Fully Integrated FPGA Molecular Dynamics Simulations

Chen Yang
Boston University, Boston, MA
cyang90@bu.edu

Tong Geng
Boston University, Boston, MA
tgeng@bu.edu

Tianqi Wang
University of Science and Technology of China, Hefei, China
tqwang@mail.ustc.edu.cn

Rushi Patel
Boston University, Boston, MA
ruship@bu.edu

Qingqing Xiong
Boston University, Boston, MA
qx@bu.edu

Ahmed Sanaullah
Boston University, Boston, MA
sanaulah@bu.edu

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

Jiayi Sheng
Falcon Computing Solutions Inc., Los Angeles, CA
jysheng@falcon-computing.com

Charles Lin
Silicon Therapeutics, Boston, MA
charles.lin@silicontx.com

Vipin Sachdeva
Silicon Therapeutics, Boston, MA
vipin@silicontx.com

Woody Sherman
Silicon Therapeutics, Boston, MA
woody@silicontx.com

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

ABSTRACT

The implementation of Molecular Dynamics (MD) on FPGAs has received substantial attention. Previous work, however, has consisted of either proof-of-concept implementations of components, usually the range-limited force; full systems, but with much of the work shared by the host CPU; or prototype demonstrations, e.g., using OpenCL, that neither implement a whole system nor have competitive performance. In this paper, we present what we believe to be the first full-scale FPGA-based simulation engine, and show that its performance is competitive with a GPU (running Amber in an industrial production environment). The system features on-chip particle data storage and management, short- and long-range force evaluation, as well as bonded forces, motion update, and particle migration. Other contributions of this work include exploring numerous architectural trade-offs and analysis of various mappings schemes among particles/cells and the various on-chip compute units. The potential impact is that this system promises to be the basis for long timescale Molecular Dynamics with a commodity cluster.

CCS CONCEPTS

• **Hardware** → **Hardware accelerators**; **Reconfigurable logic applications**.

KEYWORDS

Molecular Dynamics, FPGA, High-Performance Computing.

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356179

ACM Reference Format:
Chen Yang, Tong Geng, Tianqi Wang, Rushi Patel, Qingqing Xiong, Ahmed Sanaullah, Chunshu Wu, Jiayi Sheng, Charles Lin, Vipin Sachdeva, Woody Sherman, and Martin Herbordt. 2019. Fully Integrated FPGA Molecular Dynamics Simulations. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3295500.3356179

1 INTRODUCTION

There are dozens of MD packages in production use (e.g., [1–5]), many of which have been successfully accelerated with GPUs. Scaling, however, remains problematic for the small simulations (20K-50K particles) commonly used in critical applications, e.g., drug design [6, 7], where long timescales are also extremely beneficial. Simulation of long timescales of small molecules is, of course, a motivation for the Anton family of ASIC-based MD engines [8, 9]. Anton addresses scalability by having direct communication links–application layer to application layer–among the integrated circuits (ICs) in the cluster. But while ASIC-based solutions can have orders-of-magnitude better performance than commodity clusters, they may also have issues with general availability, plus problems inherent with small-run ASIC-based systems.

FPGAs have been explored as possible MD accelerators for many years [10–16]. The first generation of complete FPGA/MD systems accelerated only the range limited (RL) force and used CPUs for the rest of the computation. While performance was sometimes competitive, high cost and lack of availability of FPGA systems meant that they were never in production use. In the last few years, however, it has been shown that FPGA clusters can have performance approaching that of ASIC clusters for the Long Range force computation (LR) [17–20], the part of MD that is most difficult to scale.

It remains to be demonstrated, however, whether a single FPGA MD engine can be sufficiently competitive to make it worth developing such a cluster. And if so, how should it be implemented? One thing that is certain is that previous CPU-centric approaches are

not viable: long timescales require ultra-short iteration times which make the cost of CPU-device data transfers prohibitive. This leads to another question: is it possible to build such an FPGA MD engine where there is little interaction with other devices?

One advantage with current FPGAs is that it is now possible–for simulations of great interest (up to roughly 40K particles)–for all data to reside entirely on-chip for the entire computation. Although this does not necessarily impact performance (double-buffering off-chip transfers still works), it simplifies the implementation and illuminates a fundamental research question: what is the best mapping among particles, cells, and force computation pipelines? Whereas the previous generation of FPGA/MD systems only dealt with a few cells and pipelines at a time, the concern now is with hundreds of each. Not only does this lead to a new version of the problem of computing pairwise forces with cutoff (see [21, 22]), it also requires orchestrating RL with the other force computations, and then all of those with motion update and particle movement.

The major contribution is an end-to-end MD system implemented on a widely used FPGA board.We have validated simulation quality using Amber 18. In preliminary experiments with the Dihydrofolate Reductase (DFHR) dataset (23.5K particles), the system achieves a throughput of 630ns/day. Other contributions are as follows.

- The first implementation of full MD (RL, LR, and Bonded force with Motion Integration) on a single FPGA, that completely removes the dependency on off-chip devices, thus eliminating the communication overhead of data transfer;
- The first analysis of mappings among particles/cells, on-chip memories (BRAMs), on-chip compute units (pipelines) of LR, RL, and bonded forces;
- Various microarchitecture contributions related to every aspect of the system, including exploration of RL particle-pair filtering, two sets of memory architectures (distributed for RL and LR, and global for Bonded), a scoreboarding mechanism that enables motion update in parallel with the force evaluation, and integrating motion update;
- Application-aware optimizations through HDL generator scripts.

The potential impact is that this system promises to be the basis for scalable long timescale Molecular Dynamics with a commodity cluster. In the Discussion section we present preliminary scalability results based on a model derived from previous inter-FPGA communication studies.

2 MD BACKGROUND

Basics. MD alternates between force calculation and motion update. The forces computed depend on the system being simulated and may include bonded terms, pairwise bond, angle, and dihedral; and non-bonded terms, van der Waals and Coulomb [23]:

$$\mathbf{F^{total}} = F^{bond} + F^{angle} + F^{dihedral} + F^{non-bonded} \quad (1)$$

The *Bonded Force* terms involve small numbers of particles per bond, but the computations themselves can be complex; interactions can be expressed as follows: bond (Equation (2)), angle (Equations (3) and (4)), and dihedral(Equations (5) and (6)), respectively (from Equation (1) [24]).

$$\mathbf{F_i^{bond}} = -2k(r_{ij} - r_0)\vec{e_{ij}} \quad (2)$$

$\vec{e_{ij}}$ is the unit vector from one item to another, r_{ij} the distance between the two particles, k the spring constant, and r_0 the equilibrium distance;

$$\mathbf{F_i^{angle}} = -\frac{2k_\theta(\theta - \theta_0)}{r_{ij}} \cdot \frac{\vec{e_{ij}}cos(\theta) - \vec{e_{kj}}}{sin(\theta)} + f_{ub} \quad (3)$$

$$\mathbf{f_{ub}} = -2k_{ub}(r_{ik} - r_{ub})\vec{e_{ik}} \quad (4)$$

$\vec{e_{ij}}, \vec{e_{kj}}, \vec{e_{ik}}$ are the unit vectors from one item to another, θ the angle between vectors $\vec{e_{ij}}$ and $\vec{e_{kj}}$, θ_0 the equilibrium angle, k_θ the angle constant, k_{ub} the UreyBradley constant, and r_{ub} the equilibrium distance;

$$\mathbf{F_i^{dihedral}} = -\nabla\frac{U_d}{\vec{r}} \quad (5)$$

$$\mathbf{U_d} = \begin{cases} k(1 + cos(n\psi + \phi)) & n > 0, \\ k(\psi - \phi)^2 & n = 0. \end{cases} \quad (6)$$

n is the periodicity, ψ the angle between the (i, j, k)-plane and the (j, k, l)-plane, ϕ the phase shift angle, and k the force constant.

The *Non-bonded Force* uses 98% of FLOPS and includes Lennard-Jones (LJ) and Coulombic terms. For particle i, these can be:

$$\mathbf{F_i^{LJ}} = \sum_{j \neq i} \frac{\epsilon_{ab}}{\sigma_{ab}^2} \left\{ 48\left(\frac{\sigma_{ab}}{|r_{ji}|}\right)^{14} - 24\left(\frac{\sigma_{ab}}{|r_{ji}|}\right)^{8} \right\} \mathbf{\vec{r}_{ji}} \quad (7)$$

$$\mathbf{F_i^C} = \frac{q_i}{4\pi} \sum_{j \neq i} \frac{1}{\epsilon_{ab}} \left\{ \frac{1}{|r_{ji}|} \right\}^3 \mathbf{\vec{r}_{ji}} \quad (8)$$

where the ϵ_{ab} (unit: kJ or $kcal$) and σ_{ab} (unit: meter) are parameters related to the types of particles.

The LJ term decays quickly with distance, thus a *cutoff radius,* r_c, is applied: the LJ force is zero beyond it. The Coulombic term does not decaying as fast; but this term can be divided into two parts, fast decaying within r_c and slowly developing beyond it. Consequently, we approximate the LJ force and the fast decaying part of the Coulombic force as the *Range-Limited (RL) force*, and the other part of the Coulombic force force as the *Long-Range (LR) force*. RL is the more computationally intensive (90% of flops) and is calculated as:

$$\frac{\mathbf{F_{ji}^{RL}}}{\mathbf{r_{ji}}} = A_{ab}r_{ji}^{-14} + B_{ab}r_{ji}^{-8} + QQ_{ab}r_{ji}^{-3} \quad (9)$$

where $A_{ab} = 48\epsilon_{ab}\sigma_{ab}^{12}$, $B_{ab} = -24\epsilon_{ab}\sigma_{ab}^{6}$, $QQ_{ab} = \frac{q_a q_b}{4\pi\epsilon_{ab}}$.

The LR force is calculated by solving the Poisson Equation for the given charge distribution.

$$\mathbf{F_i^{LR}} = \sum_{j \neq i} \frac{q_j}{|r_{ji}|}\mathbf{\vec{r}_{ji}} \quad (10)$$

$$\rho_g = \sum_p Q_p\phi(|x_g - x_p|)\phi(|y_g - y_p|)\phi(|z_g - z_p|) \quad (11)$$

LR is often calculated with a grid-based map of the smoothing function converted from continuous space to a discrete grid coordinate system [25]. Each particle is interpolated to grid points by applying a third-order basis function for charge density calculation. Grid points obtain their charge densities from neighboring particles within a range of two grid points in each direction. There, grid electrostatics are converted into the Fourier domain, evaluated using the Green's function, then converting back through an inverse FFT.

Force Evaluation Optimizations. RL uses the cutoff to reduce the $O(N^2)$ complexity: forces on each *reference particle* are computed only for *neighbor particles* within r_c. The first approximation is the widely used partitioning of the simulation space into equal sized

cells with a size related to r_c. The particles can be indexed using *cell-lists* [26]: for any reference particle and a cell length of r_c, only neighbor particles in the 26 *neighboring cells* need to be evaluated. Another optimization is Newton's 3rd Law (N3L): since the force only needs to be computed once per pair, only a fraction of the neighboring cells need to be referenced. Most of the particles, however, are still outside the cutoff radius. In CPU implementations this can be handled by periodically creating neighbor lists. In FPGAs, the preferred method is to do this on-the-fly [15] through *filtering*.

Boundary Conditions. We assume *Periodic Boundary Conditions (PBC)*: When evaluating particles in boundary cells, we imagine a fictional space that is an exact copy of the simulated space.

Motion Integration. Changes of position and velocity of each particle can be computed using the Verlet algorithm. Since we are using a short timestep ($2fs$), we can use simple integration equations such as symplectic Euler:

$$\vec{a}(t) = \frac{\vec{F}(t)}{m} \tag{12}$$

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \vec{a}(t) \times \Delta t \tag{13}$$

$$\vec{r}(t + \Delta t) = \vec{r}(t) + \vec{v}(t + \Delta t) \times \Delta t \tag{14}$$

where m is mass, $\vec{a}$ is acceleration, $\vec{v}$ is velocity, $\vec{r}$ is position.

3 FPGA-MD SYSTEM ARCHITECTURE

In this section, we cover the four major components inside an MD simulation system, along with some high-level design decisions. We begin with a classic FPGA-based MD force evaluation pipeline and then add several function units that, in previous implementations, were executed on the host processor or embedded cores.

3.1 Overall Architecture

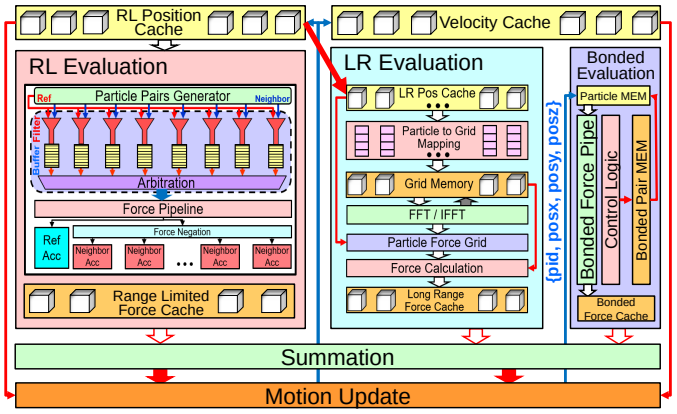

Figure 1: MD End-to-End System Overview. Details of each section is covered in following figures.

Since configuration time is long with respect to iteration time, the design is fixed within a single simulation. A design goal is to give the force computations resources such that their compute times are equalized; resource allocation to summation and motion update is analogous. All components (LR, RL, etc.) have parameterized designs with performance proportional to the parallelism applied (and thus chip resources used). This applies also to fractional parallelism: some components can be *folded*, e.g., to obtain half performance with half resources.

Figure 1 depicts the proposed FPGA-MD system. The **RL** units evaluate the pair-wise interactions. Since this is the most computationally intensive part, the base module is replicated multiple times. The **LR** unit includes: (i) mapping particles to a charge grid, (ii) conversion of charge grid to potential grid via 3D FFT (and inverse-FFT), and (iii) evaluating forces on individual particles based on the potential grid. Since our timestep is small ($2fs$), LR is only updated every few iterations. The **Bonded Evaluation** unit has pipelines for the three parts (see Eq. 1). At the end of each timestep, the **Summation** unit sums the three partial forces and sends the result to the **Motion Update** unit to update position and handle particle migration among adjacent cells.

3.2 RL Architecture

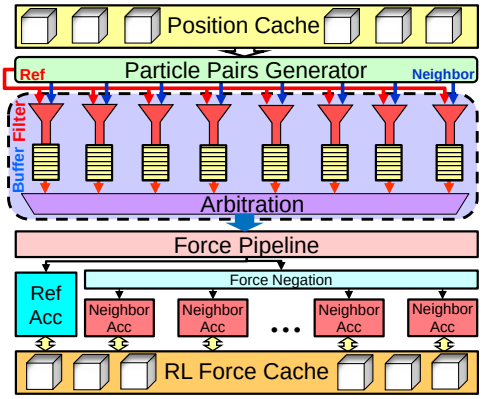

Figure 2: RL Evaluation Architecture Overview

The RL force evaluation pipeline (Figure 2) is based on a design first proposed in [27], although nearly all parts have been redesigned from scratch; see [28] for more details. The particle position cache holds the initial position of each particle. Modern high-end FPGAs like Intel Stratix 10 [29] provide enough on-chip storage to hold particle data for our range of simulations. Next is a set of filters that performs a distance evaluation of possible particle pairs (in certain neighboring cells) and only pass the pairs within the cutoff radius. Remaining data then enter the most computationally intensive part in the process: force evaluation. Since each particle is contributing to multiple pair-wise interactions, we design an efficient accumulation mechanism on the output of the force evaluation pipeline to sum up the partial forces on each particle.

3.2.1 Particle-Pair Filtering. Mapping among cells, BRAMs, and filters is complex and is described below. Once a particle pair is generated and sent to a filter, its distance is compared with r_c (actually r^2 with r_c^2 to avoid the square root). Possible neighbor particles can reside in 27 cells in 3-dimensions (13+1 if considering N3L, as shown in Figure 3). Since the average pass rate is only 15.5%, providing a force pipeline with at least one valid output per cycle requires a bank of at least seven filters plus load balancing (we use eight). If there are multiple valid outputs, round-robin arbitration is used. The not-selected valid outputs are stored in the filter buffer (on the output side of each filter) as shown in Figure 2.

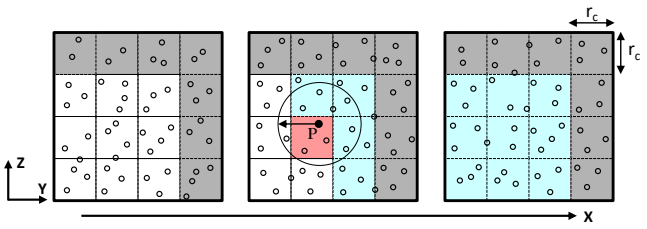

Figure 3: Simulation space about particle P. Its *cell neighborhood* is shown in non-gray; cell edge size is the cutoff radius (circle). After application of N3L, we only need to consider half of the neighborcells (blue) plus the homecell (red).

3.2.2 Force Evaluation. Various trade-offs have been explored in other FPGA/MD work [30, 31]. These are two of the most important. **Precision and Datatype:** CPU and GPU systems often use a combination of single-precision, double precision, integer, fixed-point, and floating-point. ASIC-based systems have complete flexibility and use non-standard types and precisions. FPGAs have multiple implementation possibilities. If logic cells alone are used, then ASIC-type designs would be preferred for fixed [15, 32, 33] or floating-point [34, 35]. Modern FPGAs, however, also have many thousands of embedded ASIC blocks, viz. DSP and/or floating-point units. So while the arithmetic design space is still substantial, preferred designs are likely to be quantized by these fixed-sized *hard* blocks. We find that, in contrast with earlier FPGA-MD studies, there is less advantage to use integer and fixed point; rather we primarily use the *native floating-point IP core*. For certain computations where accuracy is critical, we also employ fixed-point arithmetic; this is at the cost of high resource overhead (see Section 5.3.3).

Direct Computation vs. Interpolation with Table-lookup: The RL force calculation requires computing r^{-3}, r^{-8} and r^{-14} terms. Since r^2 is already provided from the filter unit, a total of 8 DSP units (pipelined) are needed to get these 3 values (based on the force pipeline proposed in [15]). Plus, we need 3 extra DSP units to multiply the 3 indexes, QQ_{ab}, A_{ab} and B_{ab}, with r^{-3}, r^{-14} and r^{-8} respectively. In order to reduce DSP usage, we use interpolation with table-lookup. As is common with this method, we divide the curve into several sections along the X-axis, such that the length of each section is twice that of the previous. Each section has the same number of intervals with equal size. We implement 3 sets of tables for r^{-3}, r^{-8} and r^{-14} curve. We use r^2, instead of r, as the index to further reduce resource consumption that would be needed when evaluating square root and division.

3.2.3 RL Workload Distribution. FPGAs provide abundant design flexibility that enables various workload to bare metal mapping schemes. In this subsection, we introduce two levels of mapping: particles onto Block RAMs (BRAMs), and workload onto pipelines. **Cell mapping onto BRAMs**: Figure 4 lists two of many possible mapping schemes, which we refer to as Mem 1 and Mem 2.

Mem 1: A single global memory module holds position data for all particles (Figure 4a). This design simplifies the wiring between position memory and the hundreds of pipelines. To overcome the bandwidth bottleneck, we insert an input cache at the start of each pipeline to hold the pre-fetched position data.

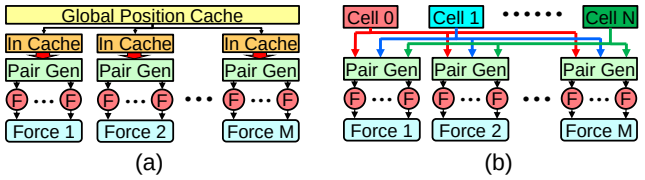

Figure 4: Cell to RAM Mapping Schemes: (a) all cells mapped onto a single memory module; (b) each cell occupies an individual memory module.

Mem 2: The bandwidth problem can also be overcome by having each cell map onto an individual memory unit (Figure 4b). But when there are hundreds of pipelines and cells, the all-to-all connect incurs large resource consumption and timing challenges.

Workload mapping onto pipelines: The simulation space is partitioned into cells. We successively treat each particle in the homecell as a *reference* particle and evaluate the distance with *neighbor* particles from its homecell and 13 neighborcells (N3L). The system then moves to the next cell and so on until the simulation space has been traversed. There are a vast number of potential mapping schemes; due to limited space, we present just three of the most promising.

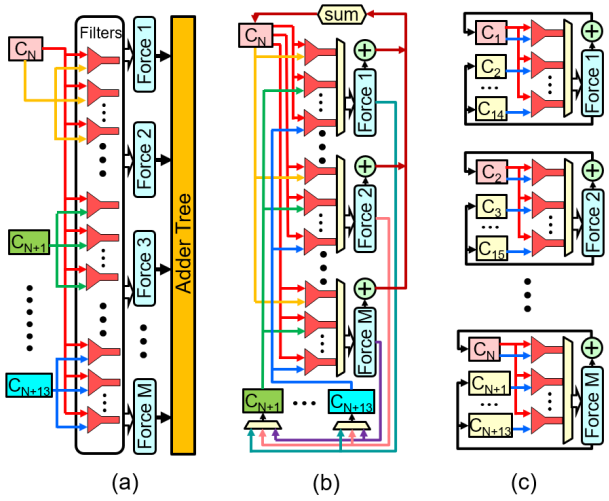

Figure 5: Workload mapping onto force pipelines: (a) all pipelines work on the same reference particle; (b) all pipelines work on the same homecell, but with different reference particles; (c) each pipeline works on a different homecell.

Distribution 1: **All pipelines work on the same reference particle (Figure 5a).** A global controller fetches a particle from the current homecell and broadcasts it to all the filters in the system, around 1000. Potential neighbor particles from home and neighbor cells are evenly distributed among all the filters. The evaluated partial force output from each pipeline is collected by an adder tree for summation and written back. At the same time, the partial forces are also sent back to the neighborcells and accumulated inside each cell. This implementation achieves the workload balance on the particle-pair level. However, it requires extremely high read bandwidth from

the position cache to satisfy the need for input data for each filter, and requires high write bandwidth when accumulating partial forces to neighbor particles, since the R&W only targets 14 cells at a time.

Distribution 2: **All pipelines work on the same homecell, but on different reference particles (Figure 5b).** To start, the particle pair generator reads out a reference particle from the homecell for filters belonging to each force pipeline. During the evaluation, the same neighbor particles are broadcast to all filters (belonging to different force pipelines) at the same time, since the neighbor particle set for every reference particle is the same as long as they belong to the same homecell. Compared with the first implementation, this one alleviates the pressure on the read port of the position cache. The tradeoff is that partial forces targeting the same neighbor particle may arrive at the neighborcell at the same time; thus a special unit is needed to handle the read-after-write data dependency. Since each force pipeline is working on different reference particles, an accumulator is needed for each force pipeline.

Distribution 3: **Each pipeline works on its own homecell (Figure 5c).** Under this mapping scheme, each filter only needs to interact with a subset of spatially adjacent homecells, along with a set of neighborcells. Compared with the previous two schemes, there is only interaction among a small set of cells. This method not only fully utilizes the parallelism in force evaluation, but also reduces the number of wires between particle caches and force evaluation units. The downside, however, is load balancing. Suppose we have 100 pipelines, but 150 cells. After each pipeline evaluates a cell, half of the pipelines will remain idle while the others evaluate a second homecell. To avoid this waste of resources, an application-aware mapping scheme is required.

3.3 LR Architecture

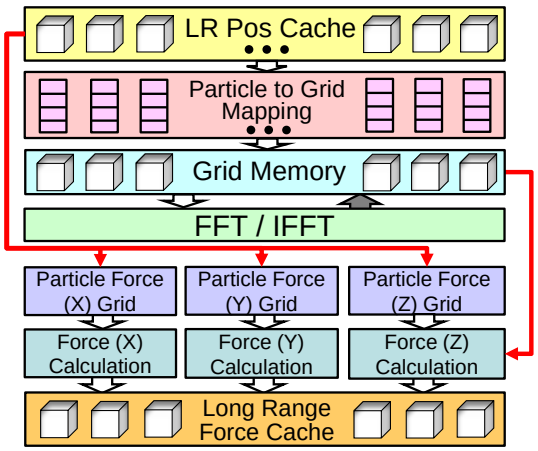

Figure 6: LR Evaluation Architecture Overview

Parts of the LR computation have been explored previously (see [36] on mapping and [37–39] on the 3D FFT), but this is the first time they have been integrated. LR (Figure 6) begins with a cache of position data, which maintains particle information when mapping to the particle grid and the force calculation. The position cache is necessary since positions may change during LR. Particle charges are evaluated and assigned to 64 neighboring cell locations using a third order basis function, with results stored in *grid memory*. After all particle data are consumed, the FFT runs on the resulting grid (through each axis X, Y, and Z). The resulting data, after multiplying with the Green's function, is replaced in the memory grid only a few cycles after evaluation. This is possible because of the pipeline implementation of the FFT. The inverse FFT is then performed on each dimension. Finally, forces are calculated for each individual particle using the final FFT grid results and the starting particle position information saved previously in the position cache. These are then saved into a force cache which is used during the motion update phase to apply long-range forces to the particle positions.

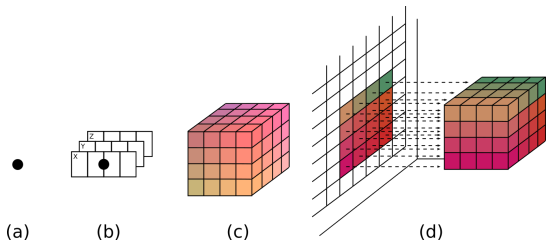

Figure 7: Particle to grid flow: (a) Initial particle position data; (b) Particle to 1D interpolation for each dimension using basis functions; (c) Mapping 1D interpolation results to a 4x4x4 3D grid; (d) Final 64 grid points to 16 independent memory banks

3.3.1 Particle to Grid Mapping. The third order basis functions Equation (15) are used to spread particle charges to the four closest grid points, based on particle position data, and can be independently evaluated for each dimension. After a particle is evaluated in each dimension, values are assigned to 64 neighboring cells and each result is accumulated into grid memory locations. Figure 7 shows the process of a single particle's influence on 64 neighborcells and their mapping to the grid memory structure. Parallel particle-to-grid mapping occurs with the use of accumulators before entering grid memory due to restrictions in using BRAMs.

$$\begin{cases} \phi_0(oi) = -1/2oi^3 + oi^2 - 1/2oi \\ \phi_1(oi) = 3/2oi^3 + 5/2oi^2 + 1 \\ \phi_2(oi) = -3/2oi^3 + 2oi^2 + 1/2oi \\ \phi_3(oi) = 1/2oi^3 - oi^2. \end{cases} \tag{15}$$

3.3.2 Grid Memory. We store grid points in BRAMs using an interleaved memory structure.This allows for stall-free access of grid locations while performing FFT calculations.

3.3.3 FFT. The FFT subsystem performs calculations in parallel using vendor supplied FFT core configured by Intel Quartus Prime Design Suite FFT IP controller. It has the capability of dictating the number of streaming values, by which we can change the core to suit the size of our design space (16, 32, ect.) [40]. To ensure high throughput memory access, we assign the FFT units to specific banks of the grid memory. As a result, grid data can be continuously streamed through all FFT cores in parallel. While output is being generated for a given vector, a new input is sent for the next set of calculations. Each dimension is performed sequentially until all

three dimensions are completed on the memory grid. Once all three dimensions are evaluated and converted into the Fourier-domain, the grid is multiplied with Green's function, before proceeding to the inverse FFT stage going through each dimension again and converting back. Final values at each grid point are used to compute the LR force for each particle based on its position.

3.4 Bonded Architecture

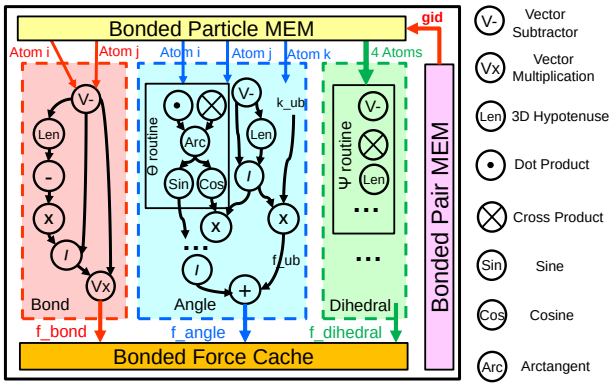

Figure 8: Bonded Force Evaluation Architecture

While the bonded force has been explored previously [41], this design and implementation is entirely new.

3.4.1 Sequential Evaluation of Bonded Interactions. As shown in Figure 8, we evaluate three types of bonded interactions: bond, angle, and dihedral, which have, respectively contributions from 2, 3, and 4 atoms. For a given dataset, the covalent bonds remain fixed as long as no chemical reaction is involved. In general, the bonded computation requires only a few percent of the FLOPs, so attenuation rather than parallelism is advantageous: we therefore process bonds sequentially.

3.4.2 Bonded Force Memory Architecture. For LR and RL we organize the particle data based on cells; this proves costly for bonded force evaluation. Rather than particles interacting with others based on their spatial locality, bonded interactions have a fixed set of contributing particles. As simulation progresses, particles can move across different cells and require extra logic to keep track of their latest memory address. Given the fact that we process bondeds sequentially, and this requires little memory bandwidth, we propose a different memory architecture: a single global memory module (*Bonded Particle MEM* in Figure 8) that maintains information on each particle position based on a fixed particle global id (gid). The gid is assigned prior to the simulation and remains fixed.

A read-only memory, *Bonded Pair MEM*, holds pairs of gids that form chemical bonds in the dataset. During force evaluation, the controller first fetches a pair of gids along with other parameters from Pair MEM, then proceeds to fetch the related particle position from Particle MEM and sends this for force evaluation. The evaluated bonded force is accumulated in the *Bonded Force Cache* addressed by *gid*. During motion update, the accumulated bonded force summed with partial results from RL and LR. Finally, Particle MEM receives the updated position, along with the particle gid, to maintain an up-to-date value.

3.5 Force Summation and Motion Integration

The three force components must be combined before the motion update. Even with load balancing, RL always finishes last; this is guaranteed, in part, by the small variance in the other computations. Therefore we can assume that the LR and bonded force caches always have data ready. Thus, as soon as RL of a certain particle is ready, we can perform the summation and motion update. As described in Section 3.2.1, for any given particle, it needs to be evaluated with respect to each neighbor particle from 27 cells. Since we make use of N3L to avoid revisiting particle pairs more than once, we need to keep track of how many times each cell has been visited (as homecell and neighborcells). To handle this we propose a *Score Boarding* mechanism. Once computations on all particles in a cell have finished, the Score Board module will access LR, RL and Bounded forces from the corresponding caches for force summation. By doing so, the positions of particles from the same cell can be updated immediately when a cell is fully evaluated; the motion update is executed in parallel with force evaluation with limited resource overhead; a large fraction of motion update latency can therefore be hidden.

After summation for a particle is finished, the aggregated force is sent to the motion update unit, along with particle's position and velocity. Since we organize particle data based on the cells they belong to (except for the bonded unit), particles can move from one cell to another. This creates a challenge on particle memory management: we need to maintain a record of which memory is ready for receiving new particles (due to particles left in the current cell, or the pre-allocated vacant memory space in each cell). It may take multiple cycles to find an available memory slot when the cell memory is almost full, or to find a valid particle in the cell when the cell memory is almost empty. Our solution is to double buffer the particle position and velocity caches; details are given in Section 4.1.1.

4 MD SYSTEM IMPLEMENTATION

In this section, we highlight a selection of implementation details.

4.1 Datatype and Particle Cache

The system maintains three sets of information for each particle: position, velocity, and force. The first two need to be maintained throughout the entire simulation, while the force data is flushed after motion update.

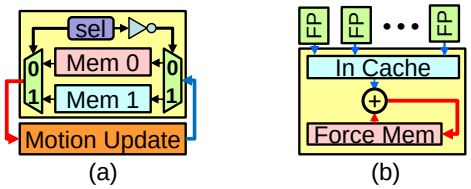

Figure 9: (a) Double buffer mechanism inside position and velocity cache; (b) Force cache with accumulator.

4.1.1 RL Particle Cache. **RL Position Cache** organizes data into cells. Double buffering is implemented (Figure 9a) with the particle *gid*s being kept along with position data, which is used during summation and motion update process. **RL Force Cache** is read and written during force evaluation. Since the system has hundreds of pipelines, that many partial forces must be accumulated each cycle. To manage the potential data hazards, we implement an accumulator inside each force cache module (see Figure 9b). After the aggregated forces are read out during motion update, they are cleared for the next iteration.

4.1.2 LR Particle Cache. The LR force evaluation is generally performed every two to four iterations while motion update happens every iteration. Since LR evaluation needs the positions to remain fixed, we allocate a separate LR particle cache (Figure 1&6). Every time LR starts a new evaluation (every second iteration in our experiments), it first performs a memory copy from RL Position Cache. To shorten the memory copy latency, we implement LR cache using *Mem 2*, which provides high write bandwidth.

4.2 RL Force Evaluation

We use the Native Floating Point IP Core controller inside Intel Quartus Prime Pro 18.1 Design Suite to configure the DSP units on FPGA to realize the IEEE floating point operations in our design.

4.2.1 Filter Logic. We propose two methods.

1. Filter v1: Direct computation uses 8 DSP units to calculate r^2 in floating-point and compare with r_c^2. If the distance is within cutoff, the evaluated r^2 is reused by the force pipeline. Since there are 8 filters per pipeline, the direct implementation consumes 48 DSP units, which limits the number of pipelines per chip.

2. Filter v2: Planar method uses Equations (16) to (18); note that the r_c terms are constants and not computed.

$$|x| < r_c, |y| < r_c, |z| < r_c \tag{16}$$

$$|x| + |y| < \sqrt{2}r_c, |x| + |z| < \sqrt{2}r_c, |y| + |z| < \sqrt{2}r_c \tag{17}$$

$$|x| + |y| + |z| < \sqrt{3}r_c \tag{18}$$

To avoid using DSPs altogether, input data is converted from floating-point to 28-bit fixed-point.

4.2.2 Filter Arbitration. Round-robin is used to select among filters with a valid output. To reduce the latency in the filter bank, which also saves buffer space, we have developed an arbitration algorithm that delivers one result per cycle.

4.2.3 RL Force Pipeline. Depending on the filter implementation, the force pipeline will receive one of two different inputs. Filter v1 provides r^2, while Filter v2 provides only the raw particle position so r^2 must be computed (Figure 10).

The pipeline evaluates forces via interpolation with table-lookup. Assuming the interpolation is second order, it has the format:

$$r^k = (C_2(x-a) + C_1)(x-a) + C_0 \tag{19}$$

where $x = r^2$, a is the x value at the beginning of the target interval, and $x - a$ is the offset into the interval. Based on different datasets, the interpolation coefficients are pre-calculated, packed into the mif file, and loaded onto the FPGA along with position and velocity data. After the coefficients are read from memory, the pipeline performs

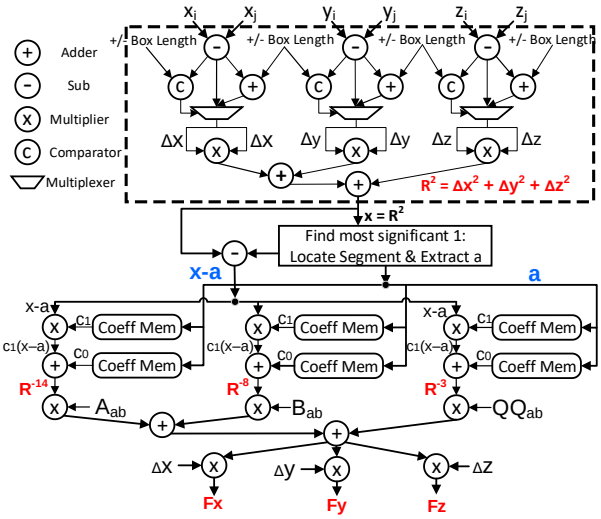

Figure 10: Force evaluation with first order interpolation.

the evaluation following Equation (19). Figure 10 shows this for the first order; the actual system supports upto the third order.

4.2.4 Partial Force Accumulator. The system traverses particles in the simulation space following cell order. When treating a particle as reference particle, the filter will check the distance with all the potential neighbor particles residing in the home and neighbor cells. We use N3L to avoid evaluating the same particle pair twice. But this also means the evaluated force need to accumulate to 2 particles. A difficulty is that the floating-point adder on FPGA has a latency of 3 cycles, leading to the classic accumulator problem: we can only perform one accumulation every 3 cycles. This is clearly unacceptable. We have two solutions, one for reference particles and one for neighbor particles.

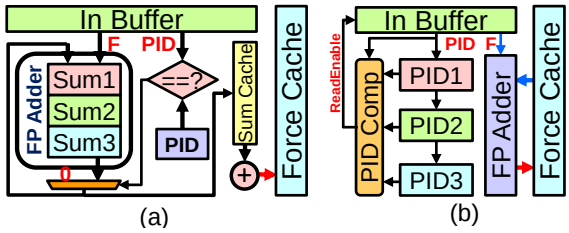

Figure 11: (a) Accumulator for reference particles, located at the output of the force pipeline; (b) Accumulator for neighbor particles, located at the input of the force cache.

Reference Particle Accumulator: The force pipeline keeps working on the same reference particle until all the assigned neighbor particles are traversed once. When a valid force arrives, the reference particle id (PID) is checked, if the particle has the same id as the accumulator is working on, then it is forwarded to the adder. If not, then it is recognized as the new reference particle, and the PID register will record the new particle ID. The returned value to the input sets to 0 for 3 cycles to reset the sum value. In the meantime,

the output writes to a small cache for 3 cycles, where they are added together and written back to force cache (see Figure 11a). The design is replicated 3 times to simultaneously handle F_x, F_y, F_z.

Neighbor Particle Accumulator: Multiple pipelines' outputs may target the same reference particle at the same time. But it is unrealistic to handle those conflicts in each pipeline. Our solution is to put the neighbor accumulator inside force cache; the pipelines forward their results to the force cache that holds the neighbor particle. The neighbor accumulator is shown in Figure 11b. All incoming data are buffered. Three registers are used to record the currently evaluated particle id which will be used to compare against the incoming particle id. If there is no conflict, then they are processed on accumulation. Otherwise, the particle will be sent back to the buffer.

4.3 LR Force Evaluation

4.3.1 Particle to Grid Mapping. Due to the large number of particles, the particle to grid mapping must be optimized to avoid adding additional stall cycles when each particle enters the system. This means replication is a must to avoid long delays. The first step is to evaluate each individual basis function per dimension to obtain a single particle contribution to an individual cell. As Figure 12 shows, one function takes 5 steps to evaluate a single equation. This unit can be replicated to evaluate all 4 functions simultaneously and each dimension is done in parallel requiring a total of 12 replications of each unit. After all functions are evaluated, values are combined to form 64 unique values representing the 4x4x4 neighbor grid of cells around the particle. These 64 cells are then accumulated with the previous information, found in their respective cells.

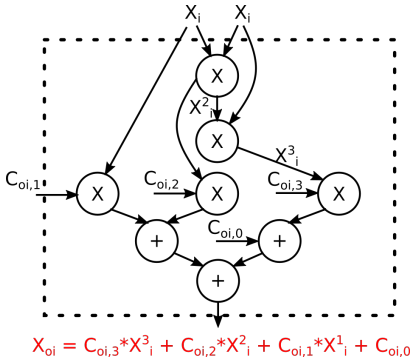

Figure 12: One instance of the particle to grid conversion equation: The unit is replicated 12 times. Four instances represent the four basis equations for each dimension X, Y, and Z.

4.3.2 FFT. Using the interleaved structure of the grid memory, the FFT implementation allows for the use of multiple FFT units to evaluate each dimension in parallel. Since this part of LR is not the bottleneck, a modest number of FFT blocks (16) is currently used.

4.3.3 Matching RL performance. By using a parameterized design, our sample implementation maintains a 2:1 timing ratio between LR and RL. Details are complex, but entail using methods such as folding and reusing logic.

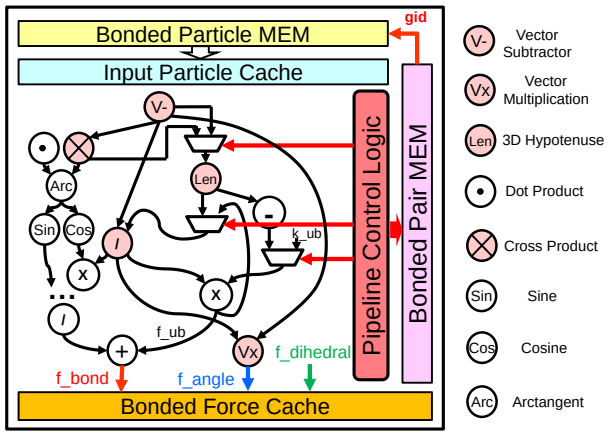

Figure 13: Bonded Force Pipeline

4.4 Bonded Force Pipeline

It is possible to stay within the time budget even if only one of the three evaluation pipelines (Figure 8) is active in a given cycle. Also, many functions overlap among the three interactions. Therefore, to maximize the DSP units' utilization ratio, we merge the three pipelines into a single one with control registers and muxes at different stages of the pipeline (Figure 13).

4.5 Summation Logic

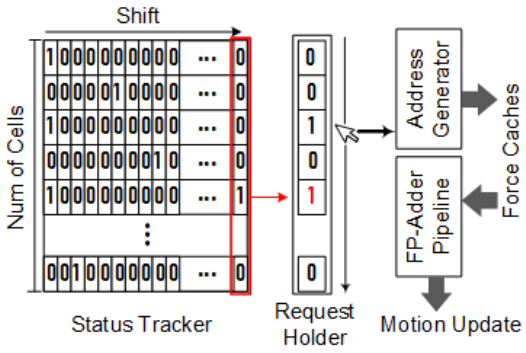

Figure 14: Summation logic with score boarding mechanism

Summation logic scoreboard support is shown in Figure 14. The *Status Tracker* tracks the force evaluation of cells with one entry tracks per cell. To start motion update of a certain cell, all its particles, and of its 26 neighborcells, must be fully evaluated. When that happens, the entries tracking the cell, as well as its neighborcells, are shifted right 1 step. The initial value of each entry is a one followed by 27 zeros. Once a cell and its 26 neighbors are all evaluated, the most-right bit of the corresponding entry becomes 1 and the scoreboard will send access request to the *Request Holder*.

Since there is only one summation pipeline, summing particles/cells is sequential. The Request Holder is used to deal with the scenario when the force summation of a cell is still in progress, but access requests for other cells have been received. The Request

Holder sends access requests to the address generator using round-robin. Once the address generator receives an access request, it can access the LR, RL, and Bonded Forces from the respective caches. The forces are summed and the results used for motion update.

4.6 Motion Update and Particle Migration

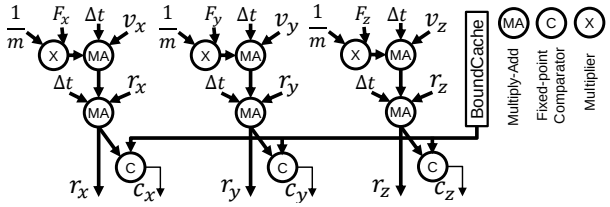

Figure 15: Motion Update Pipeline

Figure 15 shows the workflow inside the motion update module. When the updated positions are calculated, the module computes the target cell of the updated particle. Since we are using a short timestep, and we perform motion integration each iteration, particles rarely move across more than one cell. Each cell has a pre-stored lower and upper boundary. If the updated position falls in the current cell range, the output cell id remain the same as the input. Otherwise, the output cell id will add or subtract one depending on the comparison with the boundary value.

4.7 Workload-aware Adoption

There are numerous places where the design can be optimized with respect to workload. So that a user does not have to be concerned with this, we have implemented a workload-aware hardware generator script. Our script has two parts: an analytical performance estimator and an HDL generator. Given the dataset features (number of particles, particle density, etc.), the analytical estimator provides a coarse estimate of resource usage and simulation performance for various mapping schemes. The user can select the most promising combinations of mapping schemes and use the HDL generator to create the HDL code that is ready to run on FPGAs. The HDL generator script parameters include: workload mapping scheme, number of RL and LR evaluation units, and certain other hardware design parameters like buffer depth and floating-point unit configuration. This script, along with the scripts used for generating data for particle cache, interpolation indices, etc., will be made publicly available.

5 EVALUATION

5.1 FPGA Programming Methods

As is common with application acceleration using FPGAs, development proceeds in several steps. **(i)** The *golden model* is Amber 18 [42]; the force evaluation code guides the hardware design. **(ii)** The *software model* is composed of multiple independent subfunctions that are exact matches of the major hardware modules; results here are validated with respect to Amber 18. **(iii)** The hardware design is implemented using Verilog HDL derived from the software model. Simulation is performed using *ModelSim* 10.6c to establish logical correctness and, again, to validate results. **(iv)** The actual hardware implementation is generated using synthesis and place & route in the *Quartus Prime Pro* development suite; the output is a bitstream that is ready to load onto FPGA-chips. Also generated are FPGA resource usage numbers. Debugging is effected using two tools: *SignalTap Logic Analyzer*, which directly read out contents from on-chip register and memory, and a vendor-provided API to read out data in batch.

In general, even with a well-defined golden model like Amber, creating a high-quality mapping onto FPGAs still requires a significant amount of code/design restructuring. Here, the inner-most loop–pair-wise force evaluation–is a straightforward translation. But other parts of the design, including the particle data organization and data flow control, require redesign to make it efficient and guarantee the correctness. We have experimented with implementing parts of this design using OpenCL [43, 44] with promising results; with appropriate coding methods, HDL-level performance appears to be achievable [45, 46].

5.2 Experimental Setup

We have implemented, tested, and verified the designs on a Reflex XpressGX S10-FH200G Board with an Intel Stratix 10 1SG280 LU2F50E2VG chip [47]. This chip is high-end with 933,120 ALMs, 11,721 RAM blocks, and 5,760 DSP units, which makes it a good target for implementing FPGA/MD. To get the comparable MD simulation performance on CPU and GPU, we installed Amber 18 [42] on a single node with an Intel Platinum 8160 2.1GHz CPU and various Nvidia GPUs. The operating system is CentOS 7.4.

The dataset is Dihydrofolate Reductase (DFHR), a 159-residue protein in water, with 23,558 atoms [1]. The dataset is constrained to a bounding box of 62.23×62.23×62.23Å, with a cutoff radius of 9Å. The simulation timestep if 2fs with Particle Mesh Ewald (PME) every two iterations.

The generator script (introduced in Section 4.7) is run in CentOS 6.10, with a gcc version 4.4.7. Given the dataset size, the performance estimator will provide a theoretical simulation throughput and a good estimate of resource usage on the six different mapping schemes. The HDL generator is used to generate the hardware code and run the design on FPGAs installed on platform described.

5.3 RL Performance Trade-offs

Table 1: Filter bank resource usage under two implementations

Design \ Usage	ALM	BRAM	DSP
Direct Computation	5077(0.5%)	66(0.6%)	57(1%)
Planar	5605(0.5%)	65(0.6%)	15(0.3%)

5.3.1 RL Filter Resource Usage. In Section 4.2.1 we propose two designs. Since the planar method requires an extra datapath in the force pipeline to generate r^2, we evaluate aggregate resource usage, including both filter bank and force pipeline. Table 1 gives the resource usage of a single bank consisting of a force evaluation unit and eight filters. We note that a 10% increase in ALM usage saves 74% on DSPs; the planar method thus enables more pipelines per FPGA. All following evaluations assume planar filters.

Table 2: Force evaluation pipeline resource usage and performance comparisons

Design	Datatype	ALM	BRAM	DSP	Frequency (MHz)	Latency(Clock Cycle)
Direct Computation	32-bit Fixed-Point	1365(0.15%)	4(0.04%)	19(0.33%)	505	89
Interpolation	32-bit Fixed-Point	866(0.09%)	17(0.15%)	10(0.17%)	433	22
Direct Computation	Single Float	698(0.08%)	3(0.03%)	11(0.19%)	486	59
Interpolation	Single Float	462(0.05%)	17(0.15%)	6(0.09%)	654	14

5.3.2 RL Interpolation Setup. We first measure the interpolation accuracy with respect to direct computation in single-precision floating-point. Our experiments cover up to third order interpolation with different numbers of intervals in each section. The evaluation results are shown in Table 3. We notice that the higher interpolation is, the fewer intervals we need to achieve the same level of accuracy, which means less BRAM usage. But on the other hand, the DSP usage and number of indexes increase with interpolation order. In order to preserve DSP units for replicating more force evaluation pipelines, we choose first order interpolation with 256 intervals per segment.

Table 3: Accuracy comparison of interpolation with table lookup with respect to direct computation. The columns denote the different number of intervals inside each section.

Interval	16	32	64	128	256
1st Order	99.7700	99.9336	99.9842	99.9960	99.9990
2nd Order	99.9899	99.9988	99.9991	99.9999	99.9999
3rd Order	99.9993	99.9999	99.9999	99.9999	99.9999

5.3.3 RL Force Pipeline Comparison. Section 3.2.2 describes decisions between fixed and float and direct and interpolated. Resource utilization is shown in Table 2. We use first order interpolation with 256 intervals. Again, the deciding factor is DSP usage, which favors interpolation with floating-point.

5.4 LR Performance Trade-offs

We parameterize the LR modules to make trade-offs between LR evaluation time and resource usage. Here we specifically show the effect of varying particle to grid mapping modules, while maintaining a constant number of input particles (see Table 4). We note overall that the mapping units have a small effect on resource consumption. For performance, however, adding a single mapping unit improves performance by more than a third. Beyond this, however, the FFT & IFFT latency becoming the dominant factor.

Table 4: LR resource usage and evaluation time

# LR GridMapping Units	ALM	BRAM	DSP	Latency (Clock Cycle)
1	209,284	2,608	1,845	190,069
2	210,406	2,608	2,019	119,395
3	211,528	2,608	2,193	106,307
4	212,650	2,608	2,367	101,727

5.5 Bonded Force Performance Trade-offs

In Section 4.4, we propose merging three bonded force pipelines into a single one. Table 5 shows the benefits of this approach. The proposed merged pipeline saves 27%, 43%, 25% on ALM, BRAM, and DSP, respectively. As the bonded force is still almost twice as fast as LR and RL this design decision is justified.

Table 5: Bonded force pipeline resource usage

	ALM	BRAM	DSP	Latency (Clock Cycles)	Frequency (MHz)
Bond	1,481	10	18	148	398
Angle	13,691	77	153	187	401
Dihedral	12,432	77	201	244	392
Merged	20,109	93	278	276	330

5.6 Full System Performance

5.6.1 Overall System Resource Utilization. As described in Section 4.3, RL, LR, and Bonded are designed for balanced load. To recap, as introduced in Section 3.2.3, we have two particle-to-memory mapping schemes: mapping all the particle in a single large memory unit and mapping particles onto small block RAMs based on the cell it belongs to. We also have three workload to pipeline mapping schemes: all pipelines work on same reference particle, all pipelines work on the same home cell with different reference particles, and each pipeline works on a different home cell. This yields six different designs.

Table 6 lists the resource utilization and the number of function units that can fit onto a single FPGA-chip under different RL mapping schemes. We also list the stand-alone performance number for both RL and LR parts. By adjusting the number of *LR Particle to Grid Mapping* modules (column 6), we aim to make the LR evaluation time about twice as much as RL (column 8 & 9).

We note first that Designs 2 & 4 can only fit 35 pipelines. Those two designs have hundreds of memory modules, while the workload mapping requires each pipeline to receive data from all cells. Because of this, a very large on-chip switch (mux-tree based) is required, which consumes a large number of ALMs (196,805). Compared with *Mem 2*, designs using *Mem 1* all have more pipelines, due to the convenience of having a single source of input data. Given the resource usage comparison, it seems that having a global memory provides benefits of having more pipelines mapped on to a single chip. However, the stand-along RL performance shows otherwise. We describe this next.

5.6.2 MD System Performance. Table 7 lists performance numbers for the DHFR dataset on various platforms, including multi-core CPU, GPU, and our six HDL implementations on different FPGAs.

Table 6: Full system resource usage. Columns 2-4 are post place&route. Columns 5-6 give the number of replications of RL pipeline and LR Grid Mapping units in each design. Column 7 lists running frequency of each design. The last two give the stand-alone performance of RL and LR units.

Design	ALM	BRAM	DSP	# RL Pipes	# LR Units	Freq (MHz)	RL Iter Time (μs)	LR Iter Time (μs)
Design 1: Mem 1 + Dis 1	657,808 (71%)	9,430 (80%)	4,419 (77%)	52	1	350	64,376.87	817.56
Design 2: Mem 2 + Dis 1	747,075 (80%)	9,077 (77%)	4,338 (75%)	35	2	340	349.30	513.45
Design 3: Mem 1 + Dis 2	657,508 (71%)	9,430 (81%)	4,038 (70%)	52	1	340	968.95	817.56
Design 4: Mem 2 + Dis 2	746,775 (80%)	9,077 (77%)	3,957 (69%)	35	2	340	292.89	513.45
Design 5: Mem 1 + Dis 3	646,946 (69%)	9,362 (80%)	4,197 (73%)	51	2	350	270.72	513.45
Design 6: Mem 2 + Dis 3	586,336 (63%)	9,362 (80%)	4,047 (70%)	41	2	350	260.37	513.45

The CPU and Titan XP GPU numbers come from collaborators in an industrial drug design environment. The RTX 2080 and Titan RTX GPU performance numbers are public available from Amber [48]. Compared with the best-case single CPU performance, the best-case FPGA design has one order of magnitude better performance. The FPGA design has 10% more throughput than that of the GPU performance. Much more evaluation needs to be done, but we believe these results to be promising.

As shown in Table 6, RL is the limiting factor on the overall performance. The poor performance of Design 1 is due to the memory bandwidth limitation: for most cycles, pipelines are waiting for data. In Design 2, the distributed memory provides much higher read bandwidth. Design 3 faces a different problem: the number of particles per cell (70) is not a multiple of the number of pipelines (52), which means a set of pipelines (18) is idle after evaluating a single reference particle. It also suffers from memory bandwidth limitations. Design 4 has a happy coincidence that its pipeline count (35) can be divided evenly into 70 and most pipelines will have close to 100% usage (this is subject to dataset). Designs 5 & 6 might be supposed to have similar performance, but in Design 5 there is overhead on reading the first sets of input data from a single memory unit. But the subsequent read latency can be fully hidden.

5.7 Dataset Impact on Mapping Selection

Our system takes advantage of FPGAs' reconfigurability to fully customize the number of pipelines and the mapping scheme of workload and particle storage. Since RL evaluation takes both most of the resources and evaluation time, we focus here on examining the RL performance. Using the scripts introduced in Section 4.7, we can quickly estimate the number of pipelines and resource usage based on the size of the input dataset and number of cells, along with an estimation of the simulation performance from the six different mapping schemes. In order to further demonstrate the selection of mapping schemes, we use a variety of datasets (5K to 50K) and cutoff radii (leading to different cell sizes). Characteristics are shown in Table 8.

The number of pipelines and performance are shown in Figure 16. We note first (from Figure 16a) that the dataset size has little impact on the number of pipelines we can map on a single Stratix 10 FPGA until the dataset grows large enough to cause a resource conflict (in BRAMs). However, this is not the case on simulation performance as shown in Figure 16b. All the performance number is normalized to the Design 1 performance for each dataset. We have the following observations: **(i)** Design 1 with single particle memory and workload distribution 1 always has the worst performance due to memory bottleneck; **(ii)** When the dataset is sparse (see Dataset 1, 3, 5), Design 6 tends to return the best performance, and the relative performance among the six designs is similar; **(iii)** When the dataset is dense (see Dataset 2, 4, 6), workload distribution 3 provides fewer benefits comparing with workload distribution 2; this is especially clear when the dataset is small and dense.

Table 7: Performance comparison: the middle column shows time to perform one full iteration (23k dataset); the right column shows throughput with a $2fs$ timestep.

Platform	Iteration Time (μs)	Simulation Rate (ns/day)
CPU 1-core	85,544	2.02
CPU 2-core	38,831	4.45
CPU 4-core	21,228	8.14
CPU 8-core	11,942	14.47
CPU 16-core	6,926	24.95
GTX 1080 GPU	720	240.13
Titan XP GPU	542	318.97
RTX 2080 GPU	389	444.05 [48]
Titan RTX GPU	304	567.53 [48]
Design 1: Mem 1 + Distribution 1	64,411	2.68
Design 2: Mem 2 + Distribution 1	370	467.40
Design 3: Mem 1 + Distribution 2	1003	172.36
Design 4: Mem 2 + Distribution 2	313	551.55
Design 5: Mem 1 + Distribution 3	291	593.55
Design 6: Mem 2 + Distribution 3	274	630.25

Table 8: Various testing datasets evaluating the impacts on workload mapping selection

	Particle #	Cell #	Particle #/Cell
Dataset 1	5,000	63	80
Dataset 2	5,000	12	417
Dataset 3	20,000	252	80
Dataset 4	20,000	50	400
Dataset 5	50,000	625	80
Dataset 6	50,000	125	400

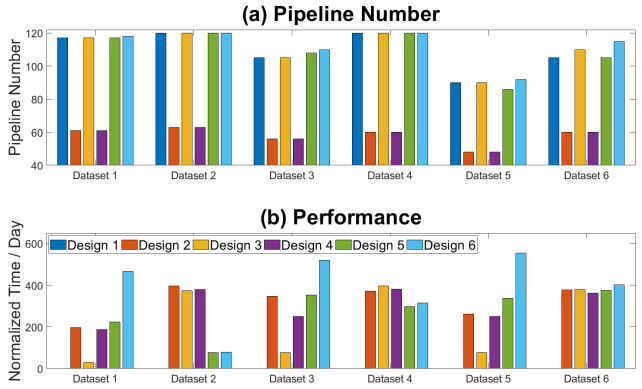

Figure 16: Performance with Different Datasets: (a) Number of RL pipelines that can map onto a single FPGA; (b) RL simulation performance, normalized to Design 1 for each dataset.

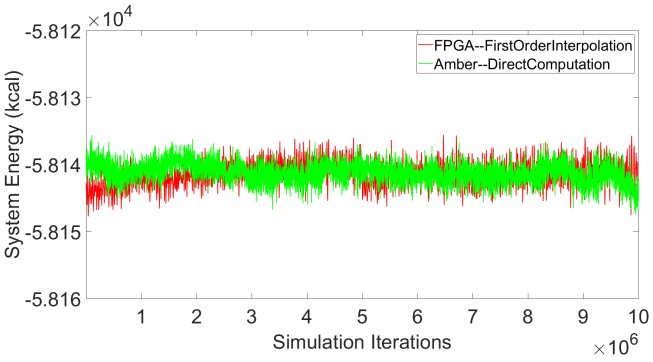

Figure 17: Energy Waveform

5.8 Verification and Validation

As is usual with complex FPGA designs, we have multiple levels of verification, starting with MatLab models of the computations, HDL simulations of components, HDL simulations of full system, and the actual implementation. These are also validated with respect to Amber 18.

To validate using energy waveforms, we run two sets of simulations to collect system energy values using different evaluation methods: the FPGA using 1st-order interpolation and Amber running on a CPU (see Figure 17). We note that our simulation system maintains an equilibrium state and that the energy level is similar to Amber's. The variance is likely due to the fact that Amber uses a sophisticated motion integration method [1] and different smoothing techniques [49], which are not yet implemented in our system.

6 EXTENSIONS TO FPGA CLUSTERS

One of the motivations for using FPGAs in HPC is their support for communication. FPGA-clusters with FPGAs directly linked through their Multi-Gigabit Transceivers (MGTs) have a proven advantage over other commodity architectures in facilitating communication that is both high bandwidth and low latency; but also in the collocation of compute and communication on the same device [50–52]: there is single cycle latency between application and physical network layers. For MD, these benefits have previously been demonstrated through strong scaling of small 3D FFTs [19, 20].

Extending the current single-chip MD simulation system onto FPGA clusters is work in progress, but we can estimate the performance of such systems using the model presented in [20] and the results in Section 5. In general, adding cluster support requires additional elements (see, e.g., [9]). In the case where the number of FPGAs is modest with respect to the number of cells, e.g., ≤ 64, the current design can be used almost as is, but augmented with communication support as described in [17, 53–55]. This support takes $< 10\%$ of chip area; the communication model has been validated on a real system up to 64 nodes.

7 SUMMARY AND FUTURE WORK

We present an end-to-end MD system on a single FPGA featuring online particle-pair generation; force evaluation on range-limited, long range, and bonded interactions; motion update; and particle data migration. We provide an analysis of the most likely mappings among particles/cells, BRAMs, and on-chip compute units. We introduce various microarchitecture contributions on routing the accumulation of hundreds of particles simultaneously and integrating motion update. A set of software scripts is created to estimate the performance of various design choices based on different input datasets. We evaluate the single-chip design on a commercially available Intel Stratix 10 FPGA and achieve a simulation throughput of 630ns/day on a 23.5K DFHR dataset, which is comparable to the analogous state-of-the-art GPU implementations.

In continuing work, besides mapping to FPGA-centric clusters, the current version has significant upgrade potential. For example, most of the resources are currently devoted to floating point units for the RL computation; the FPGA's arithmetic flexibility (which it has in common ASIC implementations) can be exploited.

ACKNOWLEDGMENTS

This work was supported, in part, by the NSF through award CCF-1618303/7960; the NIH through award 1R41GM128533; by a grant from Red Hat; and by Intel through donated FPGAs, tools, and IP.

REFERENCES

[1] D.A. Case, I.Y. Ben-Shalom, S.R. Brozell, D.S. Cerutti, T.E. Cheatham, et al. Amber 18. Technical report, University of California, 2018.
[2] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kale, and K. Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26:1781–1802, 2005.
[3] D. van der Spoel, E. Lindahl, B. Hess, G. Groenhof, A.E. Mark, and H.J.C. Berendsen. GROMACS: fast, flexible, and free. *Journal of Computational Chemistry*, 26:1701–1718, 2005.
[4] P. Eastman and V.S. Pande. OpenMM: A Hardware-Independent Framework for Molecular Simulations. *Computing in Science and Engineering*, 4:34–39, 2010.
[5] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics (JCP)*, 117(1):1–19, 1995.
[6] Z. Cournia, B. ALlen, and W. Sherman. Relative binding free energy calculations in drug discovery: Recent advances and practical considerations. *Journal of Chemical Information and Modeling (JCIM)*, 57:2911–2937, 2017.
[7] NVIDIA, https://images.nvidia.com/content/tesla/pdf/Molecular-Dynamics-July-2017-MB-slides.pdf. *Molecular Dynamics (MD) on GPUs*, 2017.
[8] Shaw, D.E., et al. Anton, a special-purpose machine for molecular dynamics simulation. In *International Symposium on Computer Architecture (ISCA)*, pages 1–12, 2007.
[9] J. Grossman, B. Towles, B. Greskamp, and D. Shaw. Filtering, Reductions and Synchronization in the Anton 2 Network. In *Proceedings of International Parallel*

and Distributed Processing Symposium (IPDPS), pages 860 – 870, 2015.

[10] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow. Reconfigurable molecular dynamics simulator. In *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, pages 197–206, 2004.

[11] T. Hamada and N. Nakasato. Massively parallel processors generator for reconfigurable system. *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, pages 329–330, 2005.

[12] R. Scrofano and V. Prasanna. Preliminary investigation of advanced electrostatics in molecular dynamics on reconfigurable computers. In *Proceedings of ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2006.

[13] V. Kindratenko and D. Pointer. A case study in porting a production scientific supercomputing application to a reconfigurable computer. In *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, pages 13–22, 2006.

[14] S.R. Alam, P.K. Agarwal, M.C. Smith, J.S. Vetter, and D. Caliga. Using FPGA devices to accelerate biomolecular simulations. *Computer*, 40(3):66–73, 2007.

[15] M. Chiu and M.C. Herbordt. Molecular dynamics simulations on high performance reconfigurable computing systems. *ACM Transaction on Reconfigurable Technology and Systems (TRETS)*, 3(4):1–37, 2010.

[16] J. Cong, Z. Fang, H. Kianinejad, and P. Wei. Revisiting FPGA Acceleration of Molecular Dynamics Simulation with Dynamic Data Flow Behavior in High-Level Synthesis. *arXiv preprint arXiv:1611.04474*, 2016.

[17] J. Sheng, B. Humphries, H. Zhang, and M.C. Herbordt. Design of 3D FFTs with FPGA Clusters. In *Proceedings of IEEE High Performance Extreme Computing Conference (HPEC)*, 2014.

[18] J. Sheng, C. Yang, and M.C. Herbordt. Towards Low-Latency Communication on FPGA Clusters with 3D FFT Case Study. In *Proceedings of International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2015.

[19] 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.

[20] J. Sheng, C. Yang, A. Caulfield, M. Papamichael, and M.C. Herbordt. HPC on FPGA Clouds: 3D FFTs and Implications for Molecular Dynamics. In *Proceedings of IEEE Conference on Field Programmable Logic and Applications (FPL)*, 2017.

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

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

[23] J.M. Haile. *Molecular Dynamics Simulation*. Wiley, New York, NY, 1997.

[24] Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign, http://www.ks.uiuc.edu/Research/cloud/. *Molecular Modeling in the Cloud*, Accessed 6/2017.

[25] C. Young, J.A. Bank, R.O. Dror, J.P. Grossman, J.K. Salmon, and D.E. Shaw. A 32x32x32, spatially distributed 3D FFT in four microseconds on Anton. In *Proceedings of ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–11, 2009.

[26] W.M. Brown, P. Wang, S. Plimpton, and A. Tharrington. Implementing molecular dynamics on hybrid high performance computers–short range forces. *Computer Physics Communications (CPC)*, 182(4):898–911, 2011.

[27] M. Chiu and M.C. Herbordt. Efficient filtering for molecular dynamics simulations. In *Proceedings of IEEE Conference on Field Programmable Logic and Applications (FPL)*, 2009.

[28] C. Yang, T. Geng, T. Wang, J. Sheng, C. Lin, V. Sachdeva, W. Sherman, and M.C. Herbordt. Molecular Dynamics Range-Limited Force Evaluation Optimized for FPGA. In *Proceedings of International Conference on Application Specific Systems, Architectures, and Processors (ASAP)*, 2019.

[29] Intel. Intel Stratix 10 Device Datasheet. 2018.

[30] Y. Gu, T. VanCourt, and M.C. Herbordt. Accelerating molecular dynamics simulations with configurable circuits. *IEE Proceedings – Computers and Digital Technology*, 153(3):189–195, 2006.

[31] Y. Gu, T. VanCourt, and M.C. Herbordt. Explicit design of FPGA-based coprocessors for short-range force computation in molecular dynamics simulations. *Parallel Computing*, 34(4-5):261–271, 2008.

[32] Toshiyuki Fukushige, Makoto Taiji, Junichiro Makino, Toshikazu Ebisuzaki, and Daiichiro Sugimoto. A highly parallelized special-purpose computer for many-body simulations with an arbitrary central force: MD-GRAPE. *Astrophysical Journal (ApJ)*, 468:51–61, 1996.

[33] Y. Komeiji, M. Uebayasi, R. Takata, A. Shimizu, K. Itsukashi, and M. Taiji. Fast and accurate molecular dynamics simulation of a protein using a special-purpose computer. *Journal of Computational Chemistry*, 18(12):1546–1563, 1997.

[34] R. Scrofano, M. Gokhale, F. Trouw, and V. Prasanna. A hardware/software approach to molecular dynamics on reconfigurable computers. In *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, pages 23–32, 2006.

[35] M. Chiu, M.A. Khan, and M.C. Herbordt. Efficient calculation of pairwise nonbonded forces. In *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2011.

[36] A. Sanaullah, A. Khoshparvar, and M.C. Herbordt. FPGA-Accelerated Particle-Grid Mapping. In *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2016.

[37] P. D'Alberto, P.A. Milder, A. Sandryhaila, F. Franchetti, J.C. Hoe, J.M.F. Moura, M. Pueschel, and J.R. Johnson. Generating FPGA-Accelerated DFT Libraries. In *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2007.

[38] C. Dick. Computing Multidimensional DFTs Using Xilinx FPGAs. In *Proceedings of International Conference on Signal Processing Applications and Technology*, 1998.

[39] B. Humphries, H. Zhang, J. Sheng, R. Landaverde, and M.C. Herbordt. 3D FFT on a Single FPGA. In *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2014.

[40] Intel. FFT IP Core User Guide. 2017.

[41] Q. Xiong and M.C. Herbordt. Bonded Force Computations on FPGAs. In *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2017.

[42] R. Salomon-Ferrer, D.A. Case, and R.C. Walker. An overview of the amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(2):198–210, 2013.

[43] C. Yang, J. Sheng, R. Patel, A. Sanaullah, V. Sachdeva, and M.C. Herbordt. OpenCL for HPC with FPGAs: Case Study in Molecular Electrostatics. In *Proceedings of IEEE High Performance Extreme Computing Conference (HPEC)*, 2017.

[44] A. Sanaullah and M.C. Herbordt. FPGA HPC using OpenCL: Case Study in 3D FFT. In *Proceedings of International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2018.

[45] H.R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, and S. Matsuoka. Evaluating aand Optimizing OpenCL Kernels for High Performance Computing with FPGAs. In *Proceedings of ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2016.

[46] A. Sanaullah and M.C. Herbordt. An Empirically Guided Optimization Framework for FPGA OpenCL. In *Proceedings of IEEE Conference on Field Programmable Technology (FPT)*, 2018.

[47] Reflex. Xpressgxs10-fh20xgt board. 2018.

[48] Amber. Amber18: pmemd.cuda performance information. http://ambermd.org/GPUPerformance.php#RCWBenchmarks, 2018.

[49] D.A. Case, T.E. Cheatham III, T. Darden, H. Gohlke, R. Luo, K.M. Merz, Jr., A. Onufriev, C. Simmerling, B. Wang, and R.J. Woods. The Amber biomolecular simulation programs. *Journal of Computational Chemistry*, 26:1668–1688, 2005.

[50] R. Sass, et al. Reconfigurable computing cluster (RCC) project: Investigating the feasibility of FPGA-based petascale computing. In *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, pages 127–138, 2007.

[51] A. Putnam et al. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, pages 13–24, 2014.

[52] 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 *Proceedings of IEEE High Performance Extreme Computing Conference (HPEC)*, 2016.

[53] J. Sheng, C. Yang, and M.C. Herbordt. Collective Communication on FPGA Clusters with Static Scheduling. In *Proceedings of International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2016.

[54] J. Sheng, C. Yang, and M.C. Herbordt. High Performance Dynamic Communication on Reconfigurable Clusters (Extended Abstract). In *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2018.

[55] J. Stern, Q. Xiong, A. Skjellum, and M.C. Herbordt. A Novel Approach to Supporting Communicators for In-Switch Processing of MPI Collectives. In *Proceeding of Workshop on Exascale MPI (ExaMPI)*, 2018.