FASDA: An FPGA-Aided, Scalable and Distributed Accelerator for Range-Limited Molecular Dynamics

Chunshu Wu
happycwu@bu.edu
Boston University
Boston, MA, USA

Tong Geng
tong.geng@rochester.edu
University of Rochester
Rochester, NY, USA

Anqi Guo
anqiguo@bu.edu
Boston University
Boston, MA, USA

Sahan Bandara
sahanb@bu.edu
Boston University
Boston, MA, USA

Pouya Haghi
haghi@bu.edu
Boston University
Boston, MA, USA

Chuan Liu
cliu81@ur.rochester.edu
University of Rochester
Rochester, NY, USA

Ang Li
ang.li@pnnl.gov
Pacific Northwest National Lab
Richland, WA, USA

Martin Herbordt
herbordt@bu.edu
Boston University
Boston, MA, USA

ABSTRACT

Conducting long-timescale simulations of small molecules using Molecular Dynamics (MD) is crucial in drug design. However, traditional methods to accelerate the process, including ASICs or GPUs, have limitations. ASIC solutions are not always generally available, while GPU solutions may not scale when processing small molecules. FPGAs are both communication processors and accelerators, with tight coupling between these capabilities, and so could be used to address strong scaling in this domain.

We present FASDA, the first FPGA-based MD accelerator available for community development. FASDA enables the use of FPGA enhanced clusters and clouds to execute range-limited MD, which is the most resource-intensive and computation-demanding component in MD. FASDA is built with a series of plugable components that are adjustable based on user requirements and demonstrates nearly linear scaling on an eight FPGA cluster. It outperforms the state-of-the-art GPU solution by 4.67x, with the resulting prospect of significantly reducing lead evaluation time.

CCS CONCEPTS

• **Hardware → Reconfigurable logic applications**; **Hardware accelerators**; • **Computer systems organization → Reconfigurable computing**.

KEYWORDS

FPGA cluster, distributed system, molecular dynamics

1 INTRODUCTION

Molecular Dynamics (MD) uses physical laws to simulate the movements and interactions of atoms and molecules. One of its applications is in drug discovery, whose high cost in time and money [35] make use of simulations crucial: MD has proven useful in predicting drug-target interactions and optimizing drug properties [18, 30, 43, 65]. These simulations often involve small ensembles of particles (~50K) [4, 34, 42], which is our focus here.

A challenge of MD is achieving timescales on the order of essential biological phenomena, which range into the microseconds to milliseconds and longer. One difficulty is that, to achieve accuracy, MD iterates over discrete, infinitesimal time intervals (generally a few femtoseconds). Another is that iterations must be performed sequentially; parallelization is therefore limited to within iterations. As a result, completing long timescale simulations within a reasonable amount of wall-clock time—say, a day, or a week for "heroic" simulations—requires timesteps to be processed within milliseconds or less. Since each timestep requires substantial communication, MD is a prototypical strong scaling problem.

There are numerous MD software packages in widespread use [3, 8, 9, 16, 40, 55], many of which also support GPUs. But while effective in weak scaling and throughput scenarios, time-scales have been limited to a few microseconds per day [19, 38, 47], a result that can already be achieved with a small number of GPUs.

As is well known, strong scaling is addressed by, first, using the minimum number of nodes and, second, reducing communication latency. The first requires packing maximal computation per device. For the second, low time-of-flight latency is already available in existing high-performance networks. But, at the application level, there is still much room to reduce overhead. The advantage of ASIC- and FPGA-based MD clusters is that computation and communication can be tightly coupled: data transfers, application level to application level, take only a few cycles beyond time-of-flight.

Simulation of long timescales is one of the motivations for the Anton family of ASIC-based MD processors [47–49] with the latest generation Anton 3 attaining a simulation rate of ~200 μs-per-day. But while ASIC-based solutions can have orders-of-magnitude better performance than commodity clusters, they may also have issues with general availability, plus problems inherent with small-run ASIC-based systems.

While FPGAs do not currently have the raw floating point compute of GPUs, their configurability and tight coupling of computation and communication lead to several MD-related benefits. These include flexible and bespoke arithmetic [21, 22], custom computation pipelines [13, 20, 59], complex custom routing of data streams among pipelines [62] (which can be used for low-cost on-the-fly neighbor list generation [12, 14]), and flexible low-latency inter-device communication [56, 58]. Putting these capabilities together, FPGAs have been shown to have the two necessary strong scaling attributes: performance competitive with the highest performing COTS devices through multiple generations [13, 27, 45, 57, 60, 61] and scalable cluster-wide communication [29, 33, 50, 51].

MD simulations are dominated by the non-bonded force computations, which are partitioned into Range-Limited with a cutoff (RL) and Long-Range (LR) force evaluation components. RL is compute-intensive and consumes approximately 90% of the computation, while LR is more memory and communication oriented. The two components are largely independent in terms of data flow and can be treated as two separate tasks. In this work, our focus is on RL. In particular, while LR parallelization and scaling in FPGA clusters and clouds has been studied [29, 50, 51], the issues related to the same with respect to RL have not. To bridge this research gap, we present FASDA, which is, to our best knowledge, the first open-source, scalable, fully distributed RL system that takes advantage of the benefits of FPGAs described above.

FASDA is built with a series of easily plugable components that can be adjusted based on user requirements (Figure 1). First is a baseline single device **Cell Building Block (CBB)**, which is itself composed of compute, storage, and routing units. Second, the system is extended to a distributed decentralized design with a hyper-ring communication topology, synchronized using **chained synchronization** and **cell ID conversion**. Finally, for strong scaling, the original CBB is augmented to form a **Scalable Cell Building Block (SCBB)** equipped with **Scalable Processing Elements (SPEs)**. The system demonstrates almost linear performance scaling with 8 FPGA nodes and achieves over 4× speedup compared to high-end GPUs performing RL evaluation with a state-of-the-art MD package optimized for GPUs.

FPGA deployments where FASTA can be run include clusters of tightly coupled FPGAs [2, 7, 32, 41] and clouds with network-facing FPGAs (SmartNICs) [1, 6, 10, 23, 25, 28, 64]. The broader impact of this work includes showing that the utility of tightly coupled communication and computation, already of established value for SDN and AI, applies also to HPC. The prospect of utilizing the huge number of network-facing FPGAs in data centers is particularly intriguing. The overall contributions are as follows:

- A strongly scalable, decentralized, and open-source MD simulation solution designed for high performance computing system is proposed.
- To maintain a decentralized system, a cell ID conversion technique and a chained synchronization method are employed, which ensure consistent nodes and Processing Elements (PEs) without the need for global synchronization.
- Two levels of plugable strong scaling modules are demonstrated, giving users the flexibility to customize the MD system according to their performance demand and available hardware resources.
- Our evaluation demonstrates that the proposed FASDA delivers over 4.67x speedup for drug discovery based on molecular dynamics simulations compared with the state-of-the-art GPU-based solution.

2 BACKGROUND

This section explains the fundamentals of RL: the physics of force calculation and its relationship with particle distance; the storage of particle information and the interactions among data containers; and the workflow of the iterative steps of RL.

2.1 Physics of RL Forces

RL forces have two components: the short range term of the electrostatic force obtained using the Particle Mesh Ewald (PME) method [15], and the force deduced from the Lennard-Jones (LJ) potential. In this presentation, for simplicity, we refer to RL forces as only the latter; in any case the RL force pipelines are nearly identical.

LJ potential is an empirical potential that describes the Van der Waals interaction between electrically neutral particles. The LJ potential between particle i and j with distance r_{ij} is given by

$$V_{ij}^{LJ} = 4\epsilon_{ij}[(\frac{\sigma_{ij}}{r_{ij}})^{12} - (\frac{\sigma_{ij}}{r_{ij}})^{6}] \tag{1}$$

where ϵ, the dispersion energy, describes the potential amplitude, and σ is the characteristic particle distance at zero LJ potential. Taking the gradient of the potential yields the RL force between particles i and j:

$$\mathbf{F}_{ij}^{LJ} = \frac{\epsilon_{ij}}{\sigma_{ij}^{2}}[48(\frac{\sigma_{ij}}{r_{ij}})^{14} - 24(\frac{\sigma_{ij}}{r_{ij}})^{8}]\mathbf{r}_{ij} \tag{2}$$

Equation 2 seems to imply that the complexity of RL force computation is O(N^2) as the indices i and j indicate the traversal of two dimensions (R4Q3). However, it can be observed that the force decays rapidly as r_{ij} increases, meaning the distant particles contribute little to force computation. In this case, a cutoff radius (R_c) is selected based on error tolerance to determine whether a pair of particles can be exempted from evaluation. To visualize, R_c is represented with halos in Figure 2(a). Only particle pair A-B is evaluated because A and B are in each other's halo. Particle C, on the other hand, is distant from other particles and thus not considered for evaluation with A or B. The overall compute complexity then becomes O(mN), where m is the average number of a particle's neighbor particles. Note that m is dependent solely on the particle density and is nearly always significantly smaller than N.

An additional optimization is the application of Newton's third law: the total computation can be reduced by half as the pairwise forces apply to both particles involved. This leads to a number of possible implementations [46, 53, 56]; the method used here is described in [56].

It is not feasible to simulate a macroscopic simulation space directly with MD, so periodic boundary conditions are commonly applied. This approach is frequently used in solid-state and computational physics to model a large homogeneous system as replicas

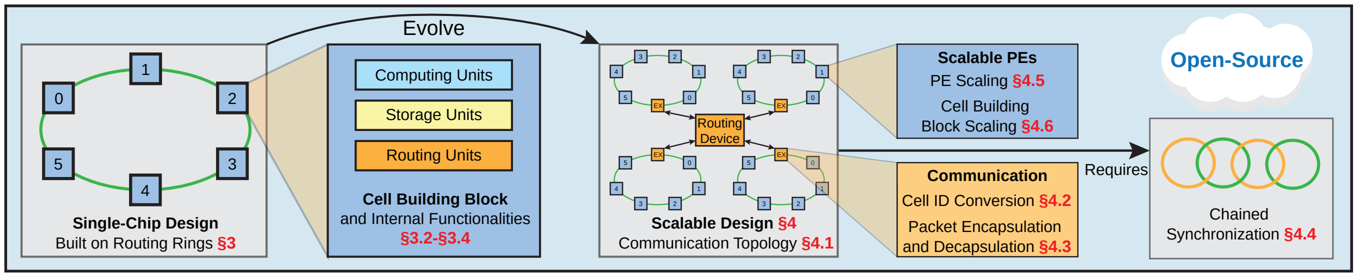

Figure 1: Overview of FASDA.

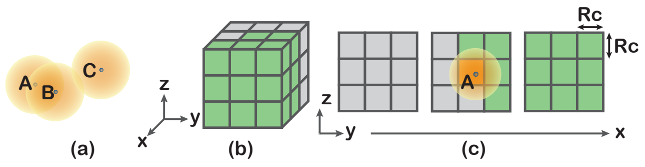

Figure 2: Fundamentals of cutoff and partitioning cell space. (a) The cutoff regions. (b) A simulation space of 3x3x3 cells with R_c cell size. (c) Unfolded view of (b).

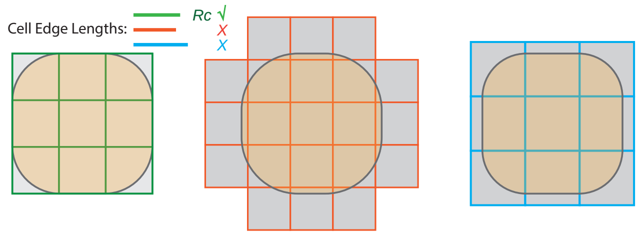

Figure 3: Choices of cell size in 2-D. If cell size is smaller than R_c, more cells must be evaluated. If greater, more invalid pairs to filter. Halo: the import volume of the central cell.

of a small system. In other words, particles exiting the small system are equivalent to the same particles with the same velocity entering from the opposite end, creating periodic boundaries.

2.2 Cell Space Partitioning

A cell list provides a fast, if coarse, way to group particles based on their spatial proximity prior to evaluation. In Figure 2(b), a simulation space is partitioned into cubic cells with a side length of R_c, the cutoff radius. R_c is selected because it is both the smallest value to maintain only 26 possible neighbor cells and the biggest value for efficient particle pair filtering [56, 57]. As illustrated in Figure 3, a slight decrease in cell size introduces more cells to evaluate, drastically complicating the inter-cell communication, while a slight increase brings in unnecessary margins in which the particles do not interact with the central cell. Note that, in FPGA implementations of RL, neighbor lists are recomputed every timestep; the usual benefit for having a margin does not apply.

To ensure that cells can be accessed in parallel, the contents of each cell are stored in distinct memory domains. For example, in the $3\times3\times3$ configuration, 27 memory regions in total are allocated for particle storage. Assuming particle A is located in the central cell

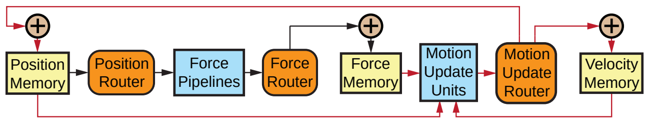

Figure 4: MD RL work flow. The black path (force evaluation) and the red path (motion update) are executed iteratively.

(Figure 2(c)), only the particles in the green surrounding neighbor cells (NC) and A's home cell (HC) can be non-trivial pairs with A. This is because, with Newton's 3rd law applied, a particle from HC only needs to be sent to the 13 NCs for pairing as the other 13 grey cells will send their particles to the HC for pairing (the half-shell method). The cell lists geometrically categorize particles for high data locality; the practical advantage here is that the method allows a particle to be broadcast to all NCs to pair *simultaneously* with hundreds of neighbor particles.

However, not all particles in two neighboring cells are valid. Theoretically, only ~15% of the neighboring particles form valid pairs with a particle according to the equation below:

$$P = \frac{\frac{4}{3}\pi R_c^3}{27R_c^3} = 15.5\% \tag{3}$$

Preliminary filtering of particle pairs is thus desirable: only those where $r_{ij} < R_C$ are passed to the force computation pipeline.

2.3 The MD Process

The workflow of MD RL is illustrated in Figure 4. Memory units are represented in yellow, routing units in orange, and compute units in blue. The force evaluation phase (indicated by black arrows) and the motion update phase (indicated by red arrows) are executed iteratively in a red-black fashion. To begin, the initial position data is routed to the force pipelines for particle pair filtering and force computation. The resulting partial forces are then accumulated in the force memory. In motion update units, the calculated forces are converted into velocity differences and then integrated with the current position and velocity using Verlet integration (see Equations 4-6). This process repeats for subsequent MD RL cycles.

$$\ddot{\mathbf{x}}(t) = \frac{\mathbf{F}(t)}{m} \tag{4}$$

$$\dot{\mathbf{x}}(t+\Delta t) = \dot{\mathbf{x}}(t) + \frac{\ddot{\mathbf{x}}(t) + \ddot{\mathbf{x}}(t+\Delta t)}{2}t \tag{5}$$

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \mathbf{x}(t+\Delta t)\Delta t \tag{6}$$

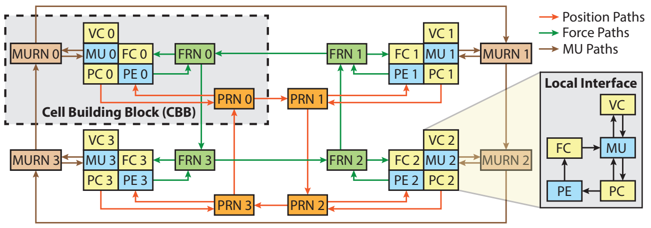

Figure 5: Single-chip architecture overview (e.g., 4 PEs). PC/FC/VC: Position/Force/Velocity Cache; PRN/FRN/MURN: Position/Force/Motion-Update Ring Node; MU: Motion Update Unit.

3 SINGLE-CHIP ARCHITECTURE

In this section, an overview of the single-chip architecture is provided, including a brief introduction to the functions of the PE and the force computation pipeline.

3.1 Overview

The MD RL design depicted in Figure 5 demonstrates a single FPGA with 4 PEs. Each PE is accompanied by a Motion-update Unit (MU) and data caches consisting of a Position Cache (PC), which stores fixed-point positions representing position offsets in a cell denoting its relative position in relation to its local cell; a Force Cache (FC), which stores 32-bit floating point forces; and a Velocity Cache (VC), which stores 32-bit floating point velocities. Additionally, there are three associated routing units: a Position Ring Node (PRN), a Force Ring Node (FRN), and a Motion-Update Ring Node (MURN). Together, a PE and its peripherals (listed above) constitute a Cell Building Block (CBB). The ID of each CBB (0 ∼ 3, as indicated in the figure) is determined by the coordinates of the corresponding cell in space, according to the formula:

$$CID = D_y D_z x + D_z y + z \tag{7}$$

where D_y and D_z represent the total number of cells in the y and z dimensions, respectively and the variables x, y, and z are positive integers that uniquely identify the location of a cell within the simulation space, as shown in Figure 2(c). This indexing method for cell IDs is not only simple, but also optimizes the travel time of a particle within a ring, as a particle is more likely to reach its destination sooner if it moves towards the positive direction of x, y, and z, as depicted in the figure.

3.2 Module Interfaces

To map the 3-D application onto 2-D FPGA fabric, we use 1-D daisy-chains (or "rings" for simplicity) as shown in Figure 5, which connect the CBBs. While there may be some latency overhead, these rings are simple to scale and incur minimal hardware cost. The first ring, known as the position ring (shown by the orange path along PRNs), directs particles from one cell to another, facilitating pairing of particles from different cells. A particle's position has multiple destination cells. The second ring, the force ring (shown by the green path along FRNs), transfers the forces applied on particles from a PE to their respective home cells. A particle's force has only one destination. Finally, the motion-update ring handles cases where particles are relocated from one cell to another, transporting the migrated particles to their target cells.

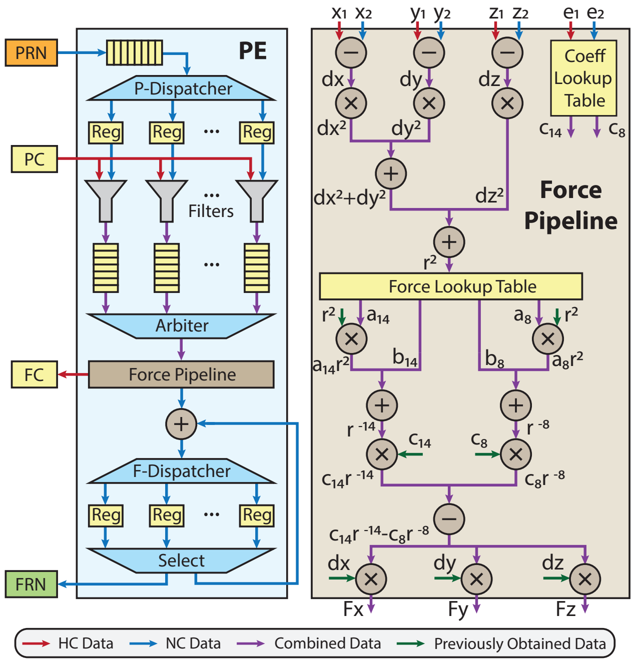

Figure 6: The architecture of a PE. P/F-Dispatchers are used to distribute position/force data. x, y, and z are particle positions; e denotes the element type (e.g., hydrogen)

It is worth noting that data in the rings rotate in predetermined directions: the PRNs move particles in a clockwise direction, while the FRNs move forces counter-clockwise. This is in accordance with the cell ID indexing in equation 7 to minimize the latency in the ring-shaped routing.

Internally, as the magnified diagram on the right indicates, a PE generates forces based solely on position data and transmits the resulting local force directly to its corresponding FC. In contrast, an MU integrates the current position, force, and velocity data to obtain the position and velocity data for the subsequent time step.

If provided with sufficient on-chip resources, the system size and computing power can be expanded by simply adding additional CBBs to the system. Specifically, the additional CBBs are inserted into the rings as ring nodes. Although this comes at the expense of longer ring length and, consequently, increased data transfer latency, the latency can be mostly hidden by the compute units.

3.3 PE Functions

The left half of Figure 6 provides the internal architecture of a PE. When a neighbor position arrives at the PRN, it is checked to determine if the local PE is one of its destinations. If it passes the check, the position is dispatched to one of the registers to pair with the positions from local PC being traversed repeatedly.

When a particle pair passes a filter, it means that the particles are close enough to be evaluated. The pair is then buffered and arbitrated into the force pipeline for force computation. The computed force is split into two paths: one is sent directly to the local FC as a

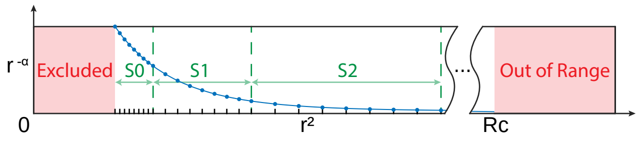

Figure 7: The interpolation method. α is a positive integer too large for $r^{-\alpha}$ to be computed. Here α is 8 or 14. The small r region is excluded due to non-physical high energy.

home force, while the other is negated and accumulated locally as a neighbor force.

In order to accumulate and store a neighbor force, a register is chosen based on the identity of the incoming force. The register's contents are then combined with the incoming force, and the resulting value replaces the current content of the register. After the evaluation of a neighbor particle is completed, its corresponding neighbor force is selected and transferred back to its home cell via the FRN.

3.4 Force Pipeline Functions

The right half of Figure 6 shows the workflow of a force pipeline. The inputs consist of the positions of both particles and their respective element types. As the values of ϵ and σ in Equation 2 are element specific, the elements are used to index a table-lookup to retrieve pre-calculated coefficients for ϵ and σ. The coefficients are then used in the force computation.

During force computation, instead of computing the r^{14} and r^{8} terms directly, we use a force table-lookup technique as described in Equations 8-10 and Figure 7. In order to get the value of $r^{-\alpha}$, we use linear interpolation, as Equation 8 shows, where a and b are lists of parameters indexed by s and b introduced in Equations 9 and 10. Since r^2 is a floating point number, the range of data covered by r^2 is divided into n_s sections denoted as S0, S1, etc. based on the exponent bits of r^2. Moreover, each section is split into n_b regular-sized bins based on the mantissa bits of r^2. The indices s (section) and b (bin) are obtained accordingly (see also, e.g., [14]).

$$r^{-\alpha} = a_\alpha(s,b)r^2 + b_\alpha(s,b) \tag{8}$$

$$s = \lfloor \log_2(r^2) \rfloor + n_s \tag{9}$$

$$b = \lfloor (2^{n_s - s} r^2 - 1) n_b \rfloor \tag{10}$$

Note that a further benefit of this method is that it supports generality by enabling different force models to be implemented with trivial modification. A detail is that setting the cutoff radius R_c to 1 results in full utilization of the precision of both fixed-point raw positions and floating-point r^2, and standardizes the force table-lookup process.

4 SCALABLE ARCHITECTURE

In this section, the extension of the single-chip version to the scalable, multi-chip version outlined and the method used to construct a synchronized and decentralized MD system is demonstrated.

4.1 Hyper-ring-like Communication Topology

The hyper-ring network topology has low bisection width and relatively low diameter [52], which minimizes the cost of adding more PEs or more nodes to the system with relatively small latency overhead. However, this comes at the cost of increased bandwidth demands for nodes communicating over greater distances. Despite having poor bandwidth demand scalability, the hyper-ring topology exhibits distinct advantages for our MD system. The number of CBBs (or nodes) may need to be adjusted to compensate for on-chip resources and scale the system to a different size, and the easy insertion of CBBs or nodes into the system offers flexibility to both the system and users. Also, RL necessarily results in there being a limited number of neighboring nodes, resulting in there being only small additions in latency overhead and bandwidth demand as the system scales. Thus bandwidth degradation in distant communication is avoided.

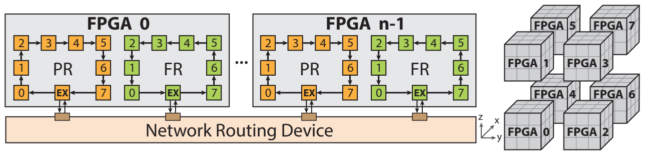

Figure 8: The topology of inter-node connections and an example of cell-to-FPGA mapping.

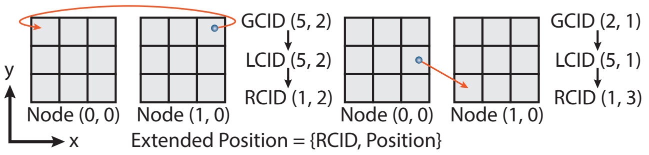

Figure 9: Two levels of cell ID conversion with examples.

Figure 8 shows the hyper-ring-like topology for inter-FPGA connections. The on-chip Position Ring (PR) and Force Ring (FR) are both equipped an extra "EX" ring node for external data transactions, adding only one cycle latency to the rings. Instead of exchanging data between PEs and caches, the EX nodes exchange data between local and remote nodes in a similar manner. This allows remote data to join the sub-network in the local node immediately upon arrival.

The network routing device can be replaced by other FPGA nodes directly connected as a ring to facilitate a hyper-ring of 2nd order, or a network switch to form a hyper-ring-like communication topology. As the system scales up, cascading switches can be used to connect nodes, and a hyper-ring of 3rd order can also be established through direct connections between FPGAs, which can be achieved using FPGA Mezzanine Cards (FMC).

The right side of Figure 8 demonstrates this with eight FPGAs, where each FPGA is assigned to evaluate a specific section of the simulation space. While physically connected on the user-defined network routing device, the FPGAs are logically organized to form a 3-D torus, which also matches the communication pattern.

4.2 Cell ID Conversion

To maintain homogeneity among FPGA nodes and CBBs, two levels of ID conversion are set up: one to convert the Global Cell ID (GCID) to Local Cell ID (LCID), and another to convert LCID to relative cell ID (RCID). Each cell has a unique GCID. That is, each cell has a unique list of neighbor cell IDs, introducing heterogeneity to the

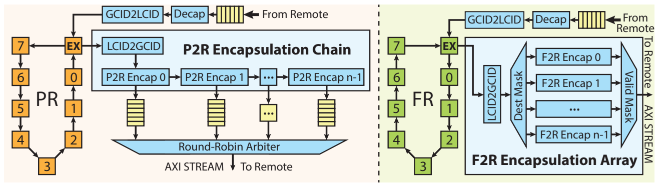

Figure 10: Arriving/departing data processing. P2R/F2R: position/force to remote.

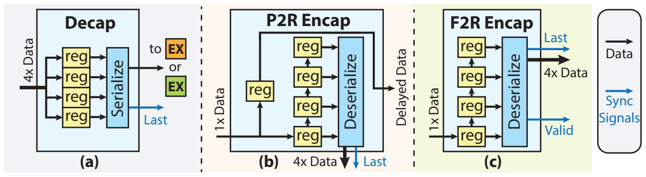

Figure 11: Details of data processing sub-modules.

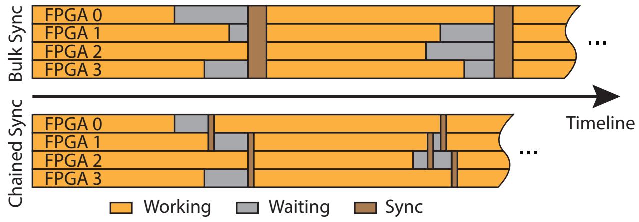

Figure 12: Two synchronization methods for 4 FPGAs. In this example, each FPGA only communicates with its neighbors.

system. An approach to preserve homogeneity is to dynamically generate the lists online, but at relatively high hardware costs.

To avoid the use of expensive dynamic IDs, the GCID of a particle from another node is converted to LCID only upon arrival. This assigns a static cell ID to each local cell as if they are all from node (0,0), ensuring homogeneous cell ID matching on any FPGA node. Figure 9 illustrates two 2D ID conversion examples. The left example shows a particle from the cell with GCID (5,2) in node (1,0) being sent to node (0,0) for evaluation. Its LCID stays the same as its GCID since node (0,0) does not require GCID conversion. The right example shows a particle from the cell with GCID (2,1) in node (0,0) being sent to node (1,0). Its converted LCID becomes (5,1), indicating the relative position between the source and the destination cell. The destination cell has GCID (3,0) but appears as (0,0) in its local node.

After a particle reaches a destination CBB, its LCID is converted to RCID and then concatenated with the fixed-point position for easy distance calculation (by direct subtraction). RCID has a range from 1 to 3 in each direction, with a particle in its local cell assigned an RCID of (2,2,2) in 3-D. This is because the leading "1" in fixed-point needs to be located for fixed-to-float conversion for further force evaluation. Therefore, starting from 1 simplifies the conversion process. The use of fixed-point positions is motivated by the reduction in hardware resource cost for filters, which can number in the hundreds in this design.

4.3 Communication Interface

The processing of data upon arrival or departure is illustrated in Figure 10. Upon arrival, a 512-bit AXI-Stream position (or force) packet that contains four pieces of data is received and unpacked into separate data pieces with headers that contain particle identification information (Figure 11(a)). The data is then serialized and sent to the EX node on a position (or force) ring, where it is injected into the ring for further processing.

The encapsulation process differs for positions and forces, as positions may have multiple destinations while forces have a unique destination. To encapsulate a position packet, an encapsulation chain enables reuse of position data to reduce fan-out. A P2R encapsulator can be regarded as a departure gate. The position data is passed through a series of n P2R encapsulators to find its departure gates, where n is the number of neighboring FPGAs to which a local position may be transmitted. Four positions are wrapped up as a position packet, and buffered once all four registers are filled as shown in Figure 11(b), and then arbitrated for departure.

Since each force only has one destination, there is at most one force packet departure in a cycle and no need for an arbiter. Thus, a force is directed to its departure gate using a destination mask and sent out once the valid signal becomes high (after all four registers are filled), as shown in Figure 11(c).

Both position and force packets contain a *last* signal for synchronization, which is activated after all data associated with a destination node has been processed. The chained synchronization mechanism is explained below.

4.4 Chained Synchronization

Distributed spatial simulations often use the Bulk Synchronous Parallel (BSP) model, which can be problematic due to its centralized nature and susceptibility to the straggler problem [5, 11], where the delay of a single worker slows down the entire system. To address these issues, instead of relying on traditional bulk synchronization methods, we leverage the advantages of our MD architecture by analyzing its data dependency pattern and implementing a chained synchronization approach. This technique maintains a fully-distributed and scalable system while mitigating the impact of stragglers on performance.

Figure 12 illustrates a comparison between the conventional bulk global synchronization and our proposed chained synchronization method. Bulk synchronization can be performed using a host or a central FPGA, but using a host can result in latency of milliseconds for a single MD iteration, while using a central FPGA can destroy the decentralized and homogeneous nature of the system, as well as introduce additional latency due to extra communication. Additionally, the number of valid particle pairs for each FPGA to evaluate can be hard to predict and is affected by the straggler problem. In contrast, our approach is to use localized, fine-grained synchronization among FPGAs as depicted in Figure 12. Each FPGA only synchronizes with its immediate neighbors in a chained manner. Although the chained synchronization method still relies on the slowest board to finish, it decouples the distant FPGA nodes from the slow one, providing them with a head start into the next iteration while preserving a decentralized and homogeneous system.

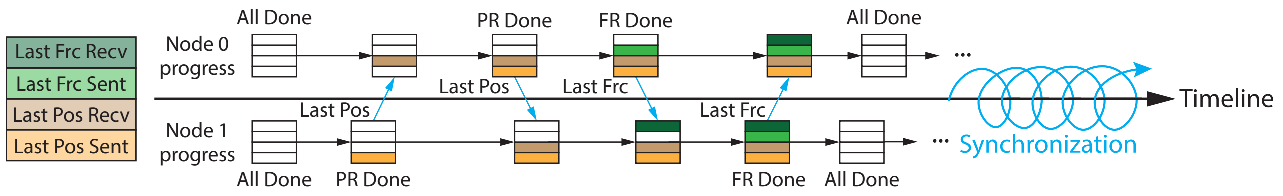

Figure 13: The behavior of the chained synchronization between two FPGA nodes as an example. Frc: force. Pos: position.

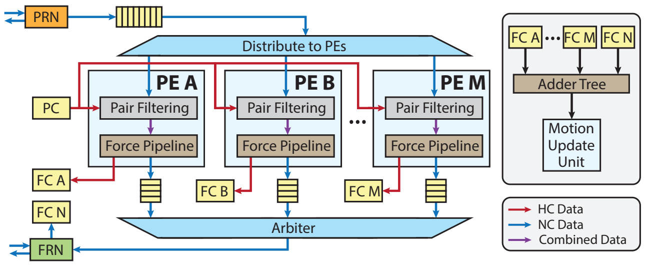

Figure 14: Architecture of a CBB with Multiple PEs

Figure 13 demonstrates the operation of the chained synchronization method, with the motion update phase excluded for brevity. At the start, node 0 and node 1 send each other their position data. At some point, node 1 transmits a "last position" signal along with the data packet after all positions have been routed by its local PR. Node 0 receives the signal and recognizes that a "last force" signal can be returned as a feedback to the "last position" signal only after processing all positions from node 1. This process is mutual, such that node 0 also sends "last position" to node 1, and receives "last force" from node 1. When all four criteria are met, a node can independently proceed to the motion update phase. The motion update phase follows a similar but simplified synchronization pattern as there is only a single receive/transmit signal from/to another node instead of two.

When an FPGA node needs to synchronize with multiple neighboring FPGAs, the synchronization pattern is unchanged, but counters are added to keep track of the number of "last position" and "last force" signals received. A criterion is met when the number of "last" signals sent or received equals the number of neighboring nodes.

4.5 PE Scaling

Thus far, we have presented a system that includes both computation and communication components in a manner to support weak scaling (in the sense that components are replicated assuming simulation size increases proportionally). In order to improve the support for strong scaling, we take advantage of the re-configurable platform and trade the number of CBBs for a greater number of PEs with respect to a certain amount of hardware resources on-chip.

To enhance the performance of an FPGA on the reduced dataset in order to achieve strong scaling, we employ multiple PEs to process a single cell, as opposed to the previous CBB architecture where one PE was used for each cell. By reducing the number of CBBs, an FPGA now is responsible for a smaller portion of a simulation space. The size of the rings is also reduced, which in turn reduces the routing latency on the rings. This results in less congestion on the rings, allowing us to allocate more resources directly to the PEs without worrying about the bandwidth on the data path.

Figure 14 displays the updated architecture. The internal structure of a PE remains unchanged, but the data input and output patterns are updated. We recognize that the PR has lower utilization compared to a PE or FR due to the exceptional data locality of position, as a single position is broadcast to many CCBs for processing. But though many neighboring positions can be processed in parallel in a PE, each position still requires over 100 cycles of processing before the next one can be processed, granting the position ring ample routing time. Thus, a single PR is sufficient to provide multiple PEs with neighbor position data. At the same time, since all PEs (in the figure) are processing the same cell, the home position can easily be broadcast to all PEs.

Regarding force output, ideally, a force pipeline produces a pair of partial forces per cycle. The home force component is accumulated directly into an FC, requiring more FCs to meet the increasing demand for home force accumulation. The neighbor forces are first accumulated in the PE for locality, and then arbitrated and injected into the FR for routing. The local FRN feeds the neighbor forces to FC N, which is designated to store neighbor forces due to the increased throughput of the shortened FR.

After the multiple FCs, force combination takes place. The forces in the FCs are stored and indexed by particle ID to maintain memory alignment, facilitating easy force accumulation. The force datapath can be simplified by not accumulating them directly during force evaluation, but rather during motion update. Upon request by an MU, partial forces are combined in an adder tree. One-per-cycle throughput is maintained to match the MU.

In addition to the benefit of having shorter routing rings, another advantage of scaling the PEs is that it only requires scaling the FCs with the number of PEs, rather than PCs or VCs. This approach leaves more memory resources for other purposes.

4.6 CBB Scaling

At a certain point, adding more PEs no longer enhances performance as the performance becomes limited by the bandwidth of the routing rings or by the particle broadcast from the PCs. Hence, we propose a "stronger"-scaling approach, which aims to scale the CBB for higher throughput.

Figure 15 shows the architecture of a Scalable CBB. It consists of two SPEs, where each SPE is grouped by n PEs, $n + 1$ FCs, a PC, a PRN, and an FRN as described in PE Scaling section. Since the bottlenecks originate from the routing rings and PCs, this grouping method reduces the bottlenecks within an SPE and allows higher performance to be achieved by introducing additional SPEs to the system. The additional SPEs can be attached to the highlighted

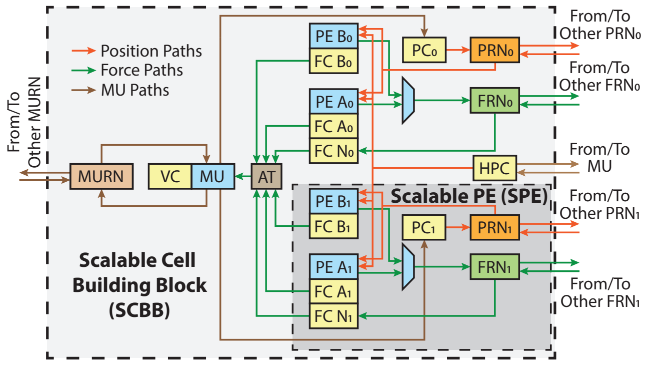

Figure 15: Architecture of an SCBB with 2 SPEs, where an SPE consists of 2 PEs. AT: Adder Tree. HPC: Home Position Cache.

SPE to increase throughput (as shown in the figure). The second SPE uses a separate set of routing paths to ensure equally short rings. Since both SPEs work on the same cell within a single SCBB, the force results can be combined directly in an adder tree. The VC, MU, and MU routing paths do not scale with the SCBB as MU takes much less time compared to force evaluation. The number of EX nodes on the rings is also scaled, sharing the communication bandwidth with other FPGA nodes.

To prevent the transmission of duplicated neighbor positions to PRNs, the positions of all home cells are divided into two disjoint subsets and stored in PC_0 and PC_1. These PCs are now used only for neighbor position broadcast, and a single separate Home Position Cache (HPC) is responsible for home position traversal. During the local MU update, HPC, PC_0, and PC_1 are all updated simultaneously. It is worth noting that PC_0 only takes positions with even particle IDs, while PC_1 only takes odd ones. If more than 2 SPEs are instantiated, they only need to work on particles with interleaved IDs to ensure a balanced workload. Meanwhile, only the HPC is used to update local position, as it contains all the position information for the cell, while the other PCs only hold partial information.

5 EVALUATION

5.1 Experimental Setup

The FASDA system was developed using Verilog/SystemVerilog HDL, implemented and validated on the New England Research Cloud (NERC) using Xilinx Vitis and Vivado 2021.2. Our experiments were carried out on the Open Cloud Testbed [23], where we used up to 8x Xilinx UltraScale+ Alveo U280 FPGA boards running at 200 MHz with both QSFP28 ports connected to a Dell Z9100-ON 100 GbE Switch, with UDP protocol established. Each FPGA contains 1303K CLBs, 2607K flip-flops, 2016x 36-Kb Block RAMs (BRAM), 960x 288-Kb Ultra RAMs (URAM), and 9024x DSP slices. To enable comparisons, we also run OpenMM, one of the state-of-the-art MD software packages, on high-end CPUs and GPUs, including an Intel Xeon Gold 6226R processor with up to 32x threads, as well as up to 2x Nvidia A100 GPUs connected via NVLink and up to 4x Nvidia V100 GPUs connected all-to-all with NVLink. To ensure fairness, we ran OpenMM with only the LJ force field.

To ensure the generality of our results, we used a custom dataset that involves the initialization of 64 randomly distributed sodium particles in each cell, while ensuring that none of the particles are too close to be excluded. In both FASDA and OpenMM simulations, a cutoff radius of 8.5 Å was used, along with a simulation time step of 2 fs (2^{-15} seconds).

5.2 Overall Performance

Figure 16 shows the simulation rate of different platforms in terms of μs/day, which refers to the number of microseconds of simulation that can be executed in a day. It shows both weak-scaling and strong scaling capabilities of FPGAs, CPUs, and GPUs. The assessment is carried out on five simulation spaces, with the first four spaces scaled up by 3x3x3 cell blocks to demonstrate weak-scalability, while the last 4x4x4 simulation space is used to exhibit the strong-scalability characterizations.

In the case of weak scaling demonstrated by the first four configurations, it is evident that both CPUs and GPUs suffer from performance loss at a larger number of computing nodes. For example CPUs exhibit competitive performance for smaller space sizes and good weak-scaling up to four threads, but performance degrades for larger numbers of threads. As for GPUs, doubling the number of GPUs does not preserve the simulation rate for a doubled workload; instead, it only provides half the simulation rate. In contrast, the simulation rate of FPGAs remains consistent at around 2 μs/day for all four configurations.

In terms of strong scaling, CPUs scale well for up to 4 threads, but suffer from significant overhead for more than 8 threads and eventually result in negative scaling for 16 threads and beyond. The simulation rate of GPUs decreases as the number of GPUs increases, primarily due to the long communication latency and low efficiency for a small number of particles. On the other hand, FPGAs provide strong scalability through the use of SCBBs and SPEs. In the case of a 4x4x4 simulation space, where each FPGA is responsible for 2x2x2 cells, the performance is increased to 5.26x with 3 PEs per SPE and 2 SPEs per SCBB compared to 1 PE per cell. In contrast, 2 GPUs and 4 GPUs result in 26% and 49% performance loss respectively compared to 1 GPU. With the negative strong scaling of GPUs, it is fair compare the FPGA results with the best GPU result and obtain a 4.67x speedup in performance.

The right section of Figure 16 displays the measured results for CPUs and GPUs, as well as the simulated results for FPGAs. The efficiency of a single GPU increases as the workload grows, as its performance only drops by 60% when transitioning from 4x4x4 to 8x8x8 cells. We then observe that the GPU's efficiency peaks at 8x8x8 cells, as its performance is halved for 10x10x10 cells, matching the scaling of the workload. Even for 10x10x10 cells (64K particles), GPUs still demonstrate negative strong scaling due to their frequent synchronization with long latency and the reduced efficiency when mapped with a smaller portion of the particles. In the FPGA simulation communication latency between two FPGAs remains the same as with 8 boards, and is performed for 64 and 125 FPGAs respectively with each working on 2x2x2 cells, providing an intuitive insight into the system.

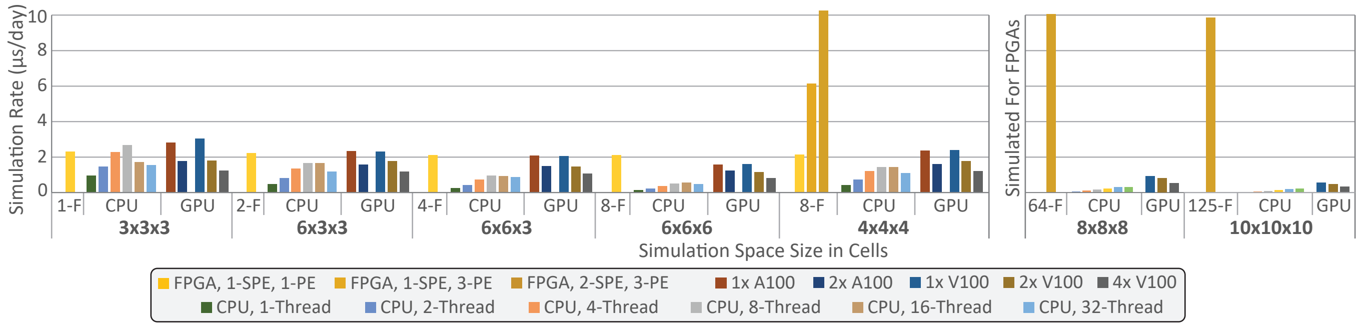

Figure 16: Scalability comparison. 1-F: 1x FPGA board. 1-SPE: Each SCBB contains 1x SPE. 1-PE: Each SPE contains 1 PE.

For the rest of the evaluations, each FPGA only works on 3x3x3 or 2x2x2 cells, and the number of FPGAs is scaled with respect to the simulation space configurations (unless otherwise specified).

5.3 Utilization Breakdown

Utilization of critical components in the design, including PRs, FRs, filters, PEs, and MUs, is illustrated in Figure 17. The figure provides a detailed breakdown of hardware and time utilization for all design variations. Hardware utilization refers to the average amount of work performed by a component in comparison to its capacity, while time utilization represents the average proportion of time that a component is active, during which the pipeline may not be full, but is functioning. The former indicates the effectiveness of the components, while the latter provides an understanding of the system's overall operation flow.

It is observed that the PEs remain active for about 80% of the total operating time, with a hardware utilization of approximately 50%~60% across all the design variations. This suggests that the FPGA design satisfies a critical requirement for strong scaling, whereby the computing units retain efficiency even when a larger number of computing units are used, with each working on a smaller number of particles. Furthermore, the upstream filters match the PEs well in terms of hardware utilization, indicating that the number of filters (6 per pipeline in our experiments) matches the PE throughput that generates one force per cycle.

The utilization of the PR and FR routing components is also presented. It is noteworthy that in weak scaling scenarios (the first four configurations of the simulation space), both the hardware and time utilizations of PR increase. This is because the position data is fragmented and needs to be sent to multiple nodes, weakening the data locality and resulting in more data pieces spinning in rings. The FR hardware utilization also increases with more nodes involved, which extends the routing path for forces. For strong scaling, the utilizations of both PR and FR increase from 4x4x4-A to 4x4x4-B due to the populated PE in an SPE. However, the utilizations remain almost the same from 4x4x4-B to 4x4x4-C. This is because doubling the SPEs involves doubling the routing rings, splitting workload into two halves and nearly doubling the performance. In general, the system is well-balanced for all the design variations, only with the PR underused due to the excellent locality of position data.

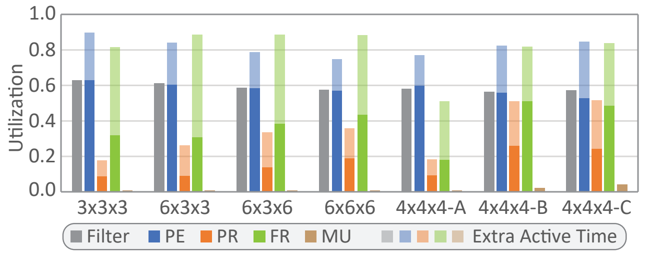

Figure 17: The utilization of key components for the design varieties. A: 1-SPE, 1-PE. B: 1-SPE, 3-PE. C: 2-SPE, 3-PE.

Finally, the MU has the lowest overall utilization (< 5%) leaving the majority of the computing power focusing on the most complex tasks, i.e., force evaluation.

5.4 Communication Intensity

Figure 18 provides the communication status of the multi-FPGA implementations, demonstrating the communication requirement and intensity between distinct FPGA nodes. In our experiments we used separate QSFP28 ports for position and force communication. Figure 18(A) shows that even in the strong scaling configuration of 2-SPEs and 3-PEs, the average bandwidth demand for an FPGA is below 25 Gbps for either position or force, which is well below the available 100 Gbps bandwidth. However, peaks in communication intensity could potentially overwhelm the routing device such as a switch, causing packet loss, and therefore we limit the transmission of each board to once per several cycles using cooldown counters, effectively spreading out a peak over a period of time. As communication and computation are executed simultaneously, with computation typically much more intense than communication, the latency loss in communication caused by cooldown is hidden.

Figure 18(B) shows the breakdown of position and force communication intensity in percentage with respect to other FPGA nodes. We assume that the FPGA nodes are logically connected as previously illustrated in Figure 8. In reality, an FPGA only communicates intensely with the nodes logically close to it, particularly for forces. This is because particles sent from node 0 to node 7 sometimes do not pass through any filter at all, since node 7 is located at the corner of node 0. This occurrence significantly reduces the bandwidth demand for force communication, as zero force is simply discarded

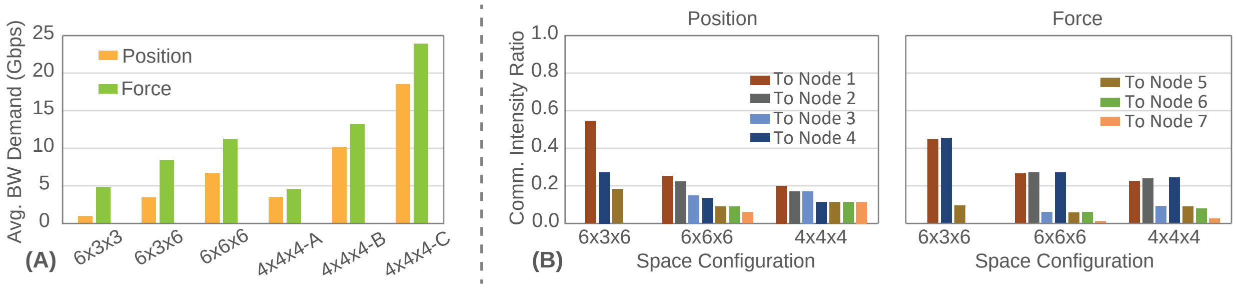

Figure 18: Communication Bandwidth Demand and Breakdown for Different Designs

Table 1: Hardware Utilization of All Design Variations

Design	# FPGA	LUT	FF	BRAM	URAM	DSP
3x3x3	1	40%	22%	29%	20%	20%
6x3x3	2	44%	24%	38%	31%	20%
6x6x3	4	46%	24%	33%	42%	20%
6x6x6	8	46%	24%	33%	42%	20%
4x4x4-A	8	23%	16%	31%	13%	6%
4x4x4-B	8	35%	20%	51%	18%	14%
4x4x4-C	8	52%	26%	76%	28%	27%

A : 1-SPE, 1-PE. B: 1-SPE, 3-PE. C: 2-SPE, 3-PE.

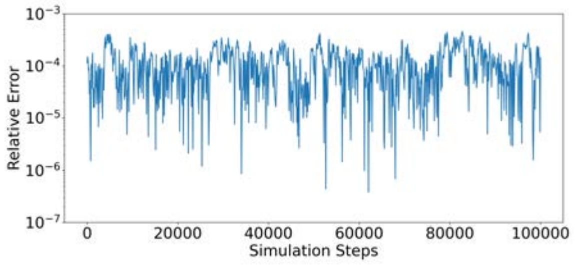

Figure 19: Energy Relative Error with Respect to OpenMM

rather than returned. This communication characteristic also provides an opportunity for the use of hyper-ring routing topology, as the bandwidth demand for routing data to more distant nodes diminishes, compensating for the shortcomings of hyper-rings due to the small number of communication links.

5.5 Resources Consumption

Table 1 presents the resource consumption for all the implementation variations discussed earlier, including three configurations for the 4x4x4 simulation space. In the weak-scaling scenario, the cost of LUTs and FFs remains stable, while the memory cost increases significantly, especially from 3x3x3 to 6x3x3. This is due to the significant change in design required to handle and process data from remote nodes. Resource consumption can be, to some extent, balanced by trading off LUT, BRAM, and URAM.

5.6 Energy Conservation

To ensure the stability of the physical simulation system, we evaluated the energy convergence status of FASDA by comparing it with a 64-bit double precision floating-point simulation that was run using OpenMM for 100,000 iterations on the 4x4x4 simulation space. Our observations from Figure 19 indicate that the relative error is always significantly less than 10^{-3} and generally below 10^{-4}. The overall energy is conserved.

6 RELATED WORK

There are many well-established MD software packages for CPUs and GPUs, including AMBER [9], Desmond [8], GROMACS [3], NAMD [40], LAMMPS [55], and OpenMM [16]. These packages offer similar main functionalities and come with various user-customizable options, such as Monte Carlo simulations and implicit water environment.

For production MD ASIC architectures, the Anton series (described above [47–49]) has provided an approximately tenfold performance increase with each successive generation. The MDGRAPE series [17, 36, 37, 54],which dates back to the 1990s, has evolved from four chips with a total peak performance of 4.2 GFLOPs to the current version with 512 SoCs with 2.5 TFLOPs per chip. Its most recent upgrade, MDGRAPE-4A [33] offloads the 3D-FFT based convolution to Intel Arria 10 FPGAs, indicating further that FPGAs are valuable in MD acceleration.

Partial surveys of FPGAs in MD include [24, 44]. An earlier project related to this one produced modest scalability with four FPGAs for complete MD (including LR and bonded forces) and was implemented in OpenCL [39]. In other relevant multi-FPGA work, a N-body accelerator is implemented multiple Intel Stratix 10 FPGAs with OpenCL and also demonstrates strong scaling on FPGAs [31]. Of special note is the ARUZ project, consisting of ~26,000 FPGAs, including Xilinx Artix and Zynq FPGAs [26]. This massive FPGA cluster uses the Dynamic Lattice Liquid (DLL) method and requires global synchronization. For single FPGA solutions, besides the work already described, there is [63], an MD accelerator designed for developing semiconductor materials; it is deployed on a Xilinx Alveo U200 FPGA, outperforming an Nvidia RTX 2080 Ti by 20%.

7 CONCLUSION

We present FASDA, an open-sourced hardware acceleration system for range-limited molecular dynamics based on FPGAs. FASDA is fully distributed and capable of significant strong scaling with a small number of particles.

To achieve this distributed capability, we utilize two techniques: (1) a two-level cell ID conversion that eliminates the need for dynamic ID computing and creates a homogeneous environment for FPGA nodes and PEs, and (2) a chained synchronization technique that enables global synchronization through local synchronizations in a chain-reaction manner, without requiring central FPGA nodes or host control.

To achieve strong scaling, we have scaled an original cell building block into a scalable cell building block which includes scalable PEs that contain several original PEs. The structured design has relatively well-balanced key components with high hardware utilization, enabling users to easily parameterize the design to meet their resource or performance requirements.

In comparison with GPUs, FASDA achieves a 4.67x speedup compared to the state-of-the-art GPU solution in MD, demonstrating the strong-scalability of the FPGA-based solution.

ACKNOWLEDGMENTS

This research was partially supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, ComPort: Rigorous Testing Methods to Safeguard Software Porting, under Award Number 78284. This work used platform resources partially supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation". This work was supported, in part, by the NSF through awards CCF-1919130 and CCF 2151021, the NIH through awards 1R41GM128533 and R44GM128533, by grants from Red Hat, and by AMD and Intel both through donated FPGAs, tools, and IP. The Open Cloud Testbed, developed through NSF award CNS-1925504, was used for development.

REFERENCES

[1] [n. d.]. FAbRIC (FPGA Research Infrastructure Cloud). https://wikis.utexas.edu/display/fabric/Home. Accessed: 2023-03-21.

[2] [n. d.]. Heterogeneous Accelerated Compute Clusters. https://www.amd-haccs.io. Accessed: 2023-08-14.

[3] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. 2015. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* 1-2 (2015), 19–25. https://doi.org/10.1016/j.softx.2015.06.001

[4] Maral Aminpour, Carlo Montemagno, and Jack A Tuszynski. 2019. An overview of molecular modeling for drug discovery with specific illustrative examples of applications. *Molecules* 24, 9 (2019), 1693.

[5] Eman Bin Khunayn, Shanika Karunasekera, Hairuo Xie, and Kotagiri Ramamohanarao. 2017. Exploiting Data Dependency to Mitigate Stragglers in Distributed Spatial Simulation. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (Redondo Beach, CA, USA) *(SIGSPATIAL '17)*. Association for Computing Machinery, New York, NY, USA, Article 43, 10 pages. https://doi.org/10.1145/3139958.3140018

[6] C. Bobda, J. Mandebi, P. Chow, M. Ewais, N. Tarafdar, J.C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M.C. Herbordt, H. Shahzad, P. Hofstee, B. Ringlein, J. Szefer, A. Sanaullah, and R. Tessier. 2022. The Future of FPGA Acceleration in Datacenters and the Cloud. *ACM Transactions on Reconfigurable Technology and Systems* 15, 3 (2022), 1–42. doi: 10.1145/3506713.

[7] Taisuke Boku, Ryohei Kobayashi, Norihisa Fujita, Hideharu Amano, Kentaro Sano, Toshihiro Hanawa, and Yoshiki Yamaguchi. 2019. Cygnus: GPU meets FPGA for HPC. In *International Conference on Supercomputing*. https://www.r-ccs.riken.jp/labs/lpnctrt/assets/img/ lspanc2020jan_boku_light .pdf.

[8] Kevin J Bowers, Edmond Chow, Huafeng Xu, Ron O Dror, Michael P Eastwood, Brent A Gregersen, John L Klepeis, Istvan Kolossvary, Mark A Moraes, Federico D Sacerdoti, et al. 2006. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. 84–es.

[9] D.A. Case, T.E. Cheatham III, T. Darden, H. Gohlke, R. Luo, K.M. Merz, Jr., A. Onufriev, C. Simmerling, B. Wang, and R.J. Woods. 2005. The Amber Biomolecular Simulation Programs. *Journal Computational Chemistry* 26 (2005), 1668–1688.

[10] A.M. Caulfield, E.S. Chung, A. Putnam, H. Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A Cloud-Scale Acceleration Architecture. In *49th IEEE/ACM Int. Symp. Microarchitecture*. 1–13.

[11] P.H. Chen, P. Haghi, J.Y. Chung, T. Geng, R. West, A. Skjellum, and M.C. Herbordt. 2022. The Viability of Using Online Prediction to Perform Extra Work while Executing BSP Applications. In *IEEE High Performance Extreme Computing Conference*.

[12] M. Chiu and M.C. Herbordt. 2009. Efficient Filtering for Molecular Dynamics Simulations. In *Proceedings of the IEEE Conference on Field Programmable Logic and Applications*.

[13] M. Chiu and M.C. Herbordt. 2010. Molecular Dynamics Simulations on High Performance Reconfigurable Computing Systems. *ACM Transactions Reconfigurable Technology and Systems* 3, 4 (2010), 1–37.

[14] M. Chiu, M.A. Khan, and M.C. Herbordt. 2011. Efficient Calculation of Pairwise Nonbonded Forces. In *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. doi: 10.1109/ FCCM.2011.34.

[15] T. Darden, D. York, and L. Pedersen. 1993. Particle Mesh Ewald: an $N \log(N)$ method for Ewald sums in large systems. 98 (1993), 10089–10092.

[16] P. Eastman and V.S. Pande. 2010. OpenMM: A Hardware-Independent Framework for Molecular Simulations. *Computing in Science and Engineering* 4 (2010), 34–39.

[17] Toshiyuki Fukushige, Makoto Taiji, Junichiro Makino, Toshikazu Ebisuzaki, and Daiichiro Sugimoto. 1996. A highly parallelized special-purpose computer for many-body simulations with an arbitrary central force: MD-GRAPE. *The Astrophysical Journal* 468 (1996), 51.

[18] Aravindhan Ganesan, Michelle L. Coote, and Khaled Barakat. 2017. Molecular dynamics-driven drug discovery: leaping forward with confidence. *Drug Discovery Today* 22, 2 (2017), 249–269. https://doi.org/10.1016/j.drudis.2016.11.001

[19] Jens Glaser, Trung Dac Nguyen, Joshua A. Anderson, Pak Lui, Filippo Spiga, Jaime A. Millan, David C. Morse, and Sharon C. Glotzer. 2015. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Computer Physics Communications* 192 (2015), 97 – 107. https://doi.org/10.1016/j.cpc.2015.02.028

[20] Y. Gu, T. VanCourt, and M.C. Herbordt. 2006. Accelerating Molecular Dynamics Simulations with Configurable Circuits. *IEE Proceedings on Computers and Digital Technology* 153, 3 (2006), 189–195. doi: 10.1049/ip-cdt:20050182.

[21] Y. Gu, T. VanCourt, and M.C. Herbordt. 2006. Improved Interpolation and System Integration for FPGA-Based Molecular Dynamics Simulations. In *2006 International Conference on Field Programmable Logic and Applications*. 21–28. doi: 10.1109/ FPL.2006.311190.

[22] Y. Gu, T. VanCourt, and M.C. Herbordt. 2008. Explicit Design of FPGA-Based Coprocessors for Short-Range Force Computation in Molecular Dynamics Simulations. *Parallel Comput.* 34, 4-5 (2008), 261–271. doi: 10.1016/j.parco.2008.01.007.

[23] S. Handagala, M.C. Herbordt, and M. Leeser. 2021. OCT: The Open Cloud FPGA Testbed. In *31st International Conference on Field Programmable Logic and Applications (FPL)*. doi: TBD.

[24] Derek Jones, Jonathan E Allen, Yue Yang, William F Drew Bennett, Maya Gokhale, Niema Moshiri, and Tajana S Rosing. 2022. Accelerators for classical molecular dynamics simulations of biomolecules. *Journal of chemical theory and computation* 18, 7 (2022), 4047–4069.

[25] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association.

[26] Rafał Kiełbik, Krzysztof Hałagan, Witold Zatorski, Jarosław Jung, Jacek Ulański, Andrzej Napieralski, Kamil Rudnicki, Piotr Amrozik, Grzegorz Jabłoński, Dominik Stożek, Piotr Polanowski, Zbigniew Mudza, Joanna Kupis, and Przemysław Panek. 2018. ARUZ — Large-scale, massively parallel FPGA-based analyzer of real complex systems. *Computer Physics Communications* 232 (2018), 22–34. https://doi.org/10.1016/j.cpc.2018.06.010

[27] V. Kindratenko and D. Pointer. 2006. A Case Study in Porting a Production Scientific Supercomputing Application to a Reconfigurable Computer. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*. 13–22.

[28] V. Krishnan, O. Serres, and M. Blocksome. 2021. Configurable Network Protocol Accelerator (COPA). *IEEE Micro* 41, 1 (2021).

[29] A. Lawande, A. George, and H. Lam. 2016. Novo-G#: a multidimensional torus-based reconfigurable cluster for molecular dynamics. *Concurrency and Computation: Practice and Experience* 28, 8 (2016).

[30] Xuewei Liu, Danfeng Shi, Shuangyan Zhou, Hongli Liu, Huanxiang Liu, and Xiaojun Yao. 2018. Molecular dynamics simulations and novel drug discovery. *Expert Opinion on Drug Discovery* 13, 1 (2018), 23–37. https://doi.org/10.1080/17460441.2018.1403419 arXiv:https://doi.org/10.1080/17460441.2018.1403419 PMID: 29139324.

[31] Johannes Menzel, Christian Plessl, and Tobias Kenter. 2021. The Strong Scaling Advantage of FPGAs in HPC for N-Body Simulations. *ACM Transactions Reconfigurable Technology and Systems* 15, 1, Article 10 (nov 2021), 30 pages. https://doi.org/10.1145/3491235

[32] A. Mondigo, T. Ueno, K. Sano, and H. Takizawa. 2020. Comparison of Direct and Indirect Networks for High-Performance FPGA Clusters. In *ARC 2020. Lecture Notes in Computer Science, vol 12083*, F. Rincon, J. Barba, H. So, P. Diniz, and

J. Caba (Eds.). Springer. 10.1007/978-3-030-44534-8_24.

[33] Gentaro Morimoto, Yohei M Koyama, Hao Zhang, Teruhisa S Komatsu, Yousuke Ohno, Keigo Nishida, Itta Ohmura, Hiroshi Koyama, and Makoto Taiji. 2021. Hardware acceleration of tensor-structured multilevel ewald summation method on MDGRAPE-4A, a special-purpose computer system for molecular dynamics simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[34] Jérémie Mortier, Christin Rakers, Marcel Bermudez, Manuela S. Murgueitio, Sereina Riniker, and Gerhard Wolber. 2015. The impact of molecular dynamics on drug design: applications for the characterization of ligand–macromolecule complexes. *Drug Discovery Today* 20, 6 (2015), 686 – 702. https://doi.org/10.1016/j.drudis.2015.01.003

[35] Asher Mullard. 2014. New drugs cost US $2.6 billion to develop. *Nature reviews. Drug discovery* 13, 12 (2014), 877.

[36] Tetsu Narumi, Yousuke Ohno, Noriaki Okimoto, Atsushi Suenaga, Ryoko Yanai, and Makoto Taiji. 2006. A high-speed special-purpose computer for molecular dynamics simulations: MDGRAPE-3. In *NIC Workshop*, Vol. 34. 29–36.

[37] Itta Ohmura, Gentaro Morimoto, Yousuke Ohno, Aki Hasegawa, and Makoto Taiji. 2014. MDGRAPE-4: a special-purpose computer system for molecular dynamics simulations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 372, 2021 (2014), 20130387.

[38] Szilárd Páll, Artem Zhmurov, Paul Bauer, Mark Abraham, Magnus Lundborg, Alan Gray, Berk Hess, and Erik Lindahl. 2020. Heterogeneous parallelization and acceleration of molecular dynamics simulations in GROMACS. *The Journal of Chemical Physics* 153, 13 (2020), 134110.

[39] C. Pascoe, L. Stewart, B.W. Sherman, V. Sachdeva, and M.C. Herbordt. 2020. Execution of Complete Molecular Dynamics Simulations on Multiple FPGAs. In *IEEE High Performance Extreme Computing Conference*.

[40] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kale, and K. Schulten. 2005. Scalable Molecular Dynamics with NAMD. *Journal Computational Chemistry* 26 (2005), 1781–1802.

[41] C. Plessl. 2018. Bringing FPGAs to HPC Production Systems and Codes. In *H2RC'18 workshop at Supercomputing (SC'18)*. doi: 10.13140/RG.2.2.34327.42407.

[42] Outi MH Salo-Ahen, Ida Alanko, Rajendra Bhadane, Alexandre MJJ Bonvin, Rodrigo Vargas Honorato, Shakhawath Hossain, André H Juffer, Aleksei Kabedev, Maija Lahtela-Kakkonen, Anders Støttrup Larsen, et al. 2020. Molecular dynamics simulations in drug discovery and pharmaceutical development. *Processes* 9, 1 (2020), 71.

[43] Outi M. H. Salo-Ahen, Ida Alanko, Rajendra Bhadane, Alexandre M. J. J. Bonvin, Rodrigo Vargas Honorato, Shakhawath Hossain, André H. Juffer, Aleksei Kabedev, Maija Lahtela-Kakkonen, Anders Støttrup Larsen, Eveline Lescrinier, Parthiban Marimuthu, Muhammad Usman Mirza, Ghulam Mustafa, Ariane Nunes-Alves, Tatu Pantsar, Atefeh Saadabadi, Kalaimathy Singaravelu, and Michiel Vanmeert. 2021. Molecular Dynamics Simulations in Drug Discovery and Pharmaceutical Development. *Processes* 9, 1 (2021). https://doi.org/10.3390/pr9010071

[44] M. Schaffner and L. Benini. 2018. *On the Feasibility of FPGA Acceleration of Molecular Dynamics Simulations*. Technical Report. ArXiv:1808.04201.

[45] R. Scrofano, M.B. Gokhale, F. Trouw, and V.K. Prasanna. 2008. Accelerating Molecular Dynamics Simulations with Reconfigurable Computers. *IEEE Trans. Parallel and Distributed Systems* 19, 6 (2008), 764–778.

[46] D.E. Shaw. 2005. A Fast, Scalable Method for the Parallel Evaluation of Distance-Limited Pairwise Particle Interactions. *Journal Computational Chemistry* 26, 13 (2005), 1318–1328.

[47] David E Shaw, Peter J Adams, Asaph Azaria, Joseph A Bank, Brannon Batson, Alistair Bell, Michael Bergdorf, Jhanvi Bhatt, J Adam Butts, Timothy Correia, et al. 2021. Anton 3: twenty microseconds of molecular dynamics simulation before lunch. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11.

[48] David E Shaw, Martin M Deneroff, Ron O Dror, Jeffrey S Kuskin, Richard H Larson, John K Salmon, Cliff Young, Brannon Batson, Kevin J Bowers, Jack C Chao, et al. 2008. Anton, a special-purpose machine for molecular dynamics simulation. *Commun. ACM* 51, 7 (2008), 91–97.

[49] David E Shaw, JP Grossman, Joseph A Bank, Brannon Batson, J Adam Butts, Jack C Chao, Martin M Deneroff, Ron O Dror, Amos Even, Christopher H Fenton, et al. 2014. Anton 2: raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 41–53.

[50] J. Sheng, B. Humphries, H. Zhang, and M.C. Herbordt. 2014. Design of 3D FFTs with FPGA Clusters. In *IEEE High Performance Extreme Computing Conference*. doi: 10.1109/ HPEC.2014.7040997.

[51] J. Sheng, C. Yang, A. Caulfield, M. Papamichael, and M.C. Herbordt. 2017. HPC on FPGA Clouds: 3D FFTs and Implications for Molecular Dynamics. In *27th International Conference on Field Programmable Logic and Applications*. doi: 10.23919/ FPL.2017.8056853.

[52] Fadi N. Sibai. 1998. The hyper-ring network: a cost-efficient topology for scalable multicomputers. In *ACM Symposium on Applied Computing*.

[53] M. Snir. 2004. A Note on N-Body Computations with Cutoffs. *Theory of Computing Systems* 37 (2004), 295–318.

[54] Ryutaro Susukita, Toshikazu Ebisuzaki, Bruce G Elmegreen, Hideaki Furusawa, Kenya Kato, Atsushi Kawai, Yoshinao Kobayashi, Takahiro Koishi, Geoffrey D McNiven, Tetsu Narumi, et al. 2003. Hardware accelerator for molecular dynamics: MDGRAPE-2. *Computer Physics Communications* 155, 2 (2003), 115–131.

[55] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton. 2022. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Comp. Phys. Comm.* 271 (2022), 108171. https://doi.org/10.1016/j.cpc.2021.108171

[56] C. Wu, S. Bandara, T. Geng, A. Guo, P. Haghi, W. Sherman, V. Sachdeva, and M.C. Herbordt. 2022. Optimized Mappings for Symmetric Range-Limited Molecular Force Calculations on FPGAs. In *International Conference on Field-Programmable Logic and Applications*. DOI: 10.1109/FPL57034.2022.00026.

[57] C. Wu, T. Geng, S. Bandara, C. Yang, V. Sachdeva, W. Sherman, and M.C. Herbordt. 2021. 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)*. doi: 10.1109/FCCM51124.2021.00024.

[58] C. Wu, T. Geng, V. Sachdeva, W. Sherman, and M.C. Herbordt. 2020. A Communication-Efficient Multi-Chip Design for Range-Limited Molecular Dynamics. In *2020 IEEE High Performance extreme Computing Conference (HPEC)*.

[59] Q. Xiong and M.C. Herbordt. 2017. Bonded Force Computations on FPGAs. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 72–75. doi: 10.1109/ FCCM.2017.49.

[60] C. Yang, T. Geng, T. Wang, R. Patel, Q. Xiong, A. Sanaullah, C. Lin, V. Sachdeva, W. Sherman, and M.C. Herbordt. 2019. Fully Integrated FPGA Molecular Dynamics Simulations. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–31. doi: 10.1145/ 3295500.3356179.

[61] C. Yang, T. Geng, T. Wang, J. Sheng, C. Lin, V. Sachdeva, W. Sherman, and M.C. Herbordt. 2019. Molecular Dynamics Range-Limited Force Evaluation Optimized for FPGA. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 263–271. doi: 10.1109/ ASAP.2019.00016.

[62] C. Yang, J. Sheng, R. Patel, A. Sanaullah, V. Sachdeva, and M.C. Herbordt. 2017. OpenCL for HPC with FPGAs: Case Study in Molecular Electrostatics. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–8. doi: 10.1109/ HPEC.2017. 8091078.

[63] Ming Yuan, Qiang Liu, Quan Deng, Shengye Xiang, Lin Gan, Jinzhe Yang, Xiaohui Duan, Haohuan Fu, and Guangwen Yang. 2022. FPGA-Accelerated Tersoff Multi-body Potential for Molecular Dynamics Simulations. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, Lin Gan, Yu Wang, Wei Xue, and Thomas Chau (Eds.). Springer Nature Switzerland, Cham, 17–31.

[64] Jiansong Zhang, Yongqiang Xiong, Ningyi Xu, Ran Shu, Bojie Li, Peng Cheng, Guo Chen, and Thomas Moscibroda. 2017. The Feniks FPGA operating system for cloud computing. In *Proceedings of the 8th Asia-Pacific Workshop on Systems*. 1–7.

[65] Hongtao Zhao and Amedeo Caflisch. 2015. Molecular dynamics in drug design. *European journal of medicinal chemistry* 91 (2015), 4–14.

Appendix: Artifact Description/Artifact Evaluation

ARTIFACT DOI

DOI: 10.5281/zenodo.8084526

ARTIFACT IDENTIFICATION

Abstract

FASDA currently runs on FPGAs connected to a network switch. The FPGAs hosts are coordinated with *dask* software, which aims to scale the work on python and perform parallel computing. We use dask to control all nodes on a single host. Furthermore, the host-FPGA interface is written with *pynq*, which allows configuring host control using python. For data transfer, *AXI-Stream* protocol is used, while *AXI-Lite* is used for debugging and key result dumping such as the number of overall operation cycles.

REPRODUCIBILITY OF EXPERIMENTS

To reproduce all FPGA results in figure 16-18, the design is compiled with the following commands:

$ cd /FPGA_MD/InterFPGA_MD
$ ./compile.sh 222 444

This configures the system for 2x2x2 cells per FPGA, and 4x4x4 cells in total. The other configurations in the experiments are also supported. This step is from 5 hours to 12 hours, depending on the size of the design. All the remaining steps should only take a few minutes and are further introduced in Artifact Evaluation.

To reproduce the CPU and GPU results, the steps are listed below, and a few minutes are estimated for each step:

(1) Install OpenMM 7.5.1 following their official guidelines: *https://github.com/openmm/openmm/releases*
(2) Go to the git cloned path.
$ cd FPGA_MD/InterFPGA_MD/CPU_GPU_reproduce
(3) For CPU experiments, first determine the number of threads to be used:
$ export OPENMM_CPU_THREADS=x
Replace x with the number of threads desired.
(4) Modify *X_DIM*, *Y_DIM*, and *Z_DIM*, number of simulation steps, and the input pdb file path in simulatePdb.py to desired configuration, then run:
$ python simulatePdb.py CPU 0
The time spent running the simulation will be shown in the terminal in microseconds.
(5) To run GPU experiments, run:
$ python simulatePdb.py CUDA x
Replace x with the number of GPUs desired. The time result will be shown in the terminal in microsecond as well.

ARTIFACT DEPENDENCIES REQUIREMENTS

Hardware environments:

We run all FPGA experiments on CloudLab with 1, 2, 4, or 8 Xilinx Alveo U280 FPGAs connected to a Dell Z9100 ON 100 GbE switch. Specifically, from Host 1 to Host 8 as the figure at https://github.com/OCT-FPGA/OCT-Tutorials shows. CPU and GPU experiments are run on Boston University Shared Computing Cluster (SCC) using Intel Xeon Gold 4262 with up to 32 threads, two Nvidia A100-40GB on an NVLink, and four Nvidia V100-16GB all-to-all connected with NVLinks.

Software environment for FPGA hosts:

- Operating System: Ubuntu 18.04
- vivado/vitis tool version 2021.2
- gcc 7.5.0
- python 3.8
- click 7+
- dask 2.9.2
- pynq 2.8.0.dev0
- ipython
- numpy 1.19

Software environment for CPU and GPU experiments:

- Operating System: CentOS 7.9
- gcc 8.3.0
- CUDA 11.1
- OpenMM 7.5.1

The data sets are protein data bank (pdb) files generated by python script, with custom force field that only enables Lennard-Jones forces. The pdb files contain neutral sodium atoms in vacuum with the number of atoms adjusted to the space size. Some early code and results, as well as the bitstream file for 8 boards are released at https://github.com/ChunshuWu/FPGA-MD.

ARTIFACT INSTALLATION DEPLOYMENT PROCESS

After the 5~12 hours compilation, the following steps are carried out.

(1) Request a desired number of FPGA nodes on CloudLab with oct-u280 profile in OCTFPGA project, or use an Alveo U280 FPGA cluster with the same configuration as CloudLab. The details are further provided in Artifact Evaluation. and node booking mentioned earlier, we upload the local files to the CloudLab server.
(2) *$ ./upload.sh pcxxx*
This is to upload the bitstream file together with host code written using pynq, a python library. The xxx is the ID number of an FPGA host. This step needs to be repeated such that the files are uploaded to all nodes. Note that the credentials in "upload.sh" need to be filled.
(3) Open a terminal for each node on CloudLab, then:
$ cd Notebooks
$ ./config.sh
This bash script automatically downloads the dependencies such as pynq, and configures the environment. This step is to be done for all nodes.
(4) Next, pick a node as the scheduler using dask. Note that dask is only used for hosts, and the FPGAs are run independently once hosts are all set.

$ dask-scheduler

(5) Copy the address starting with "tcp", then open a terminal for each node, run:
$ dask-worker <copied_address>

(6) Open a terminal on any node, run:
$ python run.py <copied_address> <dump_group> <num_iterations>
<dump_group> is to select the group of cells to dump data for demonstration, and <num_iterations> is the number of iterations to be executed. The particle information can be configured in MD_ctrl.py. The AXI-lite signals including the overall execution cycles, the execution cycles of each key component, and the communication statistics, together correspond to the results illustrated in the figures.
Specifically, *out_traffic_packets_pos, out_traffic_packets_frc, in_traffic_packets_pos, in_traffic_packets_frc* give the communication workload in 512-bit packets, *operation_cycle_cnt* shows the overall performance in cycles, *PE_cycle_cnt* and other cycle counters show the number of cycles a key component is active. The results should be the same as reported when converted from number of cycles to μs/day simulation rate.