Optimized Mappings for Symmetric Range-Limited Molecular Force Calculations on FPGAs

Chunshu Wu*, Sahan Bandara*, Tong Geng*†, Anqi Guo*, Pouya Haghi*,
Vipin Sachdeva‡, Woody Sherman‡, Martin Herbordt*
*Dept. of Electrical and Computer Engineering, Boston University
†Dept. of Electrical and Computer Engineering, University of Rochester ‡Roivant Sciences, Boston, MA
Email: *{happycwu, sahanb, tgeng, anqiguo, haghi, herbordt}@bu.edu,
†tong.geng@rochester.edu, ‡{vipin.sachdeva, woody.sherman}@roivant.com

***Abstract*—In N-body applications, the efficient evaluation of range-limited forces depends on applying certain constraints, including a cut-off radius and force symmetry (Newton's Third Law). When computing the pair-wise forces in parallel, finding the optimal mapping of particles and computations to memories and processors is surprisingly challenging, but can result in greatly reduced data movement and computation. Despite FPGAs having a distinct compute model (BRAMs/network/pipelines) from CPUs and ASICs, mappings on FPGAs have not previously been studied in depth: it was thought that the half-shell method was preferred.**

In this work, we find that the Manhattan method is surprisingly compatible with FPGA hardware. With the cache overlapping technique proposed in this paper, the ultra-fine-grained data access demanded by the Manhattan method can be satisfied, despite the fact that the memory blocks on FPGAs appear to be insufficiently fine-grained. We further demonstrate that, compared to the traditional baseline half-shell method, approximately a half of the filters (preprocessors) can be removed without performance degradation. For communication, the amount of data transferred can be reduced by 40% - 75% in the most common multi-FPGA scenarios. Moreover, data transfers are almost perfectly balanced along all directions, and the optimization requires only minimal hardware resources. The practical consequence is that nearly $2\times$ to $4\times$ the workload can be handled without upgrading the network connections between FPGAs. This is a critical finding given the relatively limited bandwidth available in many common accelerator boards and the strong-scaling applications to which FPGA clusters are being applied.

I. Introduction

Exploiting physical symmetries to reduce computation is a fundamental method in Scientific Computing. In Molecular Dynamics (MD), Newton's 3rd law (N3L) results in the force symmetry that reduces by half the computation in short range (or range-limited, RL) force evaluation. The theory is straightforward, but, when coupled with the commonly used hard cut-off (range limit), the optimal computation-processor-memory mapping is dependent on the hardware model. N-body mapping has been the focus of several high-profile studies [1]–[3], where particle, space, and force partitioning are justified. However, the models are not hardware-specific and, in particular, the mapping model for FPGA-based MD [4]–[11] remains to be established. The problem reduces to the following: How should particle data be organized with respect to block RAM (BRAM) configuration to minimize data access redundancy?

MD is the iterative application of Newtonian mechanics to ensembles of particles and alternates between force and motion computations; the latter is responsible for $< 1\%$ of the operations. A particle is defined by position, velocity, and force applied where each data component is $\sim$100 bits. The force computation is partitioned into RL and Long Range (LR) components. For the models of most interest for FPGA clusters [11], RL dominates both computation and communication. RL involves force calculations between particle pairs within a cut-off radius. N3L dictates that the RL force only needs to be computed once per particle pair. LR is generally computed with transform-based methods, such as PME [12], and, while critical for large system scalability [13]–[19], is not considered further here.

Unlike application specific integrated circuits (ASICs), FPGAs have pre-determined on-chip storage configurations, i.e., BRAM sizes, e.g., 20Kb for the M20Ks on an Intel Stratix 10. Since current BRAMs typically have a depth of 512, a block of BRAMs (whose number depends on the BRAM configuration) is used to store up to a few hundred particles, which are accessed sequentially rather than concurrently. A commonly used optimization is to group particles geometrically in cells, namely, with cell-lists [20], [21]. In previous FPGA-based MD work (since, e.g., [22], [23]), the edge length of a cubic cell is fixed at the cutoff radius (a criterion to neglect small forces, see Figure 1(a)); with current practice [24], a cell typically contains $\sim$80 particles in a water environment. The particles from a cell are evaluated in serial with respect to particles from neighboring cells (see [25] for why this access schedule is preferred for FPGAs).

Larger or smaller cells are less advantageous. Larger cells result in redundancy in computation, which directly degrades performance as a particle only interacts with a smaller portion of particles in neighboring cells, yet all the neighboring particles need to be accessed. For smaller cells, more neighbor cells need to be accessed for a particle, and a cell contains a smaller number of particles, so more slots in a BRAM are wasted. Subboxing [26] in Anton 2, for example, is an extreme case of using small cells for more fine-grained data access, but not suitable for FPGAs based on the discussion above.

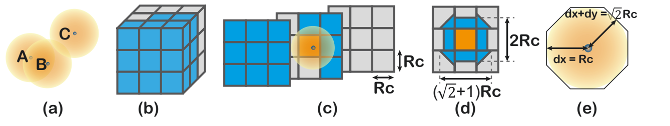


Fig. 1. Cells, spatial partitioning methods, and filtering. R_c: Cutoff radius. (a): the cutoff radius of particles. The force A-B is valid, and A-C and B-C forces are too small and neglected. (b): the import volume of the half-shell method. (c): the spread layers of (b), where the orange cell at the center interacts with 13 blue cells and itself; the middle slice also shows the 2-D import volume. (d): 2-D import volume of the Manhattan method. (e): planar filtering method. dx and dy are the distance components between two particles; particles outside the octagon are not paired with the particle at the center.

Note that the advantages of larger boxes accrued in CPU implementations, where they reduce frequency of neighbor list recalculation [27], are not applicable to ASICs/FPGAs where particle filtering [28] is used instead.

The cell size now being set, it is straightforward to evaluate a cell, with N3L applied, with respect to the surrounding 14 cells (including itself) using the half shell method [10] (illustrated in Figures 1(b) and 1(c)). The major downside, however, is the *import volume*; i.e., 13 external cells need to be imported. An alternative is the Manhattan method used by the Anton 3 [29]. Compared to the half-shell method, it demands a much lower import volume, as Figure 1(d) suggests in 2-D illustration. In the 3-D case, the import volume is reduced to the equivalent of about 7 cells.

Applying the Manhattan method to FPGAs, however, appears to require that cells be further partitioned into fine-grained subboxes, with the disadvantages just described. To tackle this problem, we

- propose a position cache overlapping method that maps our modified Manhattan method onto FPGA hardware without reducing the size of cells;
- design a complete MD RL architecture with minimal additional resources cost compared to the baseline half-shell design on FPGA;
- demonstrate that the Manhattan method used on multi-FPGA clusters can reduce the data transfer by 40% - 75%, while balancing data transfers along all directions.

The practical consequence is that nearly $2\times - 4\times$ the workload can be handled without upgrading the network of FPGA connections. This is a critical finding given the relatively limited bandwidth available in many common accelerator boards and the strong-scaling, communication-bound, applications to which FPGA clusters are being applied.

II. Background

A. Range-Limited Force

The RL force between particle i and j (equation 1) is derived from the Lennard-Jones potential (equation 2), where σ is the critical distance of i and j at which the potential is 0, and ϵ is the energy parameter determined by the substance.

$$F_i^{LJ} = \sum_{j \neq i} \frac{\epsilon_{ij}}{\sigma_{ij}^2}[48(\frac{\sigma_{ij}}{|r_{ji}|})^{14} - 24(\frac{\sigma_{ij}}{|r_{ji}|})^8]\mathbf{r}_{ji} \tag{1}$$

$$V_{ij}^{LJ} = 4\epsilon_{ij}[(\frac{\sigma_{ij}}{r_{ij}})^{12} - (\frac{\sigma_{ij}}{r_{ij}})^6] \tag{2}$$

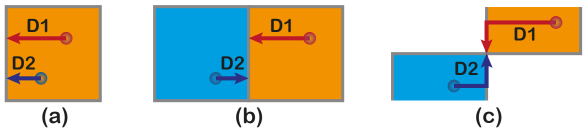


Fig. 2. The mechanism of the traditional Manhattan method shown in 2-D. The particle pairs are evaluated by the processor located in orange cells for 3 particle pair scenarios: (a) both in same cell; (b) in neighboring cells sharing a boundary; (c) in neighboring cells sharing a corner.

From the r^6 and r^{12} terms, it is evident that the RL forces decay rapidly with r. The cutoff radius (see Figure 1) is then reasonably applied to remove the negligible interaction between two distant particles: two particles are only paired when their distance is smaller than R_C. As a result, the $O(N^2)$ complexity becomes $O(N)$.

B. Particle Filtering and approximating R_C

Computing the ideal cutoff involves calculating $r^2 = dx^2 + dy^2 + dz^2$. It has previously been shown [28], [30] that this computation can be drastically reduced by using a small number of bits and by avoiding the multiplies by approximating the sphere with a polyhedron (an octagon in 2-D, see Figure 1(e)).

C. Traditional Manhattan Method

In RL forces are generally computed systematically, one *home* particle at a time, with respect to all particles within its cut-off. Because of N3L, for a particle pair p_1, p_2, the question arises whether $F_{1,2}$ should be computed when p_1 is the home particle or p_2. Figure 2 illustrates three scenarios. When both particles are in the same cell, there is an arbitrary tie-breaker (shown is "furthest to the right"). Otherwise, the home particle is the one with the greater Manhattan distance to the boundary of the two cells.

III. Design

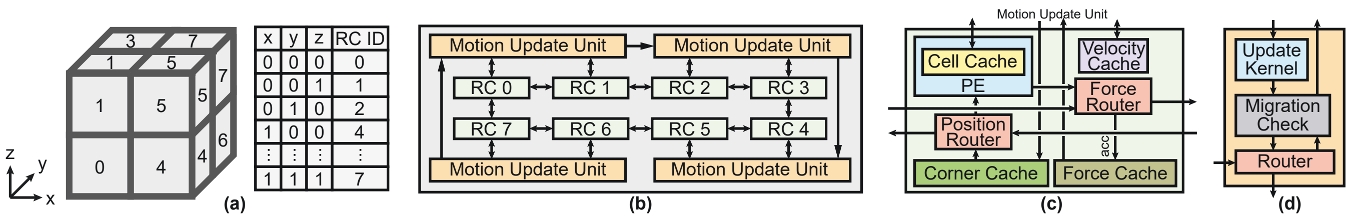
As fine-grained concurrent data access is incompatible with common FPGA BRAM configurations, we demonstrate that the use of two sets of position caches (cell caches and corner caches, see Section III-B) with overlapped contents significantly reduces memory accesses, even without fine-grained space partitioning. To coordinate the two sets of position caches, the memory misalignment problem of the particles is also resolved with address and cell indexing methods. The BRAM allocation, filters, and rings are evolved from the baseline design described in [31].

A. Logical Topology

The ring-shaped topology is inherited from the baseline design with minor modifications. To demonstrate the ring topology, we label each cell in space with a cell ID, which follows the simple 3-D to 1-D mapping method:

$$CID = xD_yD_z + yD_z + z \tag{3}$$

where D_y and D_z are dimensions of the y and z directions. For example, in Figure 3(a), D_y and D_z are both 2. Each cell is



x	y	z	RC ID
0	0	0	0
0	0	1	1
0	1	0	2
1	0	0	4
⋮	⋮	⋮	⋮
1	1	1	7

Fig. 3. The overall layout of the design. (a): cell ID numbers are calculated from their x, y, and z coordinates in space. (b): the topology of 8 ring cores (RC) and 4 motion update units (MUs) for example. (c): the internals of a ring core. (d): the internals of a MU.

assigned a ring core (RC) to process the interactions between the local particles in the cell and neighbor particles from other ring cores. The ring cores possess the same IDs as their local cells. In Figure 3(b), the RCs are logically connected as a ring corresponding to their IDs. This 3-D cell to 1-D ring mapping is both simple and minimizes the data travel time in the ring. In the case shown in the figure, two adjacent RCs share one motion update unit, and the motion update units are connected in a ring to resolve particle migration (i.e., when a particle travels to another cell).

An RC block diagram is given in Figure 3(c). The position, velocity, and force data, with respect to the particles in the local cell, are stored in cell cache, velocity cache, and force cache, respectively. In distinction from the baseline design, a duplicate cell cache is now a corner cache. A corner cache consists of ∼1/8 particles from the local cell and particles from 7 other cells. The corner cache is a key concept in this work and is discussed further in Section III-B. The position input ring (i.e., the network of position routers from RCs) and the force output ring forward data in opposite directions. Since the computed forces are returned to the RCs where the position data originates, the effect is to reduce the data travel time.

Figure 3(d) shows the functions of a motion update unit (MU). A MU gathers particles' position, velocity, and force data and computes the particles' position and velocity after, say, 2 femtoseconds ($2^{-15}s$). If a local particle has moved to another cell, the migration-check function passes the updated data to the router so the data is forwarded to the target MU through the motion update ring. The router in the target MU recognizes that the correct data have arrived and forwards it to the corresponding RC for update.

B. Corner Caches and Overlapping Position Caches

Figure 4 shows the principle of cell caches (CEC) and corner caches (COC). The particles in the caches are named as cell particles (CEP) and corner particles (COP), respectively. Both CECs (yellow) and COCs (green) contain particle position information (i.e., both, unless otherwise noted, are position caches, to be distinguished from the caches containing velocity and force data. There are also velocity and force CECs with the same spatial layout as position CECs, but there is no velocity or force COCs.). A COC is overlapped with a quarter (an eighth in 3-D) of each CEC, and that portion of position data is duplicated. The COCs cover the entire simulation space, making the number of COCs equal to the number of CECs. At runtime, execution proceeds with particles from a COC successively being broadcast to all adjacent cells to be evaluated with respect to all of the CEPs.

Position cache duplication is also necessary in the half-shell baseline method for home CEP caching in the force computation; otherwise the data locality is lost and all of the particle position data would need to be stored in the buffers attached to the filters.

C. The Modified Manhattan Method

In Figure 4, the blue particle is a COP stored in the COC and is sent to all 4 neighboring cells for pairing. The pairing is done by local PEs associated with the cells. Now that the cached positions in the CECs and COCs overlap, the traditional Manhattan method needs to be modified: because there's no clear boundary between a COC and a CEC, we compare the Manhattan distances to the corner cell boundary instead of the boundary of two cells.

Figure 4(a) and (b) show the two common particle pair cases. Both cases indicate that the pair should be evaluated when the COP has greater Manhattan distance ($D2 > D1$, opposite to the traditional), otherwise the pair may not be evaluated (the bottom-right particle is never sent to the cell holding the top-left particle).

To better illustrate the handling of the irregularity for the modified Manhattan method, we define two concepts: *shadow region* (SR) and *shadow particle* (SP). The SRs are the brown areas in Figure 4(c) and the SPs are the particles located in SRs. As the figure illustrates, the two particles are within the cutoff range and form a valid pair. However, the red particle is outside the yellow cell area. If the blue particle only interacts with the particles in its four surrounding square cells (the closest cell to the red particle is the yellow cell), then the red particle is never evaluated with the blue particle and vice versa. In other words, the two particles are not mutually included in each other's import region. Therefore, we duplicate the particles from the SRs in other cells to the related CECs. If a particle is updated into a new SR during motion update, it is then updated to more than one destination.

D. Cell Cache Partitioning and Corner Particle Pre-checking

Figure 4(d) shows the total accessed area, including the SRs. Without optimization, 71% more particles are accessed for a COP in the 3-D scenario, leading to a drastically decreased particle pairing rate. Although the size of SRs looks formidable, this situation can be improved.

Fortunately, in Figure 4(d), only the dark brown areas may be needed for the COC at the center. Therefore the SPs can use the addressing spaces separated from the normal particles in

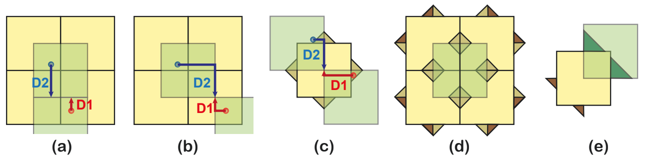


Fig. 4. The rules of position cache overlapping and the modified Manhattan method in 2-D. Yellow cells: CECs. Green: COCs. The red/blue particles are from COCs/CECs, separately. (a) and (b): the pair is evaluated if the neighbor particle has greater Manhattan distance. (c): a cell is slightly bigger than a cube (with the brown SRs), otherwise the two particles will never be paired. (d): SR overview. Dark brown: the SRs potentially need to be accessed by COPs. (e): Only the COPs in the dark green region can may be paired with the SPs in the CEC.

the CEC (e.g. normal particles are located at address 1∼120, the SR requested by its northeast corner cell is at 121∼140). The SRs are not accessed if not requested. Furthermore, Figure 4(e) shows that only a small number of COPs (in dark green regions) may request access to the SR, meaning the access can be further reduced by checking the COPs before pairing. In 3-D, ∼50% of the COPs can be exempted from accessing the SRs by pre-checking their position values.

Although the CECs look highly irregular, the actual operations required are only fixed point number additions and comparisons, and only the leading bits of the position data are used for such operations. For the rest of the paper, the CECs are illustrated as cubic cells for simplicity.

E. Architecture

The design flow is depicted in Figure 5(a) from a different perspective compared with the overall layout in figure 3. Here we mainly focus on the behaviors of PEs and abstract the caches and routing components into block diagrams for conciseness. The execution of MD RL starts from the COCs. The particle positions are injected into the position input ring as sketched in Figure 5(b), and each position has only 8 (instead of 14) destinations corresponding to 8 cells in 3-D. This also reduces the lifetime of packets in the ring. For each cycle, a position packet stops at a force processing element (PE) to check if a destination has been reached. Once there, the position packet is buffered and dispatched to one of the filters by the dispatcher.

Before dispatching a COP downstream, the particle is first pre-checked to determine if any SR needs to be accessed based on its position. After pre-checking, the requested SR ID is obtained. The ID is further used in SR access handling as Figure 5(c) shows. For the left case, COPs 0 and 1 do not need to access SRs, but 2 and 3 request both SRs be accessed. As a result, all CEPs, including the SPs, are traversed for all the COPs. For the right case, SR 1 is not requested, only the regular CEPs and particles in SR 2 are traversed. In practice, there are up to 8 SRs corresponding to 8 corner blocks surrounding a cell block, and the number of COPs processed simultaneously in a single PE is relatively small.

The filters are originally used to evaluate the distance of two particles and check if the pair is valid. At runtime, each filter is mapped to a single COP and vice versa. The Cell BRAM

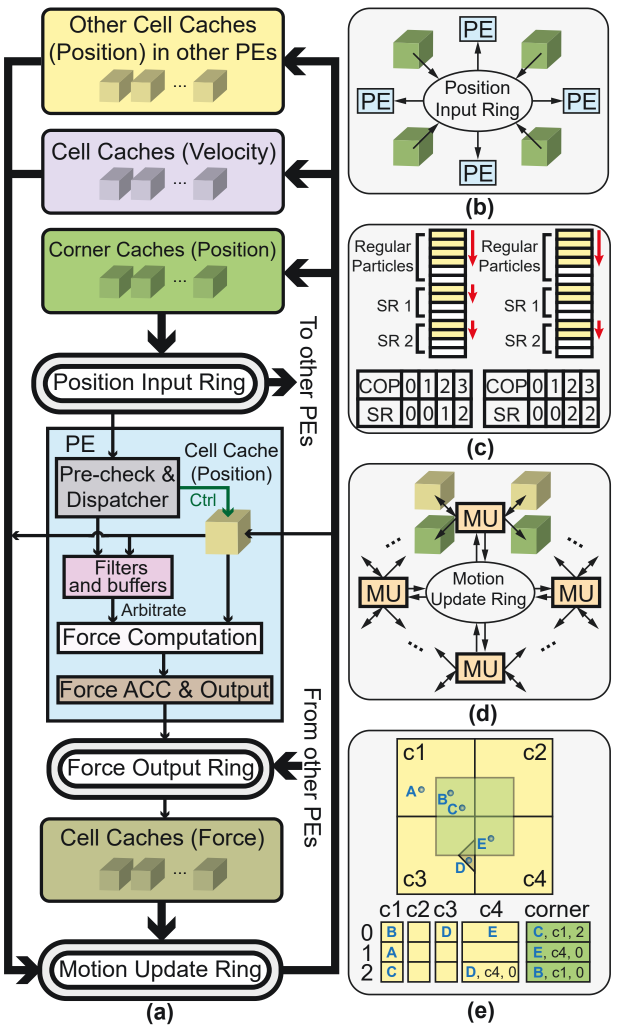


Fig. 5. The complete design with the Manhattan method. (a): the data flow with respect to a single PE. (b): the COPs enter the position input ring and are rotated to several destination PEs (c): two SR handling cases. Red arrow: data traversal order; left: all SRs are requested; right: SR 1 is not requested and is skipped. (d): the motion update ring. Each MU on the ring directly updates several CECs/COCs. (e): corner BRAMs store particle positions, cell IDs and the particle IDs for force write back and memory misalignment handling. The same mechanism is applied on SPs.

in the PE is traversed iteratively during force evaluation for pairing. With the Manhattan method involved, the filters also check the Manhattan distance between the particles and the corner block boundary. The good news is, not all the position bits are evaluated in the filters, but only, say, 8 leading bits are used. The number of bits is subject to the precision desired. Compared with the half-shell method, the filter pass rate increases from ∼17% to ∼30% (disregarding the overhead, the actual pass rate subjects to the number of filters and is further discussed in the evaluation section) for uniformly distributed particles, and the number of filters is therefore effectively reduced. The resources saved can then contribute to building more PEs for higher throughput.

The force fragments of COPs and the accumulated SP forces are then injected to the force output ring, while the forces of the regular CEPs are directly integrated in the force caches of the cell. Each force packet in the ring only has one destination, such that the force data is not duplicated, and each force cache only covers a cubic volume without SRs. The SPs are inherently far away from the COPs, and their pairing chances are slim. Among all the cell-corner pairs, only $\sim$1% is contributed by shadow-corner pairs. The extra pressure on the force output ring caused by SP forces is therefore negligible.

After all forces are evaluated and integrated in force caches, the MUs start. A MU on the motion update ring inputs all three types of data (position, velocity, force) and obtain the position and velocity of particles for the next time step. The packets are either consumed locally (used to update the directly connected caches) or injected to a motion update ring (Figure 5(d), velocity and force caches omitted) to update the corner and CECs (position and velocity).

The workload of the motion update phase is significantly smaller than force evaluation, such that a much smaller number of MUs are equipped compared to the number of cells. Furthermore, with a MU directly connected to multiple CECs/COCs, the latency of a motion update ring is short compared to other rings. Moreover, because the force evaluation is not active during motion update, the position input ring and the force output ring can be reused to construct the motion update ring with minor cost in hardware resources.

F. Memory Misalignment

During force evaluation, the particles are easily aligned in force CECs and position CECs (excluding SRs). That is, a particle has the same address in both caches. However, the alignment cannot be preserved for COPs or SPs. For example, particle B in Figure 5(e) is at address 0 in CEC C1, the force is also located at address 0 in its corresponding force cache; therefore B can be directly updated with the force at the same address. However, B is at address 2 in the COC instead of 0, i.e., memory misalignment. Similarly, the SP D is stored in the shadow addressing space in cell C4, with its original location at address 0 in C3. To correctly update the COPs and SPs, address and cell indexing information is added during motion update. In Figure 5(e), the COCs and the shadow addressing spaces in CECs not only contain the position values of particles, but also the cell IDs and addresses.

When there is no particle migration, a MU first updates a particle in a directly connected cell, then sends a packet to the motion update ring to find the destination COC. If a particle migrates to a cell handled by another MU, the packet is delivered to the MU and next sent to the destination corner cell with the updated address. Fortunately, the particle migration is rare and the latency introduced is negligible.

G. Multi-chip Solution

The advantage of the cache overlapping method is particularly remarkable when applied to multiple FPGA nodes. Figure 6 compares the half-shell method and the cache overlapping method on 4 FPGA nodes with periodic boundary condition. For demonstration, each node contains 4x4 2-D cells. If the data are structured in plain cells, as Figure 6(a) shows, 10 of 16 cells (blue) need to be transferred to other nodes. In the new mapping with the cache overlapping, only the equivalent of 7 cells are transferred as shown in Figure 6(b).

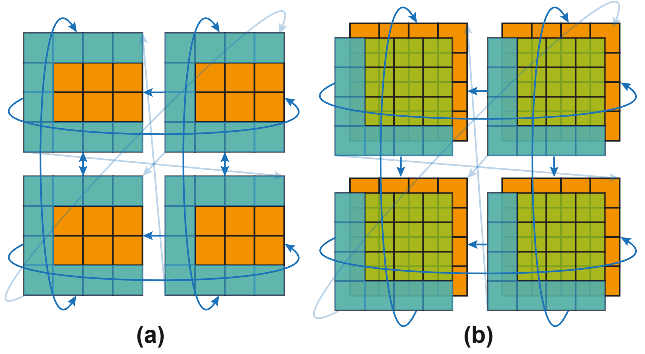


Fig. 6. 2-D illustration of the data transfer pattern for 4 FPGA nodes. (a): half-shell method; (b): the proposed position cache overlapping Manhattan method. Orange: CECs region; Blue: data to be transferred; Green: COCs region. The arrows indicate the transfer directions of the position data. For small data transfers at corners, the arrows are lightened.

The arrows only represent the position data transfer directions. The force data are returned to the source nodes along the arrows in reverse. This feature results in the natural balance between inbound force data and outbound position data with direct transceiver connections among FPGAs. However, there are two-way arrows in (a), potentially leading to heavy and imbalanced data transfer between the affected nodes compared to other transfer paths. Typically, each FPGA node provides $\sim$100 Gbps level bandwidth (2x100 Gbps QSFP28 for Intel D5005 and Xilinx Alveo U280), making the problem far more significant. The situation is much relieved in (b), where only one-way arrows are observed. This implies that data transfer is almost perfectly balanced along any direction (up, down, right, left, front in 3-D, back in 3-D) with negligible small data transfers at corners.

IV. EVALUATION

In this section, we evaluate and compare the efficiency of the half-shell and the cache overlapping methods in four aspects: How much more efficient can the filters get with the Manhattan method? How much latency can be reduced in the position input ring with reduced number of packet destinations? Can we hide the latency in the motion update ring with corner cells involved? How much data transfer can be avoided by using the cache overlapping method for different simulation space configurations and node topologies?

Although the design is relatively board-independent, the evaluations are based on the resources available on resource-abundant Intel D5005 boards with Stratix 10 SX FPGA chips, where each chip has 933120 ALMs, 5760 DSPs, and 11721 M20K BRAMs. The designs are implemented with Verilog and SystemVerilog HDL on Quartus 19.2 and validated on the D5005 boards. The resource/frequency results are obtained from reports generated by the Quartus software.

A. Performance

Figure 7 shows both the performance and PE utilization. For all four geometric cases, four Manhattan filters are almost

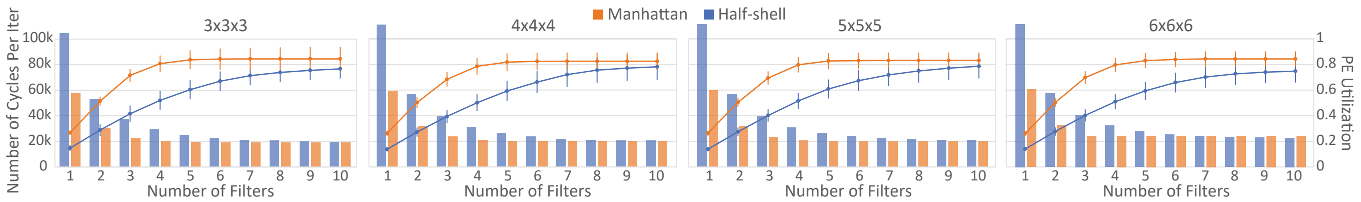


Fig. 7. The performance and PE utilization versus the number of filters for 4 cell geometries. 3 motion update rings and 3 force rings are used. For consistency, we assume each cell is processed by 1 PE, and all 4 cases share the same y-axis. Bar: cycle. Curve: utilization

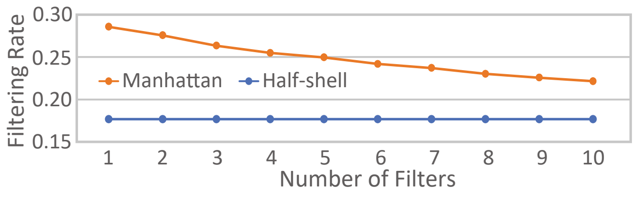


Fig. 8. Filtering rate and the number of cycles required for filtering for the two methods. For 3^3, 4^3, 5^3 and 6^3 cell geometries, the filtering rates remain the same. We assume each cell is only processed by 1 PE for consistency against the scaling of the simulation space.

as efficient as eight half-shell filters. The PE utilization is saturated at $\sim$ 85%, which can be regarded as the utilization limit of both designs. The Manhattan method reaches the limit at $\sim$5 filters, while the half-shell method requires $>$10 filters to achieve the same goal.

We also observe that for 6x6x6 cell geometry, the Manhattan method is bested by the half-shell method in performance. The reason is, the latency in the motion update rings is no longer negligible. During motion update, the Manhattan method updates more particle than half-shell due to the COCs and the SRs. Moreover, with the up-scaled number of ring nodes (27 MUs for 6x6x6 cells), the latency situation is worsened.

B. Filtering Rates

We observe from Figure 8 that the half-shell filtering rate is stable at $\sim$17%, yet the Manhattan filtering rate decreases, starting from $\sim$28%. This is because the SR handling mechanism in Figure 5(c) tends to include more filtering overhead as more COPs are processed simultaneously. As a result, more SRs are involved for pairing, but only for some of the COPs.

C. Position Input Ring Latency

Figure 9 shows the latency of the position input ring from the injection of the first particle to the departure of the last particle. The latency scales almost linearly with the number of cells per dimension for both methods and is robust against the increasing number of cells. The latency also scales linearly with the number of particles per cell, where a cell usually contains $\sim$100 particles in water environment.

The proportion of latency reduction is higher for small numbers of cells. Up to 13% of latency reduction is observed for 3x3x3 cells. The latency reduction is important because it allows more particles be allocated to PEs in a certain amount of time. 13% less latency means 15% more particles can be provided for PEs, potentially increasing the number of PEs working on a single cell, and eventually enhancing the performance for a limited number of cells.

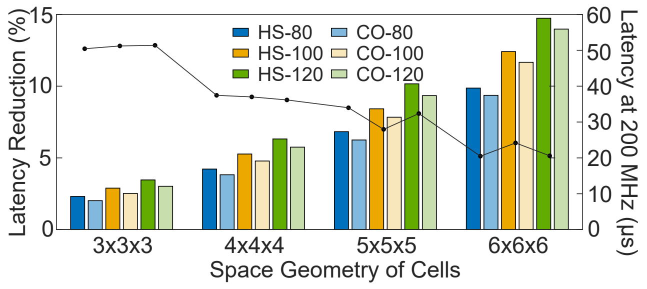


Fig. 9. The latency of the position input ring of a single MD iteration. HS: half-shell; CO: cache overlapping. 80, 100, 120: number of particles per cell. The line plot indicates the percentage of latency reduced using the cache overlapping method.

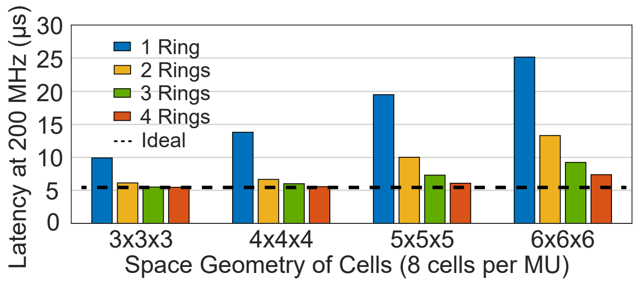


Fig. 10. MU ring latency compared with ideal. Each MU is in charge of updating 8 CECs and 8 COCs. A cell contains 80 particles.

D. Motion Update Ring Latency

As a trade-off, the latency of a motion update ring is increased. The comparison of the new and ideal latency is shown in Figure 10, where each cell contains 100 particles. For balanced resources, throughput, and wiring complexity, each MU is used to update 8 CECs and 8 COCs.

The ideal latency is obtained assuming all MUs only update their local cells without communication. The extra latency introduced is sensitive to the geometry. For smaller numbers of cells, only 2 or 3 motion update rings (2x or 3x ring concurrency) are enough to reduce the latency to a negligible degree. For 6x6x6 cells, 3μs extra latency is introduced even for 4 rings. Fortunately, the latency is small compared to the overall latency of an iteration ($\sim$ 100μs for 6x6x6 cells, 200MHz). Furthermore, because the position and force rings can be reused for the motion update ring, and the number of MUs is significantly smaller than the number of cells, only a small amount of hardware resources are required to construct the extra rings, especially for smaller numbers of cells.

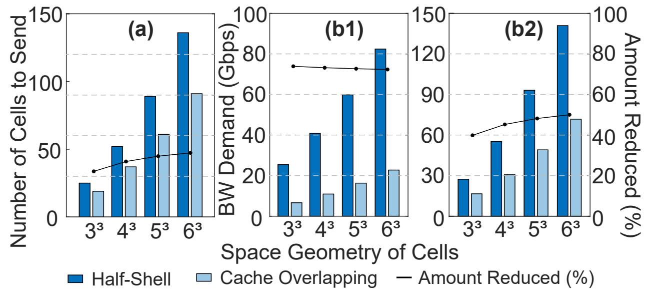


Fig. 11. Data transfer per FPGA using the two methods. Each iteration (from force evaluation to the next force evaluation phase) takes 100 μs, with 120 bits per packet and 80 particles per cell. (a): the number of cells to be sent to remote FPGAs. (b1): the estimated ideal bandwidth demand per FPGA for a 3-D torus FPGA cluster. (b2): the estimated ideal bandwidth demand per FPGA for 8 FPGAs connected as a ring.

E. Multi-FPGA Data Transfer

The number of cells to be sent to remote FPGAs is reduced by a considerable amount (∼30%), and scales almost linearly with the space edge length (Figure 11(a)). In fact, the actual amount of data transfer is reduced more significantly, as data forwarding is common in FPGA clusters (e.g., [32]–[36] and all-to-all connections [37] are not always available.

Figure 11(b) gives the data transfer behavior on two likely FPGA cluster configurations. We assume each MD RL iteration takes 100 μs, each packet is 120 bits and each cell contains 100 particles as the typical standard. The bandwidth numbers are ideal without taking imbalanced data transfer into account. In (b1), the cluster has a 3-D torus topology. For all listed space geometries, the data transfer reductions all approach 75%. In the 3-D torus, a packet needs to travel through at most 3 nodes to reach its destination. With the cache overlapping method applied, the proportion of data that require heavy forwarding is further reduced. In (b2), the cluster is 8 FPGAs in a ring, a likely scenario since many FPGA boards only have 2 or 4 transceiver ports. In this case, data may travel with more hops than a 3-D torus, such that the overall bandwidth requirement is raised. Still, 40%∼50% of data transfer can be saved.

The significance of these results is found especially in the most likely FPGA cluster use-cases, which involve strong scaling challenges (e.g., small molecule docking). With, say, 3x3x3 cells per node, FPGA capacity allows multiple PEs (8 in our case, 216 PEs in total) to work on the same cell, reducing the time per iteration to 1/8th the previous. As a result, 8x bandwidth is required, which is 200 Gbps. On the other hand, compared to the 6x6x6 case (with one PE per cell), only ∼80 Gbps are required. This is because the ratio of surface area to volume (SATV) ratio for the 6x6x6 cells is much lower. With strong-scaling (more PEs and/or fewer particles), the SATV ratio increases. At that point the computation is completely compute bound and the performance is proportional to the amount of data transferred.

F. Hardware Resource Usage

The hardware resources demanded for both designs are listed in Table I. Each half-shell PE has 6 filters, while

TABLE I
HARDWARE COSTS

Cell Space	Design	ALM	BRAM	DSP
3x3x3	M[1]	112924 (12.1%)	1296 (11.1%)	621 (10.8%)
	HS[2]	116146 (12.4%)	1215 (10.4%)	621 (10.8%)
4x4x4	M	236220 (25.3%)	3072 (26.2%)	1472 (25.6%)
	HS	243464 (26.1%)	2880 (24.6%)	1472 (25.6%)
5x5x5	M	429812 (46.1%)	6000 (51.2%)	2875 (49.9%)
	HS	443011 (47.5%)	5625 (48.0%)	2875 (49.9%)
6x6x6	M	713675 (76.5%)	10368 (88.5%)	4968 (86.3%)
	HS	735274 (78.8%)	9720 (82.9%)	4968 (86.3%)

[1] Manhattan [2] half-shell

each Manhattan PE has 4 filters. Both designs are equipped with 3 force output rings and 3 motion update rings. The Manhattan filters have higher logic expenses for their higher complexity compared to half-shell filters, but the overall ALM consumption is slightly reduced due to the reduction in the number of filters. The overall BRAM consumption is slightly higher because with ∼10 more bits included in position caches for cell indexing (see Figure 5(e)). Originally, the fixed-point position data and particle type (e.g., oxygen) together are ∼75 bits. With the 10 additional bits, we need 3 BRAMs side-by-side to satisfy the concurrent access of all 85 bits, where each BRAM is 40-bit wide, meaning another BRAM is required in each position cache. The new design requires no extra DSPs.

V. CONCLUSION

In this paper, we compare the baseline half-shell model with the improved cache overlapping model based on the Manhattan method and find almost no difference in resource usage. The findings are as follows.

First, the filters in the Manhattan design are much more efficient compared to the half-shell filters. The PE utilization approaches its limit with only 4 Manhattan filters, whereas 8 filters are needed for the same with half-shell. Although the filtering rate of the Manhattan filters decreases due to SP handling, it is still considerably higher than that of the half-shell filters. Second, the latency of the overall position data input is reduced, especially for smaller numbers of cells (3x3x3) where 15% more work can be distributed to PEs in the same amount of time. Third, as the trade-off, the motion update latency is increased, but can be negligible thanks to the reusable hardware. Fourth, ∼75% of the data transfer can be saved with the new method on a 3-D FPGA torus, and 40% to 50% of the transfer can be saved with 8 FPGAs connected as a ring. A major benefit is that the pressure in data transfer is greatly relieved for FPGA-based MD where commercially available boards have significantly less available bandwidth than custom ASIC-based systems.

ACKNOWLEDGMENT

This work was supported, in part, by the NSF through award CCF-1919130; the NIH through award R44GM128533; by a grant from Red Hat; and by AMD and by Intel both through donated FPGAs, tools, and IP.

References

[1] M. Snir, "A note on N-body computations with cutoffs," *Theory of Computing Systems*, vol. 37, pp. 295–318, 2004.

[2] 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.

[3] ——, "The midpoint method for parallelization of particle simulations," *The Journal of Chemical Physics*, vol. 124, no. 18, p. 184109, 2006.

[4] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, "Reconfigurable molecular dynamics simulator," in *IEEE Symposium on Field Programmable Custom Computing Machines*, 2004, pp. 197–206.

[5] Y. Gu, T. VanCourt, and M. Herbordt, "Accelerating molecular dynamics simulations with configurable circuits," in *IEEE Conference on Field Programmable Logic and Applications*, 2005.

[6] T. Hamada and N. Nakasato, "Massively parallel processors generator for reconfigurable system," *IEEE Symposium on Field Programmable Custom Computing Machines*, 2005.

[7] V. Kindratenko and D. Pointer, "A case study in porting a production scientific supercomputing application to a reconfigurable computer," in *IEEE Symposium on Field Programmable Custom Computing Machines*, 2006, pp. 13–22.

[8] 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.

[9] R. Scrofano, M. Gokhale, F. Trouw, and V. Prasanna, "Accelerating Molecular Dynamics Simulations with Reconfigurable Computers," *IEEE Trans. Parallel and Distributed Systems*, vol. 19, no. 6, pp. 764–778, 2008.

[10] M. Chiu and M. Herbordt, "Molecular dynamics simulations on high performance reconfigurable computing systems," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 3, no. 4, pp. 1–37, 2010.

[11] C. Yang, T. Geng, T. Wang, R. Patel, Q. Xiong, A. Sanaullah, C. Lin, V. Sachdeva, W. Sherman, and M. Herbordt, "Fully Integrated FPGA Molecular Dynamics Simulations," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[12] 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.

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

[14] J. Sheng, C. Yang, and M. Herbordt, "Towards Low-Latency Communication on FPGA Clusters with 3D FFT Case Study," in *International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, 2015.

[15] J. Sheng, C. Yang, A. Caulfield, M. Papamichael, and M. Herbordt, "HPC on FPGA Clouds: 3D FFTs and Implications for Molecular Dynamics," in *27th International Conference on Field Programmable Logic and Applications*, 2017.

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

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

[18] L. Stewart, C. Pascoe, E. Davis, B. Sherman, M. Herbordt, and V. Sachdeva, "Particle Mesh Ewald for Molecular Dynamics in OpenCL on an FPGA Cluster," in *IEEE Symposium on Field Programmable Custom Computing Machines*, 2021.

[19] C. Wu, S. Bandara, T. 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.

[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] 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.

[23] ——, "Explicit design of FPGA-based coprocessors for short-range force computation in molecular dynamics simulations," *Parallel Computing*, vol. 34, no. 4-5, pp. 261–271, 2008.

[24] A. Obeidat, A. Jaradat, B. Hamdan, and H. Abu-Ghazleh, "Effect of cutoff radius, long range interaction and temperature controller on thermodynamic properties of fluids: Methanol as an example," *Physica A: Statistical Mechanics and its Applications*, vol. 496, 2018.

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

[26] J. Grossman, B. Towles, B. Greskamp, and D. Shaw, "Filtering, reductions and synchronization in the Anton 2 network," in *Proc. International Parallel and Distributed Processing Symposium*, 2015, pp. 860 – 870.

[27] 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.

[28] M. Chiu and M. Herbordt, "Efficient filtering for molecular dynamics simulations," in *2009 International Conference on Field Programmable Logic and Applications*, 2009.

[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] M. Chiu, M. Khan, and M. Herbordt, "Efficient calculation of pairwise nonbonded forces," in *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*, 2011.

[31] 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 *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021.

[32] A. George, M. Herbordt, H. Lam, A. Lawande, J. Sheng, and C. Yang, "Novo-G#: A Community Resource for Exploring Large-Scale Reconfigurable Computing Through Direct and Programmable Interconnects," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, 2016.

[33] J. Sheng, Q. Xiong, C. Yang, and M. Herbordt, "Collective Communication on FPGA Clusters with Static Scheduling," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 4, 2017.

[34] T. Boku, R. Kobayashi, N. Fujita, H. Amano, K. Sano, T. Hanawa, and Y. Yamaguchi, "Cygnus: GPU meets FPGA for HPC," in *International Conference on Supercomputing*, 2019, https://www.r-ccs.riken.jp/labs/lpnctrt/assets/img/ lspanc2020jan_boku_light.pdf.

[35] A. Mondigo, T. Ueno, K. Sano, and H. Takizawa, "Comparison of Direct and Indirect Networks for High-Performance FPGA Clusters," in *ARC 2020. Lecture Notes in Computer Science, vol 12083*, F. Rincon, J. Barba, H. So, P. Diniz, and J. Caba, Eds. Springer, 2020, 10.1007/978-3-030-44534-8_24.

[36] H. Shahzad, A. Sanaullah, and M. Herbordt, "Survey and Future Trends for FPGA Cloud Architectures," in *IEEE High Performance Extreme Computing Conference*, 2021.

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