Tightly-Coupled FPGA Accelerator for Molecular Dynamics Simulation: Hardware-Software Co-Design and Fine-Grained Task Management

Zekang Cheng[1], Zerong He[1], and Xi Jin[1]

[1]*Institute of Microelectronics, Department of Physics, University of Science and Technology of China.*
Contact: {chengzk, hezerong}@mail.ustc.edu.cn, {xuxu, jinxi}@ustc.edu.cn

***Abstract*—Over the past decades, Molecular Dynamics (MD) has been extensively utilized for drug design, protein structure prediction, and system analysis in computational chemistry and biology. However, previous acceleration efforts have often faced challenges with either host-device execution modes, leading to costly communication overheads, or complete FPGA implementations that sacrifice flexibility and programmability.**

In this paper, we present a novel tightly-coupled MD execution framework, combining a single FPGA with a Hard-core CPU, resulting in an impressive 10x performance improvement compared to state-of-the-art CPU implementations (e.g., Gromacs). Our system exhibits characteristics reminiscent of a fine-grained CPU-based MD execution with Low latency computation accelerators. To achieve such efficiency, our hardware-software co-design approach emphasizes reducing scheduling expenses through a decentralized dependency management strategy and a hardware-assisted Multi-Producer Multi-Consumer (MPMC) queue.

Importantly, our methodology is not limited to MD applications alone but can be readily applied to a wide range of fine-grained task-based applications. Moreover, the MPMC queue has the potential to scale and be adapted for use in FPGA clusters of larger scale, further extending its applicability and relevance in diverse computing environments.

***Index Terms*—Molecular Dynamics, FPGA, Multi-Producer Multi-Consumer, Fine-grained Task Scheduling**

I. Introduction

Molecular Dynamics (MD) is a rapidly developing field with a wide range of applications, including drug mechanism studies, system simulations, and computational chemistry[1, 2, 3, 4]. The kernels involved in MD are constantly evolving. Generally, MD is a timestep-based method that requires strong synchronization between each step, and there may be dependencies between multiple kernels involved in each step[5, 6]. Over the past few decades, MD applications have undergone multiple iterations to solve more challenging practical problems. These iterations have extended the timescale of the applications and scaled up the number of particles to ensure result validity. However, this has led to a surge in computational requirements, demanding greater throughput to deliver results within an acceptable time frame. Consequently, significant efforts have been devoted to accelerating MD simulations, aiming to provide enhanced computing power and problem decomposition across processing elements (PEs).

One apparent approach is spatial dimension decomposition, such as GPU acceleration[7, 8, 9, 10, 11]. However, simple decomposition into multiple computing units often suffers from load imbalance. Additionally, frequent synchronization between MD steps is necessary. As Figure 1, the traditional host-device mode introduces a host routing result for both GPU and FPGA synchronization, resulting in high latency and synchronization overhead. This mode has limited throughput and poor energy efficiency, particularly for long-term small-system simulations. Notably, GPUs do not support custom precision well, and their task scheduling latency is higher compared to FPGAs.

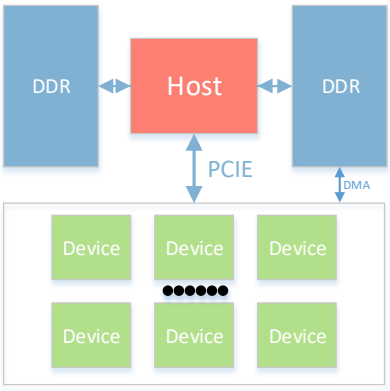

Fig. 1. Host-device Execution framework

Another approach involves tightly coupled FPGA acceleration for low-latency communication and synchronization. In 2019, Martin[12] proposed a purely FPGA-based MD solution, which optimized long-range forces and significantly improved performance compared to the baseline. However, the throughput rate still falls short, and the scheduling capability presents a bottleneck. A fixed ratio of resources based on static profiling was employed, but MD tasks, such as chemical bond calculations, require a certain degree of flexibility and access to general computing resources, thus impacting performance.

Currently, the most effective acceleration approach is exemplified by Anton[13, 14, 3], which has made significant strides by reducing task granularity, employing dedicated hardware acceleration modules, and incorporating small-scale general-purpose computing units. Besides, in fine-grained execution, the overhead of scheduling itself comprises even more, thus recently Anton3 [15]takes a homogeneous architecture to

mitigate this overhead. Anton's performance surpasses that of current commercial clusters. For calculations requiring flexibility, hard-core CPUs can be utilized to save on-chip logic resources. Nevertheless, a majority of Anton's pipeline resources remain fixed, limiting general high utilization for applications with different characteristics.

Taking inspiration from Anton, we conclude that by reducing task granularity, the better overlap between communication and calculation can enhance system performance in MD simulations. Additionally, the tightly coupled CPU-FPGA implementation significantly reduces the synchronization overhead of MD, thereby improving throughput. However, in traditional centralized scheduling, the CPU's task scheduling capability is limited to 15-30MHz[16], which hampers further task granularity refinement. Moreover, previous FPGA utilization heavily relied on fixed resources, resulting in poor versatility. To address these limitations, we have adopted a more generalized approach.

We have designed a tightly coupled FPGA-based MD distributed scheduling method, considering the use of FPGA to harden a scheduling module with a 200MHz scheduling capability to support finer-grained tasks. This approach improves utilization through overlapping and further enhances performance. Our framework is more versatile as it employs configurable pipeline resources and a hard-core CPU. Furthermore, our hardware-assisted scheduling mechanism focuses on distributed and decentralized scheduling with a spin-lock approach, rather than dedicated logic acceleration.

Our contributions can be summarized as follows:

- We propose a novel general framework for a tightly coupled Molecular Dynamics (MD) accelerator, allowing for dynamic task allocation to either FPGA logic resources or hard-core CPUs based on the evolving algorithm characteristics.
- We introduce DDDM, a dedicated data structure for memory version management and a sophisticated scheduling method that effectively manages task dependencies through it.
- We provide a high-performance hardware-assisted low latency MPMC queue, designed to significantly enhance task management efficiency.
- Compared to state-of-art works, our work achieved more than 10x speed up over CPUs and an average 2.14x improvement compared with NVDIA H100 GPU.

These contributions collectively demonstrate the efficiency and versatility of our tightly coupled MD accelerator, empowering it to adapt seamlessly to diverse MD applications with varying computational demands. The proposed scheduling method and hardware-assisted queue offer substantial performance improvements, positioning our framework as a promising solution for accelerating MD simulations on FPGA platforms.

II. MD Background

Molecular Dynamics consists of two fundamental components: force calculation and motion state update. The force calculation involves analyzing interactions between particles in the system, categorized into different force types: bond-term forces (angular forces, dihedral forces, paired bond forces) and non-bonding forces (van der Waals forces and Coulomb forces)(1)(2).

$$E_{total} = E_{bond} + E_{angle} + E_{dihedral} + E_{es} + E_{LJ} \quad (1)$$

$$F = -\nabla \frac{E}{\vec{r}} \quad (2)$$

Bond-term force calculations are intricate, involving a smaller number of particles and thereby consuming a relatively modest proportion of computational power. In contrast, non-bonding forces are predominant, constituting approximately 98% of the computational workload. Van der Waals forces exhibit rapid decay with distance, confined within a specified cutoff radius, and contribute insignificantly to long-range forces. Coulomb forces are comprised of two components: short-range forces, associated with van der Waals forces and inversely proportional to the square of the distance, and long-range forces, which decay gradually and act on all atoms(3)(4). The computation of Coulomb forces entails interpolating particles to grid points using discretization and third-order basis functions, determining charge density based on neighboring particles(5)(6), performing Fourier space calculations through Green's function, and finally obtaining results via inverse Fourier transform - ensuring precise force determination.

$$F_i^{LJ} = \sum_{j \neq i} \frac{\epsilon_{ab}}{\sigma_{ab}^2} \left\{ 48 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 24 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \vec{r}_{ji} \quad (3)$$

$$F_i^C = \frac{qi}{4\pi} \sum_{j \neq i} \frac{1}{\epsilon_{ab}} \left\{ \frac{1}{|r_{ji}|} \right\}^3 \vec{r}_{ij} \quad (4)$$

$$F_i^{LR} = \sum_{j \neq i} \frac{qj}{|r_{ji}|} \vec{r}_{ji} \quad (5)$$

$$\rho_g = \sum_p Q_p \phi(|x_g - x_p|) \phi(|y_g - y_p|) \phi(|z_g - z_p|) \quad (6)$$

MD simulations incorporate periodic boundary conditions to represent three-dimensional chemical systems in real space. This approach allows for the simulation of larger systems while mitigating boundary effects.

Following force calculations, motion states of each particle are updated through straightforward integration using Newton's laws of motion. This process involves adjusting particle velocities and positions based on computed forces.

Moreover, specific applications in computational chemistry and computational biology may introduce additional constraints and operators due to unique environmental characteristics[5]. These complexities can lead to performance degradation when relying on traditional FPGA acceleration. Consequently, the utilization of more flexible resources, particularly task execution on the hard-core CPU, becomes crucial in our application deployment.

III. System Overview

In this section, we will present the deployment and architecture of MD on FPGA with a hard-core CPU, encompassing various components designed to optimize performance. These include a hardware-assisted low latency ready queue, a fine-grained scheduling algorithm and a software-hardware co-design description about the specific performance improvements achieved through this approach.For the entirety of the simulation, Figure 2 provides an overview of the relevant modules and their execution procedures.

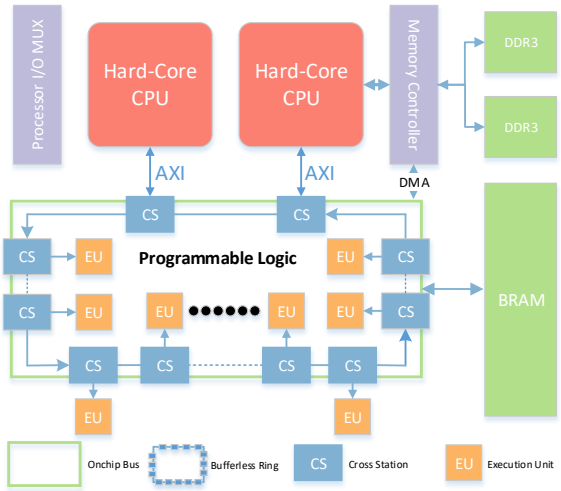

Fig. 2. System architecture overview

As for MD application, the input of the RL module is the data of a node and its neighboring nodes. The interaction calculation of particle pairs is completed by obtaining the neighbor list. The LR module discretizes particles to lattice points, does Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT), and obtains the result by potential energy function. The bonded force is calculated by the CPU because the calculation amount is not large. After that, the motion update obtains the result of the summation module to complete one step (2fs) update, and the particle belongs to the node every eight stepsorts. Bufferless Ring supports a lock-free MPMC queue through the cross station bound to each module. The data format is supported by the CHI protocol. Also, there is another bus for memory access from all EUs and CPUs

A. Fine-grained Task scheduling Algorithm

Achieving optimal overlap between molecular dynamics (MD) calculations and data handling necessitates the implementation of a fine-grained task scheduling mechanism. For system-wide efficiency, distributed scheduling is widely employed, wherein all pipeline tasks are allocated and committed by the CPU. This approach capitalizes on the lower communication latency inherent in on-chip CPUs, as observed in platforms like ZC102, compared to conventional host-device communication via PCIe. However, the existing software-based approach is impeded by the maximum frequency of enqueuing and dequeuing operations, typically ranging between 15-30 MHz. Consequently, the task initiation and commit rates are inherently limited, resulting in the CPU serving as the principal scheduling bottleneck and impeding further task refinement.

To address this limitation, we propose DDDM, a data-driven dependency management strategy that leverages memory as a key component. Upon task submission, we meticulously mark and document the input and output data in a Data Context (DataCtx) management system, using assigned physical addresses. This system captures crucial version information, including the data itself, previous versions, consumers (tasks dependent on the data), and producers (tasks generating the data). Notably, individual data blocks may undergo multiple reads and writes, resulting in multiple versions of DataCtx. However, a unique label ensures consistency across all DataCtx instances. Whenever a new task requires existing data, a new DataCtx is generated, while carefully tracking the previous one.

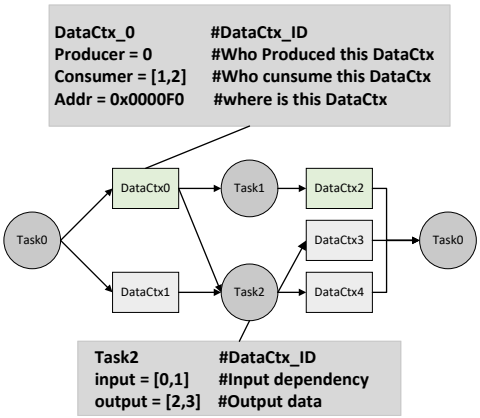

Fig. 3. Data Structure of Scheduling Algorithm

Each task record includes the task's content and the relevant DataCtxs designated for read and write operations. During task execution, the corresponding execution unit retrieves the necessary read DataCtx, updates the consumer's dependency counter, and releases the associated DataCtx after the consumer task completes its utilization. Likewise, the execution unit accesses the designated write DataCtx, updating its status upon completion. Concurrently, when a new DataCtx is created, the scheduler decrements the consumer task's dependency counter and adds executable tasks to the ready task queue.

Throughout the execution phase, the FPGA-based computing unit retrieves tasks from the ready queue, accesses the corresponding input data's start address, performs computation, and advances to the subsequent task in the execution sequence upon completion.

Figure 3 illustrates the execution process of a template task graph. When a new task is created, we construct a DataCtx connection graph based on dependency information, which is expressed as input and output data addresses. Notably, the same color represents the same memory address, which is accessed twice, resulting in two DataCtxs for each with different versions. Moreover, address 0x0000F0 is accessed by both task1 and task2, and a dual-read operation will not

create a new version, given that the data in the address remains unchanged during these two reads. Our scheduling algorithm records tasks as connections of various DataCtxs, aptly representing its data-driven nature.

By adopting DDDM, combined with distributed scheduling and fine-grained task execution, we facilitate substantial overlap between MD calculations and data handling. Consequently, this approach significantly enhances overall system performance and efficiency.

B. Hardware assisted Low Latency queue

In our distributed dependency management algorithm, we achieve decentralization, but the repeated access to the ready queue and version data structures introduces potential data racing, necessitating locking mechanisms. However, software-based lock grabbing operations can incur significant overhead, particularly on the FPGA. To address this challenge, we propose a hardware-assisted low latency queue based on the bufferless ring approach introduced by Wang [17]. We have made algorithmic and architectural adaptations to tailor the queue to our specific framework. Figure 4 illustrates the components and architecture of the ring.

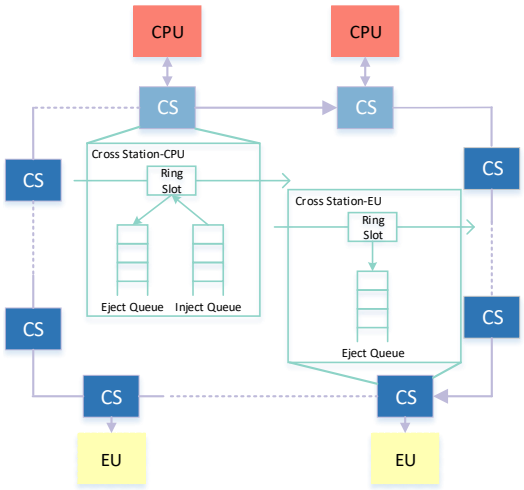

Fig. 4. Ring conponents and Architecture

For simplicity, we utilize a single-direction ring instead of a bidirectional one. Each EU is connected to its respective cross station, identified by its EU ID. The Cross Stations come in two types: one attached to the CPU, featuring both inject and eject queues as the CPU serves as both the dependency manager and an execution unit; the other Cross Stations contain only an eject queue since all EUs on the Programmable Logic solely consume tasks. Additionally, all Cross Stations have a Ring Slot, acting as a node within the ring. Tasks circulate on the ring in a clockwise direction each cycle.

When an EU completes a task, it first accesses memory to maintain dependencies and then proceeds with execution by dequeuing a new task from its eject queue. For a task to be injected into the ring, it is managed by a dependency manager running on the hard-core CPU, persistently attempting to enter the ring until it finds an empty ring slot. On-the-fly tasks inherently carry a destination EU ID, allowing them to be downloaded only to the corresponding cross stations. Moreover, several tags are embedded in the task flit for different identification purposes, as shown in the figure.

The bufferless ring serves as a ready queue in our design, providing low latency with a maximum latency of spinning on the ring. As for memory access, we reserve another bus for atomic memory access. To achieve better load balancing and ensure a lock-free operation, we introduce three improvements.

Firstly, MD kernels conduct fixed computation styles, but the different computation density resulting from atomic distribution among nodes may cause a long-tail effect. To mitigate this performance loss, we employ a sampling strategy for dynamic load balancing. When a task is generated (before it's ready), its computation density is explicitly indicated by the generator, and a 2-bit consumption Tag marks the task's computation consumption. Initially, tasks are evenly distributed to EUs with the same sum of workload Tags. During execution, workload differences may arise, so a sampling between 1000 cycles is conducted to determine which EU's eject queue is lower than the threshold, and later ready tasks are preferentially marked with their EU ID. Importantly, we use sampling as a reference for scheduling rather than a rule, eliminating the need for the most recent value of the queue condition and eliminating the necessity for locks. As a result, tasks are generally evenly dispatched to EUs.

Secondly, to avoid deadlock or live-lock situations, when a task passes the cross station connected to the CPU, its I-Tag will be set. If a task with an I-Tag passes the CPU again, it indicates that the task cannot be accessed on the targeted EU for some reason. The CPU then resets the destination EU ID and injects it back into the queue. Since the CPU can always finish any types of tasks for different durations, it may handle this task when its eject queue is empty for better utilization.

It's worth noting that the eject queue has two pointers, namely the tail and head pointer, thus avoiding data races. This means that to access such a "ready queue," all EUs only need to send the push and pop task to their own cross station. The bufferless ring mechanism will automatically complete the corresponding enqueue and dequeue operations. The FPGA EUs can submit and commit tasks at the on-chip clock frequency of 200 MHz, enabling the task granularity to be refined to even less than 1 µs.

C. Hardware-Software Co-design

Figure 5 presents a comprehensive depiction of the task dependency handling mechanism from task creation to completion. From a CPU's view , tasks are firstly identified based on workload and stored as a task graph in memory by CPU. Section III-A elaborates on the construction of the TaskGraph using datactx, with tasks stored in memory based on datactx-defined connection relationships. Then the Dependency Manager continually accesses all tasks, enqueuing tasks with a dependency counter reduced to zero into its ready queue during execution. Subsequently, the CPU endeavors to enqueue tasks from the queue to the cross station using the Inject queue enqueue process. The Dependency Manager efficiently

determines task destinations based on EU information sampled and task workload characteristics. The corresponding EU's cross station then ejects the task from the ring, sequentially placing it into the EU for execution via the eject queue. After completion, the EU will write back to memory for dependency updates.

Within the EU, only tasks tagged with a destination matching its own cross station are enqueued into the eject queue and executed sequentially in step 1. Upon task completion (step 2), the EU accesses memory to modify the dependency counter based on the datactx information linked to the task, following the algorithm detailed in III-B. Subsequently, the Dependency Manager identifies tasks with a dependency counter of zero and enqueues them into the ready queue.

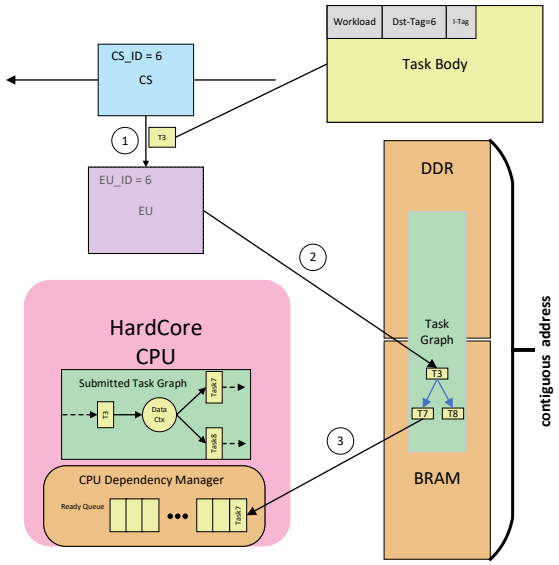

Fig. 5. Task Duration on the whole system

In the realm of computational chemistry, certain critical energy calculations involve flexible operators that prove challenging for traditional FPGA acceleration methods. In our deployment, these operations are efficiently handled by the hard-core CPU. Pre-configured on-chip pipeline resources are employed, while dynamic task dispatching enables the shifting of hotspot execution to CPU execution for online computational tasks, thus optimizing resource utilization.

IV. Evaluation

A. Experiment Setup

Our performance evaluation was conducted on the Xilinx ZCU102 Board, featuring a quad-core Arm® Cortex®-A53 CPU. This chip offers 274080 LUTs, 548160 FFs, 912 BRAM blocks, and 2520 DSP units, providing a balanced performance profile when coupled with the Hard-Core CPU. For our evaluation, we utilized protein simulation datasets from RCSB, encompassing various scales ranging from 4779 atoms to 39122 atoms. With a 9Å cut-off radius, we employed two cell sizes, 9Å and 4.5Å, to explore different task granularity distributions and assess the efficiency of our methodology in accelerating fine-grained execution.

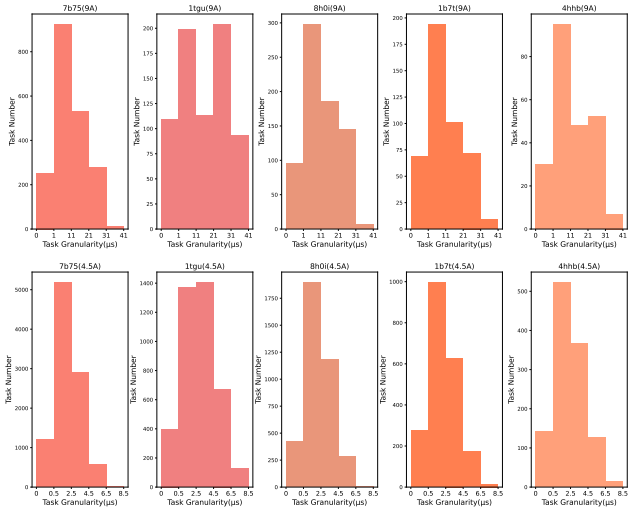

Fig. 6. Task Granularity Distribution

Figure 6 depicts the task distribution, revealing that the majority of tasks within the 9Å cell size fall within the magnitude of 10μs and 1μs in the 4.5Å cell size. Based on our scheduling frequency of 200 MHz, the scheduling overhead aligns remarkably well with these tasks.

TABLE I
RESOURCE USAGE OF EACH EXECUTION MODULE.

	RL pipeline	Grid Mapping	FFT&IFFT	LR Force
LUT	3558(1.3%)	13621(4.97%)	4904(1.79%)	3125(1.14%)
FF	3738(0.68%)	9000(1.64%)	7854(1.43%)	2283(0.42%)
BRAM	6(0.66%)	0(0%)	0(0%)	0(0%)
DSP	35(1.39%)	40(1.59%)	12(0.48%)	37(1.47%)
Number	68	1	3	1

1) HW-assisted Queue: As illustrated in section III-B, scheduling latency arises from over 60 execution units (EUs) competing on the Multi-Producer Multi-Consumer (MPMC) task queue, even with our dependency management algorithm. To address this, we devised a Bufferless Ring-based wait-free Queue, incurring minimal logic resource overhead and achieving latency of at least 2 cycles for most scenarios. In comparison, CPU-centralized scheduling for Cortex-A53 on MD executions typically achieves only about 1MHz.

2) Performance Trade-offs: As highlighted in section III-B, bonded term computations consist of complex kernels challenging for FPGA acceleration, yet they contribute to less than 2% of the overall execution time. Consequently, we opt to shift these bonded term calculations to the Hard-Core CPU. Similarly, Motion Update, FFT and iFFT executions are also handled by the CPU. This approach effectively presents our implementation as a blend of CPU execution with low-latency FPGA acceleration. Additionally, we address the discrepancy in execution times between the CPU and long-range (LR) force calculations. As a result, we experiment with an alternative design, shifting all LR calculations to the CPU, leveraging its ample computation power. In our implementation, LR calculations occur only once per 16 steps, allowing for optimal resource allocation to the LR Units. Finally, the functional modules in EU are designed with reference to work [12] and the resource utilization is shown in Table I

B. Performance comparison

TABLE II
HW SCHEDULING BENEFITS: DATA IN THE TABLE SHOWS AVERAGE TIME TO PERFORM A FULL MD STEP FOR DIFFERENT DATASET

	Both DDM & HW queue	Only DDM	No DDM & HW queue
7b75	139.24 µs	168.87 µs	195.87µs
1tgu	77.81 µs	88.46 µs	98.165µs
8h0i	56.80 µs	67.64 µs	77.52 µs
1b7t	31.40 µs	38.00 µs	44.01 µs
4hhb	19.02 µs	22.46 µs	25.60 µs

1) Benefits of HW-SW Codesign Scheduling Methodology: Table II demonstrates the significant reduction in scheduling overhead achieved through our scheduling methodology, which improves scheduling frequency. Furthermore, the HW-assisted Queue effectively minimizes the racing overhead on the MPMC queue. The results showcase an impressive 41% performance improvement on dataset 1b7t and an average speedup of 36% across all five datasets.

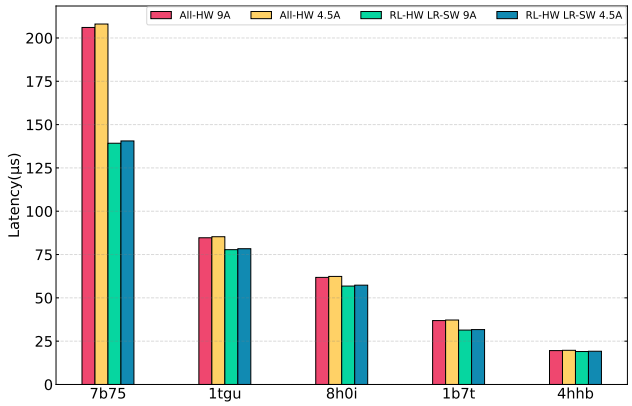

Fig. 7. Performance with different datasets per iteration

2) Design Benefits: Figure 7 presents the performance comparison of different designs for varying cell sizes on the datasets. Shifting LR calculations to the CPU yields notable performance improvements, particularly for larger datasets. This strategic allocation ensures that logic resource consumption primarily comes from RL Units, as demonstrated in Table I. While finer task granularity offers increased concurrency opportunities, it also introduces higher scheduling overhead for tasks lasting less than 1 µs, resulting in a minor drop in performance. In contrast, CPU execution experiences more significant performance losses, as summarized in Table III. This is because general CPU scheduling expenses take over even more portion in less-than-µs task granularity, resulting in an enormous performance loss.

C. Overall Performance

Our performance evaluation encompasses a comprehensive comparison between our approach, CPU, GPU, and a prior FPGA-based design. Each baseline platform is carefully selected for relevance and significance.

- CPU Baseline: We selected the Intel® Xeon® Gold 6258R as our CPU baseline, with a formidable peak performance capability of 2419.2 Gflops.

TABLE III
PERFORMANCE COMPARISON: DATA SHOWS REDUCED EXECUTION TIME OF A FULL MD STEP

	CPU	GPU[19]	FPMD-RL[18]	DDM&HW queue
7b75(39122 atoms)	1441.19µs	324.90µs	144.41µs	139.24µs
1tgu(17173 atoms)	786.82µs	142.62µs	63.39µs	77.81µs
8h0i(15018 atoms)	578.45µs	127.72µs	55.44µs	56.80µs
1b7t(8383 atoms)	306.14µs	69.62µs	30.94µs	31.40µs
4hhb(4779 atoms)	187.46µs	39.69µs	17.64µs	19.02µs

- GPU Baseline: For GPU comparison, we employed data from AMBER benchmarks running on the H100 GPU, renowned for its peak performance of 24.08 Tflops.
- Prior FPGA Implementation: To assess the FPGA domain, we refer to the work by Wu[18], which utilized a Stratix 10 SX FPGA with substantial hardware resources: 933,120 ALMs, 5,760 DSPs, and 11,721 M20K BRAMs.

Our approach delivers compelling performance results, as summarized in Table III:

- A remarkable performance improvement of over 10x when juxtaposed with the CPU baseline, signifying the potency of our FPGA-based solution.
- Noteworthy acceleration, achieving a 2.14x speedup when compared to the GPU baseline, underscoring the FPGA's prowess in fine-grained tasks.
- Demonstrating competitive performance equivalent to the prior FPGA implementation, while astutely utilizing less than half of the on-chip resources, exemplifying our efficiency and resource optimization.

V. CONCLUSION

In this paper, we have introduced a comprehensive software-hardware co-design framework for tightly-coupled Molecular Dynamics (MD) execution on a single FPGA with a Hard-core CPU. Our approach incorporates a decentralized scheduling algorithm and an efficient wait-free Multi-Producer Multi-Consumer (MPMC) queue, based on the Bufferless Ring architecture. Through a thorough performance evaluation on a set of real applications, we have achieved an average execution speed improvement of 1.36x. Importantly, our approach has demonstrated a significant 10x performance increase compared to state-of-the-art CPU implementations, a 2.14x speedup over the H100 GPU, and nearly equivalent performance to a prior-art FPGA-based MD implementation.

Furthermore, the general nature of our framework and scheduling methodology allows for broad applicability across a diverse range of scenarios. Beyond the fundamental protein folding applications, our approach can seamlessly accommodate more flexible operators in MD, including non-standard pairwise interactions, by leveraging the computational capabilities of the Hard-core CPU. In future endeavors, we envisage the introduction of an inter-FPGA bufferless ring for FPGA clusters, enabling scalable message queue communication. On the single FPGA execution front, our framework can readily support other fine-grained applications, thereby paving the way for even greater performance improvements. The versatility and scalability of our approach bode well for its potential impact in various high-performance computing scenarios.

References

[1] Tomas Hansson, Chris Oostenbrink, and WilfredF van Gunsteren. "Molecular dynamics simulations". In: *Current opinion in structural biology* 12.2 (2002), pp. 190–196.

[2] John D McCorvy et al. "Structure-inspired design of β-arrestin-biased ligands for aminergic GPCRs". In: *Nature chemical biology* 14.2 (2018), pp. 126–134.

[3] Kevin J. Bowers et al. "Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters". In: *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. SC '06. Tampa, Florida: Association for Computing Machinery, 2006, 84–es. ISBN: 0769527000. DOI: 10.1145/1188455.1188544. URL: https://doi.org/10.1145/1188455.1188544.

[4] Chun Wu and Joan-Emma Shea. "Structural similarities and differences between amyloidogenic and non-amyloidogenic islet amyloid polypeptide (IAPP) sequences and implications for the dual physiological and pathological activities of these peptides". In: *PLoS computational biology* 9.8 (2013), e1003211.

[5] Scott A Hollingsworth and Ron O Dror. "Molecular dynamics simulation for all". In: *Neuron* 99.6 (2018), pp. 1129–1143.

[6] Derek Jones et al. "Accelerators for Classical Molecular Dynamics Simulations of Biomolecules". In: *Journal of Chemical Theory and Computation* 18.7 (2022), pp. 4047–4069.

[7] Romelia Salomon-Ferrer et al. "Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald". In: *Journal of Chemical Theory and Computation* 9.9 (2013), pp. 3878–3888. DOI: 10.1021/ct400314y.

[8] James C Phillips et al. "Scalable molecular dynamics with NAMD". In: *Journal of computational chemistry* 26.16 (2005), pp. 1781–1802.

[9] Steve Plimpton. "Fast parallel algorithms for short-range molecular dynamics". In: *Journal of computational physics* 117.1 (1995), pp. 1–19.

[10] David Van Der Spoel et al. "GROMACS: fast, flexible, and free". In: *Journal of computational chemistry* 26.16 (2005), pp. 1701–1718.

[11] David Case et al. *Amber 2018*. Apr. 2018.

[12] Chen Yang et al. "Fully integrated FPGA molecular dynamics simulations". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2019, pp. 1–31.

[13] David E Shaw et al. "Anton, a special-purpose machine for molecular dynamics simulation". In: *Communications of the ACM* 51.7 (2008), pp. 91–97.

[14] David E Shaw et al. "Anton 3: twenty microseconds of molecular dynamics simulation before lunch". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2021, pp. 1–11.

[15] David E. Shaw, Peter J. Adams, Azaria, et al. "Anton 3: Twenty Microseconds of Molecular Dynamics Simulation before Lunch". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '21. St. Louis, Missouri: Association for Computing Machinery, 2021. ISBN: 9781450384421. DOI: 10.1145/3458817.3487397. URL: https://doi.org/10.1145/3458817.3487397.

[16] *Proof Points of Intel® Dynamic Load Balancer (Intel® DLB)*. 2021. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/proof-points-of-dynamic-load-balancer-dlb.html#gs.z0yyam.

[17] Tianqi Wang et al. "Application Defined On-chip Networks for Heterogeneous Chiplets: An Implementation Perspective". In: *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 2022, pp. 1198–1210. DOI: 10.1109/HPCA53966.2022.00091.

[18] Chunshu Wu et al. "Upgrade of FPGA Range-Limited Molecular Dynamics to Handle Hundreds of Processors". In: *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2021, pp. 142–151. DOI: 10.1109/FCCM51124.2021.00024.

[19] *Amber23: pmemd.cuda performance information*. 2023. URL: http://ambermd.org/GPUPerformance.php#RCWBenchmarks.