kohn-Sham (KS) equations, which correspond to the following eigenvalue problem

$$\widehat{H}\psi_i(\mathbf{r}) = \epsilon_i\psi_i(\mathbf{r}) \tag{1}$$

where $\widehat{H}$ is the Hamiltonian operator, ψ_i is the i-th KS orbit, and ϵ_i is the corresponding orbital energy. Prior to solving the KS equations, it is customary to discretize them by expressing each KS orbital as a linear combination of a set of basis functions as

$$\psi_i(\mathbf{r}) = \sum_{\mu}^{N_b} \phi_\mu(\mathbf{r})C_{\mu i} \tag{2}$$

where $C_{\mu i}$ is the expansion coefficient at the μth atomic orbital and N_b is the number of basis set. Subsequently, the KS equations can be rewritten into matrix notations as

$$HC = SCE \tag{3}$$

where C is the coefficient matrix, H are the Hamiltonian matrices with $H_{\mu\nu} = \langle\phi_\mu|\hat{H}|\phi_\nu\rangle$, S are the overlap matrices with $S_{\mu\nu} = \langle\phi_\mu|\phi_\nu\rangle$, and E is the diagonal matrix consisting of eigenvalues ϵ_i. Solving the above eigenvalue problem by a conventional diagonalization method usually has a computational cost that scales $O(N^3)$, which hinders the application of DFT to large systems.

The single-particle density matrix $P = CC^T$ offers a significant alternative to directly solving the eigenvalue problem in traditional cubic-scaling DFT methods. Density matrix formalism in a local basis has become a key ingredient of $O(N)$ DFT calculations and can be determined by using linear-scaling methods.

3.2 DGDFT Methodology

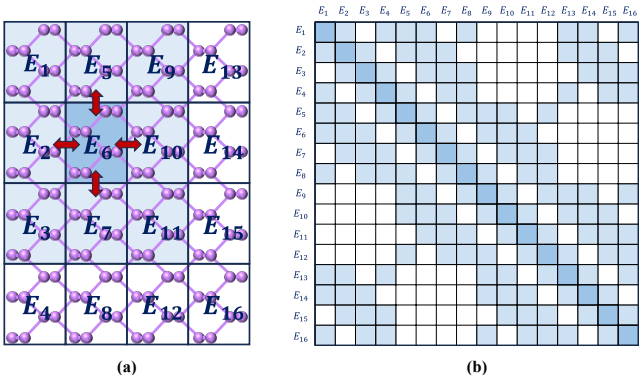

Figure 1: (a) A monolayer phosphorene system partitioned into 4×4 elements; (b) The block-sparse DG Hamiltonian matrix.

The DG method [1, 2] partitions the global computational domain into a number of subdomains, referred to as elements, as shown in Fig.1(a). Each adaptive local basis (ALB) function of DGDFT is strictly localized within a certain element in the real space and is discontinuous from the perspective of the global domain. This results in a block-sparse Hamiltonian matrix (Fig.1(b)) in each self-consistent field (SCF) iteration, and the non-zero matrix blocks correspond to interactions between neighboring elements are shown in Fig.1(a). These matrix blocks are significantly larger and denser compared to conventional cases.

3.3 Density Matrix Purification Method

The density matrix P can be accurately approximated through the matrix function of the Hamiltonian matrix H. In this work, we employed the 4th order trace resetting (TRS4) density matrix purification proposed by Niklasson et al [29], as shown in Algorithm 1. Since the DG Hamiltonian matrix is orthogonal, the overlap matrix is actually an identity matrix, so we can simplify the algorithm to some extent. TRS4 constructs the initial value P_0 of P using H and then iteratively updates P_n until convergence. The TRS4 method defines a parameter domain within which, when the parameter γ_n lies, it employs a linear combination of a pair of polynomials, $F(x)$ and $G(x)$, to update P_n. If γ_n exceeds the defined parameter domain, it switches to the update approach of the 2nd order trace-correcting (TC2) purification [28].

Algorithm 1 The pseudocode for the TRS4 method

Input: Hamiltonian matrix H, the number of electrons N_e

Compute the eigenvalue $\varepsilon_{\max}$ and $\varepsilon_{\min}$ of H
$P_0 \leftarrow (\varepsilon_{\max} I - H)/(\varepsilon_{\max} - \varepsilon_{\min})$
for $n = 1$ to max_iterations **do**
  $F(P_n) \leftarrow -3P_n^4 + 4P_n^3$; $G(P_n) \leftarrow P_n^4 - 2P_n^3 + P_n^2$
  Compute the parameter γ_n according to N_e
  if $\gamma_n > 6$ **then**
    $P_{n+1} \leftarrow 2P_n - P_n^2$
  else if $\gamma_n < 0$ **then**
    $P_{n+1} \leftarrow P_n^2$
  else
    $P_{n+1} \leftarrow F(P_n) + \gamma_n G(P_n)$
  end if
  Compute the total energy e, exit the loop if e converged
end for

Output: Density matrix P, total energy e

3.4 3D Matrix Partitioning

The DB-SpGEMM algorithm we propose adopts a matrix parallel partitioning scheme similar to the 3D algorithm [3]. Taking $C = A \times B$ as an example, Fig.2 illustrates the mechanism of 3D matrix partitioning. All processes are organized into a three-dimensional topology, establishing communication domains among processes in the same row, column, and layer. The input matrices A and B are reused across different layers of processes, with each layer responsible for computing a portion of the resulting matrix C.

During the computation, each process sends its a portion of matrix A to processes within the same row and a portion of matrix B to processes within the same column. After communication, each process integrates the received data and performs local matrix multiplication. Finally, processes in different layers with the same row and column positions perform inter-layer communication to aggregate and obtain the final result of matrix C.

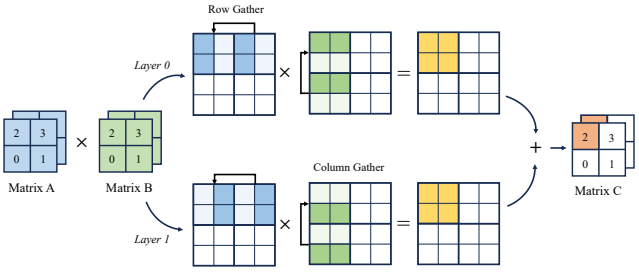

Figure 2: 3D SpGEMM

3.5 The New Sunway Architecture

The new Sunway supercomputer is the upgraded version of the Sunway Taihulight [12]. The architecture of each computing node

is shown in the Fig.3, which consists of a sw26010pro many-core processor with 6 core groups (CGs) and 96 GB memory. Each CG contains a Management Processing Element (MPE) and 8 × 8 Computing Processing Elements (CPEs), which are connected to each other through an on-chip network. Each CPE has a 256 KB Local Data Memory (LDM), and the data is transferred between LDM and main memory via direct memory access (DMA).

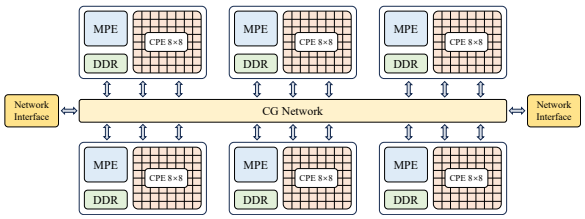

Figure 3: Architecture of sw26010 pro many-core processor

4 ALGORITHM DESIGN AND OPTIMIZATIONS

4.1 Block-Sparse Matrix Storage of DB-SpGEMM

To fully exploit the spatial structure of block-sparse matrices, we designed the distributed block-sparse matrix (DBSM) data structure for DB-SpGEMM algorithm. As shown in Fig4, this data structure evenly partitions the sparse matrix on a two-dimensional process grid and further subdivides it into matrix blocks within each process. For each non-zero block, DBSM uses the dense format (e.g., array) instead of the sparse format (e.g., CSR, CSC) for storage. In DGDFT, the benefits of this design are:

(1) Compatibility with existing data structures.
(2) No need for additional format conversions during data redistribution before and after DFT calculations.
(3) Preservation of the original size and sparsity of the matrix blocks.

It is worth noting that the same rationale applies to other DFT codes that uses dense format to store matrix blocks.

In the NTPoly library's implementation of 3D SpGEMM, if the number of row processes N_R equals the number of column processes N_C, each process needs to use $N_L \times N_L$ blocks (N_L represents the number of layers in the process grid) to store all the local data for matrices A, B, and C. However, in the scenario of DFT, the total number of matrix blocks and the dimension of dense matrix block in block-sparse matrices are independent of the process grid configuration. Assuming that there are a total of $M \times M$ blocks, the data divided among each process will cover $(M/N_R) \times (M/N_C)$ matrix blocks. As long as the smaller value between M/N_R and M/N_C is greater than N_L, the subdivision within each process can follow the original structure of the block-sparse matrix in DBSM. Overall, DB-SpGEMM determines the storage structure of DBSM based on the attributes of the application itself, rather than the process grid configuration unrelated to the application.

In the implementation of DBSM, we dynamically manage the memory occupied by matrix blocks. During the data redistribution phase, we only allocate space for non-zero blocks. For a block that is initially empty, space is allocated only if it generates non-zero elements during computation. If all the non-zero blocks are completely dense at the initial time, DBSM will actually store much less index information than DBCSR. The memory overhead on each process during computation can also be effectively reduced by 3D layering, we just need to control the sparsity of each non-zero block.

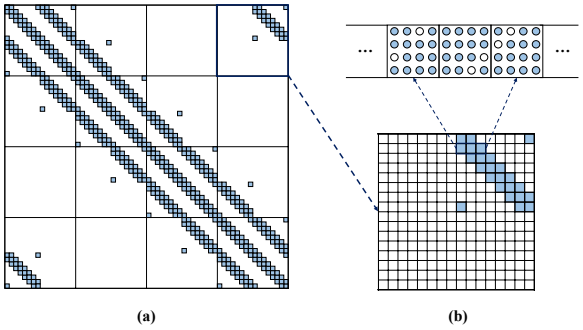

Figure 4: Block-sparse matrix storage: (a) The block-sparse matrix is divided within a 4×4 process grid; (b) The local data on each process is further subdivided into matrix blocks in dense format.

4.2 Computational Task Decomposition

To enhance the performance of DB-SpGEMM in matrix multiplication, we devised a computational task decomposition strategy based on the characteristics of block-sparse matrices. This strategy aims to more effectively utilize high-performance linear algebra libraries such as BLAS, which offer dense general matrix-matrix multiplication (DGEMM) kernels.

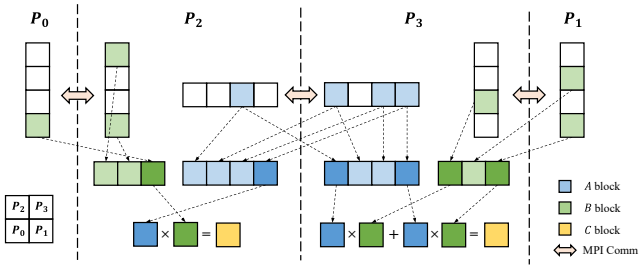

Figure 5: An example of computational task decomposition for $C = A \times B$, where matrices are composed of 8×8 blocks and are divided within a 2×2 process grid, showing the calculation of matrix blocks in processes P_2 and P_3.

To simplify the communication tasks in stages A and B of the workflow shown in Fig.6(a), we directly utilize MPI communication with the dense format non-zero blocks from DBSM. As a result, each process receives two concatenated one-dimensional arrays. Performing matrix multiplication directly on the received data requires not only transposing the two arrays but also padding them with zero elements. Such an operation will introduce additional overhead and significantly impact the performance. As shown in Fig.5, when process P_3 calculates a C block, it first needs to communicate with processes in the same row and column to obtain their

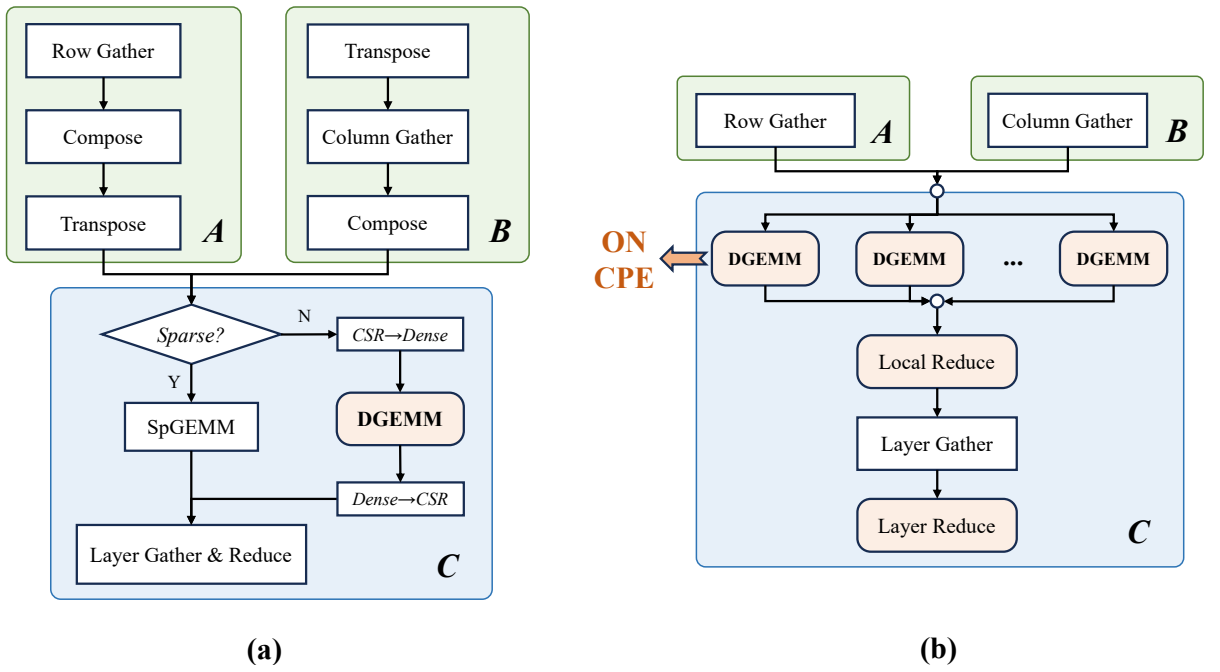

Figure 6: (a) The workflow of 3D SpGEMM in NTPoly, with three stages denoted as A, B, and C. Stages A and B focus on communication, and stage C focuses on matrix multiplication; (b) The optimized workflow of DB-SpGEMM, with components highlighted in light orange, indicates tasks suitable for execution on the CPE cluster.

respective matrix block data. For the combined data, it is evident that not all matrix blocks will participate in the computation. Hence, we decompose the matrix multiplication into a set of independent matrix block multiplications.

To achieve this, we need to record the positions of non-zero blocks in the original matrix. We use a flag array to indicate whether matrix blocks are empty. The indices of this array correspond to the positions of each block. This flag array, along with the matrix data, is sent to other processes in the same row or column. As shown in Algorithm 2, we inspect the two flag arrays to determine if both matrix blocks are non-zero. If either block is empty, we can skip the matrix multiplication at that position. In some cases, no computation may occur after inspection. When the global matrix is very sparse, this approach aids in simplifying the GEMM of two tall-and-skinny matrices into several groups of GEMM involving two small matrix blocks, thus better utilizing data locality.

Algorithm 2 The pseudocode for computational task decomposition

Input: Row flag F^r, column flag F^c, A blocks, B blocks, the number of blocks N

1: **for all** $i = 1$ to N **do**
2: **if** $F_i^r == 1$ & $F_i^c == 1$ **then**
3: Compute $A_i \times B_i$
4: **end if**
5: **end for**

As the matrix multiplication is decomposed, we also need to perform a local sum afterward. Subsequently, communication and reduction with processes from other layers are conducted. If there is only one layer, stage C can be terminated after this step.

4.3 Asynchronous Task Scheduling

In order to exploit more parallelism of DB-SpGEMM, fully utilize the computational capabilities of CPEs on the Sunway many-core architecture, and achieve better cooperation between MPE and CPEs, we design a runtime for DB-SpGEMM, as shown in Fig.7(b), which is used to support the asynchronous execution and scheduling of communication tasks and computational tasks. Our runtime is based on the classical dataflow model [23], where tasks are abstracted into a directed acyclic graph (DAG). Each task maintains a dependency count equal to the number of its predecessor tasks, and a task can only be executed when all its predecessors have completed.

The runtime starts two threads on the MPE: MPE Worker and CPE Worker. The former is responsible for starting and executing communication tasks on the MPE, while the latter is responsible for scheduling computational tasks on the CPE cluster, as well as allocating and recycling resources. Associated with these two threads are two concurrent queues that manage the tasks being executed on the MPE and CPE, respectively, in addition to a concurrent queue that manages all the tasks waiting to be executed. All tasks are created dynamically during runtime. When a task is created, it can be executed immediately if the dependency count is zero. Otherwise, it waits for all predecessor tasks to be completed before execution.

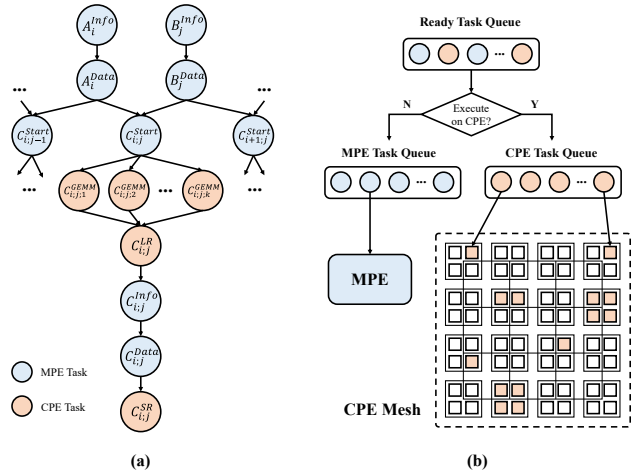

Figure 7: Task scheduling of DB-SpGEMM: (a) DAG of tasks; (b) Runtime system design on the Sunway many-core architecture.

Since the number of A, B, and C blocks set by default for each process is $(M/N_R) \times (M/N_C)$, DB-SpGEMM will generate M/N_R A tasks, M/N_C B tasks, and $(M/N_R) \times (M/N_C)$ C tasks. We refer to specific tasks as A_i, B_j and $C_{i;j}$. For A_i and B_j, we split them into two tasks as shown in Fig.7(a), A_{i_Info} and B_{j_Info} are responsible for communicating the size and position information of the block, and A_{i_Data} and B_{j_Data} are responsible for communicating the data. Since A_{i_Info} and B_{j_Info} still need to do some preprocessing work first, using the MPI non-blocking communication will help us to hide the communication within these processes.

The task $C_{i;j}$ can be split into six sub-tasks. It begins with $C_{i;j\_Start}$, which ensures that both A_i and B_j have been completed, and also clears the memory and handling the flag arrays in preparation for computational task decomposition. Next is the computational tasks $C_{i;j;k\_GEMM}$, where the number of these task corresponds to the number of matrix block multiplications after decomposition. It is evident that there is no dependency between these multiplications, allowing them to be performed in parallel. Without the runtime, the CPE cluster would have to compute them sequentially, which limits the use of the CPE cluster's computational capabilities and DMA bandwidth. With the runtime, different $C_{i;j;k\_GEMM}$ can be dispatched to the CPE cluster nearly simultaneously, allowing up to 64 tasks to be in progress at any given time.

$C_{i;j;k\_GEMM}$ is followed by $C_{i;j\_LR}$, which is responsible for summing the results of all the matrix block multiplications and this can also be performed on the CPE. If the process grid is configured in 3D, $C_{i;j\_LR}$ is followed by communication tasks $C_{i;j\_Info}$ and $C_{i;j\_Data}$ similar to those in A_i and B_j. These tasks also use MPI non-blocking communication to prevent thread blocking and allow other tasks to start. Finally, a computational task $C_{i;j\_LR}$ similar to $C_{i;j\_SR}$ will be completed.

With such a runtime design and non-blocking communication, we can maximize the parallelism in DB-SpGEMM, eliminating the need for serialization of different A_i, B_j, and $C_{i;j}$ tasks. Moreover, the primary computational bottleneck $C_{i;j;k\_GEMM}$ can also be parallelized using the CPE cluster. Currently, considering the difficulty of the runtime design and the size of matrix blocks, we only utilize a single CPE to execute one multiplication [22].

4.4 Optimization

4.4.1 Block-based Load Balancing. Due to the concentration of non-zero elements along the diagonal direction in sparse matrices used in linear-scaling DFT calculations, directly partitioning the matrix in two dimensions can result in uneven distribution of nonzeros among processes. This imbalance can lead to uneven computational loads across processes, causing certain processes to become bottlenecks and affecting overall performance. A conventional approach is to perform random permutations on each row and column. However, in DB-SpGEMM, transforming block-sparse matrices into completely random sparse matrices and storing them in dense format would consume a lot of extra memory, significantly impact scalability. Therefore, we propose a block-based load balancing strategy to redistribute data more effectively.

Similar to the conventional approach, block-based load balancing involves left and right multiplication of two random permutation matrices with the original matrix. However, we perform permutations at the level of matrix block rows or block columns. The random permutation matrices we construct have the same number of matrix blocks as the original matrix. Each block row or block column containing only one unique matrix block as the identity matrix, while all other blocks are empty. The mathematical formulation of this process is as follows:

$$A' = PAP^T = \begin{bmatrix} 0 & 0 & I \\ I & 0 & 0 \\ 0 & I & 0 \end{bmatrix} \cdot \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & A_{23} \\ 0 & A_{32} & A_{33} \end{bmatrix} \cdot \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \\ I & 0 & 0 \end{bmatrix} = \begin{bmatrix} A_{33} & 0 & A_{32} \\ 0 & A_{11} & A_{12} \\ A_{23} & A_{21} & A_{22} \end{bmatrix} \quad (4)$$

After the calculation of density matrix is completed, the inverse permutation can be achieved through $P^T A' P$, restoring the matrix to its original data distribution. This approach helps balance the computational load across different processes without affecting the design of DBSM or the computational task decomposition strategy. Additionally, since only four extra sparse matrix-matrix multiplications are generated in total, in density matrix computations requiring multiple iterations, the benefits of the load balancing strategy are sufficient to offset these additional overheads.

4.4.2 Two-level On-the-fly Filtering. To control the overall sparsity of block-sparse matrix and the sparsity of each matrix block during the iterations, we have designed a two-level on-the-fly filtering procedure.

The reason SpGEMM achieves linear-scaling is because the number of non-zero elements exhibits an $O(N)$ relationship with the matrix size, rather than the $O(N^2)$ relationship seen in dense matrices. However, in the iterations, after multiple matrix multiplications, the sparsity of density matrix significantly decreases, increasing the computational complexity. The traditional approach is to apply a numerical threshold δ_n to filter out all non-zero elements with an absolute value smaller than this threshold after each matrix multiplication [36]. This method effectively mitigates the decrease in global matrix sparsity but also reduces the sparsity of each matrix block, thereby affecting the computational efficiency of the DGEMM kernel.

Ideally, we aim for each matrix block to be as dense as possible. Thus, after filtering out non-zero elements within each matrix block, as shown in Algorithm 3, we also check the density of matrix block. If the density is below a specified threshold δ_d, we discard all non-zero elements within the entire block.

Algorithm 3 The pseudocode for the two-level filtering

Input: Matrix block M, the size of matrix block S, the numerical threshold δ_n, the density threshold δ_d

```
Initialize counter c ← 0
for i = 1 to S do
    if |M_i| > δ_n then
        c ← c + 1
    else
        M_i ← 0
    end if
end for
if c/S <= δ_d then
    Clear the matrix block M
end if
```

Since the convergence of the iterations relies on evaluating the total energy, which involves taking the dot product of the Hamiltonian matrix H with the density matrix P_n and then accumulating the result, it follows that newly appearing non-zero elements in P_n do not affect the energy calculation. The non-zero elements that impact the energy calculation are mainly concentrated near the diagonal, where their values are relatively larger and their corresponding blocks are denser, making them less affected by δ_d. Therefore, filtering out some sparser blocks has minimal impact on accuracy and can actually reduce solution time by decreasing the number of matrix-matrix multiplications.

5 NUMERICAL RESULTS AND ANALYSIS

5.1 Setup of the Test Physical Systems and Testing Environment

We select monolayer phosphorene as our test system. Phosphorene is a two-dimensional semiconductor material composed of ordered phosphorus atoms, holding vast potential applications in field-effect transistors, optoelectronic devices, solar cells, and more. Our tests encompass various scales ranging from 560 to 35,840 atoms. All tests are conducted using the DGDFT with double precision on the new Sunway supercomputer. We initiate six MPI tasks per computing node, with each task bound to a CG.

5.2 Numerical Accuracy

We initially analyze the numerical accuracy of the density matrix purification method. Using the results obtained from direct diagonalization as a benchmark, we compare them with those computed using TRS4 method based on DB-SpGEMM. We utilize the total energy error per atom ΔE as the evaluation metric, defined as

$$\Delta E = |E^{TRS4} - E^{DIAG}|/N_A \quad (5)$$

where N_A represents the total number of atoms. We select four systems of different scales, namely Ph_{140}, Ph_{560}, Ph_{2240}, and Ph_{3500},

Table 1: Total energy error per atom with respect to various filter of four phosphorene systems

Systems	ΔE (filter=10^{-8})	ΔE (filter=10^{-7})	ΔE (filter=10^{-6})
Ph_{140}	4.28×10^{-7}	6.85×10^{-6}	4.85×10^{-4}
Ph_{560}	5.36×10^{-7}	3.63×10^{-5}	1.03×10^{-3}
Ph_{2240}	7.59×10^{-7}	1.71×10^{-5}	2.53×10^{-3}
Ph_{3500}	1.43×10^{-7}	2.95×10^{-5}	3.81×10^{-3}

where the subscript denotes the number of atoms. For each system, we test three different numerical thresholds δ_n of 10^{-6}, 10^{-7}, and 10^{-8}. The convergence tolerance for iterations is set to 10^{-6}. The experimental results are presented in Table 1, where it can be observed that as δ_n decreases gradually, ΔE reaches the order of 10^{-7} eV/atom, a precision level acceptable for linear-scaling methods.

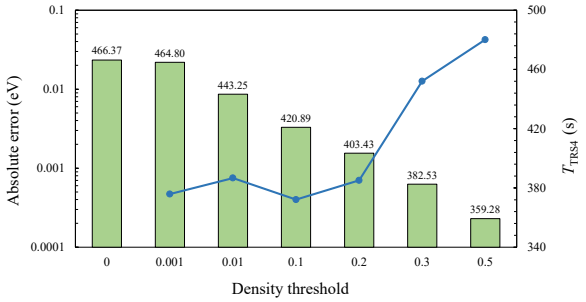

Figure 8: Time to solution and the total energy absolute error relative to $\delta_d = 0$ at different density thresholds δ_d.

We also examine the impact of two-level on-the-fly filtering on accuracy and solution time. For this analysis, we utilize Ph_{3500} as the test system, employing 6,400 processes for computation and only measuring the time taken for computing the density matrix. δ_n is set to 10^{-8}, and density threshold δ_d is varied from 0 to 0.5 in seven steps. The results are shown in Fig.8. It can be observed that as δ_d increases to 0.2, the absolute error in total energy E^{TRS4} remains within the order of 10^{-4} eV/atom to 10^{-3} eV/atom, with the impact on ΔE still within the order of 10^{-7} eV/atom. However, the total computation time decreases by approximately 15% compared to when δ_n is not set. Based on our statistics, when δ_n is set to 0.5, up to 15% of non-zero elements are additionally filtered out in a single iteration, and at this point, the impact on accuracy becomes more noticeable. Utilizing this filtering method allows us to combine δ_d and δ_n, enabling us to find an optimal configuration tailored to the requirements of the simulated system.

5.3 Speedup

The performance results of DB-SpGEMM when computing once $C = A \times A$ is shown in Fig.9. We compare the speedup of the runtime version (discussed in subsection 4.3) and the baseline version without runtime against the MPE version (using LAPACK 3.8.0), both of them utilize MPI non-blocking operations to hide communication. In the baseline version, we utilize the xMath BLAS library [27],

which is optimized for the Sunway many-core architecture. As the xMath use a CG's entire CPE cluster for each multiplication, the DGEMM needs to be performed serially.

We choose two different sparse matrix sizes: $48,000 \times 48,000$ and $96,000 \times 96,000$, both exhibiting a conventional block-tridiagonal structure as shown in Fig.9. The block-sparse matrix is divided into 400×400 matrix blocks in two dimension. We adjust the sparsity from 84% to 98% by varying the number of non-zero blocks, specifically initializing a certain number of matrix blocks as fully dense, while leaving the remaining matrix blocks empty. All tests are conducted using a 20×20 process grid in two dimensions.

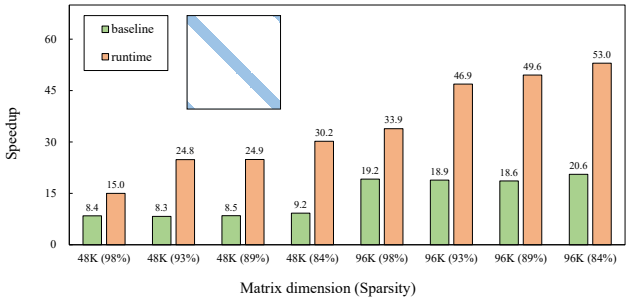

Figure 9: The many-core speedup of the baseline version and the runtime version under different matrix sizes and sparsity levels.

From the results in Fig.9, it is clear that at the same sparsity level, both versions show better acceleration as the matrix size increases . Although the number of matrix blocks allocated per process remains constant, the size of each matrix block increases, enabling more efficient utilization of local LDM space and DMA bandwidth. Conversely, at the same matrix size, the speedup of the baseline version remains consistent despite decreasing sparsity. This is primarily due to load imbalance leading to certain processes becoming bottlenecks. In contrast, the runtime version can execute computational tasks in parallel, reducing susceptibility to individual processes becoming bottlenecks. Moreover, as the matrix density increases, the runtime version can increase the number of concurrent computational tasks, resulting in improved speedup.

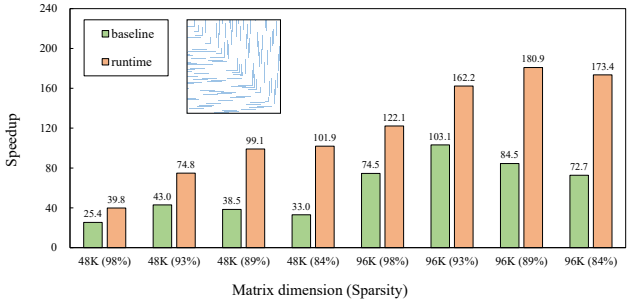

Figure 10: The many-core speedup of the baseline version and the runtime version with block-based load balancing.

We also analyze the impact of block-based load balancing on performance, as shown in Fig.10. Across the four configurations, there is an average performance improvement of approximately 3x when load balancing is employed. The speedup of the runtime version reach a maximum of 181x. This improvement is partly attributed to the computational capabilities of the 64 CPEs in a CG and partly to the runtime system's ability to schedule asynchronous tasks more effectively under load balancing strategy. Overall, the runtime version exhibit an average speedup of 2x over the baseline version, with this effect becoming more pronounced as matrix size increased and sparsity decreased.

In addition to comparing the speedup between different versions of DB-SpGEMM to validate the feasibility of runtime for performance enhancement, we also compare it with NTPoly implemented on the new Sunway supercomputer. In this experimental setup, we focus primarily on sparsity of 89% and 84%. This choice is made because in the current version of NTPoly, direct utilization of xMath library for many-core acceleration is not feasible when sparsity exceeds a certain threshold, resulting in significantly slows down compared to DB-SpGEMM. To better evaluate the two algorithms, we opt for lower sparsity levels for testing. Both DB-SpGEMM and NTPoly utilize load balancing strategy. All experiments are conducted using a 40×40 process grid.

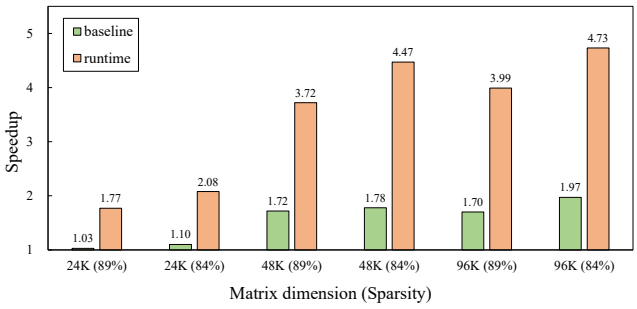

Figure 11: The speedup of DB-SpGEMM compared to NTPoly across various matrix sizes and sparsity levels.

As shown in Fig.11, the baseline version of DB-SpGEMM offers limited improvement over 3D SpGEMM. However, with the use of runtime system, there is a noticeable performance boost. This improvement becomes more pronounced as the matrix size increases and the sparsity decreases.

5.4 Scalability Test

For the strong scalability experiments, we choose Ph_{3500} and Ph_{14000} as the test systems, with the number of matrix blocks are 400×400 and 1600×1600, respectively. The size of matrix blocks is 240×240, and the δ_n is set to 10^{-8}. We compare the TRS4 method implemented by NTPoly and DB-SpGEMM. As in previous experiments, only the time to compute the density matrix is considered, and only the first ten rounds of TRS4 iterations are counted. Both implementations utilize CPEs for acceleration. However, it is important to note that due to the design of NTPoly, computations on a small number of processes during the iterations are completed solely on the MPE. The experimental results are shown in Fig.12. In Ph_{3500}, our implementation achieves an average speedup of 10x over NTPoly, and an average speedup of 8x in Ph_{14000}. In terms of parallel

efficiency, our proposed method maintains 67.1% parallel efficiency when scales to 64,000 processes (4,160,000 cores) compared to using 12,800 processes in Ph_{14000}, while NTPoly has 57.5% parallel efficiency.

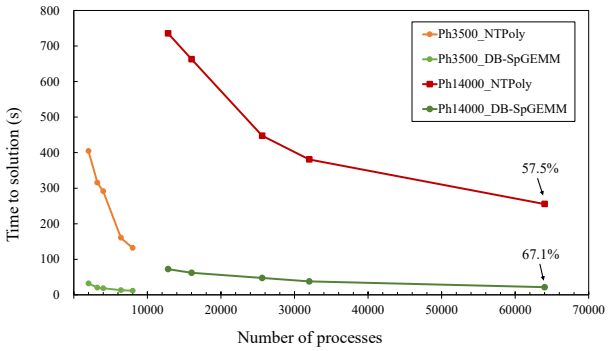

Figure 12: The strong scaling of the TRS4 method implemented based on DB-SpGEMM and NTPoly.

Fig.13 shows the weak scaling of DGDFT in computing the density matrix. We test five systems Ph_{3500}, Ph_{8960}, Ph_{14000}, Ph_{20160}, and Ph_{31500}, with corresponding process counts of 16,000, 40,960, 64,000, 92,160, and 144,000. In our tests, NTPoly could not complete the computation correctly at certain scales, so we compare the results with those from PEXSI currently used by DGDFT. Since PEXSI uses a different computational method than TRS4, we select a certain percentage of iterations to count the solution time. The results show that DB-SpGEMM has an advantage in solution time across all system sizes and exhibits better weak scaling, our proposed method can still effectively solve the density matrix by increasing the number of processes, even at very large system sizes.

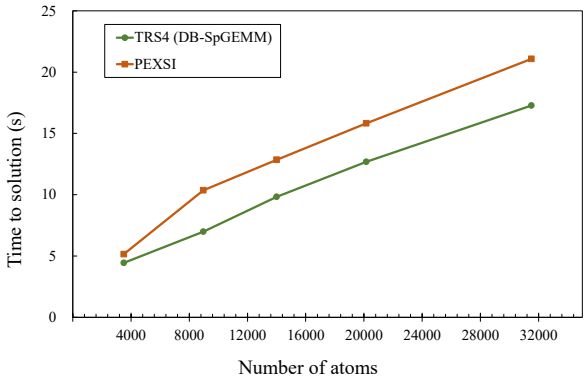

Figure 13: The weak scaling of computing the density matrix across different system sizes and the number of processes.

In addition, to validate the linear-scaling of the TRS4 method, we keep the number of processes fixed at 4,096 and increase the number of atoms. Since the computational complexity of PEXSI in two-dimensional systems scales as $O(N^{1.5})$, the difference between PEXSI and linear-scaling methods is clearly shown in Fig.14. As the system size increases, the time required for PEXSI calculations diverges significantly from linear-scaling methods.

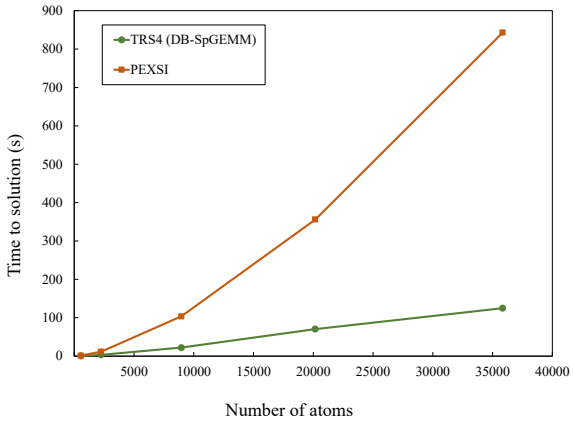

Figure 14: The linear-scaling of the TRS4 method

6 CONCLUSION AND FUTURE WORK

In this work, we propose the DB-SpGEMM algorithm for massively distributed block-sparse matrix-matrix multiplication, building upon the foundation of the 3D SpGEMM. Leveraging this algorithm, we achieve large-scale DFT calculations on the new Sunway supercomputer. Our approach utilizes a distributed matrix storage method, avoiding redundant format conversions. Through computational task decomposition and a runtime system, we fully utilize the computational resources of the many-core processors. Experimental results demonstrate that with runtime support, our algorithm achieves an additional 2 ~ 3x speedup. Compared to NTPoly, the TRS4 method implemented based on DB-SpGEMM in DGDFT shows significant performance improvements, maintaining 67.1% parallel efficiency when scales to 4,160,000 cores.

Our proposed method is not limited to the Sunway architecture, it can be deployed on general platforms similarly to the NTPoly library, utilizing MPI+OpenMP for acceleration. On CPU platforms, it is necessary to appropriately adjust the task granularity by merging matrix blocks to avoid the overhead of frequent thread switching. On GPU platforms, DB-SpGEMM can be accelerated using libraries like cuBLAS and rocBLAS, similar to the baseline version with xMath library. Computational task decomposition can be implemented using batched DGEMM interface. However, to achieve finer-grained control like that provided by the runtime system, we need to design the runtime and adapt the DB-SpGEMM algorithm for GPU platforms in the future.

Although we only focus on DFT calculations in this work, DB-SpGEMM is actually applicable to other fields involving block-sparse matrix-matrix multiplication. For instance, it can be used in some computational fluid dynamics (CFD) [31] and electromagnetics applications [16] that also employ the DG method. Additionally, the deep learning applications also deals with similar block-sparse matrices [13]. In the future, we plan to extend this algorithm to other domains, making it more versatile.

ACKNOWLEDGMENTS

This work is partly supported by the Strategic Priority Research Program of Chinese Academy of Sciences (XDB0500102) and the National Natural Science Foundation of China (Grant No. 62102389). The computing resources are financially supported by Laoshan Laboratory (LSKJ202300305). We thank Prof. Wei Hu (University of Science and Technology of China) and Prof. Wenjing Ma (Institute of Software, Chinese Academy of Sciences) for helpful discussions. We also appreciate the reviewers for dedicating their time and providing constructive comments on this paper.

REFERENCES

[1] Douglas N Arnold. 1982. An interior penalty finite element method with discontinuous elements. *SIAM journal on numerical analysis* 19, 4 (1982), 742–760.

[2] Douglas N Arnold, Franco Brezzi, Bernardo Cockburn, and L Donatella Marini. 2002. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM journal on numerical analysis* 39, 5 (2002), 1749–1779.

[3] Ariful Azad, Grey Ballard, Aydin Buluc, James Demmel, Laura Grigori, Oded Schwartz, Sivan Toledo, and Samuel Williams. 2016. Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication. *SIAM Journal on Scientific Computing* 38, 6 (2016), C624–C651.

[4] Grey Ballard, Aydin Buluc, James Demmel, Laura Grigori, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. 2013. Communication optimal parallel multiplication of sparse random matrices. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*. 222–231.

[5] Volker Blum, Ralf Gehrke, Felix Hanke, Paula Havu, Ville Havu, Xinguo Ren, Karsten Reuter, and Matthias Scheffler. 2009. Ab initio molecular simulations with numeric atom-centered orbitals. *Computer Physics Communications* 180, 11 (2009), 2175–2196.

[6] Urban Borštnik, Joost VandeVondele, Valéry Weber, and Jürg Hutter. 2014. Sparse matrix multiplication: The distributed block-compressed sparse row library. *Parallel Comput.* 40, 5-6 (2014), 47–58.

[7] Aydin Buluc and John R Gilbert. 2008. Challenges and advances in parallel sparse matrix-matrix multiplication. In *2008 37th International Conference on Parallel Processing*. IEEE, 503–510.

[8] Xin Chen, Yingxiang Gao, Honghui Shang, Fang Li, Zhiqian Xu, Xin Liu, and Dexun Chen. 2022. Increasing the efficiency of massively parallel sparse matrix-matrix multiplication in first-principles calculation on the new-generation Sunway supercomputer. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 4752–4766.

[9] Sambit Das, Bikash Kanungo, Vishal Subramanian, Gourab Panigrahi, Phani Motamarri, David Rogers, Paul Zimmerman, and Vikram Gavini. 2023. Large-scale materials modeling at quantum accuracy: Ab initio simulations of quasicrystals and interacting extended defects in metallic alloys. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[10] Murray S Daw. 1993. Model for energetics of solids based on the density matrix. *Physical Review B* 47, 16 (1993), 10895.

[11] William Dawson and Takahito Nakajima. 2018. Massively parallel sparse matrix function calculations with NTPoly. *Computer Physics Communications* 225 (2018), 154–165.

[12] Haohuan Fu, Junfeng Liao, Jinzhe Yang, Lanning Wang, Zhenya Song, Xiaomeng Huang, Chao Yang, Wei Xue, Fangfang Liu, Fangli Qiao, et al. 2016. The Sunway TaihuLight supercomputer: system and applications. *Science China Information Sciences* 59 (2016), 1–16.

[13] Scott Gray, Alec Radford, and Diederik P Kingma. 2017. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224* 3, 2 (2017), 2.

[14] Thomas Herault, Yves Robert, George Bosilca, and Jack Dongarra. 2019. Generic matrix multiplication for multi-GPU accelerated distributed-memory platforms over PaRSEC. In *2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*. IEEE, 33–41.

[15] Thomas Herault, Yves Robert, George Bosilca, Robert J Harrison, Cannada A Lewis, Edward F Valeev, and Jack J Dongarra. 2021. Distributed-memory multi-GPU block-sparse tensor contraction for electronic structure. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 537–546.

[16] Jan S Hesthaven and Tim Warburton. 2004. High–order nodal discontinuous Galerkin methods for the Maxwell eigenvalue problem. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 362, 1816 (2004), 493–524.

[17] Pierre Hohenberg and Walter Kohn. 1964. Inhomogeneous electron gas. *Physical review* 136, 3B (1964), B864.

[18] Wei Hu, Hong An, Zhuoqiang Guo, Qingcai Jiang, Xinming Qin, Junshi Chen, Weile Jia, Chao Yang, Zhaolong Luo, Jielan Li, et al. 2022. 2.5 million-atom ab initio electronic-structure simulation of complex metallic heterostructures with dgdft. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–13.

[19] Wei Hu, Lin Lin, and Chao Yang. 2015. DGDFT: A massively parallel method for large scale density functional theory calculations. *The Journal of chemical physics* 143, 12 (2015).

[20] Wei Hu, Xinming Qin, Qingcai Jiang, Junshi Chen, Hong An, Weile Jia, Fang Li, Xin Liu, Dexun Chen, Fangfang Liu, et al. 2021. High performance computing of DGDFT for tens of thousands of atoms using millions of cores on Sunway TaihuLight. *Science Bulletin* 66, 2 (2021), 111–119.

[21] Jürg Hutter, Marcella Iannuzzi, Florian Schiffmann, and Joost VandeVondele. 2014. cp2k: atomistic simulations of condensed matter systems. *Wiley Interdisciplinary Reviews: Computational Molecular Science* 4, 1 (2014), 15–25.

[22] Lijuan Jiang, Chao Yang, and Wenjing Ma. 2020. Enabling highly efficient batched matrix multiplications on SW26010 many-core processor. *ACM Transactions on Architecture and Code Optimization (TACO)* 17, 1 (2020), 1–23.

[23] Wesley M Johnston, JR Paul Hanna, and Richard J Millar. 2004. Advances in dataflow programming languages. *ACM computing surveys (CSUR)* 36, 1 (2004), 1–34.

[24] Walter Kohn and Lu Jeu Sham. 1965. Self-consistent equations including exchange and correlation effects. *Physical review* 140, 4A (1965), A1133.

[25] Alfio Lazzaro, Joost VandeVondele, Jürg Hutter, and Ole Schütt. 2017. Increasing the efficiency of sparse matrix-matrix multiplication with a 2.5 D algorithm and one-sided MPI. In *Proceedings of the Platform for Advanced Scientific Computing Conference*. 1–9.

[26] Lin Lin, Jianfeng Lu, Lexing Ying, Roberto Car, and Weinan E. 2009. Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems. (2009).

[27] Fangfang Liu, Wenjing Ma, Yuwen Zhao, Daokun Chen, Yi Hu, Qinglin Lu, WanWang Yin, Xinhui Yuan, Lijuan Jiang, Hao Yan, et al. 2023. xmath2. 0: a high-performance extended math library for sw26010-pro many-core processor. *CCF Transactions on High Performance Computing* 5, 1 (2023), 56–71.

[28] Anders MN Niklasson. 2002. Expansion algorithm for the density matrix. *Physical Review B* 66, 15 (2002), 155115.

[29] Anders MN Niklasson, CJ Tymczak, and Matt Challacombe. 2003. Trace resetting density matrix purification in O (N) self-consistent-field theory. *The Journal of chemical physics* 118, 19 (2003), 8611–8620.

[30] Adam HR Palser and David E Manolopoulos. 1998. Canonical purification of the density matrix in electronic-structure theory. *Physical Review B* 58, 19 (1998), 12704.

[31] Jaime Peraire, Ngoc Nguyen, and Bernardo Cockburn. 2010. A hybridizable discontinuous Galerkin method for the compressible Euler and Navier-Stokes equations. In *48th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*. 363.

[32] Emil Prodan and Walter Kohn. 2005. Nearsightedness of electronic matter. *Proceedings of the National Academy of Sciences* 102, 33 (2005), 11635–11638.

[33] Robert Schade, Tobias Kenter, Hossam Elgabarty, Michael Lass, Thomas D Kühne, and Christian Plessl. 2023. Breaking the exascale barrier for the electronic structure problem in ab-initio molecular dynamics. *The International Journal of High Performance Computing Applications* 37, 5 (2023), 530–538.

[34] Ole Schütt, Peter Messmer, Jürg Hutter, and Joost VandeVondele. 2016. GPU-Accelerated Sparse Matrix–Matrix Multiplication for Linear Scaling Density Functional Theory. *Electronic Structure Calculations on Graphics Processing Units: From Quantum Chemistry to Condensed Matter Physics* (2016), 173–190.

[35] Honghui Shang, Hongjun Xiang, Zhenyu Li, and Jinlong Yang. 2010. Linear scaling electronic structure calculations with numerical atomic basis set. *International Reviews in Physical Chemistry* 29, 4 (2010), 665–691.

[36] Joost VandeVondele, Urban Borstnik, and Jurg Hutter. 2012. Linear scaling self-consistent field calculations with millions of atoms in the condensed phase. *Journal of chemical theory and computation* 8, 10 (2012), 3565–3573.

[37] Weitao Yang. 1992. Electron density as the basic variable: a divide-and-conquer approach to the ab initio computation of large molecules. *Journal of Molecular Structure: THEOCHEM* 255 (1992), 461–479.