NDFT: Accelerating Density Functional Theory Calculations via Hardware/Software Co-Design on Near-Data Computing System

Qingcai Jiang[1]*, Buxin Tu[1]*, Xiaoyu Hao[1], Junshi Chen[1],[2] and Hong An[1],[2],†
[1]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[2]Laoshan Laboratory, Qingdao, China

***Abstract*—Linear-response time-dependent Density Functional Theory (LR-TDDFT) is a widely used method for accurately predicting the excited-state properties of physical systems. Previous works have attempted to accelerate LR-TDDFT using heterogeneous systems such as GPUs, FPGAs, and the Sunway architecture. However, a major drawback of these approaches is the constant data movement between host memory and the memory of the heterogeneous systems, which results in substantial *data movement overhead*. Moreover, these works focus primarily on optimizing the compute-intensive portions of LR-TDDFT, despite the fact that the calculation steps are fundamentally *memory-bound*.**

To address these challenges, we propose NDFT, a Near-Data Density Functional Theory framework. Specifically, we design a novel task partitioning and scheduling mechanism to offload each part of LR-TDDFT to the most suitable computing units within a CPU-NDP system. Additionally, we implement a hardware/software co-optimization of a critical kernel in LR-TDDFT to further enhance performance on the CPU-NDP system. Our results show that NDFT achieves performance improvements of 5.2x and 2.5x over CPU and GPU baselines, respectively, on a large physical system.

***Index Terms*—Near-Data Processing, Density Functional Theory, Workload Scheduling, Hardware/Software Co-Design**

I. Introduction

First-principle calculations, also known as *ab initio* methods, predict the properties of materials by solving the electronic motion equations derived from the fundamental principles of quantum mechanics. Among these methods, density functional theory (DFT) is widely used to determine the ground-state properties of systems by focusing on electron density rather than wavefunctions. Building on DFT, **linear response time-dependent density functional theory (LR-TDDFT)** is particularly notable for its ability to accurately describe the excited states of complex systems. LR-TDDFT is extensively applied in material analysis, condensed matter physics, and quantum chemistry. However, it often encounters significant computational challenges, including high execution time, memory complexity, and frequent data accesses, which can easily lead to memory bottlenecks. Given the crucial role of LR-TDDFT in scientific research and the computational challenges it presents, exploring optimization techniques for this method has become a prominent topic in the field of high-performance computing.

*Equal contribution, listed alphabetically

†Corresponding Author

Recent studies have tried to accelerate LR-TDDFT calculations using heterogeneous systems such as graphics processing units (GPUs) [20] and the Sunway architecture [22]. These architectures leverage parallel processing capabilities to execute a large number of computations simultaneously, significantly speeding up the *compute-intensive* aspects of the software. However, executing LR-TDDFT on these systems requires frequent data transfers between main memory and local memory. This overhead from this data movement can negate the performance gains achieved by heterogeneous architectures, leading to a phenomenon known as the data movement bottleneck.

Near-data processing (NDP) is a promising solution to this bottleneck in modern computing systems. By positioning computing units closer to the main memory, the NDP mechanism reduces the overhead associated with data access [28]. With recent advancements in memory technology [1]–[4], [6], [9]–[11], [13] and the growing demand from data-intensive applications [6], [7], the near-memory computing paradigm has become increasingly feasible [14], [17].

However, effectively harnessing the potential of NDP systems to accelerate LR-TDDFT calculations presents several key challenges. First, NDP systems typically operate alongside CPU systems, forming a CPU-NDP system. For LR-TDDFT calculations, which involve both compute-intensive and memory-intensive kernels, such systems require a precise scheduling and partitioning scheme to ensure efficient execution. Second, in the pseudopotential computation of LR-TDDFT (see Section III-B), each process stores its copy of the data in memory, leading to significant redundancy given the large number of NDP cores and corresponding processes. Therefore, we need to rethink the data organization of pseudopotential in the NDP scenario to mitigate this issue.

To address these challenges, **our goal** in this work is to design a partitioning and scheduling mechanism specifically tailored for DFT calculations to achieve better performance, along with a hardware/software co-design solution that optimizes the data organization and memory access in the pseudopotential calculations.

This paper makes the following key contributions:

- We conduct a detailed study of the computation and memory characteristics in the LR-TDDFT application, identifying the key performance bottlenecks in its execution and determining LR-TDDFT as a promising application for NDP systems.

- We design NDFT, a near-data density functional theory calculation framework, featuring a novel workload partitioning and scheduling mechanism, along with a hardware/software co-design based on a CPU-NDP heterogeneous architecture.
- We evaluate NDFT across physical systems of varying sizes and compare it with CPU and GPU baselines. The results show that NDFT achieves a performance speedup of 5.2x and 2.5x over CPU and GPU baselines on a large physical system.

II. Background

A. LR-TDDFT

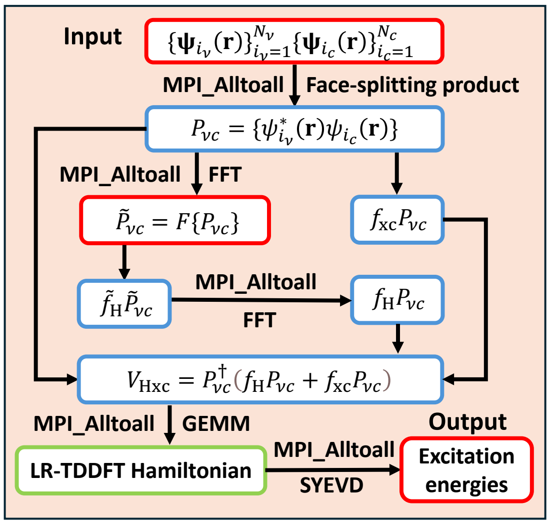
Input
$\{\psi_{i_v}(\mathbf{r})\}_{i_v=1}^{N_v}\{\psi_{i_c}(\mathbf{r})\}_{i_c=1}^{N_c}$
MPI_Alltoall
Face-splitting product
$P_{vc}=\{\psi_{i_v}^*(\mathbf{r})\psi_{i_c}(\mathbf{r})\}$
MPI_Alltoall
FFT
$\tilde{P}_{vc}=F\{P_{vc}\}$
$f_{xc}P_{vc}$
$\tilde{f}_H\tilde{P}_{vc}$
MPI_Alltoall
FFT
$f_H P_{vc}$
$V_{Hxc}=P_{vc}^\dagger(f_H P_{vc}+f_{xc}P_{vc})$
MPI_Alltoall
GEMM
Output
LR-TDDFT Hamiltonian
MPI_Alltoall
SYEVD
Excitation energies

Fig. 1. Computation flowchart of LR-TDDFT. $\psi_{i_v}(\mathbf{r})$ and $\psi_{i_c}(\mathbf{r})$ stand for the valence and conduction orbitals in real space $(\{\mathbf{r_i}\}_{i=1}^{N_r}$.

LR-TDDFT [30] is a widely used computational method for studying the excited-state properties of molecules and solids in condensed matter physics, computational chemistry, and materials science. In recent years, advancements in both computational hardware and numerical algorithms have significantly improved the scalability and performance of LR-TDDFT, enabling its application to large and complex physical systems [26], [33]. As shown in Figure 1, the computation of LR-TDDFT involves several major operations: (1) fast Fourier transform (**FFT**), (2) face-splitting product, i.e., **point-to-point multiplication** between two vectors, (3) general matrix multiplications (**GEMM**), (4) **MPI_Alltoall** between all processes to transpose data, and (5) diagonalization (**SYEVD**).

Due to its scientific significance, numerous prior works have focused on accelerating the *computation-intensive* aspects of LR-TDDFT using heterogeneous architectures, such as GPUs [20], [34], the Sunway supercomputer [21], [22], and field-programmable gate arrays (FPGAs) [29]. However, accelerating LR-TDDFT on these heterogeneous architectures presents two major problems: (1) These approaches primarily target the *compute-intensive* components, neglecting the *memory-intensive* parts, which constitute a significant portion of LR-TDDFT's overall execution time; (2) Executing kernels on heterogeneous architectures requires frequent data transfers between the host device's memory and the memory of the heterogeneous device, resulting in substantial bottlenecks.

B. NDP Architecture

The fundamental idea of NDP is to place computing logic close to where data resides, offering significant potential for improving the performance of *memory-intensive applications*. NDP can be implemented by integrating processing logic within DRAM banks [1], [13], in the logic layer of 3D-stacked memory [2], [6], [10], or within the buffer chip of DDR-DIMMs [3], [4], [9], [11]. As shown in Figure 2, in such architectures, the bottom-most layer, known as the logic layer, can be equipped with **wimpy cores** that perform simple, parallelizable tasks close to the data. These cores leverage the high internal bandwidth and low data movement latency of the memory stacks to improve efficiency. Typical 3D-stacked NDP systems, such as Hybrid Memory Cube (HMC) [8] and High Bandwidth Memory (HBM) [18], [19], consist of multiple memory stacks interconnected in a memory network. By bringing computation closer to the data, NDP systems take advantage of the high bandwidth within the memory structure and reduce data access latency.

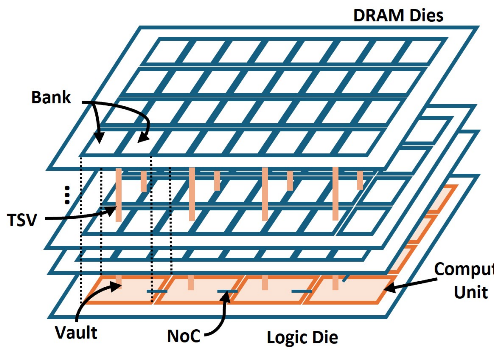

Fig. 2. An example of 3D-stacked memory.

C. CPU-NDP Hybrid Execution

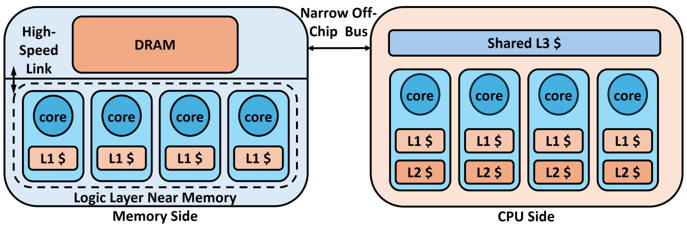

Fig. 3. A high-level CPU-NDP architecture.

In a typical configuration, NDP cores operate alongside CPU cores within a computing system, forming what we refer to as the CPU-NDP architecture in this paper, as illustrated in Figure 3. This architecture includes two types of computing units: the CPU, which excels at complex, compute-intensive tasks but suffers from high-latency and low-bandwidth memory access, and the NDP cores, which are more numerous and benefit from lower-latency, higher-bandwidth memory access, but feature simpler logic. Thus, CPU cores are suitable for *compute-intensive* workloads, while NDP cores handle *memory-intensive* tasks more efficiently. In such systems, a key challenge is determining which code regions of a program should be offloaded to NDP for optimal performance. Several prior works have attempted to solve this problem [12], [23], [35]. However, none of these approaches have been applied to exploit the performance potential of DFT applications in CPU-NDP systems.

III. Motivational Study

In this section, we conduct a detailed study on several key characteristics of LR-TDDFT, aiming to identify the key bottlenecks in CPU-NDP hybrid execution and demonstrate its potential for performance improvement on CPU-NDP heterogeneous architecture.

A. LR-TDDFT kernel characteristics

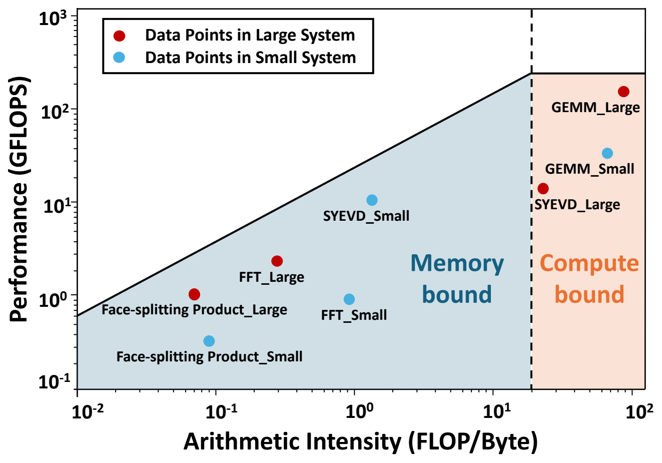

Fig. 4. Roofline model analysis of LR-TDDFT kernels across two different system sizes.

We analyze the performance characteristics of representative kernels in LR-TDDFT calculations on the Roofline model, as introduced in Section II-A, using two different physical system sizes: Si_64 (small system) and Si_1024 (large system). The Roofline model [36] integrates floating-point operations (FLOPs) and arithmetic intensity (AI) to assess the computational and memory characteristics of a given kernel. We use Intel VTune Profiler [16] for the analysis, with system configurations described in Section V. From the results shown in Figure 4, we make three key observations: (1) LR-TDDFT calculations are fundamentally **memory-bound**: most computing kernels in LR-TDDFT fall within the memory-bound region, and the MPI_Alltoall operations used in LR-TDDFT introduce a large amount of irregular memory accesses [22]. (2) Different kernels in LR-TDDFT calculations exhibit distinct compute-bound or memory-bound characteristics: FFT kernels are memory-bound, while GEMM kernels are compute-bound across different physical system sizes. (3) The compute- or memory-bound nature of LR-TDDFT kernels depends on the system size: for large physical systems, the SYEVD operation is compute-bound, whereas, for small systems, SYEVD is memory-bound; GEMM becomes more compute-bound as the system size increases.

Based on these observations, we conclude that LR-TDDFT is well-suited for CPU-NDP acceleration due to its diverse kernel characteristics. To fully leverage the strengths of both CPU and NDP cores, the system requires an intelligent scheduling mechanism that can dynamically allocate tasks to the appropriate computing units based on their characteristics. For example, memory-bound kernels such as FFT could benefit from execution on NDP units optimized for memory access, while compute-bound kernels like GEMM and SYEVD (in large systems) would perform better on CPU cores, which excel at handling complex computations. However, designing a precise task scheduling framework for LR-TDDFT in a CPU-NDP system is not trivial, as it faces two main challenges: (1) It must account not only for the suitability between kernels and computing units but also for the *data movement overhead* between the two types of computing units. The data required by the kernels must also be scheduled accordingly to the appropriate execution units. (2) The selection of offloading granularity, ranging from fine-grained instructions to basic blocks or entire kernels, requires careful consideration.

B. Analysis of pseudopotential calculations

Pseudopotentials are commonly used in DFT calculations to simplify the treatment of inner electrons, as the outer valence electrons primarily determine an atom's physical and chemical properties. In LR-TDDFT, pseudopotentials simulate the influence of the atomic nucleus and inner electrons. During this procedure, each wavefunction is updated based on the pseudopotential information associated with each atom. In a multi-process execution, each process must maintain a complete copy of the pseudopotential information in its memory space. However, in systems with numerous cores, such as NDP architectures [12], [32], these redundant data copies can lead to significant memory inefficiency.

TABLE I
Memory Footprint of Pseudopotentials in CPU and NDP Systems

Memory Footprint	size (GB)	Percentage (%)
NDP in Small system	4.43	6.92
CPU in Small system	1.84	2.88
NDP in Large system	35.3	55.15
CPU in Large system	13.8	21.56

We profile the memory footprint during LR-TDDFT calculations on typical NDP and CPU systems using two physical systems of different scales, where the NDP system and the CPU system are isolated. The configuration is detailed in Section V. As shown in Table I, our observations are as follows: (1) In the small physical system, the memory footprint of pseudopotentials in NDP execution occupies 6.92% of the system memory, which is 140.2% higher than that in CPU execution; (2) In the large physical system, the memory footprint reaches 55.15%, representing an increase of 155.7% compared to CPU execution. We conclude that the large number of NDP cores leads to an increase in parallel processes, resulting in pseudopotentials consuming a significant amount of memory space. Moreover, in more complex physical systems, the traditional per-process pseudopotential approach causes an out-of-memory (OOM) problem. Therefore, optimizing the data structure of pseudopotential information in memory is critical to reducing the substantial memory footprint in the CPU-NDP architecture.

IV. NDFT Design

In this section, we elaborate on the design of NDFT, aiming to maximize the performance potential of the CPU-NDP system for DFT calculations. This includes a novel workload partitioning and scheduling mechanism, which is introduced in Section IV-A, as well as a hardware/software co-design strategy to better orchestrate an important computational scheme in LR-TDDFT calculations, which is covered in Section IV-B and IV-C, respectively.

A. Workload Partition and Scheduling

1) Offload Granularity: To provide a better offloading granularity for LR-TDDFT, it is crucial to understand the costs associated with scheduling contiguous code segments to different computing units. We identify two primary sources of overhead: (1) The *data transfer overhead*, which arises from maintaining the consistency of data across two computing units. This overhead is proportional to the amount of data that needs to be transferred. (2) The *context switch overhead*, which involves synchronizing the context (e.g., register values) between threads on different units, is a constant time cost.

In this work, we choose *function-level* offloading granularity based on two key observations: (1) Dividing the program into too many small segments introduces significant offloading overhead. (2) Most functions in LR-TDDFT exhibit consistent compute/memory characteristics throughout their execution. For example, GEMM and FFT maintain the same computing and memory access patterns across the entire function.

2) Cost-Aware Offloading mechanism: To fully leverage the performance benefits of a heterogeneous CPU-NDP system, we design a kernel offloading mechanism in NDFT based on a static code analyzer (SCA) [15], [25]. The SCA analyzes the intrinsic properties of a given code segment, such as estimated execution time, memory access patterns, and instruction dependencies. These insights help inform decisions on whether each code segment is compute- or memory-bound, and provide an estimation of the data transfer overhead when scheduling different code segments across CPU and NDP units. In this work, we leverage the SCA to profile the compute and memory intensity of each function and use this information, along with a cost model that accounts for the overhead of scheduling between CPU and NDP cores, to formulate a cost-aware offloading mechanism.

$$\text{Scheduling Overhead} = \sum_{i \in \text{NDP}} \sum_{j \in \text{CPU}} (DT(i,j) + CXT) \quad (1)$$

By evaluating the suitability of each function for execution on either CPU or NDP cores, the SCA allows us to determine whether it is more efficient to offload the function to the NDP or retain it on the CPU. The overall scheduling strategy then incorporates the scheduling overhead, as shown in (1), which is defined as the sum of data transfer (DT) overhead and context switch (CXT) overhead between the CPU and NDP cores. This approach enables a cost-aware offloading mechanism that balances the performance gains from offloading with the associated overheads, optimizing the performance of the heterogeneous system.

B. Optimization of Pseudopotential Algorithm

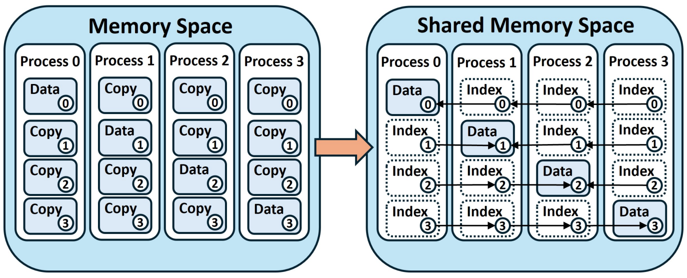

Fig. 5. The data structure optimization to eliminate the data redundancy of pseudopotential.

To reduce the memory footprint in LR-TDDFT and enhance the scalability for large physical systems, we propose an optimization based on an improved pseudopotential data structure. As shown in Figure 5, each process stores **only a portion of the pseudopotentials**, along with indices pointing to the pseudopotentials in other processes, accessing the shared data via these indices. Compared to the original approach, our optimization enables seamless data sharing between processes, significantly reducing the memory footprint and making LR-TDDFT more suitable for our CPU-NDP heterogeneous design.

In our optimization, we reduce the memory overhead of each process by addressing the complexity of each atom's pseudopotential data, which includes arrays of integers and double-precision floating-point matrices, through the design of a **shared block** to efficiently manage and share this information. This data structure facilitates unified management of pseudopotential data and simplifies data sharing. As shown in Algorithm 1, the highlighted gray section represents the main optimization of the algorithm: (1) In lines 7-9, the algorithm reorganizes the pseudopotential information of each atom into a shared block and writes it into shared memory. (2) In line 13, the algorithm retrieves the addresses of pseudopotential data in shared memory based on the distribution of the shared block. (3) In line 19, the algorithm accesses the data from shared memory for wavefunction updating.

Although this approach significantly reduces memory usage, it introduces the need for inter-process data communication, leading to some performance overhead. We design a novel shared memory hardware structure to mitigate this overhead, detailed in Section IV-C.

C. Shared Memory Design for Pseudopotential

As we improve the data structure of the pseudopotential in each process, data communication is needed to synchronize the pseudopotential. To mitigate this overhead, we design shared memory based on scratchpad memory (SPM) for processes to share the pseudopotential information within the same stack, and a hierarchical communication scheme to manage the communication between the shared memory of different stacks.

Algorithm 1 Pseudopotential Algorithm in LR-TDDFT, with the gray background highlighting our optimizations.

```
Input: Pseudopotential
Output: Wavefunction updated using pseudopotential
 1: for each local atom do
 2:     calculate pseudopotential
 3: end for
 4: for each global atom do
 5:     if this process holds the information then
 6:         for each atom's pseudopotential do
 7:             calculate the data size
 8:             allocate a continuous space in shared memory
 9:             write local pseudopotential information as
                a block into shared memory
10:         end for
11:     else
12:         for each shared memory block do
13:             obtain the address of the shared block
14:         end for
15:     end if
16: end for
17: for each local wavefunction do
18:     for each pseudopotential of every atom do
19:         access pseudopotential via shared block address
20:         apply pseudopotential to the wavefunction
21:     end for
22: end for
```

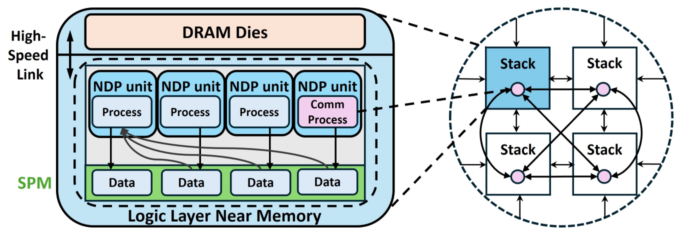

Fig. 6. Shared Memory Hardware Design and Hierarchical Communication Scheme. Green color highlights the shared memory architecture.

SPM-based Shared Memory. The left part of Figure 6 shows our shared memory hardware design. We design an SPM in the logic layer of each memory stack to enable the shared memory functionality. The processes within the same stack can access local pseudopotentials and also access the pseudopotentials of other processes in SPM-based shared memory. The high read/write speed and low access cost of SPM [5] allow these processes to efficiently access pseudopotentials, avoiding the expensive memory access in DRAM.

In addition, we develop a set of communication APIs to support shared memory functionality. As shown in Table II, these APIs implement multiple synchronization semantics, providing the flexibility and comprehensiveness needed for essential communication primitives. For example, `NDFT_Alloc_Shared` enables NDP units to allocate memory for pseudopotential data in `sharedBL`. The function's parameters include the pseudopotential data (`pseu_info`) and the NDP unit ID (`stID* id`), and it returns a `sharedBL`. The definition of `sharedBL` corresponds to the shared block introduced in Section IV-B. We also design primitives such as `NDFT_Read` and `NDFT_Write` to facilitate reading and writing pseudopotentials within the shared block. By managing pseudopotentials with SPM-based shared memory, our design reduces communication overhead between NDP units within the same stack. To further enhance scalability, we implement a hierarchical communication scheme.

TABLE II
PROGRAMMING INTERFACES OF NDFT

NDFT's Message Programming Interface(i.e., API)
sharedBL NDFT_Alloc_Shared (Var* pseu_info, stID* id);
void NDFT_Read (sharedBL* data, void* address, int length);
void NDFT_Write (sharedBL* data, void* address, int length);
void NDFT_Read_Remote (sharedBL* data, void* address, int length, stID* source_id, stID* dest_id);
void NDFT_Write_Remote (sharedBL* data, void* address, int length, stID* source_id, stID* dest_id);
void NDFT_Broadcast (sharedBL* data);
......

Hierarchical Communication Scheme. For pseudopotential data communication between different stacks, we designate one NDP unit per stack as a communication arbiter, which runs a *comm process* (indicated in pink in Figure 6) to manage inter-stack communication. When a process requests data from another stack via `NDFT_Read_Remote`, it submits the request to the *comm process* within its own stack. The `dest_id` parameter specifies the stack ID from which the data is needed. After exchanging data with the communication process of the destination stack, the local *communication process* writes the data into shared memory and returns the index to the requesting process. This hierarchical design acts as a filter, maximizing intra-stack communication and only transmitting essential data across stacks, thereby reducing overall communication overhead.

V. EVALUATION METHODOLOGY

System model. We implement NDFT in zsim [31], a fast and accurate Pin-based simulator, and use Ramulator [24] as the memory simulator. Table III summarizes the NDFT's CPU-NDP system configurations.

TABLE III
CPU-NDP SYSTEM CONFIGURATION

CPU	8 General purpose core;
	3 GHz, 4-way superscalar;
	32 KB L1I/D, 256 KB L2, 2 MB L3

NDP	8 NDP units per stack, 2GHz, in order;
	64 GB in total, 512 MB per unit;
	2 GHz, 2 cores per NDP unit, 32KB L1I/D
	Shared Memory: 16 KB per core, 256 KB per stack

Memory	HBM2, 4 × 4 stacks in mesh, 8 channels per stack; 128-bit bus width, 1000Mhz, 64 GB capacity

Baseline Architectures. We compare NDFT with two baselines: (1) CPU: 2 CPUs of Intel Xeon E5-2695@2.40 GHz, 12 cores per socket, with 64 GB DDR4 memory; (2) GPU: 2 GPUs of NVIDIA V100 in DGX-1 server.

Physical Systems. We use various crystal silicon systems with 16, 32, 64 (small system), 128, 256, 1024 (large system), and 2048 silicon atoms, denoted as Si_16, Si_32, Si_64, Si_128, Si_256, Si_1024, and Si_2048, respectively.

VI. Evaluation Result

A. *Performance Analysis on Execution Time*

Figure 7 shows the performance on CPU, GPU, and NDFT, along with the time distribution across different kernels in 2 different sizes of physical systems. The execution time breakdown includes the execution time of FFT, point-point multiplication, Global Communication (Global Comm) phase, SYEVD, etc. The NDFT design includes the additional *scheduling* overhead between CPU and NDP, as mentioned in Section IV-A. We compare NDFT with both GPU and CPU baselines, and obtain the following evaluation results:

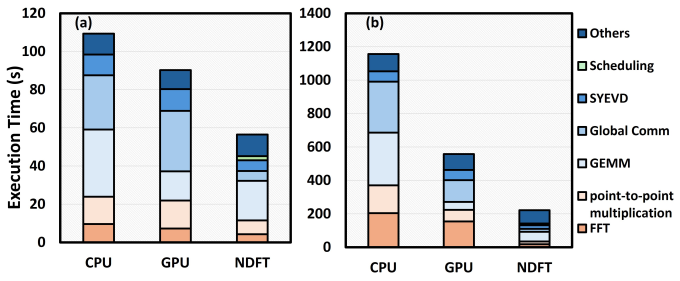

Fig. 7. Performance comparison between CPU, GPU, and NDFT design using a small physical system (a) and a large physical system (b).

Comparison with CPU. (1) Despite the scheduling overhead, NDFT achieves 1.9x and 5.2x speedups in small and large physical systems, respectively, thanks to our scheduling mechanism and software/hardware co-design. (2) The performance of memory-intensive kernels is significantly improved; for example, FFT achieves an 11.2x speedup in the large system, while the Face-splitting Product achieves a 1.99x speedup in the small system.

Comparison with GPU. (1) NDFT achieves 1.6x and 2.5x speedups compared to the GPU baseline in small and large physical systems, respectively. (2) While compute-bound kernels like GEMM on the GPU outperform those on NDFT by 35.9% and 22.2% in the two physical systems, respectively, memory-bound kernels show significant improvements with NDFT, achieving 2.1x and 5.2x speedups in the small and large physical systems, respectively.

Other Discussion. (1) The scheduling overhead accounts for only 3.8% and 4.9% in small and large physical systems, respectively, demonstrating that NDFT's scheduling strategy carefully keeps this overhead minimal while improving kernel performance. (2) We study the memory footprint of pseudopotential information in NDFT. We observe that, in the large physical system, NDFT reduces memory footprint by 57.8% compared to NDP in Table I, bringing it close to CPU execution (1.08x). Global Comm only increases by 3.2% over the GPU baseline, showing that NDFT's hardware/software co-design resolves the OOM issue in NDP systems with low communication overhead.

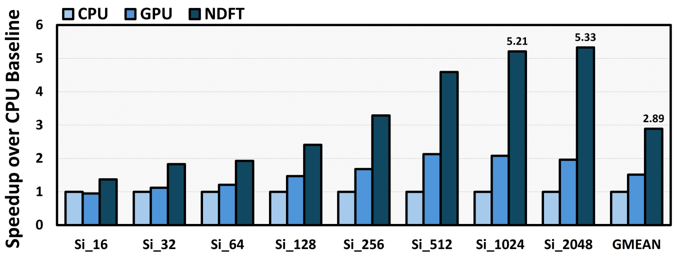

Fig. 8. Speedup provided by NDFT, GPU over CPU baseline in several physical system scales.

B. *Scalability Analysis*

To explore NDFT's performance across different physical system scales, we conduct a study ranging from Si_16 to Si_2048. Figure 8 demonstrates that as the size of the physical system increases, NDFT's performance advantage becomes more pronounced (up to 5.33x speedup in Si_2048). We conclude that NDFT improves the performance of LR-TDDFT calculations in most cases, showing NDFT's great potential for addressing large-scale computational problems in the field of high-performance computing.

VII. Related Work

To our knowledge, NDFT is the first work to accelerate DFT calculations through optimizations of a general NDP architecture. A prior work [27] proposes a heterogeneous approach focused on alleviating data movement bottlenecks in quantum chemistry simulations based on *ab initio* methods. That work analyzes two specific components—FFT and time-consuming loops—proposing a custom hardware design tailored to these kernels. In contrast, NDFT emphasizes task scheduling and hardware-software co-design, offering a more comprehensive solution within general-purpose NDP-CPU heterogeneous systems, thereby providing greater applicability and scalability.

VIII. Conclusion

In this work, we propose NDFT, a Near-Data Density Functional Theory framework to address performance bottlenecks in LR-TDDFT calculations on CPU-NDP heterogeneous systems. NDFT involves a task partitioning and scheduling mechanism that efficiently assigns memory- and compute-bound kernels, minimizing data movement and fully leveraging CPU-NDP heterogeneous architectures, and a hardware-software co-design optimizing memory access for pseudopotential calculations, solving OOM issues in NDP systems. Evaluations show speedups of 5.2x and 2.5x over CPU and GPU baselines in a large physical system, as well as NDFT's scalability across different system sizes.

Acknowledgment

This work is partly supported by Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDB0500102). The computing resources are financially supported by Laoshan Laboratory (LSKJ202300305).

References

[1] S. Aga, N. Jayasena, and M. Ignatowski. Co-ml: a case for co llaborative ml acceleration using near-data processing. In *Proceedings of the International Symposium on Memory Systems*, pages 506–517, 2019.

[2] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 105–117, 2015.

[3] B. Asgari, R. Hadidi, J. Cao, S.-K. Lim, H. Kim, et al. Fafnir: Accelerating sparse gathering by using efficient near-memory intelligent reduction. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 908–920. IEEE, 2021.

[4] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim. Chameleon: Versatile and practical near-dram acceleration architecture for large memory systems. In *2016 49th annual IEEE/ACM international symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[5] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad memory: design alternative for cache on-chip memory in embedded systems. In *Proceedings of the tenth international symposium on Hardware/software codesign*, pages 73–78, 2002.

[6] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, et al. Google workloads for consumer devices: Mitigating data movement bottlenecks. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 316–331, 2018.

[7] C. P. Chen and C.-Y. Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information sciences*, 275:314–347, 2014.

[8] H. M. C. Consortium. Hmc specification 2.0, 2014.

[9] G. Dai, Z. Zhu, T. Fu, C. Wei, B. Wang, X. Li, Y. Xie, H. Yang, and Y. Wang. Dimmining: pruning-efficient and parallel graph mining on near-memory-computing. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 130–145, 2022.

[10] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim. Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 283–295. IEEE, 2015.

[11] S. Feng, X. He, K.-Y. Chen, L. Ke, X. Zhang, D. Blaauw, T. Mudge, and R. Dreslinski. Menda: A near-memory multi-way merge solution for sparse transposition and dataflows. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 245–258, 2022.

[12] N. M. Ghiasi, N. Vijaykumar, G. F. Oliveira, L. Orosa, I. Fernandez, M. Sadrosadati, K. Kanellopoulos, N. Hajinazar, J. G. Luna, and O. Mutlu. Alp: Alleviating cpu-memory data movement overheads in memory-centric systems. *IEEE Transactions on Emerging Topics in Computing*, 11(2):388–403, 2022.

[13] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie. ipim: Programmable in-memory image processing accelerator using near-bank architecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 804–817. IEEE, 2020.

[14] X. Hu, D. Stow, and Y. Xie. Die stacking is happening. *IEEE micro*, 38(1):22–28, 2018.

[15] Intel. Intel architecture code analyzer user's guide, 2017.

[16] Intel Corporation. *Intel VTune Profiler*, 2024. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html.

[17] J. Jeddeloh and B. Keeth. Hybrid memory cube new dram architecture increases density and performance. In *2012 symposium on VLSI technology (VLSIT)*, pages 87–88. IEEE, 2012.

[18] JEDEC. High Bandwidth Memory (HBM) DRAM. https://www.jedec.org/standards-documents/docs/jesd235a, 2021.

[19] JEDEC Solid State Technology Assn. JESD23-5D: High Bandwidth Memory (HBM) DRAM Standard, March 2021.

[20] W. Jia, L.-W. Wang, and L. Lin. Parallel transport time-dependent density functional theory calculations with hybrid functional on summit. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–23, 2019.

[21] Q. Jiang, Z. Cao, J. Chen, X. Qin, W. Hu, H. An, and J. Yang. Pwdft-sw: Extending the limit of plane-wave dft calculations to 16k atoms on the new sunway supercomputer. *arXiv preprint arXiv:2406.10765*, 2024.

[22] Q. Jiang, Z. Cao, X. Cui, L. Wan, X. Qin, H. Cao, H. An, J. Chen, J. Liu, W. Hu, et al. Extending the limit of lr-tddft on two different approaches: Numerical algorithms and new sunway heterogeneous supercomputer. *Parallel Computing*, 120:103085, 2024.

[23] Q. Jiang, S. Tan, J. Chen, and H. An. A^{3}pim: An automated, analytic and accurate processing-in-memory offloader. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[24] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer architecture letters*, 15(1):45–49, 2015.

[25] C. Lattner and V. Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *CGO*, 2004.

[26] J. Liu and W. Liang. Analytical hessian of electronic excited states in time-dependent density functional theory with tamm-dancoff approximation. *The Journal of chemical physics*, 135(1), 2011.

[27] Z. Liu, Z. Xie, W. Dong, M. Yuan, H. You, and D. Li. A heterogeneous processing-in-memory approach to accelerate quantum chemistry simulation. *Parallel Computing*, 116:103017, 2023.

[28] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun. A modern primer on processing in memory. In *Emerging computing: from devices to systems: looking beyond Moore and Von Neumann*, pages 171–243. Springer, 2022.

[29] A. Ramaswami, T. Kenter, T. D. Kühne, and C. Plessl. Efficient ab-initio molecular dynamic simulations by offloading fast fourier transformations to fpgas. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 353–354. IEEE, 2020.

[30] E. Runge and E. K. Gross. Density-functional theory for time-dependent systems. *Physical review letters*, 52(12):997, 1984.

[31] D. Sanchez and C. Kozyrakis. Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. *ACM SIGARCH Computer architecture news*, 41(3):475–486, 2013.

[32] B. Tian, Q. Chen, and M. Gao. Abndp: Co-optimizing data access and load balance in near-data processing. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 3–17, 2023.

[33] L. Wan, X. Liu, J. Liu, X. Qin, W. Hu, and J. Yang. Hybrid mpi and openmp parallel implementation of large-scale linear-response time-dependent density functional theory with plane-wave basis set. *Electronic Structure*, 3(2):024004, 2021.

[34] L. Wang, Y. Wu, W. Jia, W. Gao, X. Chi, and L.-W. Wang. Large scale plane wave pseudopotential density functional theory calculations on gpu clusters. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–10, 2011.

[35] Y. Wei, M. Zhou, S. Liu, K. Seemakhupt, T. Rosing, and S. Khan. Pimprof: An automated program profiler for processing-in-memory offloading decisions. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 855–860. IEEE, 2022.

[36] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.