Portable and Scalable All-Electron Quantum Perturbation Simulations on Exascale Supercomputers

Zhikun Wu
SKLP, Institute of Computing Technology, CAS
UCAS
Beijing, China
zhikun.wu@outlook.com

Yangjun Wu
SKLP, Institute of Computing Technology, CAS
UCAS
Beijing, China
wuyangjun21s@ict.ac.cn

Ying Liu*
SKLP, Institute of Computing Technology, CAS
Beijing, China
liuying2007@ict.ac.cn

Honghui Shang*
University of Science and Technology of China
Hefei, China
shanghui.ustc@gmail.com

Yingxiang Gao
National Supercomputer Center in Tianjin
Tianjin, China
gaoyx@nscc-tj.cn

Zhongcheng Zhang
SKLP, Institute of Computing Technology, CAS
UCAS
Beijing, China
zhangzhongcheng20s@ict.ac.cn

Yuyang Zhang
SKLP, Institute of Computing Technology, CAS
UCAS
Beijing, China
zhangyuyang22s@ict.ac.cn

Yingchi Long
SKLP, Institute of Computing Technology, CAS
HIT
Beijing, China
i@lyc.dev

Xiaobing Feng
SKLP, Institute of Computing Technology, CAS
UCAS
Beijing, China
fxb@ict.ac.cn

Huimin Cui
SKLP, Institute of Computing Technology, CAS
UCAS
Beijing, China
cuihm@ict.ac.cn

ABSTRACT

Quantum perturbation theory is pivotal in determining the critical physical properties of materials. The first-principles computations of these properties have yielded profound and quantitative insights in diverse domains of chemistry and physics. In this work, we propose a portable and scalable OpenCL implementation for quantum perturbation theory, which can be generalized across various high-performance computing (HPC) systems. Optimal portability is realized through the utilization of a cross-platform unified interface and a collection of performance-portable heterogeneous optimizations. Exceptional scalability is attained by addressing major constraints on memory and communication, employing a locality-enhancing task mapping strategy and a packed hierarchical collective communication scheme. Experiments on two advanced supercomputers demonstrate that our implementation exhibits remarkably performance on various material systems, scaling the system to 200,000 atoms with all-electron precision. This research enables all-electron quantum perturbation simulations on substantially larger molecular scales, with a potentially significant impact on progress in material sciences.

KEYWORDS

Quantum mechanics, Perturbation theory, All-electron, Scalability, Heterogeneous many-core supercomputers, Biological systems

*Corresponding authors.

SC '23, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0109-2/23/11…$15.00
https://doi.org/10.1145/3581784.3607085

ACM Reference Format:
Zhikun Wu, Yangjun Wu, Ying Liu, Honghui Shang, Yingxiang Gao, Zhongcheng Zhang, Yuyang Zhang, Yingchi Long, Xiaobing Feng, and Huimin Cui. 2023. Portable and Scalable All-Electron Quantum Perturbation Simulations on Exascale Supercomputers. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23), November 12–17, 2023, Denver, CO, USA.* ACM, Denver, CO, USA, 12 pages. https://doi.org/10.1145/3581784.3607085

1 INTRODUCTION

A material's properties are generally determined by how it reacts to external perturbations. The quantum perturbation theory [1, 2] uses a mathematical framework to calculate the response of the quantum system and can be used to determine how the system will interact with the environment. As a result, quantum perturbation theory based calculations play a critical role in our comprehension of the physical properties of materials. For example, quantum perturbation theory offers a framework for unraveling the intricate mechanisms underlying superconductivity [3]. This theory can also gauge the mobility of the (electron or hole) carriers in the material [4]. Most strikingly, it provides a direct link between the theoretical and the experimentally measured physical spectrum [5].

Although quantum perturbation theory can precisely calculate the properties of molecules and predict their behavior, it has been traditionally limited to small systems and remains challenging for large systems consisting of up to thousands of atoms. Therefore, a highly scalable quantum perturbation computational method is increasingly necessary for efficient calculations. On the other hand, even the supercomputers can be employed for quantum perturbation calculations, the hardware architecture of different supercomputers varies a lot (e.g. various GPUs or other heterogeneous many-core architectures), making portability between these diverse architectures of utmost importance. Therefore, a highly efficient portable and scalable first-principles code for quantum perturbation calculation on various modern heterogeneous many-core supercomputers is urgently needed. Here in this work, we re-designed the quantum perturbation calculations within the all-electron full-potential framework [5–7] for the heterogeneous many-core supercomputer, with a portable and scalable OpenCL implementation to address the above challenges, and our major innovations include:

- A portable quantum perturbation implementation has been developed using OpenCL, i.e., a cross-platform unified programming framework, to enable efficient all-electron all-potential simulations across various supercomputers.
- Major scaling limitations in quantum perturbation has been removed, by a locality-enhancing task mapping strategy. It enables neighbouring atoms to be simulated in the same MPI process, leading to significantly reduced per-process memory consumption and more efficient memory accesses.
- Efficient collective communication has been enabled, by introducing a packed hierarchical communication scheme, which reduces the total number of collective communications by packing several of them together, and accelerates each communication by applying a hierarchical approach for inter-node and intra-node data synthesizing.
- A set of performance-portable OpenCL optimizations have been implemented to improve the efficiencies of compute units and memory sub-system for various heterogeneous processors, by reordering OpenCL kernel invocations, memory accesses and arithmetic operations.
- We have evaluated our OpenCL-accelerated quantum perturbation simulation on two advanced supercomputers, and results show that our implementation exhibits remarkably performance on various material systems, enabling the system scale to 200,000 atoms with all-electron precision.

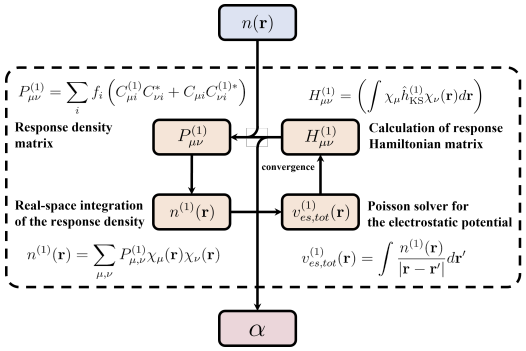


Figure 1: Schematic overview of first-principles quantum perturbation theory.

The optimization methods proposed in this work could be extended to other quantum perturbation code with the same computational characteristics; hence, they will be broadly beneficial to the quantum chemistry, biological, and material science communities.

2 BACKGROUND

2.1 The first-principles quantum perturbation theory

Firstly, we provide a brief summary of the first-principles quantum perturbation method. In the following chapters, we use subscripts i, j to denote occupied quantum molecular orbitals, a for unoccupied (virtual) quantum molecular orbitals, and p, q for the entire set of quantum molecular orbitals. μ, ν represent atomic basis sets. In density function theory (DFT), the total-energy functional is given as

$$E_{\mathrm{KS}} = T_{\mathrm{s}}[n] + E_{\mathrm{ext}}[n] + E_{\mathrm{H}}[n] + E_{\mathrm{xc}}[n] + E_{\text{nuc-nuc}} \tag{1}$$

where $n(\mathbf{r})$ is the electron density, T_{s} denotes the kinetic energy of non-interacting electrons, E_{ext} represents external energy resulting from electron-nuclear attraction, E_{H} refers to Hartree energy, E_{xc} denotes exchange-correlation energy, and $E_{\text{nuc-nuc}}$ refers to the nucleus-nucleus repulsion energy. The ground state electron density $n_0(\mathbf{r})$ is obtained by variationally minimizing Eq. (1) under the constraint that the number of electrons N_e is conserved. This yields the Kohn-Sham single particle equations

$$\hat{h}_{\mathrm{KS}} \psi_p = \left[\hat{t}_{\mathrm{s}} + v_{\mathrm{ext}}(r) + v_{\mathrm{H}} + v_{\mathrm{xc}} \right] \psi_p = \epsilon_p \psi_p \tag{2}$$

for the Kohn-Sham Hamiltonian $\hat{h}_{\mathrm{KS}}$, where $\hat{t}_{\mathrm{s}}$ denotes the kinetic energy operator, v_{ext} the external potential, v_H the Hartree potential, and v_{xc} the exchange-correlation potential. Solving Eq. (2) yields the Kohn-Sham single particle states ψ_p and their eigenenergies ϵ_p, with ψ_p determining the electron density via

$$n(\mathbf{r}) = \sum_i f_i |\psi_i|^2 \tag{3}$$

in which f_i denotes the Fermi-Dirac distribution function. In order to solve Eq. (2) in numerical implementations, the Kohn-Sham states are expanded in a finite basis set $\chi_\mu(\mathbf{r})$

$$\psi_p(\mathbf{r}) = \sum_\mu C_{\mu p} \ \chi_u \mu(\mathbf{r}) \tag{4}$$

with the expansion coefficients $C_{\mu p}$. This expansion leads to a generalized eigenvalue problem for Eq. (2), which is expressed as

$$\sum_{\nu} H_{\mu\nu} C_{\nu p} = \epsilon_p \sum_{\nu} S_{\mu\nu} C_{\nu p} \tag{5}$$

where $H_{\mu\nu}$ and $S_{\mu\nu}$ are the Hamiltonian and overlap matrices, respectively. These matrices are obtained using the bra-ket notation $\langle .|. \rangle$ for the inner product in Hilbert space, where $H_{\mu\nu}$ denotes the elements $\langle \chi_\mu | \hat{h}_{\mathrm{KS}} | \chi\nu \rangle$ of the Hamiltonian matrix and $S_{\mu\nu}$ denotes the elements $\langle \chi_\mu | \chi_\nu \rangle$ of the overlap matrix. The density matrix for the ground state is obtained using the basis set representation as

$$P_{\mu\nu} = \sum_i f_i C_{\mu i} C_{\nu i} \tag{6}$$

where f_i are the occupation numbers. As shown in Figure 1, the first step in the quantum perturbation self-consistency cycle is to calculate the response of the density matrix using the given expansion coefficients C and $C^{(1)}$ as

$$P^{(1)}_{\mu\nu} = \sum_i f_i \left(C^{(1)}_{\mu i} C_{\nu i} + C_{\mu i} C^{(1)}_{\nu i} \right) \tag{7}$$

Using the density matrix formalism, we can then obtain the response of the electronic density as

$$n^{(1)}(\mathbf{r}) = \sum_{\mu,\nu} P^{(1)}_{\mu,\nu} \chi_\mu(\mathbf{r}) \chi_\nu(\mathbf{r}) \tag{8}$$

and the response of the Hartree potential as

$$v^{(1)}_H(\mathbf{r}) = \int \frac{n^{(1)}(\mathbf{r})}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' \tag{9}$$

and the corresponding total electrostatic potential $v^{(1)}_{es,tot}(\mathbf{r}) = v^{(1)}_{\mathrm{ext}} + v^{(1)}_{\mathrm{H}}$, then we get the response of the Kohn-Sham Hamiltonian matrix,

$$H^{(1)}_{\mu,\nu} = \left(\int \chi_\mu \hat{h}^{(1)}_{\mathrm{KS}} \chi_\nu(\mathbf{r}) d\mathbf{r} \right) \tag{10}$$

Here $\hat{h}^{(1)}_{\mathrm{KS}}$ denotes the response of the Hamiltonian operator under the homogeneous external electrical field perturbation with strength ξ along coordinate direction J.

$$\hat{h}^{(1)}_{\mathrm{KS}} = \frac{d(\hat{h}_{\mathrm{KS}} + \hat{h}_{\mathrm{E}})}{d\xi_J} = v^{(1)}_{es,tot} + v^{(1)}_{xc} - r_J \tag{11}$$

In the case of the LDA functional, the exchange-correlation energy can be written as $E_{xc} = \int f_{xc}(n(\mathbf{r})) d\mathbf{r}$. Evaluating the functional derivative in the latter term yields simply

$$v^{(1)}_{xc}[n(\mathbf{r})] = \frac{\partial^2 f_{xc}}{\partial n \partial n} n^{(1)}(\mathbf{r}) = \frac{\partial v_{xc}[n(\mathbf{r})]}{\partial n(\mathbf{r})} n^{(1)}(\mathbf{r}) \tag{12}$$

In turn, all these ingredients then allow to set up the Sternheimer equation [5, 7], the solution of which allows to update the response of the expansion coefficients $C^{(1)}$. We iteratively restart the quantum perturbation loop until self-consistency is reached, i.e., until the changes in $C^{(1)}$ become smaller than a user-given threshold. In the last steps, the polarizability and dielectric constants are computed.

$$\alpha_{IJ} = \frac{\partial \mu_I}{\partial \xi_J} = \int r_I \frac{\partial n(\mathbf{r})}{\partial \xi_J} d\mathbf{r} \tag{13}$$

2.2 Current state of the art

First-principles calculations necessitate the representation of single-particle wave functions through the linear combination of basis functions, resulting in a matrix equation that requires numerical solutions. Various types of basis sets are utilized in different computational codes. For instance, plane wave (PWs) basis functions are employed in codes such as Quantum ESPRESSO [8], VASP [9], ABINIT [10], and Qbox [11]. Additionally, uniform real-space grid basis sets are utilized in Octopus[12], and finite elements is used in DFT-FE [13]. Although these basis sets can achieve systematic convergence, their computational requirements for all-electron calculations are exceptionally high. To address this issue, pseudopotential or projector augmented wave methods are introduced when utilizing the aforementioned basis sets. Despite the careful construction of pseudopotentials to ensure consistency between valence electrons and all-electron calculations, core electron information remains unavailable[14]. On the other hand, all-electron approaches based on atomic orbitals can account for both core and valence electrons, offering higher precision compared to pseudopotential methods. Examples of all-electron software include Gaussian [15] and CRYSTAL [16], which employ Gaussian atomic orbital methods, as well as DMol [17] and FHI-aims [6], which utilize the all-electron numerical atomic orbital basis set. So far, the all-electron quantum perturbation studies of physical properties have been limited in small systems because of the following reasons: (1) For all-electron full-potential calculation, the load balancing for non-uniform grid points is needed; (2) The location-priority algorithm is needed to enhance computational locality and the scalability. (3) Cooperative computing challenges for irregular computations on heterogeneous computing resources. (4) Memory challenges stemming from large-scale matrix computations. In summary, the current quantum perturbation software is limited in terms of accuracy, computational efficiency, and scalability, severely constraining quantum perturbation calculations on large-scale supercomputing systems and forming a critical bottleneck in simulating the physical properties of realistic systems.

2.3 Algorithmic Challenges

Designing an efficient quantum perturbation implementation that is capable of exploiting various accelerators on supercomputers, involves major challenges on both scalability and portability.

The scalability challenge. Quantum perturbation simulation requires huge memory consumption and frequent communication, to keep and update data that involve all atoms (e.g., the Hamiltonian matrix obtained by Equation 10), making it difficult to scale among a large number of MPI processes. This challenge is overcome in this work by a locality-enhancing task mapping strategy and a packed hierarchical collective communication scheme, which will be introduced in detail by Section 3.

The portability challenge. Making a single-source code implementation run efficiently across various supercomputers is demanding yet challenging, since modern supercomputers are typically equipped with various heterogeneous accelerators, favoring different programming languages, e.g., Nvidia-GPU-accelerated supercompupters favor CUDA codes, whereas Sunway favors customized Athread codes [18–20]. This challenge is overcome in this work by

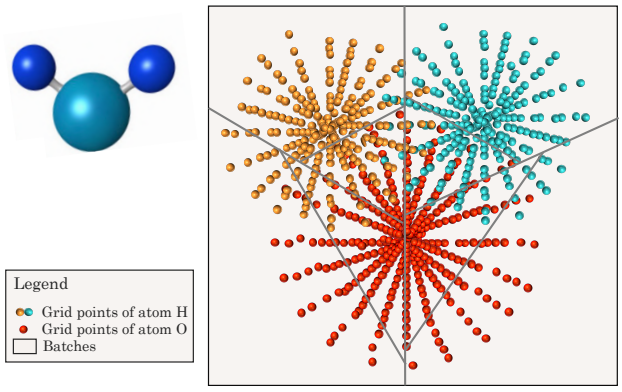


Figure 2: 2D illustration of discretized grids constructed for parallelization in a water molecule (H_2O), with non-uniform grid points generated and batches (of grid points) formed.

implementing each MPI process with a common OpenCL interface and performing massive optimizations, which will be introduced in detail by Section 4.

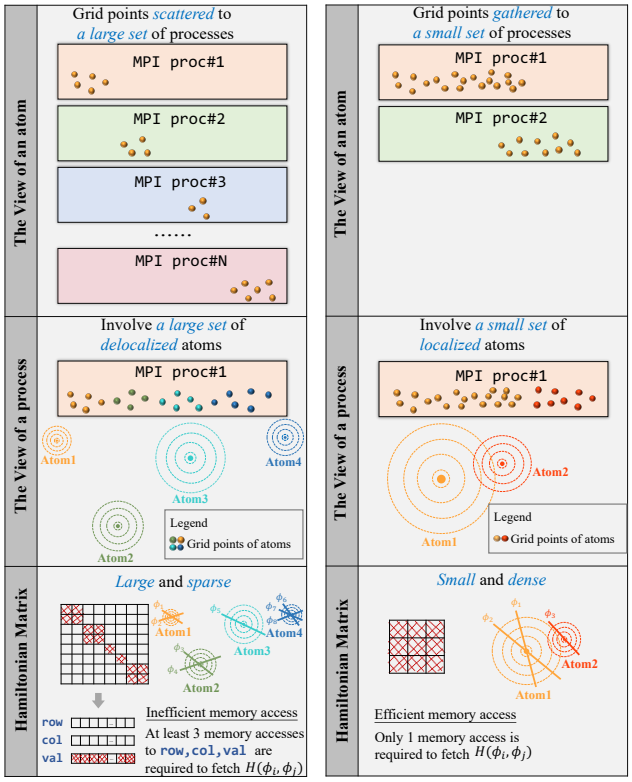
3 INNOVATIONS ON SCALABILITY

Our quantum perturbation implementation improves its scalability among multiple MPI processes, by (1) a locality-enhancing task mapping strategy that improves intra-process locality among atoms (Section 3.1), and (2) a packed hierarchical collective communication scheme that reduces both the number of communications and the cost of each communication (Section 3.2).

3.1 Locality-Enhancing Task Mapping

To harvest the convenience of being parallelized, discretized three-dimensional physical grids have been constructed for perturbation property calculations in Equations (8) (9) (also shown in Figure 1), generating a set of non-uniform radial spherical grid points centered on the geometric coordinates of the nucleus for each atom, to discretize its all-electron atomic orbitals [21, 22]. All points in those discretized grids are further divided into disjoint batches based on their spatial locations, with each batch formed with a grid-adapted cut-plane method [23] and then mapped to a certain MPI process for execution. Figure 2 examplifies the discretized grids using a 3-atom system of H_2O, with each dot representing a grid point and each segmented area representing a batch.

3.1.1 Existing Load-Balancing Strategy and Its Scaling Obstacle. When determining on which MPI process a batch (of grid points) will be executed, a load-balancing strategy [6] has been adopted to deal with variable-sized batches (typically consisting of 100-300 grid points) caused by the non-uniform spatial locations of grid points. The load-balancing approach would assign the new batch, to the MPI process that currently owns the least grid points, without checking to which atoms the grid points in the new batch belong. As a result, grid points belonging to the same atom, may be scattered to a large set of MPI processes, as shown in the first row of Figure 3(a). At the same time, each MPI process may be assigned with grid



(a) Existing load-balancing strategy. (b) Proposed locality-enhancing strategy.

Figure 3: Main differences of two task mapping strategies.

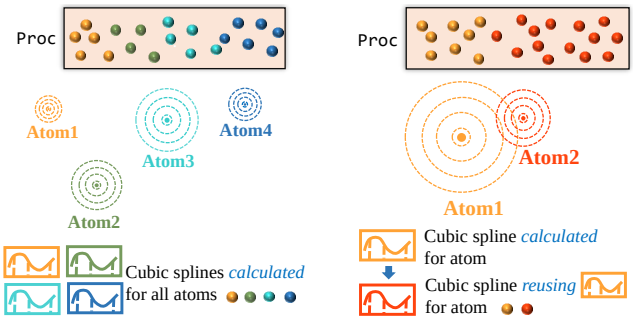
points belonging to a large set of delocalized atoms, as shown in the second row of Figure 3(a).

Memory explosion led by large sparse Hamiltonian matrix is the major scaling obstacle met by letting an MPI process deal with delocalized batches, which may belong to a large set of atoms located far from each other. In this case, in the phases of calculating response density ($n^{(1)}(\mathbf{r})$) (Equation 8) and response Hamiltonian ($H^{(1)}_{\mu,\nu}$) (Equation 10), huge storage space would be required in this MPI process, for storing a large Hamiltonian matrix. It is a square matrix which uses $N_b \times N_b$ elements with N_b denoting the total number of orbits for all atoms involved, which could be extremely huge when this MPI process deals with a large number of atoms, since each atom owns at least one orbit. For example, for a 49-atom system as shown in Figure 8(b), this number of orbitals is $N_b = 1359$, resulting in a 1359×1359 Hamiltonian matrix. In addition, it's typically sparse, since the atoms can only have interactions with its neighbor atoms. As shown in the third row of Figure 3(a), an MPI process may be assigned with grid points belonging to a large set of delocalized atoms located far from each other, resulting in a Hamiltonian matrix that is both large (due to the large number of atoms involved) and sparse (due to their far-away locations), to be kept in its memory. In addition, it's usually stored using a compressed format (e.g., compressed sparse row (CSR) format), leading to inefficiency when accessing its elements, i.e., multiple memory accesses are required to fetch a single element in it.

3.1.2 Proposed Locality-Enhancing Strategy and Its Benefits. To prevent memory explosion for storing the Hamiltonian matrix, our key insight is to gather grid points belonging to adjacent atoms preferentially to the same MPI process, preventing an MPI process from dealing with scattered atoms, to turn a large sparse Hamiltonian matrix into a small dense one, as shown in Figure 3(b).

Memory saving brought by small dense Hamiltonian matrix is the major scaling benefit of the locality-enhancing strategy. As shown in the second row of Figure 3(b), an MPI process would be assigned with grid points belonging to neighbouring atoms, producing a Hamiltonian matrix that is both small (due to the small number of atoms involved) and dense (due to their close locations), to be kept in its memory, as shown in the third row of Figure 3(b). Furthermore, accessing its elements is far more efficient than accessing the sparse one.

It is quite interesting that we have observed an additional performance benefit brought by the proposed locality-enhanced strategy. Certain results (e.g., cubic spline for calculating response potential $(v^{(1)}_{es,tot}(\mathbf{r}))$) could be reused across grid points belonging to adjacent atoms, thus a few redundant computations common to these atoms could be eliminated with improved atom localities. For example, in Figure 4(a), an MPI process has been assigned with grid points from 4 delocalized atoms, making cubic spline calculation performed for all of them, without reusing any result since they locate far away from each other. Whereas in Figure 4(b), 2 closely-located atoms are involved in this MPI process, enabling one atom to reuse the cubic spline result from another.



(a) All splines are calculated in the existing load-balancing strategy.

(b) One spline is reused in the proposed locality-enhancing strategy.

Figure 4: Comparison on cubic spline performed when calculating response potential $(v^{(1)}_{es,tot}(\mathbf{r}))$, between two strategies.

3.1.3 Algorithm. Given a set of batches $\mathcal{B}$ to be assigned to N MPI processes, the locality-enhanced mapping is achieved, by recursively bisecting $\mathcal{B}$ into N subsets. In each iteration of bisection (shown in Figure 5), all batches first project their locations onto a selected dimension, and then form two subsets which own two disjoint sets of such projections and contain similar number of grid points. In particular, the location of each batch is specified with the coordinate averaged across all its grid points, and a dimension (i.e., x, y or z) is selected for a given set of batches if their projections on this dimension spread the largest range.

Algorithm 1 Locality-Enhancing Task Mapping

Input:
$\mathcal{P} = \{proc_1, proc_2, \ldots, proc_N\}$ ▷ A set of N MPI processes
$\mathcal{B} = \{batch_1, batch_2, \ldots, batch_M\}$ ▷ A set of M ($\geq N$) batches
Output:
Map(a set of batches $\rightarrow proc$) ▷ Assign the set to $proc$
1: **function** LOCALITY_ENHANCING_MAPPING($\mathcal{P}$, $\mathcal{B}$)
2: **if** $\mathcal{P}$ includes only one element $proc_k$ **then**
3: Map.insert($\mathcal{B} \rightarrow proc_k$);
4: **else**
5: $\mathcal{P}_l \leftarrow \{proc_1, \ldots, \text{proc}_{\lceil n/2 \rceil}\}$; ▷ n: current size of $\mathcal{P}$
6: $\mathcal{P}_r \leftarrow \mathcal{P} - \mathcal{P}_l$; ▷ Equally partition $\mathcal{P}$ into two sets
7: Let $dim \in \{x, y, z\}$ be the dimension on which the projected coordinates of all $batch \in \mathcal{B}$ spread the largest range
8: $\mathcal{B}$.sort(dim); ▷ Sort all $batch \in \mathcal{B}$ with their projections on dim in non-decreasing order
9: $pivot \leftarrow \frac{1}{2} \sum_{batch \in \mathcal{B}} batch.points$;
10: $\mathcal{B}_l \leftarrow \{batch_1, \ldots, batch_p\}$;
11: $\mathcal{B}_r \leftarrow \mathcal{B} - \mathcal{B}_l$; ▷ p satisfies inequalities $\sum_{1 \leq i \leq p} batch_i.points \leq pivot$ and $\sum_{1 \leq i \leq p+1} batch_i.points > pivot$
12: LOCALITY_ENHANCING_MAPPING($\mathcal{P}_l$,$\mathcal{B}_l$); ▷ Recursive bisection
13: LOCALITY_ENHANCING_MAPPING($\mathcal{P}_r$,$\mathcal{B}_r$);
14: **end if**
15: **end function**

Algorithm 1 describes this strategy in detail, with lines 5-6,12-13 facilitating the recursive bisection scheme, and lines 7-11 performing an iteration of bisection (i.e., the procedure shown in Figure 5). In particular, line 7 determines the projection dimension, line 8 makes the projection and sorts the projected coordinates for all batches, lines 9-11 split those batches into two sets with both batch location projections and total grid points considered.

3.2 Packed Hierarchical Collective Communication

We have further enhanced scalability by two means. First, the number of collective communications has been reduced, by packing some of them into one. Then, each collective communication has been accelerated, by applying a hierarchical approach for intra-node and inter-node data synthesizing.

3.2.1 Packed Collective Communication. The idea is to fuse several invocations of the same MPI collective function into one invocation, which packs together all data previously to be synthesized in those invocations, as shown in Figure 6.

Let an MPI collective function be invoked for c times, with the $i^{th}(1 \leq i \leq c)$ invocation synthesizing an amount of $size_i$ data, then they would be packed into one invocation synthesizing $\sum_{i=1}^{c} size_i$ data. As a result, the number of MPI collective communications would be reduced from c to 1 (e.g., 2 `AllReduce` have been packed into 1 in Figure 6), at the cost of increasing memory consumption from $\max_i(size_i)$ to $\sum_{i=1}^{c} size_i$ in the worst case (e.g., might be doubled in Figure 6). To avoid memory explosion in case of packing excessive data, we have used a simple heuristic to choose a proper c, so that $\sum_{i=1}^{c} size_i$ requires a memory space no more than 30MB. Typically, this is smaller than the capacity of last level cache on main-stream processors, thus negligible costs may be introduced.

A typical such scenario exists in our simulation when calculating response potential $(v^{(1)}_{es,tot}(\mathbf{r}))$, in which the multipole expansion of the response density (i.e., **rho_multipole**) is updated by iteratively invoking `MPIAllReduce`, with each invocation reducing a row of **rho_multipole** among all MPI processes. Several such invocations

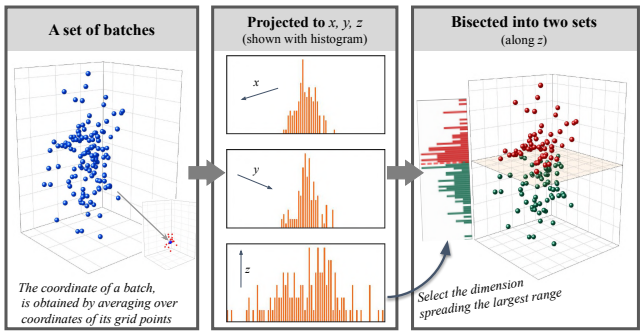


Figure 5: Bisect a set of batches with locality enhanced.

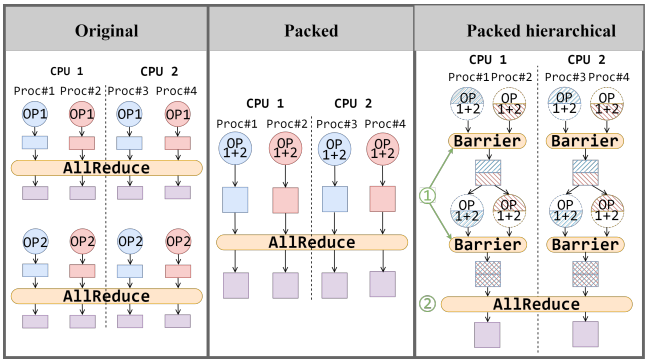


Figure 6: Packed hierachical collective communication, with two 4-process `AllReduce` first packed into one, and then broken into ① 2 local `Barriers`, and ② a 2-process `AllReduce`.

could be packed, with each packed invocation updating several rows in **rho_multipole**.

3.2.2 Hierarchical Collective Communication. The idea is to break each collective communication, into a set of light-weight local synchronizations for intra-node data synthesizing, followed by a less-intensive collective communication for inter-node data synthesizing. Each local synchronization is performed among processes executed on a shared-memory computing node (e.g., a multi-core CPU), thus requires negligible costs. The less-intensive collective communication involves only a small subset of processes that are previously involved, thus improves its efficiency significantly.

Let an N-process collective communication be conducted on a certain matrix $\mathbf{A}$, thus before optimization, partial results from all N duplicated copies of $\mathbf{A}$ need to be synthesized. Our key insight is to make one such copy accessible by m MPI processes on a shared-memory node (with MPI SHM extension [24]), so that it could be updated leveraging a set of m-process synchronizations. After updating all $\frac{N}{m}$ copies in this way, the N-process collective communication could be narrowed to $\frac{N}{m}$ processes, with memory consumption also reduced to $\frac{1}{m}$.

To update an m-process-shared copy of $\mathbf{A}$, we first sliced it into m chunks, and then perform m synthesizations sequenced by local barriers, with each chunk synthesizing its m partial results from m processes in turn without write conflicts. Figure 6 illustrates the hierarchical scheme using an example of `AllReduce` with $N = 4, m = 2$, in which the intensity of `AllReduce` has been reduced from among 4 processes to 2, and the required storage has been reduced from 4 data copies to 2.

4 INNOVATIONS ON PORTABILITY

In this section, we first describe our OpenCL implementation for quantum perturbation (Section 4.1), then introduce optimizations for its kernels (Sections 4.2- 4.4), i.e., codes that execute on accelerators such as GPUs, and finally discuss its portability to HPC systems equipped with various accelerators (Section 4.5).

4.1 A Portable Implementation with OpenCL

We have re-written all time-consuming calculations for quantum perturbation (as shown in Figure 1) with the OpenCL programming interface, enabling them to be accelerated on most existing accelerators, thus portable among various HPC systems. In particular, OpenCL-accelerated calculations include four parts of calculation of response density matrix ($P^{(1)}_{\mu\nu}$)) , real-space integration of the response density ($n^{(1)}(\mathbf{r})$) , Poisson solver for the response potential ($v^{(1)}_{es,tot}(\mathbf{r})$) , and calculation of response Hamiltonian ($H^{(1)}_{\mu,\nu}$) . Principals below have been applied to our implementation, which theorically could be generalized onto any acclerator and have been validated on the two accelerators with diverse architecture (i.e., SW39010 and AMD GCN GPU).

Parallelism. In each OpenCL kernel, two-level fine-grained parallelism is facilitated, across batches and grid points respectively. In particular, each OpenCL work-item (i.e., a thread) deals with a grid point, each OpenCL work-group (i.e., a group of threads) consists of work-items that process grid points in the same batch, and the OpenCL NDRange (i.e., the entire work space) includes all batches assigned to the MPI process launching this kernel.

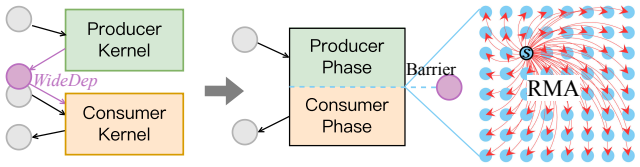
Data. Arrays required or produced by each OpenCL kernel, are stored using buffers residing in the *__global* address space, mapped onto the accelerator's off-chip memory. For those arrays that are reused for several times, *__local* address space is exploited, to place array elements reused in a long distance into the accelerator's on-chip memory, reducing costly data round trip to off-chip memory.

Synchronization. In each OpenCL kernel, only synchronization among work-items in the same work-group is allowed, in need of synthesizing all work-groups in a kernel, it would be split into two kernels at the synchronizing point.

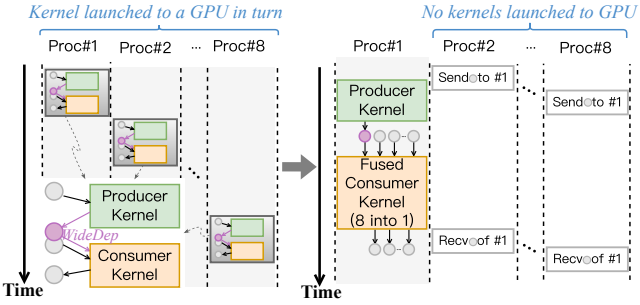
In particular, our OpenCL implementation has included extensive code optimizations (e.g., loop fusion/tiling/permutation [25–27], array contraction [28–30], control flow simplification [31, 32], etc), enabling considerable computing efficiencies. However in this paper, we focus on further optimizations beyond these traditional approaches as following.

4.2 Fusing Kernels with Wide Dependence

A number of benefits could be brought by fusing several OpenCL kernels into one, e.g., reducing data trips to/from long-latency off-chip memories, eliminating inter-kernel redundancies, improving parallelism, etc. There exist some compiler tools [33, 34] to automate this, but restricted to kernels with narrow dependence, i.e., one or more threads in a producer kernel correspond to a thread in a



(a) Vertical fusion on SW39010, with data exchanged by RMA.



(b) Horizontal fusion on AMD GPU, with data residing in GPU memory.

Figure 7: Fusing kernels with wide dependence (related with the data marked as purple circles).

consumer kernel. As a result, fusing kernels with wide dependence, i.e., one thread in a producer kernel corresponds to multiple threads in a consumer kernel, is still challenging.

We have succeeded to fuse widely-dependant kernels by leveraging various on-chip data-sharing mechanisms, with two alternative fusing strategies (i.e., vertical fusion and horizontal fusion), proposed targeting SW39010 and AMD GCN GPU respectively.

The response potential ($v^{(1)}_{es,tot}(\mathbf{r})$) is calculated by iteratively invoking two OpenCL kernels that are widely dependent. In each invocation, the producer kernel would generate the two sets of spline coefficients (i.e., **rho_multipole_spl** for multipole expansion of the response density, and **delta_v_hart_part_spl** for hartree potential) that are required by each thread in the consumer kernel, for calculating the spline-interpolated values of the multipole components and the multipole density. In particular, an identical producer kernel (for the same atom), is invoked across all MPI processes, trading redundant calculations for communication avoidance. Our fusing methods are introduced with this example.

4.2.1 Vertical Fusion Leveraging RMA on SW39010. Widely-dependent producer and consumer kernels invoked in the same MPI process, have been fused into one new kernel vertically on SW39010, as shown in Figure 7(a). In the fused kernel, the order between its two phases (i.e., calculating spline coefficients and spline-interpolated values) is preserved by a global barrier, which stalls all threads from preceding to the latter phase until all elements in **rho_multipole_spl** and **delta_v_hart_part_spl** produced by the former phase have been gathered and then broadcasted to each thread. Efficient global barriers are enabled by the RMA (Remote Memory Access) mechanism on SW39010, which allows data (less than 64KB) to be transferred asynchronously among on-chip memories for 64 neighbouring cores.

In this way, data like **rho_multipole_spl** could be kept on-chip as intermediate data in the fused kernel, avoiding costly data round trip to the off-chip memory as kernel arguments (before fusion), thus yielding notable performance improvements.

4.2.2 Horizontal Fusion Leveraging GPU Memory on AMD GPU. We have observed that by horizontally fusing kernels invoked across different MPI processes, redundant calculations in the producer kernel could be eliminated. As shown in Figure 7(b), kernels from adjacent 8 MPI processes are fused on a supercomputer whose computing node consists of a 32-core CPU and 4 GPUs, thus before fusion, those 8 MPI processes launch their kernels to the same GPU in turn. The horizontal fusion generates a fused producer kernel and a fused consumer kernel, yielding two major benefits.

First, costly redundant calculations have been eliminated. Before fusion, identical producer kernels are invoked across neighbouring MPI processes to generate identical sets of **rho_multipole_spl** and **delta_v_hart_part_spl** to be feed into various consumers. After fusion, one such producer kernel is able to serve a horizontally fused consumer kernel assembled from 8 un-fused consumers (shown in Figure 7(b)), with redudencies reduced.

Second, data round trips between CPU and GPU memories have been reduced, by letting **rho_multipole_spl** and **delta_v_hart_part_spl** reside in GPU memory, without transferring to CPU memory back and forth. More importantly, the kept data could be fed into 8 consumers in the fused consumer kernel, as illustrated by the purple circles in Figure 7(b).

4.3 Eliminating Indirect Memory Accesses

Indirect memory access patterns of the form $\mathbf{A}[\mathbf{B}[i]]$, exhibit weak spatial locality (of $\mathbf{A}$) thus lead to significant memory inefficiencies.

To eliminate them, our key insight is to build a mapping f from array $\mathbf{A}$ to a new array $\mathbf{C}$ (i.e., $\mathbf{C} = f(\mathbf{A})$), so that $\mathbf{C}[i] = \mathbf{A}[\mathbf{B}[i]]$ is satisfied for all possible values of i. In this way, the indirect access $\mathbf{A}[\mathbf{B}[i]]$ could be replaced with a direct access $\mathbf{C}[i]$, improving memory sub-system efficiencies.

Typically, huge run-time overheads would be introduced for building a such mapping, but we have enabled it to be built once and reused multiple times, to reduce the overhead. Given a simulated system, we have observed that for some arrays, f is fixed across all its simulations, e.g., on various supercomputers, using different numbers of computing nodes. Thus f could be built in the first simulation and kept for further simulations. Take indirect access **coord_center**[**atom_list**[*icenter*]] (in the initialization phase for grid partitioning) for example, where *icenter* denotes an atom's global ID but array **coord_center** is indexed by a local atom ID in a batch. This mismatch requires to convert an atom's global ID to its local ID by **atom_list**[*icenter*], and we have built a mapping f to re-arrange elements in **coord_center** to generate an array indexed directly by global atom ID, thus eliminate this indirect access. In addition, this mapping (i.e., array elements re-arrangement) is only required when simulating a system for the first time.

4.4 Fine-Grained Parallelization

To improve the efficiency of parallel compute units, we have enabled fine-grained parallelization within the same work-group under the SIMT (Single Instruction Multiple Threads) execution model (e.g., GPUs), by removing inter-thread dependence.

This fine-grained parallelism has been achieved, by collapsing a nested loop with loop-carried dependence into a single-level loop without such dependence, so that this loop could be paralleled among multiple threads. For example, the following loop is excerpted from calculating the partitioned hartree potential using the Adams-Moulton linear multistep integrator, in the phase of calculating response potential ($v^{(1)}_{es,tot}(\mathbf{r})$) . It is two-level nested, with innermost level dependent on the outermost level, thus can only be paralleled among p_{max} threads. This results in poor parallelism due to the small value (≤ 9) of p_{max}, which refers to the maximum angular momentum component of the atom.

```
for (p = 0; p <= p_max; p++)
  for (m = -p; m <= p; m++){
    idx = p^2 + m + p; A[idx] = func(p, m);
```

We have collapsed it into a loop without such dependence,

```
for (idx = 0; idx < (p_max + 1)^2; idx++){
    p = sqrt(idx); m = idx - p^2 - p; A[idx] = func(p, m);
```

so that it could be parallelized among $(p_{max} + 1)^2$ threads, achieving significantly improved parallelism.

4.5 Discussions on Portability

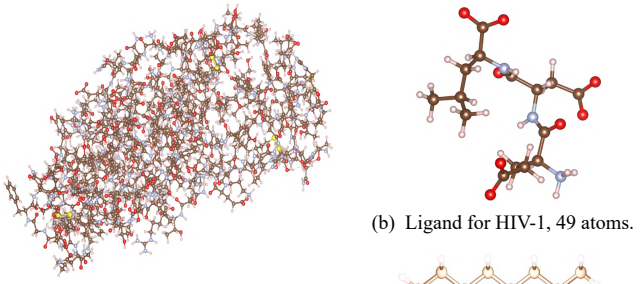
In aspect of functional portability, our OpenCL implementation described in Section 4.1 is capable of ensuring correct quantum perturbation simulation on any OpenCL-supported supercomputer.

In aspect of performance portability, performance improvements could be expected across multiple supercomputers with optimizations in Sections 4.2- 4.4, since these optimizations are designed based on typical heterogeneous accelerating architectures (e.g., GPUs). In addition, accelerators with specific target architecture could also benefit from a part of those optimizations, and we have included some brief but in-depth analysis in the following.

First, fusing kernels with wide dependence as in Section 4.2, provides two alternatives of vertical fusion and horizontal fusion, to adapt various architectures. The former would benefit accelerators with on-chip communication mechanisms, whereas the latter would benefit accelerators allowing data to reside in memory even after kernel completion (e.g., GPUs).

Second, improving memory sub-system efficiencies as in Section 4.3 as well as traditional optimizations mentioned in the last paragraph of Section 4.1 (e.g., loop tiling, array contraction, etc), is applicable to a large set of accelerators with multiple-level memory hierarchy, since those memory optimizations are designed following the common principle of minimizing cross-level data movements. As a result, when ported to other accelerators, performance profits could be expected with just a few tuning, e.g., tiling size.

Third, exploiting fine-grained parallelism as in Section 4.4, can improve performances on accelerators that equip a large set of compute units, which is typical on modern GPUs, with Nvidia GPU having CUDA cores, whereas AMD GPU having vector lanes, yielding significant performance benefits by preventing their compute units from idleness. However this may not be profitable on processors exploiting a few coarse-grained parallel compute units, e.g., A64FX (ARM64 on Fukagu) includes 52 cores.



(a) Receptor-binding domain (RBD) on the Spike protein of SARS-CoV-2, 3006 atoms.

(b) Ligand for HIV-1, 49 atoms.

(c) Polyethylene molecular.

Figure 8: The structure of three biomolecular systems.

5 EVALUATION

5.1 Evaluation Setup

We have evaluated our OpenCL implementation for quantum perturbation, on two supercomputers with major differences in their architectures, across a set of valuable bio-molecular systems.

Two Supercomputers are used in our evaluation. The first one (`HPC #1`), is the new generation Sunway supercomputer, i.e., the latest machine in Sunway family. Each node contains a SW39010 heterogeneous CPU with 390 cores including 6 managing cores and 384 accelerating cores, and nodes are connected via a customized network. For compilation, `swcl` [35] is used. The second one (`HPC #2`), is an AMD-GPU-accelerated supercomputer, on which each compute node is equipped with one 32-core 2.50GHz x86 CPU and accelerated by 4 AMD Radeon Instinct MI50 GPUs interconnected by the PCIe, with each GPU designed based on GNU architecture and consisted of 4096 cores in 64 CUs. Compute nodes are connected using an Infiniband network. For compilation, `ROCM` [36] is used.

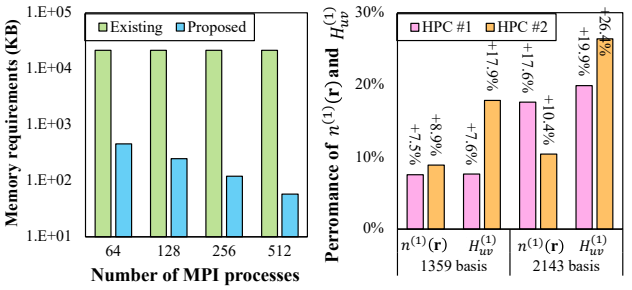
Three systems are simulated in our evaluation, as shown in Figure 8. The first one (Figure 8(a)) is the receptor-binding demain (RBD) on the Spike protein of SARS-CoV-2, for which quantum perturbation simulation helps obtain more protein structure information to understand the binding process to ACE2 [37]. The second one (Figure 8(b)) is the ligand for the HIV-1 protease complex (PDB 1a30). The third one (Figure 8(c)) is the $H(C_2H_4)_nH$ molecules, which are used for the scaling evaluation, all calculations use light settings and the LDA functional.

5.2 Performance Results

In this section, performance speedups for all innovations proposed in Sections 3- 4, will be presented on both supercomputers.

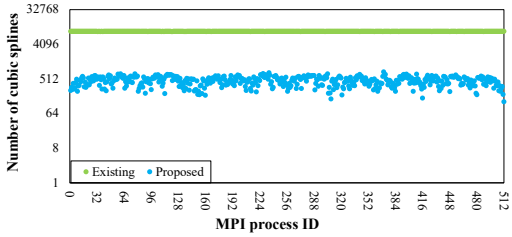
In particular, performances have been further improved relative to the previous work in [37], which accelerated DFPT using Athread programming interface on `HPC #1`. Speedups have been calculated over an OpenCL implementation [38] that is both algorithmically equivalent to and performance-comparable with [37], so that our innovations could be evaluated on both `HPC #1` and `HPC #2`.

5.2.1 Locality-Enhancing Task Mapping. Figure 9 gives results on the locality-enhancing task mapping strategy introduced in



(a) Per-process memory used for Hamiltonian matrix of RBD (9210 basis funcs).

(b) Performance improvements obtained for HIV − 1 with different basis funcs.

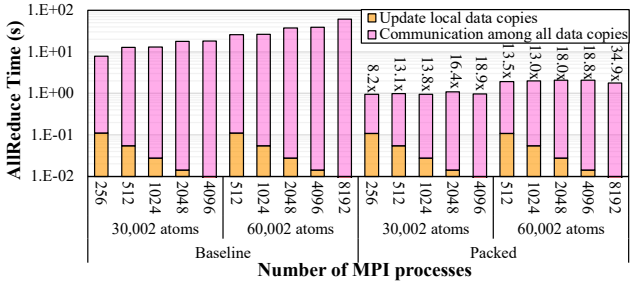


(c) Number of cubic splines performed for RBD.

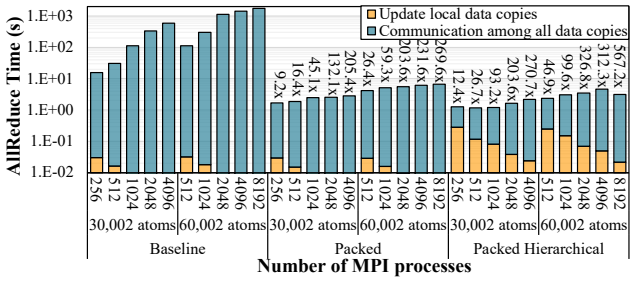
Figure 9: Results on locality-enhancing task mapping.

Section 3.1, compared with existing load-balancing strategy. Figure 9(a) compares their memory requirements, with 21,373KB storage needed for each task to keep a large sparse Hamiltonian matrix under the existing load-balancing strategy, whereas only 58-455KB is needed to store a local dense Hamiltonian matrix on average (across all MPI tasks), under our proposed locality-enhancing strategy. This 2-order of magnitude saving on memory consumption, not only avoids memory explosion thus improves scalability, also enhances the performances of calculating the response density ($n^{(1)}(\mathbf{r})$) and response Hamiltonian ($H^{(1)}_{\mu,\nu}$) as shown in Figure 9(b). Due to improved efficiencies on dense-matrix-access compared with sparse-matrix-access, performances of calculating response density ($n^{(1)}(\mathbf{r})$) have been increased by 7.5% − 19.9%, and for response Hamiltonian ($H^{(1)}_{\mu,\nu}$) 7.6% − 26.4%, benefiting both supercomputers. Figure 9(c) also shows that we are able to reduce the number of cubic splines performed when calculating response potential ($v^{(1)}_{es,tot}(\mathbf{r})$), yielding 9.5% improvement on HPC #1.

5.2.2 Packed Hierarchical Collective Communication. Figure 10 shows the AllReduce time spent on synthesizing rho_multipole among all MPI tasks after calculating response density ($n^{(1)}(\mathbf{r})$), obtained with baseline MPIAllReduce and our proposed packed hierarchical scheme in Section 3.2, respectively. Results show that by packing every 512 MPIAllReduce invocations into one, communication time among all copies of **rho_multipole** has been significantly reduced, achieving speedups of 8.2 × −34.9× on HPC #1 and 9.2 × −269.6× on HPC #2 respectively, across thousands of MPI processes when simulating 30,002/60,002 atoms. Those speedups have been further enhanced to 12.4 × −567.2× on HPC #2, by letting every 32 MPI process keep one data copy and perform local update on it in-turn, yielding more savings on global communication time at a slight cost of local update. Whereas this is not applicable to HPC #1, since MPI processes mapping to the same node are executed on cores with their memories physically dis-connected.



(a) AllReduce time for $H(C_2H_4)_nH$ on HPC #1, with speedups annotated.



(b) AllReduce time for $H(C_2H_4)_nH$ on HPC #2, with speedups annotated.

Figure 10: Results on packed hierarchical communication.

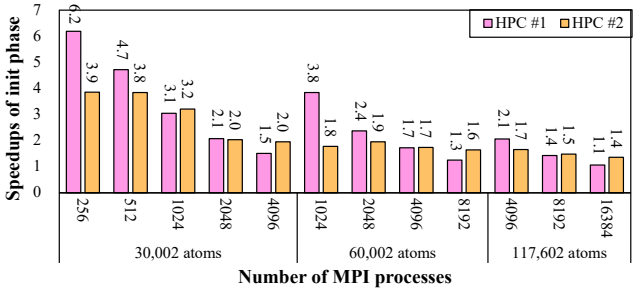
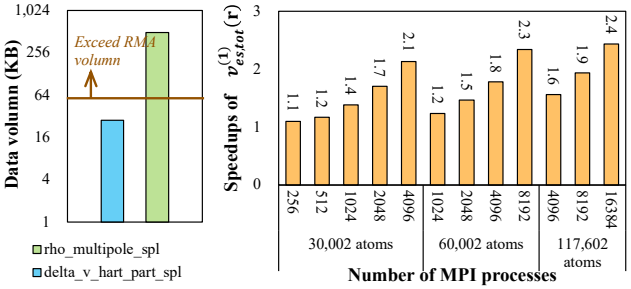


Figure 11: Results on eliminating indirect memory accesses.



(a) Inter-kernel shared data.

(b) Speedups of horizontal fusion on HPC #2.

Figure 12: Results on fusing widely-dependent kernels.

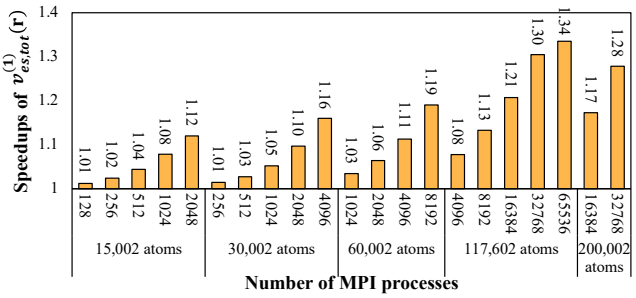


Figure 13: Results on fine-grained parallelism for HPC #2.

5.2.3 Fusing Kernels with Wide Dependences. Figure 12 shows results of fusing widely-dependent kernels as presented in Section 4.2. The response potential ($v^{(1)}_{es,tot}(\mathbf{r})$) is originally calculated by two widely-dependent kernels, with spline coefficients produced by the first kernel and then consumed by each thread in the second kernel. As shown in Figure 12(a), the two sets of coefficents requires 28KB/498KB memory to be allocated on each MPI process, with the latter exceeding the RMA volume limit (64KB) on HPC #1, thus noticeable speedups would not be observed with vertical fusion. In comparison, calculation strength of the first kernel could be reduced by horizontal fusion on HPC #2, leading to speedups upto 2.4× as shown in Figure 12(b).

5.2.4 Eliminating Indirect Memory Accesses. Figure 11 shows speedups obtained by eliminating indirect memory accesses as in Section 4.3, for $H(C_2H_4)_nH$. By re-arranging elements in **coord_center**, we have eliminated the indirect access **coord_center**[**atom_list**[$icenter$]], yielding speedups upto 6.2 × /3.9× on HPC #1/HPC #2, for the initialization (i.e., partitioning the 3D grid) phase. Greater improvements have been observed on HPC #1 due to longer off-chip memory access latency compared with HPC #2, but both performances have been notably enhanced, demonstrating a good performance portability of our approach.

5.2.5 Fine-Grained Parallelism. Figure 13 shows speedups obtained by removing inter-thread dependence as introduced in Section 4.4, when calculating response potential ($v^{(1)}_{es,tot}(\mathbf{r})$) for $H(C_2H_4)_nH$. By collapsing a two-level loop with loop-carried dependence, into a one-level loop without such dependence, the loop is able to be parallelized among all threads in a wavefront on HPC #2, yielding speedups upto 1.34×. We have noticed that as the number of MPI processes grows, better performance could be achieved due to less idleness in compute units.

5.2.6 Overall Impacts. Figure 14 illustrates the execution time spent on each phase before and after optimization, to demonstrate the overall effectiveness of our innovations. Several typical cases simulating different biomolecular systems using various numbers of MPI tasks, are given for both HPC #1 and HPC #2 respectively. Before optimization, time-consuming phases vary across cases, e.g., calculating response potential ($v^{(1)}_{es,tot}(\mathbf{r})$) and response density matrix ($P^{(1)}_{\mu\nu}$)) take a notable time share when

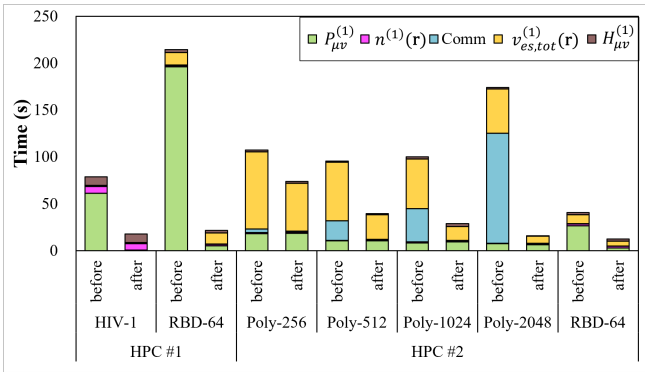
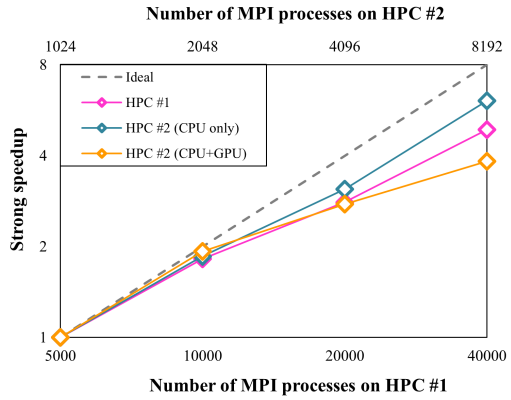


Figure 14: Overall impacts on various phases across several typical cases, with each case denoted by both simulated biomolecular system and used MPI tasks, where Poly represents $H(C_2H_4)_{5000}H$ consists of 30,002 atoms.

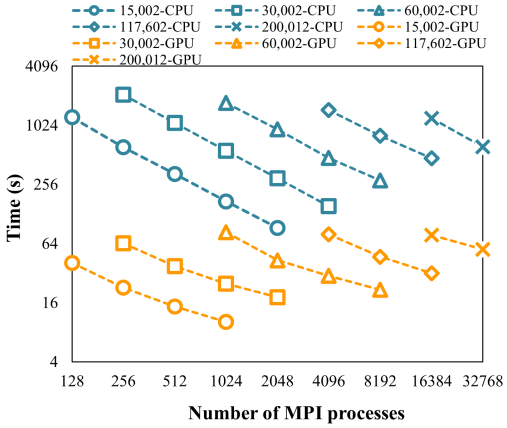
simulating RBD and $H(C_2H_4)_{5000}H$ respectively. Their execution time has been reduced significantly with proposed innovations, e.g., achieving 36.5× speedups for the phase of calculating response density matrix ($P^{(1)}_{\mu\nu}$)) in RBD with 64 MPI tasks on HPC #1 and 6.47× speedups for the phase of calculating response potential ($v^{(1)}_{es,tot}(\mathbf{r})$) in $H(C_2H_4)_{5000}H$ with 2048 MPI tasks on HPC #2. Moreover, communication costs (denoted as Comm) grows large when the number of MPI tasks increases before optimization, which have been significantly reduced after optimization, e.g., by 90.7% when simulating $H(C_2H_4)_{5000}H$ with 2048 MPI tasks on HPC #2. By reducing various phases costs, our innovations have succeeded in reducing overall execution time across different cases, achieving speedups upto 11.1×.

5.3 Scalability Results

5.3.1 Strong Scalability. Figure 15 shows strong scaling results on both supercomputers, calculated with the polyethylene molecular system $H(C_2H_4)_nH$. Five systems ($n \in \{2500, 5000, 10000, 19600, 33335\}$) are used in our evaluation, involving up to 200,012 atoms and 533,360 electrons. First, strong speedups for both supercomputers, are given in Figure 15(a), using the 60,002-atom system ($n = 10000$). On HPC #1, a speedup of 1.85× could be achieved with 10000 MPI processes compared with 5000 processes, yielding a parallel efficiency of 92.6%. This slightly drops beyond 10000 processes, i.e., 2.81 × /4.88× for 20000/40000 tasks, due to mildly increased communication costs. Results are similar on HPC #2 using only its CPU cores, with speedups of 1.86 × /3.10 × /6.08× obtained using 2048/4096/8192 processes compared with 1024 processes. While GPUs on HPC #2 are included for acceleration, strong speedups are slightly less impressive. This performance deterioration arises, mainly from communications in the phase of calculating response density matrix ($P^{(1)}_{\mu\nu}$)) . In particular, 22.5%/28.6%/38.9%/39.1% of time has been spent on $P^{(1)}_{\mu\nu}$ using 1024/2048/4096/8192 MPI processes, leading to reduced parallel efficiency. Figures 15(b) provides more detailed execution time for them on HPC #2 (using GPUs),



(a) Log-log strong speedup with 60,002 atoms.



(b) Time to solution per cycle on HPC #2.

Figure 15: Strong scaling on two HPC systems for the $H(C_2H_4)_nH$, with $n \in \{2500, 5000, 10000, 19600, 33335\}$.

showing that with our efficient OpenCL implementation, quantum perturbation simulation per cycle on 200,002 atoms could be completed within 1 minutes.

5.3.2 Weak Scalability. Figure 16 gives weak scaling results on both supercomputers when simulated atoms increase from 30,002 to 200,012. Results shows that, satisfied weak scalability has been achieved on both supercomputers, with parallel efficiencies 76.7%/75.3%/74.1% obtained for 200,012 atoms, on HPC #1/HPC #2 (CPU only)/HPC #2 (with GPUs) respectively. For small systems, the scaling from the response density matrix computation ($O(N^{1.2})$) dominates the performance, whereas for large systems, the computation of the response potential determines the value and make it increase to $O(N^{1.7})$. In other words, an increase of the atom numbers increases the scaling and thus decreases the weak scalability.

5.3.3 Discussions on Scalability Results. Comparing scaling results in Figures 15 16, before-optimizations results tend to (1) fail on large molecular systems, and (2) exhibit higher scaling efficiencies but lower FLOPs/sec. First, without innovations in Section 3.1, before-optimizations results will fail to scale due to memory explosion, e.g., the Hamiltonian matrix for 50000 atoms requires approximately

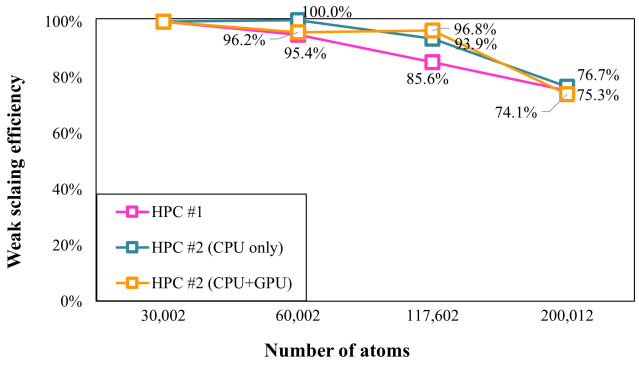


Figure 16: Weak scaling on two HPC systems for $H(C_2H_4)_nH$, with HPC #1 uses 2500/5000/10000/20480 MPI processes and HPC #2 uses 2048/4096/8192/16384, respectively.

16GB memory (assume two basis functions per atom and 10% sparsity), exceeding typical per-process memory capacity (e.g., 4GB on HPC #2). Second, before-optimizations results will scale more efficiently but run much slower, since OpenCL parts will execute longer without innovations in Sections 4.2- 4.4, making the non-accelerated part less dominant.

6 CONCLUSION

In this paper, a portable and scalable OpenCL implementation for quantum perturbation theory has been proposed, enabling efficient all-electron all-potential simulations to be performed across various high-performance computing systems. Portability on both functionality and performance has been achieved, by utilizing a cross-platform unified interface and a collection of advanced heterogeneous optimizations. Exceptional scalability has been obtained by eliminating major memory and communication limitations with a locality-enhancing task mapping strategy and a packed hierarchical collective communication scheme. We have evaluated our OpenCL implementation on two advanced supercomputers, enabling the system scale to 200,000 atoms with all-electron precision, with the potential for a substantial impact on progress in the prediction of the chemical and physical properties.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (2021ZD0110101), the National Natural Science Foundation of China (T2222026, 22003073, 62232015, 62090024), and the Innovation Funding of ICT CAS (E361010).

REFERENCES

[1] Xavier Gonze. First-principles responses of solids to atomic displacements and homogeneous electric fields: Implementation of a conjugate-gradient algorithm. *Phys. Rev. B*, 55(16):10337–10354, April 1997.
[2] Stefano Baroni, Stefano de Gironcoli, Andrea Dal Corso, and Paolo Giannozzi. Phonons and related crystal properties from density-functional perturbation theory. *Rev. Mod. Phys.*, 73:515–562, Jul 2001.
[3] Feliciano Giustino, Jonathan Yates, Ivo Souza, Marvin Cohen, and Steven Louie. Electron-Phonon Interaction via Electronic and Lattice Wannier Functions: Superconductivity in Boron-Doped Diamond Reexamined. *Phys. Rev. Lett.*, 98(4):047005, jan 2007.
[4] Samuel Poncé, Wenbin Li, Sven Reichardt, and Feliciano Giustino. First-principles calculations of charge carrier mobility and conductivity in bulk semiconductors

and two-dimensional materials. *Reports on Progress in Physics*, 83(3):036501, mar 2020.

[5] Honghui Shang, Nathaniel Raimbault, Patrick Rinke, Matthias Scheffler, Mariana Rossi, and Christian Carbogno. All-electron, real-space perturbation theory for homogeneous electric fields: theory, implementation, and application within DFT. *New Journal of Physics*, 20(7):073040, jul 2018.

[6] Volker Blum, Ralf Gehrke, Felix Hanke, Paula Havu, Ville Havu, Xinguo Ren, Karsten Reuter, and Matthias Scheffler. Ab initio molecular simulations with numeric atom-centered orbitals. *Comput. Phys. Commun.*, 180(11):2175–2196, November 2009.

[7] Honghui Shang, Christian Carbogno, Patrick Rinke, and Matthias Scheffler. Lattice dynamics calculations based on density-functional perturbation theory in real space. *Computer Physics Communications*, 215:26–46, jun 2017.

[8] Paolo Giannozzi, Stefano Baroni, Nicola Bonini, Matteo Calandra, Roberto Car, Carlo Cavazzoni, Davide Ceresoli, Guido L Chiarotti, Matteo Cococcioni, Ismaila Dabo, Andrea Dal Corso, Stefano de Gironcoli, Stefano Fabris, Guido Fratesi, Ralph Gebauer, Uwe Gerstmann, Christos Gougoussis, Anton Kokalj, Michele Lazzeri, Layla Martin-Samos, Nicola Marzari, Francesco Mauri, Riccardo Mazzarello, Stefano Paolini, Alfredo Pasquarello, Lorenzo Paulatto, Carlo Sbraccia, Sandro Scandolo, Gabriele Sclauzero, Ari P Seitsonen, Alexander Smogunov, Paolo Umari, and Renata M Wentzcovitch. Quantum espresso: a modular and open-source software project for quantum simulations of materials. *Journal of Physics: Condensed Matter*, 21(39):395502 (19pp), 2009.

[9] G. Kresse and J. Hafner. Ab initio molecular dynamics for liquid metals. *Phys. Rev. B*, 47:558–561, Jan 1993.

[10] Aldo H. Romero, Douglas C. Allan, Bernard Amadon, Gabriel Antonius, Thomas Applencourt, Lucas Baguet, Jordan Bieder, François Bottin, Johann Bouchet, Eric Bousquet, Fabien Bruneval, Guillaume Brunin, Damien Caliste, Michel Côté, Jules Denier, Cyrus Dreyer, Philippe Ghosez, Matteo Giantomassi, Yannick Gillet, Olivier Gingras, Donald R. Hamann, Geoffroy Hautier, François Jollet, Gérald Jomard, Alexandre Martin, Henrique P.C. Miranda, Francesco Naccarato, Guido Petretto, Nicholas A. Pike, Valentin Planes, Sergei Prokhorenko, Tonatiuh Rangel, Fabio Ricci, Gian Marco Rignanese, Miquel Royo, Massimiliano Stengel, Marc Torrent, Michiel J. Van Setten, Benoit Van Troeye, Matthieu J. Verstraete, Julia Wiktor, Josef W. Zwanziger, and Xavier Gonze. ABINIT: Overview and focus on selected capabilities. *Journal of Chemical Physics*, 152(12), 2020.

[11] Francois Gygi, Erik W. Draeger, Martin Schulz, Bronis R. de Supinski, John A. Gunnels, Vernon Austel, James C. Sexton, Franz Franchetti, Stefan Kral, Christoph W. Ueberhuber, and Juergen Lorenz. Large-scale electronic structure calculations of high-z metals on the bluegene/l platform. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, page 45–es, New York, NY, USA, 2006. Association for Computing Machinery.

[12] Xavier Andrade, David Strubbe, Umberto De Giovannini, Ask Hjorth Larsen, Micael J. T. Oliveira, Joseba Alberdi-Rodriguez, Alejandro Varas, Iris Theophilou, Nicole Helbig, Matthieu J. Verstraete, Lorenzo Stella, Fernando Nogueira, Alán Aspuru-Guzik, Alberto Castro, Miguel A. L. Marques, and Angel Rubio. Real-space grids and the octopus code as tools for the development of new simulation approaches for electronic systems. *Phys. Chem. Chem. Phys.*, 17:31371–31396, 2015.

[13] Sambit Das, Phani Motamarri, Vikram Gavini, Bruno Turcksin, Ying Wai Li, and Brent Leback. Fast, scalable and accurate finite-element based ab initio calculations using mixed precision computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, New York, NY, USA, nov 2019. ACM.

[14] Kurt Lejaeghere, Gustav Bihlmayer, T. Bjorkman, Peter Blaha, S. Blugel, Volker Blum, Damien Caliste, Ivano E Castelli, Stewart J Clark, Andrea Dal Corso, Stefano de Gironcoli, Thierry Deutsch, John Kay Dewhurst, Igor Di Marco, Claudia Draxl, M. Du ak, Olle Eriksson, José A Flores-Livas, Kevin F Garrity, Luigi Genovese, Paolo Giannozzi, Matteo Giantomassi, Stefan Goedecker, Xavier Gonze, O. Granas, E K U Gross, Andris Gulans, François Gygi, D R Hamann, Phil J Hasnip, N A W Holzwarth, D. Iu an, Dominik B Jochym, François Jollet, Daniel Jones, Georg Kresse, Klaus Koepernik, E. Kucukbenli, Yaroslav O Kvashnin, Inka L M Locht, Sven Lubeck, Martijn Marsman, Nicola Marzari, Ulrike Nitzsche, L. Nordstrom, Taisuke Ozaki, Lorenzo Paulatto, Chris J Pickard, Ward Poelmans, Matt I J Probert, Keith Refson, Manuel Richter, G.-M. Rignanese, Santanu Saha, Matthias Scheffler, Martin Schlipf, Karlheinz Schwarz, Sangeeta Sharma, Francesca Tavazza, P. Thunstrom, Alexandre Tkatchenko, Marc Torrent, David Vanderbilt, Michiel J van Setten, Veronique Van Speybroeck, John M Wills, Jonathan R Yates, G.-X. Zhang, and Stefaan Cottenier. Reproducibility in density functional theory calculations of solids. *Science*, 351(6280):aad3000–aad3000, mar 2016.

[15] Michael Frisch, Martin Head-Gordon, and John Pople. Direct analytic scf second derivatives and electric field properties. *Chem. Phys.*, 141(2-3):189 – 196, 1990.

[16] Roberto Dovesi, Fabien Pascale, Bartolomeo Civalleri, Klaus Doll, Nicholas M. Harrison, Ian Bush, Philippe D'Arco, Yves Noël, Michel Rérat, Philippe Carbonnière, Mauro Causà, Simone Salustro, Valentina Lacivita, Bernard Kirtman, Anna Maria Ferrari, Francesco Silvio Gentile, Jacopo Baima, Mauro Ferrero, Raffaella Demichelis, and Marco De La Pierre. The CRYSTAL code, 1976-2020 and beyond, a long story. *The Journal of chemical physics*, 152(20):204111, 2020.

[17] B Delley. An all-electron numerical method for solving the local density functional for polyatomic molecules. *J. Chem. Phys.*, 92(1):508, 1990.

[18] Y. Ye, Z. Song, S. Zhou, Y. Liu, Q. Shu, B. Wang, W. Liu, F. Qiao, and L. Wang. swnemo_v4.0: an ocean model based on nemo4 for the new-generation sunway supercomputer. *Geoscientific Model Development*, 15(14):5739–5756, 2022.

[19] Qianchao Zhu, Hao Luo, Chao Yang, Mingshuo Ding, Wanwang Yin, and Xinhui Yuan. Enabling and scaling the hpcg benchmark on the newest generation sunway supercomputer with 42 million heterogeneous cores. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[20] Yuxuan Li, Xiaohui Duan, Lin Gan, Wubing Wan, Yuhu Chen, Kai Xu, Jinzhe Yang, Weiguo Liu, Wei Xue, Haohuan Fu, and Guangwen Yang. Enabling large-scale simulation of cam on the sunway taihulight supercomputer. *IEEE Transactions on Computers*, 71(4):824–837, 2022.

[21] V.I. Lebedev. Quadratures on a sphere. *USSR Computational Mathematics and Mathematical Physics*, 16(2):10–24, 1976.

[22] V. I. Lebedev and Dimitri N. Laikov. A quadrature formula for the sphere of the 131st algebraic order of accuracy. *Doklady Mathematics*, 59:477–481, 1999.

[23] V. Havu, V. Blum, P. Havu, and M. Scheffler. Efficient O (N) integration for all-electron electronic structure calculation using numeric basis functions. *Journal of Computational Physics*, 228(22):8367–8379, 2009.

[24] Mark Lubin Mikhail Brinskiy. An introduction to mpi-3 shared memory programming, July 2018.

[25] Michael E Wolf and Monica S Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel & Distributed Systems*, 2(04):452–471, 1991.

[26] Michael E Wolf and Monica S Lam. A data locality optimizing algorithm. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, pages 30–44, 1991.

[27] Antoine Fraboulet, Karen Kodary, and Anne Mignotte. Loop fusion for memory space optimization. In *Proceedings of the 14th international symposium on Systems synthesis*, pages 95–100, 2001.

[28] Kathryn S McKinley, Steve Carr, and Chau-Wen Tseng. Improving data locality with loop transformations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 18(4):424–453, 1996.

[29] G Gao, R Olsen, V Sarkar, and R Thekkath. 18 collective loop fusion for array contraction. In *Languages and Compilers for Parallel Computing: 5th International Workshop, New Haven, Connecticut, USA, August 3-5, 1992. Proceedings*, volume 757, page 281. Springer Science & Business Media, 1993.

[30] Amy W Lim, Shih-Wei Liao, and Monica S Lam. Blocking and array contraction across arbitrarily nested loops using affine partitioning. In *Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, pages 103–112, 2001.

[31] Joseph CH Park and Mike Schlansker. *On predicated execution*. Hewlett-Packard Laboratories Palo Alto, California, 1991.

[32] John R Allen, Ken Kennedy, Carrie Porterfield, and Joe Warren. Conversion of control dependence to data dependence. In *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 177–189, 1983.

[33] Tadej Ciglarič, Rok Češnovar, and Erik Štrumbelj. Automated opencl gpu kernel fusion for stan math. In *Proceedings of the International Workshop on OpenCL*, pages 1–6, 2020.

[34] Jiri Filipovic and Siegfried Benkner. Opencl kernel fusion for gpu, xeon phi and cpu. In *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 98–105. IEEE, 2015.

[35] Mingchuan Wu, Ying Liu, Huimin Cui, Qingfu Wei, Quanfeng Li, Limin Li, Fang Lv, Jingling Xue, and Xiaobing Feng. Bandwidth-aware loop tiling for dma-supported scratchpad memory. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, PACT '20, page 97–109, New York, NY, USA, 2020. Association for Computing Machinery.

[36] AMD. Rocm, 2016.

[37] Honghui Shang, Fang Li, Yunquan Zhang, Ying Liu, Libo Zhang, Mingchuan Wu, Yangjun Wu, Di Wei, Huimin Cui, Xin Liu, Fei Wang, Yuxi Ye, Yingxiang Gao, Shuang Ni, Xin Chen, and Dexun Chen. Accelerating all-electron ab initio simulation of raman spectra for biological systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[38] Zhikun Wu, Honghui Shang, Yangjun Wu, Zhongcheng Zhang, Ying Liu, Yuyang Zhang, Yucheng Ouyang, Huimin Cui, and Xiaobing Feng. Opencl-accelerated first-principles calculations of all-electron quantum perturbations on hpc resources. *Frontiers in Chemistry*, 11, 2023.

Appendix: Artifact Description/Artifact Evaluation

ARTIFACT DOI

https://doi.org/10.5281/zenodo.8085489

ARTIFACT IDENTIFICATION

Contributions of the paper and how they will be justified with our artifacts.

This work re-designs the quantum perturbation calculations within FHI-aims (a all-electron full-potential framework), with a portable and scalable OpenCL implementation for the heterogeneous many-core supercomputer, with following major innovations:

(1) A portable quantum perturbation implementation has been developed using OpenCL, i.e., a cross-platform unified programming framework, to enable efficient all-electron all-potential simulations across various supercomputers.
(2) Major scaling limitations in quantum perturbation has been removed, by a locality-enhancing task mapping strategy. It enables neighbouring atoms to be simulated in the same MPI process, leading to significantly reduced per-process memory consumption and more efficient memory accesses.
(3) Efficient collective communication has been enabled, by introducing a packed hierarchical communication scheme, which reduces the total number of collective communications by packing several of them together, and accelerates each communication by applying a hierarchical approach for inter-node and intra-node data synthesizing.
(4) A set of performance-portable OpenCL optimizations have been implemented to improve the efficiencies of compute units and memory sub-system for various heterogeneous processors, by reordering OpenCL kernel invocations, memory accesses and arithmetic operations.
(6) We perform a full *ab initio* simulation of a real material containing 3006 atoms and compare it with experimental results. We have evaluated our OpenCL-accelerated quantum perturbation simulation on two advanced supercomputers, and results show that our implementation exhibits remarkably performance on various material systems, enabling the system scale to 200,000 atoms with all-electron precision.

Our artifacts include an optimized version, and several sub-optimized versions with certain optimization excluded. By running the optimized version, contributions (1) and (5) could be justified directly. By comparing results of the optimized version with certain sub-optimized version, contributions (2)-(4) could be testified with corresponding speedup, respectively.

Software architecture of our artifacts.

All versions of artifacts provided leverage the same software architecture (also the same as the open source FHI-aims-ocl-sc23) as follows. The entry point and function name of each subprocedure can be found at the README file of the project.

(1) DFT phase. It serves to provide data for the DFPT phase, with scf_solver as its entry point.
(2) DFPT phase. It completes quantum perturbation calculation by iteratively calling following functions, until self-consistency has been reached. In our research, we only focus on the polar_reduce_memory version of DFPT, with cpscf_solver_polar_reduce_memory as its entry point.
   a. DM. It calculates the response of the density matrix.
   b. Sumup (OpenCL-accelerated). It calculates the response of the electronic density, launching 2 OpenCL kernels.
   c. Rho (OpenCL-accelerated). It calculates the response of the total electrostatic potential, launching 1 OpenCL kernel.
   d. H (OpenCL-accelerated). It calculates the response of the Kohn-Sham Hamiltonian matrix, with as its entry point, launching 1 OpenCL kernel.

To what extent our artifacts contribute to reproductivity of experiments.

All results in our evaluation, could be reproduced by our provided artifacts, given enough computing nodes on the two HPC systems. However, speedup results could be reproduced even with a few computing nodes.

REPRODUCIBILITY OF EXPERIMENTS

All calculations and scalability tests were run with FHI-aims on two supercomputers, the new generation Sunway suptercomputer and an AMD-GPU-accelerated supercomputer. The experiment workflow includes the following three steps. Note that different versions of the code and different test cases (including input files) should be chosen for different evaluations.

(1) Build: Install and compile the libraries. Then "cd src", modify the path of the libraries in Makefile and "make scalapack.mpi -jN' to generate binary aims.191127.scalapack.mpi.x under "bin" directory.
(2) Execution: Run the binary on a cluster with specific input file. First, go to the the testcase directory which contains control file (control.in) and the geometry file (geometry.in). Then submit the calculation task to the job management system of the cluster.
(3) Data Processing: Use a specific python script to extract relevant information including the execution time of some subprocedures and so on from the output file, which is generated during execution.

The installation of the software is expected to take 30 minutes. The execution time of the binary ranges from 10 minutes to 4 hours according to different testcases. The extraction takes about 2 minutes.

The key result of this workflow is the output file generated during execution, which includes execution time of different parts, the value of some runtime variable corresponding to memory consumption and the result of the DFPT calculation.

The numerical results found in the article include execution time of some subprocedures, speedups between and scalability efficiency, which can be directly obtained, calculated from the above results.

ARTIFACT DEPENDENCIES REQUIREMENTS

(1) Hardware resources required and utilized.
  (a) Hardware requirements. The artifact needs multi-core x86-64 CPU and accelerators which supports OpenCL 2.0 including supports for atomic functions, constant kernel parameters and CL_MEM_USE_HOST_PTR. However, because the code is optimized for specific hardware, its performance may not be good. An AMD-GPU-accelerated cluster is recommended.
  (b) Hardware for the experiments. All experiments of the paper were run on two supercomputers, the new generation Sunway suptercomputer and an AMD-GPU-accelerated supercomputer with a special version of AMD GPU (between MI50 and MI60, with 64 computing units like mi60, but only 16GB of memory like MI50.).
(2) Operating systems. Linux.
(3) Software libraries. Some software libraries have been packed into the artifact as source code. However, the following libraries are still needed.
  (a) MPI Fortran, C and C++ compiler.
  (b) OpenCL driver, e.g. ROCM on AMD GPU.
  (c) Mathlibs providing LAPACK and SCALAPACK support, e.g. INTEL MKL on x86 and xMath on sunway.
(4) The datasets has been packed into the artifact. In particular, a dataset is a series of atomic types and coordinates, i.e., a geometry file. They are publicly available.

ARTIFACT INSTALLATION DEPLOYMENT PROCESS

Detailed guidelines are provided in the README of the artifact. Here's an simpler guideline.

(a) Prepare dependencies
  (i) Install or load mathlibs to provide LAPACK and SCALAPACK support, e.g. INTEL MKL on x86 and xMath on sunway.
  (ii) Install or load a OpenCL driver to provide OpenCL support, e.g. ROCM on AMD GPU and xMath on Sunway.
  (iii) Install or load a compiler supporting MPI and C+Frotran.
  (iv) Set PATH and LD_LIBRARY_PATH for the software and libraries above. We recommend you to package the environment variable design as an env.sh for convenience, so that you just need to source env.sh later. We provide the examples under testcases-SC/envs-example/example-name/env.sh of the artifact.
  (v) It takes about 1-60 minutes to load the dependencies, depending on whether the user's system provides relevant software.
(b) Build
  (i) Copy initial_cache.example.cmake as initial_cache.cmake. We provide the examples under testcases-SC/envs-example/example-name/initial_cache.cmake. CMAKE_Fortran_FLAGS, CMAKE_C_FLAGS options and so on in file based on the your environment. We provide the examples under testcases-SC/envs-example/example-name/initial_cache.cmake.
  (ii) Then compile the code after source env.sh.
  (iii) cmake -C ./initial_cache.cmake -S . -B build -DCMAKE_PREFIX_PATH="" -DOpenCL_FOUND=True -DOpenCL_LIBRARY=YOUR_PATH/opencl/lib/ -DOpenCL_INCLUDE_DIR=YOUR_PATH/opencl/include/
  (iv) cmake –build build -j4
  (v) It takes about 5-30 minutes to compile, depending on the peformance of the CPU and the number of CPU cores used for compilation. After successful compilation, you will be able to see build/aims.191127.scalapack.mpi.x.
(c) Prepare datesets. Four kinds of test cases under testcases-SC directory of the artifact, among which Polypeptide is used for basic test because it is simple, time-consuming and representative. All datesets are directly provided.
(d) Run the code. Enter one of the dateset directories and then execute the aims.191127.scalapack.mpi.x. We provide a example script sbatch.sh to run the code. The README of the artifact provides more detail of the experiments. It takes about 4 minutes to 4 hours dependending on the dateset. We have provided small cases to run.