

FPGA-Accelerated Range-Limited Molecular Dynamics

Chunshu Wu, Chen Yang, Sahan Bandara, Tong Geng, Anqi Guo, Pouya Haghi, Ang Li, and Martin Herbordt

Abstract—Long timescale Molecular Dynamics (MD) simulation of small molecules is crucial in drug design and basic science. To accelerate a small data set that is executed for a large number of iterations, high-efficiency is required. Recent work in this domain has demonstrated that among COTS devices only FPGA-centric clusters can scale beyond a few processors. The problem addressed here is that, as the number of on-chip processors has increased from fewer than 10 into the hundreds, previous intra-chip routing solutions are no longer viable. We find, however, that through various design innovations, high efficiency can be maintained. These include replacing the previous broadcast networks with ring-routing and then augmenting the rings with out-of-order and caching mechanisms. Others are adding a level of hierarchical filtering and memory recycling. Two novel optimized architectures emerge, together with a number of variations. These are validated, analyzed, and evaluated. We find that in the domain of interest speed-ups over GPUs are achieved. The potential impact is that this system promises to be the basis for long timescale MD with commodity clusters.

Index Terms—FPGA, Micro-architecture, Molecular Dynamics

1 INTRODUCTION

MOLECULAR Dynamics (MD) is a computer simulation technique that executes iteratively over discrete, infinitesimal time intervals. At MD runtime, particles inside a simulation box interact mutually as subjected to the Newtonian physics. Subsequently, after each time interval, particle states (position, velocity, etc.) are updated given their previous states and the results of the particle interactions.

Due to the brevity of the time interval, typically on the order of femtoseconds (10^{-15} seconds), carrying out simulations for just a few nanoseconds (10^{-9} seconds) in real life necessitates millions of iterations, making even 100 nanosecond-scale simulations “long.” Long timescale simulations enable the observation of collective behaviors resulting from microscopic interactions, such as protein folding, allowing for predictions of reactions and verification of real-life observations, and are therefore a vital tool in a wide variety of physical simulations.

One of the most important benefits of observing collective behaviors is the ability to confirm whether a drug candidate can be applied to a target, which is essential in drug development. MD has been used, for instance, to identify COVID-19 protein-ligand interaction sites as potential drug targets and to locate inhibitors of the main protease [1], [2], [3], [4]. A vast array of research investigating protein-ligand binding using MD is also currently underway [5], [6], [7].

In drug design, space and time differ significantly in scale. When it comes to space, simulations involving small

sets of particles are especially valuable since drugs typically consist of small molecules with fewer than 50 atoms (as per Lipinski’s rule of five), and the target site molecules are also relatively compact (such as the main protease of COVID-19, which contains approximately 2000 atoms). Regarding time, simulation duration in drug design usually ranges from hundreds of nanoseconds to microseconds, as it can take tens of nanoseconds for the protein and drug candidate to attain structural stability [8], and even longer to achieve overall convergence. Consequently, the combination of small space and long simulation duration results in limited data parallelism, necessitating the use of efficient computing methods.

From the long timescale simulations the demand for high performance computing (HPC) arises. Aiming at faster MD simulations, a variety of software packages have been developed [9], [10], [11], [12], [13], with many supporting GPU acceleration. But while GPU-based systems excel at handling large-scale simulations, for the small datasets critical in a number of domains, there is often sub-optimal scalability [14], [15], [16]. An alternative approach is to develop systems based on application-specific integrated circuits (ASICs) optimized for MD. For example, Anton 3 can achieve over 200 μ s-per-day for 10K particles with 512 nodes. But while ASIC-based solutions can have an order-of-magnitude better performance than commodity clusters, they may also have issues with general availability, plus problems inherent with small-run ASIC-based systems.

A more general solution is to create MD simulation systems using clusters accelerated with commercial off-the-shelf (COTS) integrated circuits (ICs), namely, field-programmable gate arrays (FPGAs). FPGAs are unique among COTS ICs in their support for communication, through both the large number of high-speed transceivers and the ability to couple the application-level processing with those transceivers in a small number of cycles. These

- Chunshu Wu, Sahan Bandara, Anqi Guo, Pouya Haghi and Martin Herbordt are with the Department of Electrical and Computer Engineering, Boston University.
E-mail: {happywu|sahan|anqiguo|haghi|herbordt}@bu.edu
- Tong Geng was with the ECE Department at Boston University and is now with the ECE Department at the University of Rochester.
- Chen Yang was with the ECE Department at Boston University and is now with Citadel LLC.
- Ang Li is with Pacific Northwest National Laboratory.

enable FPGA-accelerated clusters to approach ASIC clusters for performance of the Long Range force computation (LR) [17], [18], [19], the part of MD that is most difficult to scale. The critical question has been, therefore, whether a *single* FPGA MD engine can be sufficiently competitive, and, if so, how should it be implemented?

In classical MD, the computation is dominated by non-bonded forces, which can be divided into two components: range-limited (RL) and long-range (LR). RL is computing-intensive and consists of 90% of the computation, while LR is more memory and communication oriented. These two components are relatively independent in terms of data flow and can be considered as separate tasks. This work specifically focuses on the RL component. Despite over nearly two decades of research on FPGA-MD, there is still much work to be done. With the growing on-chip resources, it is now possible to implement the entire MD engine on an FPGA, thus avoiding frequent CPU-FPGA data transfers. However, this presents a new system-level challenge of how to map particles to processing elements (PEs) effectively. In addition, the number of force processors has significantly increased from 8-10 to over a hundred, which can worsen the already heavily coupled data structures, leading to issues such as routing complexity and maintaining operating frequency. Thus, a question arises as to how to ensure scalability in the design for hundreds of PEs on-chip, each of which sources and sinks dozens of particle pairs every cycle.

To address the particle-to-PE mapping problem, the simulation space is first partitioned into cells using the cell-list method [20], [21]: data is grouped into boxes for both parallel processing and to reduce the complexity from $O(N^2)$ to $O(N)$. Applying the Newton's third law optimization [23], e.g., through the half-shell method [22], further halves the number of computations. This leaves three high-level mapping schemes to be considered: all PEs working on the same particle [23.1], all PEs working on different particles in the same cell [23.2], and each PE working on a different cell [23.3].

This brings us to the essence of the single-FPGA (or ASIC) scalability problem: after the high-level mapping scheme is determined, how can a scalable data path from memory to PEs be constructed. Directly connecting memory to PEs in a 3-D application that is mapped to a 2-D FPGA fabric leads to a complex interconnect between PEs and memory, resulting in a significant decrease in frequency. For instance, in the 512-node (8x8x8) Anton 2 machine, even with its direct 3-D torus topology to deploy such a 3-D application, the longest connection between two nodes is still almost two meters [23]. On the other hand, in a single FPGA, it is not practical to directly map 3-D application to 2-D structure due to space and wiring constraints. There, a transposed memory block method and ring routing are proposed and proved adequate for RL force evaluations (as shown in the following discussions [5]).

Routing in the ring method, however, requires more cycles and results in bubbles in the network limiting throughput. To compensate, a data caching method is developed to minimize data transfer demands by boosting data locality [5.2.2]; to coordinate the PEs, instead of applying bulk synchronization, we propose a novel dynamic data dispatching

method to allow PEs to work continuously without frequent synchronizations [5.2.3]; an out-of-order data broadcast mechanism is developed to dynamically fill the empty slots in the ring to remove bubbles [5.2.5].

A crucial aspect of these designs is to maintain compatibility with a third standard optimization: prefiltering particle pairs (within neighboring cells) to eliminate where the mutual force is negligible [24] (analogous to neighbor lists in CPU implementations). Theoretically, for evenly distributed particles, the average pass rate for a filter is 15.5%, and $\sim 17\%$ for a planar filter [25]. In this work, we upgrade the conventional filters to hierarchical filters [5.2.5]. This method not only increases the pass rate to 25%, but also alleviates the pressure in data transfer.

To summarize this introduction: a new generation FPGA-based MD RL accelerator is proposed with the following major contributions.

- An optimized FPGA-based MD RL accelerator with several design variations that can process 50K particles without off-chip memory.
- On system-level, three particle/cell to PE mapping schemes are proposed and evaluated in depth.
- The ring routing method is studied to match the 3-D to 2-D mapping of the up-scaling number of PEs on a single FPGA chip. With an out-of-order data broadcast mechanism and a data caching mechanism applied, the latency and concurrency problem introduced by the ring are solved.
- Other optimizations, for example, the hierarchical filtering method and memory partitioning mechanism are proposed; they reduce hardware consumption and improve hardware efficiency.

2 BACKGROUND

2.1 Physical Models of Range-Limited Forces

RL force features. The RL forces consists of two components: the short range term of electrostatic force typically obtained using the Particle Mesh Ewald (PME) [26] or related method, and the force deduced from Lennard-Jones (LJ) potential.

The LJ potential is an empirical model of the Van der Waals interaction between electrically neutral particles. The LJ potential between particle i and j with distance r_{ij} is given in the following equation:

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

The potential is determined by ϵ , the dispersion energy describes the potential amplitude, and σ , the characteristic particle distance at zero LJ potential. Take the gradient of the potential and we obtain the RL force between particle i and j :

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

Equation [2] indicates that the complexity of RL is $O(N^2)$. It can be observed, however, that the force decays rapidly with r_{ij} , meaning that distant particles contribute negligibly to the force. Therefore a cutoff radius (R_c) is introduced: for a particle pair with $r_{ij} > R_c$ the force is set to zero with the consequence that that pairwise force computation can

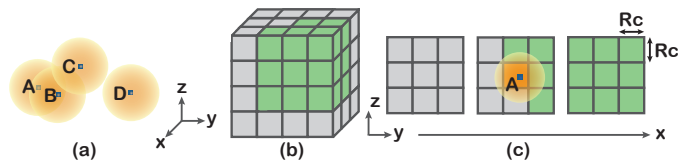


Fig. 1. Fundamentals of R_c and the cell list method. (a) The cutoff regions of four particles. Only A and B interact with each other. (b) A simulation space divided into $3 \times 4 \times 4$ cells with side length the same as R_c . (c) The home cell of particle A and 26 adjacent cells unfolded from (b). The green cells are regarded as neighbor cells.

be ignored. To visualize, R_c is represented with halos in Figure 1(a). Particle A only interacts with particle B because A is in B's halo and vice versa. Particle C and D contribute little to the resultant force of A, therefore are ignored in A's computation. The overall computing complexity then becomes $O(mN)$, where m is the average number of a particle's neighbor particles. Typically, $m \ll N$.

Furthermore, the total computation can be reduced by a half, for the pairwise forces take effect on both particles in a pair, according to Newton's 3rd law.

Periodic boundary condition. Periodic boundary conditions are frequently used in solid state and computational physics: a large homogeneous system can be modelled into replicas of a small system. That is, a particle exiting the small system is equivalent to a same particle with the same velocity entering from the opposite end.

2.2 RL Procedure

The general procedure includes two phases, **force evaluation** and **motion update**, which are executed iteratively. In force evaluation, based on the current particle position, pair-wise forces are computed using Equation 2 and then accumulated to obtain the resultant force for each particle. After all forces are computed, the resultant forces are processed in motion update units to calculate velocity and position differences from current particle states, following Verlet Integration with $O(\Delta t^2)$ error introduced:

$$\mathbf{a}(t) = \frac{\mathbf{F}(t)}{m} \quad (3)$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \frac{\mathbf{a}(t) + \mathbf{a}(t + \Delta t)}{2} \Delta t \quad (4)$$

$$\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \mathbf{v}(t + \Delta t) \Delta t \quad (5)$$

Compared with force evaluation, motion update is much less computationally intensive as it is applied only once per particle. There is a concern, however, with particle migration resulting from particles entering and exiting partitioned regions. The handling of particle migration is explained in detail in § 4 and § 5.

2.3 Data Structure: Neighbor Lists vs. Cell Lists

Neighbor lists. Particle data can be organized into neighbor lists and cell lists. A neighbor list is a list of neighbor particles regarding a reference particle. It allows quick data lookup when traversing the neighbor particles. However, the data locality can hardly be exploited for the neighbor lists are highly decoupled with space. As a result, high memory bandwidth is demanded to compensate for the limited data

locality and so this method is generally replaced by filtering in hardware implementations.

Cell lists. A cell list allows particles to be grouped spatially before being evaluated. In Figure 1(b), the simulation space is partitioned into cubic cells (say, $3 \times 4 \times 4$) with side length R_c . The particles in each cell are stored in a separate memory domain. In the $3 \times 4 \times 4$ case, 48 memory regions in total are allocated for particle storage, i.e., each cell has a list of particles. Assuming particle A is located anywhere in cell (1, 2, 2), only the particles in the green surrounding neighbor cells (including the home cell itself) are checked with particle A for valid particle pairs. With Newton's 3rd law applied, only 14 out of all 27 surrounding cells are considered as neighbor cells (in the half-shell method), as Figure 1(c) illustrates. Other layouts for importing neighbor data are discussed in § 2.4. The cell lists geometrically categorize particles for high data locality. In other words, a particle can be broadcast to its neighbor cells for pairing. However, theoretically, only $\sim 15\%$ of the neighbor particles form valid pairs with the reference particle according to the equation below:

$$P = \frac{\frac{4}{3}\pi R_c^3}{27R_c^3} = 15.5\% \quad (6)$$

As a result, preliminary particle pair filtering is desired; a force compute unit receives particle pairs from multiple filters to avoid being idle.

For hardware implementations, locality out-weighs resources. That is, locality can be exploited to avoid hundreds of data transactions per particle, while filtering only consumes a small amount of on-chip resources [25], especially with planar filtering, which uses only comparisons of low-precision integers. Therefore throughout this work cell lists and planar filtering are used.

2.4 Neighbor Data Importing Layout

The half-shell method is only one of the feasible methods that handle import of neighbor data. The Neutral Territory (NT) method is used in Anton 1 and 2 [27], [28], and the Hybrid Manhattan method is used in Anton 3 [29]. However, unlike ASICs, FPGAs rely on block RAMs (BRAMs) for data storage. Each BRAM nowadays typically has hundreds of entries in depth and tens of bits in width, and is therefore suitable of storing data chunks; this is in contrast to the data fragments that require fine-grained memory access and allocation. Compared with the NT and Manhattan methods, with the half-shell method data are only accessed by cells and are easily organized. In summary, although the half-shell method has a greater import volume, it is more hardware-friendly for FPGAs. Moreover, the redundant import volume can be removed by inexpensive filters.

3 HIGH LEVEL MAPPING SCHEMES

In this Section, we discuss three distribution schemes for mapping work from cells to PEs. A PE is defined as a force computing unit and its associated logic including filters and accumulators. The particle data in cells are stored in memories abstracted as HC (position caches in home cells), NC (position caches in neighbor cells), and Force Caches

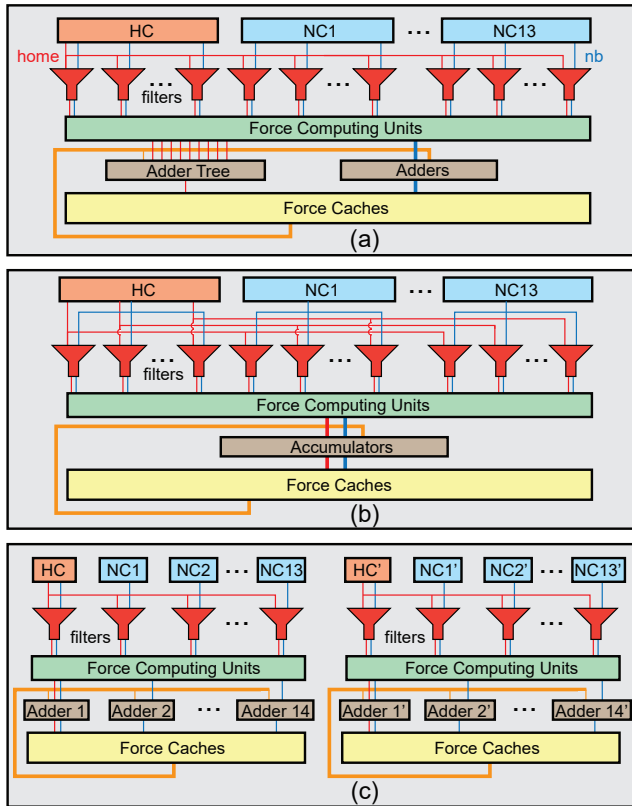


Fig. 2. High level memory-to-PE mapping schemes. (a): particle-centric, all PEs work on a single reference particle at a time. (b): cell-centric, all PEs work on a same cell at a time. (c): uniformly-spread, each PE works on a separate cell.

(contain forces of particles in individual cells). Here we disregard the data paths related to motion update units as they only read from force caches in cells and write to position and velocity caches in the same cells and add little complexity.

3.1 All PEs Work on the Same Reference Particle

In Figure 2(a), one single reference particle from the home cell is broadcast to all filters in all PEs. In the meantime, the particles from neighbor cells are sent to filters to pair with the reference particle.

This mapping method allows the partial forces of the reference particle to be accumulated with an adder tree before being stored in the force cache, reducing the traffic in force return. The partial forces of the neighbor particles are individually accumulated into the force caches by the adders associated with cells. This method also achieves workload balance on particle pair level.

However, the disadvantages are obvious. First, the cost for maintaining data locality is high. Although the neighbor particles fetched can be reused for all the particles broadcast from the home cell, the number of neighbor particles is too high (~ 1000 , assuming we have enough PEs, ~ 150) for the particles to be cached in registers for concurrent reading. Second, all the neighbor force fragments are returned to only 14 neighbor cells, which requires tremendous bandwidth per cell to perform the accumulation without conflict. Even if the particles are interleaved in memory to alleviate bandwidth problem, the maintenance cost is high.

3.2 All PEs Work on the Same Home Cell

Figure 2(b) shows that the home cell broadcasts multiple particles to the filters. Meanwhile, each neighbor cell only needs to broadcast one particle to the filters.

As the spatial locality is achieved for neighbor cells, the demand for bandwidth is drastically decreased: instead of traversing neighbor particles for a fixed set of home particles, we traverse the home particles for a fixed set of neighbor particles instead. The neighbor particles are then cached in registers and no neighbor particle is required until the current set of neighbor particles is processed. As for force return, such localization also introduces the opportunity for accumulating the neighbor forces before sending them back; thus the force return bandwidth is also reduced.

Still, only 14 cells are evaluated and the rest of the cell storage remains idle, unless an expensive memory interleaving method is applied for dynamically changing memory contents. Also, a heavy bandwidth requirement persists in the broadcast of the home cell position.

3.3 Each PE Works on a Different Cell

Figure 2(c) shows two individual PEs, unlike the former cases where the PEs are blended. In this case, a PE only works on one single home cell, which broadcasts only one particle to all the filters in one PE at a time, while each of the 14 neighbor cells sends one particle to a filter in the PE. In fact, a neighbor cell broadcasts the single particle to 13 other home cells as well. For example, NC1 and NC2' in Figure 2(c) may be the same cell.

This mapping method addresses both the bandwidth and the memory idle problems. First, at most one particle is broadcast from any cell, allowing concurrent position reading without requiring a large number of BRAMs per cell. Second, now that multiple home cells can be evaluated, the PEs can be applied simultaneously across the simulation space to fully utilize the memory blocks. Third, similar to the second mapping scheme, the data transfer rate can be further reduced by traversing home particles for the neighbor particles.

4 BASELINE ARCHITECTURES

In this Section, we introduce the detailed baseline designs corresponding to the high level memory-to-PE mapping schemes above.

4.1 Design 1: Particle Centric

The design layout is depicted in Figure 3(a). Assume there are m cells in the simulation space, and each cell is equipped with a set of block RAM based caches (including a position cache, a force cache and a velocity cache). These caches contain particle information at consecutive addresses, namely, particle ID.

The force evaluation phase starts from position caches, where neighbor particle positions are distributed to separate filters to pair with a reference particle. The distribution is done by a position distributor (Figure 3(d)), which selects particle position data from 14 specific position caches, for only the particles from 14 neighbor cells can be paired with the reference particle. Meanwhile, the filters are partitioned

into 14 groups, where each group only receives neighbor particles from a single neighbor cell. Once a neighbor particle is received, it is pushed into a chain of registers as the blue arrows on the registers show. After all neighbor particles are pre-loaded into the registers, the filtering process is initiated by traversing and broadcasting the particles in a cell as reference particles to all the filters. Next, the streaming reference particles are paired with the registered neighbor particles. If a reference-neighbor particle pair is within the cutoff radius, the pair passes the filter and is buffered for force computation.

As introduced in § 3.1, ~ 1000 filters are available, which makes it possible for a reference particle to be evaluated in one cycle (normally, a cell only contains < 80 particles, 14 neighbor cells contain ~ 1000 particles, matching the number of filters). Besides, based on equation (6), a force computing unit accepts particle pairs from seven filters to keep it busy.

The force outputs from the force computing units are fed into two sets of adders. On one hand, the partial forces of a reference particle (reference forces) are accumulated with an adder tree and then accumulated into a force cache. The adder tree only processes one reference particle at a time, resulting in bubbles between the evaluations of two reference particles. On the other hand, the forces applied on neighbor particles (neighbor forces) are first registered and accumulated in separate adder groups (see Figure 3(e)) during the evaluation of all the reference particles in a cell, and then accumulated into their force caches. However, there are still ~ 1000 neighbor force chunks in total to be accumulated into 14 force caches. To alleviate the mismatch between the force throughput and the force cache bandwidth, more force caches may be put into use if there are available block RAM resources, but the key problem is: there are m force caches but only 14 are being used at a time.

4.2 Design 2: Cell Centric

The design is given in Figure 3(b), which is similar to Figure 3(a) but very much different in mechanism.

From a position cache, reference particles are broadcast to the filter groups sequentially and cached in registers. Each filter group consists of 14 filters designed to receive particle position data from 14 neighbor cells, respectively. The neighbor particle positions stream down through the filters and are paired with the registered reference particles. Still, a force computing unit is linked to seven filters.

Now that many reference particles are being processed at the same time, the adder tree in design 1 is no longer required. The reference forces are first accumulated before integrated into the force caches. As for neighbor forces, a neighbor particle is shared by many filters at a same time, making the neighbor forces able to be accumulated before writing to their force caches. Consequently, adder trees are demanded to coordinate the high throughput of the neighbor forces and the limited number of force caches.

As the input forces to an adder tree have to be from the same neighbor particle, stalls are required to make sure the neighbor forces from other particles are not dispatched to the adder trees before the previous are completed. That is to say, the neighbor particles are grouped in batches with batch size = 14, as they are from 14 neighbor cells. The force



Fig. 3. The baseline designs and distributors. (a)-(c): baseline designs based on the three high-level mapping schemes, where the buffering FIFOs and the forces read from the force caches are omitted for conciseness. The thick lines and arrows stand for buses. (d)-(e): position and force distributors. All adders are 32-bit floating point adders.

computing units only start to process the next 14 neighbor particles after they finish the previous batch entirely.

4.3 Design 3: Uniform Spread

In this design, We no longer focus on a single reference particle or a reference cell at a time, but all cells are being processed simultaneously.

In Figure 3(c), each of the m position caches broadcasts a reference particle to its designated filter group, where each group contains 14 filters. A new reference particle is not broadcast until all filter buffers of all PEs are empty. This is because, first, the position broadcast mechanism demands all PEs receive particles with the same ID – it's convenient for all PEs to start with the same particle ID, thus the global synchronization. Second, the reference position is not stored in filter buffers, but in a register (shared by the force computation unit and all the filters) for conserved filter buffer sizes. As a result, the reference position cannot be discarded until all related operations are done.

Still, a force computing unit is served by seven filters. After the reference particles are registered, the position caches start distributing neighbor particles to the filters in streams. The neighbor particles are distributed by a position distributor different from the previous two designs (see Figure 3(f)) as the mechanism of this design is substantially different.

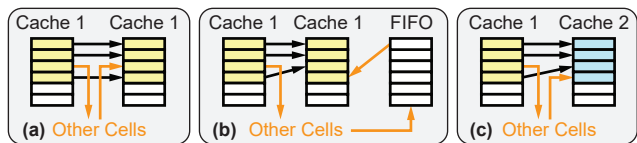


Fig. 4. Particle migration handling methods. (a): directly searching for empty slots. (b): the one-pointer method. (c): double buffering.

Once all the current reference particles are evaluated, the position caches dispatch the next reference particles to PEs.

The reference particles have good temporal locality. Therefore, the reference forces can be directly accumulated into their force caches through adders. On the other hand, the neighbor forces are highly fragmented and are flooding into force caches for accumulation. Adder trees are not used because the resources are limited as all the cells are being processed at the same time. The force distribution mechanism (Figure 3(g)) is also changed from the previous two designs for adjustment.

4.4 Motion Update and Particle Migration

All three baseline designs use cell-based cache data structure and the same motion update method.

After force evaluation, the motion update units request position, velocity, and force data from individual caches in parallel. Upon requested, the force data are sent to motion update units with their original entries in force caches erased to prepare for the next force evaluation phase.

With particle migration, we cannot simply send the updated position and velocity data back to overwrite the cache contents. The reason is shown in Figure 4(a): the particle being migrated out from Cache 1 leaves an empty slot in the cache. Without further optimization, a particle being migrated in has to find an empty slot at high costs.

A more advanced way is to use a pointer to keep track of the previous particle evaluated, see Figure 4(b). The local particles are updated first, while the particles being migrated in are stored in a FIFO in the mean time. As the 3rd particle is migrating away, the pointer points at the vacancy so the 4th particle is updated to the 3rd slot, with the pointer moving to the 4th. As a result, the local particles without migration are compactly stored in the cache. Subsequently, the particles migrated in are read from the FIFO and integrated into the cache in order.

However, the problem to the advanced method is that the hardware cost of a FIFO on an FPGA is not very different from a cache itself. Therefore, we may as well just use two caches to handle this problem, as Figure 4(c) shows. When a particle migrates in, the local update is paused and the new particle is written to Cache 2 in order. Eventually, all updated particles are stored in Cache 2 while Cache 1 is out-dated. In the next MD iteration, the data from Cache 2 are used and the particle stats are updated to Cache 1. In practice, the double buffering mechanism is applicable to a single cache with two ports (read and write), if the depth of a single cache is sufficient.

5 OPTIMIZED DESIGNS

The problems with the three baseline designs are evident. First, all suffer from complex data routing paths as indicated

by the position and force distributors in Figure 3. The high fan-in/fan-out are detrimental to frequency and the wiring resources are easily exhausted; second, the data locality is not fully exploited. Aiming to solve these two fundamental problems of the baseline designs, we put forward two optimized designs evolved from design 1 and design 3. Further benefits are also presented.

5.1 Optimized Design 1: Transposed Memory Blocks

Optimized design 1 originates from the first mapping scheme, where one reference particle is paired with all its neighbor particles. The complete design is given in Figure 5(a). Storing particles by cell, as used in baseline 1, is a good way to make data well-organized, but is not suitable for this workload mapping scheme. In the particle-centric scheme, only 14 cells are being used at a time and the performance is limited by the throughput of the particle data caches. In this case, the optimization goal is to make use of as many caches as possible to satisfy the data bandwidth requirement, and to keep the particles organized in the meantime.

The key to the solution is shown in Figure 5(b). The solution transposes the original block RAM arrays, which store particles by cells. Rather, the block RAMs now store particles by particle ID with the entries representing different cell IDs (assuming there are m cells).

5.1.1 Optimized Data Paths

The position data can now be more rapidly distributed to filters. Physically, the number of particles per cell is usually less than 80, limiting the number of block RAMs required. With each cache in charge of particles with the same particle ID from different cells, all neighbor particles with respect to one reference particle can be loaded to the registers above the filters in 14 cycles (~ 100 cycles with baseline 1 due to the limited active position caches), and reused for other reference particles from the same cell.

As Figure 5(a) shows, the position caches and filters are highly localized. Each filter group is only served by neighbor particles from the same position cache, eliminating the extremely overlapped position data paths and the huge fan-in/fan-out in baseline 1, where the position caches and filters are connected almost all-to-all.

The data paths of the neighbor forces are also simplified. As a filter group only evaluates neighbor particles with the same particle ID, a neighbor force produced is only accumulated into its designated force cache directly. Note that the conventional ration of seven filters to one force computation unit mapping is preserved.

At the same time, the locality of the reference particle is exploited. As mentioned earlier, the adder tree in baseline 1 results in bubbles that may diminish performance. Given this fact, a buffer array is inserted between the adder tree and the force computation units. Unlike the force caches, all n buffers contain reference forces from the same cell, addressed by particle ID. Once a reference particle is evaluated and its reference force fragments are fully accumulated into the buffers, the accumulated reference force pieces enter the adder tree to perform all-reduce. The reference force chunk is then accumulated into a force cache. With the help of the

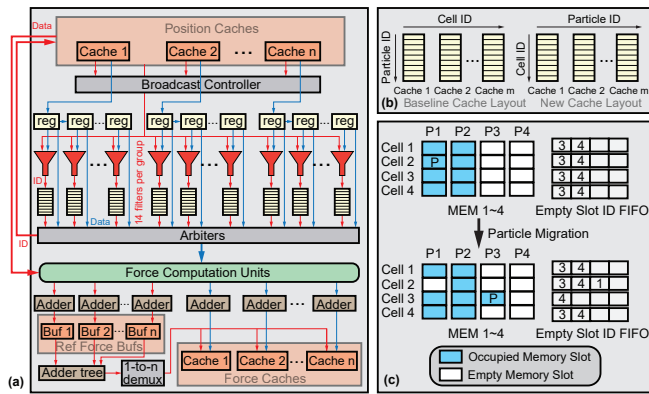


Fig. 5. The optimized architecture for the first mapping scheme. (a): the data flow in force evaluation. (b): the new memory layout compared to baseline. (c): particle migration handling in motion update. P is the particle being migrated from cell 2 to cell 3.

buffers, the force compute process of a single reference cell can be pipelined without stalls.

Another optimization during data reference is that the buffers attached to the filters now only contain neighbor cell IDs, which are much more light-weight than particle positions. Once a pair is selected by an arbiter, the neighbor cell ID is sent to the position cache for position data. The position data is then sent to the force compute unit for processing.

5.1.2 Particle Migration Handling

Now that the caches are transposed, greater opportunities emerge in contrast to the double-buffering migration handling method.

In Figure 5(c), we use four transposed caches as an example. The last two caches are initially empty to make sure no particle is lost due to migration. In practice, we use $\sim 25\%$ more caches for reservation. Assuming particle P is moving from Cell 2 to Cell 3, it checks the empty slot ID available from the empty slot ID FIFO of Cell 3 and moves it accordingly. Meanwhile, the vacancy ID is pushed to its empty slot ID FIFO. Particle P may get updated again in Cell 3 during parallel motion update, but the data integrity is not harmed because the force on particle P is 0. Note that the empty slot ID FIFOs are very small and can be constructed using a few registers.

After multiple MD iterations, similar to particle P , the empty slots (namely, vacancies) populate, and eventually are scattered among the caches. In fact, the vacancies are not detrimental but rather advantageous. At the beginning of MD, the empty caches reserved have no workload to distribute to their filters or force pipelines downstream, resulting in idle computing units. As the vacancies spread, the empty caches are gradually filled with migrating particles, such that the workload becomes more balanced as the simulation proceeds.

5.1.3 High Spatial Capacity

The demand for hardware resources increases little with more cells, even scaled to ~ 500 cells. Compared to baseline 1 and baseline 2, where the number of cells dominates the wiring complexity and resource consumption, the memory

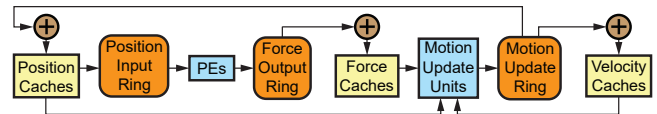


Fig. 6. RL dataflow in brief with rings. Yellow: Memory blocks; orange: Routing rings; blue: computing units.

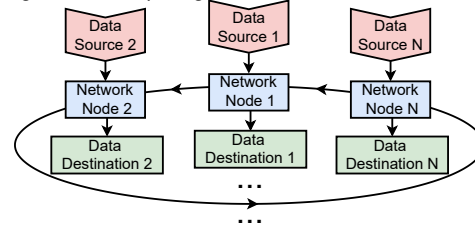


Fig. 7. The ring routing path. Position input ring: from position caches to PEs. Force output ring: from PEs to force caches. Motion update ring: from motion update units to position caches and velocity caches.

block transpose method allows easy upscaling of the number of cells. Whenever a new cell is added to the simulation space, the particles within are uniformly distributed to caches. As each cache has depth of 512, up to 512 cells can be handled with such a data structure. For the number of cells greater than 512, the design layout still holds. We can simply extend the caches with more BRAMs and slightly modify the cell selection logic to adjust.

5.2 Optimized Design 2: On-chip Ring Network

This architecture is an upgrade to the 3rd mapping scheme: each PE works on a different cell. In baseline 3, the wiring complexity increases drastically when the number of PEs is scaled up. Also, the neighbor particle data have very poor locality. The neighbor positions are not reused and neighbor forces are not aggregated before being integrated to force caches.

Targeting the routing problem, we replace the original direct connections with a ring. However, using a ring results in longer latency and longer data life in the network, leading to congestion. In this Section, we demonstrate a solution that minimizes the data transfers in the network by localizing the neighbor particle data. Also, the bubbles in pipelines and the number of invalid particle pairs can be significantly reduced.

A method that removes design redundancy is discussed. The previously used double buffers in motion update are replaced by linked lists, saving 50% of the memory for the position and velocity caches.

5.2.1 Memory-PE Interconnect

To avoid frequency degradation, we replace the direct interconnects in Figure 2 with daisy-chain-based uni-directional (shown in Figure 6). The 1-D topology is chosen over other dimensions for its high cost-efficiency, and the fact that the long latency can be hidden (as discussed later in this Section). The routing rings are used for three different types of data transfer: position caches to PEs, PEs to force caches, and motion update units to position and velocity caches.

Ring Behavior. The three rings have different behaviors. That is, a position packet is broadcast to multiple destinations and is therefore kept in the ring until it reaches the final destination. In contrast, a force or motion update

packet has only a single destination. It is worth pointing out that particles may migrate to other cells after an iteration. For this reason, a ring is required to route motion update packets. Also, the motion update ring is less congested because particle migration is relatively rare, and is therefore further details are omitted.

Figure 7 depicts the interconnect of the ring nodes. For each ring node, two data sources and two destinations are available. New data from a source can only be injected to the network when the node is not occupied. A packet in the network is discarded after its final destination is reached, otherwise it is sent to the next node.

Ring Configuration. The 3-D cell to 1-D ring mapping is yet to be described. A straightforward method follows:

$$I_r = N_{cell}^y N_{cell}^z (x - 1) + N_{cell}^z (y - 1) + z \quad (7)$$

where I_r is the index shown in Figure 7 from 1 to N , N_{cell}^x is the number of cells along x axis, and x is the coordinate of cells on x direction, ranging from 1 to N_{cell}^x . In fact, finding the optimal mapping from 3-D cells to 1-D rings is complicated. But instead of permuting all the indexing possibilities, we use the mapping method above because it overwhelmingly reduces the average packet lifetime.

5.2.2 Data Caching

By removing the direct connections, we merge the specific datapaths for cell-PE pairs into the narrow rings. As a consequence, the desired throughput cannot be satisfied. To tackle this problem, we develop a two-fold data caching method in order to reuse the input position data and to effectively aggregate the force fragments.

Neighbor Particle Caching. Instead of feeding neighbor particle data directly from neighbor caches, the neighbor data are cached in registers located on the top of the filters. That means, instead of traversing the neighbor particles for a single home cell particle (only caching a single particle), the home cell particles are traversed for multiple neighbor cell particles (caching many particles). This method significantly alleviates the pressure in data transfer from position caches to force pipelines. For example, if two neighbor particles are evaluated together with the home cell, 50% fewer transfers are required.

Another benefit of the neighbor particle caching method is a more balanced workload among force computation units. Figures 8(a) and (b) give two different scenarios regarding the filtering rate. If we traverse neighbor particles with respect to a single reference particle, (a) and (b) may happen and lead to highly imbalanced filtering. In the current scheme, multiple neighbor particles behave as reference particles and the chance of filtering imbalance is greatly reduced.

Completing the design: the neighbor force data are cached in registers (at the bottom of Figure 9(a)). The locality of neighbor particles is taken advantage of and the neighbor forces are accumulated before being sent to the force output ring. As a result, the payload on the force output ring is reduced similarly to the position input ring.

Home Particle Caching. Instead of storing the entire position data in the filter buffers below the filters, only the particle IDs are used (see Figure 9(a)) to save BRAM

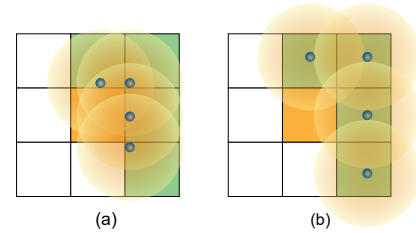


Fig. 8. Example cases of (a) high filter pass rate, (b) low filter pass rate in 2D illustration.

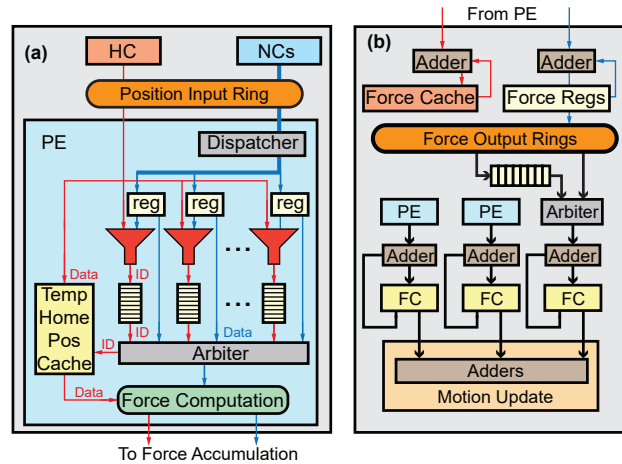


Fig. 9. The optimized design. (a): single force computation pipeline datapath. (b): force datapath, the two PEs work on a same home cell.

resources. To compensate the home cell particles are localized in the temporary home position cache so that the home particles can be accessed with particle IDs. A home particle is collected in the temporary home position cache upon arrival, and can be overwritten after the evaluation of the current home cell is finished. As a result, both home and neighbor particles are localized for arbitration and are ready to be fed into force computation units.

Merits in System Design. With data caching methods the designs in Figure 2 can be reduced in concurrency demand except (a) for its overly fine-grained data access mode. The home cells now do not need to broadcast multiple particles simultaneously as the cached data is sufficient to keep the force pipelines busy.

5.2.3 Neighbor Particle Dispatching

With the data caching methods being used, other components need to be adaptive. To properly cache the neighbor particles, a dispatch mechanism is developed. Position data from the position input ring are injected directly into the position input buffer and ready to be dispatched. An arbiter associated with the dispatcher actively checks the status of the registers below and fills new data into the registers if any is completely evaluated.

To determine if a neighbor particle has done filtering, the home particle ID is recorded upon the neighbor particle's arrival; the filtering is done when the ID is observed again.

5.2.4 Partial Force Caches

Because in this design the home particle forces are hard to cache, they are accumulated into force caches directly without going through a force output ring (see Figure 9(a) bottom and Figure 9(c)). To match the throughput of the

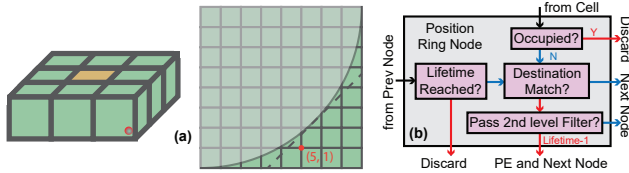


Fig. 10. (a): The 1st level filtering in a position input ring node. Dashed line: planar filter criterion. (b): The flowchart of a position input ring node.

force pipelines, more than one force cache is required for a cell when multiple force pipelines are working on a single cell (system design (b) and (d) in Figure 2).

The forces from the rings are accumulated in a dedicated force cache (Figure 9(c)). If more than one force output ring is deployed for higher concurrency, the forces are buffered and arbitrated round-robin and accumulated to the same force cache; this works because the arrival rate of forces through rings is much less than that of the forces from PEs.

The force caches that hold partial forces are aligned in memory. The forces do not need to be aggregated until motion update starts. This implies that only a small number of adders are required to aggregate the partial forces, since motion update is not compute intensive and the number of motion update units is small compared to force pipelines.

5.2.5 Position Input

The position input units upstream are also upgraded to support the force processors downstream. To be specific, the position routing and position broadcast are upgraded to enhance hardware utilization.

Hierarchical Filters. Some particles can be discarded before entering PEs. For example, Figure 10(a) shows that a neighbor particle cannot be paired with any particle in the home cell. If sent to PEs, each of such particles wastes tens of filtering cycles. It is therefore beneficial to implement a higher level filter to eliminate the “bad” particles during data transfer.

Since a particle can have multiple destinations in a position input ring, it makes sense to deploy second level filters in position input ring nodes. A second level filter is the same as a first level filter except it compares the particle position with the coordinates of the corner. With hierarchical filtering, Monte Carlo experiments show that the filtering rate of the first level filter increases from 17% (planar filter) to 25% for uniformly distributed homogeneous particles.

Figure 10(b) shows how the second level filters are integrated into position ring nodes. Each node can receive data from the previous node or the position cache, where the data from the previous node has higher priority.

Out-of-Order Position Broadcast. Higher priority is given to the previous node because the data directly from position caches can be sent again later, since the position caches are iterated repeatedly for particle filtering with cached neighbor particles. The problem is that the position caches not only provide home particle position, but also provide that of the neighbor particles. The problem is how to efficiently do both at the same time.

Figure 11 depicts the efficient out-of-order broadcast method in four representative cycles. The method allows seamless neighbor position injection during home position traversal, such that the data can be sent down-stream

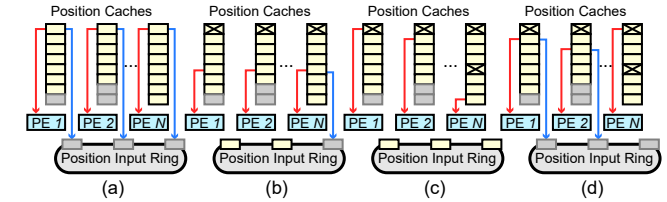


Fig. 11. The out-of-order broadcast method. Gray boxes: Empty slots. Boxes with “x”: Marked as used. (a) The 1st cycle. (b) The 5th cycle. (c) The 8th cycle. (d) the 9th cycle. Red arrows: home particle position; blue arrows: neighbor particle position.

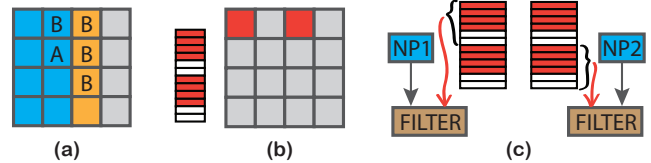


Fig. 12. The solution to the problems emerge in large scale simulations. We intuitively use 4x4 2-D cells with eight PEs for example. (a): blue: the cells currently being processed; orange: the extra cells required to process the blue cells. Particles from cell A are broadcast to all B cells. (b): memory partitioning. The red cells share the same memory block. (c): memory partitioning with double buffers. Two neighbor particles (NP1 and NP2) require data from different cells.

whenever the ring nodes are available. During broadcast, two paths are available for each position cache: directly connected to a PE as home particles, and connected to a position input ring as neighbor particles. In the first cycle (a), the first entries are sent to both PEs and the ring, since the ring nodes are not occupied. In the fifth cycle (b), the data sent to the ring previously are marked as dirty so that they will not be sent to the ring again. Position caches 1 and 2 cannot send data to the position input ring because the slots are occupied, but position cache N can still pass data through for the slot is empty. In the eighth cycle (c), the home particles (signified by the red arrows) are no longer synchronized. The gray empty boxes are skipped to avoid bubbles. In the ninth cycle (d), the neighbor particles sent to the ring also become asynchronous. Eventually, the neighbor particles are evaluated out-of-order.

5.2.6 Supporting Larger Simulations

The discussions above are based on an assumption in line with the proposed utilization of FPGAs in the MD processing ecosystem, i.e., the number of PEs is greater than or equal to the number of cells. But what if the simulation space is large and contains, say, a thousand cells? A natural extension is for PEs to work on multiple cells. There are, however, a number of challenges. First, with more cells involved, how to store the extra data on-chip given limited BRAM resources? Second, the latency in rings becomes overwhelming if more ring nodes are added to support a larger number of cells. And third, for each step, the number of cells accessed is still greater than the number of PEs due to the lack of periodic boundary condition (see Figure 12(a), the orange cells are to be processed in the next step but still are accessed).

Memory Partitioning. As a BRAM has 512 entries in depth and a cell typically has <80 particles per cell, most BRAM entries are wasted if a BRAM only contains the particle data from a single cell. Based on this fact, a BRAM can be partitioned into several interleaving cell regions,

where the two red cells are the cells to be processed in separate steps, as Figure 12(b) shows. This method also helps reducing the length of rings, as the interleaved cells can share the same ring node for routing.

Although the storage and the ring latency problems are solved, the concurrency problem persists. In Figure 12(c), two neighbor particles are being processed by the same PE, but with data from different cells. One way is to traverse all the particles in a memory block, but time is wasted in waiting for unnecessary data to pass. Therefore, an economical way is to take advantage of the previously mentioned double buffers. Originally, only one of the two buffers is used during the force evaluation phase while the other is only used during motion update. Now with the extra concurrency demand, the unused buffer can be duplicated from the other for 2x concurrency, such that NP1 and NP2 can be evaluated simultaneously with only the data from the desired cells.

5.3 General Summary of the Optimized Designs

The two designs have distinct strengths and weaknesses. Optimized design 1, inherited from high-level mapping scheme 1, makes the MD process straightforward yet with two major problems. First, the design has difficulty being extended to multiple FPGA chips. Intuitively, each FPGA is in charge of evaluating a certain region in a simulation space and they only need to exchange data for among a few boundary cells. With the particles in a single cell scattered among all caches, all the caches on a single chip are uniformly involved in inter-FPGA communications, intensifying the complexity of communication. Second, the neighbor particles need to be reloaded after the evaluation of each reference cell, with about ~20% overhead introduced.

Optimized design 2 is good for multi-chip extension: particles are still stored by cell, localizing particles in space. However, it demands resources to construct the routing rings, and its scalability on a single chip is relatively weak because it relies on relatively sophisticated mechanism to support larger scale simulations.

6 EVALUATION

6.1 Experiment Setup

The designs are implemented on a Xilinx Alveo U280 acceleration card, which has 1065K CLB LUTs, 2134K CLB registers, 1490 block RAMs, 960 ultra RAMs, and 8490 DSPs available in the dynamic region.

We assume each cell initially has 64 particles, which is physically sound. We also have eight motion update units equipped for all the designs in all scenarios, basically making the time spent in motion update negligible compared to the force evaluation phase. The baseline designs 1 and 2 (B1, B2), and the optimized design 1 (O1) have a fixed number of PEs. The former two designs have 128 PEs and the latter has 160 PEs (with 20% PEs reserved to handle particle migration). Baseline design 3 (B3) and the optimized design 2 (O2), however, have a number of PEs equal to the number of cells in the simulation space.

The hierarchical filters in O2 are temporarily disabled for performance comparison with the other designs; the effects

of several configurations are discussed in further detail in Section 6.3.

6.2 Performance and Comparison

To obtain a comprehensive analysis to all five designs, we evaluate the performance of the designs on two aspects: scalability and PE efficiency.

Note that in these design comparisons, the O2 results for 6x6x6 cell space are derived from cycle-accurate simulation since the on-chip resources of an Alveo U280 are not sufficient to map all 216 PEs. Results for the 6x6x6 space with 108 PEs are presented in a later Section.

6.2.1 Scalability Comparison

Scaling results are given in Figure 13. The performances of the three designs on the left with fixed number of PEs scale linearly with the increasing space size. This shows there is no extra overhead involved as the simulation space scales up. From another perspective, O1 is suitable for evaluating a limited number of cells on a single chip, i.e., strong scaling. The performance is not down-graded with a larger number of PEs working on a small number of particles/cells.

Ideally, for the designs on the right, the number of PEs scales linearly with the number of cells, and the bars in the figure should have the same height. However, the results are clearly influenced by the increasing number of cells. In B3, as discussed above, traversing all neighbor particles for a single reference particle leads to an imbalanced filtering rate. This situation deteriorates as the number of PEs increases as the highest and the lowest filtering rates are farther apart. In O2, the poor scaling is due to the extended rings. Long rings result in long latency and congestion in the rings. The slots in the rings are occupied by the packets still in transit, preventing the new packets from entering the rings. This suggests O2 also works well with a limited number of cells.

6.2.2 PE Efficiency Comparison

We define PE utilization as the average number of valid force pairs evaluated per PE divided by the overall working cycles in an MD iteration. PE utilization reflects hardware efficiency because the most hardware resources are used to for the force computation.

Results are shown in Figure 14. We observe that B1 and B2 have substantially lower PE utilization around 15%~20%. The greatest disadvantage of the two baselines lies in the global synchronization, which is used to prevent forces of different reference particles entering the adder trees. Another important factor for B1 is the neighbor particle pre-loading process, introducing considerable overhead.

Compared with the two baselines, O1 improves utilization by more than 30%. With the buffers above the adder tree, we can only allow the forces of the same reference particle entering the adder tree, such that global synchronization is unnecessary. However, the reserved PEs and neighbor particle pre-loading overhead prevents limits PE utilization.

B3 and O2 have higher PE utilizations for 3x3x3 cells. The two designs have fully streaming particles without pre-loading. Compared with O2, B3 suffers from the overhead brought by global synchronization, leading to a lower PE utilization for a 3x3x3 cell space. O2, however, has its

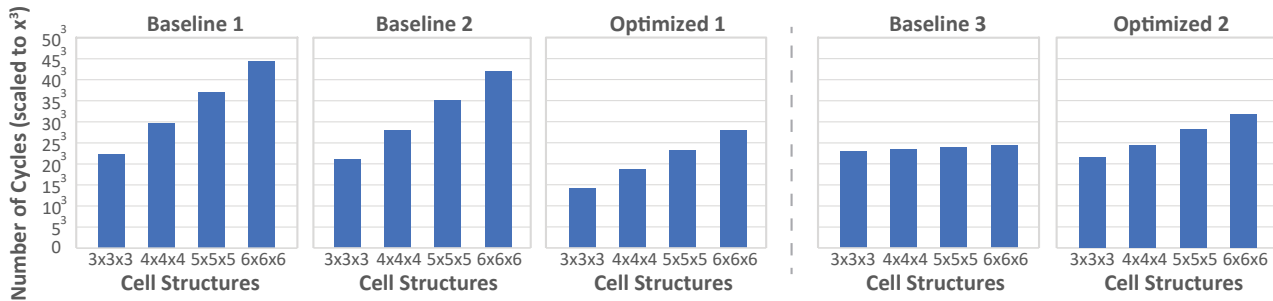


Fig. 13. The performance in cycles per iteration of three baseline and two optimized designs. We show results for four different cubic cell structures in the simulation space to show scalability. The y -axis is scaled to x^3 for a better illustration of linearity. The dashed line is to split the designs with substantial scalability differences.

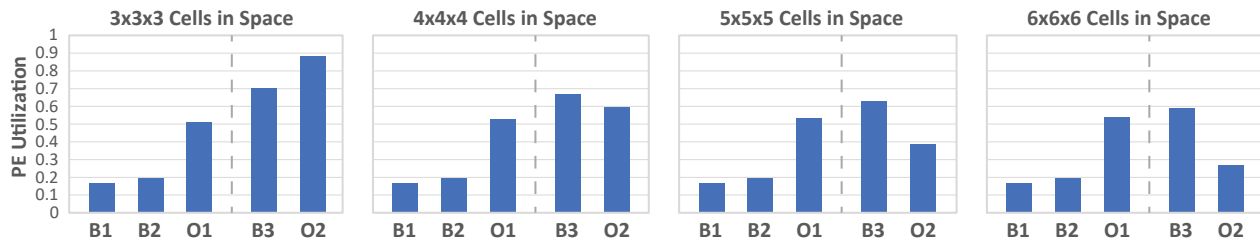


Fig. 14. The PE utilizations of three baseline and two optimized designs. The results for four different cubic cell structures in the simulation space are provided. B1~B3: baseline designs; O1 and O2: optimized designs. The dashed lines split the designs with substantial scalability differences.

utilization drop for larger simulation spaces due to the over-extended rings, especially the force output ring where heavy congestion takes place.

All baseline designs are subject to complicated data routing paths as shown and discussed in Figure 3(d). This leads to unacceptable frequency degradation. The frequency of the optimized designs can generally reach >200 MHz, while the baselines can only operate at 70 MHz or lower for more than $4 \times 4 \times 4$ cells. From this perspective, both the optimized designs are a great improvement over the baselines.

6.3 Evaluating Design Options

6.3.1 Hierarchical Filtering

The hierarchical filtering mechanism not only enhances the pass rate of the filters in a PE, but also frees the PEs computing zero forces. Without hierarchical filtering, every force register in Figure 9(a) requires zero checking before injecting the forces to a force output ring; otherwise load on the force output rings escalates due to unnecessary transfers. Although only eight exponent bits are checked per dimension per register (24 bits in 3-D), the total cost in hardware is not always worth it as the number of force registers can be massive. In our evaluation, we compare the original O2 design without zero force checking with the modified O2 with hierarchical filters.

Figure 15 shows the performance of the two O2 configurations. In the $3 \times 3 \times 3$ cell space case, the hierarchical version reaches equilibrium at four filters, the original at six filters. For larger cell spaces, they reach equilibrium at almost the same number of filters but with varied performance. This is because the bottleneck for performance lies in the force output and/or position input. The rings reach their maximum capacity, such that their performances do not improve with an increasing number of filters. As discussed above, the hierarchical version reduces force output traffic without the cost of zero checking, leading to the better performance with a heavily congested force output ring.

6.3.2 Number of Rings

Given that the rings become the bottleneck with larger cell spaces, higher ring bandwidths are expected. We intuitively add more rings to the system to tackle the problem. We inspect the number of force rings from one to four with up to two position input rings. The results are given in Figure 16.

The number of rings are denoted as [number of position input rings, number of force output rings]. We observe in the figure that the number of cycles saturate at [1, 1], [1, 2], [2, 3] and [2, 4] rings for the four example cell spaces in the case of original O2, and [1, 1], [1, 2], [2, 2], and [2, 3] rings in terms of O2 with hierarchical filters. This indicates that with hierarchical filters, the number of rings required is efficiently reduced. With both configurations applied, the utilization of PEs is greater than 75%.

6.4 Benchmarking

We use a state-of-the-art OpenMM liquid argon benchmark. The reference systems are an Intel Xeon Gold 6226R CPU with 64 threads and a Quadro RTX 8000 GPU. The results are demonstrated in two halves with different levels of simulation space sizes with FPGA-O2 design as the reference. We use ns/day as the unit of performance (FPGA designs run at 200 MHz).

Table 1 shows the first half of the results. For small simulation spaces of $3 \times 3 \times 3$ and $4 \times 4 \times 4$ cells, FPGA-O1 has overwhelming speedup compared to others due to its straightforward scaling capability, utilizing all PEs on a small cell space, achieving 4.17x speedup with respect to GPU on $3 \times 3 \times 3$ cells. As for FPGA-O2, it has steady performance scaling with the cell space size and it offers considerable speedup against GPU, up to 74% improvement.

The second half of the results is shown in Table 2. The 216-PE version and the 108-PE version are both demonstrated for $6 \times 6 \times 6$ cell space, where the former shows the scalability of the hardware, and the latter shows the capability of handling larger simulation spaces. The resources

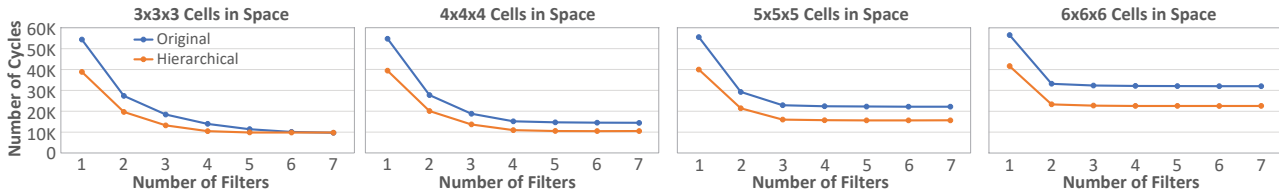


Fig. 15. The performance comparison of the original O2 and O2 with hierarchical filters for four representative cell spaces.

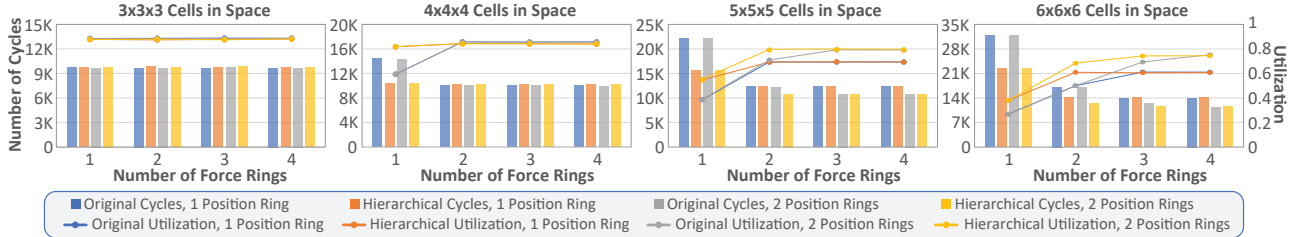


Fig. 16. The performance of the original and hierarchical O2 for four representative cell spaces with different numbers of rings equipped.

TABLE 1
FPGA-O2 performance compared to FPGA-O1, GPU, and CPU with up to 32 threads.

Cell #	Particle #	Performance	FPGA-O1	FPGA-O2	GPU	CPU-1x	CPU-2x	CPU-4x	CPU-8x	CPU-16x	CPU-32x
3x3x3	1728	ns/day	12222	3540	2921	80.06	135.5	204.6	236.3	183.4	152.9
		speedup	0.29	1.00	1.21	44.2	26.1	17.3	15.0	19.3	23.2
4x4x4	4096	ns/day	5303	3411	2126	35.25	60.56	92.8	110.8	95.32	101.5
		speedup	0.64	1.00	1.60	96.8	56.3	36.8	30.8	35.8	33.6
5x5x5	8000	ns/day	2750	3195	1841	19.35	32.79	60.05	88.7	78.03	77.5
		speedup	1.16	1.00	1.74	165	97.4	53.2	36.0	40.9	41.2

TABLE 2
FPGA-O2 performance of large simulation spaces compared to FPGA-O1, GPU, and 32-thread CPU.

Cell #	Particle #	Performance	FPGA-O1	FPGA-O2	GPU	CPU-1x	CPU-2x	CPU-4x	CPU-8x	CPU-16x	CPU-32x
6x6x6	13824	ns/day	1602	2997/1637 ¹	1542	11.05	18.72	31.42	49.19	53.12	52.7
		speedup	1.87/1.02	1.00	1.94/1.06	271/148	160/87.5	95.4/52.1	60.9/33.3	56.4/30.8	56.8/31.0
8x8x8	32768	ns/day	680.4	765.9	801.8	5.258	8.818	15.95	20.92	28.54	33.19
		speedup	1.13	1.00	0.96	146	86.8	48.0	28.4	26.8	23.1
12x8x8	49152	ns/day	454.5	510.6	618.7	3.490	5.784	10.55	17.63	21.54	27.06
		speedup	1.12	1.00	0.83	146	88.3	48.4	29.0	23.7	18.9

¹ 216 PEs (estimated) / 108 PEs, where the former assumes sufficient resources are provided on an FPGA chip.

on-chip allow us to simulate $\sim 50K$ particles in total, with the performances comparable to GPU performance.

6.5 Hardware Utilization

To study the availability of the configurations of O2, Table 3 highlights the hardware resources consumption for the performances of O2 in the representative cell spaces evaluated above. The number of position rings and force rings are optimal to make sure the performance is not down-graded by the congestion in rings. Due to the limited number of BRAMs, some of the buffers are implemented with logic as LUTRAMs.

Note that the resource utilizations of 8x8x8 and 8x8x12 cell spaces are almost the same. This is because a cache is not fully utilized in depth and therefore can contain data from more cells, leading to zero extra cost in memory. Most importantly, the number of PEs is not changed, such that only a negligible amount of extra resources is consumed to adjust to the space size.

6.6 Energy Validation

Energy convergence validation is performed on a 20K liquid argon dataset. Figure 17 compares the reference double-precision floating point OpenMM result with the implementation described in this paper (23-bit offset precision scheme in Figure 17(c) and the linear interpolation described in

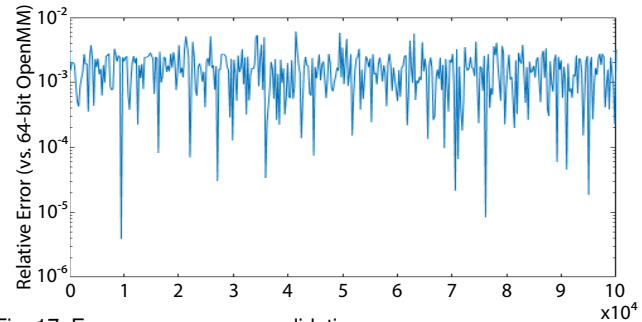


Fig. 17. Energy convergence validation

III-B). Over 100K iterations (2 fs per iteration) the relative difference is typically on the order of $10^{-4} \sim 10^{-3}$.

7 CONCLUSION

Accelerating MD with FPGAs has been studied for many years [30], [31], [32], [33], [34], [35], [36], [37], [38]. As resources per chip have multiplied, so has the number of compute units to the point where thousands of elements need to be sourced and sinked every cycle. MD is fundamentally a data movement problem: even with standard optimizations (cell lists and Newton's third law) data are necessarily used and updated simultaneously by many compute units. With the previous routing solutions no longer viable, maintaining high efficiency has required several design innovations.

TABLE 3
Hardware Utilization of FPGA-O2 with spatial configurations.

Cell #	Particle #	PE #	Position Ring #	Force Ring #	LUT	FF	BRAM	URAM	DSP
3x3x3	1728	27	1	1	16%	9%	13%	17%	16%
4x4x4	4096	64	1	2	38%	22%	33%	40%	39%
5x5x5	8000	125	2	2	73%	41%	63%	78%	74%
6x6x6	13824	108	1	2	65%	38%	55%	68%	64%
8x8x8	32768	128	2	3	85%	51%	74%	80%	84%
8x8x12	49152	128	2	3	85%	51%	74%	80%	84%

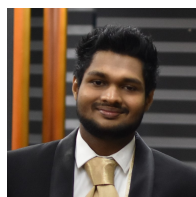
In this study of FPGA-based range-limited MD work, we present multiple 3D-to-2D workload mapping schemes: three baseline designs based on distinct, and, at a high level, exhaustive mappings, and two optimized designs. The two optimized designs have scalability in two directions. The first design scales easily with the size of a simulation space with a constant number of PEs, while the second allows the number of PEs scale with the size of a simulation space to obtain sustained performances. We also analyze several configurations of the 2nd optimized design, and the effectiveness of the configurations is highlighted.

The overall motivation for using FPGAs for MD computation is that FPGA-based clusters fill a vital niche: scalable COTS-based systems that provide long timescales on problem sizes crucial to both drug discovery and basic science. The necessary condition is for the per-device performance to be comparable to that of the accelerator alternatives. That has been demonstrated here in multiple applicable scenarios. For example, optimized design 2 achieves a 1.74x speed-up for 8000 particles, and optimized design 1 achieves 4.18x for 1728 particles. While these simulations are a small fraction of tens of thousands typical in production runs, they are exactly the workload per device when these runs are partitioned over 8 or 32 devices, respectively.

REFERENCES

- [1] A. Yu, A. J. Pak, P. He, V. Monje-Galvan, L. Casalino, Z. Gaieb, A. C. Dommer, R. E. Amaro, and G. A. Voth, "A multiscale coarse-grained model of the SARS-CoV-2 virion," *Biophysical Journal*, vol. 120, no. 6, pp. 1097–1104, 2021.
- [2] P. R. Arantes, A. Saha, and G. Palermo, "Fighting covid-19 using molecular dynamics simulations," 2020.
- [3] R. Alnajjar, A. Mostafa, A. Kandeil, and A. A. Al-Karmalawy, "Molecular docking, molecular dynamics, and in vitro studies reveal the potential of angiotensin ii receptor blockers to inhibit the covid-19 main protease," *Heliyon*, vol. 6, no. 12, p. e05641, 2020.
- [4] M. A. White, W. Lin, and X. Cheng, "Discovery of covid-19 inhibitors targeting the sars-cov-2 nsp13 helicase," *The journal of physical chemistry letters*, vol. 11, no. 21, pp. 9144–9151, 2020.
- [5] T.-S. Lee, B. K. Allen, T. J. Giese, Z. Guo, P. Li, C. Lin, T. D. McGee, D. A. Pearlman, B. K. Radak, Y. Tao, H.-C. Tsai, H. Xu, W. Sherman, and D. M. York, "Alchemical binding free energy calculations in amber20: Advances and best practices for drug discovery," *Journal of Chemical Information and Modeling*, vol. 60, no. 11, pp. 5595–5623, 2020, pMID: 32936637. [Online]. Available: <https://doi.org/10.1021/acs.jcim.0c00613>
- [6] T.-S. Lee, Z. Lin, B. K. Allen, C. Lin, B. K. Radak, Y. Tao, H.-C. Tsai, W. Sherman, and D. M. York, "Improved alchemical free energy calculations with optimized smoothstep softcore potentials," *Journal of Chemical Theory and Computation*, vol. 16, no. 9, pp. 5512–5525, 2020, pMID: 32672455. [Online]. Available: <https://doi.org/10.1021/acs.jctc.0c00237>
- [7] Z. Cournia, B. K. Allen, T. Beuming, D. A. Pearlman, B. K. Radak, and W. Sherman, "Rigorous free energy simulations in virtual screening," *Journal of Chemical Information and Modeling*, vol. 60, no. 9, pp. 4153–4169, 2020, pMID: 32539386. [Online]. Available: <https://doi.org/10.1021/acs.jcim.0c00116>
- [8] M. M. Rahman, T. Saha, K. J. Islam, R. H. Suman, S. Biswas, E. U. Rahat, M. R. Hossen, R. Islam, M. N. Hossain, A. A. Mamun, M. Khan, M. A. Ali, and M. A. Halim, "Virtual screening, molecular dynamics and structure–activity relationship studies to identify potent approved drugs for covid-19 treatment," *Journal of Biomolecular Structure and Dynamics*, vol. 0, no. 0, pp. 1–11, 2020.
- [9] D. Case, T. Cheatham III, T. Darden, H. Gohlke, R. Luo, K. Merz, Jr., A. Onufriev, C. Simmerling, B. Wang, and R. Woods, "The Amber biomolecular simulation programs," *Journal Computational Chemistry*, vol. 26, pp. 1668–1688, 2005.
- [10] J. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. Skeel, L. Kale, and K. Schulten, "Scalable molecular dynamics with NAMD," *Journal Computational Chemistry*, vol. 26, pp. 1781–1802, 2005.
- [11] P. Eastman and V. Pande, "OpenMM: A Hardware-Independent Framework for Molecular Simulations," *Computing in Science and Engineering*, vol. 4, pp. 34–39, 2010.
- [12] A. Goetz, M. Williamson, D. Xu, D. Poole, S. L. Grand, and R. Walker, "Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. Generalized Born," *J. Chem. Theory Comput.*, vol. 8, pp. 1542–1555, 2012.
- [13] D. van der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. Mark, and H. Berendsen, "GROMACS: fast, flexible, and free," *Journal Computational Chemistry*, vol. 26, pp. 1701–1718, 2005.
- [14] S. Páll, A. Zhmurov, P. Bauer, M. Abraham, M. Lundborg, A. Gray, B. Hess, and E. Lindahl, "Heterogeneous parallelization and acceleration of molecular dynamics simulations in gromacs," *The Journal of Chemical Physics*, vol. 153, no. 13, p. 134110, 2020.
- [15] C. Wu, T. Bandara, S. Geng, V. Sachdeva, B. Sherman, and M. Herberdt, "System-level modeling of gpu/fpga clusters for molecular dynamics simulations," in *IEEE 2021 High Performance Extreme Computing Conference*, 2021.
- [16] J. Glaser, T. D. Nguyen, J. A. Anderson, P. Lui, F. Spiga, J. A. Millan, D. C. Morse, and S. C. Glotzer, "Strong scaling of general-purpose molecular dynamics simulations on gpus," *Computer Physics Communications*, vol. 192, pp. 97 – 107, 2015.
- [17] J. Sheng, B. Humphries, H. Zhang, and M. Herberdt, "Design of 3D FFTs with FPGA Clusters," in *Proceedings of the IEEE High Performance Extreme Computing Conference*, 2014.
- [18] A. Lawande, A. George, and H. Lam, "Novo-G#: a multidimensional torus-based reconfigurable cluster for molecular dynamics," *Concurrency and Computation: Practice and Experience*, 2016.
- [19] J. Sheng, C. Yang, A. Caulfield, M. Papamichael, and M. Herberdt, "HPC on FPGA Clouds: 3D FFTs and Implications for Molecular Dynamics," in *27th International Conference on Field Programmable Logic and Applications*, 2017, doi: 10.23919/FPL.2017.8056853.
- [20] Z. Yao, J.-S. Wang, G.-R. Liu, and M. Cheng, "Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method," *Computer Physics Communications*, vol. 161, no. 1, pp. 27 – 35, 2004.
- [21] W. Brown, P. Wang, S. Plimpton, and A. Tharrington, "Implementing molecular dynamics on hybrid high performance computers—short range forces," *Computer Physics Communications (CPC)*, vol. 182, no. 4, pp. 898–911, 2011.
- [22] D. Shaw, "A Fast, Scalable Method for the Parallel Evaluation of Distance-Limited Pairwise Particle Interactions," *Journal Computational Chemistry*, vol. 26, no. 13, pp. 1318–1328, 2005.
- [23] B. Towles, J. Grossman, B. Greskamp, and D. Shaw, "Unifying on-chip and inter-node switching within the Anton 2 network," in *Proceedings of the International Symposium on Computer Architecture*, 2014, pp. 1–12.
- [24] M. Chiu and M. Herberdt, "Molecular dynamics simulations on high performance reconfigurable computing systems," *ACM Transactions Reconfigurable Technology and Systems*, vol. 3, no. 4, pp. 1–37, 2010.

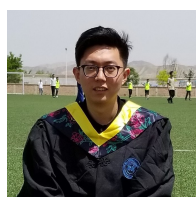
- [25] —, "Efficient filtering for molecular dynamics simulations," in *Proceedings of the IEEE Conference on Field Programmable Logic and Applications*, 2009.
- [26] T. Darden, D. York, and L. Pedersen, "Particle Mesh Ewald: an $N \log(N)$ method for Ewald sums in large systems," vol. 98, pp. 10 089–10 092, 1993.
- [27] K. Bowers, R. Dror, and D. Shaw, "Zonal methods for the parallel execution of range-limited n-body simulations," *Journal Computational Physics*, vol. 221, no. 1, pp. 303–329, 2007.
- [28] K. J. Bowers, R. O. Dror, and D. E. Shaw, "The midpoint method for parallelization of particle simulations," *The Journal of Chemical Physics*, vol. 124, no. 18, p. 184109, 2006. [Online]. Available: <https://doi.org/10.1063/1.2191489>
- [29] D. E. Shaw, P. J. Adams, A. Azaria, J. A. Bank, B. Batson, A. Bell, M. Bergdorf, J. Bhatt, J. A. Butts, T. Correia *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.
- [30] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, "Reconfigurable molecular dynamics simulator," in *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, 2004, pp. 197–206.
- [31] Y. Gu, T. VanCourt, and M. Herbordt, "Accelerating molecular dynamics simulations with configurable circuits," in *IEEE Conference on Field Programmable Logic and Applications (FPL)*, 2005, dOI: 10.1109/FPL.2005.1515767.
- [32] T. Hamada and N. Nakasato, "Massively parallel processors generator for reconfigurable system," *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, 2005.
- [33] R. Scrofano and V. Prasanna, "Preliminary investigation of advanced electrostatics in molecular dynamics on reconfigurable computers," in *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2006.
- [34] Y. Gu, T. VanCourt, and M. Herbordt, "Accelerating molecular dynamics simulations with configurable circuits," *IEE Proceedings on Computers and Digital Technology*, vol. 153, no. 3, pp. 189–195, 2006, doi: 10.1049/ip-cdt:20050182.
- [35] V. Kindratenko and D. Pointer, "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*, 2006, pp. 13–22.
- [36] S. Alam, P. Agarwal, M. Smith, J. Vetter, and D. Caliga, "Using FPGA devices to accelerate biomolecular simulations," *Computer*, vol. 40, no. 3, pp. 66–73, 2007.
- [37] M. Schaffner and L. Benini, "On the feasibility of FPGA acceleration of molecular dynamics simulations," 2018.
- [38] D. Jones, J. E. Allen, Y. Yang, W. F. Drew Bennett, M. Gokhale, N. Moshiri, and T. S. Rosing, "Accelerators for classical molecular dynamics simulations of biomolecules," *Journal of Chemical Theory and Computation*, vol. 18, no. 7.



Sahan Bandara received his bachelor's degree in Electronic and Telecommunication Engineering from the University of Moratuwa, Sri Lanka in 2015, and his master's degree in Computer Engineering from Boston University in 2019. In the interim between academic degrees, Sahan worked as a corporate application engineer for Synopsys. His research interests are computer architecture, hardware security, and hardware operating systems for reconfigurable hardware.



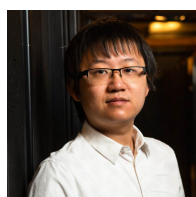
Tong Geng is an assistant professor in the ECE and CS departments of the University of Rochester and the director of the IntelliArch Lab. Previously Tong worked in the Physical & Computational Sciences Directorate at Pacific Northwest National Laboratory. He received his Ph.D. in Computer Engineering at Boston University in 2020. His research interests include computer architecture & systems, machine learning, graph intelligence, and high-performance computing.



Anqi Guo is from Dalian, China and received his Master's Degree in Electrical and Computer Engineering at Boston University in 2019. He is now a Ph.D. student at Boston University. His current research is on heterogeneous SmartNIC system for coupling software and hardware in machine learning applications.



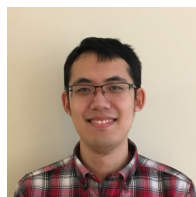
Pouya Haghi received the B.Sc. degree in electrical and electronics engineering from the Iran University of Science and Technology in 2017, and the M.Sc. degree in electrical engineering and digital systems from the University of Tehran, Tehran, in 2019. He joined ECE department of Boston University in Sep. 2019 and is currently conducting research on in-network computing on FPGAs.



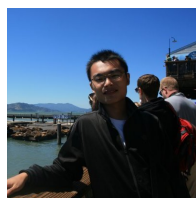
Ang Li is a senior computer scientist who joined the High-Performance Computing (HPC) group at Pacific Northwest National Laboratory in 2016. His research has been focused on software-hardware co-design for scalable heterogeneous HPC, including GPUs, FPGAs, CGRAs, artificial intelligence/machine learning accelerators, and quantum processors. His research covers the full-stack design from circuit level up to architecture, system, library, and applications.



Martin Herbordt is a Professor with the Department of Electrical and Computer Engineering, Boston University, where he directs the Computer Architecture and Automated Design Lab.



Chunshu Wu received his bachelor's and master's degrees in physics from Dalian University of Technology, Liaoning, China in 2016 and from Brown University in 2018 respectively. After that, he joined ECE department of Boston University as a graduate student. Chunshu's current research is on FPGA-based acceleration for Molecular Dynamics.



Chen Yang received his Bachelor of Engineering and Master of Science degrees in Electrical Engineering from Wuhan University, China in 2012 and in Computer Engineering from University of Florida in 2014, respectively. He has received his Ph.D. degree in Computer Engineering from Boston University in 2019.