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 scalable long timescale MD with commodity clusters.

Index Terms—FPGA, Micro-architecture, Molecular Dynamics

1 Introduction

Molecular Dynamics (MD) is a simulation technique for analyzing ensembles of atoms and molecules that is executed iteratively over discrete, infinitesimal time intervals. During an MD iteration, forces on particles in a simulation box are computed and, then, particle states are updated given their previous state and the results of the particle interactions. Forces are due to particles interacting mutually as indicated by Newtonian physics.

Due to the brevity of the time interval, typically on the order of femtoseconds (10^{-15} seconds), carrying out simulations of just a few nanoseconds (10^{-9} seconds) of reality necessitates millions of iterations, making even a 100 nanosecond-scale simulation “long.” However, 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 critical benefit 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 currently underway [5], [6], [7].

In drug design 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). Simulation duration can range from hundreds of nanoseconds to many microseconds, as it can take tens of nanoseconds for the protein and drug candidate to attain structural stability [8], longer to achieve overall convergence, and far longer still for certain vital conformations to emerge. This combination of small space and long duration leads to extreme challenges in strong scaling.

A variety of high-performance software packages have been developed [9], [10], [11], [12], [13], with many supporting GPU acceleration. But while GPU-based systems excel at handling throughput-oriented and 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 uses application-specific integrated circuits (ASICs) optimized for MD. For example, Anton 3 can achieve over 200 μs-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 in cost and development time inherent with small-run ASIC-based systems.

An alternate MD architecture is a cluster 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 application-level processing with those transceivers* in a small number of cycles. These capabilities enable FPGA-accelerated clusters to approach ASIC cluster performance for the long range force computation (LR) [17], [18], [19], the part of MD that is most difficult to scale. The critical question is whether a *single node* FPGA-

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

based MD engine can be sufficiently competitive.

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 consists of 90% of the computation, while LR is more memory and communication oriented. RL and LR are substantially independent in data flow and can be considered as separate tasks. This work specifically focuses on RL. While there have been two decades of research on FPGA-MD, increases in on-chip resources have necessitated a complete design overhaul. In particular, compute logic now supports hundreds of force processors (PEs), up from around ten previously. Routing capability, however, has not increased proportionally making previously used interconnects impracticable. Moreover, load balancing issues have increased exponentially.

FPGA-MD is at its heart a communication problem. Each of the hundreds of PEs sources and sinks dozens of particle pairs every cycle; each particle pair is unique, as is each particle's partner set, although there is substantial overlap when particles are proximate (which must be exploited). To address the particle-to-PE mapping problem, the simulation space is partitioned into cells using the cell-list method [20], [21]: data are grouped into boxes for both parallel processing and to reduce the computational complexity from all-to-all pairwise $O(N^2)$ to $O(mN)$, where m is the average number of the particles within a small range (so $m << N$). Applying the Newton's third law optimization (N3L) (§ 2.3), e.g., through the half-shell method [22], further halves the number of computations. Particle pairs can therefore be enumerated as follows: $\forall$ particles p_i in a cell, $\forall$ of its neighbor particles n_j, $\forall$ cells c_k. Since each particle pair computation is independent, there are three high-level mapping schemes to be considered: all PEs working on the same particle (§ 3.2), all PEs working on different particles in the same cell (§ 3.3), and each PE working on a different cell (§ 3.4).

This brings us to the essence of the single-FPGA (or ASIC) problem: for any possible high-level mapping scheme, can a scalable interconnect between memory and PEs be constructed? Note that, with a small number of PEs, only particle- and neighbor-centric need be considered (one cell at a time is processed); and that liberal use can be made of broadcast [23]. With hundreds of PEs, however, directly interconnecting memory with PEs in a 3-D application that is mapped to a 2-D chip fabric leads to huge complexity and a drastic decrease in frequency [24]. Entirely new methods are needed: a transposed memory block method and ring routing are proposed and proved adequate for RL computation with high efficiency (§ 5).

The transposed memory block method (§ 5.1) allows a memory block to store a portion of particles from all cells to efficiently simplify the design scaling with respect to the simulation space size. The ring-based routing method connects cells in rings [25], enabling flexible data movement between cells. Routing in these new methods, however, requires more cycles and results in bubbles in the network limiting throughput. To compensate, several innovations are proposed. First, a data caching method is developed to minimize data transfers by boosting data locality (§ 5.2.2). Second, 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). Third, to improve computing capability, the design can be configured to support a group of PEs working on a single cell (§ 5.2.5). Fourth, for scalability, a ring can be decomposed into smaller pieces to reduce latency (§ 5.2.6). Finally, an out-of-order data broadcast mechanism is developed to dynamically fill the empty slots (bubbles) in the ring (§ 5.2.7).

A crucial aspect of these designs is to maintain compatibility with a third standard optimization: prefiltering particle pairs (within a cell neighborhood) to eliminate computations where the mutual force is negligible [23] (replacing the neighbor lists often used in CPU implementations). Theoretically, for evenly distributed particles, the average pass rates for spherical and planar [26] filters are 15.5% and 17%, respectively. In this work, we upgrade these standard filters to hierarchical filters (§ 5.2.7). This method not only increases the pass rate to 25%, but also alleviates 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;
- Three particle/cell to PE mapping schemes are proposed and evaluated in depth;
- The ring routing method is studied as an alternative for the 3-D to 2-D mapping and found to scale with the increasing number of PEs on a single FPGA chip. Latency and concurrency problems introduced by the ring are solved with an out-of-order data broadcast and a data caching mechanism;
- Other optimizations, e.g., hierarchical filtering and memory partitioning are proposed and found to reduce hardware consumption and improve efficiency.

2 Background

2.1 Physical Models of Range-Limited Forces

RL consists of two components: the short range term of the electrostatic force, typically obtained using the Particle Mesh Ewald (PME) [27] (or related method), and the force deduced from the Lennard-Jones (LJ) potential, an empirical model combining the attractive van der Waals and repulsive molecular interactions. The LJ potential between particle i and j with distance r_{ij} is given in the following equation:

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

The potential is characterized by ϵ, the dispersion energy describing the potential amplitude, and σ, the characteristic particle distance at zero LJ potential. Taking the gradient of the potential, we obtain the RL force between particles i,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 indicates that the complexity of RL is O(N^2). However, 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

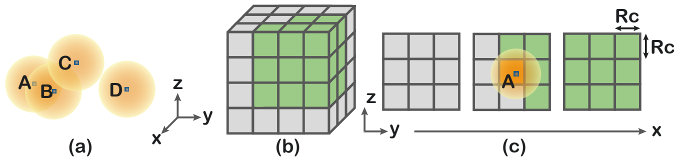


Fig. 1. Fundamentals of R_c and the cell list and N3L optimizations. (a) Cutoff regions of four particles. Only A and B interact. (b) Simulation space divided into 3x4x4 cells with side length the same as R_c. (c) Home cell of particle A and 26 adjacent cells unfolded from (b). Green cells are the neighbor cells remaining after applying the N3L optimization.

that the pairwise force can be ignored. To visualize, R_c is represented with halos in Figure 1(a). Particle A interacts only with particle B because A is in B's halo and vice versa. Particles C and D contribute little to the resultant force of A, and are therefore ignored in A's computation. Furthermore, the total computation can be reduced by half, as pairwise forces affect both particles symmetrically (the N3L optimization).

Periodic boundary conditions are frequently used in computational physics: a large homogeneous system can be modeled into replicas of a small system. That is, a particle exiting the small system is matched by the same particle with the same velocity entering from the opposite end.

2.2 RL Procedure

The general procedure has two phases, **force evaluation** and **motion update**, which are executed iteratively. During force evaluation, 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. Motion update is much less compute intensive than force evaluation 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 Structures: Neighbor Lists and Cell Lists

For any reference particle, its neighbor list identifies the particles within its cutoff radius: traversing neighbor lists is therefore efficient in that only non-zero forces are computed. The cost, however, is in data redundancy or complex memory traversal, and also that high memory bandwidth is needed to compensate for the limited data locality. While these are acceptable in CPU codes, in hardware implementations this method is generally replaced by filtering [26].

Cell lists group particles spatially. They allow memory locality to follow physical locality and are easy to generate. In Figure 1(b), the simulation space is partitioned into cubic cells (3x4x4) with side length R_c (see [28] on choosing side length). The particles in each cell are stored in a separate memory domain, 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 non-zero force. With N3L, only 14 out of all 27 surrounding cells need to be considered as neighbor cells (in the half-shell method), as Figure 1(c) illustrates. Other layouts for

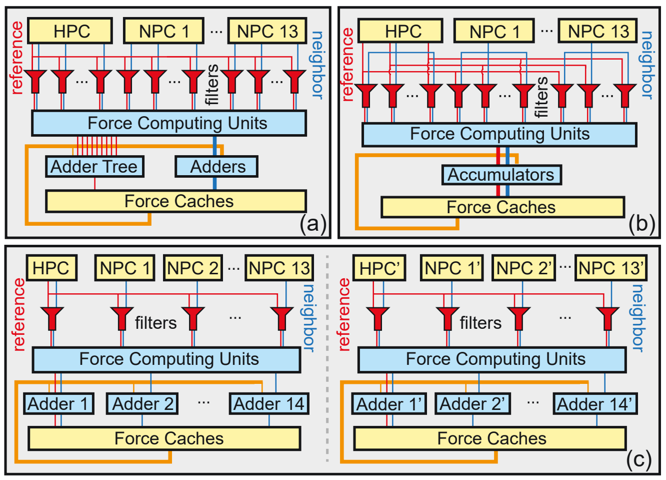


Fig. 2. High level memory-to-PE mapping schemes. (a) particle-centric: all PEs work on a single reference particle at a time. (b) cell-centric: all PEs work on a same cell at a time. (c) uniformly-spread: each PE works on a separate cell. HPC/NPC: Home/Neighbor Position Cache.

importing neighbor data are discussed in § 2.4. However, for uniform particle distributions, only 15.5% neighbor particles form valid pairs. 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 [26], especially with planar filtering, which uses only comparison operations and low-precision integers. Therefore, throughout this work, cell lists and planar filtering are used.

2.4 Neighbor Data Import 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 [29], [30], and the Hybrid Manhattan method in Anton 3 [31]. 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 more fragmented data storage that requires fine-grained memory access and allocation. In contrast to the NT and Manhattan methods, with the half-shell method data are only accessed by cell and thus easily organized. In summary, although the half-shell method requires a larger import volume, it is more hardware-friendly for FPGAs. Moreover, the excess import volume can be removed by inexpensive filters.

3 High Level Mapping Schemes

In this section, we systematically explore the FPGA-MD design space for RL and determine the most likely candidate architectures.

3.1 Design Space Exploration

The RL design space consists of the following elements: PEs, data organization, and data transfers among the PEs and data structures. A PE is defined as a force computing unit together with its associated logic, including filters and accumulators. As PE design is well-established, and as PEs comprise most of the compute logic in RL, the first priority is simply that the number of PEs should be maximized until all resources are consumed or routing becomes so cumbersome as to substantially reduce operating frequency or utilization. For data organization, particle data in cells are stored in logical memories referred to as HPCs (home cell position caches), NPCs (neighbor cell position caches), and Force Caches (forces of particles in individual cells). The name *cache* is used to denote that these memories are created from BRAMs (on-chip with fast access). Unlike CPU caches they are managed explicitly. A design consideration is how these caches are mapped to BRAMs. Also, for now we disregard the motion update units as they add little complexity.

The remainder of the design space involves data transfers. Four questions need to be answered: In what order should the data be fetched? To which PEs should data be sent and how (parallel, broadcast)? What physical interconnect should be implemented? And, what routing strategy best implements the data transfers?

We begin with an analogy between RL and matrix-matrix multiplication (MMM). In both, computational independence leads to enormous degrees of freedom and an exponential number of alternatives. But then locality-based constraints are applied to prune the space (in MMM, to variations of loop ordering and blocking). A similar strategy is applied here for RL giving us a strategy for the first two questions. For the third, while there are again exponentially many alternatives, the principle of parsimony can be applied. That is, once a simple solution has been found (i.e., where the PEs process particle-pairs with high utilization), then the more complex solutions can be ignored. In particular, fan-in/out must be minimized, both to reduce routing complexity and to avoid contention. For the final question (routing strategies), it is again sufficient to find a particular solution.

Algorithm 1 High Level Mapping

```
1: for each cell $hcell$ in all cells as the home cell do
2:     for each particle $p$ in $hcell$ do
3:         for each cell $ncell$ in neighbor cells do parallel
4:             for each particle $q$ in $ncell$ do parallel
5:                 $f_{pq} = ComputeForce(r_p, r_q)$
```

We now explore data fetch order and PE mapping by considering locality, i.e., while each particle's neighbor sets are disjoint, there is substantial overlap for nearby particles. Simply, some particles should be taken as *reference* particles with other *neighbor* particles pairing with them in parallel. An example is given in Algorithm 1, where computations involving multiple particles p_i in a home cell have good locality by being paired with the same set of neighbor particles for an entire loop nest.

The second question is now addressed. A PE with N_f filters is capable of processing N_f particle pairs in parallel. To take advantage of locality, a reference particle should be evaluated with N_f of its neighbor particles in a single PE. This constraint is extended as follows:

$$1 \leq N_{\text{ref}} \leq N_{\text{PE}} \tag{3}$$

where N_{PE} is the number of PEs and N_{ref} is the number of reference particles being processed. When $N_{\text{ref}} = 1$, all PEs work on the same reference particle (the 1st mapping scheme). When $N_{\text{ref}} = N_{\text{PE}}$, each PE can still work on the same cell, but on different reference particles (the 2nd scheme). Preferably, the reference particles are from different memory blocks to avoid memory access conflicts; and given particle mapping to cells, it is natural to let the PEs work on different cells (the 3rd scheme).

The mapping schemes can also be viewed as loop reorderings of Algorithm 1. That pseudocode indicates that all PEs work on the same reference particle p in $hcell$. The first *parallel* (line3) indicates PEs working in parallel to process different neighbor cells while the second (line 4) refers to filters in a PE working in parallel on different neighbor particles within a neighbor cell.

In the following subsections, three mapping schemes corresponding to the three scenarios extracted from Equation 3 (and corresponding loop orderings) are analyzed, each with reference particle locality satisfied.

3.2 All PEs Work on the Same Reference Particle

In Figure 2(a) (Algorithm 1 and $1 = N_{\text{ref}}$), a single reference particle at a time is broadcast to all filters in all PEs. In parallel, the particles from neighbor cells are sent to filters to pair with the reference particle. This mapping 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 the cells. This method achieves workload balance on particle pair level.

There are, however, obvious disadvantages. First, the cost for maintaining locality is high. While all particles in a home cell have the same set of neighbor particles, the number of neighbor particles is likely to be too high for temporal locality to be exploited. Second, all the neighbor partial forces are returned to only 14 neighbor cells, which requires tremendous bandwidth per cell to perform the accumulation without conflict. In general, successful FPGA applications continually use a large fraction of BRAM bandwidth. A possible solution is to interleave cells across BRAMs and is described in Section 5.1.

3.3 All PEs Work on the Same Home Cell

Figure 2(b) shows that the home cell broadcasts multiple particles to the filters. Algorithm 1 is modified so that now lines 2 & 3 are *parallel*, rather than 3 & 4, and $N_{\text{ref}} = N_{\text{PE}}$. Each neighbor cell only needs to broadcast one particle.

As 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. The neighbor particles are then cached in registers and no new neighbor particle is required until the current set of neighbor particles is processed. As for force return, this localization also introduces the opportunity for accumulating the neighbor forces before sending them back; thus the force return bandwidth is also reduced. That is, loops of Algorithm 1 are reordered to yield Algorithm 2.

Algorithm 2 High Level Mapping 2 Reordered

```
1: for each cell $ncell$ in neighbor cells do parallel
2:     for each particle $q$ in $ncell$ do
3:         for each cell $hcell$ in all cells as the home cell do
4:             for Home particle $p$ in $hcell$ do parallel
5:                 $f_{pq} = ComputeForce(r_p, r_q)$
```

Still, only 14 cells are evaluated at a time and the rest of the cell storage bandwidth remains idle. But again, an interleaving solution as proposed for Algorithm 1 is possible. Also, there are heavy bandwidth and fanout requirements in the broadcast of the home cell position.

3.4 Each PE Works on a Different Cell

Algorithm 3 High Level Mapping 3 reordered

```
1: for each home cell $hcell$ in all cells do parallel
2:     for each cell $ncell$ in neighbor cells do parallel
3:         for each particle $q$ in $ncell$ do
4:             for each particle $p$ in $hcell$ do
5:                 $f_{pq} = ComputeForce(r_p, r_q)$
```

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, NPC1 and NPC2′ in Figure 2(c) may be the same cell. Here, *parallel* is applied to lines 1 & 3 of Algorithm 1 (rather than 2 & 3 or 3 & 4) and N_{ref} is between 1 and N_{PE} (from Equation 3).

This mapping method addresses both the bandwidth and the idle memory bandwidth problems. First, at most one particle is broadcast from any cell, allowing concurrent reading of positions 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 (see Algorithm 3 with the new loop ordering).

4 Baseline Architectures

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

4.1 Design 1: Particle Centric

Based on Algorithm 1, the design layout is shown in Figure 3(a). Assume that there are m cells in the simulation space, and each cell has a set of BRAM based caches (position, force cache, and velocity). 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 carried out by a position distributor (Figure 3(d)), which selects particle position data from 14 specific neighbor position caches, for only 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 § 3.2, $\sim$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 $\sim$1000 particles, matching the number of filters). Besides, based on equation (**??**), 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. First, 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. Second, the forces applied on the neighbor particles (neighbor forces) are 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 $\sim$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

Based on Algorithm 2, the design is given in Figure 3(b). While appearing similar to Figure 3(a) the mechanism is quite distinct. 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 through the filters and are paired with the registered reference particles. A force computing unit is connected 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 being sent to the force caches. As for neighbor forces, a neighbor particle is shared by many filters at the same time, allowing the neighbor forces to be accumulated before writing to their force caches. Consequently, local adder trees are used to coordinate the high throughput of the neighbor forces and the limited number of force caches.

As the input forces to an adder tree must 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 ones are completed. That is, neighbor particles are grouped in batches with batch

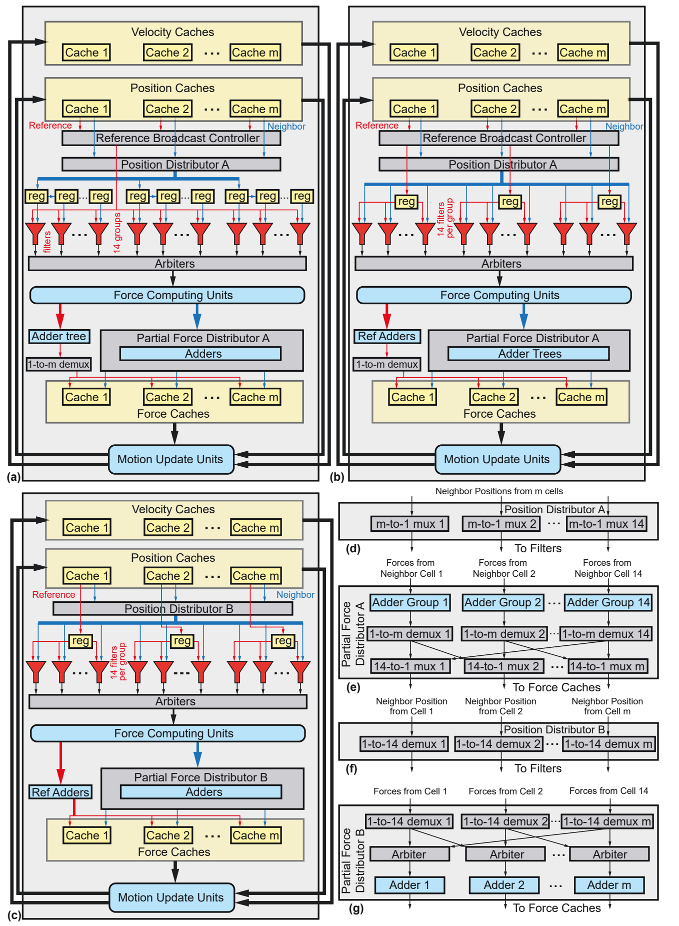


Fig. 3. The baseline designs and distributors. (a)-(c): baseline designs based on the three high-level mapping schemes, where the buffering FIFOs and the forces read from the force caches are omitted for conciseness. The thick lines and arrows stand for buses. (d)-(e): position and force distributors.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

size = 14, since they are from 14 neighbor cells. The force computing units only start to process the next 14 neighbor particles after the previous batch is completed.

4.3 Design 3: Uniform Spread

Based on Algorithm 3, this design no longer focuses on a single reference particle or a reference cell at a time; rather, all cells are 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 requires that all PEs receive particles with the same ID: it is convenient for all PEs to start with the same particle ID, thus the global synchronization. Second, the reference position is not stored in a filter buffer, but in a register (shared by the force computation unit and all the filters) to minimize the buffer sizes. As a result, the reference position cannot be discarded until all filter buffers are empty.

As with the other designs, each 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

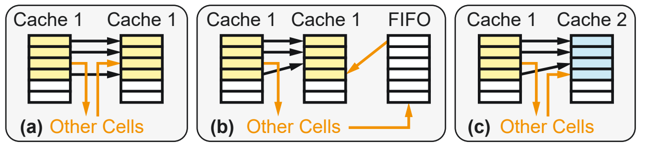


Fig. 4. Particle migration handling methods. (a): directly searching for empty slots. (b): the one-pointer method. (c): double buffering.

previous two designs (see Figure 3(f)) as the mechanism of this design is substantially different. Once all current reference particles have been evaluated, the position caches dispatch the next reference particles to the PEs.

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

4.4 Motion Update and Particle Migration

All three baseline designs use a cell-based cache data structure and the same motion update method. After force evaluation, the motion update units (in parallel) request position, velocity, and force data from individual caches. Upon receiving the requests, the force data are sent to motion update units; their original entries in force caches are 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 exiting Cache 1 leaves an empty slot in the cache. Without further optimization, a new particle must find an empty slot, a costly procedure.

A second method is to use a pointer to track the next available entry (vacancy) in a cache (Figure 4(b)). The local particles are updated first, while entering particles are stored in a FIFO. As the 3rd particle exits, the pointer points at the vacancy so the 4th particle is updated to the 3rd slot, with the pointer moving to the 4th slot. As a result, the non-migrating particles are stored compactly. The entering particles are then read from the FIFO and sent to the cache.

A drawback of the second method is that the hardware cost of a FIFO on an FPGA is similar to that of the cache itself. Therefore, we can instead simplify the logic and use two caches (Figure 4(c)). When a particle migrates in, the local update is paused, and the new particle written to Cache 2. Eventually, all updated particles are stored in Cache 2. In the next MD iteration, the roles are reversed. In practice, the double buffering method can be implemented with a single cache as long as it has two ports (read and write) and sufficient depth.

5 Optimized Designs

Problems with the three baseline designs are evident. First, all suffer from complex routing logic 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, data locality is not fully exploited. To solve these two fundamental problems of the baseline designs, we propose two optimized designs evolved from design 1 and design 3, plus extensions.

5.1 Design-T: Transposed Memory Blocks

Design-T originates from Design 1, where a single reference particle is paired with all its neighbor particles (see Figure 5(a)). Storing particles by cell, as used in Baseline 1, is simple, but not suitable. Since only 14 cells at a time are used, performance is limited by the throughput of the particle data caches. The optimization goal, therefore, is to increase the number of caches, and therefore bandwidth, without introducing too much overhead. The key to the solution is shown in Figure 5(b): The BRAM arrays are transposed so that they now store particles by particle ID with the entries representing different cell IDs (m cells).

5.1.1 Data Path Optimization

Position data can now be more rapidly distributed to filters. Since the number of particles per cell is usually < 80 a limited number of block RAMs is required. With each cache in charge of particles with the same particle ID from different cells, all neighbor particles with respect to a single reference particle can be loaded to the filter registers in 14 cycles and reused for other reference particles from the same cell. This compares with $\sim$100 cycles for Baseline 1.

As Figure 5(a) shows, the position caches and filters are highly localized. Each filter group is served only by neighbor particles from the same position cache, eliminating the highly 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. Note that the ratio 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 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 partial forces enter the adder tree to be summed. The reference force chunk is then accumulated into a force cache. With the help of the buffers, the force computation of each reference cell can be pipelined without stalls.

Another benefit is that the buffers attached to the filters now only contain neighbor cell IDs, which require less storage 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 Handling Particle Migration

With transposed caches, particle migration can also be optimized. The example in Figure 5(c) uses four transposed

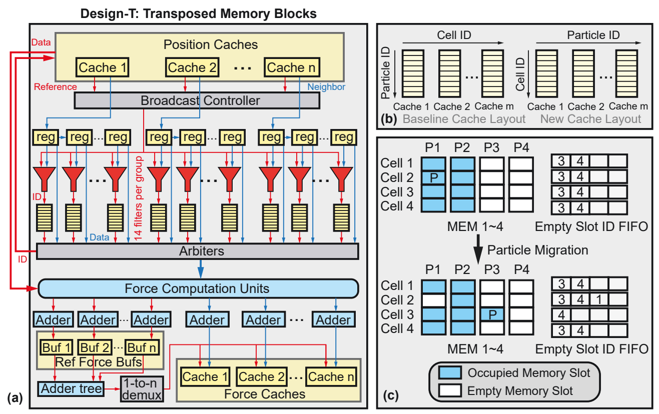


Fig. 5. The optimized architecture for the first mapping scheme. (a): the data flow in force evaluation. (b): the new memory layout compared to baseline. (c): particle migration handling in motion update. P is the particle being migrated from cell 2 to cell 3.

caches. The last two caches are initially empty to ensure no particle is lost during migration. In practice, we use $\sim$25% more caches in case many particles are migrated into the same cell. Let particle P move from Cell 2 to Cell 3. If a slot is available in the empty-slot-ID FIFO of Cell 3, then it is used. Meanwhile, the migrated particle leaves a vacated entry in the original cache, and the index of the entry is pushed to its empty-slot-ID FIFO.

After migrating to a new memory block, particle P may get updated again by the motion update unit updating column P3, since double buffering is not used in this design. The data integrity is not harmed, however, because the empty slot particle P moves into is associated with zero force. Note that the empty-slot-ID FIFOs are very small and can be constructed using a few registers.

After multiple MD iterations, the empty slots (vacancies) become scattered among the caches. These vacancies, however, are actually advantageous. At the beginning of MD, the reserved empty caches have no work to distribute to their filters, resulting in idle compute units. As the vacancies spread, however, the empty caches are gradually filled with migrating particles: the workload becomes more balanced as the simulation proceeds.

5.1.3 Scalability

The demand for hardware resources increases little when adding cells, even scaled to $\sim$500 cells. Compared to Baseline 1 and Baseline 2, where the number of cells dominates the wiring complexity and resource consumption, the memory block transpose method allows easy addition 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 with more BRAMs and slightly modified cell selection logic.

5.2 Design-R: Ring-Based Network

This architecture is an upgrade to the 2nd and the 3rd mapping schemes: each PE works on a different cell, while allows multiple PEs to work on the same cell. In Baselines 2 and 3, the wiring complexity increases drastically when the number of PEs is increased. Also, the neighbor particle

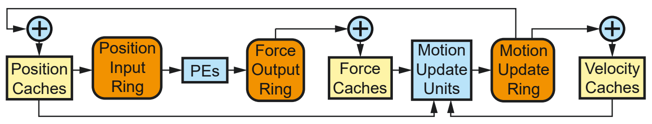


Fig. 6. RL dataflow in brief with rings. Yellow: Memory blocks; orange: Routing rings; blue: computing units.

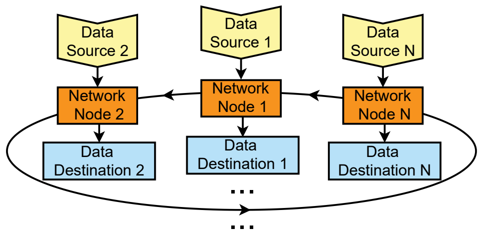


Fig. 7. The ring routing path. Position input ring: from position caches to PEs. Force output ring: from PEs to force caches. Motion update ring: from motion update units to position caches and velocity caches.

data have poor locality: neighbor positions are not reused and neighbor forces are not aggregated before being accumulated to the 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 supports multiple PEs working on the same cell, minimizes the data transfer latency in the network, and significantly reduces the number of invalid particle pairs.

5.2.1 *Memory-PE Interconnect*

To avoid frequency degradation, we replace the direct interconnects (Figure 2) with daisy-chain-based unidirectional ring (Figure 6). The 1-D topology is chosen because it is most resource-efficient while still allowing for the long latency to be hidden. The rings are used for three 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. A position packet is broadcast to multiple destinations and therefore kept in the ring until reaching its final destination. Force and motion update packets have only a single destination. Finally, the motion update ring is less congested because particle migration is relatively rare; details are therefore omitted. Figure 7 depicts the interconnect of the ring nodes. For each ring node, two data sources and two destinations are available. New data can only be injected when the node is free. Packets are either advanced to the following node or, if at the final destination, discarded.

Ring Configuration. The mapping of 3-D cells to 1-D ring is as follows:

$$I_r = N^y_{cell}N^z_{cell}(x-1) + N^z_{cell}(y-1) + z \tag{4}$$

where I_r is the index from 1 to N shown in Figure 7, N^x_{cell} is the number of cells along the x axis, and x is the coordinate of cells in the x direction, ranging from 1 to N^x_{cell}. While this mapping is not necessarily optimal, it succeeds in dramatically reducing the average packet lifetime.

5.2.2 *Data Caching*

By removing the direct connections, the various cell-PE datapaths are merged into rings. The desired throughput can still be achieved, however, by using a two-fold data caching method to reuse the input position data and to effectively aggregate the force fragments.

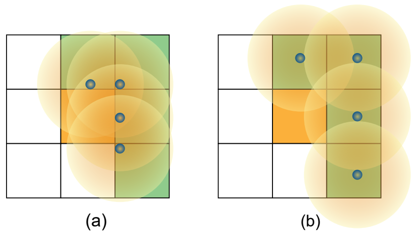


Fig. 8. 2D illustrations of (a) high filter pass rate, (b) low filter pass rate.

Neighbor Particle Caching. Instead of feeding neighbor particle data directly from neighbor caches, the neighbor data are cached in registers located with the filters. 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 significantly alleviates the pressure on data transfer from position caches to force pipelines, e.g., if two neighbor particles are evaluated together with the home cell, 50% fewer transfers are required.

Another benefit of the neighbor particle caching is balancing the workload among force computation units. Figures 8(a) and (b) show scenarios for the filtering rate, with too much and too little work, respectively. If neighbor particles are traversed with respect to a single reference particle, (a) and (b) may happen and lead to highly imbalanced filtering. In this 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 exploited, 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 all of the position data in the buffers below the filters, only the particle IDs are used (Figure 9(a)) saving BRAM 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 in (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 Dispatch*

With the data caching methods being used, other components must be adapted. To properly cache the neighbor particles, a dispatch mechanism is developed. Position data from the position input ring are injected directly into the

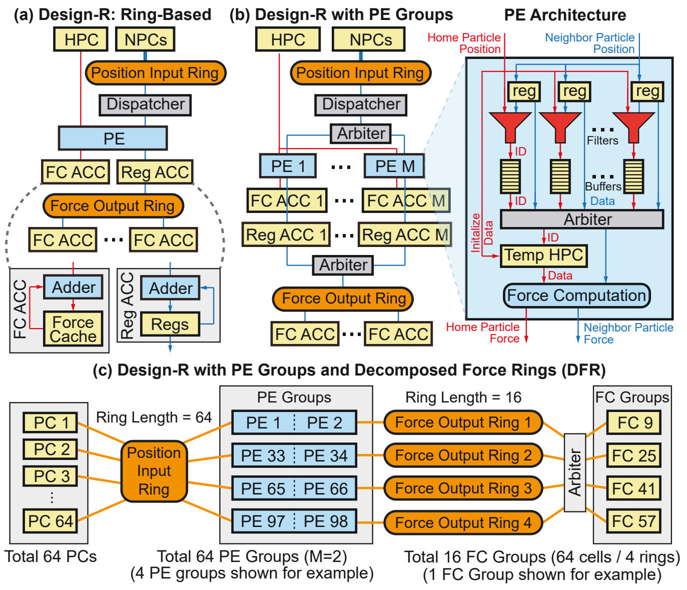


Fig. 9. The configurations of Design-R. (a): The vanilla Design-R with a single PE for each cell. (b): Supporting multiple PEs for the same cell. (c): Breaking the force ring down into shorter pieces to reduce latency.

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 whether a neighbor particle is 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 in 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 force pipelines, more than one force cache is required per cell when multiple force pipelines work 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 in 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 with force pipelines.

5.2.5 *PE Groups*

In the vanilla Design-R, a cell is assigned only one single PE, limiting the performance even with sufficient on-chip resources, especially for small simulation spaces. To address this, Figure 9(b) illustrates that the vanilla Design-R can be configured to support multiple PEs serving the same cell.

Instead of dispatching particles to a single PE, the dispatcher now serves M PEs as a PE group round-robin, where each PE independently pairs the home particle position (broadcast from HPC) and the registered neighbor position (from the dispatcher).

To meet the throughput of the output home particle forces (almost one force produced per cycle), each PE is assigned a home force cache to store the frequently produced home particle forces, while sharing the same neighbor force output datapath via arbitration.

5.2.6 *Decomposed Force Rings*

With more PEs, the latency of the force output ring becomes the bottleneck. Compared to the position input ring, where each position can be reused in 14 places, the force output ring lacks the broadcast mechanism (each force only has one destination). The high fragmentation of partial force in the ring leads to more severe routing congestion.

Figure 9(c) shows an example solution with 64 cells and 128 PEs. It turns out that the long force ring can be decomposed into several ring pieces, reducing the latency with little cost induced. The original force output ring with 64-unit length is broken into four *stacked* rings with 16-unit length without extending the total length.

As a force travels along a Decomposed Force Ring (DFR), four exit options are available at any time. The force caches for neighboring forces are also organized into groups to accept forces from all four rings. To break the occasional tie when two or more forces are to be accumulated into the same FC, small buffers are added to the arbiter.

5.2.7 *Position Input*

The upstream position input units are also upgraded to support the force processors downstream and improve hardware utilization.

Hierarchical Filters. Some particles can be discarded before entering PEs. For example, Figure 10(a) shows a neighbor particle that cannot be paired with any particle in the home cell. As these particles waste tens of filtering cycles it is 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 that 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; data from the previous node have higher priority.

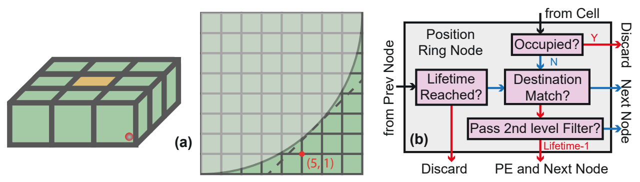


Fig. 10. (a): The 1st level filtering in a position input ring node. Dashed line: planar filter criterion. (b): The flowchart of a position input ring node.

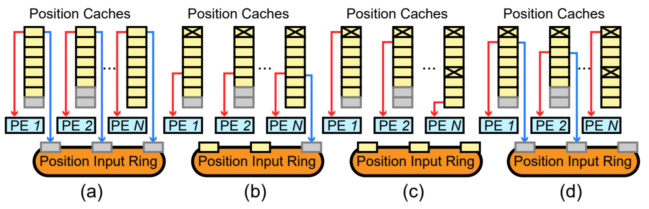


Fig. 11. The out-of-order broadcast method. Gray boxes: Empty slots. Boxes with "x": Marked as used. (a) The 1st cycle. (b) The 5th cycle. (c) The 8th cycle. (d) the 9th cycle. Red arrows: home particle position data movement; blue arrows: neighbor particle position data movement.

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 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 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), since the ring nodes are not occupied, the first entries are sent to both PEs and the ring. 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 because the slot is empty. In the eighth cycle (c), the home particles (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.

5.2.8 Supporting Larger Simulations

The above designs are based on an assumption about the proposed utilization of FPGAs in the MD ecosystem, i.e., the number of PEs is $\geq$ 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, several challenges. First, with more cells, how should extra data be stored on-chip given limited BRAM resources? Second, the ring latency increases as ring nodes are added to support additional 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 processed in the next step but are still 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. A BRAM can therefore be partitioned into several interleaved cell regions, where the two red cells are the cells to be processed in separate steps, as Figure 12(b) shows. This method also reduces the ring length as the interleaved cells can share the same ring node.

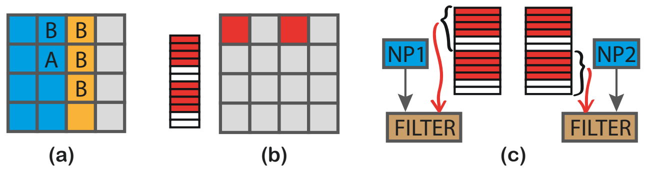


Fig. 12. The solution to the problems emerge in large scale simulations. We intuitively use 4x4 2-D cells with eight PEs for example. (a): blue: the cells currently being processed; orange: the extra cells required to process the blue cells. Particles from cell A are broadcast to all B cells. (b): memory 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.

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 method 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 used double buffers. Originally, one of the buffers is used during the force evaluation phase, while the other is used during motion update. Now with the extra concurrency demand, the cache in use can be duplicated into the previously unused cache 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. Design-T, inherited from high-level mapping scheme 1, makes MD processing straightforward, but has two problems. First, the design is difficult to extend to multiple FPGAs. There, each FPGA is charged with evaluating a certain region in a simulation space and the only data exchanges are for the 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, increasing the complexity of communication. Second, the neighbor particles need to be reloaded after the evaluation of each reference cell, with about $\sim$20% overhead introduced.

Design-R is good for multi-chip extension [32]. First, the rings allow simple data injection from external FPGA chips such that little modification is required: the external data can be treated as local data when being processed. Second, a multi-chip extension allows multiple PEs to work on the same cell, such that a single chip only works on a small number of cells, reducing the length of rings. As a result, more PEs can be accommodated without data congestion.

Design-R also provides a higher degree of parallelism given increasing on-chip resources. The performance of Design-T is limited by the number of particles per cell. For example, if each cell contains 64 particles, a reference particle is paired with 64$\times$14 neighbor particles, where 14 is the number of neighbor cells. If each PE has 7 filters, only 128 PEs can be fully utilized. In contrast, with sufficient PEs and enough routing resources, the neighbor position data are immediately processed upon arriving at the PEs, and evaluated with the traversed home cell particle positions. Additionally, the resulting forces are accumulated into force caches immediately given sufficient routing resources to hide the latency. Therefore, in the limit, each iteration is finished with the number of cycles required for 2× position cache data traversal plus the routing latency and pipeline depth, totaling a few hundred cycles for 64 particles per cell. However, in actual experiments with current resources, the number of cycles is an order of magnitude higher.

6 Evaluation

6.1 Experiment Setup

The designs are implemented on a Xilinx Alveo U280 acceleration card, which has 1304K CLB LUTs, 2607K CLB registers, 2016 block RAMs, 960 ultra RAMs, and 9024 DSPs.

We assume each cell initially has 64 particles, which is typical in this domain. For motion update, all scenarios for all designs have eight units; this is sufficient for the time spent in motion update to be negligible compared with force evaluation. Baselines 1 and 2 (B1, B2) and Design-T have a fixed number of PEs. The former two designs have 128 PEs while the latter has 160 PEs (with 20% PEs reserved to handle particle migration). Baseline 3 (B3) and Design-R have a number of PEs equal to the number of cells in the simulation space. Design-R can also be configured to accommodate more PEs per cell, resulting in up to 4× PEs as many as the number of cells.

We use ns/day as the unit of performance, referring to the simulated time that can be processed within a wall clock day. The baselines and the optimized designs run at 70 MHz (the highest frequency achievable for $3 \times 3 \times 3$ cells to simplify the comparison) and 200 MHz, respectively.

6.2 Performance Comparison Among the Designs

6.2.1 Simulation Speed Comparison

The simulation results of all baselines, Design-T, and a variety of Design-R configurations are given in Figure 13, where *RH-xPE-yDFR* refers to x PEs working on the same cell and y decomposed force rings with hierarchical filtering. To clearly demonstrate the difference of Design-R configurations, a linear y-axis is used.

Baselines 1 and 2 are heavily affected by the overheads discussed in Section § 4. Also, all three baselines are influenced by their poor routing scalability, leading to an overall lower operating frequency. For the optimized designs, Design-T benefits from the large number of PEs to achieve sustained performance across the simulation space sizes, while Design-R and its configuration variations achieve higher performance for the simulation space smaller than $6 \times 6 \times 6$ cells.

The blue bars indicate that in the case of only 1 PE per cell, the force ring latency can be completely hidden when the number of cells is small, suggesting that it is unnecessary to increase the number of DFRs. With more PEs per cell, to push the performance to a higher level, more DFRs are needed to hide the latency (see the red bars).

6.2.2 PE Efficiency Comparison

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

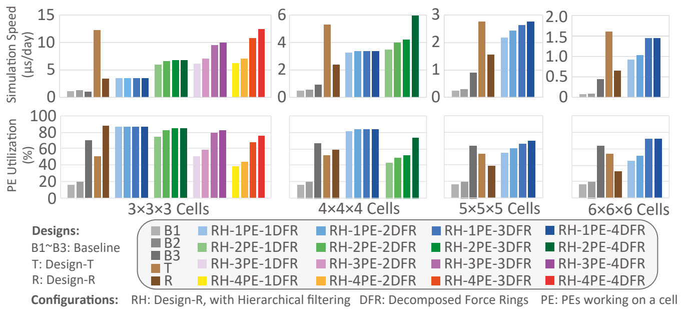


Fig. 13. The ns/day performance and PE utilization of three baseline and two optimized designs with different configurations. 5 filters are equipped for Design-R and its variations.

Results are shown in Figure 13. We observe that B1 and B2 have a substantially lower PE utilization (15%-20%). The greatest disadvantage of the two baselines lies in the global synchronization, which is used to prevent forces of different reference particles from entering the adder trees. Another important factor for B1 is the neighbor particle pre-loading process, which introduces considerable overhead.

Compared with the two baselines, Design-T improves utilization by more than 30%. With the buffers above the adder tree, only forces of the same reference particle can enter the adder tree making global synchronization unnecessary. However, the reserved PEs and overhead of pre-loading neighbor particles limit the PE utilization.

B3 and most of the Design-R variations have significantly higher PE utilization. The two designs fully stream the particles without pre-loading. Compared with Design-R, B3 suffers from the overhead of global synchronization. Design-R, however, has its utilization drop for larger simulation spaces due to the over-extended rings, especially the force output ring where there is heavy congestion. That drop in utilization, however, can be offset with the hierarchical filters and DFRs, boosting the Design-R PE utilization from $\sim 30\%$ to $> 75\%$ for $6 \times 6 \times 6$ cells.

6.3 Evaluating Configurations of Design-R

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 previously computing zero forces. Without hierarchical filtering, every force register in Figure 9(a) embedded in Reg ACC requires zero checking before injecting the forces into a force output ring. 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 Design-R without zero force checking with the modified O2 with hierarchical filters.

Figure 14 shows the performance of the Design-R configurations. In the 3x3x3 cell space case, the RH-1PE-1DFR version reaches equilibrium at four filters, and the vanilla Design-R at six filters. For larger cell spaces, they reach equilibrium with almost the same number of filters, but with varied performance. This is because the performance bottleneck lies mostly in the force output. The rings reach

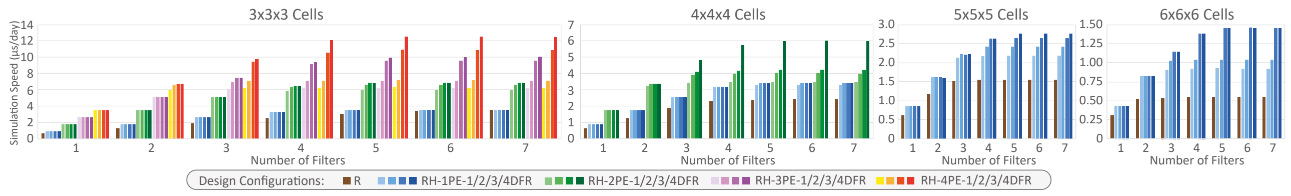


Fig. 14. The performance comparison of the vanilla Design-R and different configurations for four representative cell spaces.

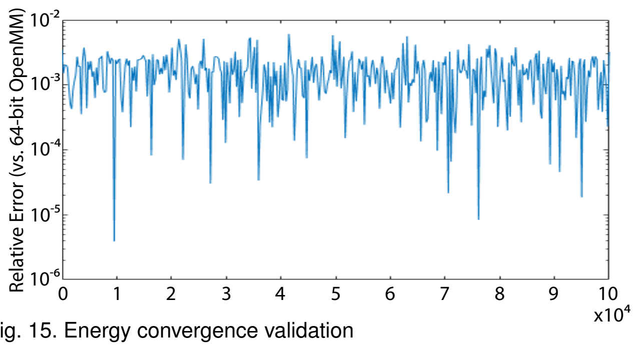


Fig. 15. Energy convergence validation

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 better performance with a heavily congested force output ring.

6.3.2 Number of PEs and Decomposed Force Rings

As the number of PEs per cell increases while the number of DFRs is fixed to 1, the equilibrium is also reached at 4 filters for $3 \times 3 \times 3$ cells. The exceptions are the 3 and 4 PE cases, as adding more PEs is equivalent to adding more filters with the same force throughput. In fact, with more PEs, the latency problem of the force output ring becomes much more severe. This situation is similar for $4 \times 4 \times 4$ cells.

With the force ring decomposition method adopted, even with a large number of PEs per cell, equilibrium is reached at $\sim$ 4 - 5 filters, which is consistent with the theoretical filtering rate (25%) for hierarchical filters. This suggests that the decomposed force rings effectively reduce force ring latency and overcome the performance bottleneck.

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 for both small and large simulation spaces are demonstrated with the best configuration of Design-R as the reference.

Table 1 shows the performance results. For small simulation spaces from 3×3×3 to 5×5×5 cells, Design-R achieves 4.28× and 2.81× speedup with respect to GPU, with slightly higher performance compared to Design-T. The best configuration of Design-R refers to 4PE-4DFR for 3×3×3 cells, 2PE-4DFR for 4×4×4 cells, and 1PE-4DFR for 5×5×5 cells.

For simulation spaces $\geq$ 6x6x6, the performance of Design-R and Design-T are comparable, while the performance of GPU is superior, as the workload is large enough for the GPU to parallelize. The best configuration corresponds to 1PE-4DFR for all three cases, with 108, 128, and 128 PEs available, respectively.

6.5 Hardware Utilization

To study the availability of Design-T and Design-R with configurations, Table 2 highlights the hardware resources consumption for the designs in the representative cell spaces evaluated above. The number of PEs per cell and the number of decomposed force rings are optimal to ensure the performance corresponding to the hardware. Due to the limited number of BRAMs, some of the small buffers are implemented with logic as LUTRAMs.

Note that the resource utilizations for the 8x8x8 and 8x8x12 cell spaces are nearly identical. This is because the Ultra RAMs with which the cache is implemented are not fully utilized and therefore can contain data from additional cells with no marginal memory cost. Most significantly, the number of PEs remains unchanged.

6.6 Energy Validation

Energy convergence validation is performed on a 20K liquid argon dataset. Figure 15 compares the reference double-precision floating point OpenMM result with the implementation described in this paper (23-bit offset precision scheme in Figure 15(c) and the linear interpolation described in 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 [33], [34], [35], [36], [37], [38], [39], [40], [41]. 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.

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 dimensions. The first scales with the size of a simulation space with a constant number of PEs, while the second allows the number of PEs to increase directly with the increased size of a simulation space, provided there are sufficient hardware resources on the chip. In addition, the Design-R configurations are proven useful in achieving higher performance by boosting computing power and hiding the routing latency.

The overall motivation for using FPGAs for MD computation is that FPGA-based clusters fill a vital niche: scalable

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

Cell #	Particle #	Performance	Design-R (best config)	Design-T	GPU	CPU-1x	CPU-2x	CPU-4x	CPU-8x	CPU-16x	CPU-32x
3x3x3	1728	ns/day	12495	12222	2921	80.06	135.5	204.6	236.3	183.4	152.9
		speedup	1.00	1.02	4.28	156	92.2	61.1	52.9	68.1	81.7
4x4x4	4096	ns/day	5978	5303	2126	35.25	60.56	92.8	110.8	95.32	101.5
		speedup	1.00	1.13	2.81	170	98.7	64.4	54.0	62.7	58.9
5x5x5	8000	ns/day	2905	2750	1841	19.35	32.79	60.05	88.7	78.03	77.5
		speedup	1.00	1.06	1.58	150	88.6	48.4	32.8	37.2	37.5
6x6x6	13824	ns/day	1456	1602	1542	11.05	18.72	31.42	49.19	53.12	52.7
		speedup	1.00	0.91	0.94	132	77.8	46.3	29.6	27.4	27.6
8x8x8	32768	ns/day	644.5	680.4	801.8	5.258	8.818	15.95	20.92	28.54	33.19
		speedup	1.00	0.95	0.80	123	73.1	40.4	30.8	22.6	19.4
12x8x8	49152	ns/day	429.7	454.5	618.7	3.490	5.784	10.55	17.63	21.54	27.06
		speedup	1.00	0.95	0.69	123	74.3	40.7	24.4	19.9	15.9

TABLE 2
Hardware Utilization of FPGA-O2 with spatial configurations.

Design	Number of Cells	Number of Particles†	Number of PEs	LUT	FF	BRAM	URAM	DSP
T	<512	<32768	160	1065237 (82%)	859815 (33%)	1544 (77%)	800 (83%)	8494 (94%)
	[512, 1024)	[32768, 65536)	160	1086542 (83%)	877011 (34%)	1784 (88%)	800 (83%)	8494 (94%)
R (4DFR)	3x3x3	1728	108	789863 (61%)	661095 (25%)	942 (47%)	648 (68%)	5136 (57%)
	4x4x4	4096	128	947757 (73%)	803750 (31%)	1112 (55%)	768 (80%)	6056 (67%)
	5x5x5	8000	125	957627 (73%)	832411 (32%)	1087 (54%)	750 (78%)	5918 (66%)
	6x6x6	13824	108	885157 (68%)	803261 (31%)	942 (47%)	648 (68%)	5136 (57%)
	8x8x8	32768	128	980026 (75%)	851891 (33%)	1112 (55%)	768 (80%)	6056 (67%)
	8x8x12	49152	128	996160 (76%)	875962 (34%)	1112 (55%)	768 (80%)	6056 (67%)

† 64 particles per cell as a reference value.

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, Design-R achieves a 4.28x speed-up for 1728 particles compared to GPU, whereas Design-T achieves 4.18x. 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.

Acknowledgments

This work was supported, in part, by the NSF through awards CCF-1919130, CCF 2151021, and SHF-2326494; the NIH through award R44GM128533; 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. This research was partially supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 78284 and platform resources under award 66150.

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," *ACS Central Science*, vol. 6, 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," *J. 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," *J. Chem. Info. and Modeling*, vol. 60, no. 11, pp. 5595–5623, 2020.

[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," *J. Chemical Theory and Computation*, vol. 16, no. 9, pp. 5512–5525, 2020.

[7] Z. Cournia, B. K. Allen, T. Beuming, D. A. Pearlman, B. K. Radak, and W. Sherman, "Rigorous free energy simulations in virtual screening," *J. Chem. Info. and Modeling*, vol. 60, no. 9, 2020.

[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," *J. Biomolecular Structure and Dynamics*, 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," *J. 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. Herbordt, "System-Level Modeling of GPU/FPGA Clusters for Molecular Dynamics Simulations," in *IEEE High Performance Extreme Computing Conference*, 2021.

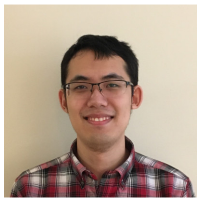
[16] J. Glaser, T. D. Nguyen, J. A. Anderson, P. Lui, F. Spiga, J. A. Millan, D. C. Morse, and S. C. Glotzer, "Strong scaling of general-purpose molecular dynamics simulations on gpus," *Computer Physics Communications*, vol. 192, pp. 97 – 107, 2015.

[17] J. Sheng, B. Humphries, H. Zhang, and M. Herbordt, "Design of 3D FFTs with FPGA Clusters," in *IEEE High Performance Extreme Computing Conference*, 2014.

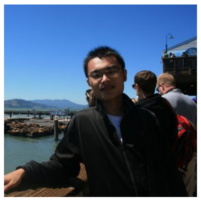
[18] A. Lawande, A. George, and H. Lam, "Novo-G#: a multidimensional torus-based reconfigurable cluster for molecular dynamics," *Concurrency and Computation: Practice and Experience*, 2016.

[19] J. Sheng, C. Yang, A. Caulfield, M. Papamichael, and M. Herbordt,

"HPC on FPGA Clouds: 3D FFTs and Implications for Molecular Dynamics," in *Int. Conf. on Field Prog. Logic and Apps.*, 2017.
[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] M. Chiu and M. Herbordt, "Molecular dynamics simulations on high performance reconfigurable computing systems," *ACM Trans. Reconfig. Tech. and Systems*, vol. 3, no. 4, pp. 1–37, 2010.
[24] C. D. Thompson, "Area-time complexity for VLSI," in *11th Annual ACM Symposium on Theory of Computing*, 1979, pp. 81–88.
[25] C. Wu, T. Geng, S. Bandara, C. Yang, V. Sachdeva, W. Sherman, and M. Herbordt, "Upgrade of FPGA Range-Limited Molecular Dynamics to Handle Hundreds of Processors," in *29th Int. Symp. on Field-Prog. Custom Computing Machines*, 2021, pp. 142–151.
[26] M. Chiu and M. Herbordt, "Efficient filtering for molecular dynamics simulations," in *Int. Conf. Field Programmable Logic and Applications*, 2009.
[27] T. Darden, D. York, and L. Pedersen, "Particle Mesh Ewald: an $N\log(N)$ method for Ewald sums in large systems," *J. Chemical Physics*, vol. 98, pp. 10 089–10 092, 1993.
[28] C. Wu, S. Bandara, T. Geng, A. Guo, P. Haghi, W. Sherman, V. Sachdeva, and M. Herbordt, "Optimized Mappings for Symmetric Range-Limited Molecular Force Calculations on FPGAs," in *Int. Conf. Field-Programmable Logic and Applications*, 2022.
[29] 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.
[30] 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.
[31] 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.
[32] C. Wu, T. Geng, A. Guo, S. Bandara, P. Haghi, C. Liu, A. Li, and M. Herbordt, "FASDA: An FPGA-Aided, Scalable and Distributed Accelerator for Range-Limited Molecular Dynamics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023.
[33] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, "Reconfigurable molecular dynamics simulator," in *IEEE Symp. on Field Prog. Custom Computing Machines*, 2004, pp. 197–206.
[34] Y. Gu, T. VanCourt, and M. Herbordt, "Accelerating molecular dynamics simulations with configurable circuits," in *IEEE Conf. Field Programmable Logic and Applications*, 2005.
[35] T. Hamada and N. Nakasato, "Massively parallel processors generator for reconfigurable system," *Symp. Field Prog. Custom Computing Machines*, 2005.
[36] R. Scrofano and V. Prasanna, "Preliminary investigation of advanced electrostatics in molecular dynamics on reconfigurable computers," in *ACM/IEEE Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2006.
[37] Y. Gu, T. VanCourt, and M. Herbordt, "Accelerating molecular dynamics simulations with configurable circuits," *IEE Proc. on Computers and Digital Technology*, vol. 153, no. 3, pp. 189–195, 2006.
[38] V. Kindratenko and D. Pointer, "A case study in porting a production scientific supercomputing application to a reconfigurable computer," in *Symp. Field Prog. Custom Comp. Machines*, 2006.
[39] 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.
[40] M. Schaffner and L. Benini, "On the feasibility of FPGA acceleration of molecular dynamics simulations," arXiv1808.04201, 2018.
[41] 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, 2022.

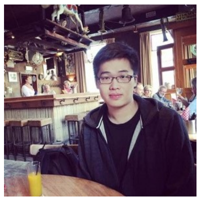


Chunshu Wu received his bachelor's and master's degrees in physics from Dalian University of Technology, Liaoning, China in 2016 and from Brown University in 2018 respectively. After that, he joined ECE department of Boston University as a graduate student. Chunshu's current research is on FPGA-based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

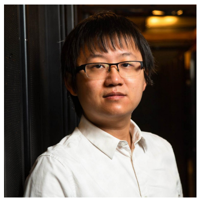
Sahan Bandara received his bachelor's degree in Electronic and Telecommunication Engineering from the University of Moratuwa, Sri Lanka in 2015, and his master's degree in Computer Engineering from Boston University in 2019. His research interests are computer architecture, hardware security, and hardware 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. 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.

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.