Version of Record: https://www.sciencedirect.com/science/article/pii/S0010465522002351
Manuscript_ac0f49e7718fd2ce6a17e0617372b5cc

TTDFT: A GPU accelerated Tucker tensor DFT code for large-scale Kohn-Sham DFT calculations

Chih-Chuen Lin[a], Vikram Gavini[a,b,*]

[a]*Department of Mechanical Engineering, University of Michigan, MI 48109-2125, United States*
[b]*Department of Materials Science & Engineering, University of Michigan, MI 48109-2125, United States*

Abstract

We present the Tucker tensor DFT (TTDFT) code which uses a tensor-structured algorithm with graphic processing unit (GPU) acceleration for conducting ground-state DFT calculations on large-scale systems. The Tucker tensor DFT algorithm uses a localized Tucker tensor basis computed from an additive separable approximation to the Kohn-Sham Hamiltonian. The discrete Kohn-Sham problem is solved using Chebyshev filtered subspace iteration method that relies on matrix-matrix multiplications of a sparse symmetric Hamiltonian matrix and a dense wavefunction matrix, expressed in the localized Tucker tensor basis. These matrix-matrix multiplication operations, which constitute the most computationally intensive step of the solution procedure, are GPU accelerated providing ~8-fold GPU-CPU speedup for these operations on the largest systems studied. The computational performance of the TTDFT code is presented using benchmark studies on aluminum nano-particles and silicon quantum dots with system sizes ranging up to ~7,000 atoms.

Keywords: Kohn-Sham Density Functional Theory; L-1 localization; Tucker tensor; Tensor-structured methods; Real-space

PROGRAM SUMMARY/NEW VERSION PROGRAM SUMMARY

Program Title: TTDFT: Tucker tensor density functional theory code

CPC Library link to program files: (to be added by Technical Editor)

Licensing provisions: LGPL

Programming language: C/C++

External routines/libraries:
TuckerMPI(https://gitlab.com/tensors/TuckerMPI),
cuBLAS (https://docs.nvidia.com/cuda/cublas/index.html),
cuSparse(https://docs.nvidia.com/cuda/cusparse/index.html),
ALGLIB (http://www.alglib.net/), Boost (https://www.boost.org/),
BLAS (http://www.netlib.org/blas/), LAPACK (http://www.netlib.org/lapack/),
PETSc (https://www.mcs.anl.gov/petsc), SLEPc (http://slepc.upv.es)

Nature of problem: Real-space Kohn-Sham density functional theory calculations using localized Tucker tensor basis.

*Corresponding author.
E-mail address: vikramg@umich.eduu

Solution method: We present a real-space Kohn-Sham density functional code based on tensor-structured techniques with GPU acceleration. Tensor-structured techniques are adopted for computing a Tucker tensor basis, representing the eigenfunctions of an additive separable approximation to the Kohn-Sham Hamiltonian. The Tucker tensor basis is further localized using L_1 regularization to improve the sparsity of the Kohn-Sham Hamiltonian matrix, and improve the computational efficiency and parallel scalability of the proposed algorithm. The solution to the Kohn-Sham problem in the localized Tucker tensor basis is computed using the Chebyshev filtered subspace iteration (ChFSI) method.

Restrictions: The code works with Troullier-Martin (TM) pseudopotentials in Kleinman-Bylander form. The current release supports only non-periodic DFT calculations with the local density approximation (LDA) for exchange-correlation functional.

Additional comments: This TTDFT project uses GitHub via Git, a free distributed version control software. The archived version at the time of submission of this work can be found on the CPC program library through program files DOI provided above. The GitHub repository of this project can be found on `https://github.com/ttdftdev/ttdft_public`.

1. Introduction

Electronic structure calculations have provided many insights into the quantum mechanical properties of various materials over the past few decades. Density functional theory (DFT) [1, 2], owing to the great balance it provides between accuracy and computational efficiency, has emerged as the workhorse of electronic structure calculations. DFT reduces the Schrödinger equation involving the many-body wavefunction in $3N_e$ spatial coordinates (N_e denoting the number of electrons), to an equivalent problem of non-interacting electrons in a mean-field that is dependent on the electron density—a variable in only 3 spatial coordinates, thus substantially reducing the computational complexity. While DFT is exact in principle, the many-body quantum mechanical interactions are encapsulated in the exchange-correlation (XC) functional whose form is unknown, and approximate models are used to model the XC functional. The development of increasingly accurate XC functionals is an active area of research [3–8].

Despite the wide adoption of DFT for electronic structure calculations, the computational complexity of DFT calculations—conventionally, $O(MN_e^2)$, where M is the number of the basis functions required to achieve desired chemical accuracy, and is usually proportional to the number of electrons in the system (N_e)—limits typical DFT calculations to a few hundred atoms. Thus, to improve the computational efficiency of DFT calculations and enable accurate DFT calculations on large-scale systems, it is highly desirable to develop computational methods that can provide systematic convergence and are scalable to large number of MPI tasks, yet with a small basis set.

The plane-wave basis, which is the most widely used basis in DFT calculations [9–12], provides systematic convergence, and is well suited for periodic calculations. However, the global nature of the plane-wave basis limits the parallel scalability, and its uniform spatial resolution makes it inefficient for non-periodic systems, such as isolated molecules or clusters. On the other hand, while atomic orbital type basis functions [13–17] are very efficient—typically involving only few tens of basis functions per atom—systematic convergence is often a concern, especially in metallic systems. Further, the global nature of the basis functions can limit the parallel scalability of calculations. Recent efforts have also focused on developing adaptive reduced-order basis functions [18, 19] with a focus towards computationally efficient large-scale DFT calculations. Over the past few decades, there is also a growing use of systematically improvable real-space methods in electronic structure codes such as the finite-difference discretization [20, 21], the finite-element basis [22, 23] and the wavelet basis [24], which offer systematic convergence and exhibit good parallel scalability. In particular, the finite-element basis has been demonstrated to be highly scalable [22, 23, 25]—with parallel scalability demonstrated on $\sim 200,000$ MPI tasks [22] and on $\sim 22,000$ GPUs [26]. However, the number of basis functions required to achieve chemical accuracy using these systematically convergent real-space methods is typically much higher than the plane-wave basis.

Recent progress in using tensor-structured techniques for electronic structure calculations has provided a path forward for developing a reduced-order basis that is systematically improvable, efficient and exhibits good parallel scalability. In particular, an analysis of various molecules has revealed that the electronic structure, in particular the electron density, admits a low-rank Tucker and canonical decomposition [27]. Further, *a posteriori* results have shown that the rank required to approximate the electronic density is only weakly dependent on the system size [28]. Based on these observations, a tensor-structured basis was proposed for systematically convergent and efficient large-scale DFT calculations [29]. The main ideas included constructing an additive separable approximation of the Kohn-Sham Hamiltonian, and using the eigenbasis of this approximate Hamiltonian—which has a Tucker tensor format—as a reduced-order basis for DFT calculations. Importantly, being the eigenbasis of a Hermitian operator, the resulting Tucker tensor basis provides systematic convergence. Further, being adapted to the Kohn-Sham Hamiltonian, it was demonstrated to be a more efficient basis than the plane-wave basis, requiring fewer basis functions than the plane-wave basis to

achieve similar accuracy. In particular, recent benchmark studies showed that the number of basis functions required to achieve chemical accuracy are $\sim 3-5\times$ lower using the aforementioned Tucker tensor basis in comparison to the plane-wave basis [29, 30]. However, the global nature of the Tucker tensor basis resulted in a dense Hamiltonian matrix, which limited the accessible system sizes and parallel scalability of the method. In order to alleviate this limitation, we recently proposed an L_1 localization approach to construct a localized Tucker tensor basis [30], whose span is a close approximation to the subspace spanned by the eigenbasis of the additive separable approximation to the Kohn-Sham Hamiltonian. DFT calculations using the resulting localized Tucker tensor basis were demonstrated on large-scale systems involving many thousands of atoms. Further, this tensor-structured approach was shown to substantially outperform plane-wave implementations even for modest system sizes beyond 2,000 electrons.

The solution of Kohn-Sham equations in the localized Tucker tensor basis involves many operations that are amenable to acceleration using graphics processing units (GPU). In this work, we present the TTDFT code—Tucker tensor DFT code—that optimizes various parts of the tensor-structured algorithm using GPUs, and provides the code base for conducting large-scale DFT calculations using localized Tucker tensor basis. In particular, we optimize various compute intensive kernels using CUDA library: (i) the matrix-matrix multiplication between the Kohn-Sham Hamiltonian in the localized Tucker tensor basis and the wavefunction matrix expressed in this basis, which appears in the Chebyshev filtering procedure to compute the occupied subspace of the Kohn-Sham Hamiltonian; (ii) the solution of the Kohn-Sham equations by projecting the problem onto the Chebyshev filtered subspace. Our numerical study shows that the implementation substantially accelerates the Chebyshev filtering step—the most time-consuming part in a many-core CPU-based calculation—by $7-8\times$ and substantially reduces the wall-times for DFT calculations. Further, we demonstrate the capability of conducting large-scale DFT calculations, with systems as large as $\sim 7,000$ atoms, on GPUs efficiently.

The remainder of this paper is organized as follows. The Kohn-Sham formulation is presented in Sec. 2 for completeness. Section 3 presents the outline of the Tucker tensor algorithm with L_1 localization for the solution of the Kohn-Sham equations that is implemented in the TTDFT code. We describe the GPU acceleration scheme for improving the computational efficiency in Sec. 4. The numerical results from our implementation of GPU accelerated TTDFT code are presented in Sec. 5, and we summarize in Sec. 6.

2. Kohn-Sham DFT formulation

Kohn-Sham DFT addresses the ground state energy of a quantum mechanical system with N_a atoms and N_e electrons by solving a non-interacting single-particle Schrödinger equation subjected to a mean-field effective potential $v_{\text{eff}}(\rho;\mathbf{R})$

$$\begin{aligned} &\mathcal{H}\Psi_i = \epsilon_i\Psi_i\,, \qquad i\in\{1,...,N_{\text{orb}}\}\\ &\mathcal{H} = -\frac{1}{2}\nabla^2 + v_{\text{eff}}(\rho;\mathbf{R})\,. \end{aligned} \tag{1}$$

In the above, $\mathcal{H}$ denotes the Kohn-Sham Hamiltonian, $\{\epsilon_i,\Psi_i\}$ denotes the i-th eigenstate, N_{orb} denotes the number of eigenstates at the lower end of the spectrum that are computed ($N_{\text{orb}} > \frac{N_e}{2}$), and $\mathbf{R}$ denotes the vector with the positions of atoms. The electron density—the central quantity of interest in DFT— is denoted by $\rho = \rho(\mathbf{x})$ in real-space, with coordinates $\mathbf{x} = (x_1, x_2, x_3)$. The electron density is related to the Kohn-Sham orbitals by

$$\rho(\mathbf{x}) = 2\sum_{i=1}^{N_{\text{orb}}} f(\epsilon_i;\mu)\left|\Psi_i(\mathbf{x})\right|^2\,, \tag{2}$$

where $f(\epsilon;\mu)$ denotes the orbital occupancy function, and, in the present work, is represented by the Fermi-Dirac distribution

$$f(\epsilon;\mu) = \frac{1}{1+\exp(\frac{\epsilon-\mu}{k_B T})}\,. \tag{3}$$

Here, k_B is the Boltzmann constant, T is the temperature controlling the smearing of the orbital occupancy function, and μ is the Fermi energy that is solved using the constraint on the total number of electrons given by

$$2\sum_{i=1}^{N_{\text{orb}}} f(\epsilon_i;\mu) = N_e\,. \tag{4}$$

The effective potential in the Kohn-Sham Hamiltonian, $v_{\text{eff}}(\rho)$, is a functional of electron density, and is comprised of three contributions

$$v_{\text{eff}}(\rho) = \frac{\delta E_{\text{H}}}{\delta\rho} + \frac{\delta E_{\text{XC}}}{\delta\rho} + v_{\text{ext}}(\mathbf{x};\mathbf{R})\,. \tag{5}$$

E_{H} is the Hartree energy, which represents the classical Coulomb electrostatic interaction between electrons and is given by (in a non-periodic setting)

$$E_{\text{H}} = \frac{1}{2}\int_{\mathbb{R}^3}\int_{\mathbb{R}^3} \frac{\rho(\mathbf{x})\rho(\mathbf{x}')}{|\mathbf{x}-\mathbf{x}'|} d\mathbf{x}d\mathbf{x}' = \int_{\mathbb{R}^3} \rho(\mathbf{x}) v_{\text{H}}(\rho) d\mathbf{x}\,, \tag{6}$$

where $v_{\text{H}}(\rho)$ is the Hartree potential defined by the functional derivative of the Hartree energy

$$v_{\text{H}}(\rho) = \frac{\delta E_{\text{H}}}{\delta\rho} = \int_{\mathbb{R}^3} \frac{\rho(\mathbf{x}')}{|\mathbf{x}-\mathbf{x}'|} d\mathbf{x}'. \tag{7}$$

E_{XC} is the exchange-correlation energy, which describes all the many-body quantum mechanical interactions between electrons. The functional derivative of E_{XC} is labeled as the exchange-correlation potential

$$v_{\text{XC}}(\rho) = \frac{\delta E_{\text{XC}}}{\delta\rho}\,. \tag{8}$$

In this work, the local density approximation (LDA) in the form of Ceperley-Alder parametrization with Perdew-Zunger data [31, 32] is used for the exchange-correlation functional. The last term in Eq. (5), $v_{\text{ext}}(\mathbf{x};\mathbf{R})$, is the electrostatic potential acting on electrons induced by the nuclei. Typically, the core electrons do not participate in chemical reactions, hence a pseudopotential approximation is commonly adopted to replace the all-electron Coulomb potential by a smoother potential acting only on valence electrons. The behavior of the pseudopotential operator v_{ext} acting on valence electrons is decomposed into a local part v_{ext}^{loc} and a non-local part v_{ext}^{nl}. In this work, the norm-conserving Troullier-Martin [33] pseudopotential in Kleinman-Bylander [34] form is used. The action of the pseudopotential operator on the Kohn-Sham orbitals in real space is defined as

$$v_{\text{ext}}(\mathbf{x};\mathbf{R})\Psi(\mathbf{x}) = v_{\text{ext}}^{loc}(\mathbf{x};\mathbf{R})\Psi(\mathbf{x}) + v_{\text{ext}}^{nl}(\mathbf{x};\mathbf{R})\Psi(\mathbf{x})\,. \tag{9}$$

$$v_{\text{ext}}^{loc}(\mathbf{x};\mathbf{R})\Psi(\mathbf{x}) = \sum_{J=1}^{N_a} v_{\text{ext}}^{loc,J}(\mathbf{x}-\mathbf{R}_J)\Psi(\mathbf{x})\,, \tag{10}$$

where $v_{\text{ext}}^{loc,J}(\mathbf{x}-\mathbf{R}_J)$ is the corresponding local potential for the J-th atom, and $\mathbf{R}_J$ is the coordinate of the J-th atom.

$$v_{\text{ext}}^{nl}(\mathbf{x};\mathbf{R})\Psi(\mathbf{x}) = \sum_{J}^{N_a}\sum_{lm} C_{lm}^{J}\varphi_{lm}^{J}(\mathbf{x}-\mathbf{R}_J)\Delta v_l^J(\mathbf{x}-\mathbf{R}_J)\,, \tag{11}$$

where

$$C_{lm}^{J} = \frac{\int \varphi_{lm}^{J}(\mathbf{x}-\mathbf{R}_J)\Delta v_l^J(\mathbf{x}-\mathbf{R}_J)\Psi(\mathbf{x})d\mathbf{x}}{\int \varphi_{lm}^{J}(\mathbf{x}-\mathbf{R}_J)\Delta v_l^J(\mathbf{x}-\mathbf{R}_J)\varphi_{lm}^{J}(\mathbf{x}-\mathbf{R}_J)d\mathbf{x}}$$

and

$$\Delta v_l^J(\mathbf{x}-\mathbf{R}_J) = v_l^J(\mathbf{x}-\mathbf{R}_J) - v_{\text{ext}}^{loc,J}(\mathbf{x}-\mathbf{R}_J).$$

Therein, $v_l^J(\mathbf{x})$ is the pseudopotential component of the J-th atom corresponding to the l azimuthal quantum number; $\varphi_{lm}^J(\mathbf{x})$ is the single atom pseudo-wavefunction of the J-th atom corresponding to the azimuthal and magnetic quantum numbers l and m, respectively.

Finally, upon solving Eq. (1), Eq. (2), and Eq. (4) self-consistently in a suitable basis, the ground state energy of the given system can be obtained by

$$E_{\text{tot}} = E_{\text{band}} + E_{\text{XC}} - \int_{\mathbb{R}^3}\rho v_{\text{XC}}(\rho)d\mathbf{x} - \frac{1}{2}\int_{\mathbb{R}^3}\rho v_{\text{H}}(\rho)d\mathbf{x} + E_{\text{ZZ}}\,, \tag{12}$$

where

$$E_{\text{band}} = 2\sum_{i=1}^{N_{\text{orb}}} f(\epsilon_i;\mu)\epsilon_i$$

is the band energy. Finally,

$$E_{\text{ZZ}} = \sum_{I=1}^{N_a}\sum_{J>I}^{N_a}\frac{Z_I Z_J}{|\mathbf{R}_I-\mathbf{R}_J|}$$

is the repulsion energy between nuclei, where Z_I is the valence charge of the I-th atom.

3. Tensor-structured algorithm with L_1 localization

In this section, we present the tensor-structured approach of using L_1 localized Tucker tensor basis for Kohn-Sham DFT calculations. We note that these ideas have been developed in our prior works [29, 30], and we present the details of the algorithm as implemented in the TTDFT code, before discussing the GPU acceleration strategy for the various compute intensive kernels. In this section, we first provide a brief overview to the Tucker tensor representation, and refer to [35] for more detailed review. Next, the algorithm to construct localized Tucker tensor basis that is adapted to the Kohn-Sham Hamiltonian is presented. Finally, the solution of the Kohn-Sham equations in the localized Tucker tensor basis by using Chebyshev filtered subspace iteration [36, 37] is discussed.

3.1. Tucker tensor representation

Tucker tensor representation can be regarded as a higher-order generalization of the singular value decomposition of an N-dimensional tensor. For an N-dimensional tensor, its Tucker tensor representation has the form of a smaller N-dimensional tensor and N factor matrices whose column vectors are its rank-1 components. We restrict the discussion to a 3-D tensor as relevant to this work. Let $A \in \mathbb{R}^{I_1\times I_2\times I_3}$ be a real-valued 3-D tensor of size $I_1 \times I_2 \times I_3$ indexed by a set of integers (i_1, i_2, i_3)

$$A_{(i_1,i_2,i_3)} = a_{i_1 i_2 i_3}\,, \tag{13}$$

where $i_d \in \{1, 2, ..., I_d\}, I_d \in \mathbb{N}$ and $d \in \{1, 2, 3\}$ denotes the dimensions. A Tucker tensor representation of the tensor A with decomposition rank $\mathbf{R} = (R_1, R_2, R_3)$ has the form

$$A \approx A^{(\mathbf{R})} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} \sigma_{r_1 r_2 r_3} \mathbf{u}_1^{r_1} \mathbf{u}_2^{r_2} \mathbf{u}_3^{r_3}, \tag{14}$$

where $\sigma_{r_1 r_2 r_3} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ denotes the core tensor, $\mathbf{u}_d^{r_d} \in \mathbb{R}^{I_d}$ are the rank-1 components for the factor matrix $\mathbf{U}_d \in \mathbb{R}^{I_d \times R_d}$. A graphical illustration of the Tucker decomposition process is presented in Fig. 1. The core tensor could be viewed as the higher-order generalization of singular values and stores the coefficients $\sigma_{r_1 r_2 r_3}$ for each rank-1 tensor $\mathbf{u}_1^{r_1} \otimes \mathbf{u}_2^{r_2} \otimes \mathbf{u}_3^{r_3}$. The factor matrices can as well be seen as the higher-order correspondence of the matrices comprising the singular vectors. We note that many approaches have been suggested to perform Tucker decomposition of a given tensor. In this work, we adopt high-order singular value decomposition (HOSVD) techniques for Tucker decomposition, and we refer to [35, 38, 39] for more details on the various methods for Tucker tensor decomposition. In particular, in this work, we use the TuckerMPI code for performing HOSVD, which is an MPI implementation of tensor operations in Tucker representation. We refer to [40, 41] for the library, and details of the implementation.

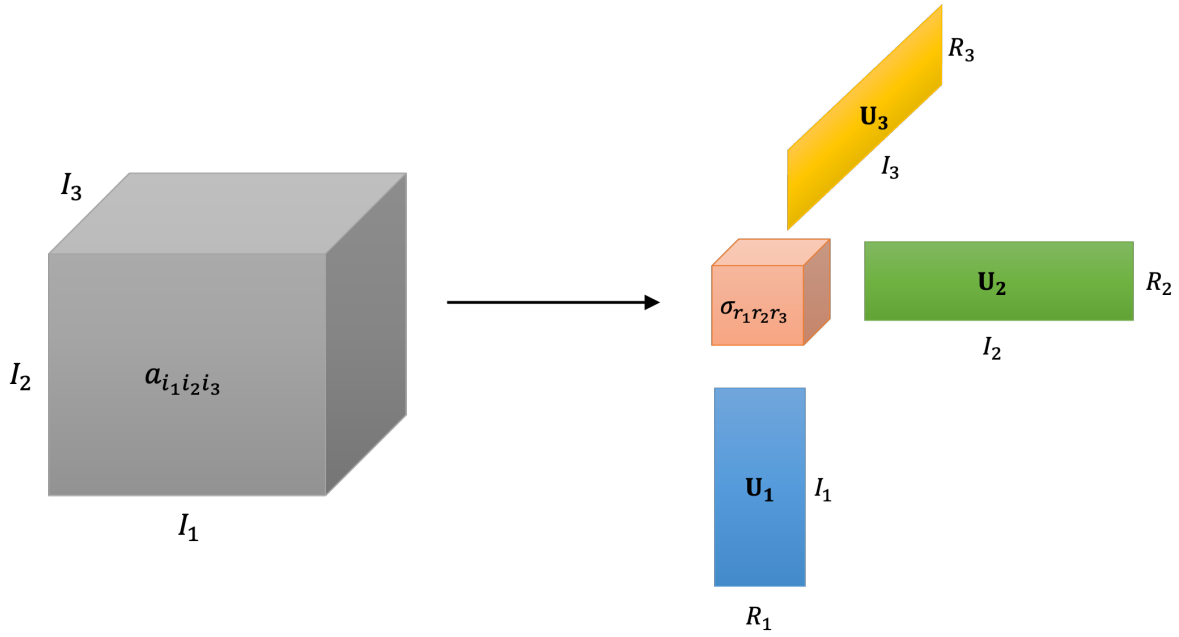


Figure 1: Schematic of Tucker decomposition.

3.2. Construction of L_1 Tucker tensor basis

The construction of L_1 Tucker tensor basis includes the following steps. (i) Compute an additive separable approximation to the Kohn-Sham Hamiltonian in a cuboidal domain Ω spanned by three 1-D real domains $\omega_{k=1,2,3}$ along the spatial coordinates and enclosing the compact support of the Kohn-Sham wavefunctions [29]. The resulting separable approximation to the Kohn-Sham Hamiltonian has the form $\mathcal{H}_1(x_1)+\mathcal{H}_2(x_2)+\mathcal{H}_3(x_3) \approx \mathcal{H}(\mathbf{x})$. We note that the eigenspace of this separable approximation represents a useful reduced-order subspace and the eigenfunctions have a Tucker tensor structure. (ii) While the Tucker tensor basis constructed from the 1-D eigenfunctions of $\mathcal{H}_k$ is efficient [29], requiring fewer basis functions

to achieve chemical accuracy in comparison to the plane-wave basis, the global nature of this basis limits the computational efficiency and parallel scalability for large-scale DFT calculations. To this end, we employ L_1 localization, to construct localized 1-D functions that closely approximate the eigen-subspace of $\mathcal{H}_k$. (iii) Finally, these localized 1-D functions are used to construct the 3-D localized Tucker tensor basis. In particular, the 3-D Tucker basis is constructed as the tensor product of the localized 1-D functions. We now will elaborate the details of each aspect for constructing the L_1 localized Tucker tensor basis.

3.2.1. Separable approximation to the Kohn-Sham Hamiltonian

The separable approximation to the Kohn-Sham Hamiltonian is constructed based on a rank-1 approximation of the eigenfunction corresponding to the lowest eigenvalue [29]. To this end, we seek the solution to the lowest eigenstate of Eq. (1) to belong to $S = \{\Psi(\mathbf{x}) | \Psi(\mathbf{x}) = \psi_{x_1}(x_1)\psi_{x_2}(x_2)\psi_{x_3}(x_3)\}$ that has a rank-1 tensor structure. The solution to lowest eigenstate of the Kohn-Sham equation such that it belong to S is equivalent to finding the minimizer of the following energy functional

$$L(\Psi) = \int_\Omega \frac{1}{2} \sum_{\ell=1}^{3} \left| \frac{d\psi_{x_\ell}}{dx_\ell} \right|^2 \prod_{m \neq \ell}^{3} \psi_{x_m}^2 + \left(v_{\text{eff}}^{loc}(\mathbf{x}) + \lambda \right) \prod_{\ell=1}^{3} \psi_{x_\ell}^2 + \left(\prod_{\ell=1}^{3} \psi_{x_\ell} \right) v_{\text{ext}}^{nl}(\mathbf{x}) \left(\prod_{\ell=1}^{3} \psi_{x_\ell} \right) d\mathbf{x} \,, \tag{15}$$

where λ denotes the Lagrange multiplier corresponding to normality of the eigenstate, and $v_{\text{eff}}^{loc} = v_{\text{H}} + v_{\text{XC}} + v_{\text{ext}}^{loc}$ denotes the local part of the effective potential. Upon writing the Euler-Lagrange equations corresponding to variations with respect to ψ_{x_1}, ψ_{x_2}, ψ_{x_3}, and using Tucker tensor decomposition on both v_{eff}^{loc} and v_{ext}^{nl}, we obtain simultaneous 1-D problems in the form

$$\begin{aligned} &\mathcal{H}_k \psi_k = -(\lambda + a_k)\psi_k \,, \\ &\mathcal{H}_k = -\frac{1}{2}\frac{d^2}{dx_k^2} + v_k^{loc}(x_k; \psi_{l \neq k}) + v_k^{nl}(x_k; \psi_{l \neq k}) \,, \end{aligned} \tag{16}$$

where $v_k^{loc}(x_k)$, $v_k^{nl}(x_k)$ are the local and the non-local contribution to the 1-D potentials respectively, and a_k is a constant parametrized by $\psi_{l \neq k}$. The solution to Eq. 16 can be obtained via a self-consistent field iteration, and we refer to [30] for complete details on the formulation and solution procedure.

3.2.2. SOC algorithm and L_1 localization

The minimizer of Eq. (15) also yields an additive separable approximation to the Kohn-Sham Hamiltonian ($\mathcal{H}_1(x_1) + \mathcal{H}_2(x_2) + \mathcal{H}_3(x_3)$). We note that the eigenfunctions of this approximate Hamiltonian are a tensor product of the eigenfunctions of the 1-D Hamiltonians ($\mathcal{H}_k$, $k = 1, 2, 3$), owing to the additive separable structure. Further, these eigenfunctions represent a suitable reduced-order basis for the solution of the Kohn-Sham equations in Eq. (1). We note that the plane-wave basis, the mostly widely used basis for DFT calculations, represent the eigenfunctions of the Laplace operator, whereas the eigenfunctions of the additive separable approximation also have some information of the Kohn-Sham potential and are expected to have better approximation properties. In fact numerical studies [29] have shown exponential convergence with increasing basis size, and chemical accuracy was attained with fewer basis functions in comparison to plane-wave basis. While efficient in terms of basis size, this basis is spatially extended and results in a dense discrete Kohn-Sham Hamiltonian matrix that limits the computational efficiency and parallel scalability. This limitation was addressed in our recent work [30], where localized 1-D functions are generated such that the span of these functions is a close approximation to the space spanned by the eigenbasis of $\mathcal{H}_k$. In particular, the localized functions are generated using an L_1 localization technique by solving the following constraint minimization problem

$$\min_{\boldsymbol{\psi}'_k \in \mathbb{R}^{n\times N_k}} \frac{1}{\mu}\left|\boldsymbol{\psi}'_k\right| + \mathrm{Tr}(\boldsymbol{\psi}'^{T}_k \mathbf{H}_k \boldsymbol{\psi}'_k) \quad \text{with } \boldsymbol{\psi}'^{T}_k \boldsymbol{\psi}'_k = I, \tag{17}$$

where $\mathbf{H}_k$ is the 1-D separable approximation to the Kohn-Sham Hamiltonian represented in a suitable orthogonal basis, $\boldsymbol{\psi}'_k$ is a matrix comprising N_k trial 1-D functions represented in the orthogonal basis, n denotes the number of rows (and columns) of $\mathbf{H}_k$, N_k is the number of 1-D functions to be computed.

The splitting orthogonality constraint algorithm (SOC) is used in this work for solving the constraint minimization problem in Eq. (17). We refer to [30] for details of the SOC algorithm used in the context of the tensor-structured algorithm for generating 1-D L_1 localized functions from the separable approximation to the Kohn-Sham Hamiltonian. We also refer to [42, 43] for more information on the method and its wider applications.

3.2.3. 3-D localized tensor-structured basis construction

Upon solving the constraint minimization problem in Eq. (17), the localized 1-D functions $\psi^L_{x_1,r_1}$, $\psi^L_{x_2,r_2}$, $\psi^L_{x_3,r_3}$ are computed, and the number of localized 1-D functions in each direction—denoted by R_1, R_2 and R_3—constitutes the Tucker rank in each direction of the 3-D localized Tucker tensor basis. The 3-D Tucker tensor basis is given by the tensor product of the 1-D localized functions as

$$T^L_K(\mathbf{x}) = \psi^L_{x_1,r_1}(x_1)\psi^L_{x_2,r_2}(x_2)\psi^L_{x_3,r_3}(x_3), \tag{18}$$

where $1 \le r_d \le R_d$ $(d = 1, 2, 3)$ and K is the composite index $K = (r_1, r_2, r_3)_{1\le r_d \le R_d}$. The space spanned by the 3-D localized tensor-structured basis functions are denoted as $\mathbb{T}^L$.

3.3. Discrete Kohn-Sham eigenvalue problem

The discrete Kohn-Sham Hamiltonian in the localized tensor-structured basis functions T^L_I is given by

$$H^L_{I,J} = \left\langle T^L_I \right| -\frac{1}{2}\nabla^2 + v_{\mathrm{eff}}(\rho;\mathbf{R}) \left| T^L_J \right\rangle, \tag{19}$$

where I and J are composite indices $I = (i_1, i_2, i_3)$, $J = (j_1, j_2, j_3)$. We note that in practice, the effective potential v_{eff} is represented in Tucker format to take advantage of the efficient tensor-structured calculation for computing entries of the Kohn-Sham Hamiltonian. Upon computing the Hamiltonian matrix, the matrix elements that are smaller than a prescribed tolerance are set to zero to attain better sparsity in the discrete Hamiltonian. Further, while the 3-D localized Tucker tensor basis can be computed for every self-consistent field (SCF) iteration of the Kohn-Sham problem, numerical studies have suggested that it suffices to construct the Tucker tensor basis in the first iteration and keep this fixed during the SCF iteration [30], as the error from the separable approximation of the Kohn-Sham Hamiltonian typically dominates the SCF error. Owing to the orthonormality of the 3-D localized Tucker tensor basis, the discrete Kohn-Sham eigenvalue problem is given by

$$\mathbf{H}^L\boldsymbol{\Psi} = \boldsymbol{\Psi}\boldsymbol{\Lambda}\,. \tag{20}$$

3.4. Chebyshev filtered subspace iteration (ChFSI) method

The standard eigenvalue problem in Eq. (20) is solved using the Chebyshev filtered subspace iteration (ChFSI) method [37]. The ChFSI method has been demonstrated to be an effective method for large-scale real-space DFT calculations [22, 26]. In every SCF iteration, the ChFSI method seeks to compute a good approximation to the subspace spanned by the occupied states of the Kohn-Sham Hamiltonian. This is

realized by taking advantage of the property of Chebyshev polynomials that are bounded in the interval $[-1, 1]$, but grow rapidly outside this interval. To this end, the discrete Kohn-Sham Hamiltonian is scaled and shifted such that the unwanted spectrum maps to $[-1, 1]$ and the desired spectrum of the occupied and partially occupied states maps to $(-\infty, -1)$. Thus, the application of a Chebyshev polynomial filter, constructed from the scaled-and-shifted Hamiltonian, on a set of vectors provides a subspace that is a close approximation to the desired occupied eigenspace. The Chebyshev filtered vectors are orthogonalized using Cholesky Gram-Schmidt (CholGS) orthogonalization procedure [44, 45] , and the Kohn-Sham eigenvalue problem (Eq.(19)) is solved by projecting the problem onto the Chebyshev filtered subspace.

Algorithm 1: ChFSI [36]

Input: $\mathbf{H}^L$, $\mathbf{X}$, m, ϵ_0, ϵ_{ub}^{w}, ϵ_{ub}^{uw}
Output: $\mathbf{\Psi}$, diag $(\mathbf{\Lambda})$
1. Chebyshev filtering procedure
   Initialize: $e = \frac{1}{2}\left(\epsilon_{ub}^{uw} - \epsilon_{ub}^{w}\right)$; $c = \frac{1}{2}\left(\epsilon_{ub}^{uw} + \epsilon_{ub}^{w}\right)$; $\sigma = \frac{e}{\epsilon_0 - c}$
   $\sigma_1 = \sigma$; $\gamma = \frac{2}{\sigma_1}$; $\tilde{\mathbf{X}} = \frac{\sigma_1}{e}\left(\mathbf{H}^L\mathbf{X} - c\mathbf{X}\right)$;
   for $i = 2 : m$
      $\sigma_2 = \frac{1}{\gamma - \sigma}$;
      $\tilde{\mathbf{X}}_{new} = \frac{2\sigma_2}{e}\left(\mathbf{H}^L\tilde{\mathbf{X}} - c\tilde{\mathbf{X}}\right) - \sigma\sigma_2\mathbf{X}$;
      $\mathbf{X} = \tilde{\mathbf{X}}$; $\tilde{\mathbf{X}} = \tilde{\mathbf{X}}_{new}$; $\sigma = \sigma_2$;
   end for
2. Orthonormalize the Chebyshev filtered basis functions with Cholesky Gram-Schmidt procedure (cf. Algorithm 2), and denote by $\mathbf{X}_F$ = CholGS($\mathbf{X}$)
3. Perform subspace projection: $\mathbf{H}_F^L = \mathbf{X}_F^{\mathrm{T}}\mathbf{H}^L\mathbf{X}_F$
4. Diagonalize $\mathbf{H}_F^L$ with eigen-decomposition $\mathbf{H}_F^L\mathbf{Q} = \mathbf{Q}\mathbf{\Lambda}$
5. Rotate the basis $\mathbf{\Psi} = \mathbf{X}\mathbf{Q}$

Algorithm 2: CholGS

Input: $\mathbf{X}$
Output: $\mathbf{X}_{\mathrm{F}}$
1. Compute the overlap matrix $\mathbf{S} = \mathbf{X}^T\mathbf{X}$
2. Perform the Cholesky factorization for the overlap matrix $\mathbf{S} = \mathbf{L}\mathbf{L}^T$
3. Generate the orthogonalized vectors $\mathbf{X}_{\mathrm{F}} = \mathbf{X}\mathbf{L}^{-1^T}$

The ChFSI method and the CholGS procedure are outlined in the Algorithm 1 and Algorithm 2for the sake of completeness, and we refer to [36, 37] for further information. In the Algorithm 1 and 2 , m denotes the Chebyshev polynomial degree; ϵ_0 and ϵ_{ub}^{w} are the lower and upper bound of the wanted spectrum, respectively; ϵ_{ub}^{uw} is the upper bound of the unwanted spectrum; $\mathbf{X}$ is the input wavefunction matrix; $\mathbf{\Psi}$ is the output wavefunction; $\mathbf{X}_{\mathrm{F}}$ is the matrix comprising orthonormalized Chebyshev fileterd vectors . As suggested in [36], the lower bound of the wanted spectrum is used to introduce a further scaling to prevent $\mathbf{X}$ from overflowing. In the first SCF iteration, $\mathbf{X}$ is typically set to either a random full-rank matrix or represented by atomic orbitals, and the Chebyshev filtering is performed using higher polynomial degree m. In the subsequent iterations, $\mathbf{X}$ is set to be the resultant $\mathbf{\Psi}$ from the previous SCF iteration, which provides a good guess and thus does not need a large m. For the various benchmark systems studied in this work, m

is chosen to be $10 - 20$.

4. GPU acceleration

In the solution of the Kohn-Sham equations using the localized Tucker tensor basis, the Chebyshev filtering step in each SCF iteration is the most computationally expensive step for even systems comprising $\sim 10,000$ electrons. The main kernel in the Chebyshev filtering is the sparse-dense matrix-matrix multiplication, and a GPU acceleration of this kernel can result in substantial reductions in the wall-times of the DFT calculation.

To this end, the Hamiltonian matrix $\mathbf{H}^L$ and the wavefunction matrix $\mathbf{X}$ are partitioned row-wise. We note that this work takes advantage of band-parallelism to reduce the communication costs and improve parallel scalability, where a subset of wavefunctions are assigned to each group of MPI tasks via sub-communicators. Thus, each GPU owns multiple rows of the Hamiltonian matrix and the wavefunction matrix corresponding to the sub-group. The details of the data layout for the Hamiltonian matrix and the wavefunction matrix are elaborated in following sections. We also remark that the sparsity pattern of $\mathbf{H}^L$ is such that the matrix has less sparsity around the diagonal, whereas the sparsity increases away from the diagonal. This structure is due to the spatial locality of the L_1 localized Tucker tensor basis. We take advantage of this structure to develop an efficient implementation of the matrix-matrix multiplication kernel in the Chebyshev filtering step. Further, we take advantage of the fact that $\mathbf{H}^L$ is symmetric to reduce communication costs. Upon computing the Chebyshev filtered vectors, the filtered vectors are orthonormalized and the Kohn-Sham Hamiltonian is projected onto the Chebyshev filtered subspace. These two steps involve the evaluation of $\mathbf{X}^T \times \mathbf{Y}$, where $\mathbf{X}$ is the wavefunction matrix and $\mathbf{Y}$ is a matrix with the same size as $\mathbf{X}$. This operation is also GPU ported for better performance.

The remainder of this section will present our implementation of the various aspects of the TTDFT code that have been GPU accelerated, which include: (i) the details of the data layout for the Kohn-Sham Hamiltonian matrix $\mathbf{H}^L$ and the wavefunction matrix $\mathbf{X}$, (ii) the algorithm for matrix-matrix multiplication of $\mathbf{H}^L\mathbf{X}$ and $\mathbf{X}^T \times \mathbf{Y}$ on GPUs, and using these matrix-matrix multiplication kernels for the subspace projection $\mathbf{H}_F^L = \mathbf{X}_F^{\mathrm{T}}\mathbf{H}^L\mathbf{X}_F$.

4.1. Data layout for $\mathbf{H}^L$ *and* $\mathbf{X}$

Figure 2 provides a schematic of the data layout of the sparse Kohn-Sham Hamiltonian matrix $\mathbf{H}^L$ in the localized Tucker tensor basis. The ownership of the rows of the Hamiltonian matrix is distributed as evenly as possible so that each GPU shares similar working load. Particularly, given the Hamiltonian matrix $\mathbf{H}^L$ of size $M \times M$, the matrix is distributed across N GPUs labeled from 0 to $N - 1$ as shown in Fig. 2. Let τ be the quotient of M divided by N, then block of the Hamiltonian matrix $\mathbf{H}^L$ residing on the k-th GPU owns the $k\tau$-th to the $((k + 1)\tau - 1)$-th rows of the Hamiltonian matrix, and is of size $\tau \times M$. In the case that M is not divisible by N, and ν be the remainder of M divided by N, the local block of the Hamiltonian matrix of the first ν GPUs are adjusted to be of size $(\tau + 1) \times M$.

We remark that the sparsity pattern of $\mathbf{H}^L$ is such that most of the non-zero entries of the matrix are concentrated on and around the diagonal of the matrix, owing to the spatial locality of the L_1 localized Tucker tensor basis. Thus, in a tiling of the matrix, the diagonal blocks are much denser compared to the off-diagonal blocks. Thus, the non-zero terms in the diagonal blocks could easily exceed 5%, which is the suggested minimal sparsity for sparse algorithm to be efficient [46], and deteriorate the overall performance. To this end, the diagonal blocks and the off-diagonal blocks of the Hamiltonian matrix are stored as dense and sparse matrices, respectively. The proposed data layout for the row-wise partitioned matrix on the 0-th

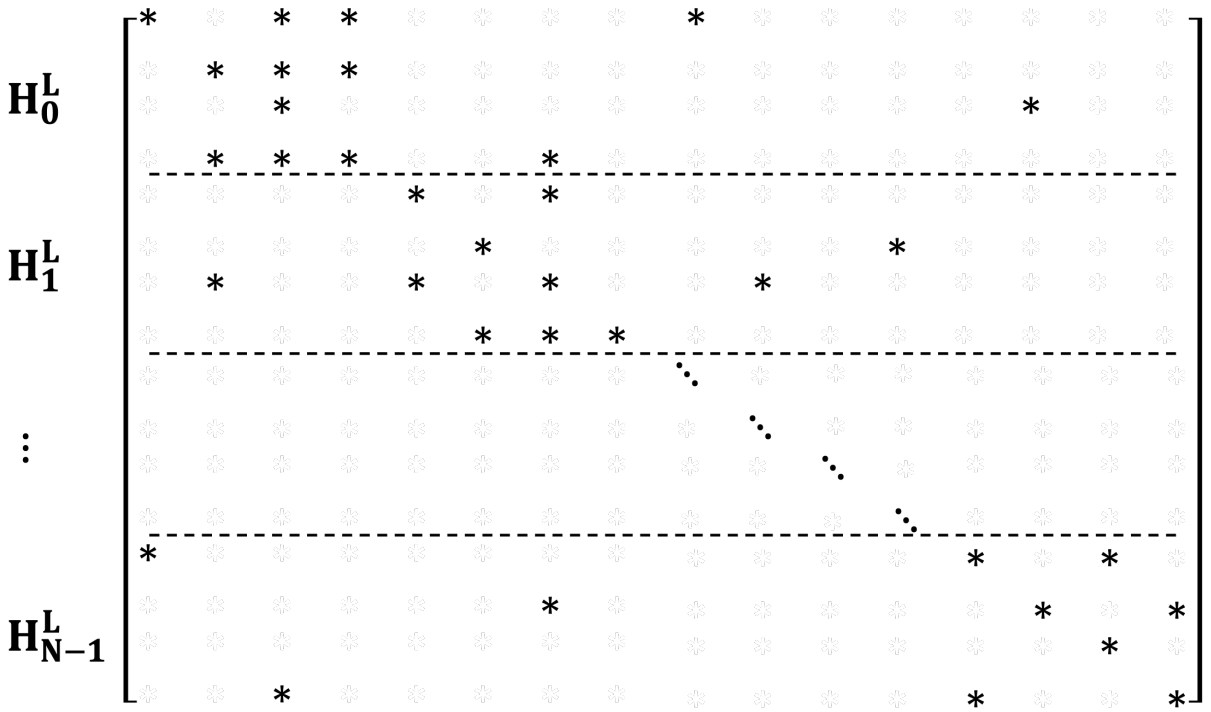


Figure 2: Schematic of the distribution of the projected Hamiltonian $\mathbf{H}^L$ on each GPU. The $\mathbf{H}_k^L$ ($k = 0, 1, \ldots N-1$) partition of the projected Kohn-Sham Hamiltonian is assigned to the k-th GPU.

GPU is illustrated in Fig. 3. The diagonal dense square matrix part of $\mathbf{H}_k^L$ is denoted as $\mathbf{H}_k^{L(D)}$ and the off-diagonal sparse matrix is denoted as $\mathbf{H}_k^{L(OD)}$. The two parts of $\mathbf{H}_k^L$ will then be treated using dense and sparse linear algebra library for the matrix-matrix multiplication kernel, respectively. We note that the number of rows owned by each GPU is chosen to be ∼30,000 in the current implementation so that the density of the diagonal block of the matrix exceeds 5%, yet fits in the GPU memory. Above the 5% threshold, the dense algorithm is generally considered to outperform the sparse algorithm.

Figure 3: Schematic of the data layout of the row-wise partitioned Hamiltonian matrix $\mathbf{H}^L$ on the 0-th GPU.

The wavefunction matrix $\mathbf{X}$ is of size $M \times N_{\text{orb}}$, where N_{orb} is the number of computed Kohn-Sham orbitals. Owing to double occupancy of the orbitals for spin-independent Hamiltonian, N_{orb} is usually chosen to be slightly larger than $N_e/2$, typically ∼10-15% larger. The rows of the wavefunctions are distributed consistently with the row-ownership of the Hamiltonian matrix $\mathbf{H}^L$. We note that during the computation of

$\mathbf{X}' = \mathbf{H}^L\mathbf{X}$, regardless of the implementation, collective communication over either $\mathbf{X}$ or $\mathbf{X}'$ will be needed. The cost for the collective communication is proportional to the number of processors and the data to be communicated within the (sub-)communicator [47, 48]. In a GPU calculation, this also requires data to be transferred from the device memory to the host memory and communication to other processors. Thus, this step will substantially increase the communication cost and deteriorate the overall performance. It is thus desirable to reduce this communication cost. To this end, in addition to the row-wise parallelization, columns of the wavefunction matrix $\mathbf{X}$ are further partitioned into groups (bands) labeled as $G_{p=0\ldots P-1}$, and this is referred to as band parallelization henceforth in keeping with the nomenclature of DFT literature. In the present implementation, the Hamiltonian matrix $\mathbf{H}^L$ is stored on each GPU group. Hence, each group will perform the matrix-matrix multiplication corresponding to the band of wavefunctions, and the collective communication after the matrix-matrix multiplication is only within the processors in the group. The number of processors to be communicated will thus be reduced by a factor P by using band-parallelism. Thus, the communication burden is significantly alleviated, and the overall performance of the Chebyshev filtering step is improved. A schematic illustration of the data layout for the wavefunction matrix $\mathbf{X}$ is provided in Fig. 4, where $\mathbf{X}_k$ is the portion of the wavefunction matrix having the same row-ownership of the Hamiltonian matrix $\mathbf{H}^L_k$ in Fig. 2. $\mathbf{X}^{G_i}_k$ is the portion of the wavefunction matrix $\mathbf{X}_k$ belonging to the G_i processor group. The data layout for the Hamiltonian matrix $\mathbf{H}^L$ and the wavefunction matrix $\mathbf{X}$ are then used to implement the sparse-dense matrix-matrix multiplication kernel, which is subsequently discussed.

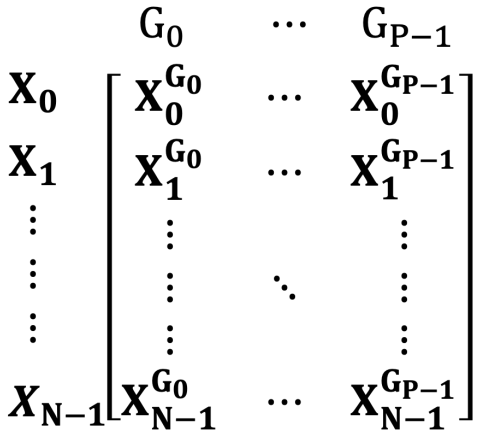


Figure 4: Schematic of the data layout of the wavefunction matrix $\mathbf{X}$.

4.2. $\mathbf{H}^L \times \mathbf{X}$ *implementation*

As noted previously, the Hamiltonian matrix is distributed row-wisely across GPUs in each group G_i. In the matrix-matrix multiplication, each $\mathbf{H}^L_k \times \mathbf{X}^{G_i}$, where $\mathbf{X}^{G_i}$ of size $M \times \frac{N_{\text{orb}}}{P}$ is the collection of all $\mathbf{X}^{G_i}_k$ ($k = 0, 1, \ldots, N-1$), yields $(\mathbf{H}^L\mathbf{X})^{G_i}_k$ on the k-th GPU in the group G_i. The evaluation of each $(\mathbf{H}^L\mathbf{X})^{G_i}_k$ requires communication of the off-diagonal block of the Hamiltonian matrix $\mathbf{H}^{L(OD)}_k$ to all processors other than k, as well as collecting information back from those processors. This communication also involves data transfer between the host and the device memory of GPUs, and can severely diminish the performance of the sparse-dense matrix-matrix multiplication kernel.

In order to avoid the aforementioned communication of $\mathbf{H}^{L(OD)}_k$, the matrix-matrix multiplication kernel

is recast by taking advantage of the symmetric nature of $\mathbf{H}^L$. We note that as $\mathbf{H}^L$ is real and symmetric,

$$\mathbf{H}^L_{(:,a:b)} = \left(\mathbf{H}^L_{(a:b,:)}\right)^{\mathrm{T}}. \tag{21}$$

Eq. (21) states that a column block of the Hamiltonian matrix $\mathbf{H}^L_{(:,a:b)}$, which is a matrix containing the a-th to b-th columns of $\mathbf{H}^L$, is equivalent to the transpose of a row block $\mathbf{H}^L_{(a:b,:)}$ comprising the a-th to b-th rows of $\mathbf{H}^L$. Further, we note that the evaluation of a matrix-matrix product $\mathbf{C} = \mathbf{AB}$, where A and B are $m \times n$ and $n \times m$ matrices, is given by $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ with a_{ij}, b_{ij} and c_{ij} denoting the matrix elements of $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, respectively. The expression $\sum_{k=1}^{n} a_{ik} b_{kj}$ can also be viewed as a summation over k of the outer product of the k-th column vector of $\mathbf{A}$ with the k-th row vector of $\mathbf{B}$. Thus, using this interpretation of the matrix-matrix multiplication as a sum of the outer product of column and row vectors of the constituent matrices, we can write

$$\begin{aligned}\mathbf{H}^L\mathbf{X} &= \sum_{k=0}^{N-1} \sum_{\xi=a:=k\tau}^{b:=(k+1)\tau-1} \mathbf{H}^L_{(:,\xi)} \mathbf{X}_{(\xi,:)} := \sum_{k=0}^{N-1} \mathbf{H}^L_{(:,a:b)} \mathbf{X}_{(a:b,:)} = \sum_{k=0}^{N-1} \left(\mathbf{H}^L_{(a:b,:)}\right)^{\mathrm{T}} \mathbf{X}_{(a:b,:)} \\ &= \sum_{k=0}^{N-1} \left(\mathbf{H}^L_k\right)^{\mathrm{T}} \mathbf{X}_k \,.\end{aligned} \tag{22}$$

In the above, $\tau = \frac{M}{N}$ follows the definition in Sec. 4.1, $\mathbf{H}^L_k$ and $\mathbf{X}_k$ follow the notation in Fig. 2 and Fig. 4. A schematic for $\left(\mathbf{H}^L_k\right)^{\mathrm{T}} \mathbf{X}_k$ on the 0-th GPU is illustrated in Fig. 5. As shown in the figure, the multiplication involves $(\mathbf{H}^L_0)^{\mathrm{T}}$ of size $M \times \tau$ and $\mathbf{X}^{G_i}_0$ of size $\tau \times \frac{N_{\mathrm{orb}}}{P}$, resulting in matrix $(\mathbf{H}^L_0)^{\mathrm{T}}\mathbf{X}^{G_i}_0$ of size $M \times \frac{N_{\mathrm{orb}}}{P}$. The final outcome $\mathbf{H}^L\mathbf{X}^{G_i}$ can then be obtained by summing over k using `Allreduce` communication with MPI, as evident from the last equality of Eq. (22). We note that both $(\mathbf{H}^L_k)^{\mathrm{T}}$ and $\mathbf{X}_k$ are locally stored on the k-th GPU. Thus, for each matrix-matrix multiplication call during the Chebyshev filtering step, this approach avoids the MPI communications and overheads associated with transferring data between the device and the host memory for the off-diagonal block of the Hamiltonian matrix.

To understand the improvement in the efficiency by avoiding communicating the off-diagonal block of the Hamiltonian matrix, we present a performance comparison by computing $\mathbf{H}^L\mathbf{X}$ using the proposed algorithm and using the method with off-diagonal block communication (henceforth referred to as the general method). On the i-th processor, the general method is implemented by sending the off-diagonal blocks of the Hamiltonian matrix to all j-th ($j \neq i$) processors whose row-ownership coincide with the columns of the off-diagonal blocks. The off-diagonal block is then multiplied by the locally owned block of wavefunction matrix on the j-th processor and the result is reduced back to the i-th processor. To ensure the representability of this comparison, we choose the Hamiltonian matrix of Al_{147}, the benchmark system used for performance analysis in the later sections, to run this calculation. This benchmark calculation is run on the GreatLakes HPC cluster with each node comprising 2 Intel Xeon Gold 6148 CPUs with 40 physical cores per node. In this numerical experiment, the general method takes 481.92 cpu-secs and our proposed approach takes 232.18 cpu-secs. The $\sim 2\times$ improvement resulting from the communication efficiency, validates the use of the proposed approach.

Next, we turn our attention to leveraging the sparsity structure of $\mathbf{H}^L$ to further optimize the matrix-matrix multiplication kernel. As we noted earlier, the density of the diagonal square block $\mathbf{H}^{L(D)}_k$ can be large making sparse linear algebra operations inefficient [46]. To this end, we use different linear algebra libraries to treat the dense and the sparse blocks of the Hamiltonian matrix separately. As shown in Fig. 3, the diagonal blocks $\mathbf{H}^{L(D)}_k$ are stored as a dense matrix and the off-diagonal blocks $\mathbf{H}^{L(OD)}$ are stored

as a sparse matrix. Further, the matrix-matrix multiplication operation depicted in Fig. 5 is further split into a dense-dense multiplication for the diagonal block $\left(\left(\mathbf{H}_k^{L(D)}\right)^{\mathrm{T}} \mathbf{X}_k^{G_i}\right)$ and a sparse-dense multiplication for the off-diagonal block $\left(\left(\mathbf{H}_k^{L(OD)}\right)^{\mathrm{T}} \mathbf{X}_k^{G_i}\right)$. The dense-dense multiplication kernel is implemented using `cublasDgemm` provided by `cuBLAS` [49], NVIDIA GPU-accelerated implementation for basic linear algebra subroutines (BLAS). On the other hand, the sparse-dense multiplication kernel is computed with `cusparseDcsrmm` provided by `cuSPARSE` [46], NVIDIA GPU-accelerated implementation for sparse basic linear algebra subroutines. The two libraries are available in `CUDA Toolkit` or NVIDIA High performance computing software development kit (`NVIDA HPC SDK`). The `NVIDIA` library is usually accessible and optimally installed on most HPC clusters and thus used in this work. Once the computation is completed for the dense-dense and the sparse-dense matrix multiplication, the resultant matrices are assembled as $\left(\mathbf{H}_k^{L}\right)^{\mathrm{T}} \mathbf{X}_k^{G_i}$ (see Fig. 5 for a schematic plot for the 0-th processor). The assembled matrix $\left(\mathbf{H}_k^{L}\right)^{\mathrm{T}} \mathbf{X}_k^{G_i}$ is transferred back to the host memory. On the host memory, summation over k in Eq. (22) is completed using `MPI_Allreduce` within the wavefunction group.

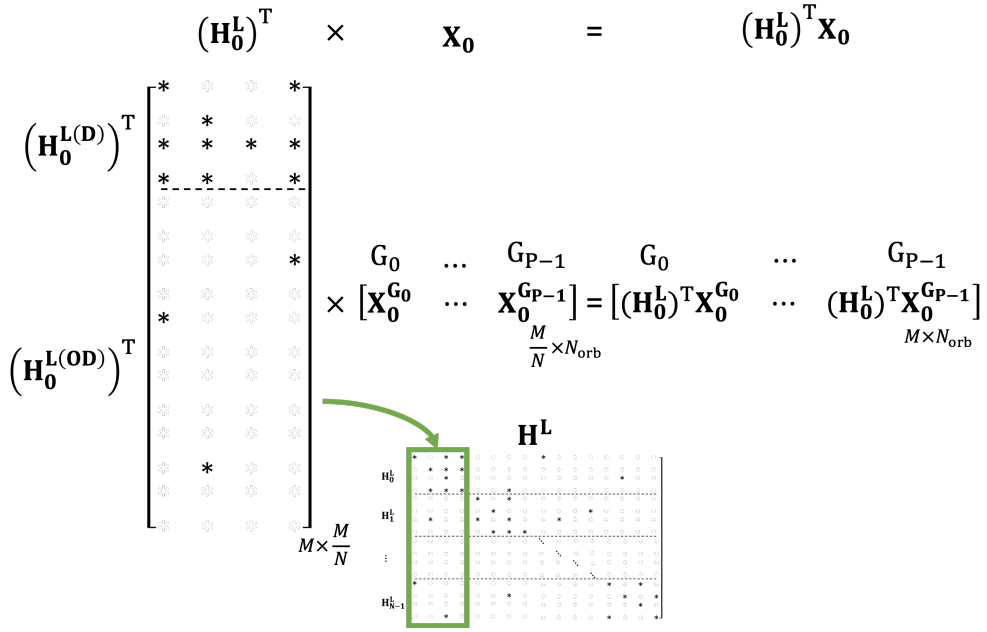


Figure 5: Schematic of the computation on 0-th processor in the evaluation of $\mathbf{H}^L\mathbf{X}$. As $\mathbf{H}^L$ is symmetric, the transpose of $\mathbf{H}_0^L$ is equivalent to the columns of $\mathbf{H}^L$ enclosed by the highligted box. $(\mathbf{H}_0^{L(D)})^{\mathrm{T}}$ and $(\mathbf{H}_0^{L(OD)})^{\mathrm{T}}$ denote the dense diagonal square block and the sparse off-diagonal part of the $(\mathbf{H}_0^L)^{\mathrm{T}}$ matrix, respectively.

4.3. $\mathbf{X}^T \times \mathbf{Y}$ *implementation*

In the Chebyshev filtered subspace iteration algorithm, upon computing the Chebyshev filtered vectors that represent a close approximation to the eigen-subspace of interest, the vectors are orthonormalized (step 2 in the Algorithm 1 and Algorithm 2) and the Kohn-Sham eigenvalue problem is projected onto this Chebyshev filtered subspace to solve the eigenvalue problem in this subspace (steps 3-5 in Algorithm 1). These entail operations of the form $\mathbf{X}^T \times \mathbf{Y}$ where $\mathbf{X}$ is the wavefunction matrix and $\mathbf{Y}$ is a matrix with the same dimensions as $\mathbf{X}$. In the operations involved, $\mathbf{Y}$ can be either $\mathbf{X}$ itself for evaluating the overlap matrix in the Algorithm 2, or $\mathbf{Y} = \mathbf{H}^L\mathbf{X}_F$ in the step 3 of the Algorithm 1.

Recalling the data layout of the wavefunction matrix $\mathbf{X}$ (and henceforth $\mathbf{Y}$) in Fig. 4, the evaluation of $\mathbf{X}^T \times \mathbf{Y}$ can be written as

$$\begin{aligned}\mathbf{X}^T\mathbf{Y} &= \begin{bmatrix} \mathbf{X}_\mathbf{0}^{\mathbf{G_0}\,T} & \mathbf{X}_\mathbf{1}^{\mathbf{G_0}\,T} & \cdots & \mathbf{X}_\mathbf{N-1}^{\mathbf{G_0}\,T} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{X}_\mathbf{0}^{\mathbf{G_{P-1}}\,T} & \mathbf{X}_\mathbf{1}^{\mathbf{G_{P-1}}\,T} & \cdots & \mathbf{X}_\mathbf{N-1}^{\mathbf{G_{P-1}}\,T} \end{bmatrix} \begin{bmatrix} \mathbf{Y}_\mathbf{0}^{\mathbf{G_0}} & \cdots & \mathbf{Y}_\mathbf{0}^{\mathbf{G_{P-1}}} \\ \mathbf{Y}_\mathbf{1}^{\mathbf{G_0}} & \cdots & \mathbf{Y}_\mathbf{1}^{\mathbf{G_{P-1}}} \\ \vdots & \ddots & \vdots \\ \mathbf{Y}_\mathbf{N-1}^{\mathbf{G_0}} & \cdots & \mathbf{Y}_\mathbf{N-1}^{\mathbf{G_{P-1}}} \end{bmatrix} \\ &= \sum_{i=0}^{N-1} \begin{bmatrix} \mathbf{X}_\mathbf{i}^{\mathbf{G_0}\,T} \\ \vdots \\ \mathbf{X}_\mathbf{i}^{\mathbf{G_{P-1}}\,T} \end{bmatrix} \begin{bmatrix} \mathbf{Y}_\mathbf{i}^{\mathbf{G_0}} & \cdots & \mathbf{Y}_\mathbf{i}^{\mathbf{G_{P-1}}} \end{bmatrix} = \sum_{i=0}^{N-1} \begin{bmatrix} \mathbf{X}_\mathbf{i}^{\mathbf{G_0}\,T}\mathbf{Y}_\mathbf{i}^{\mathbf{G_0}} & \cdots & \mathbf{X}_\mathbf{i}^{\mathbf{G_0}\,T}\mathbf{Y}_\mathbf{i}^{\mathbf{G_{P-1}}} \\ \vdots & \ddots & \vdots \\ \mathbf{X}_\mathbf{i}^{\mathbf{G_{P-1}}\,T}\mathbf{Y}_\mathbf{i}^{\mathbf{G_0}} & \cdots & \mathbf{X}_\mathbf{i}^{\mathbf{G_{P-1}}\,T}\mathbf{Y}_\mathbf{i}^{\mathbf{G_{P-1}}} \end{bmatrix}. \end{aligned} \tag{23}$$

In order to evaluate the sub-matrix $\mathbf{X}_\mathbf{i}^{\mathbf{G_a}\,T}\mathbf{Y}_\mathbf{i}^{\mathbf{G_b}}$, where $a, b = 0 \cdots P-1$, in each i matrix component in the last expression of Eq. 23,

$$\begin{bmatrix} \mathbf{X}_\mathbf{i}^{\mathbf{G_0}\,T}\mathbf{Y}_\mathbf{i}^{\mathbf{G_0}} & \cdots & \mathbf{X}_\mathbf{i}^{\mathbf{G_0}\,T}\mathbf{Y}_\mathbf{i}^{\mathbf{G_{P-1}}} \\ \vdots & \ddots & \vdots \\ \mathbf{X}_\mathbf{i}^{\mathbf{G_{P-1}}\,T}\mathbf{Y}_\mathbf{i}^{\mathbf{G_0}} & \cdots & \mathbf{X}_\mathbf{i}^{\mathbf{G_{P-1}}\,T}\mathbf{Y}_\mathbf{i}^{\mathbf{G_{P-1}}} \end{bmatrix},$$

multiple host-device data transfer and MPI calls are required to collect data from all other devices $G_{a\neq b}$. In order to optimize this operation, we create N sub-communicators corresponding to each i in Eq. 23 and re-distribute the column-wisely distributed $\mathbf{X_i}$ and $\mathbf{Y_i}$ to row-wise distribution as

$$\mathbf{X_i} := \begin{bmatrix} \mathbf{X}_\mathbf{i}^{\mathbf{G_0}} & \cdots & \mathbf{X}_\mathbf{i}^{\mathbf{G_{P-1}}} \end{bmatrix} \xmapsto{\text{column-wise to row-wise distribution}} \begin{bmatrix} \mathbf{X}_\mathbf{i}^{\mathbf{G_0,row}} \\ \vdots \\ \mathbf{X}_\mathbf{i}^{\mathbf{G_{P-1},row}} \end{bmatrix} := \mathbf{X_i}\,.$$

The $\mathbf{X}^T\mathbf{Y}$ evaluation can then be recast as

$$\mathbf{X}^T\mathbf{Y} = \sum_{i=0}^{N-1}\sum_{j=0}^{P-1} (\mathbf{X}_\mathbf{i}^{\mathbf{G_j,row}})^T \mathbf{Y}_\mathbf{i}^{\mathbf{G_j,row}}\,. \tag{24}$$

In this algorithm, the host-device data transfer and MPI communications are reduced to two sets of calls: (1) those before the operation to obtain row-wisely distributed matrix, which only involve the communication of small matrices $\mathbf{X}_\mathbf{i}^{\mathbf{G_j,row}}$ and $\mathbf{Y}_\mathbf{i}^{\mathbf{G_j,row}}$ within the i sub-communicator, and (2) the summation over i and j indices after the multiplication $(\mathbf{X}_\mathbf{i}^{\mathbf{G_j,row}})^T\mathbf{Y}_\mathbf{i}^{\mathbf{G_j,row}}$ is completed, which only communicates local small $N_{orb} \times N_{orb}$ matrix.

The Cholesky Gram-Schmidt process (step 2 of Algorithm 1) is listed in the Algorithm 2. The computation of the overlap matrix $\mathbf{S} = \mathbf{X}^T\mathbf{X}$ in the first step of CholGS process is completed using the aforementioned approach. Upon evaluating the overlap matrix $\mathbf{S}$, the overlap matrix is transferred to the device memory for Cholesky factorization $\mathbf{S} = \mathbf{L}\mathbf{L}^T$ and the computation of $\mathbf{X}_F = \mathbf{X}\mathbf{L}^{-1\,T}$. In step 3 of Algorithm 1, the evaluation of $\mathbf{H}_F^L$ includes a matrix-matrix multiplication between $\mathbf{H}^L$ and $\mathbf{X}_F$. Thus, it is natural to adopt the strategy discussed in Sec. 4.2 in evaluating $\mathbf{H}^L\mathbf{X}_F$. Upon evaluating $\mathbf{H}^L\mathbf{X}_F$, the $\mathbf{X}_F^T \times (\mathbf{H}^L\mathbf{X}_F)$ operation is completed by treating $\mathbf{Y} = \mathbf{H}^L\mathbf{X}_F$.

5. Results

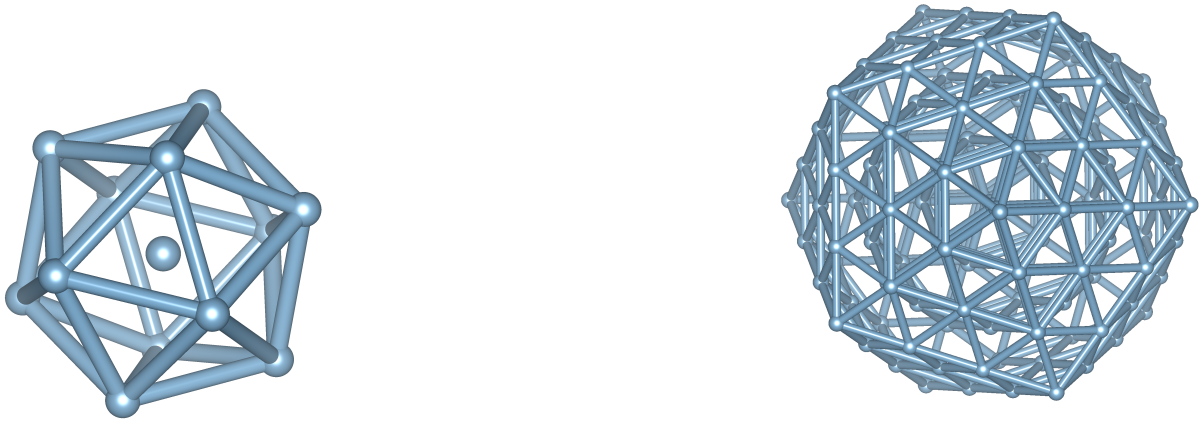
The systematic convergence, accuracy, and efficacy of the Tucker tensor basis and the tensor-structured algorithm for DFT calculations have been established in prior works [29, 30]. In particular, it was demonstrated that the Tucker tensor basis was systematically improvable and the basis discretization error decreased exponentially with increasing Tucker rank [29, 30], thus providing spectral convergence similar to plane-wave discretization. We refer to [30] for a comprehensive numerical study of the approximation properties of the localized Tucker tensor basis in DFT calculations. Further, a comparative study of the computational efficiency of the localized Tucker tensor basis with a plane-wave basis has revealed that the Tucker tensor basis is not only more efficient in terms of the number of basis functions required to achieve chemical accuracy, but also provides significant computational savings owing to the reduced-order scaling with system size. Benchmark calculations on both systems with and without a gap have revealed that the solution of the DFT problem in the Tucker tensor basis is substantially more efficient than the plane-wave basis for systems beyond 2,000 electrons, with up to 8× improvement in computational efficiency (measured in node-hrs) over plane-wave calculations conducted using Quantum Espresso (cf. [30]).

In the present work, besides providing the code for the TTDFT calculation, we focus on optimizing the most computationally expensive part of the calculation—the repetitive matrix-matrix multiplication kernel called during the Chebyshev filtering step and the $\mathbf{X}^T\mathbf{Y}$ evaluation —and further using GPU acceleration to improve the computational efficiency of this calculations. In order to assess the optimization realized, we use the benchmark systems from our previous work [30] comprising aluminum nano-particles and silicon quantum dots of various sizes. The aluminum nano-particles ranging from Al_{13} to Al_{6525} are constructed using icosahedral symmetry. The silicon quantum dots are constructed by rounding the diamond-structured silicon crystal and passivating the surface with hydrogen atoms. The silicon quantum-dots considered here range from $Si_{10}H_{16}$ to $Si_{6047}H_{1308}$. Ball and stick models for the two smallest clusters of both systems are depicted in Fig. 6 and Fig. 7. In order to conduct a performance analysis, we ran the benchmark calculations by solely using CPUs and compared with the acceleration obtained by utilizing GPUs for the matrix-matrix multiplication kernel in the Chebyshev filtered and subspace projection steps. Further, to ensure the accuracy of the code with GPU acceleration, we compare the ground-state energies obtained via CPU-only and CPU-GPU calculations of the two smallest clusters for both systems.

The numerical parameters used in the present study follow the previously converged CPU-based calculations of the benchmark systems [30]. In this work, we use local density approximation (LDA) for exchange-correlation functional [31, 32, 50] and a norm-conserving Troullier-Martin pseudopotential in Kleinmann-Bylander form [33, 34]. The Fermi-Dirac smearing temperature is set to $T = 500K$ for computing the fractional occupancy of the orbitals. The Chebyshev polynomial degree is chosen to be $10 - 20$ for various materials systems. The Tucker decomposition ranks [30] in the evaluation of the Hartree potential (R_H), in the representation of local part of the effective Kohn-Sham potential (R_V) and the non-local part of the effective potential (R_V^{nl}) are chosen to be $R_H = 40$, $R_V = 50$, $R_V^{nl} = 25$ for all aluminum nano-particles system and $R_H = 55$, $R_V = 55$ and $R_V^{nl} = 25$ for all silicon quantum dots system. The prescribed truncation tolerance for the Kohn-Sham Hamiltonian is set to 10^{-4} Ha for both aluminum nano-particles and silicon quantum dots according to the previous error analysis in [30]. The numerical parameters used are consistent for both the CPU- and GPU-based calculations in the performance analysis.

The performance benchmarks have been conducted on the Summit supercomputer, with each node comprising 2 IBM Power 9 CPUs (with 42 physical cores) and 6 NVIDIA Tesla V100 GPUs. We note that the number of nodes used to conduct the calculations is chosen such that the calculation is within the good parallel-scaling regime to obtain a representative measure of computational efficiency. In particular, for the larger systems considered in this work, the number of nodes are chosen such that the number of rows

of $\mathbf{H}_L$—the Kohn-Sham Hamiltonian matrix in the localized Tucker tensor basis—owned by each GPU is around 30,000, which maintains a good balance between memory limitation and parallel scaling efficiency.



(a) Al_{13} (b) Al_{147}

Figure 6: Schematics of the benchmark aluminum nano-particles.

(a) $Si_{10}H_{16}$ (b) $Si_{220}H_{144}$

Figure 7: Schematics of the benchmark silicon quantum dots.

5.1. Accuracy analysis of CPU and GPU implementation

In this section, we perform full ground-state calculations for the smaller benchmark systems of both aluminum nano-particles and silicon quantum dots to verify that the GPU implementation provides the same results as the CPU implementation. For the aluminum nano-particles, we choose Al_{13} and Al_{147}. The ball and stick model for the systems are provided in Fig. 6. For the silicon quantum dots, $Si_{10}H_{16}$ and $Si_{220}H_{144}$ are investigated. The ball and stick model for the silicon quantum dots are illustrated in Fig. 7. The converged results are tabulated in Table 1. We note that the results from the CPU-based and GPU-based calculations are identical up to the chemical accuracy of interest, with the differences at $O(10^{-8})$ eV in the

ground-state energy per atom. This small difference is possibly a result of the round-off error accumulations during the course of the ground-state calculation.

	Al_{13}	Al_{147}	$Si_{10}H_{16}$	$Si_{220}H_{144}$
CPU	-55.996571	-56.617932	-51.027192	-71.384192
GPU	-55.996571	-56.617932	-51.027192	-71.384192
|Error|	8.97×10^{-9}	4.35×10^{-8}	9.26×10^{-9}	1.21×10^{-8}

Table 1: Accuracy comparison of CPU and GPU implementation in ground state energy per atom (eV) for Al_{13}, Al_{147}, $Si_{10}H_{16}$, and $Si_{220}H_{144}$.

5.2. Performance analysis

In order to assess the computational efficiency derived from GPU acceleration, we compare the single SCF execution time in node-hours for the CPU-based implementation with that of the GPU acceleration. The computational times (in node-hours) for the various benchmark systems are provided in Table 2 and Table 3 for the aluminum nano-particles and silicon quantum dots, respectively. In particular, the breakdown of the single SCF computational time is provided for all the major steps of Algorithm 1: (i) ChF: Chebyshev filtering step; (ii) Orth: Orthogonalization of the Chebyshev filtered vectors; (iii) Sub proj: Projection of the Kohn-Sham Hamiltonian matrix onto the Chebyshev filtered subspace; (iv) Others: all other costs in the SCF, including solution of the eigenvalue problem in the Chebyshev filtered subspace. The total computational cost for a single SCF iteration and the speedup from GPU acceleration is also provided.

5.2.1. Aluminum nano-particles

Table 2 shows the single SCF breakdown of the computational times of the CPU-based calculations and that of the GPU accelerated calculations for aluminum nano-particles of various sizes. We remark that the main focus of this work is to optimize the matrix-matrix multiplication kernel in the Chebyshev filtering step—the most expensive step in each SCF iteration— and the $\mathbf{X}^T\mathbf{Y}$ kernel using GPU acceleration with the approach proposed in Sec. 4.2 and 4.3. The other parts of the calculation are done on CPUs, which are identified using * in the table. For the smallest benchmark system considered (Al_{13}), we note that the calculation using GPU acceleration is slower than the CPU-based calculation. This is due to the overhead costs for transferring data between the host and the device memory that are competing in small system sizes with the arithmetic efficiency gained from GPU acceleration. However, for all the other systems, we obtain overall GPU acceleration in the Chebyshev filtering step, the orthogonalization step,and the subspace projection step—the three parts of the algorithm that are affected by the GPU acceleration of the matrix-matrix multiplication kernel. In particular, for the largest system size considered, Al_{6525} nano-particle, we obtain $\sim$ 8.1×, $\sim$ 9.8× and $\sim$ 7.7× computational efficiency in the Chebyshev filtering step, orthonormalization step, and the subspace projection step, respectively, due to the GPU acceleration. This, in turn, provides a $\sim$ 5× improvement in the computational efficiency for the SCF iteration step that is representative of the full ground-state calculation.

5.2.2. Silicon quantum dots

Table 3 shows the single SCF breakdown of computational times of CPU-based and GPU-accelerated calculations for various sizes of silicon quantum dots. Similar to the aluminum nano-particles, the benefits of GPU acceleration improve with system size. Notably, for the largest quantum dot system $Si_{6047}H_{1308}$ which contains 7355 atoms, the computational efficiency gain by using GPU acceleration is $\sim$ 7.2× in

		ChF (node-hrs)	Orth (node-hrs)	Sub proj (node-hrs)	Others* (node-hrs)	Time/SCF (node-hrs)	$\frac{\text{GPU}}{\text{CPU}}$
Al_{13}	CPU	1.87E-04	8.70E-05	3.57E-05	1.42E-04	4.52E-04	0.76
	GPU	2.01E-04	2.02E-04	3.05E-05	1.58E-04	5.92E-04	
Al_{147}	CPU	3.21E-02	2.14E-04	5.32E-03	1.12E-02	4.88E-02	1.29
	GPU	2.41E-02	9.42E-05	2.73E-03	1.08E-02	3.77E-02	
Al_{561}	CPU	0.331	0.008	0.047	0.156	0.542	1.69
	GPU	0.148	0.002	0.012	0.158	0.320	
Al_{2057}	CPU	3.724	0.228	0.482	1.327	5.761	3.11
	GPU	0.538	0.020	0.073	1.221	1.852	
Al_{6525}	CPU	30.119	6.192	3.422	4.132	43.865	4.98
	GPU	3.715	0.631	0.442	4.023	8.811	

Table 2: Breakdown of single-SCF computational times (in node-hours) for CPU-based and GPU-based calculations for the benchmark systems comprising Al nano-particles. The columns marked with asterisk * are computed on the host (CPU) without GPU optimization.

Chebyshev filtering step, $\sim 9.1\times$ for the orthonormalization step, and $\sim 7.2\times$ for the subspace projection step. The computational efficiency gain for the full SCF iteration is $\sim 5.3\times$.

		ChF (node-hrs)	Orth (node-hrs)	Sub proj (node-hrs)	Others* (node-hrs)	Time/SCF (node-hrs)	$\frac{\text{GPU}}{\text{CPU}}$
$Si_{10}H_{16}$	CPU	1.37E-03	5.99E-04	1.83E-04	2.42E-03	4.57E-03	1.05
	GPU	1.42E-03	6.12E-04	1.29E-04	2.21E-03	4.37E-03	
$Si_{220}H_{144}$	CPU	6.12E-02	1.10E-03	6.99E-03	2.33E-02	9.26E-02	1.69
	GPU	2.91E-02	7.10E-04	2.81E-03	2.22E-02	5.48E-02	
$Si_{525}H_{276}$	CPU	0.515	0.018	0.067	0.214	0.814	1.97
	GPU	0.185	0.003	0.024	0.201	0.413	
$Si_{1214}H_{504}$	CPU	2.132	0.132	0.258	0.552	3.074	2.55
	GPU	0.622	0.019	0.074	0.491	1.206	
$Si_{6047}H_{1308}$	CPU	38.511	6.525	4.259	3.515	52.810	5.25
	GPU	5.324	0.721	0.596	3.411	10.052	

Table 3: Breakdown of single-SCF computational times (in node-hours) for CPU-based and GPU-based calculations for the benchmark systems comprising silicon quantum dots. The columns marked with asterisk * are computed on the host (CPU) without GPU optimization.

The numerical analysis has shown that the proposed GPU acceleration provides $\sim 8.1\times$ improvement in the Chebyshev filtering procedure, the most computationally intensive part, and $\sim 5.3\times$ overall improvement over pure CPU-based implementation for the largest studied case. Comparing with the DFT-FE code [22, 23]—a massively parallel GPU accelerated real-space code based on finite-element discretization, which also uses the Chebyshev filtered subspace iteration scheme for solving the Kohn-Sham eigenvalue problem—the GPU accelerated version of the DFT-FE code outperforms the CPU implementation by $\sim 20\times$ in the Chebyshev filtering step, with overall CPU-GPU speedups for the single SCF iteration to be $\sim 15-20\times$ [23]. The significant GPU performance of DFT-FE in the Chebyshev filtering step is due to the finite-element structure, where the matrix-matrix multiplication between the sparse Hamiltonian matrix and the dense wavefunction matrix can be recast as smaller dense-dense matrix multiplications at the finite

element cell level that efficiently utilizes the GPU parallelism. However, this is not possible in the present approach as the Tucker tensor basis, while localized, does not have an elemental structure. Nevertheless, we note that the Tucker tensor basis requires far fewer basis functions (10 – 20 fold fewer basis functions) to achieve chemical accuracy in comparison to the finite element basis, and thus has a smaller memory footprint enabling large-scale DFT calculations on computing platforms with modest computational resources.

We also note that, in this work, `cuSparse` library is used to evaluate the sparse-dense matrix-matrix multiplication due to its accessibility and optimal installation on most HPC clusters. However, the sparse-dense matrix-matrix kernel provided by the `cuSparse` library has been reported to access memory sub-optimally and hence not utilize the best performance that modern GPU architectures can provide [51]. Recent efforts have proposed a more efficient algorithm that utilizes the modern GPU acceleration scheme [51–53] for sparse-dense matrix multiplication appearing in this work. The improvement in GPU acceleration of the TTDFT code could be even greater if the `cuSparse` kernel is replaced by a more efficient sparse-dense matrix-matrix multiplication implementation. Further, exploiting mixed precision ideas, asynchronous compute and communication (already adopted in DFT-FE), and GPU porting other aspects of the calculation can further enhance the performance of the TTDFT code and remain key aspects for future efforts.

6. Summary

We have presented the TTDFT code with GPU acceleration for the main compute intensive kernels of the calculation. In particular, the TTDFT algorithm is based on using a systematically convergent localized basis that is generated from an additive separable approximation of the Kohn-Sham Hamiltonian [30]. The solution to the discrete Kohn-Sham problem is computed via the Chebyshev filtered subspace iteration [37] method. The compute intensive kernels in the TTDFT code that involve matrix-matrix multiplication of a symmetric sparse matrix (Hamiltonian matrix) and a dense matrix (wavefunction matrix) have been GPU accelerated. The benchmark studies show a substantial improvement for the GPU-accelerated steps of the algorithm— $\sim 7 - 10\times$ for the largest system sizes—which improves the overall computational efficiency of the calculation. We note that recent studies have shown that the TTDFT algorithm can substantially outperform plane-wave implementations for large-scale systems [30], owing to the reduced-order scaling with system size. The present GPU-based TTDFT code is a further step towards enabling systematically convergent and computationally efficient large-scale DFT calculations.

Acknowledgments

We gratefully acknowledge the support of the Air Force Office of Scientific Research through grant number FA9550-21-1-0302 under the auspices of which this work was conducted. V.G. also gratefully acknowledges the support of the Army Research Office through the DURIP grant W911NF1810242, which provided computational resources for this work. This research also used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

References

[1] P. Hohenberg, W. Kohn, Inhomogeneous electron gas, Phys. Rev. 136 (1964) B864–B871. doi:10.1103/PhysRev.136.B864.

[2] W. Kohn, L. J. Sham, Self-consistent equations including exchange and correlation effects, Phys. Rev. 140 (1965) A1133–A1138. doi:10.1103/PhysRev.140.A1133.

[3] A. D. Becke, Perspective: Fifty years of density-functional theory in chemical physics, The Journal of Chemical Physics 140 (18) (2014) 18A301. doi:10.1063/1.4869598.

[4] R. Jones, Density functional theory: Its origins, rise to prominence, and future, Reviews of Modern Physics 87 (3) (2015) 897–923. doi:10.1103/revmodphys.87.897.

[5] N. Mardirossian, M. Head-Gordon, Thirty years of density functional theory in computational chemistry: an overview and extensive assessment of 200 density functionals, Molecular Physics 115 (19) (2017) 2315–2372. doi:10.1080/00268976.2017.1333644.

[6] J. P. Perdew, A. Ruzsinszky, J. Tao, V. N. Staroverov, G. E. Scuseria, G. I. Csonka, Prescription for the design and selection of density functional approximations: More constraint satisfaction with fewer fits, The Journal of Chemical Physics 123 (6) (2005) 062201. doi:10.1063/1.1904565.

[7] M. G. Medvedev, I. S. Bushmarinov, J. Sun, J. P. Perdew, K. A. Lyssenko, Density functional theory is straying from the path toward the exact functional, Science 355 (6320) (2017) 49–52. doi:10.1126/science.aah5975.

[8] K. P. Kepp, Comment on "density functional theory is straying from the path toward the exact functional", Science 356 (6337) (2017) 496.2–496. doi:10.1126/science.aam9364.

[9] G. Kresse, J. Furthmüller, Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set, Phys. Rev. B 54 (1996) 11169–11186. doi:10.1103/PhysRevB.54.11169.

[10] X. Gonze, J.-M. Beuken, R. Caracas, F. Detraux, M. Fuchs, G.-M. Rignanese, L. Sindic, M. Verstraete, G. Zerah, F. Jollet, M. Torrent, A. Roy, M. Mikami, P. Ghosez, J.-Y. Raty, D. Allan, First-principles computation of material properties: the abinit software project, Computational Materials Science 25 (3) (2002) 478 – 492. doi:10.1016/S0927-0256(02)00325-7.

[11] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. B. Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. D. Corso, S. de Gironcoli, P. Delugas, R. A. D. Jr, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Küçükbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N. L. Nguyen, H.-V. Nguyen, A. O. de-la Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, S. Baroni, Advanced capabilities for materials modelling with QUANTUM ESPRESSO, Journal of Physics: Condensed Matter 29 (46) (2017) 465901. doi:10.1088/1361-648X/aa8f79.

[12] S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. I. J. Probert, K. Refson, M. C. Payne, First principles methods using CASTEP, Zeitschrift für Kristallographie - Crystalline Materials 220 (5-6) (2005) 567–570. doi:10.1524/zkri.220.5.567.65075.

[13] W. J. Hehre, R. F. Stewart, J. A. Pople, Self-consistent molecular-orbital methods. i. use of gaussian expansions of slater-type atomic orbitals, The Journal of Chemical Physics 51 (6) (1969) 2657–2664. doi:10.1063/1.1672392.

[14] E. Epifanovsky, A. T. B. Gilbert, X. Feng, J. Lee, Y. Mao, N. Mardirossian, P. Pokhilko, A. F. White, M. P. Coons, A. L. Dempwolff, Z. Gan, D. Hait, P. R. Horn, L. D. Jacobson, I. Kaliman, J. Kussmann, A. W. Lange, K. U. Lao, D. S. Levine, J. Liu, S. C. McKenzie, A. F. Morrison, K. D. Nanda, F. Plasser, D. R. Rehn, M. L. Vidal, Z.-Q. You, Y. Zhu, B. Alam, B. J. Albrecht, A. Aldossary, E. Alguire, J. H. Andersen, V. Athavale, D. Barton, K. Begam, A. Behn, N. Bellonzi, Y. A. Bernard, E. J. Berquist, H. G. A. Burton, A. Carreras, K. Carter-Fenk, R. Chakraborty, A. D. Chien, K. D. Closser, V. Cofer-Shabica, S. Dasgupta, M. de Wergifosse, J. Deng, M. Diedenhofen, H. Do, S. Ehlert, P.-T. Fang, S. Fatehi, Q. Feng, T. Friedhoff, J. Gayvert, Q. Ge, G. Gidofalvi, M. Goldey, J. Gomes, C. E. González-Espinoza, S. Gulania, A. O. Gunina, M. W. D. Hanson-Heine, P. H. P. Harbach, A. Hauser, M. F. Herbst, M. Hernández Vera, M. Hodecker, Z. C. Holden, S. Houck, X. Huang, K. Hui, B. C. Huynh, M. Ivanov, A. Jász, H. Ji, H. Jiang, B. Kaduk, S. Kähler, K. Khistyaev, J. Kim, G. Kis, P. Klunzinger, Z. Koczor-Benda, J. H. Koh, D. Kosenkov, L. Koulias, T. Kowalczyk, C. M. Krauter, K. Kue, A. Kunitsa, T. Kus, I. Ladjánszki, A. Landau, K. V. Lawler, D. Lefrancois, S. Lehtola, R. R. Li, Y.-P. Li, J. Liang, M. Liebenthal, H.-H. Lin, Y.-S. Lin, F. Liu, K.-Y. Liu, M. Loipersberger, A. Luenser, A. Manjanath, P. Manohar, E. Mansoor, S. F. Manzer, S.-P. Mao, A. V. Marenich, T. Markovich, S. Mason, S. A. Maurer, P. F. McLaughlin, M. F. S. J. Menger, J.-M. Mewes, S. A. Mewes, P. Morgante, J. W. Mullinax, K. J. Oosterbaan, G. Paran, A. C. Paul, S. K. Paul, F. Pavošević, Z. Pei, S. Prager, E. I. Proynov, A. Rák, E. Ramos-Cordoba, B. Rana, A. E. Rask, A. Rettig, R. M. Richard, F. Rob, E. Rossomme, T. Scheele, M. Scheurer, M. Schneider, N. Sergueev, S. M. Sharada, W. Skomorowski, D. W. Small, C. J. Stein, Y.-C. Su, E. J. Sundstrom, Z. Tao, J. Thirman, G. J. Tornai, T. Tsuchimochi, N. M. Tubman, S. P. Veccham, O. Vydrov, J. Wenzel, J. Witte, A. Yamada, K. Yao, S. Yeganeh, S. R. Yost, A. Zech, I. Y. Zhang, X. Zhang, Y. Zhang, D. Zuev, A. Aspuru-Guzik, A. T. Bell, N. A. Besley, K. B. Bravaya, B. R. Brooks, D. Casanova, J.-D. Chai, S. Coriani, C. J. Cramer, G. Cserey, A. E. DePrince, R. A. DiStasio, A. Dreuw, B. D. Dunietz, T. R. Furlani, W. A. Goddard, S. Hammes-Schiffer, T. Head-Gordon, W. J. Hehre, C.-P. Hsu, T.-C. Jagau, Y. Jung, A. Klamt, J. Kong, D. S. Lambrecht, W. Liang, N. J. Mayhall, C. W. McCurdy, J. B. Neaton, C. Ochsenfeld, J. A. Parkhill, R. Peverati, V. A. Rassolov, Y. Shao, L. V. Slipchenko, T. Stauch, R. P. Steele, J. E. Subotnik, A. J. W. Thom, A. Tkatchenko, D. G. Truhlar, T. Van Voorhis, T. A. Wesolowski, K. B. Whaley, H. L. Woodcock, P. M. Zimmerman, S. Faraji, P. M. W. Gill, M. Head-Gordon, J. M. Herbert, A. I. Krylov, Software for the frontiers of quantum chemistry: An overview of developments in the q-chem 5 package, The Journal of Chemical Physics 155 (8) (2021) 084801.

doi:10.1063/5.0055522.

[15] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. V. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. J. Bearpark, J. J. Heyd, E. N. Brothers, K. N. Kudin, V. N. Staroverov, T. A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. P. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo, R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman, D. J. Fox, Gaussian˜16 Revision C.01, gaussian Inc. Wallingford CT (2016).

[16] E. Aprà, E. J. Bylaska, W. A. de Jong, N. Govind, K. Kowalski, T. P. Straatsma, M. Valiev, H. J. J. van Dam, Y. Alexeev, J. Anchell, V. Anisimov, F. W. Aquino, R. Atta-Fynn, J. Autschbach, N. P. Bauman, J. C. Becca, D. E. Bernholdt, K. Bhaskaran-Nair, S. Bogatko, P. Borowski, J. Boschen, J. Brabec, A. Bruner, E. Cauët, Y. Chen, G. N. Chuev, C. J. Cramer, J. Daily, M. J. O. Deegan, T. H. Dunning, M. Dupuis, K. G. Dyall, G. I. Fann, S. A. Fischer, A. Fonari, H. Früchtl, L. Gagliardi, J. Garza, N. Gawande, S. Ghosh, K. Glaesemann, A. W. Götz, J. Hammond, V. Helms, E. D. Hermes, K. Hirao, S. Hirata, M. Jacquelin, L. Jensen, B. G. Johnson, H. Jónsson, R. A. Kendall, M. Klemm, R. Kobayashi, V. Konkov, S. Krishnamoorthy, M. Krishnan, Z. Lin, R. D. Lins, R. J. Littlefield, A. J. Logsdail, K. Lopata, W. Ma, A. V. Marenich, J. Martin del Campo, D. Mejia-Rodriguez, J. E. Moore, J. M. Mullin, T. Nakajima, D. R. Nascimento, J. A. Nichols, P. J. Nichols, J. Nieplocha, A. Otero-de-la Roza, B. Palmer, A. Panyala, T. Pirojsirikul, B. Peng, R. Peverati, J. Pittner, L. Pollack, R. M. Richard, P. Sadayappan, G. C. Schatz, W. A. Shelton, D. W. Silverstein, D. M. A. Smith, T. A. Soares, D. Song, M. Swart, H. L. Taylor, G. S. Thomas, V. Tipparaju, D. G. Truhlar, K. Tsemekhman, T. Van Voorhis, A. Vázquez-Mayagoitia, P. Verma, O. Villa, A. Vishnu, K. D. Vogiatzis, D. Wang, J. H. Weare, M. J. Williamson, T. L. Windus, K. Woliński, A. T. Wong, Q. Wu, C. Yang, Q. Yu, M. Zacharias, Z. Zhang, Y. Zhao, R. J. Harrison, Nwchem: Past, present, and future, The Journal of Chemical Physics 152 (18) (2020) 184102. doi:10.1063/5.0004997.

[17] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, M. Scheffler, Ab initio molecular simulations with numeric atom-centered orbitals, Computer Physics Communications 180 (11) (2009) 2175–2196. doi:10.1016/j.cpc.2009.06.022.

[18] E. Cancès, C. LeBris, Y. Maday, G. Turinici, Towards reduced basis approaches in ab initio electronic structure computations, Journal of Scientific Computing 17 (1) (2002) 461–469. doi:10.1023/a:1015150025426.

[19] L. Lin, J. Lu, L. Ying, W. E, Adaptive local basis set for kohn–sham density functional theory in a discontinuous galerkin framework i: Total energy calculation, Journal of Computational Physics 231 (4) (2012) 2140–2154. doi:10.1016/j.jcp.2011.11.032.

[20] L. Kronik, A. Makmal, M. L. Tiago, M. M. G. Alemany, M. Jain, X. Huang, Y. Saad, J. R. Chelikowsky, PARSEC – the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nanostructures, physica status solidi (b) 243 (5) (2006) 1063–1079. doi:10.1002/pssb.200541463.

[21] Q. Xu, A. Sharma, B. Comer, H. Huang, E. Chow, A. J. Medford, J. E. Pask, P. Suryanarayana, SPARC: Simulation package for ab-initio real-space calculations, SoftwareX 15 (2021) 100709. doi:10.1016/j.softx.2021.100709.

[22] P. Motamarri, S. Das, S. Rudraraju, K. Ghosh, D. Davydov, V. Gavini, DFT-FE – A massively parallel adaptive finite-element code for large-scale density functional theory calculations, Computer Physics Communications 246 (2020) 106853. doi:10.1016/j.cpc.2019.07.016.

[23] S. Das, P. Motamarri, V. Subramanian, D. M. Rogers, V. Gavini, DFT-FE 1.0: A massively parallel hybrid CPU-GPU density functional theory code using finite-element discretization (2022). doi:10.48550/ARXIV.2203.07820.

[24] L. E. Ratcliff, W. Dawson, G. Fisicaro, D. Caliste, S. Mohr, A. Degomme, B. Videau, V. Cristiglio, M. Stella, M. D'Alessandro, S. Goedecker, T. Nakajima, T. Deutsch, L. Genovese, Flexibilities of wavelets as a computational basis set for large-scale electronic structure calculations, The Journal of Chemical Physics 152 (19) (2020) 194110. doi:10.1063/5.0004792.

[25] P. Motamarri, M. Nowak, K. Leiter, J. Knap, V. Gavini, Higher-order adaptive finite-element methods for Kohn–Sham density functional theory, Journal of Computational Physics 253 (2013) 308–343. doi:10.1016/j.jcp.2013.06.042.

[26] S. Das, P. Motamarri, V. Gavini, B. Turcksin, Y. W. Li, B. Leback, Fast, scalable and accurate finite-element based ab initio calculations using mixed precision computing: 46 pflops simulation of a metallic dislocation system, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19, Association for Computing Machinery, New York, NY, USA, 2019. doi:10.1145/3295500.3357157.

[27] W. Hackbusch, B. N. Khoromskij, Tensor-product approximation to operators and functions in high dimensions, Journal of Complexity 23 (4) (2007) 697 – 714, festschrift for the 60th Birthday of Henryk Woźniakowski. doi:10.1016/j.jco.2007.03.007.

[28] T. Blesgen, V. Gavini, V. Khoromskaia, Approximation of the electron density of aluminium clusters in tensor-product format,

Journal of Computational Physics 231 (6) (2012) 2551 – 2564. doi:10.1016/j.jcp.2011.12.009.
[29] P. Motamarri, V. Gavini, T. Blesgen, Tucker-tensor algorithm for large-scale Kohn-Sham density functional theory calculations, Physical Review B 93 (12) (2016) 125104. doi:10.1103/PhysRevB.93.125104.
[30] C.-C. Lin, P. Motamarri, V. Gavini, Tensor-structured algorithm for reduced-order scaling large-scale Kohn–Sham density functional theory calculations, npj Computational Materials 7 (1) (2021) 50. doi:10.1038/s41524-021-00517-5.
[31] D. M. Ceperley, B. J. Alder, Ground state of the electron gas by a stochastic method, Phys. Rev. Lett. 45 (1980) 566–569. doi:10.1103/PhysRevLett.45.566.
[32] J. P. Perdew, A. Zunger, Self-interaction correction to density-functional approximations for many-electron systems, Phys. Rev. B 23 (1981) 5048–5079. doi:10.1103/PhysRevB.23.5048.
[33] N. Troullier, J. L. Martins, Efficient pseudopotentials for plane-wave calculations, Phys. Rev. B 43 (1991) 1993–2006. doi:10.1103/PhysRevB.43.1993.
[34] L. Kleinman, D. M. Bylander, Efficacious form for model pseudopotentials, Phys. Rev. Lett. 48 (1982) 1425–1428. doi:10.1103/PhysRevLett.48.1425.
[35] T. G. Kolda, B. W. Bader, Tensor Decompositions and Applications, SIAM Review 51 (3) (2009) 455–500. doi:10.1137/07070111X.
[36] Y. Zhou, Y. Saad, M. L. Tiago, J. R. Chelikowsky, Parallel self-consistent-field calculations via Chebyshev-filtered subspace acceleration, Phys. Rev. E 74 (2006) 066704. doi:10.1103/PhysRevE.74.066704.
[37] Y. Zhou, Y. Saad, M. L. Tiago, J. R. Chelikowsky, Self-consistent-field calculations using chebyshev-filtered subspace iteration, Journal of Computational Physics 219 (1) (2006) 172–184. doi:10.1016/j.jcp.2006.03.017.
[38] L. Grasedyck, D. Kressner, C. Tobler, A literature survey of low-rank tensor approximation techniques, GAMM-Mitteilungen 36 (1) (2013) 53–78. doi:10.1002/gamm.201310004.
[39] W. Hackbusch, Tensor Spaces and Numerical Tensor Calculus, Vol. 42 of Springer Series in Computational Mathematics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-28027-6.
[40] W. Austin, G. Ballard, T. G. Kolda, Parallel tensor compression for large-scale scientific data, in: IPDPS'16: Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium, 2016, pp. 912–922. doi:10.1109/IPDPS.2016.67.
[41] G. Ballard, A. Klinvex, T. G. Kolda, TuckerMPI: A parallel C++/MPI software package for large-scale data compression via the tucker tensor decomposition, ACM Trans. Math. Softw. 46 (2) (2020) 13:1–13:31. doi:10.1145/3378445.
[42] V. Ozolinš, R. Lai, R. Caflisch, S. Osher, Compressed modes for variational problems in mathematics and physics, Proceedings of the National Academy of Sciences 110 (46) (2013) 18368–18373. doi:10.1073/pnas.1318679110.
[43] R. Lai, S. Osher, A splitting method for orthogonality constrained problems, Journal of Scientific Computing 58 (2) (2014) 431–449. doi:10.1007/s10915-013-9740-x.
[44] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-optimal parallel and sequential QR and LU factorizations, SIAM Journal on Scientific Computing 34 (1) (2012) A206–A239. doi:10.1137/080731992.
[45] A. Stathopoulos, K. Wu, A block orthogonalization procedure with constant synchronization requirements, SIAM Journal on Scientific Computing 23 (6) (2002) 2165–2182. doi:10.1137/s1064827500370883.
[46] NVIDIA, cusparse, release: Nov 28, 2019 (2019).
URL https://docs.nvidia.com/cuda/archive/10.2/cusparse/index.html
[47] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. Fagg, E. Gabriel, J. Dongarra, Performance analysis of mpi collective operations, in: 19th IEEE International Parallel and Distributed Processing Symposium, 2005, pp. 8 pp.–. doi:10.1109/IPDPS.2005.335.
[48] W. Gropp, E. Lusk, R. Thakur, Using MPI-2, 2019. doi:10.7551/mitpress/7055.001.0001.
[49] NVIDIA, cublas, release: Nov 28, 2019 (2019).
URL https://docs.nvidia.com/cuda/archive/10.2/cublas/index.html
[50] J. P. Perdew, Y. Wang, Accurate and simple analytic representation of the electron-gas correlation energy, Phys. Rev. B 45 (1992) 13244–13249. doi:10.1103/PhysRevB.45.13244.
[51] S. Shi, Q. Wang, X. Chu, Efficient sparse-dense matrix-matrix multiplication on gpus using the customized sparse storage format, in: 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), 2020, pp. 19–26. doi:10.1109/ICPADS51040.2020.00013.
[52] M. Parger, M. Winter, D. Mlakar, M. Steinberger, spECK, in: Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, 2020. doi:10.1145/3332466.3374521.
[53] P. Jiang, C. Hong, G. Agrawal, A novel data transformation and execution strategy for accelerating sparse matrix multiplication on GPUs, in: Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, 2020. doi:10.1145/3332466.3374546.