Accelerating Quantum Light-Matter Dynamics on Graphics Processing Units

Taufeq Mohammed Razakh
Collaboratory for Advanced Computing and Simulations
University of Southern California
Los Angeles, CA, USA
razakh@usc.edu

Thomas Linker
Stanford PULSE Insitute
SLAC National Accelerator Laboratory
Menlo Park, CA, USA
tlinker@slac.stanford.edu

Ye Luo
Computational Science Division
Argonne National Laboratory
Lemont, IL, USA
yeluo@anl.gov

Rajiv K. Kalia
Collaboratory for Advanced Computing and Simulations
University of Southern California
Los Angeles, CA, USA
rkalia@usc.edu

Ken-ichi Nomura
Collaboratory for Advanced Computing and Simulations
University of Southern California
Los Angeles, CA, USA
knomura@usc.edu

Priya Vashishta
Collaboratory for Advanced Computing and Simulations
University of Southern California
Los Angeles, CA, USA
priyav@usc.edu

Aiichiro Nakano
Collaboratory for Advanced Computing and Simulations
University of Southern California
Los Angeles, CA, USA
anakano@usc.edu

***Abstract*—To study light-matter interaction, we have developed a linear-scaling DC-MESH (divide-and-conquer Maxwell-Ehrenfest-surface hopping) simulation algorithm, where our globally-sparse and locally-dense electronic solvers, multiple time-scale splitting, and shadow dynamics achieve high scalability and allow the most compute-intensive quantum dynamics kernel based on time-dependent density functional theory to reside on GPU with minimal CPU-GPU data transfer. GPU computation based on OpenMP target constructs is accelerated by: (i) data and loop reordering for better memory access patterns; (ii) hierarchical GPU offloading using teams-distribute and parallel constructs, respectively, for coarse and fine computations; (iii) algebraic 'BLASification' of the nonlocal computational bottleneck; and (iv) GPU-resident data structures facilitated by custom C++ class initializer and destructor based on OpenMP target data constructs. We have thereby achieved 644-fold speedup on Nvidia A100 GPU over AMD EPYC 7543 CPU of the Polaris computer at Argonne Leadership Computing Facility. In addition, the DC-MESH code exhibits a weak-scaling parallel efficiency of 96.73% on 256 nodes (or 1,024 GPUs) of Polaris for 5,120-atom $PbTiO_3$ material. This enables the study of light-induced topological switching for future ultrafast and ultralow-power ferroelectric topotronics applications.**

Keywords—quantum dynamics, light-matter interaction, time-dependent density functional theory, algebraic BLASification, GPU acceleration

I. Introduction

How light and matter interact is one of the most fundamental scientific questions. For example, nonlinear interaction of high-intensity laser with matter generates ultrashort attosecond (10^{-18} second) pulses. This discovery has opened up the new era of attosecond physics, for which Agostini, Krausz, and L'Hullier received the 2023 Nobel prize in physics [1]. While high-end supercomputing has successfully been applied to quantum-mechanical study of *static* materials properties [2-7], its application to *quantum dynamics* (QD) such as attosecond physics remains in its infancy [8].

Nonlinear, non-steady dynamics like attosecond light-matter interaction is theoretically described by Maxwell's equations for light along with time-dependent density functional theory (TDDFT) for electrons [9, 10]. This is a multiscale physics problem encompassing fast (10^{-18} second) elementary processes of light-electron coupling and slower (10^{-12} second) materials response through electron-atom coupling. In addition to this temporal disparity, disparate length scales need be accounted for, ranging from electronic wave functions (10^{-10} m) to large topological features of quantum materials (10^{-6} m) [11, 12]. For static quantum properties, the length-scale problem has been addressed by linear-scaling density functional theory (DFT) algorithms [13], in which the $O(N^3)$ complexity of the DFT problem (1998 Nobel prize in chemistry for Walter Kohn; N is the number of electrons) [14] is reduced to $O(N)$ based on the physical data locality principle called quantum nearsightedness [15]. Among various $O(N)$ DFT approaches, the most scalable on high-end supercomputers is the divide-and-conquer (DC) DFT algorithm, where local electronic Kohn-Sham (KS) wave functions and global KS potential are determined in global-local self-consistent-field (SCF) iterations [16, 17].

To move on to the time-scale challenge in QD, the key insight is that fundamental physics equations are all local at the finest spatiotemporal scales, *i.e.*, simple partial differential equations with differential operators acting locally. On the other hand, coarse-grained schemes to approximately describe complex chemical interactions often come with an excessive computational cost of nonlocal operations in space and time. Simple data parallelism in the former—which we call Local Field Dynamics (LFD)—fits naturally to hardware accelerators such as graphic processing unit (GPU). On the other hand,

979-8-3503-6460-6/24/$31.00 ©2024 IEEE
DOI 10.1109/IPDPSW63119.2024.00176

complex chemical interaction in the latter—which we call Quantum eXcitation Molecular Dynamics (QXMD)—can take advantage of complex instruction sets in central processing unit (CPU). To minimize data transfer between CPU and GPU, we adopt a shadow dynamics approach [18], in which a GPU-resident proxy is solved to effectively describe the action of LFD on QXMD. In this way, LFD-QXMD handshaking is reduced to minimal, *i.e.*, electronic occupation numbers, which are negligible compared to the large memory footprint of many KS wave functions [19].

Dynamics involving excited electrons and atoms is called nonadiabatic quantum molecular dynamics (NAQMD) [20-22], for which there are two major approaches. The first approach called Ehrenfest dynamics relies on TDDFT equations for electrons, which in turn dictates interatomic interaction for molecular dynamics (MD) at short time scales [21]. At longer time scales, adiabatic electronic states with fixed atomic positions are good representation of excited electronic states, and the second approach called surface hopping describes transitions between the excited states *via* nonadiabatic coupling due to atomic motions [21]. Combined with Maxwell equations for light, we have developed a multiscale NAQMD approach within a DC scheme named DC-MESH (divide-and-conquer Maxwell-Ehrenfest-surface hopping). A preliminary version of DC-MESH with limited functionality is described in Ref. [12]. Several software packages exist for Maxwell+TDDFT simulations on parallel computers such as Octopus [23] and SALMON [24], where a multiscale DC approach has been applied to the Maxwell-Ehrenfest subproblem [24], but not to the entire Maxwell-Ehrenfest-surface hopping problem.

II. DC-MESH Method

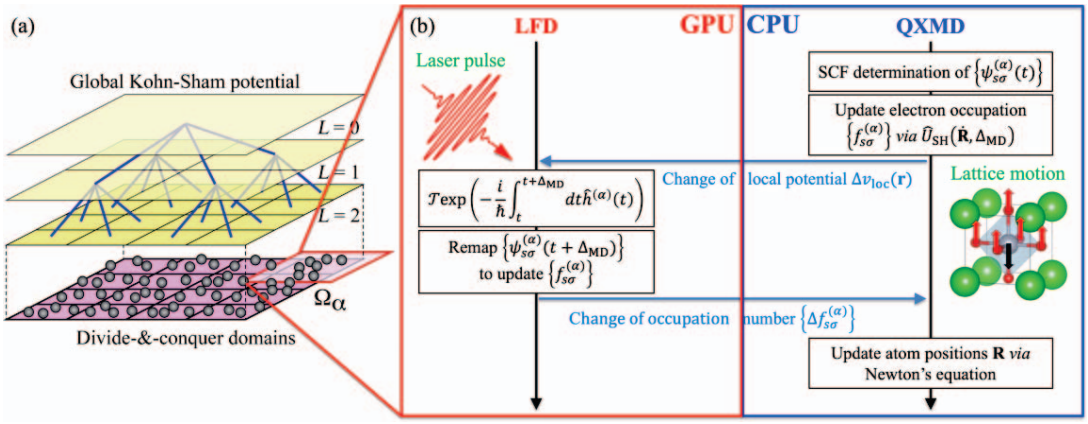

Fig. 1: (a) Divide-and-conquer domains embedded in a global potential. (b) DC-MESH method consists of (i) LFD to describe light-electron interaction on GPU and (ii) QXMD to describe electron-atom coupling on CPU, with minimal CPU-GPU data transfer *via* shadow dynamics.

DC-MESH is based on an extension of DC called divide-conquer-recombine (DCR) [25]. In DCR, the three-dimensional space Ω is subdivided into spatially localized domains Ω_α as $\Omega = \cup_\alpha \Omega_\alpha$ (Fig. 1a) [5, 25]. The initial DC phase constructs globally informed local solutions, which are used in the subsequent recombine phase as a compact basis to synthesize global properties. The recombine phase typically performs range-limited *n*-tuple computations to account for higher-order correlations that are not captured by the tree topology used in the DC phase. Specifically in TDDFT, the mean electrostatic field (or Hartree potential) is computed globally using the scalable $O(N)$ multigrid method, whereas higher-order correlations represented by the exchange-correlation (XC) kernel are treated locally within each DC domain since they are known to be short-ranged [26]. Our DC-DFT algorithm employs a globally scalable and locally fast (GSLF) electronic solver that combines an $O(N)$ tree-based multigrid method to represent global potential and fast Fourier transform (FFT) to represent local KS wave functions [25]. We have also designed a lean divide-and-conquer (LDC) DFT algorithm, which significantly reduces the prefactor of the $O(N)$ computational cost by applying a density-adaptive boundary condition at the peripheries of the DC domains. Hybrid space-band decomposition is used to implement the LDC-DFT algorithm on parallel computers. In DC-MESH, the QXMD subprogram on CPU solves global-local SCF iterations in the DC phase using multiple computing nodes based on message passing interface (MPI); see Fig. 1b. In the recombines phase, our globally-sparse and locally-dense (GSLD) electronic solvers allow the compute-intensive, data-parallel LFD subproblem to reside on GPU with minimal CPU-GPU data transfer thanks to shadow dynamics (Fig. 1b). LFD computation on GPU is based on OpenMP target constructs, which is accelerated by several computational innovations: (i) data and loop reordering and blocking for better memory access patterns; (ii) hierarchical GPU offloading using OpenMP teams-distribute and parallel constructs, respectively, for coarse and fine computations; (iii) algebraic 'BLASification' of the nonlocal computational bottleneck; and (iv) simplified GPU resident computation facilitated by custom C++ class initializer and destructor based on OpenMP target data constructs. GPU computing *via* OpenMP maximizes ease of programming and portability. With

the resulting high performance demonstrated in our result section, this paper serves as a promising GPU-offloading pathway for many scientific and engineering codes.

***Multiple time-scale splitting*:** Since our DCR algorithm for the QXMD subprogram has been reported previously [5, 25], we here focus on the LFD subprogram. In the α-th domain, we numerically integrate Maxwell-TDDFT equations:

$$i\hbar\frac{\partial}{\partial t}\left|\psi_{s\sigma}^{(\alpha)}(t)\right\rangle = \hat{h}^{(\alpha)}(t)\left|\psi_{s\sigma}^{(\alpha)}(t)\right\rangle, \quad (1)$$

where $\left|\psi_{s\sigma}^{(\alpha)}(t)\right\rangle$ is the s-th complex-valued Kohn-Sham (KS) wave function with spin σ within Ω_α at time t, and the Hamiltonian operator is defined as [9, 10]

$$\hat{h}^{(\alpha)}(t,\mathbf{R}(t)) = \frac{1}{2m}\left(\frac{\hbar}{i}\nabla + \frac{e}{c}\mathbf{A}_{\mathbf{X}(\alpha)}(t)\right)^2 + \hat{v}_{\text{ion}}(\mathbf{r},\mathbf{R}(t)) - e\phi_\alpha(\mathbf{r},t) + \hat{v}_{\text{xc}}[\mathbf{r},t;\rho_\alpha(\mathbf{r},t)]. \quad (2)$$

Here, m and e are the electron mass and charge, $\hbar$ is the Planck constant, c is the light speed, $\mathbf{A}_{\mathbf{X}(\alpha)}$ is the electromagnetic vector potential at the spatial position of the α-th domain $\mathbf{X}(\alpha)$, $\hat{v}_{\text{ion}}$ is the ionic pseudopotential, $\mathbf{R}(t)$ collectively denotes the positions of all atoms, and ϕ_α is the scalar potential. In Eq. (2), $\hat{v}_{\text{xc}}$ is the exchange-correlation (xc) potential, which is a functional of the electron number density, $\rho_\alpha(\mathbf{r},t) = \sum_{s\sigma} f_{s\sigma}^{(\alpha)}|\psi_{s\sigma}(\mathbf{r},t)|^2$, with $f_{s\sigma}^{(\alpha)}$ ($\in [0,1]$) being the occupation number. We solve Maxwell's equation for $\mathbf{A}_{\mathbf{X}(\alpha)}$ and an auxiliary partial differential equation [27, 28] for ϕ_α.

Solution of Eq. (1) should account for disparate time scales: $\Delta_{\text{QD}}\sim10^{-18}$ second for electrons, $\left|\psi_{s\sigma}^{(\alpha)}(t)\right\rangle$, and $\Delta_{\text{MD}}\sim10^{-15}$ second for atoms, $\mathbf{R}(t)$. By expanding the ionic pseudopotential in terms of slow atomic velocities, $\dot{\mathbf{R}} = d\boldsymbol{R}/dt$, and retaining up to the linear term, we can time-propagate electrons for one molecular-dynamics step Δ_{MD} as

$$\left|\psi_{s\sigma}^{(\alpha)}(t+\Delta_{\text{MD}})\right\rangle = \mathcal{T}\exp\left(-\frac{i}{\hbar}\int_t^{t+\Delta_{\text{MD}}} dt\hat{h}^{(\alpha)}(t)\right)\widehat{U}_{\text{SH}}\left(\dot{\mathbf{R}},\Delta_{\text{MD}}\right)\left|\psi_{s\sigma}^{(\alpha)}(t)\right\rangle, \quad (3)$$

where $\mathcal{T}$ is the time-ordering operator and $\widehat{U}_{\text{SH}}\left(\dot{\mathbf{R}},\Delta_{\text{MD}}\right)$ is the standard surface-hopping (SH) procedure to update the electron occupation $f_{s\sigma}^{(\alpha)}$ perturbatively according to nonadiabatic coupling (NAC) arising from slow atomic motions [21]. Subsequently, we operate $\mathcal{T}\exp\left(-\frac{i}{\hbar}\int_t^{+\Delta_{\text{MD}}} dt\hat{h}^{(\alpha)}(t)\right)$ in Eq. (3) using Suzuki-Trotter expansion and space-splitting method [28]:

$$\mathcal{T}\exp\left(-\frac{i}{\hbar}\int_t^{t+\Delta_{\text{MD}}} dt\hat{h}^{(\alpha)}(t)\right) \cong \prod_{n=0}^{N_{\text{QD}}-1}\exp\left(-\frac{i\Delta_{\text{QD}}}{\hbar}\hat{h}^{(\alpha)}\left(t+\left(n+\frac{1}{2}\right)\Delta_{\text{QD}}\right)\right), \quad (4)$$

where $N_{\text{QD}} = \Delta_{\text{MD}}/\Delta_{\text{QD}}$ is the number of QD time steps per MD step. To ensure stable time propagation during each QD time step Δ_{QD}, we employ a self-consistent, time-reversible unitary approach that handles nonlinearity, *i.e.*, the time-propagation operator itself depends on the wave functions being propagated [29, 30].

***Shadow dynamics*:** The purpose of the electronic time-propagator within LFD is to determine the change of electron occupation number $f_{s\sigma}$ due to light-matter interaction during one MD time step, Δ_{MD}, so that it modifies the excited-state energy landscape [12, 22, 25] to inform atomic motions in the SH approach. This can be achieved in a computationally efficient manner similar to the shadow dynamics [18]. Namely, we refactor the Hamiltonian $\hat{h}^{(\alpha)}$ in Eq. (2) as follows:

$$\hat{h}^{(\alpha)} = \frac{1}{2}\left(\frac{\hbar}{i}\nabla + \frac{e}{c}\mathbf{A}_{\mathbf{X}(\alpha)}(t)\right)^2 + v_{\text{loc}}^{(\alpha)}(\mathbf{r},t) + \hat{v}_{\text{nl}}^{(\alpha)} = \hat{h}_{\text{loc}}^{(\alpha)}(t) + \hat{v}_{\text{nl}}^{(\alpha)}, \quad (5)$$

where the local potential $v_{\text{loc}}^{(\alpha)}$ represents the local pseudopotential, as well as the Hartree and local exchange-correlation potentials, which apply spatial point-by-point, while the nonlocal operator $\hat{v}_{\text{nl}}^{(\alpha)}$ here collectively denotes the nonlocal ionic pseudopotential and nonlocal exchange-correlation potential, which has much more complex computational characteristics [31]. According to this refactoring, the electronic time-propagator is approximated as [32]

$$\exp\left(-\frac{i\Delta_{\text{QD}}}{\hbar}\hat{h}^{(\alpha)}(t)\right) \cong \frac{1-\frac{i\Delta_{\text{QD}}}{2\hbar}\hat{v}_{\text{nl}}^{(\alpha)}}{\left\|1-\frac{i\Delta_{\text{QD}}}{2\hbar}\hat{v}_{\text{nl}}^{(\alpha)}\right\|}\exp\left(-\frac{i\Delta_{\text{QD}}}{\hbar}\hat{h}_{\text{loc}}^{(\alpha)}(t)\right)\frac{1-\frac{i\Delta_{\text{QD}}}{2\hbar}\hat{v}_{\text{nl}}^{(\alpha)}}{\left\|1-\frac{i\Delta_{\text{QD}}}{2\hbar}\hat{v}_{\text{nl}}^{(\alpha)}\right\|}. \quad (6)$$

Here, the local propagator, $\exp\left(-i\Delta_{\text{QD}}\hat{h}_{\text{loc}}^{(\alpha)}(t)/\hbar\right)$, can be cast into data-local stencil operations using the $O(N)$ space-splitting method [28]. To efficiently compute the nonlocal part in Eq. (6), we project $\hat{v}_{\text{nl}}^{(\alpha)}$ onto the vector space spanned by $\left\{\left|\psi_{s\sigma}^{(\alpha)}\right\rangle = \left|\psi_{s\sigma}^{(\alpha)}(t=0)\right\rangle\right\}$ [33]:

$$\left(1-\frac{i\Delta_{\text{QD}}}{2\hbar}\hat{v}_{\text{nl}}^{(\alpha)}\right)\left|\psi_{s\sigma}^{(\alpha)}(t)\right\rangle \cong \left|\psi_{s\sigma}^{(\alpha)}(t)\right\rangle - i\frac{\Delta_{\text{sci}}\Delta_{\text{QD}}}{2\hbar}\sum_{u\geq\text{LUMO},\sigma}\left|\psi_{u\sigma}^{(\alpha)}\right\rangle\left\langle\psi_{u\sigma}^{(\alpha)}\middle|\psi_{s\sigma}^{(\alpha)}(t)\right\rangle, \quad (7)$$

where $\langle\ |\ \rangle$ denotes the inner product of two wave functions and the scissor shift is defined as

$$\Delta_{\text{sci}}^{(\alpha)} = \left(\epsilon_{\text{LUMO,nl}}^{(\alpha)} - \epsilon_{\text{HOMO,nl}}^{(\alpha)}\right) - \left(\epsilon_{\text{LUMO,loc}}^{(\alpha)} - \epsilon_{\text{HOMO,loc}}^{(\alpha)}\right). \quad (8)$$

Here, the lowest unoccupied molecular orbital (LUMO) and highest occupied molecular orbital (HOMO) KS energies, $\epsilon_{\text{LUMO}}^{(\alpha)}$ and $\epsilon_{\text{HOMO}}^{(\alpha)}$, are computed with the expensive nonlocal computation (nl) and inexpensive local computation (loc) only once at each MD step, which are reused for $N_{\text{QD}} = 10^2\sim10^3$ steps to amortize the computational cost.

III. Optimized Implementation of Electronic Time-Propagation and Nonlocal Correction: Vectorization, Hierarchial Parallelism, and BLAS Operation

In this section, we describe optimizations applied to improve performance of the LFD subprogram. We use the Open Multi-Processing (OpenMP) parallel programming model to enable minimally invasive offloading to GPUs. We also avoid unnecessary overheads by creating a common device data environment to reduce the overall amount spent in host-to-device data transfer in the OpenMP target region. We first focus on the local time-propagator, $\exp\left(-i\Delta_{\text{QD}}\hat{h}_{\text{loc}}^{(\alpha)}(t)/\hbar\right)$ in Eq. (6), which is a sequence of stencil operations [28]. A series of optimizations are applied before enabling offload, which

include loop-interchange, memory re-use, and tiling. We then describe computation transformation of nonlocal correction, $\hat{v}_{\mathrm{nl}}^{(\alpha)}$ in Eq. (7), to BLAS operations and persistent GPU kernel transformation.

The electronic time-propagation kernel (specifically the kinetic propagator kernel arising from the gradient operator in the Hamiltonian in Eq. (5)) is a stencil operation [28], with repeated applications of the time-stepping operator to discretized KS wave functions on spatial mesh points. Algorithm 1 shows the baseline algorithm. When carrying out time propagation along a certain Cartesian axis, the input consists of the wave function $\psi_{s\sigma}^{(\alpha)}(t)$, stencil direction $d \in \{x, y, z\}$, time step $p \in \{\Delta_{\mathrm{QD}}/2\,, \Delta_{\mathrm{QD}}\}$, as well as diagonal, upper-diagonal, and lower-diagonal coefficients, $\alpha_{dp}, \beta_{l,dp}, \beta_{u,dp}$, defined for each mesh point.

Here, a straightforward implementation stores data for the wave function $\psi_{s\sigma}^{(\alpha)}(t)$ in array *psi*, such that the first index specifies one of the *N* KS orbitals and subsequent indices specifies one of the *M* grid points in the *x*, *y* and *z* Cartesian directions. When traversing the wave function in line 3, we first iterate over the orbitals and then iterate over the grid points in lines 4, 5 and 6. The operations in lines 7 and 8 yield the value for the real and imaginary parts of the complex-valued wave function after time propagation. Once the wave function has been time-propagated for all mesh points in each orbital, we exit the nested loop and update the wave function in line with the values accumulated in line 11.

Algorithm 1: Baseline implementation of time propagation of electronic wave functions

```
1:  void kin_prop (psi, al, bl, bu, p, d, Norb, Nr, Nx, Ny, Nz) {
2:    complex<float> wrk[Nx+2][Ny+2][Nz+2], w;
3:    for (int n=0; n < Norb; n++)
4:      for (int i=1; i <= Nr[0]; i++)
5:        for (int j=1; j <= Nr[1]; j++)
6:          for (int k=1; k <= Nr[2]; k++) {
7:            w = al[d][p]*psi[n][i][j][k]
8:            ...
9:            wrk[i][j][k] = w;
10:         }
11:     #update psi[n][i][j][k] ← wrk[i][j][k]
12:   }
```

A. Loop Interchange and Memory re-use

Algorithm 1 is inefficient since the range of data swept across the wave function and coefficient arrays, which are multiplied in the stencil operation, is very large, whereas the update operation takes place at the orbital level. This means larger strides of data will be out of the cache when grabbing values to perform the update. In the current implementation scanning the *D*-dimensional mesh (D = 3) of *M* grid points across *N* orbitals creates a memory overhead in the order of $O(M^D)$ bytes in line 7. Keeping in mind that the wave function itself requires allocating $O(L^D)$ bytes (L is the number of grid points in one direction), where $L < M$, such implementation results in the memory demand growing at the rate of $O(M^D L^D)$ during the stencil operation.

To minimize the possibility of reaching the memory bandwidth, we eliminate storing a copy of the propagated wave in line 7 and instead proceed to immediately update the wave function with the partial values computed at that grid point. This update is achieved through a loop re-ordering such that the fastest-changing index corresponds to the orbital, resulting in the move of the update operation inside the loop as shown in Algorithm 2, line 6 and line 9. We also change the data layout of the wave function *psi* such that the wave function at each grid point stores the value for all orbitals, thereby making it a structure of arrays (SoA) over the original arrays of structures (AoS). A combination of loop interchange and an SoA data layout offers better memory access patterns in the available registers for both single-instruction multiple-data (SIMD) paradigm on CPU and single-instruction multiple-thread (SIMT) paradigm on GPU.

Algorithm 2: Loop re-ordering in time propagation of electronic wave functions

```
1:  void kin_prop (psi, al, bl, bu, p, d, Norb, Nr, Nx, Ny, Nz) {
2:    complex<float> float w;
3:    for (int j=1; j <= Nr[1]; j++)
4:      for (int k=1; k <= Nr[2]; k++)
5:        for (int i=1; i <= Nr[0]; i++)
6:          for (int n=0; n < Norb; n++) {
7:            w = al[d][p]*psi[i][j][k][n]
8:            ...
9:            # update psi[i][j][k][n] ← w
10:         }
11:   }
```

While the update operation changes the value of the wave function for that orbital, there are still computations in the stencil which rely on the value prior to the update step. For this reason, we store a small portion of the data structure before doing each update, to ensure computational correctness as shown in Algorithm 3, line 6. Updating grid points remains fully independent in the *y* and *z* directions but not in the *x* direction when working on an x-direction stencil for example.

Algorithm 3: Optimized stencil in time propagation of electronic wave functions

```
1:  void kin_prop (psi, al, bl, bu, p, d, Norb, Nr, Nx, Ny, Nz) {
2:    complex<float> w;
3:    for (int j=1; j <= Nr[1]; j++)
4:      for (int k=1; k <= Nr[2]; k++) {
5:        for (int n=0; n < Norb; n++) {
6:          psi_old[n] = psi[0][j][k][n];
7:        }
8:        for (int i=1; i <= Nr[0]; i++)
9:          for (int n=0; n < Norb; n++) {
10:           w = al*psi[i][j][k][n];
11:           w += bl[i]*psi_old[n];
12:           …
13:           # update psi_old ← psi[i][j][k][n]
14:           # update psi[i][j][k][n] ← w
15:         }
16:     }
17:   }
```

B. Blocking/Tiling

When the number of orbitals, *Norb,* is not small, the whole wave function, *psi_old,* array may not fit in cache and add traffic to the slower tier of memory. Blocking the loop of *Norb* reduces the size of *psi_old* array to only the desired block size instead of *Norb*, as shown in Algorithm 4. The added loop of blocks also allows distributing the computation to more GPU blocks when offloading is used.

Algorithm 4: Cache blocking optimization in time propagation of electronic wave functions

```
1:  void kin_prop (psi, al, bl, bu, p, d, Norb, Nr, Nx, Ny, Nz) {
2:    complex<float> w;
3:    for (int j=1; j <= Nr[1]; j++)
4:      for (int k=1; k <= Nr[2]; k++) {
5:        for (int ib=0; ib < (Norb+1)/block_size; ib++) {
6:          complex<float> psi_old[block_size];
7:          int begin = ib*block_size;
            int end = min((ib+1)*block_size, Norb);
8:          for (int n=begin; n < end; n++)
9:            psi_old[n-begin] = psi[0][j][k][n];
10:         for (int i=1; i <= Nr[0]; i++)
11:           for (int n=begin; n <end; n++) {
12:             w = al*psi[i][j][k][n];
13:             w += bl[i]*psi_old[n-begin];
14:             …
15:             # update psi_old ← psi[i][j][k][n]
16:             # update psi[i][j][k][n] ← w
17:           }
18:       }
19:     }
20:   }
```

C. Multiple Parallel Regions

To offload the computation to the accelerator devices on the blade we test, we use the OpenMP programming model. Through our loop re-ordering and SoA optimization, we expose the computation kernel to a high level of parallelism. The propagation of grid points of the *y*-*z* plane can be concurrently computed for an *x*-direction stencil. This is because the propagation of the electronic wave function along the *x*-direction requires the i^{th} index of the wave function $\psi_{i,j,k}$ to inter-mix with every (*j*,*k*). Hence, the first level of parallelism is achieved as the evolution requires only knowledge of the wave function at the current time step and the previous step within the same plane. A second level of parallelism comes into effect from the ability to propagate the wave function independently of the orbital. This hierarchical parallelism applies to both SIMD and SIMT paradigms. The parallelization over planes and orbitals are collapsed into a larger loop. This grid geometry makes efficient targets of Cooperative Thread Arrays (CTA) that are available as well as the limited Streaming Multiprocessor (SM) register file size. Algorithm 5 shows this parallelism, where the data structures are now aligned such that all orbitals for a mesh point are aligned in a single stride. Also, note here we flatten structures of *psi* and *psi_old* into one-dimensional arrays of complex numbers.

Algorithm 5: OpenMP stencil in time propagation of electronic wave functions

```
1:  void kin_prop (psi, al, bl, bu, p, d, Norb, Nr, Nx, Ny, Nz) {
2:    complex<float> w;
3:    #pragma omp target teams distribute collapse(3)
4:    for (int j=1; j <= Nr[1]; j++)
5:      for (int k=1; k <= Nr[2]; k++) {
6:        for (int ib=0; ib < (Norb+1)/block_size; ib++) {
7:          complex<float> psi_old[block_size];
8:          int begin = ib*block_size;
            int end = min((ib+1)*block_size, norb);
11:         #pragma omp parallel for simd nowait
9:          for (int n=begin; n < end; n++)
10:           psi_old[n-begin] = psi[0][j][k][n];
8:          for (int i=1; i <= Nr[0]; i++)
9:            #pragma omp parallel for simd nowait
10:           for (int n=begin; n < end; n++) {
11:             w = al*psi[i][j][k][n]
                w += bl[i]*psi_old[n-begin]
12:             …
13:             # update psi_old ← psi [i][j][k][n]
14:             # update psi [i][j][k][n] ← w
15:           }
16:       }
        }
17:   }
```

D. BLASification of Nonlocal Correction

The compute-intensive nonlocal correction in Eq. (7) for time propagation of electronic wave functions can be cast into matrix operations. To do so, let us define a $Ngrid \times Norb$ wave-function matrix $\Psi(t)$, where $Ngrid$ and $Norb$ are the number of grid points to represent each wave function and that of KS wave functions, respectively. Equation (7) then reads

$$\Psi(t) -= c\Psi(0)\Psi^{\dagger}(0)\Psi(t), \tag{9}$$

where c is a complex number and $\Psi^{\dagger}$ denotes a Hermitian transpose matrix. We implement Eq. (9) using BLAS level 3 calls. In addition to time propagation of electronic wave functions in function *nlp_prop*(), BLASified nonlocal correction appears in two other functions in LFD: energy calculation in function *calc_energy*() and remapping the final wave functions to occupation numbers in function *remap_occ*().

E. Persistent GPU kernel

The key computational advantage of the shadow dynamics is that the large wave-function arrays, $\Psi(t)$ and $\Psi(0)$, can be made GPU-resident, thereby eliminating massive CPU-GPU data transfer. Such persistent GPU data structures are facilitated by our custom C++ class constructor and destructor based on OpenMP target data constructs; see Algorithm 6. The custom allocator named *OMPallocator* is used for container classes like *std::vector*, which are intended to be GPU-resident. Upon initialization, the allocator calls *#pragma omp target enter data map(alloc)*, while upon destruction, it calls *#pragma omp target exit data map(delete)*. This significantly eases the programmability of persistent GPU dataset, while keeping the use-side code neat. In addition, the *HostAllocator* may be replaced with a customized allocator using pinned host memory to further improve host-device transfer rate.

Algorithm 6: OpenMP allocator

```
1:  template<typename T, class HostAllocator =
    std::allocator<T>>
2:  struct OMPallocator : public HostAllocator {
3:    OMPallocator() = default;
4:    value_type* allocate(std::size_t n) {
5:      value_type* pt = HostAllocator::allocate(n);
6:      #pragma omp target enter data map(alloc:pt[0:n])
7:      return pt;
8:    }
      void deallocate(value_type* pt, std::size_t n) {
11:     #pragma omp target exit data map(delete:pt[0:n])
9:      HostAllocator::deallocate(pt, n);
10:   }
8:  }
```

IV. Performance Evaluation

We measure performance of DC-MESH on the Polaris supercomputer at Argonne Leadership Computing Facility

(ALCF). It is a Hewlett Packard Enterprise (HPE) Apollo 6500 Gen 10+ based system consisting of two computing nodes per chassis, seven chassis per rack, and 40 racks that amount to a total of 560 nodes. Each Polaris node has one 2.8 GHz AMD EPYC Milan 7543P 32-core CPU with 512 GB of DDR4 RAM, four Nvidia A100 GPUs, two 1.6 TB of SSDs in RAID0, and two Slingshot network endpoints. Polaris uses the Nvidia A100 HGX platform to connect all 4 GPUs via NVLink, with a GPU interconnect bandwidth of 600 GB/s. The GPU's PCIe bandwidth is 64 GB/s. HBM2 memory for GPUs is available on both HGX and PCIe and is 60 GB and 40 GB, respectively. Designed by Cray, the Slingshot interconnect is based on high radix 64-port switches arranged in dragonfly topology and offers adaptive routing, congestion control, and bandwidth guarantees by assigning traffic classes to applications. Polaris uses Slingshot 11 with a node interconnect bandwidth of 200 GB/s. Polaris' peak performance is 44 Petaflop/s, with node-level performance at 78 Teraflop/s, for double precision.

The DC-MESH code consists of the QXMD subprogram written in Fortran with MPI and the LFD subprogram written in C++ with OpenMP. For performance evaluation on Polaris, DC-MESH is built using Gfortran and clang 15 compilers.

A. Weak and Strong Scalability

We first perform a weak-scaling benchmark of DC-MESH on Polaris, in which the number of atoms per MPI rank, N/P is kept constant, *i.e.*, $PbTiO_3$ material consisting of 40 atoms. For each MPI rank, 288 KS wave functions are represented using the plane-wave basis in QXMD, while each complex-valued KS wave function in LFD is represented on 70×70×72 finite-difference mesh points. Weak scaling test is carried out up to 256 computing nodes with 4 MPI ranks per node, where each rank is accelerated by one GPU. The largest system on 256 nodes thus consists of 10,240 atoms.

We measure the wall-clock time per MD simulation step with scaled workloads — 40P-atom $PbTiO_3$ material on P MPI ranks on Polaris. The execution time includes 3 self-consistent field (SCF) iterations to determine the KS wave functions and the global potential in QXMD, with 3 conjugate-gradient (CG) iterations per SCF cycle to refine each wave function. We run 1,000 QD steps in LFD per MD step. By increasing the number of atoms linearly with the number of MPI ranks, the wall-clock time remains nearly constant, indicating excellent weak scalability. To quantify the weak-scaling parallel efficiency, we first define the speed of the DC-MESH program as a product of the total number of atoms and the number of MD simulation steps executed per second. The isogranular speedup is given by the ratio between the speed on P MPI ranks and that on 4 MPI ranks (*i.e.*, one computing node) as a reference system. The weak-scaling parallel efficiency is the isogranular speedup divided by $P/4$. Figure 2 shows the weak-scaling parallel efficiency as a function of the number of MPI ranks. With the granularity of 40 atoms per MPI rank, the parallel efficiency is 0.9673 on P = 256 for a 10,240-atom $PbTiO_3$ material. This result demonstrates the very high scalability of the DC-MESH program, mainly due to the globally-sparse and locally-dense electronic solvers within the divide-conquer-recombine algorithmic framework.

Next, we perform strong-scaling tests for two problem sizes: 5,120- and 10,240-atom $PbTiO_3$ materials. In this test, the number of MPI ranks ranges from P = 64 to 256 for the 5,120 atoms and P = 128 to 512 for the 10,240 atoms, while keeping the total problem size constant in each case. The strong-scaling speedup is defined as the wall-clock time on the smallest number, P_{min}, of MPI ranks divided that on the largest number, P_{max}, of MPI ranks for each problem size. The strong-scaling parallel efficiency is the strong-scaling speedup divided by P_{max}/P_{min}. Figure 3 shows the strong-scaling parallel efficiency as a function of P. The strong-scaling parallel efficiency is 0.8083 with 512 MPI ranks for 10,240 atoms, while it is 0.6634 with 256 MPI ranks for 5,120 atoms. It is more difficult to achieve high strong-scaling parallel efficiency compared with weak-scaling parallel efficiency. This is due to the increased communication/computation ratio as the workload per rank reduces. This is partly understood by analyzing the parallel efficiency η as a function of the number of MPI ranks P and that of atoms N. For the weak-scaling parallel efficiency with constant granularity ($n = N/P$), $\eta = 1/[1 + \alpha n^{-\frac{1}{3}} + \beta n^{-1}\log P]$, exhibiting a very weak logarithmic dependence on P [34]. For the strong-scaling parallel efficiency with constant

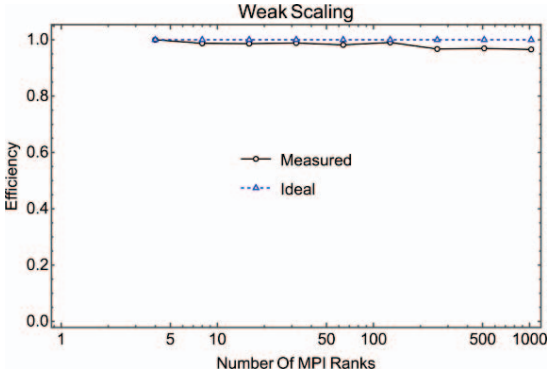

Fig. 2. Weak-scaling parallel efficiency of the DC-MESH program, with scaled workloads — 40P-atom $PbTiO_3$ material with P MPI ranks (P = 4, ..., 1,024) on Polaris. Black circles are measured data, whereas blue triangles show ideal speedup.

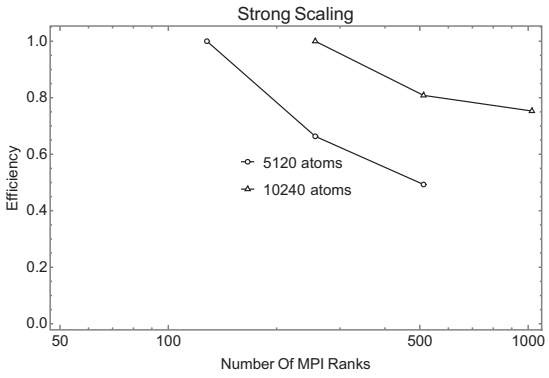

Fig. 3. Strong-scaling parallel efficiency of the DC-MESH program as a function of the number of MPI ranks on Polaris for two problem sizes: 5,120- and 10,240-atom $PbTiO_3$ materials.

N, in contrast, $\eta = 1/[1 + \alpha(P/N)^{1/3} + \beta N^{-1} P\log P]$, which exhibits much stronger dependency on P, *i.e.*, $P^{1/3}$ and $P\log P$ in the denominator.

B. GPU Performance

To test single-node GPU performance of the DC-MESH program, we spawn 4 MPI ranks on one computing node with a 40-atom $PbTiO_3$ material per MPI rank. Figure 4 compares the throughput on CPU+GPU and that on CPU only. Here, the throughput is defined as the number of ranks that complete execution per unit time for a fixed problem: $P/t_{\text{completion}}$. By offloading key computations to GPU, we obtain a 19-fold speedup over CPU. This signifies decent utilization of GPU resources on Polaris by the DC-MESH code.

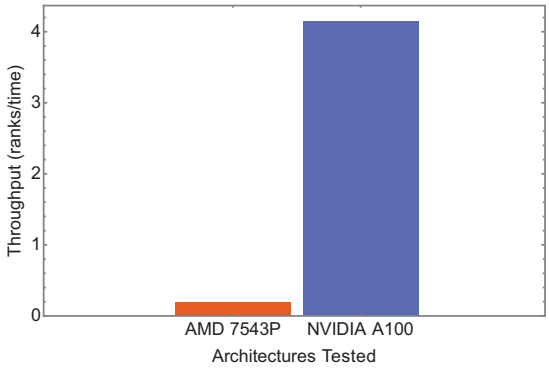

Fig. 4. Throughput of DC-MESH on a single computing node of Polaris. We compare CPU-only throughput on AMD 7543P and that of CPU plus Nvidia A100 GPU.

C. Performance Improvement

We next examine GPU performance of the stencil-based time propagation of KS wave function in the LFD subprogram as shown in Algorithms 1-5. Table I compares runtime of the *kin_prop*() function in the LFD subprogram on Polaris system using the Clang compiler shipped as a part of LLVM 16 in comparison with the corresponding runtime on CPU over each successive optimization mentioned in sections III A-C. The timing is for 1,000 QD steps involving 64 KS wave functions each on $70 \times 70 \times 72$ finite-difference mesh points. For simplicity, a single GPU timing is compared with a single CPU-core timing. We first measure incremental performance improvement due to Algorithms 3 and 4 over that of the baseline Algorithm 1. The results in Table I shows 3.67- and 9.22-fold speedups over the baseline for Algorithms 3 and 4, respectively. Further with GPU offloading using Algorithm 5, we overall achieve 338-fold speedup over the baseline. Next, as an ablation study, we disable the asynchronous offloading feature (*i.e.*, the *nowait* keyword in Algorithm 5) in the GPU-offloaded code to make it synchronous (fifth entry in Table I). The results show 298-fold speedup due to GPU offloading compared to the CPU code for the synchronous offloading. The asynchronous offloading code thus achieves 10.35% speedup compared to the synchronous offloading code. These results demonstrate high GPU utilization as a result of the series of data-structure and code restructuring outlined through Algorithms 1-5.

Table I. Runtime of the *kin_prop*() function in the LFD subprogram.

Implementation	Target	Runtime (s)	Speedup
Algorithm 1	CPU	8.655	1
Algorithm 3	CPU	2.356	3.67
Algorithm 4	CPU	0.939	9.22
Algorithm 5	GPU	0.026	338
Algorithm 5 (disable *nowait*)	GPU	0.029	298

Another key performance optimization is the transformation of nonlocal correction to BLAS 3 operations described in section III-D. Table II compares runtime of various versions of the code for both single precision (SP) and double precision (DP) floating-point formats of KS wave functions. The timing is for 1,000 QD steps involving 64 KS wave functions each on $70 \times 70 \times 72$ finite-difference mesh points. Here, we enumerate the types of builds of the LFD subprogram, starting with a purely CPU build without invoking any linear algebra libraries, followed by that using the AMD Optimizing CPU Libraries (AOCL)—BLAS library. We then offload the self-consistent equation kernels to GPU, utilizing optimization in section III-E. In addition to AOCL-BLAS library, we subsequently use the native cuBLAS library on A100 before we finally harness faster data transfers between host and device

Table II. Runtime comparison of several versions of the DC-MESH program for SP and DP floating-point formats. Measurement was made using a single OpenMP thread for simplicity.

	Electron propagation (sec)		Nonlocal correction (sec)		Total runtime (sec)	
	SP	DP	SP	DP	SP	DP
CPU OpenMP Parallel	444.44	470.73	442.84	455.75	1082	1167
CPU OpenMP Parallel + BLAS	19.72	30.92	10.71	21.54	38.83	65.93
GPU OpenMP Offload + BLAS	7.03	11.45	6.75	11.12	17.14	29.23
GPU OpenMP Offload + cuBLAS	0.61	0.94	0.46	0.761	1.33	2.11
GPU OpenMP Offload + cuBLAS (Pinned Memory w/ Cuda Streams)	0.512	0.68	0.35	0.51	1.06	1.48

with pinned memory. We track the runtime of some of the most time-consuming operations: (i) time propagation of electronic wave functions (or electron propagation) including potential propagation, kinetic propagation, and nonlinear propagation (*cf.* Eq. (6)); (ii) additional nonlocal correction operations (*cf.* section III-D); as well as (iii) the total time spent in the LFD subroutine. All runs are carried out with a single OpenMP thread. Table II shows a 35% reduction in electron propagation and a 42% reduction in nonlocal correction kernel completion times using SP compared to DP.

To quantify performance gains from vectorization and offloading the code to GPU, Fig. 5 shows DP runtime of compute-intensive kernels: electron time-propagation (Eq. (6)), nonlocal electron time-propagation (Eq. (7)), and energy calculation kernels for the benchmark test by building with available options. Here, we start with the purely CPU implementation with OpenMP and AOCL-BLAS and show subsequent reduction of runtime with GPU offload kernels, cuBLAS, and pinned memory. When comparing the purely CPU implementation with AOCL-BLAS build and the GPU kernel offload build with cuBLAS and pinned memory, we see 45-fold speedup in electron propagation, 42-fold speedup in nonlocal propagation and nearly 46-fold speedup in energy calculation kernels in the latter.

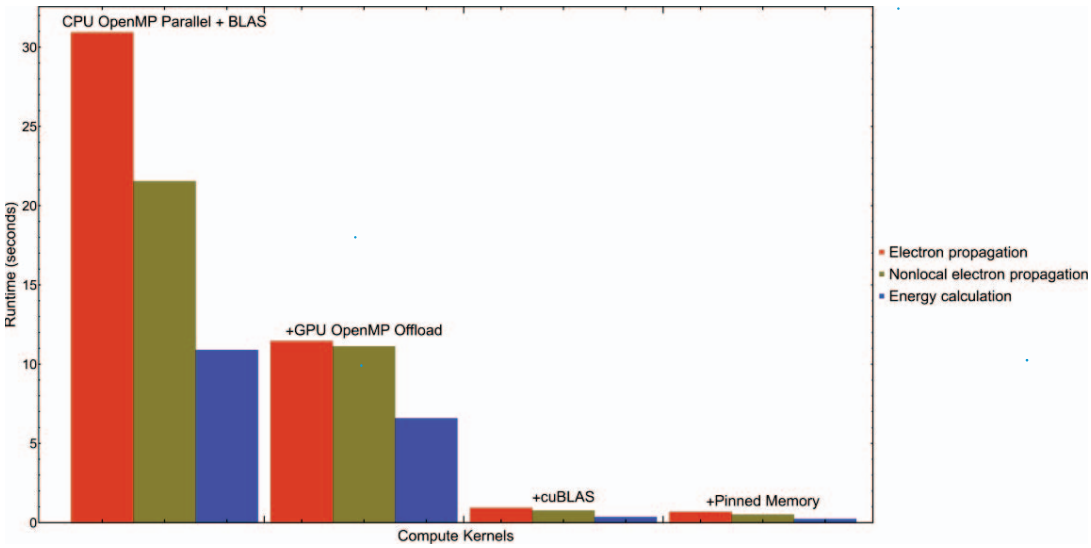

Fig. 5. Runtime of compute-intensive kernels when building with different parallel computing interfaces. Measurement was made using a single OpenMP thread for simplicity.

Figure 6 shows the speedup of the total DC-MESH code due to a sequence of code versions as shown in Fig. 5. The BLASification of the nonlocal computations are highly effective on both CPU and GPU. Accordingly, we first achieve 25.2-fold speedup with BLAS on CPU compared to the non-BLAS baseline on CPU. The BLASified code is then offloaded to GPU, achieving 18.6-fold speedup over the BLASified CPU code. By the memory-pinning optimization, we achieve additional 37.6% speedup. Overall, we achieve 644-fold cumulative speedup.

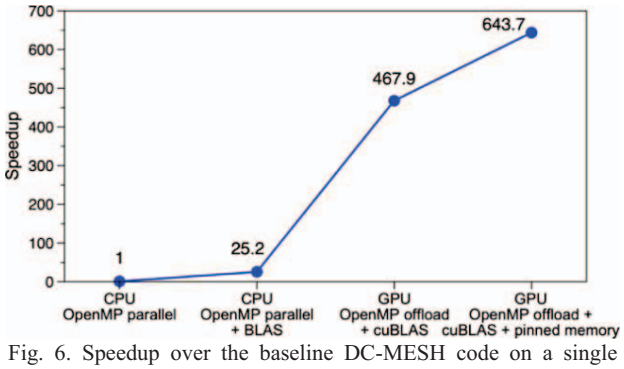

Fig. 6. Speedup over the baseline DC-MESH code on a single Polaris node resulting from a series of code optimizations. Measurement was made using a single OpenMP thread for simplicity.

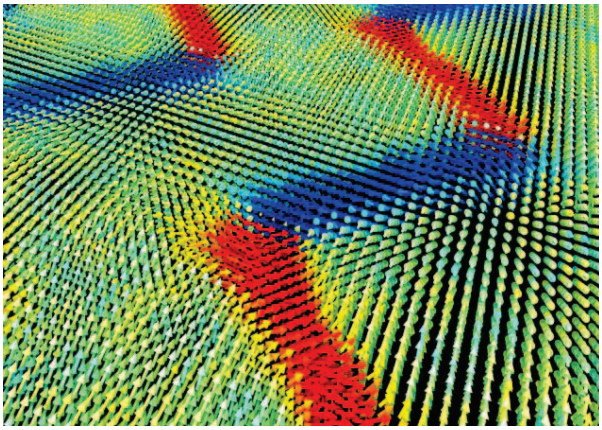

Fig. 7. Flux closure structure during ferroelectric switching in $PbTiO_3$.

V. Application

Performance-optimized DC-MESH code has enabled the study of light-induced topological switching for future ultrafast and ultralow-power ferroelectric topotronics applications. We have adopted a multiscale simulation approach [12], where we first prepare a complex polar topology such as the flux closure domain illustrated in Fig. 7, which has been investigated for next-generation transducer and sensor applications. Our multiscale approach utilizes molecular dynamics (MD) simulations with a neural-network force field trained with ground-state quantum MD simulations [35]. This allows for quickly generating ground-state polar topologies that is then investigated for their electronic and structural responses to femtosecond laser fields with DC-MESH. It is currently an open question how to control attosecond electronic excitation dynamics initiated by laser pulses to generate longer-time structural changes. Using our DC-MESH code, we are currently exploring those dynamics to understand laser-induced topological changes, such as fs laser induced ultrafast switching of the flux closure domain in Fig 7. Such light-matter interaction can be directly compared to/inform state-of-the-art experiments performed using free-electron lasers such as the newly upgraded LCLS-II at Stanford [36]. Integrated computational and experimental studies will be essential for developing controllable topological switching for ultralow-power technologies arising from topological protection from thermal noise [37].

VI. Conclusion

To study light-matter interaction on emerging exaflop/s supercomputers in the new era of attosecond physics, we have developed a linear-scaling DC-MESH (divide-and-conquer Maxwell-Ehrenfest-surface hopping) simulation algorithm. Our globally-sparse and locally-dense electronic solvers, multiple time-scale splitting, and shadow dynamics have achieved high scalability, while allowing the most compute-intensive quantum dynamics kernel based on time-dependent density functional theory to reside on GPU with minimal CPU-GPU data transfer. GPU computation based on minimally invasive OpenMP target constructs is accelerated by: (i) data and loop reordering for better memory access patterns; (ii) hierarchical GPU offloading using teams-distribute and parallel constructs, respectively, for coarse and fine computations; (iii) algebraic 'BLASification' of the nonlocal computational bottleneck; and (iv) GPU-resident data structures facilitated by custom C++ class initializer and destructor based on OpenMP target data constructs. We have thereby achieved 644-fold speedup on Nvidia A100 GPU over AMD EPYC 7543 CPU on the Polaris computer at Argonne Leadership Computing Facility. In addition, the DC-MESH code exhibited a high weak-scaling parallel efficiency of 96.73% on 256 nodes (or 1,024 GPUs) of Polaris for 5,120-atom $PbTiO_3$ material. This enables the study of light-induced topological switching for future ultrafast and ultralow-power ferroelectric topotronics applications for sustainable future. Most recently, the DC-MESH code has been ported to the Aurora supercomputer at Argonne, which will be presented elsewhere.

Acknowledgment

This work was supported by Department of Energy (DOE), Office of Science, Basic Energy Sciences, award DE-SC0000267409. K.N. was supported by an NSF grant OAC-2118061. The scalable code development was supported by the Aurora ESP program. An award for computer time was provided by the U.S. DOE Innovative and Novel Computational Impact on Theory and Experiment (INCITE) Program. This research used resources from the Argonne Leadership Computing Facility, a U.S. DOE Office of Science user facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. DOE under Contract No. DE-AC02-06CH11357.

Publisher's Note

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan.

References

1 https://www.nobelprize.org/uploads/2023/10/advanced-physicsprize2023-2.pdf.

2 Gygi, F., Draeger, E.W., Schulz, M., de Supinski, B.R., Gunnels, J.A., Austel, V., Sexton, J.C., Franchetti, F., Kral, S., Ueberhuber, C.W., and Lorenz, J.: 'Large-scale electronic structure calculations of high-Z metals on the BlueGene/L platform', Proceedings of Supercomputing, SC06, IEEE/ACM, 2006, pp. 45–es

3 Eisenbach, M., Zhou, C.G., Nicholson, D.M., Brown, G., Larkin, J., and Schulthess, T.C.: 'A scalable method for ab initio computation of free energies in nanoscale systems', Proceedings of Supercomputing, SC09, ACM/IEEE, 2009, pp. 64

4 Hasegawa, Y., Iwata, J., Tsuji, M., Takahashi, D., Oshiyama, A., Minami, K., Boku, T., Shoji, F., Uno, A., Kurokawa, M., Inoue, H., Miyoshi, I., and Yokokawa, M.: 'First-principles calculations of electron states of a silicon nanowire with 100,000 atoms on the K computer', Proceedings of Supercomputing, SC11, ACM/IEEE, 2011, pp. 1

5 Nomura, K., Kalia, R.K., Nakano, A., Vashishta, P., Shimamura, K., Shimojo, F., Kunaseth, M., Messina, P.C., and Romero, N.A.: 'Metascalable quantum molecular dynamics simulations of hydrogen-on-demand', Proceedings of Supercomputing, SC14, IEEE/ACM, 2014, pp. 661-673

6 Lass, M., Schade, R., Kuhne, T.D., and Plessl, C.: 'A submatrix-based method for approximate matrix function evaluation in the quantum chemistry code CP2K'. Proc. Proceedings of Supercomputing, SC20, Atlanta, Georgia, Nov IEEE/ACM, 2020

7 Das, S., Kanungo, B., Subramanian, V., Panigrahi, G., Motamarri, P., Rogers, D., Zimmerman, P.M., and Gavini, V.: 'Large-scale materials modeling at quantum accuracy: Ab initio simulations of

quasicrystals and interacting extended defects in metallic alloys', Proceedings of Supercomputing, SC23, ACM/IEEE, 2023, pp. 1
8 Jia, W., Wang, L.-W., and Lin, L.: 'Parallel transport time-dependent density functional theory calculations with hybrid functional on Summit'. Proceedings of Supercomputing, SC19, Denver, Colorado, Nov ACM/IEEE, 2019
9 Yabana, K., Sugiyama, T., Shinohara, Y., Otobe, T., and Bertsch, G.F.: 'Time-dependent density functional theory for strong electromagnetic fields in crystalline solids', Phys Rev B, 2012, 85, (4), pp. 045134
10 Jestadt, R., Ruggenthaler, M., Oliveira, M.J.T., Rubio, A., and Appel, H.: 'Light-matter interactions within the Ehrenfest–Maxwell–Pauli–Kohn–Sham framework: fundamentals, implementation, and nano-optical applications', Adv Phys, 2019, 68, (4), pp. 225-333
11 Basov, D.N., Averitt, R.D., and Hsieh, D.: 'Towards properties on demand in quantum materials', Nat Mater, 2017, 16, (11), pp. 1077-1088
12 Linker, T., Nomura, K., Aditya, A., Fukshima, S., Kalia, R.K., Krishnamoorthy, A., Nakano, A., Rajak, P., Shimmura, K., Shimojo, F., and Vashishta, P.: 'Exploring far-from-equilibrium ultrafast polarization control in ferroelectric oxides with excited-state neural network quantum molecular dynamics', Sci Adv, 2022, 8, (12), pp. eabk2625
13 Bowler, D.R., and Miyazaki, T.: 'O(N) methods in electronic structure calculations', Rep Prog Phys, 2012, 75, (3), pp. 036503
14 https://www.nobelprize.org/prizes/chemistry/1998/summary/.
15 Kohn, W.: 'Density functional and density matrix method scaling linearly with the number of atoms', Phys Rev Lett, 1996, 76, (17), pp. 3168-3171
16 Yang, W.T.: 'Direct calculation of electron-density in density-functional theory', Phys Rev Lett, 1991, 66, (11), pp. 1438-1441
17 Shimojo, F., Kalia, R.K., Nakano, A., and Vashishta, P.: 'Embedded divide-and-conquer algorithm on hierarchical real-space grids: parallel molecular dynamics simulation based on linear-scaling density functional theory', Comput Phys Commun, 2005, 167, (3), pp. 151-164
18 Niklasson, A.M.N.: 'Extended Lagrangian Born–Oppenheimer molecular dynamics: from density functional theory to charge relaxation models', Euro Phys J B, 2021, 94, (8), pp. 164
19 Lee, C.W., and Schleife, A.: 'Hot-electron-mediated ion diffusion in semiconductors for ion-beam nanostructuring', Nano Lett, 2019, 19, (6), pp. 3939-3947
20 Craig, C.F., Duncan, W.R., and Prezhdo, O.V.: 'Trajectory surface hopping in the time-dependent Kohn-Sham approach for electron-nuclear dynamics', Phy Rev Lett, 2005, 95, (16), pp. 163001
21 Tully, J.C.: 'Perspective: nonadiabatic dynamics theory', J Chem Phys, 2012, 137, (22), pp. 22A301
22 Shimojo, F., Ohmura, S., Mou, W., Kalia, R.K., Nakano, A., and Vashishta, P.: 'Large nonadiabatic quantum molecular dynamics simulations on parallel computers', Comput Phys Commun, 2013, 184, (1), pp. 1-8
23 Tancogne-Dejean, N., Oliveira, M.J.T., Andrade, X., Appel, H., Borca, C.H., Le Breton, G., Buchholz, F., Castro, A., Corni, S., Correa, A.A., De Giovannini, U., Delgado, A., Eich, F.G., Flick, J., Gil, G., Gomez, A., Helbig, N., Hübener, H., Jestädt, R., Jornet-Somoza, J., Larsen, A.H., Lebedeva, I.V., Lüders, M., Marques, M.A.L., Ohlmann, S.T., Pipolo, S., Rampp, M., Rozzi, C.A., Strubbe, D.A., Sato, S.A., Schäfer, C., Theophilou, I., Welden, A., and Rubio, A.: 'Octopus, a computational framework for exploring light-driven phenomena and quantum dynamics in extended and finite systems', J Chem Phys, 2020, 152, (12), pp. 124119
24 Noda, M., Sato, S.A., Hirokawa, Y., Uemoto, M., Takeuchi, T., Yamada, S., Yamada, A., Shinohara, Y., Yamaguchi, M., Iida, K., Floss, I., Otobe, T., Lee, K.-M., Ishimura, K., Boku, T., Bertsch, G.F., Nobusada, K., and Yabana, K.: 'SALMON: Scalable Ab-initio light–matter simulator for optics and nanoscience', Comput Phys Commun, 2019, 235, pp. 356-365
25 Shimojo, F., Kalia, R.K., Kunaseth, M., Nakano, A., Nomura, K., Ohmura, S., Shimamura, K., and Vashishta, P.: 'A divide-conquer-recombine algorithmic paradigm for multiscale materials modeling', J Chem Phys, 2014, 140, (18), pp. 18A529
26 Nakano, A., and Ichimaru, S.: 'Dynamic correlations in electron liquids. 1. General formalism', Phys Rev B, 1989, 39, (8), pp. 4930-4937
27 Car, R., and Parrinello, M.: 'The unified approach to density functional and molecular dynamics in real space', Solid State Commun, 1987, 62, (6), pp. 403-405
28 Nakano, A., Vashishta, P., and Kalia, R.K.: 'Massively-parallel algorithms for computational nanoelectronics based on quantum molecular dynamics', Comput Phys Commun, 1994, 83, (2-3), pp. 181-196
29 Sato, S.A., Taniguchi, Y., Shinohara, Y., and Yabana, K.: 'Nonlinear electronic excitations in crystalline solids using meta-generalized gradient approximation and hybrid functional in time-dependent density functional theory', J Chem Phys, 2015, 143, (22), pp. 224116
30 Lian, C., Guan, M.X., Hu, S.Q., Zhang, J.N., and Meng, S.: 'Photoexcitation in solids: first-principles quantum simulations by real-time TDDFT', Adv Theory Sim, 2018, 1, (8), pp. 1800055
31 Martin, R.M.: 'Electronic Structure: Basic Theory and Practical Methods' (Cambridge University Press, 2008. 2008)
32 Vlcek, V., Baer, R., and Neuhauser, D.: 'Stochastic time-dependent DFT with optimally tuned range-separated hybrids: application to excitonic effects in large phosphorene sheets', J Chem Phys, 2019, 150, (18), pp. 184118
33 Wang, C.Y., Elliott, P., Sharma, S., and Dewhurst, J.K.: 'Real time scissor correction in TD-DFT', J Phys-Condens Mat, 2019, 31, (21), pp. 214002
34 Tiwari, S.C., Sakdhnagool, P., Kalia, R.K., Krishnamoorthy, A., Kunaseth, M., Nakano, A., Rajak, P., Shimojo, F., Luo, Y., and Vashishta, P.: 'Quantum dynamics at scale: ultrafast control of emergent functional materials', Proceedings of International Conference on High Performance Computing in Asia-Pacific Region, HPCAsia2020, ACM, 2020
35 Linker, T., Nomura, K., Fukushima, S., Kalia, R.K., Krishnamoorthy, A., Nakano, A., Shimamura, K., Shimojo, F., and Vashishta, P.: 'Induction and Ferroelectric Switching of Flux Closure Domains in Strained $PbTiO_3$ with Neural Network Quantum Molecular Dynamics', Nano Lett, 2023, 23, (16), pp. 7456-7462
36 Rini, M.: 'First light for a next-generation light source', Phys, 2023, 16, pp. 160
37 Tian, G., Yang, W.D., Gao, X.S., and Liu, J.M.: 'Emerging phenomena from exotic ferroelectric topological states', APL Mater, 2021, 9, (2), pp. 020907