2021

Accelerated long range electrostatics computations on single and multiple FPGAs

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Thesis

ACCELERATED LONG RANGE ELECTROSTATICS COMPUTATIONS ON SINGLE AND MULTIPLE FPGAS

by

ANTHONY DUCIMO

B.S., Worcester Polytechnic Institute, 2012

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science

2021

Approved by

First Reader

Martin C. Herbordt, PhD
Professor of Electrical and Computer Engineering

Second Reader

Wenchao Li, PhD
Assistant Professor of Electrical and Computer Engineering
Assistant Professor of Systems Engineering

Third Reader

Tom VanCourt, PhD
Software Engineer
Akamai Technologies

Find your light and keep it ever at your front,
lest you be resigned to chasing shadows.

— Anonymous

Acknowledgments

I would like to take this opportunity to thank my advisor, Prof. Martin Herbordt. Despite being a part-time student, never having registered for a course with him, and being candid about a prior unsuccessful project venture, he decided to take me on as an advisee. Shortly thereafter, he tasked me with very fulfilling research work in the MD area. Any and all successes stemming from this thesis are credited to his patience in dealing with my inexpertise and his resourcefulness in playing to my strong suits.

Thank you to my Thesis committee, Professor Wenchao Li and Dr. Tom VanCourt. It means a lot to take time out of your busy schedules to review drafts of this document and assess the accomplishments reported here, not to mention the risk of sponsoring someone you have never met.

Many thanks to my mates in the CAAD lab. They say that saints often realized just how far away they were from Godliness as they seemingly, increasingly approached it. Working with these brilliant people was a great and humbling experience, often times leaving me to desire more from myself. To Tong Geng and Chunshu Wu, my *de facto* project leads, thank you for picking up where my ineptitude or lack of effort left off. To Rushi Patel, thank you for allowing me to take on the effort you started and for the questions answered and advice offered early on in this thesis effort. Thank you to Sahan Lakshitha Bandara for providing solid design implementation advice despite having joined us halfway through this effort. Thank you to Anqi Guo for all the MS Thesis preparation device you offered, I made great use of it. Lastly, but equally as importantly, thank you to Robert Munafo and Pierre-François Wolfe. Though not directly involved in this project, they managed to keep all the infrastructure up and running smoothly so that I could accomplish all of my tasks, including the ones listed here, but perhaps more importantly, the ones that are taken for granted.

I'd like to take this opportunity to thank Intrinsix Corp. who had to put up with my fluid scheduling not only throughout this effort, but over the 4.5 years of my part-time Master's career. The work opportunities afforded to me as their employee have polished me much like a stone tumbler, but given the results so far, I would not have it any other way. The methods and techniques learned by working on numerous projects were greatly utilized in this thesis effort and allowed me to accomplish many things in a relatively short amount of time.

To my mother, whom I lost in the first year of this 4.5 year journey, I am most thankful to you. Looking back on things, you were solely responsible for instilling in me the desire to learn. Without that, I probably would have never even considered pursuing my Master's degree. It's tragically ironic how this desire to learn, which brought us together in my earliest years, is responsible for the neglect I sent your way in your final couple of years. This guilt I continue to carry will never go away, but I take some some solace in knowing that you are forever proud of all my accomplishments, no matter how trivial they may be.

To my father, though your years of raising a family as a single parent following mom's illness may not earn you any medals or trophies, it is laudable and never goes unappreciated. You kept our family together, made sure that we were all fed and cared for, and continued to push education as the end goal for all of us kids. Thank you always being there and for continuing to be there for us even if we are all well into our adulthood.

Lastly, thank you to all my friends, coworkers, and family who I have not already mentioned. Your constant encouragement did not go unnoticed and your striving to keep me sane in some of the more hectic moments of this Master's career is something I cannot and will not forget.

ACCELERATED LONG RANGE ELECTROSTATICS COMPUTATIONS ON SINGLE AND MULTIPLE FPGAS

ANTHONY DUCIMO

ABSTRACT

Classical Molecular Dynamics simulation (MD) models the interactions of thousands to millions of particles through the iterative application of basic Physics. MD is one of the core methods in High Performance Computing (HPC). While MD is critical to many high-profile applications, e.g. drug discovery and design, it suffers from the strong scaling problem, that is, while large computer systems can efficiently model large ensembles of particles, it is extremely challenging for *any* computer system to increase the timescale, even for small ensembles. This strong scaling problem can be mitigated with low-latency, direct communication. Of all Commercial Off the Shelf (COTS) Integrated Circuits (ICs), Field Programmable Gate Arrays (FPGAs) are the computational component uniquely applicable here: they have unmatched parallel communication capability both within the chip and externally to couple clusters of FPGAs. This thesis focuses on the acceleration of the long range (LR) force, the part of MD most difficult to scale, by using FPGAs.

This thesis first optimizes LR acceleration on a single-FPGA to eliminate the amount of on-chip communication required to complete a single LR computation iteration while maintaining as much parallelism as possible. This is achieved by designing around application specific memory architectures. Doing so introduces data movement issues overcome by pipelined, toroidal-shift multiplexing (MUXing) and pipelined staggering of memory access subsets. This design is then evaluated comprehensively and comparatively, deriving equations for performance and resource consumption and drawing metrics from previously developed LR hardware designs. Using this single-FPGA LR architecture as a base, FPGA

network strategies to compute the LR portion of larger sized MD problems are then theorized and analyzed.

This thesis has seven chapters. Chapter one formally introduces the topic with additional detail. Chapter two provides the theoretical background for MD, honing in on LR and its computational approximations while highlighting related work that either directly or indirectly sets the foundation for the single-FPGA LR hardware design. Chapter three discusses the single-FPGA LR accelerator hardware, detailing architectural the features as they relate to the theoretical background of MD and system performance. Chapter four evaluates the hardware, providing performance and resource consumption metrics from actual FPGA implementations, while drawing comparisons to pre-existing hardware solutions. It also provides a means of extrapolating both performance and resource consumption to larger MD sizes. Chapter five discusses single-development-board solutions to MD sizes that cannot fit on a single-FPGA alone. Chapter six expands hardware solutions for large MD sizes to networks of FPGA-boards. Chapter seven draws conclusions on the hardware designs, both developed and theorized, and lays out future work to be performed as a result of what was accomplished in this thesis.

Contents

List of Tables

List of Figures

List of Abbreviations

1D	....	One-Dimensional
2D	....	Two-Dimensional
3D	....	Three-Dimensional
ALM	....	Adaptive Logic Module
ASIC	....	Application-Specific Integrated Circuit
BRAM	....	Block RAM
COTS	....	Commercial Off the Shelf
CPU	....	Central Processing Unit
DFT	....	Design for Test
DSP	....	Digital Signal Processor
FCC	....	FPGA-Centric Cluster
FFT	....	Fast Fourier Transform
FIFO	....	First-In-First-Out
FLOPS	....	Floating-Point Operations per Second
FPGA	....	Field Programmable Gate Array
FS	....	Femtoseconds
GBS	....	Gigabits per second
GPU	....	Graphics Processor Unit
HDL	....	Hardware Description Language
HBM	....	High Bandwidth Memories
IC	....	Integrated Circuit
IFFT	....	Inverse Fast Fourier Transform
IO	....	Input/Output
LR	....	Long Range
MAC	....	Multiply-Accumulate
MD	....	Molecular Dynamics
MGT	....	Multi-Gigabit Transceiver
MUX	....	Multiplex
MS	....	Milliseconds
NS	....	Nanoseconds
PME	....	Particle Mesh Ewald
RAM	....	Random Access Memory
RAW	....	Read-After-Write
RL	....	Range Limited
SV	....	SystemVerilog

Chapter 1

Introduction

1.1 Molecular Dynamics acceleration for strong scaling

Molecular Dynamics simulation (MD) is a central method in high performance computing (HPC) with applications throughout engineering and science. MD uses an iterative application of Newtonian mechanics on ensembles of atoms and molecules (Rapaport, 2004). *Acceleration* of MD is a critical problem — there is a many order-of-magnitude gap between the largest current simulations and the potential physical systems to be studied (Tajkhorshid et al., 2003; Shirvanyants et al., 2012). There are dozens of MD packages in production use, e.g. (Case et al., 2005; Phillips et al., 2005; van der Spoel et al., 2005; Eastman and Pande, 2010; Plimpton, 1995), all of which exist in parallel versions and many of which have been accelerated with GPUs. While many of these packages support efficient *weak scaling*, i.e. performance scales with computing resources as long as the problem size scales in proportion, *strong scaling* remains problematic. That is, for fixed, modest problem sizes, adding significant computing resources does not result in an increase in performance. The failure to support strong scaling is, in large part, the cause of the performance gap. For instance, small problem sizes (20K-50K particles) are common in drug discovery and design (Cournia et al., 2017; NVIDIA, 2017). There, long timescales, on the order of milliseconds per day, would be extremely beneficial, but only hundreds of nanoseconds per day are currently being achieved (Yang et al., 2019a).

The strong scaling problem arises in MD because of the necessarily iterative nature of the application. Each timestep (of force computation and motion update) simulates roughly two femtoseconds (E-15s) of reality; thus simulating a millisecond of reality per day requires that each timestep be computed in hundreds of nanoseconds. The critical path lies with the Long Range Force computation (LR), which requires global communication; this makes accelerating LR a classic strong scaling problem. Large numbers of processors are required to generate sufficient FLOPs, but then proportionally increase the communication time. This disconnect becomes overwhelming when simulating small molecules: applying large computing resources is both unhelpful and extremely wasteful.

The only way to address the strong scaling problem is to reduce communication latency to the barest minimum which, in the end, requires direct communication links–application layer to application layer–among the integrated circuits (ICs) in the cluster. This approach was originally taken by the Grape family of ASIC-based MD processors (Ohmura et al., 2014), but currently being used (with great success) by DE Shaw with the Anton family (Shaw, D.E., et al., 2007; Young et al., 2009; Grossman et al., 2015). But while ASIC-based solutions can have orders-of-magnitude better performance than commodity clusters, they also have issues with general availability, prohibitive cost, plus all the problems inherent with small-run ASIC-based systems.

As is often the case when looking for cost-effective ASIC replacement, FPGAs provide a viable alternative (Gokhale and Graham, 2005; Herbordt et al., 2007; Hauck and DeHon, 2008; Herbordt et al., 2008a; VanCourt and Herbordt, 2009; Benkrid and Vanderbauwhede, 2013). FPGAs have been explored as possible MD accelerators for many years (Azizi et al., 2004; Hamada and Nakasato, 2005; Gu et al., 2006a; Gu et al., 2006c; Gu et al., 2006b; Alam et al., 2007); see (Schaffner and Benini, 2018) for a survey. The first generation of complete FPGA/MD systems accelerated only the range limited (RL) force and used CPUs for the rest of the computation (Gu et al., 2005a; Gu et al., 2005b; Kindratenko and Pointer, 2006; Scrofano et al., 2008). 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, proof-of-concept studies have shown that FPGA clusters (Pascoe et al., 2010; Khan and Herbordt, 2012; Sheng et al., 2015; George et al., 2016; Sheng et al., 2017a; Plessl, 2018; Miyajima et al., 2018; Boku et al., 2019) can have performance approaching that of ASIC clusters for LR (Sheng et al., 2014; Sheng et al., 2017b; Lawande et al., 2016). These studies, however, only implement the critical path as little more than a microbenchmark; creating a complete LR, even for a single FPGA, remains unstudied.

In this thesis, a state-of-the-art, single-FPGA LR architecture is developed and evaluated. Following the evaluation of this single-FPGA design, the accommodation of larger grid sizes using both a single FPGA coupled with off-chip but on-board memory and a network of FPGAs are explored.

1.2 Real World Applicability

MD can be found at the core of computational chemistry and is central to computational biology. Variations in particle ensemble size are used to model differing biological constructs. Typically, the more complex the structure, the more particles in the ensemble. Over time, these particle interactions can model various biological processes, with more complex processes requiring longer timescales. Table 1.1 lists some of these structures and processes.

Directly applying the equations from Newtonian physics, MD simulations of any particle ensemble size over any timescale can be achieved, but it may take decades, if not centuries, to do so. Single time-steps (typically $2 * 10^{-15}$s or 2fs) of reality can be simulated on a single core in around a second for small particle ensembles ($< 100K$). Staying at the protein level, thousand-core and small multi-GPU systems (2-4 GPUs) have been able to simulate 500ns of reality in a single day. Anton 2, a 512-ASIC processor, has been able to model these ensembles for hundreds of microseconds and milliseconds of reality in a day and 16 days, respectively (Grossman et al., 2015).

We now give an example of how a physical application requirement translates into a computational requirement. Protein folding involves proteins (10s of thousands of particles) and folding (100s of microseconds to seconds). With a 2fs MD timestep, simulating 20ms of reality requires E13 timesteps. If MD is executed for, say, 10 days, then reality must be simulated at 2ms per day. This translates to E12 timesteps per day or roughly E7 timesteps per second or 1 timestep every 100ns.

Biological Construct	
Type	**Ensemble Size** [particles]
Protein	10^4-10^5
Virus	10^6-10^7
Chromotophore	10^8
Bacteria	10^{10}
Eukaryote	10^{13}
Biological Process	
Type	**Timescale Range** [s]
Chemical Reactions	10^{-15}-10^{-14}
Backbone Librations	10^{-12}-10^{-10}
Side-Chain Motions	10^{-11}-10^{-8}
Aromatic Ring Flipping	10^{-7}-10^{-5}
Large Conformational Changes	10^{-5}-10^{-3}
Protein Folding	10^{-4}-10^{1}
Protein Association and Dissociation	10^{0}-10^{4}
Protein Aggregation	10^{3}-10^{4}

Table 1.1: Computational Biology MD Modeling Examples. Table adapted from (Sharma et al., 2007; Proctor et al., 2011; Shirvanyants et al., 2012).

Two observations are made from these results. The first is that adding cores or GPUs does not improve the timescales achieved for small simulations; this is the MD strong scaling problem already described. The second is that there is a factor of 100 difference in performance between the best performance using off-the-shelf processing and the best performance using *heroic* dedicated technology developed at, conservatively, several orders of magnitude greater cost.

This thesis aims to explore the gap between the off-the-shelf and the heroic implementations of MD. The end goal, beyond the current scope, is to implement tightly coupled

FPGA clusters with dozens to hundreds of FPGAs. Such a cluster may achieve performance somewhat comparable to Anton 2 — but without having to design a full ASIC or develop such an ASIC network — while using much more recent technology. But the critical next step in this program is to address, on a single FPGA, the most difficult hurdle to achieving MD strong scaling: the efficient evaluation of the long-range force.

1.3 Contributions

Contributions of this thesis are as follows:

1. **LR friendly memory architecture.** Application-specific memory interleaving is implemented to simultaneously support a single particle's mapping to its 64 nearest-neighboring grid-points in three dimensions and remove the need for the transposition of data prior to successive 1D FFT calculations.

2. **Novel 1D FFT parallelization.** Given the inherent constraints of the application-specific memory architecture, a memory access pattern was developed to maximize the parallelization of 1D FFTs.

3. **Single-FPGA LR accelerators.** Reusing portions of previously developed designs and deploying some of the above contributions, working single-FPGA LR accelerators were implemented for multiple MD grid sizes and particle ensembles. Solutions for both a 16x16x16 MD grid accommodating 16^3 particles and a 32x32x32 accommodating 32^3 particles were successfully implemented with data suggesting the feasibility for a solution for a 64x64x64 grid accommodating 64^3 particles.

4. **Better Performance vs. Hardware Tradeoffs.** The single-FPGA LR accelerator developed here reduces the LR critical path of MD, enabling the simulation of ensembles on the order of 10^5 particles over hundreds of nanoseconds of reality to

complete in a single day. This performance is achieved with a single FPGA, avoiding the prohibitive resource costs of multi-core systems or ASIC design.

5. **Extensions to FPGA clusters.** Leveraging some of the features deployed in the single-FPGA accelerators, three multi-node network architectures are proposed to accelerate LR even further. Each network proposal includes high-level analysis of the messaging methods required for the various phases of LR acceleration.

1.4 Outline

The remainder of this thesis is broken up into 6 chapters. Chapter 2 provides additional MD background information and discusses prior work. Chapter 3 details the architecture of the single-FPGA LR design. Chapter 4 evaluates the single-FPGA LR design. Chapter 5 discusses the plausibility of using off-chip memory to accommodate larger grid sizes while keeping the entire LR design on a single FPGA board. Chapter 6 proposes multiple FPGA network topologies to distribute large-grid-sized LR problems over. In chapter 7 conclusions are drawn and future work is discussed.

Chapter 2

Background

This chapter provides an MD overview. First it is approached from a pure theoretical physics standpoint. Then, methods of MD approximation are introduced along with their aptitude for hardware acceleration. Having laid the groundwork for hardware accelerated MD, this chapter goes on to detail prior work used as the foundation for the design developed in conjunction with this thesis.

2.1 Molecular Dynamics Overview

This overview follows material found in similar presentations in a number of excellent texts and surveys, e.g., (Allen and Tildesley, 1987; Haile, 1997; Frenkel and Smit, 2002; Schlick, 2002; Rapaport, 2004). MD models interactions between ensembles of particles by iteratively calculating the forces acting on each particle exhibited by the remaining particles in the ensemble and the movement of each particle caused by the forces acting upon it. The forces computed depend on the system being simulated, but include bonded and non-bonded terms (Haile, 1997):

$$\mathbf{F}^{\mathbf{total}} = F^{bond} + F^{angle} + F^{dihedral} + F^{non-bonded} \tag{2.1}$$

Direct computation of the first three force components, the bonded forces, each have a complexity of $O(n_p)$, whereas direct computation of the non-bonded force is $O(n_p^2)$, where n_p is the number of particles in the ensemble. In order to reduce the computational complexity, computation of the non-bonded forces is typically approximated as discussed below. And because it represents over 99% of the FLOPs it is the part most likely to be performed by hardware accelerators (Essmann et al., 1995; Lee, 2005; Herbordt, 2013).

The non-bonded force can be broken up into two forces (Haile, 1997), the Lennard-Jones force :

$$\frac{F_i^{LJ}}{r_{ji}} = \sum_{j \neq i} \frac{\varepsilon_{ab}}{\sigma_{ab}} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \tag{2.2}$$

which, as seen in Figure 2·1, decays quickly as the radial distance between two particles in the pair increases and the Coulombic Force:

$$\frac{F_i^{C}}{r_{ji}} = q_i \sum_{j \neq i} \left(\frac{q_j}{|r_{ji}|^3} \right) \tag{2.3}$$

which does not decay with radial distance. This Coulombic force can be further

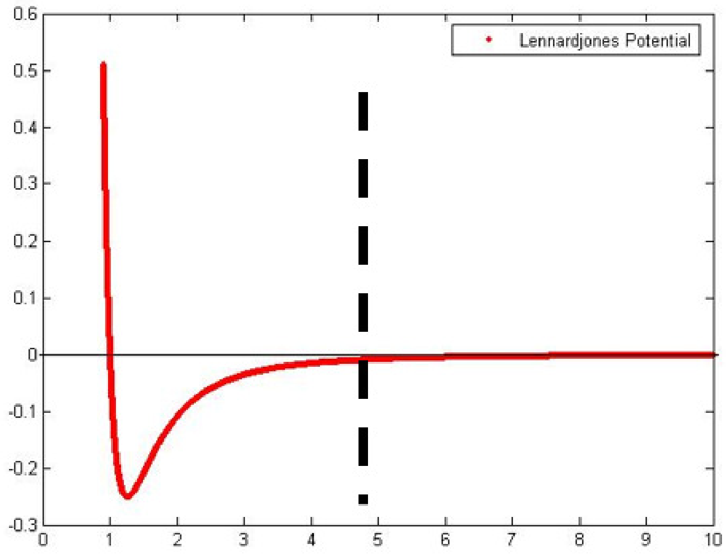

Figure 2·1: Lennard-Jones Force Plot

approximated by partitioning it into short-range and long-range components (Haile, 1997). The short-range component of the Coulombic force can be combined with the Lennard-Jones force as the range limited (RL) force, while the long-range component of the Coulombic force is computed on its own as the long range (LR) force.

2.2 Computing the Long Range Force

The LR force is calculated by solving the Poisson Equation for the given charge distribution (Haile, 1997).

$$\mathbf{F_i^{LR}} = \sum_{j \neq i} \frac{q_j}{|r_{ji}|} \vec{\mathbf{r_{ji}}} \tag{2.4}$$

LR force is often calculated with a grid-based map of the smoothing function converted from continuous space to a discrete grid coordinate system (Young et al., 2009). This method reduces the asymptotic complexity from $O(n_p^2)$ to $O(n_g \log n_g)$, where n_g is the number of grid-points, and results in the following equation (Procacci, 2009):

$$\mathbf{F}^{LR} \approx \sum_{x=0}^{K_x-1} \sum_{y=0}^{K_y-1} \sum_{z=0}^{K_z-1} \frac{\partial Q(x,y,z)}{\partial r_{ji}} \Theta_{i\ rec} * Q(x,y,z) \tag{2.5}$$

Each particle is interpolated to grid-points ($Q(x,y,z)$) by applying the following third-order basis function for charge density calculation (Skeel et al., 2002):

$$\phi(\xi) = \begin{cases} (1-|\xi|)(1+|\xi|-\frac{3}{2}\xi^2) & |\xi| \leq 1 \\ -\frac{1}{2}(|\xi|-1)(2-|\xi|)^2 & 1 \leq |\xi| \leq 2 \\ 0 & 2 \leq |\xi| \end{cases} \tag{2.6}$$

where ξ is the distance between the particle and any grid-point. Grid-points obtain their charge densities from neighboring particles within a range of two grid points in each direction. The resulting charge grid is then convolved with a Green's function ($\Theta_{irec} * Q(x,y,z)$) to transform it into a grid of electrostatic potentials. To simplify convolution, the charge grid is converted from a spatial domain to the Fourier Domain, where convolution becomes a simple multiplication, and then converted back to the spatial domain. The Fast Fourier Transform (FFT) is used to convert to the Fourier domain and the inverse FFT (IFFT) is used to convert back to the spatial domain. Once the grid of electrostatics has been generated, the three directional forces on each particle is calculated by taking the partial derivatives of the mapping function, multiplying the corresponding electrostatic potential grid-points, and summing across all grid points in 3D.

Overall, LR has the following phases.

1. Map particles into a charge grid
2. Perform 3D FFT
3. Perform convolution by multiplication
4. Perform 3D IFFT to create force grid
5. Map forces onto particles.

2.3 Previous Work and Design in FPGA Acceleration of MD

2.3.1 FPGA Architecture for HPC

FPGAs are well-known as ICs with configurable logic; the image is often of a sea of available logic configurable with an appropriate description. More accurate is that the FPGA is actually mostly VLSI building blocks embedded in a sea of configurable interconnects. The three most important building blocks are (i) DSP units, basically multipliers or floating point units; (ii) Block Rams (BRAMs), which are small (1KB) independently accessible multiported memories; and (iii) multi gigabit transceivers (MGTs) for I/O. There are on the order of 10,000 DSPs and BRAMs and 100 MGTs.

A typical FPGA/HPC design uses some fraction of the FPGA's resources in the following design pattern. The working set of data is stored in BRAMs (scratchpads). Computation is performed by custom logic pipelines constructed from DSPs and random logic. Data is streamed from scratchpads through one or more compute pipelines and back to scratch pads. The datapaths among scratchpads and pipelines can be almost arbitrarily complex.

The power of FPGA/HPC comes from the efficiency with which applications can be mapped to the hardware. Since control is embedded into the logic, every cycle generates payload. Pipelines are exactly what is required by the application and can be trivially short or hundreds of stages. Data types can be any precision or mode using only the logic needed. Since there are many thousands of scratchpads and DSPs, there can be that many data streams running in parallel. These scratchpads and compute pipelines can be interconnected chip-wide in almost any configuration. Of course some methods are more efficient than others; the most performant FPGA/HPC design patterns are described, e.g., in (Sanaullah and Herbordt, 2018a).

2.3.2 FPGA Acceleration of HPC

While not yet having the penetration of GPUs in HPC, FPGAs have been shown to be a likely component of future HPC systems. Other types of Molecular Modeling accelerated with FPGAs include Molecular Dynamics using Discrete Event Simulation, rather than the timestep driven studies here (Model and Herbordt, 2007; Herbordt et al., 2008b; Herbordt et al., 2009; Khan and Herbordt, 2011) and Molecular Docking, which often uses MD as one of its functions (VanCourt et al., 2004; VanCourt and Herbordt, 2005; VanCourt and Herbordt, 2006b; Sukhwani and Herbordt, 2008; Sukhwani and Herbordt, 2010). Other scientific applications include Adaptive Mesh Refinement (Wang et al., 2019b; Wang et al., 2019a) and Algebraic Multigrid (Haghi et al., 2020b). Of particular interest for FPGA use in computation, especially in data centers, is Machine Learning (Liu et al., 2016; Sanaullah et al., 2018b; Geng et al., 2018b; Geng et al., 2018a; Geng et al., 2019c; Geng et al., 2019b; Geng et al., 2019a; Geng et al., 2020a; Geng et al., 2020b; Li et al., 2019; Shi et al., 2020; Wang et al., 2020a; Wu et al., 2020; Geng et al., 2021). Programmability has often been a problem, but has recently been addressed in design tools (Sanaullah and Herbordt, 2018c; Sanaullah and Herbordt, 2018a; Sanaullah et al., 2018a; Herbordt, 2019) and middleware (Xiong et al., 2018b; Xiong et al., 2018a; Xiong et al., 2019; Xiong et al., 2020; Stern et al., 2018; Haghi et al., 2020c; Haghi et al., 2020a).

2.3.3 FPGA Acceleration of MD

Prior MD/FPGA studies not otherwise mentioned here include surveys, (Chiu et al., 2008; Sukhwani and Herbordt, 2009; Chiu and Herbordt, 2010b; Herbordt, 2013; Khan et al., 2013), integration (Gu et al., 2006c), RL datapath optimization (Gu et al., 2008), handling neighbor lists in RL (Chiu and Herbordt, 2009; Chiu and Herbordt, 2010a; Chiu et al., 2011), the bonded force (Xiong and Herbordt, 2017), parallel implementations (Pascoe et al., 2020), and complete FPGA integration (Yang et al., 2019b; Yang et al., 2019a).

2.4 Previous Work and Design in FPGA Acceleration of LR

2.4.1 Overview of Previous LR Design

Figure 2·2 depicts LR computation hardware that was previously under development.

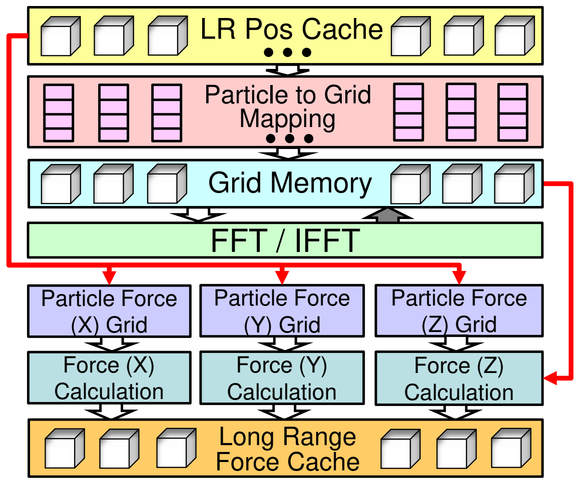

Figure 2·2: LR Evaluation Architecture Overview (Yang et al., 2019a)

Computation begins with the caching of position and charge data. Position and charge information is used when mapping changes to the grid and when calculating the forces exhibited on each particle. The caching of particle positions and charges allows a host to offload data to this accelerator and perform other tasks why LR computations are taking place. 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 is consumed, the FFT evaluation runs on the resulting grid (through each axis X, Y, and Z). 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.

2.4.2 3D FFT

The initial design of the FFT subsystem is based on previous work (Humphries et al., 2014) and performs calculations in parallel using vendor supplied FFT cores. This work was later extended to clusters and clouds of tightly coupled FPGAs (Sheng et al., 2014) and used for OpenCL case studies (Sanaullah and Herbordt, 2017; Yang et al., 2017; Sanaullah and Herbordt, 2018b). Recent work has created an efficient version of the 3D FFT for a small cluster of FPGAs and implemented in OpenCL (Stewart et al., 2020).

The FFT units are assigned to specific banks of the grid memory to ensure high throughput memory access. 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.

Figure 2·3 shows a block diagram of the logic used. The Crossbars work in conjunction with the RAMs to select the flow of data so as to affect transpose and reverse-transpose as needed. The Controller is a large state machine that drives all of the inputs to the RAMs, Crossbars, and FFT Pipelines. The FFT pipelines are 1D and supplied by the vendors with standard packages.

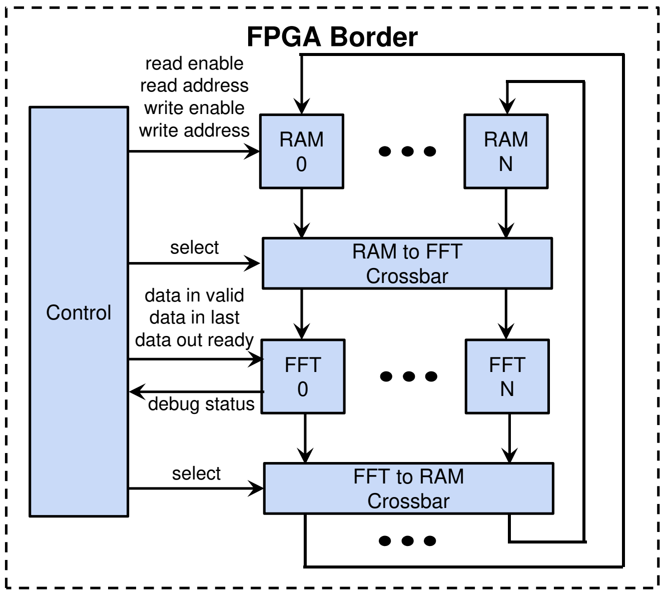

Figure 2·3: Block Diagram for 3D FFT Design (Humphries et al., 2014)

2.4.3 Particle to Grid Mapping

The third order basis functions in Equation 2.7 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. The following were used in (Gu and Herbordt, 2007). Previous work on particle-grid mapping can be found in (Sanaullah et al., 2016a; Sanaullah et al., 2016c;

Sanaullah et al., 2016b).

$$\begin{cases} \phi_0(oi) = -\frac{1}{2}oi^3 + oi^2 - \frac{1}{2}oi \\ \phi_1(oi) = \frac{3}{2}oi^3 - \frac{5}{2}oi^2 + 1 \\ \phi_2(oi) = -\frac{3}{2}oi^3 + 2oi^2 + \frac{1}{2}oi \\ \phi_3(oi) = \frac{1}{2}oi^3 - \frac{1}{2}oi^2 \end{cases} \tag{2.7}$$

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 2·4 shows the process of a single particle's influence on 64 neighbor cells and their mapping to a prior work's grid memory structure. Starting with a particle's initial position and charge (a), it mapped to its nearest neighboring 4 grid-point in each of the three dimensions (b), permuting the combinations of 1D mappings to 3D, a cube of 3D grid-points is created (c), and finally mapped to 64 address locations within 16 independent memory units.

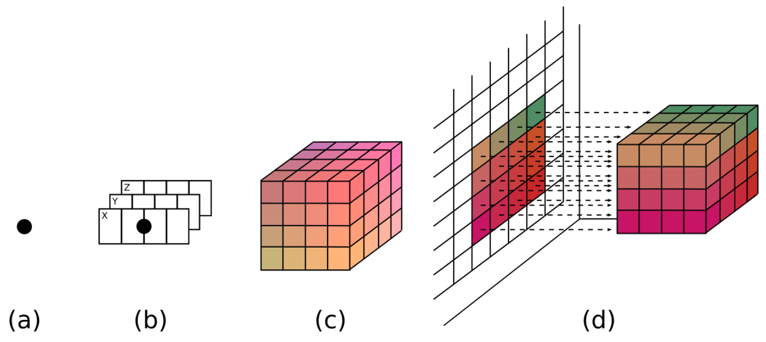

Figure 2·4: Previously Developed Particle-to-Grid-Memory Flow (Yang et al., 2019a)

2.4.4 Grid Memory Network

One issue with the particle-grid converter is that a large number of grid points must be accessed on every cycle; this requires both high bandwidth and highly parallel addressing logic. Fortunately, modern FPGAs, with their thousands of independent BRAMs, have just such capability. The interleaved memory design described in (VanCourt and Herbordt, 2006a) is one such example. Figure 2·5 motivates the need for this network.

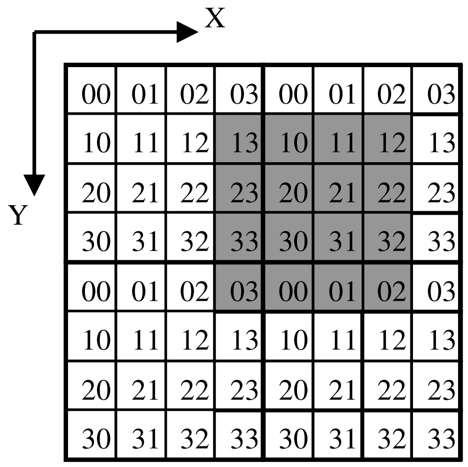

Figure 2·5: 2D Example Motivating an Alignment Network (Gu and Herbordt, 2007)

Given an address reference (x,y), the grid points within a 4x4 window, i.e. (x,y), (x,y+1), . . . , (x+3,y+3), must be accessed. Obviously, 16 independent memory accesses are required for each interleaved memory access. As shown in Figure 8, when grid points are stored in 16 separate banks marked from 00 to 33, any 4x4 access window contains exactly one grid points from each bank. The bank's index is either the same as that of the bank of the reference address, or the one greater than that in the X and/or Y dimension, respectively. The outputs from memory banks are shifted (with rotation) in both X and Y based on the reference address. 3D interleaving memory is analogous.

Chapter 3

Single-FPGA LR Design

Part of this thesis builds an improved and parameterized LR architecture using resources on a single FPGA. Figure 3·1 depicts a high-level block diagram of the new LR architecture.

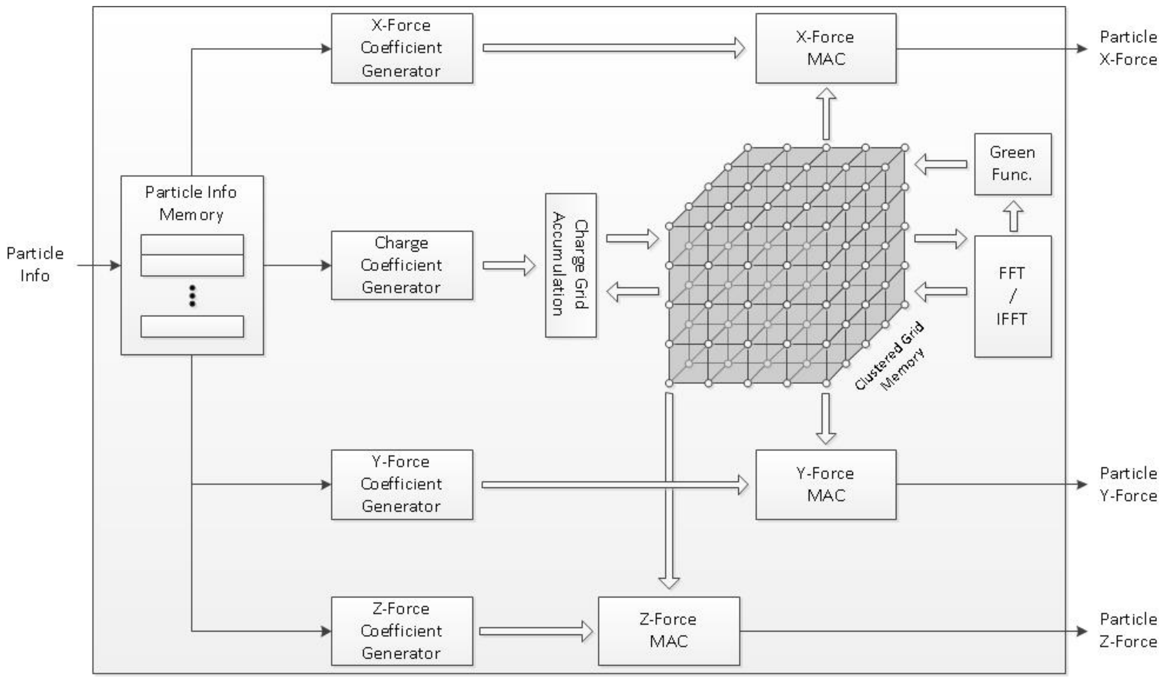

Figure 3·1: Enhanced Single-FPGA LR System Block Diagram

Particle position and charge information are loaded into a particle info memory (analogous to the particle position cache in Figure 2·2) through a ready-valid handshaking interface with a 'last' indicator to inform the system when the last of the particle information is transferred. As previously mentioned, this particle info memory is used in the charge mapping and force calculation phases of the LR computation. Expanding on (VanCourt and

Herbordt, 2006a), a 3D-interleaving of BRAMs is used to implement the grid memory. This 3D interleaving of BRAMs offers up some algorithmic performance enhancements, details of which can be found in section 3.3. Once mapped, the resulting charge-grid is transformed into a grid of Coulombic potentials by passing through a 3D FFT, passing through a Green's function immediately following the 3D FFT, and lastly passing through a 3D IFFT. The grid of potentials is then combined with the cached particle position and charge to calculate the three one-dimensional force vectors acting on each particle. The final force values are offloaded to the host using a separate ready-valid handshaking interface also equipped with a 'last' indicator to inform the host when the last particle's force vectors are transferred.

The LR design developed during this thesis leverages Hardware Description Language (HDL) code used to build the system in Figure 2·2. Upon review of the HDL, it was found that the Green's function implementation was incomplete. The new LR design completes that, adds the two ready-valid handshaking interfaces, implements 3D-interleaved grid memory, and implements FFT pipelines that complement the structure of this new grid memory. Only the coefficient generators and force multiply-accumulate (MAC) blocks were found to be reusable. Though reusable, they were enhanced to reduce the amount of FPGA resources they consume (see Sections 3.2 and 3.6).

The remainder of this chapter discusses the architectural details of the various submodules that comprise the single-FPGA LR design.

3.1 Particle Info Memory

LR computation begins with the loading of particle position and charge information into a memory. Information is transferred over a ready-valid interface equipped with a 'last' indicator to inform the system when the last of the information is transmitted by an off-chip host. The particle information transmitted includes the position of particle and its charge. A particle's position relative to neighboring grid-points is illustrated in Figure 3·2.

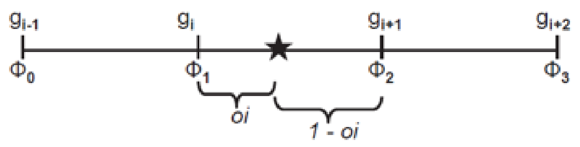

Figure 3·2: Particle Positioning (Sanaullah et al., 2016c)

The particle's true position is denoted by the $\star$ but can be broken up into a nearest-floor-grid-point, ϕ_1 and the offset from that nearest-floor-grid-point, *oi*. ϕ_1 is stored as an unsigned integer whose width is dependent on the number of grid-points in the given dimension. In this design offsets are stored in an unsigned Q27 fixed-point format and charge values are stored in a signed Q5.27 fixed-point format.

Since LR involves running per-timestep LR computations, the grid memory needs to be cleared prior to charge mapping. This grid clearing happens, grid-point by grid-point, as particle information is transferred from the host. This clearing mechanism imposes that the number of particles to be mapped be greater than or equal to the number of grid-points in the simulation. Though this is a reasonable assumption, future hardware could leverage the parallelized nature of grid memory access (see section 3.3) to clear the grid over fewer particle information transfers.

3.2 Charge Mapping

Charge mapping interpolates charge densities at grid-points using a particle's charge value (Sanaullah et al., 2016a). Charge density values are calculated with Equation 3.1.

$$\rho_g = \sum_p Q_p \phi(|x_g - x_p|)\phi(|y_g - y_p|)\phi(|z_g - z_p|) \tag{3.1}$$

These ϕ functions are a result of plugging in the definition of a particle's position relative to its nearest neighbors (see Figure 3·2) into Equation 2.6, resulting in the functions defined in Equation 2.7. Expanding across three dimensions, a single particle is mapped to 64 grid-points. The charge coefficient generator sub-module pipelines this spreading of charge across all 64 nearest-neighboring grid-points, utilizing the FPGA's floating point (FP) IP to perform the arithmetic operations implied by Equations 2.7 and 3.1.

The accumulation of charge densities interpolated from separate particles requires read-modify-write operations to be performed on data in the grid memory. This requirement, coupled with the pipelined nature of the charge mapping, introduces the possibility of data hazards. A read-after-write (RAW) hazard occurs if a grid-point to be updated with accumulated charge data is required to be read out for the mapping of a subsequent particle. Currently, the hardware stalls the pipeline to avoid these hazards. If particle charge and position information were to be sorted prior to being cached, such that particles mapping to overlapping sets of grid-points were spaced out to avoid these RAW hazards, this stalling could be removed. Alternatively, additional hardware could be implemented to provide a means of data forwarding where possible. However, to simplify the complexity and verification of this design in the face of time constraints, it was decided to move forward with pipeline stalling. It is well known that stalling is detrimental to the pipeline performance. For this design to be viable, stalling should be replaced with a better hazard resolution strategy.

As previously mentioned, the coefficient generator block is implemented through partial reuse of pre-existing HDL files. The sub-module design is enhanced to reduce the number of FPGA FP resources consumed based on the coefficients of the functions in Equation 2.7 by:

1. Driving pipeline stage outputs to zero when portions of the algebraic equation call for multiplications by zero.

2. Replacing FP IP with pipeline register stages when portions of the algebraic equation call for additions of zero.

3. Replacing FP IP with pipeline register stages when portions of the algebraic equation call for multiplications by one.

This is performed using SystemVerilog (SV) parameters and generate statements (SVLRM, 2018). Utilizing SV parameters and generate statements allows for easy hardware implementation updates if the basis functions were to change. It also allows this block to be reused when mapping particle forces (see section 3.6).

A few notes for anyone that plans to reuse/enhance the charge mapping hardware. Work was performed to verify the hardware against OpenMM models (OpenMM, 2020). It was found that the OpenMM modeling utilized the the following third-order basis functions (when configured for third-order polynomial interpolation):

$$\begin{cases} \phi_0(oi) = -\frac{1}{6}oi^3 + \frac{1}{2}oi^2 - \frac{1}{2}oi + \frac{1}{6} \\ \phi_1(oi) = \frac{1}{2}oi^3 - oi^2 + 4 \\ \phi_2(oi) = -\frac{1}{2}oi^3 + \frac{1}{2}oi^2 + \frac{1}{2}oi + \frac{1}{6} \\ \phi_3(oi) = \frac{1}{6}oi^3 \end{cases} \tag{3.2}$$

This change in interpolation function and therefore all the hardware built around it, simply required changes to the SV parameters defined in the charge mapping module. The generate statements process the parameter changes to make up-front hardware optimizations.

3.3 Grid Memory

The storage of grid-point data affects the operation of all phases of LR computation, from the number of grid-points that can be mapped per clock-cycle (assuming RAW hazards are taken care of), to the number of FFT pipelines that can run in parallel, to the number of grid-points that can be read out per clock cycle when performing the force interpolations. Previously explored architectures lent themselves well to performing the 3D FFT, but often required transposing the stored grid-point values between one or more 1D FFT phases to maximize the the number of parallel 1D FFT computations (Humphries et al., 2014; Sanaullah and Herbordt, 2017; Sanaullah and Herbordt, 2018b). Even an architecture comprised of X_G by Y_G blocks, each with a depth of Z_G, where X_G, Y_G, and Z_G are the number of grid-points in the X, Y, and Z dimensions, respectively, requires transposing stored data. With that in mind, the clustered grid memory is an interleaving of FPGA block RAMs designed to:

1. Store 64-bits of data for each grid-point in the system. 64 bits are needed to store two 32-bit, single-precision, floating-point values for the real and complex components of grid-point data.

2. Support the mapping of one particle to its 64 nearest-neighboring grid-points in 3D per clock cycle.

3. Avoid the need for data reordering prior to performing 1D FFTs over the grid-points of charge data.

Meeting requirement 1 is trivial. To meet requirement 1 imposes that the memory architecture have at least 64 two-port (one read, one write) memories. Meeting requirement 3 imposes that the same mechanism used to access grid-points sequentially placed along any one of the three dimensions be agnostic to the direction of that dimension. Meeting requirements 2 and 3 is achieved by expanding on (VanCourt and Herbordt, 2006a) making use of low-order address interleaving.

3.3.1 Clustered Grid Memory

The clustered grid memory is comprised of 64 memory units (neighbors), one unit per a particle's nearest-neighbor in 3D, with each unit possibly being comprised of multiple FPGA BRAMs. The ordered triplet used to specify the location of the grid-point in 3D space is the address for the data at that grid-point. Partitioning the grid memory into 4x4x4 neighborhoods, the location of a grid-point can be thought of in terms of a neighborhood ID and a neighbor ID, where the neighbor ID is used to choose among the 64 memory units and the neighborhood ID is used to index into each memory unit. The lower two bits of each coordinate concatenated form the neighbor ID, while upper address bits of each coordinate concatenated for the neighborhood ID. The following is an example of how the ordered triplet of a grid-point in a 32x32x32 grid is decomposed into its cluster ID and neighbor ID.

$$\begin{aligned}
&= 11, 23, 8 \\
&= (01011_2, 10111_2, 01000_2) \\
&= (010_2, 101_2, 010_2).(11_2, 11_2, 00_2) \\
&= (2, 5, 2).(3, 3, 0)
\end{aligned}$$

In a 32x32x32 grid, grid-point (11,23,8) refers to neighbor (3,3,0) inside of neighborhood (2,5,2).

The clustered grid memory presents 64 two-port access channels in the form of a cluster to the rest of the LR system. Figure 3·3 depicts the access cluster access in the context of the grid memory. The figure depicts how the access cluster moves about to access neighbors from neighborhoods, thereby accessing grid-points from the grid. Neighbor IDs range from 0-64 and are associated with neighborhoods based on their 3D position. Neighbors accessed by the access cluster appear as lighter shades of gray or orange. From (a) to (b) to (c), the access cluster moves along the *Y* dimension. From (c) to (d) the access cluster moves along the *Z* dimension.

Notice that each neighbor ID appears at most once inside of an access cluster. Notice also that the access cluster is not required to be neighborhood-aligned, that is, the access cluster can access some neighbors from one neighborhood while accessing neighbors from other neighborhoods. This lack of alignment requirement means that any neighbor ID

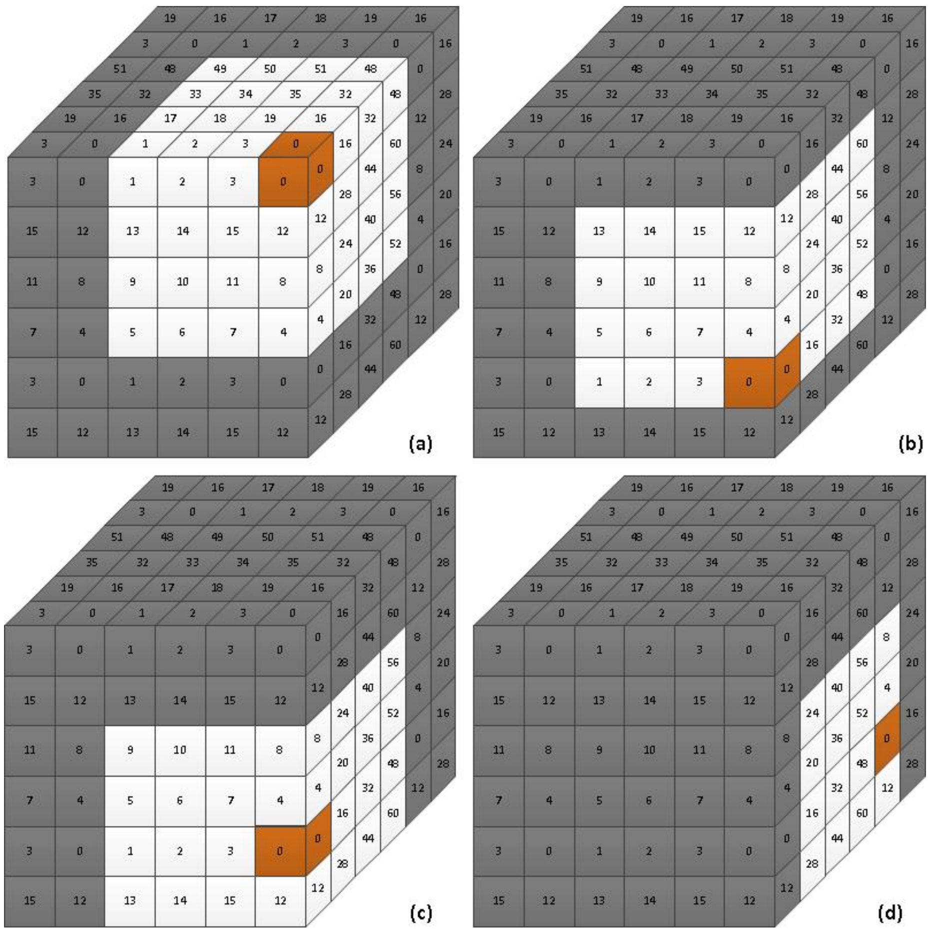

Figure 3·3: Memory Access Patterns for Charge Mapping in 3D

can appear on any of the ports in the access cluster. This is evidenced by the location of neighbor 0 (potentially from different neighborhoods) relative to a port within the access cluster as the access cluster moves around in 3D space. Achieving this flexibility requires all-access-cluster-port-to-all-neighbor connectivity. Originally, this all-to-all connectivity was performed in a single clock cycle and proved to be the critical path when attempting to target the system for a 100MHz operating frequency. To improve performance, this all-to-all connectivity was pipelined. Figure 3·4 depicts a three-stage, toroidal, shift MUX used to pipeline the all-to-all connectivity from access cluster ports to neighbor memories.

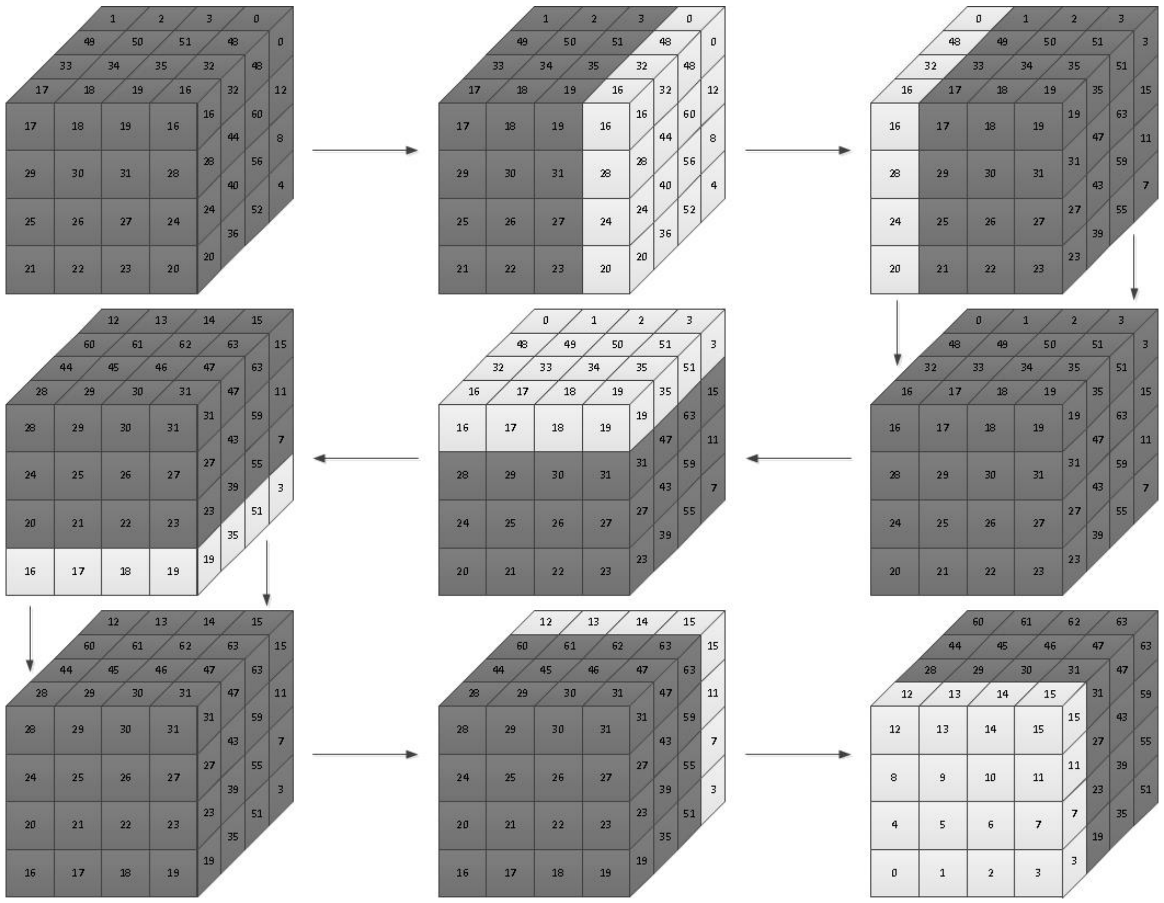

Figure 3·4: Three-Stage Toroidal-Shift MUXing Pipeline for Memory Access Alignment

The figure depicts how an non-neighborhood-aligned access is aligned in each dimension. The top row of the figure depicts alignment in the X dimension while the second and third rows depict alignment along the Y and Z dimensions, respectively. Once control information is aligned, the data read from the neighbors is re-unaligned before arriving back at the access ports so the rest of the system sees the data on the same ports it originally requested the access over.

Alignment in each dimension is performed in one clock cycle. Alignment in one dimension involves looking at the appropriate 1D component of the neighbor ID and comparing it to the appropriate 1D component of the access port ID. If these components match, no realignment is required. If these components do not match, a toroidal shift occurs such that the 1D ID components will match following the next clock edge.

The clustered grid memory was parameterized to handle various 3D grid sizes. Grids are not required to be cubic, but each dimension of the grid is required to be a power of 2. To support this flexibility the toroidal shift MUXing was performed using one-hot MUXing. This one-hot MUXing may not prove to be the best performing MUXing architecture, but accommodating various grid sizes would require HDL file modification rather than the overriding of SV parameters.

3.4 FFT/IFFT

Once the mapping of all particle charges is complete, the resultant charge grid is then transformed into a grid of Coulombic potential energies by:

1. Converting the charge data from the spatial domain to the Fourier domain by running it through a 3D FFT.
2. Convolving the result of the 3D FFT with the 3D FFT of a Green's function.
3. Running a 3D IFFT over the results of the convolution.

The 3D FFT (and IFFT) is decomposed into sequential 1D FFTs (IFFTs), one in each dimension. The LR design makes use of FPGA DSP resources that stream in FFT inputs into a pipeline and some number of clock cycles later stream out the FFT results sequentially. Preservation of 1D FFT (and IFFT) input data is not required, so the clustered grid memory is used to temporarily store the results of each 1D FFT (and IFFT). Before the results of the third 1D FFT are written back to the clustered grid memory, they are first convolved with the 3D FFT of a Green's function. This convolution equates to a simple multiplication in the Fourier domain and therefore leverages additional FPGA FP resources.

With the ability to read and write 64 clustered grid memory addresses simultaneously, implementing 64 FFT pipelines maximizes data throughput and reduces the overall 1D FFT execution time. Unfortunately, the access structure of the clustered grid memory supports accessing a 3D, 4x4x4 set of grid-points where each port is required to access a unique neighbor. This is not conducive to the FPGA DSP resources, which require FFT points to be inputted in a specific order. This order is configurable when building the hardware (Intel, 2017) and is chosen to be natural. To work around this limitation, 1D FFTs (and IFFTs) are performed by staggering 2D slices of the access cluster and mapping these access-cluster slices to neighbor memories from neighborhoods in different districts, where a district is an array of neighborhoods in a given dimension. Figure 3·5 illustrates how this staggering of access cluster slices works on a 4x4x4 grid with a 2x2x2 access cluster.

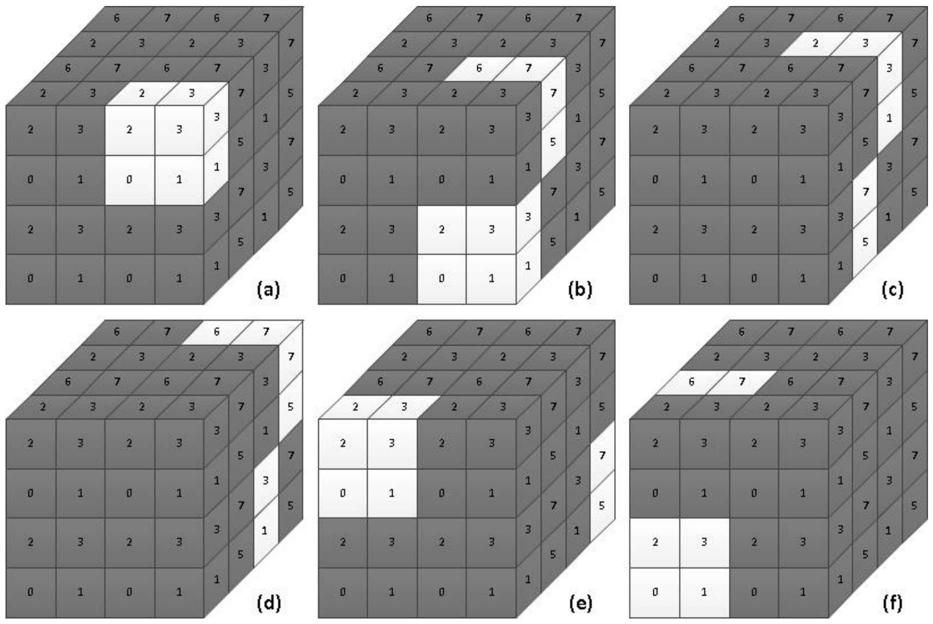

Figure 3·5: Memory Access Cluster Slice Staggering for Optimum FFT Throughput

In this figure the x-axis trends positive in a westerly direction, the y-axis trends positive in a southerly direction, and the z-axis trends positive moving into the paper (this was done intentionally for illustrative purposes). The FFT access scheme moves from (a) to (f). In the first clock cycle (a), access-cluster ports 0-3 are mapped to neighbors-memories 0-3 of neighborhood (0,0,0). In the second clock cycle (b), ports 0-3 are mapped to neighbors 4-7 of neighborhood (0,0,0) while ports 4-7 are mapped to neighbors 0-3 of neighborhood (0,1,0). When a 2D access slice is accessing neighbor 4-7 of the last neighborhood in of a district (d and e), it will access neighbors 0-3 of the first neighborhood of a previously unaccessed district (e and f). This pattern continues until neighbors 4-7 of the last neighborhood of the last district are accessed.

Originally it was planned to configure the FFT resources to be bidirectional, but simulations of the IFFT revealed that the ordering of IFFT outputs were not consistent with the documentation. To workaround this setback, IFFTs are implemented with forward-direction FFT resources (Lyons, 2015). Doing so, requires:

1. Modifications to the boundary checking conditions of the access-cluster-slice staggering. This modification is only required in the inverse FFT direction only.

2. Dividing the outputs of the FFT pipeline. Since the IP uses floating-point values and any dimension of the grid is a power of two, this division is performed by a subtraction of the exponential bits in the floating-point value.

Later on, it was determined that this subtraction could be removed due to compensations made inside the Green's function (see section 3.5).

3.5 Green's Function

The Green's function is dependent on the position of the grid-point, the true volume of the grid, the order of the basis function used to map the particle charges, and a convergence parameter (Procacci, 2009). Thus, the values of the Green's function at each grid-point can be computed offline and programmed into ROM as the rest of the FPGA logic is being programmed. This Green's ROM architecture is similar to the clustered grid memory with the one exception: the Green's function value at each grid-point is only 32 bits wide. Since the Green's ROM is only ever accessed during the 1D FFT in the Z dimension using the staggering of 2D access cluster slices, alignment in the X and Y dimensions is guaranteed. This guarantee does allow for the reduction toroidal shift MUX pipeline stages from 6 to 2, but for developmental expedience, this reduction was not implemented.

When comparisons of this accelerator were compared against OpenMM modeling it was found that since:

1. The computation of the Green's function requires multiplication by $X_G Y_G Z_G$.
2. 3D IFFTs using FFT require a division by $X_G Y_G Z_G$.
3. FFT/IFFTs are linear functions.

the computation of Green's ROM values could omit the multiplication step allowing for the removal of the division steps of IFFT using FFT (OpenMM, 2020).

3.6 Force Computation

Computing the forces acting on each particle in each dimension uses the following three equations:

$$\vec{\mathbf{F}_{\mathbf{p},\mathbf{x}}} = \sum_g \varphi_g \partial\phi(|x_g - x_p|)\phi(|y_g - y_p|)\phi(|z_g - z_p|) \tag{3.3}$$

$$\vec{\mathbf{F}_{\mathbf{p},\mathbf{y}}} = \sum_g \varphi_g \phi(|x_g - x_p|)\partial\phi(|y_g - y_p|)\phi(|z_g - z_p|) \tag{3.4}$$

$$\vec{\mathbf{F}_{\mathbf{p},\mathbf{z}}} = \sum_g \varphi_g \phi(|x_g - x_p|)\phi(|y_g - y_p|)\partial\phi(|z_g - z_p|) \tag{3.5}$$

where p refers to a particle, g refers to a grid-point, and φ_g refers to the subset of electrostatic potential grid-points that map to the particle in question. For completeness, the derivative if *phi* is listed below:

$$\begin{cases} \frac{d\phi_0}{doi} = -\frac{3}{2}oi^2 + 2oi - \frac{1}{2} \\ \frac{d\phi_1}{doi} = \frac{9}{2}oi^2 - 5oi + 1 \\ \frac{d\phi_2}{doi} = -\frac{9}{2}oi^2 + 4oi + \frac{1}{2} \\ \frac{d\phi_3}{doi} = \frac{3}{2}oi^2 - oi \end{cases} \tag{3.6}$$

With that, force computation can be broken into two stages:

1. Coefficient generation - mapping the particle to the grid using the partial derivatives of the mapping function in each dimension.

2. MAC - Multiplying the results of the mapping function by the subset of electrostatic grid-points that the particle maps to and summing the products across all grid-points in that subset.

A couple notes for those that plan to reuse/enhance the force computation sub-modules: When the interpolation function was changed to verify the design against OpenMM modeling, the the derivative if *phi* became:

$$\begin{cases} \frac{d\phi_0}{doi} = -\frac{1}{2}oi^2 + oi - \frac{1}{2} \\ \frac{d\phi_1}{doi} = \frac{3}{2}oi^2 - 2oi \\ \frac{d\phi_2}{doi} = -\frac{3}{2}oi^2 + oi + \frac{1}{2} \\ \frac{d\phi_3}{doi} = \frac{1}{2}oi^2 \end{cases} \tag{3.7}$$

Again, that hardware change was made made painlessly by changing parameter settings and having SV generate statements take care of the rest.

3.6.1 Coefficient Generation

Coefficient generation involves the same mapping process that mapped particle charges to the grid, but uses the derivative of the functions in Equation 2.7 in one of the three dimensions. To this effect, the same coefficient generator HDL used in charge mapping is used to generate the Force coefficients in each of the three dimensions. The coefficient generator is implemented 3 additional times, but the configuration of each sub-module instance is fixed at implementation by choosing between the coefficients of 2.7 and 3.6 based on the direction of the force to be calculated.

It should be noted that FPGA DSP resources could be saved by implementing only 3 coefficient generators with one of them being used for both charge mapping and force coefficient generation. Doing so however, would require a different coefficient generator architecture that is run-time programmable.

3.6.2 MAC

The MAC stage of force computation involves taking the force coefficients generated and multiplying them by the electrostatic potentials stored in the clustered grid memory on a grid-point-by-grid-point basis and then summing across the 64 products. Sequencer logic was implemented such that the force coefficients and the associated read data from the clustered grid memory arrive at the inputs of a bank of floating-point multipliers within the same clock cycle. The products are then fed into a tree of floating-point adders to sum across all the grid-points.

Once force results are valid they are presented on the particle-force ready-valid interface and remain there until the host is ready to accept the force information, stalling the entire force computation pipeline when necessary. This interface also includes a ‘last’ signal to qualify the final particle-force value to be transferred.

3.7 Hardware Development Summary

Building from previous work, this enhanced LR design performs charge mapping at one particle per clock cycle, does not require FFT staging transpositions, and also maximizes FFT throughput given the constraints imposed by the first two features. The ready-valid interfaces allow for easy integration of this accelerator into a complete MD system. The hierarchical partitioning of the LR design into the aforementioned sub-modules allows for more detailed design introspection. As seen in the next chapter, resource utilization reporting is easily partitioned according to sub-module. The flexible nature of the parameterized design has already proved useful as it allowed for a quick change of interpolation functions for functional verification. As will be seen in the next chapter, this flexibility results in quick design turn around time, allowing for the exploration of more design configurations in parallel, or at the very least requiring little to no changes to the HDL files. These design configurations include, but are not limited to, grid size and particle ensemble size.

Chapter 4

Single-FPGA LR Evaluation

This chapter evaluates single-FPGA LR designs and is broken up into three sections. Section 4.1 provides details of LR hardware implementations when mapped to FPGA devices and section 4.2 details the computational run-times of the designs. Section 4.3 discusses the results, while drawing comparisons to related accelerator hardware where possible and offering suggestions for improvement.

4.1 Implementation Details

A total for four single-FPGA LR designs were implemented. Table 4.1 enumerates each of the configurations, highlighting the major architectural differences in each implementation.

Cfg. ID	Grid Size	Particle Size	FFT FP Type	Clock Freq.
0	16x16x16	4,096	Hard	211.24MHz
1	32x32x32	32,768	Hard	187.83MHz
2	16x16x16	4,096	Soft	239.29MHz
3	32x32x32	32,768	Soft	191.32MHz

Table 4.1: Configuration Summary of Single-FPGA LR Implementations

Two separately sized designs were each implemented with and without taking advantage of the hardened floating point resources. Attempts were made to implement a 64x64x64 design with support for 64^3 particles, but the tool seemed to hang trying to synthesize the design, sitting at 33% of the way through synthesis after 3 days. These issues were reported to Intel, but no responses were received upon submittal of this thesis.

Figure 4·1 captures the on-device resource utilization of each design implementation.

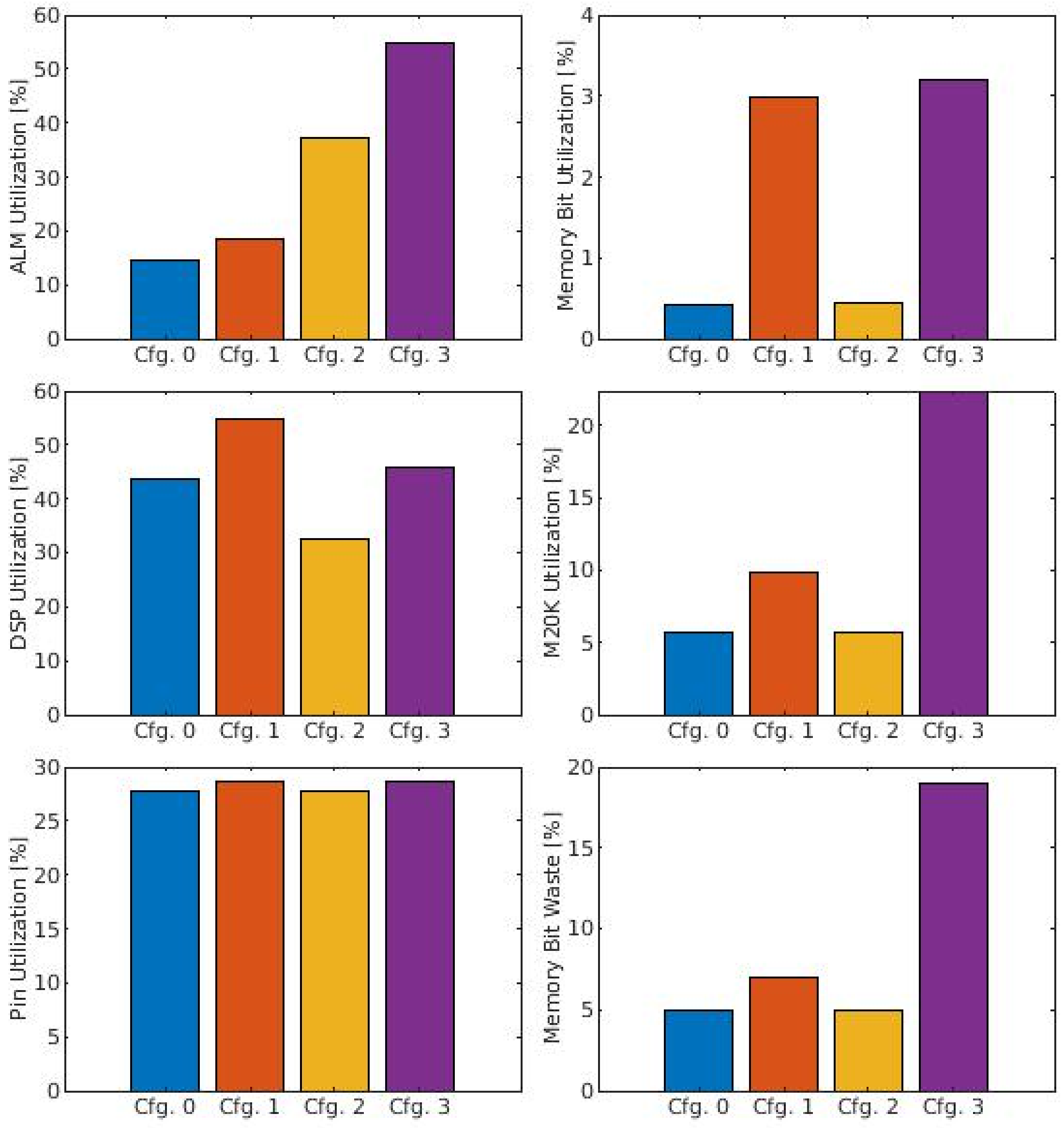

Figure 4·1: Device Resource Consumption per Single-FPGA LR Implementation

The utilization of all resources increases with the size of the grid and particle ensemble. The increase in ALM, pin, and memory utilization is easily implied purely from the increase in the number of grid-points and particles. The increase in DSP utilization is a result of the reconfiguration of the maximum FFT length for the 64 pipelines. The choice in implementation of floating point blocks within the FFT pipelines results in the trade-off between DSPs and a combination ALMs and M20Ks. Memory bit utilization refers to the number of memory bits that are functionally used in LR computation. Most functionally used memory bits are physically implemented using 512x40 BRAMs (M20Ks). However, not all bits of implemented M20Ks are functionally used, thereby wasting memory bits on the device. As one would expect, the memory bit utilization, M20K utilization, and memory bit waste increase with the size of the design. When preventing the use of hardened floating point units inside the FFT pipelines, the increase in M20K utilization nearly triples as the design size increases, but is mostly wasted.

Considering the number of resources each design consumes, Figure 4·2 breaks down the actual number of resources utilized by the major portions of the design. The particle info memory, the clustered grid memory, and FFT pipelines are largely dependent on the size of the grid and the number of particles in the system. The resource consumption increase in the FFT pipelines can be traced back to the configuration of the IP. In Configurations 0 and 2, the FFT IP is configured to support a maximum FFT length of 16 points, whereas Configurations 1 and 3 support a maximum of 32. The control sequencer also depends on the grid size a particle count, but to a much lesser extent. Additional ALMs are consumed when the the number of particles and grid-points cross power-of-two boundaries, requiring additional address bits to be generated when storing/retrieving data to/from the particle info memory, grid memory, or Green's ROM. The same scan be said about the ALM utilization of the Green's ROM and the clustered grid memory: ALM usage will increase as the address bits used to access grid-points increase.

The amount of resources consumed by the charge coefficient generator, charge accumulation, convolution, force coefficient generators, and force trees depend upon the number of 3D nearest-neighbors a particle has and should therefore be constant as the number of grid-points and particles increase. Not readily implied by the graphs in the figure are the

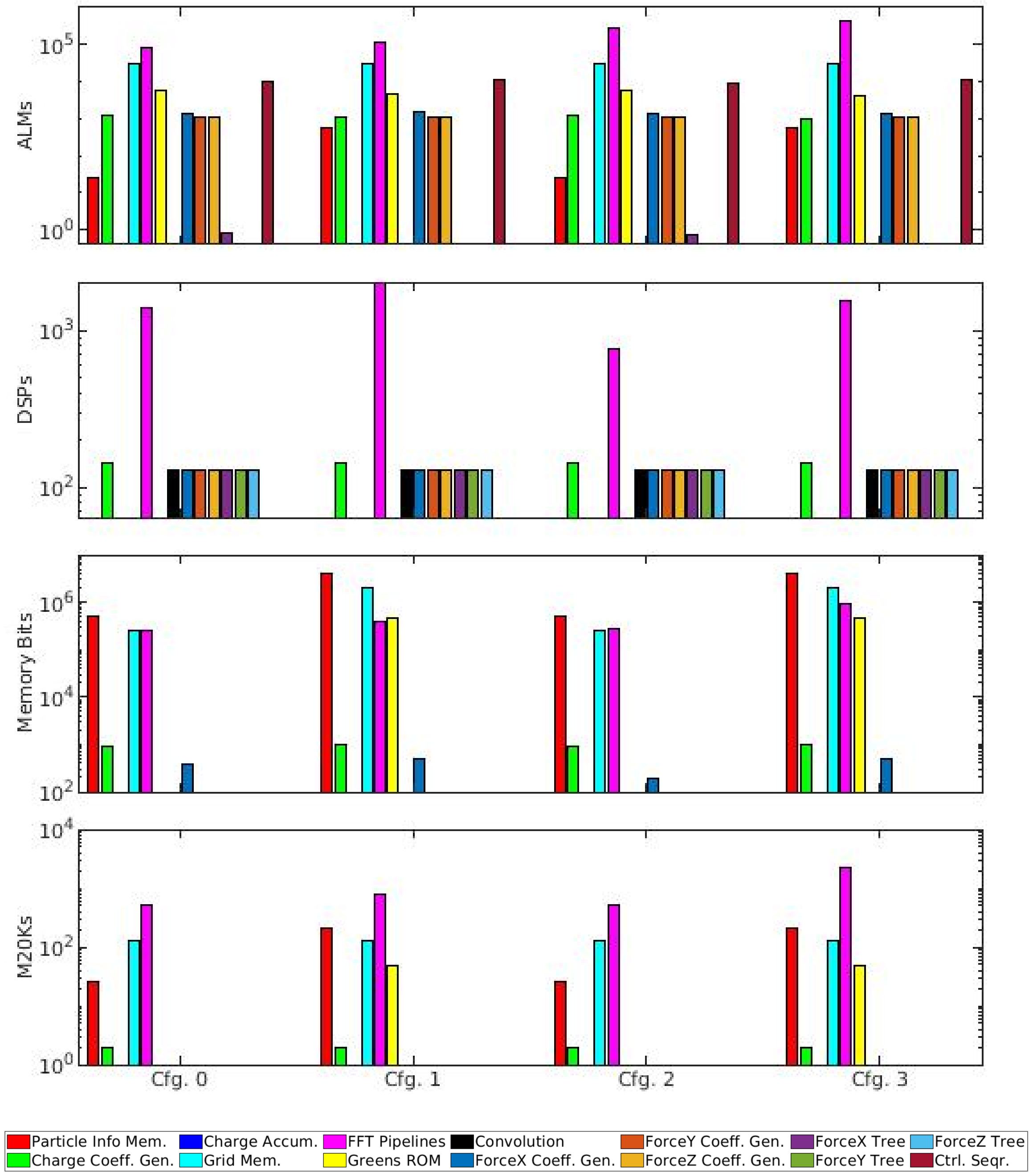

Figure 4·2: Per-Implementation LR Design Resource Consumption Breakdown

differences in the resource consumption of these modules as the size of the grid and particle ensemble increases. These noted increases can probably be attributed to the logic optimization of the FPGA implementation software. Similar optimizations can be observed when reviewing the resource consumption of the force coefficient generators in each configuration. These modules are nearly identical, with the only differences being the one dimension that uses the coefficients of Equation 3.6, but most of the resources are consumed by the force coefficient generator in the X-dimension.

On the surface, one would think that the resources consumed by the Green's ROM would be directly dependent on the size of the grid. However, in the 16x16x16 configurations, the Green's ROM is implemented with ALMs only. In the 32x32x32 configurations the Green's ROM is implemented with a mix of ALMs and M20Ks, but not to the same extent as the Clustered Grid memory. This may be attributed to the read-only nature of Green's ROM and the potential for ROM-bit reuse across addressable data values.

The amount of memory bit waste is attributed largely to the FFT pipelines. Across all configurations, most of the memory bits that are implemented and serve a true functional purpose are found in the particle memory, clustered grid memory, and in the 32x32x32 configurations, the Green's ROM. However, more than 50% of the M20Ks consumed reside in the FFT pipelines, meaning that the FFT pipelines only functionally make use of a fraction of each M20K they consume.

Using the information above and assuming:

1. $X_G = Y_G = Z_G = gsize_{1D}$
2. $X_G Y_G Z_G = gsize_{3D} = n_{particles}$
3. The resource consumption of the coefficient generators, accumulators, FFT pipelines, convolution, and force trees remains relatively constant.
4. The Green's ROM consumes half the amount of resources as the clustered grid memory.
5. The growth in ALM consumption is negligible as $gsize_{1D}$ increases.

6. FFT pipelines make use of hardened FP resources.

the main concern in resource consumption is with the IO pins and block memory bits. IO pin consumption as a function $gsize_{1D}$ is expressed below:

$$pins = 3\lceil \log_2(gsize_{1D}) \rceil + 3b_o + 4b_{qf} + 2\lceil \log_2(gsize_{3D}) \rceil + 7 \tag{4.1}$$

where b_o is the number of bits used to specify a particles offset in one dimension, b_{qf} is the number of bits used to specify a charge or 1D force value. The additional 7 pins account for the design's clock and two ready-valid-last interfaces. The particle info memory's block memory bit consumption can be calculated with the following equation:

$$pmem_{bits} = (3\lceil \log_2(gsize_{1D}) \rceil + 3b_o + b_{qf})n_{particles} \tag{4.2}$$

The consumption of block memory bits of the clustered grid memory and the Green's ROM can boils down to 96 bits per grid-point. Figure 4·3 plots IO pin and memory bit consumption as functions of $gsize_{1D}$. Under these assumptions, the maximum cubic, power-of-2 grid size a single Stratix-10 FPGA can support is a 64x64x64. The maximum number of block memory bits contained in any Stratix-10 device is 240,046,080. A 101x101x101 grid consumes 237,249,470 and a 102x102x102 grid consumes 244,358,080 and that is assuming block RAM bits are not wasted during mapping. Larger designs come nowhere near consuming all the 912 IOs available on the device. Considering that the Green's ROM consumes nowhere near half the resources the clustered grid memory does, due to design optimization, it may be possible that a 128x128x128 design could fit on a single-FPGA. However, attempts at even a 64x64x64 design supporting 64^3 particles have seemingly generated tool issues.

The replacement of hardened floating point units inside the FFT pipelines with ALMs, did increase the operational frequency of the design for both the 16x16x16 and 32x32x32 grid sizes at the cost of ALM and M20K utilization. With this performance trade-off, the

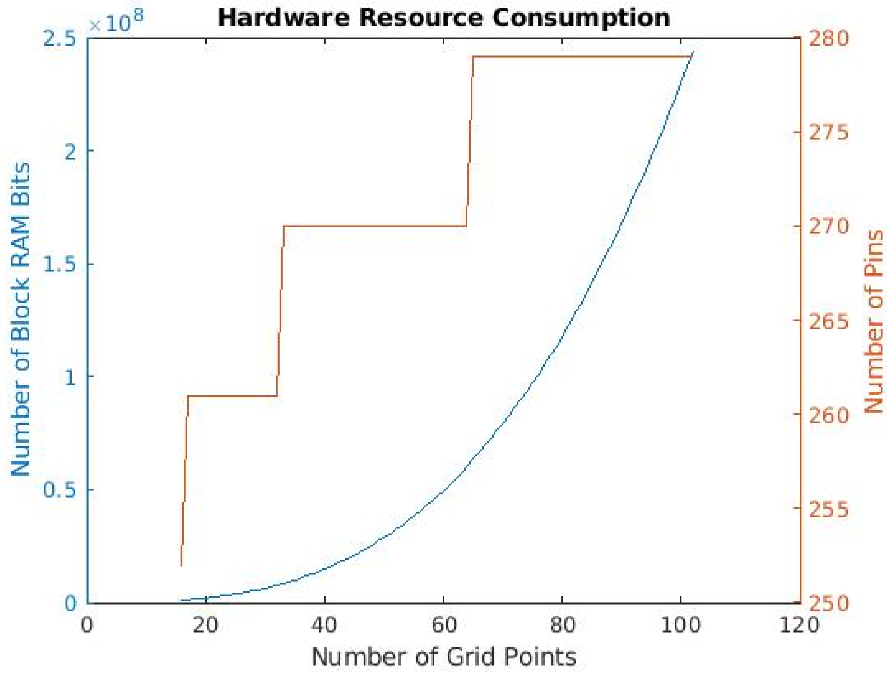

Figure 4·3: LR Design Hardware Resource Consumption vs. $gsize_{1D}$

increase in ALM consumption becomes a concern as the size of the grid increases. This is due largely in part to the dependence of ALM consumption on the number of points the FFT pipelines need to operate over. Unfortunately, ALM consumption cannot be easily captured as a function of grid size. Attempts were made to implement a 64x64x64 grid based LR design, but failed to synthesize. Projecting the reported ALM consumption of Configuration 2 and Configuration 3 outwards, a 64x64x64 configuration, if synthesized, may run into routing issues. If it can be routed, it can be assumed that the operating frequency would drop further.

4.2 Computation Details

In this section the computation time is first characterized into a series of equations derived from the architecture of the system. These equations are then used along with the intrinsic properties of the hardware and the results from 4.1 to calculate the wall clock time of the computation.

4.2.1 System Equations

Overall computation time is the sum of the run-time of all the phases of LR, that is, charge mapping all particles, five 1D FFTs, one 1D FFT coupled with convolution of the Green's function, and force computation of all particles:

$$t_{LR} = t_{CMAll} + 2t_{FFT1D_X} + 2t_{FFT1D_Y} + t_{FFT1D_Z} + t_{FFT1D_ZNG} + t_{FCAll} \tag{4.3}$$

Due to its pipelined architecture, the run-time of charge mapping particles is governed by the following equation:

$$t_{CMAll} = n_{particles} + d_{CM} - 1 \tag{4.4}$$

where d_{CM} is the latency of the charge mapping pipeline. This assumes that no pipeline stalls occur, which is not the case for the hardware as presently constructed, but can be assumed here as viable methods to remove pipeline stalls when charge mapping do exist. The latency of the charge mapping pipeline is further broken down into the latencies of one fixed-point-to-floating-point converter, 5 floating-point multipliers, 3 floating-point adders, the readback latency of the particle memory, and the latency to store values into the clustered grid memory:

$$d_{CM} = 1 + d_{ToFP} + 5d_{FPMul} + 3d_{FPAdd} + 5 \tag{4.5}$$

The staggered access cluster slicing during 1D FFTs allows for 64 FFT pipelines to run simultaneously once all access cluster slices are operational, resulting in the following run-time equations:

$$t_{FFT1D_X} = \frac{X_G Y_G Z_G}{NNN3D} + d_{FFT1D_X} + NNN1D - 1 - 4 \tag{4.6}$$

$$t_{FFT1D_Y} = \frac{X_G Y_G Z_G}{NNN3D} + d_{FFT1D_Y} + NNN1D - 1 - 4 \tag{4.7}$$

$$t_{FFT1D_Z} = \frac{X_G Y_G Z_G}{NNN3D} + d_{FFT1D_Z} + NNN1D - 1 - 4 \tag{4.8}$$

where X_G, Y_G, and Z_G are the number of grid-points in the X,Y, and Z dimensions, respectively, $NNN1D$ is the bumber of nearest-neighbors in 1D, $NNN3D$ is the number of nearest-neighbors in 3D, and d_{FFT1D} is the latency of performing the 1D FFT on one grid district. In this equation $NNN1D-1$ is the latency of filling the pipeline. The latency of running a 1D FFT on one district is governed by the following equations:

$$d_{FFT1D_X} = 7 + X_G - 1 + d_{FFTIP_X} + 5 \tag{4.9}$$

$$d_{FFT1D_Y} = 7 + Y_G - 1 + d_{FFTIP_Y} + 5 \tag{4.10}$$

$$d_{FFT1D_Z} = 7 + Z_G - 1 + d_{FFTIP_Z} + 5 \tag{4.11}$$

where d_{FFTIP_X}, d_{FFTIP_Y}, and d_{FFTIP_Z} are the latencies of the FFT computations in the X, Y, and Z dimensions, respectively. The 7 clock cycles up front account for the read-back latency of the clustered grid memory and the 5 clock cycles at the end account for the storage of the FFT result in the clustered grid memory.

The only difference between t_{FFT1D_Z}, which accounts for the IFFT in the Z dimension and $t_{FFT1D_Z NG}$, which accounts for the 1D FFT in the Z dimension along with the convolution, is latency of the floating point multiplication that takes place before the write-back to the clustered grid memory:

$$d_{FFT1D_Z NG} = d_{FFT1D_Z} + d_{FPMul} \tag{4.12}$$

Due to its pipelined architecture, the run-time of the force computation phase is governed by the following equation:

$$t_{FCAll} = n_{particles} + d_{FC} - 1 \tag{4.13}$$

where d_{FC} is the latency of the force computation pipeline. The latency of the force computation pipeline is further broken down into the latencies of one fixed-point-to-floating-point converter, 6 floating-point multipliers, 8 floating-point adders, the readback latency of the particle memory:

$$d_{FC} = 1 + d_{ToFP} + 6d_{FPMul} + 8d_{FPAdd} \tag{4.14}$$

It should be noted that this latency equation assumes that the host is always ready to accept valid data from the accelerator when presented for transfer.

4.2.2 Computation run-time

The latency of all floating-point IP used is dependent on its configuration (Intel, 2020a). The same can be said for the FFT IP (Intel, 2017). Substituting those values into the equations above and applying the achievable clock frequencies of each design, Figure 4·4 breaks down the wall-clock run-time of a single LR iteration according to Equation 4.3. The run-times for a single LR iteration for the four configurations are 43.63μs, 371.89μs, 38.49μs, and 364.20μs, respectively. Recalling that one timestep simulates roughly 2fs of reality, these designs can simulate 3.96μs, 464.65ns, 4.49μs, and 474.46ns of reality per day for their respective particle ensembles.

4.3 Evaluation Discussion

Drawing comparisons to previous work where possible, it is noted that the 3D FFT run-times of Configuration 1 and 2 are 10.2242μs and 10.2713μs, respectively. These results beat out the best single-FPGA, 32^3 FFT, run-time reported by (Humphries et al., 2014). This improvement is largely due in part to the difference in the number of FFT pipelines and the difference in FPGA technology nodes. This single-FPGA LR accelerator's 3D FFT run-time is slightly less than double the run-time proposed in (Sheng et al., 2014) and less than three times the run-time of the original Anton network (Young et al., 2009), both of

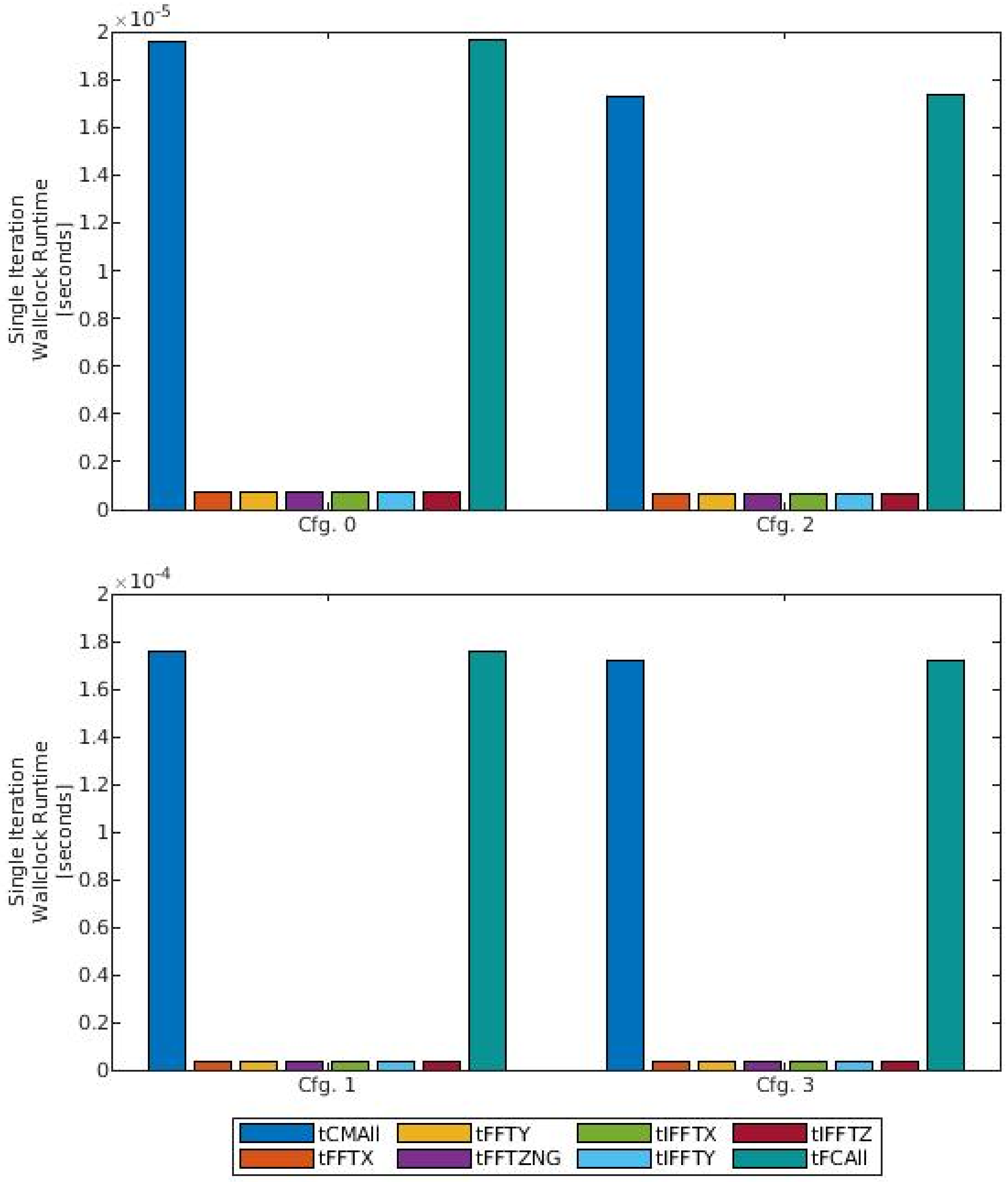

Figure 4·4: LR Design Performance Breakdown

which utilize multi-FPGA clusters.

Analyzing the results across the four configurations, the resource consumption penalty paid for a performance increase, by switching to soft floating-point resources in the FFT pipelines, is much too severe. The ALM utilization jumps 22% in for a 16x16x16 design, while adding hundreds of nanoseconds of simulated reality per day. For a 32x32x32 design, ALM consumption jumps 36% only adding approximately 10ns of simulated reality per day. Not only that, ALM consumption is heavily dependent on the size of the grid, which could lead to routing issues for design larger than 32x32x32. Not to mention that all the FP resources offered by the device are unused. One could argue that the waste of FP resources may suggest that a different device should be used, unfortunately devices with large amounts of block memory bits include a large number of hardened FP resources.

Breaking down the computational run-times of these designs according to architectural latencies and operational phases, it is blatantly obvious that the run-times are dominated by the number of particles in the system. The percentages of run-time due to the number of particles in the system for configurations 0, 1, 2, and 3 are 88.99%, 94.24%, 89.04%, and 94.21%, respectively. Taking full advantage of the hardened FP resources inside the FFT pipelines offers up the potential of implementing a 128x128x128 design on a single FPGA. However, even with the potential offered by the capacities of these large FPGAs, and despite only having two data points, it's safe to assume that as the grid size continues to increase, the operational frequency of the system will decrease. That said, the only options to reduce this dominant run-time component are to:

1. Have the charge mapping and force computation run at higher clock frequencies. Considering that all LR computational phases use the same clustered grid memory, this is impossible.

2. Parallelize the mapping and force computation. That is, map multiple particles per clock cycle and compute the forces acting on multiple particles per clock cycle.

The parallelization of charge mapping and force computation can be achieved by breaking out the grid into multiple clustered grid memories and adding one charge mapping and force computation pipeline per added clustered grid memory. With this restructuring of grid memory, arbitration would be required to account for a particle that maps to multiple clustered grid memories. The dynamic throughput of particle mapping and force computation will decrease when arbitration is taking place, but with large enough clustered grid memories, the need for active arbitration should minimal. Breaking up the grid into four rectangular prisms and assuming 25% of throughput is lost due to arbitration, a single LR iteration would take about 124μs to complete leading to in 1.39μs of simulated reality per day. The added resource consumption of this parallelization should not be a problem assuming the design makes use of hardened floating point resources.

Considering the disparity between the charge mapping and force computation run-times when compared to the FFT run-times, performance gains offered by parallelization can be achieved through the redistribution of resources currently being utilized. Said differently, resources taken from the DSP pipelines may offer more impactful performance increases in particle mapping while taking small performance hits in FFT-phase run-times. Rewriting Equation 4.4 in terms of hardware resources results in:

$$t_{CMAll} = \frac{n_{particles}}{\frac{r_{CMBase}+r_{CMAdd}}{r_{CMBase}}} + d_{CM} - 1 \tag{4.15}$$

where r_{CMBase} is the base set of resources the charge mapping pipeline utilizes and r_{CMAdd} is the number of resources added to charge mapping pipeline following the redistribution of FFT-pipeline resources. Similarly, Equation 4.13 can be rewritten in terms of the hardware resources it uses as:

$$t_{FCAll} = \frac{n_{particles}}{\frac{r_{FCBase}+r_{FCAdd}}{r_{FCBase}}} + d_{FC} - 1 \tag{4.16}$$

where r_{FCBase} is the base set of resources the force computation pipeline utilizes and r_{FCAdd} is the number of resources added to force computation pipeline following the redistribution of FFT-pipeline resources. Modeling the run-times of each 1D FFT/IFFT phase in terms of hardware utilization is slightly more complex because of the nested pipelining. The $NNN1D$ in Equations 4.6, 4.7, and 4.8 stems from the fact that there ate $NNN3D$ FFT pipelines and that the *NNN*3*D* pipelines are initiated in a staggered fashion *NNN*2*D* after another. With that in mind, the increase in run-time for FFT/IFFT phases in terms of resources lost is as follows:

$$t_{FFT_{1D}} = \frac{X_G Y_G Z_G}{\gamma} + d_{FFT_{1D}} + \frac{\gamma}{\sqrt[3]{\gamma^2}} - 1 - 4 \tag{4.17}$$

where:

$$\gamma = NNN3D \frac{r_{FFTBase} - r_{FFTSub}}{r_{FFTBase}} \tag{4.18}$$

and $r_{FFTBase}$ is the base set of resources the FFT pipelines and r_{FFTSub} is the number of resources taken from the FFT pipelines to be distributed among both the charge mapping and force computation pipelines. Figure 4·2 reveals that DSP resources limit potential hardware distribution. That said, Figure 4·5 plots the performance of the single-FPGA LR accelerator, as defined in Configuration 1, as resources are taken from the FFT pipelines and redistributed among the charge mapping and force computation pipelines. The different curves plotted are the result of varying the charge-mapping-to-force-computation distribution ratio. The amount of resources traded off varies from 0 to 2016, the maximum number of DSP resources that can be taken from the FFT pipelines while still retaining 1 FFT pipeline. From this plot it can be observed that there is a point where the FFT run-times start to dominate the overall run-time of a single LR iteration. This is simply due to the lack of parallelization offered by the reduced number of FFT pipelines. Of the 12 curves, the 40:60 charge-mapping-to-force-computation redistribution of FFT DSPs offers the best performance. In reality, this performance model is a 2D function operating over the DSP resources traded off and redistribution ratio. Table 4.2 highlights some of the best resulting run-times of this 2D function. The first 12 entries of this table list the minimum of each of the curves in Figure 4·5. The next entry captures the global minimum of the 2D function. Realistically, only integer numbers of pipelines can be implemented, so true

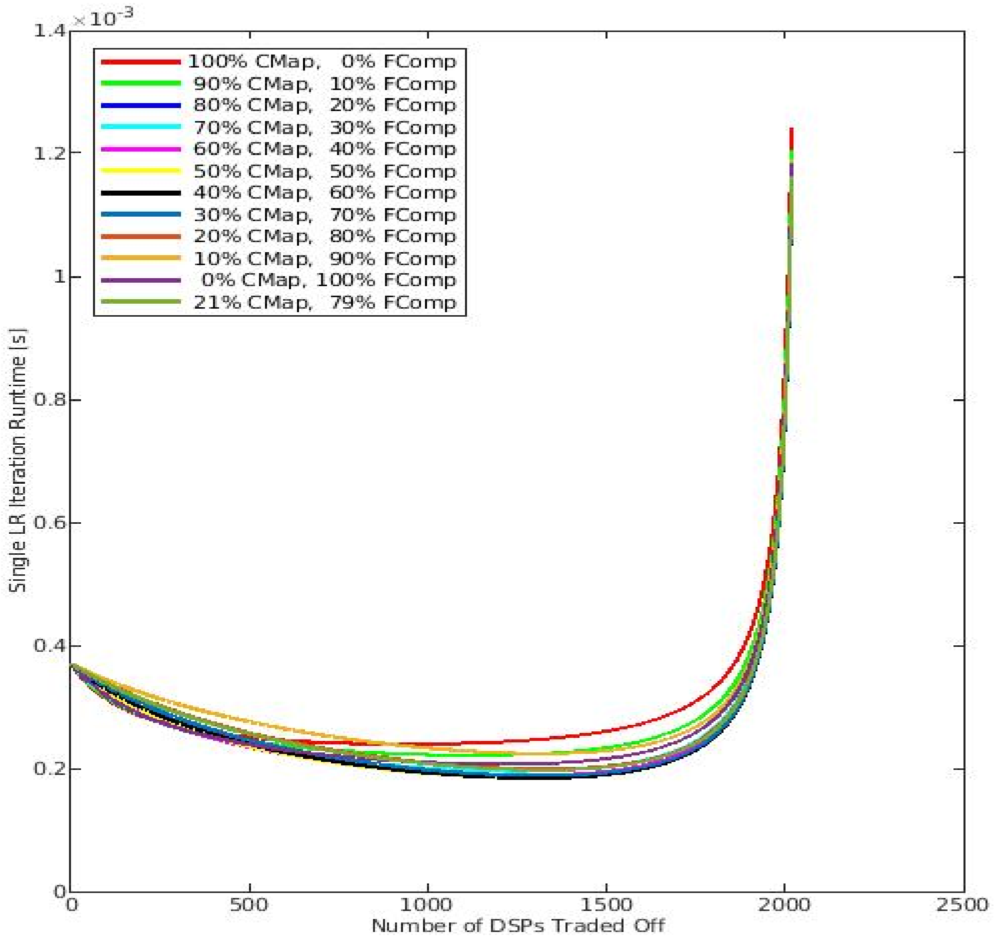

Figure 4·5: LR Run-Times vs. DSP Tradeoffs

performance lies somewhere in the middle of all these results. Again, it should be mentioned that this model does not account for the contention introduced when parallelizing the charge mapping and force computation pipelines, so real hardware performance will be slightly less than what is approximated here.

Ratio	**DSP Count**	**Run-Time**	**CM Pipelines**	**FC Pipelines**	**FFT Pipes**
Figure 4·5 Curve Minima					
100% CM	940	240.88μs	5.5854	1	34
90:10	1,123	222.78μs	5.9302	1.1462	28.90625
80:20	1,206	208.06μs	5.7063	1.3141	26.3125
70:30	1,248	197.26μs	5.2615	1.4875	25
60:40	1,273	189.99μs	4.7259	1.6630	24.21875
50:50	1,290	186.03μs	4.1463	1.8398	23.6875
40:60	1,304	185.61μs	3.5444	2.0187	23.25
30:70	1,317	189.65μs	2.9273	2.2004	22.84375
20:80	1,327	200.66μs	2.2946	2.3823	22.53125
10:90	1,319	225.58μs	1.6434	2.5457	22.78125
100% FC	1,206	208.06μs	1	2.5703	26.3125
205:768	1,326	199.01μs	2.3628	2.3628	22.5625
Global Minimum					
839:1177	1,690	132.55μs	4.42	2.29	11.1875

Table 4.2: Run-Times Following DSP Resource Redistribution

Lastly, recall that LR is just one portion of MD simulation. Considering that each of these MD phases are run on their own FPGA, it's not unreasonable to consider the distribution of LR acceleration across multiple FPGAs. Similar arbitration issues to the grid-partitioning solution above will need to be addressed, but the main advantages of a multi-FPGA network would be the increase in parallelization and the speed of the the network.

Chapter 5

Extending Single-FPGA LR Designs: Off Chip Memory

As MD grid sizes increase, the need for more FPGA resources, especially BRAMs, becomes evident. As the number of resources increases, so does the routing difficulty. Keeping that in mind, is there an opportunity to keep the accelerator design all on a single board making use of off-chip memories? High bandwdith memory (HBM) is the latest offering from semiconductor memory companies attempting to bridge the gap between processor latencies and memory latencies (Samsung, 2019; AMD, 2020). More and more FPGAs are being packaged with on-board HBMs to alleviate the memory bandwidth bottleneck that many FPGA-accelerated applications encounter (Xilinx, 2020a; Intel, 2020b). HBMs have already been used inside of MD acceleration applications (Stewart et al., 2020). This chapter discusses the plausibility of leveraging HBM inside of the LR design. There are two main areas of the LR design where HBM could alleviate resource consumption inside the FPGA and potentially offer better system performance: particle mapping and FFT. The remainder of this chapter discusses the use of HBMs in each area and the discusses the overall viability of HBMs in an LR design.

5.1 HBMs for Particle Mapping

When it comes to particle mapping, HBMs can be used in isolation inside the particle information memory the grid memory to alleviate on-FPGA resource consumption in hopes that the resource savings equate to higher system clock frequencies. Building off of this isolated usage of HBMs in the two memory subsystems, there is potential to increase the throughput of particle mapping to something higher than one particle per clock cycle.

5.1.1 HBMs for Particle Information Memory

Using the per-particle data storage requirement derived in Equation 4.2 16^3, 32^3, 64^3, and 128^3 grid-point designs require 62.5KB, 512KB, 4.08MB, 33.5MB, respectively, of particle information memory. Currently, FPGA vendors are offering FPGAs packaged with 8GB-16GB HBMs (Xilinx, 2020a; Intel, 2020b). With more than enough storage offered by HBMs, the main issue becomes performance. Even when the HBM subsystem can operate at frequencies higher than the rest of the system and access the HBM over multiple channels, the latency penalty is too high (Wang et al., 2020b). That coupled with the burst size requirement of HBM controllers, results in the need for ping-pong particle information caching. These caches need only be large enough to amortize the latency penalty paid when accessing the HBMs while also fulfilling the burst requirements of the HBM access controller. During charge mapping and force interpolation particle information is only ever read out and can be accessed sequentially by simply incrementing the memory address. So this caching does not require any complex replacement or write-back schemes.

5.1.2 HBMs in the Grid Memory

With 64 bits of data storage per grid-point, the current HBM offerings by FPGA vendors provide more than enough storage capacity. Not only is latency an issue here, the random and parallelized memory access nature of charge mapping and force interpolation present another issue: access flexibility.

HBM access controllers currently provided by FPGA vendors offer access to 16 physically independent HBM structures, but by isolating address regions within each HBM structure, accesses can be spread out over 32 64-bit pseudo channels (Wang et al., 2020b; Intel, 2020b). Charge mapping and force interpolation require 64 64-bit parallel accesses.

Therefore another type of grid-point cache clustering would be required. This cache must only store subsections of the entire grid, otherwise it defeats the purpose of using HBMs. The caching of grid-point information will require write-back and replacement logic to account for cache-misses when particles map to grid-points not currently located in the cache. These cache misses penalize charge mapping and force interpolation performance, but can be mitigated by sorting the particle information off-line. However, misses cannot be avoided due to the inherent capacity issue of the caching architecture. To compensate for the random access nature of a particles mapped grid-points, all of which existing in the cache, the idea would be to implement a smaller-sized clustered grid memory than in the lone-FPGA LR design.

5.1.3 Better Particle Mapping Bandwidth with HBMs

By strategically combining the caching methodologies of the particle information memory and the grid memory, it may be possible to map more than one particle per clock cycle. The particle information caching can pull information of more than on particle per HBM access. If multiple grid memory caches are made, and particles are sorted ahead of time to reduce grid-point cache misses, it may be possible to achieve a higher average particle mapping bandwidth. This increased mapping bandwidth can be used both in charge mapping and during force interpolation, the latter provided there are enough parallel force-tree pipelines to support it.

5.2 HBMs for FFTs

With most of the grid-point storage in HBMs and with enough HBM capacity to store grid values many times over, this can allow for the use of a different memory architecture for the running of the FFT stages. The clustered grid memory in the lone-FPGA design was shared by all portions of the LR computation, thus limiting the number of FFT pipelines to 64. However, with more on-FPGA resources available in this off-chip memory solution, following charge mapping and 5 of the 6 FFT stages of LR computation, grid-point information could be streamed into a separate memory units allowing for more FFT parallelization. For example, information could be stored in a X_SxY_S matrix of Z_G-deep memories (where $X_S < X_G$,$Y_S < Y_G$) allowing X_SY_S FFTs to happen in parallel. Now to complete the FFT in one dimension multiple iterations of parallelized X_SY_S FFTs would be required, but with the ease-of-predictability of these FFT grid-point subset accesses, ping-pong buffering can be employed to hide the latency of loading the FFT memories. With enough HBM capacity the FFT results could be transposed as they are written into a separate HBM address region. Writing to a separate HBM space prevents the corruption of data required to complete the 1D FFT in the current dimension while allowing for the storage of data required for the 1D FFT in the next dimension. The transpose would account for the re-purposing of the FFT memory architecture from an X_SxY_SxZ_G-deep to a Y_SxZ_SxX_G-deep, for example.

Considering that the Green's ROM requires half the storage of the grid memory, current HBM offerings easily meet the needs of the LR system. Again, with latency being the issue, a caching subsystem would need to be developed to absorb that penalty but also accommodate the FFT memory architecture. Using HBMs as ROM does require a one-time programming of the Green's function values prior to running any LR iterations. This is not necessarily a drawback from a performance perspective as it is a system configuration task, but it sure is convenient to have all the Green's ROM values simply provided via FPGA programming.

5.3 Viability Discussion

There are multiple opportunities for the usage of HBMs when accelerating the LR portion of MD. Though its implementation does not free up all the memory resource requirements of the particle information memory, the grid memory, and the Green's ROM, the savings can be significant depending on the caching methodologies deployed to mitigate latency and access flexibility issues. However, if these resource savings do not result in a significant increase in the operational clock frequency of the LR system, then using HBMs to accelerate the LR computation is assumed to be not viable. Storing the particle information can free up some memory resources but it does not directly allow the charge mapping or force interpolation to run faster. Storing the grid-point data inside of HBMs offers similar benefits, but also does not increase the mapping bandwidth its own. It also introduces performance penalties upon cache misses. Putting together the particle information and grid-point caches may still save the number of resources consumed, but performance is heavily dependent on the locations of particles in the ensemble. Also the throughput of particle information mapping gets halved when each particle requires more than 128-bits of storage. HBM usage for FFT throughput improvement has the potential to be helpful as FFT computation time is not insignificant, but overall LR computation is dominated by particle mapping.

With the use of HBMs seemingly offering limited particle mapping bandwidth improvement and with particle mapping bandwidth accounting for over 85% of LR computation time, the use of HBMs to further improve LR computation performance is viable, but may not be entirely worthwhile. A better solution may be to split the grid over multiple FPGAs, allowing for the particle mapping run-time to be nearly divided by the number of FPGAs in the network.

Chapter 6

Beyond Single-FPGA: FPGA Clusters

With larger MD grid sizes consuming more FPGA resources while reducing system clock frequency and off-chip memories seemingly offering little performance improvement, an obvious choice is to spread LR computation across multiple FPGAs. With portions of the grid mapped to individual FPGAs, particle mapping run-times can be nearly divided by the number of FPGAs in the system. However spreading LR across multiple FPGAs requires inter-node communication when performing the 1D FFTs and when a particles map to grid-points spread across multiple FPGAs.

Leveraging the results of the single-FPGA LR implementations, that is, without multiple charge mapping and force computation pipelines, this chapter evaluates three FPGA network architectures: a direct-connect toroid, an all-to-all connection switching network, and bidirectional ring. It describes the architectures of each topology and the inter-FPGA communication methods as they pertain to the various phases of LR in an attempt to characterize the performance of the system. The chapter concludes with a trade-off discussion between the various systems.

6.1 Direct-Connect Toroid

This section evaluates the 3D MD grid overlaid onto a direct-connect toroidal network of FPGAs.

6.1.1 Architecture

Figure 6·1 depicts the toriodal network. Each FPGA is directly connected to each of its

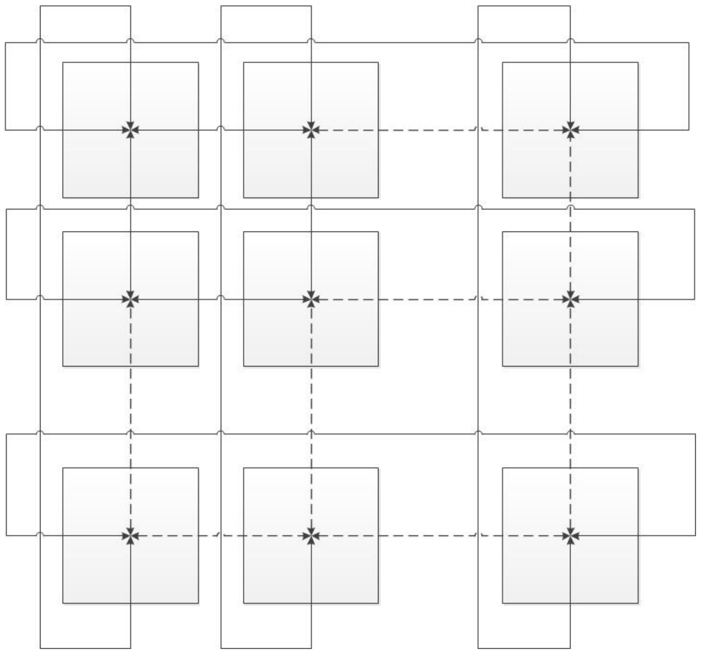

Figure 6·1: 2D Direct-Connect Toroidal FPGA Cluster Network

neighbors in 2D via bidirectional communication links. This requires each FPGA to support four bidirectional links. Such devices are offered by both Xilinx and Intel (Intel, 2020b; Xilinx, 2020b). The idea with this network is that each FPGA is mapped to a subset of grid-points in the shape of a rectangular prism (a grid-prism) as shown in Figure 6·2. Each grid-prism at least spans all the grid-points in a single dimension while typically only spanning subsets of grid-points in the other two. This grid-prism construction constraint allows each FPGA to perform a 1D FFT over its entire grid-prism without the need for inter-FPGA communication. Thus, the run-time of the 1D FFT over the entire grid is equal to the run-time of the 1D FFT over the grid-prism in this one dimension, where as 1D FFTs in the other two dimensions will require inter-FPGA communication. This grid-prism structuring governs the 2D toroidal network requiring only 4 network links. If a grid-prism did not

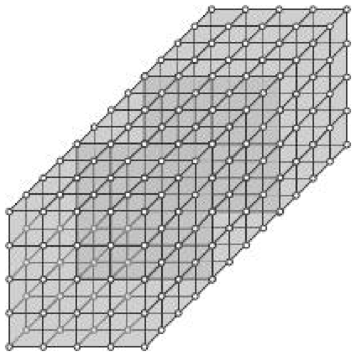

Figure 6·2: FPGA Cluster Node Grid-Prism

span all the grid points in one dimension, a 3D toroidal network would be required for this direct-connect architecture.

6.1.2 Inter-FPGA Messaging

In all multi-FPGA LR systems, data movement needs to be quantified. The following subsections describe the various data movement patterns in order to spell out a network messaging topology that can be used to assess system performance.

Particle Information Transmission

With the underlying grid-prism memory structuring warranting a 2D toroidal network, each particle only maps to a subset of grid-points that could exist on at most 4 FPGAs. Therefore, the particle information may need to be relayed over multiple FPGAs before it arrives at all its destinations. Figure 6·3 depicts how particle information traverses a 4x4 2D toroidal network.

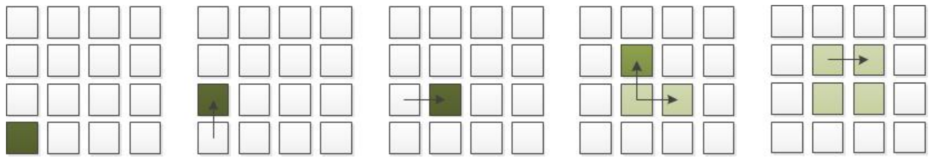

Figure 6·3: Direct-Connect Particle Info Messaging

From left to right, a message is first transmitted from a host to the host-node of the network. The particle then traverses the network until it finds a home-node, that is, the closest node to the host-node that contains grid-points to which a particle will be mapped. Part of the message that gets passed is the particle's nearest-floor-grid-point. Each node in the network is aware of where in the overall grid its grid-points are located. It reviews the nearest-floor-grid-point in the message to determine whether it is the home-node. If it is not, it passes the message along to the node nearest to the home-node. When a node identifies itself as the home-node for the particle information message, it locally stores the information and relays the message to other nodes when a particle maps to grid-points spread across multiple nodes. When appropriate, one of the two relay messages will be again relayed in order for the particle information to be mapped to all its respective nodes. If the self-aware nodes identify the port in which the information is passed over and combine it with the particle location information in the message, only one message type is required to transmit particle information.

Each node in the network needs to know when it can start its localized processing of particle information. With a particle mapping to a minimum of one node, the message used to map the last particle should be relayed to all nodes to signify the end of particle information transmittal. Figure 6·4 shows how this additional message type is related across a 4x4 2D toroidal network.

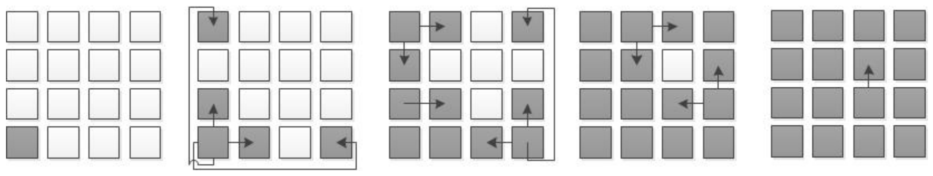

Figure 6·4: Direct-Connect Last Particle Info Messaging

The last particle's information message is first relayed by the host-node to all its 2D neighbors. Then, each self-aware node relays the message in a predetermined direction to avoid unnecessary message duplication while minimizing the time for all nodes to receive the last-particle message.

Lastly, it should be noted that the grid-prisms need to be cleared out before each iteration. This clearing is performed as particle information is being sent over to each FPGA.

FFT

In this network of grid-prisms, FFT in one dimensions is performed completely on chip. FFTs in the other two dimensions require inter-FPGA communication. Figure 6·5 depicts FFT messaging in the second dimension assuming a 4xY toroid.

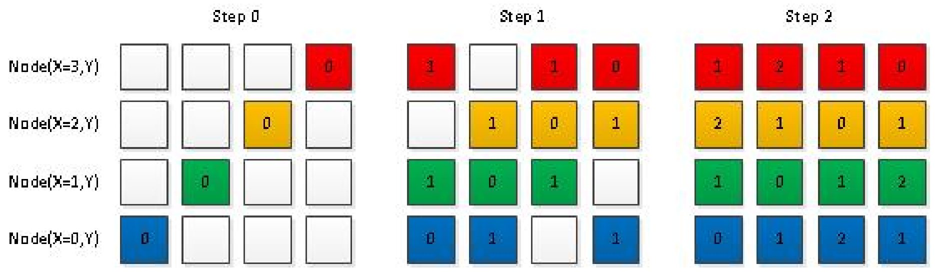

Figure 6·5: Direct-Connect Second Dimension FFT Messaging

From top to bottom, the messaging for each node in a horizontal line is shown, message traversal is shown from left to right. In step 0 each node possesses an FFT point that needs to be passed to all other nodes. In step 2 each node passes its FFT point to its two nearest horizontal neighbors. In step 3 the message passed horizontally forward is then relayed to the last node in the line.

This two-forward-one-back messaging style is repeated until all nodes transfer all of their points in a given FFT pencil to all the remaining in-line nodes. Each node contains FFT-pencil ping-pong buffers to store the FFT points for a given pencil until all of them have been received. Once the complete pencil has been received it can then enter an FFT pipeline. The messaging of points for the next FFT pencil happens concurrent to the running of each node's FFT pipeline. Each node duplicates the same FFT operation but only stores results applicable to the subset grid-points it houses. Each node is assumed to have enough FFT pipelines and FFT pencil buffers to keep the network saturated while generating and storing FFT results.

Given the 2D toroidal connectivity of the network, FFTs in the second and third dimension can follow the same pattern, albeit transposed. Assuming that grid-point location information is passed alongside the FFT-point data, the self-awareness of each node, and the ability of each node to identify the optical port in which the information was passed over, FFTs in the second and third dimensions result in the addition of one message type.

Force Computation

With a maximum of four nodes containing force information, Figure 6·6 captures the force computation messaging of a 4x4 toroidal network.

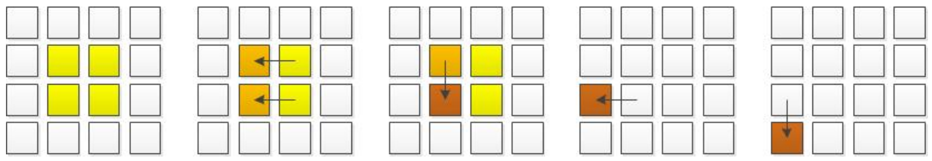

Figure 6·6: Direct-Connect Force Computation Messaging

Partial force interpolation and summation is performed on each node that a particle maps to. Next, each non-home-node transmits its partial force sum in the direction of the home-node, performing additional summations along the way, when appropriate. The home-node performs the last round of force summation, having received all partial sums from all non-home-nodes, then transmits final sum in the direction of the host-node. Assuming particle location information is passed along with the force data in each message, the self-aware routing logic can use link identifiers to limit the amount of messages required to send force information to one.

6.2 All-to-All

This section evaluates a 3D MD grid distributed over multiple FPGAs connected through an optical switch, covering scenarios where:

1. Each FPGA node supports one optical link.
2. Each FPGA node supports two optical links.

6.2.1 Architecture

In these two multi-FPGA network scenarios, each FPGA can support either one or two bidirectional optical links. All links are connected up to an optical switch. Such FPGA clusters are currently in active deployment (Plessl, 2018). Messages sourced by one FPGA enter the switch and are routed to their appropriate destination.

Each node in this network is assumed to be mapped to a set of MD grid-points in the form of the previously described grid-prism. In these network topologies the use of grid-prisms only cuts down on the amount of inter-FPGA communication during the 3D FFT phase of LR.

6.2.2 Inter-FPGA Messaging

The following sub sections describe the messaging strategies employed for all phases of LR computation to allow for later performance analysis.

Particle Information Transmission

Figure 6·7 captures particle information messaging for the all-to-all network assuming that the 3D grid is mapped to a 4x4 matrix of grid-prisms and two network links per node.

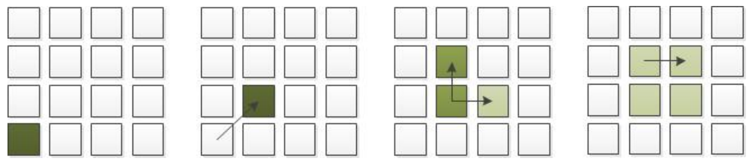

Figure 6·7: All-to-All Two-Link Particle Info Messaging

Much similar to the direct-connect toroidal network, a particle's information is transmitted from a host to the host-node, then to the particle's home-node, then spread to additional nodes when applicable. Alternatively, the host-node could transmit a particle's information to all nodes that a particle would map to sequentially, but doing so wastes the network's bandwidth potential, creating an unnecessary network bottleneck at the host-node. The all-to-all connectivity allows for one-hop message transmittal from the host-node to the home-node, allowing the message to arrive at its destination sooner.

Cluster synchronization by relaying the 'last particle' message is also performed in a similar fashion to the 2D toroidal network. Figure 6·8 shows one way the message can be relayed.

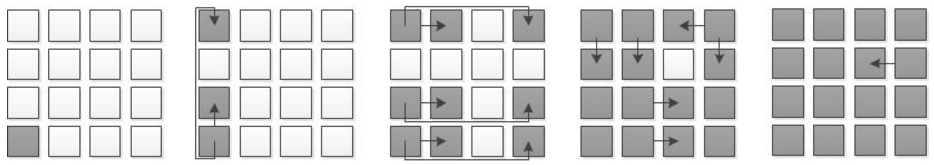

Figure 6·8: All-to-All Two-Link Last Particle Info Messaging

Even with half the communication links as a 2D toroid, the system can be synchronized after 4 relays.

Figure 6·9 captures particle information messaging for the all-to-all network with a one-link-per-node constraint.

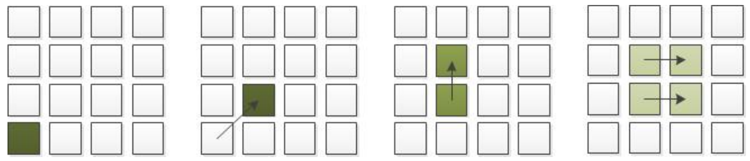

Figure 6·9: All-to-All One-Link Particle Info Messaging

With the one-link-per-node constraint, a particle's information cannot be locally broadcast from its home-node. However, with slight hardware modifications, mapping data from the home-node to the remainder of nodes mapped to a particle can happen in the same amount of hops. The same can be said for system synchronization using the 'last particle' message, details of which are shown in Figure 6·10.

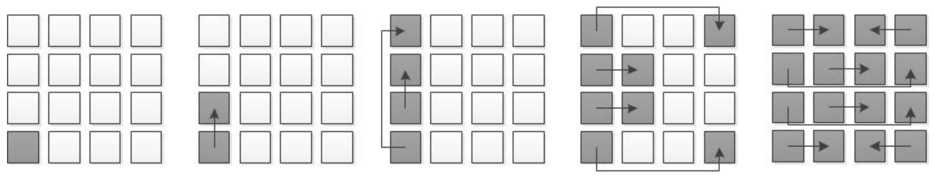

Figure 6·10: All-to-All One-Link Last Particle Info Messaging

Four relays is all that is needed to synchronize the entire cluster even with each node supporting one communication link.

The need for grid clearing is agnostic to network topology. Again, assuming that the number of particles equals the number of grid-points and localizing that assumption to each node, grid clearing can be performed as particle information is being distributed.

It should noted that there are current FPGA clusters where each node supports more than two communication links (Plessl, 2018). Since particle information is stored at unique addresses on each node, arbitration logic can be added to make multi-port particle information memories, thereby doubling the bandwidth to both map particles to nodes and to synchronize the cluster.

FFT

Again, with the grid-prism-based node mapping 1D FFTs in one dimension can be performed by all nodes individually in parallel, requiring only FFTs in the second and third dimensions to require inter-node communication. Despite the all-to-all connectivity, maximizing throughput requires the inter-node FFT communications to follow the same topology as shown in 6·5, in the two-link-per-node case. An all-to-all connected network with one-link-per-node requires a slightly different messaging. Figure 6·11 details this method for a 4xY matrix of nodes.

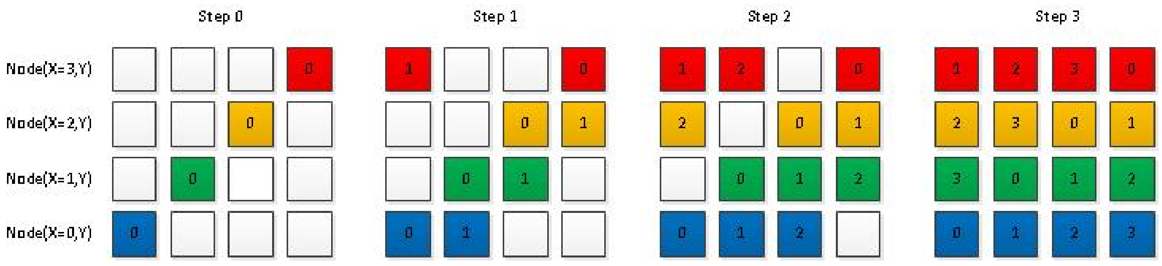

Figure 6·11: All-to-All One-Link Second Dimension FFT Messaging

From top to bottom, the messaging for each node in a horizontal line is shown, message traversal is shown from left to right. In step 0 each node possesses an FFT point that needs to be passed to all other nodes. Steps 1 to 3 relays the FFT points in a forward, toroidal fashion. Doing so maximizes throughput while avoiding node contention, but ends up requiring one additional relay than the two-link, all-to-all and direct-connect toroidal networks.

Assuming that each node has enough ping-pong buffering and FFT pipelines, FFTs can be run over a pencil of points while points for another pencil are being transmitted.

Force Computation

Force computation messaging for the all-to-all network with one or two links per FPGA is very similar that of the direct-connect toroidal network, as seen in Figure 6·12.

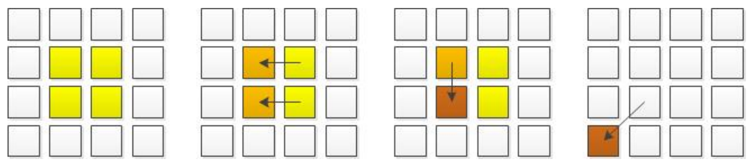

Figure 6·12: All-to-All Force Computation Messaging

The only difference with this messaging method and the one presented with the direct-connect toroidal network is that once the home node sums together the forces acting on a single particle, the sum is sent to the host-node in one hop.

In both the two-link-per-node all-to-all network and for all-to-all networks with more than two links per node force computation could be parallelized, by additional summation pipelines to take advantage of traffic over the additional network links. However, additional hardware would need to account for the possibility of node contention when computing and transmitting the forces on more than one particle in parallel.

6.3 Bidirectional Ring

For FPGA clusters without an optical switch, but with nodes that can support at least two, but not four, links per node, the 3D MD grid can be overlaid onto a 2D matrix of nodes connected in a ring.

6.3.1 Architecture

Again, mapping each node to a grid-prism subset of grid-points, as defined in Figure 6·2, allows for the 3D grid to be mapped to a 2D matrix of network nodes. As shown in Figure 6·13, node connectivity snakes along the nodes primarily along one dimension, only

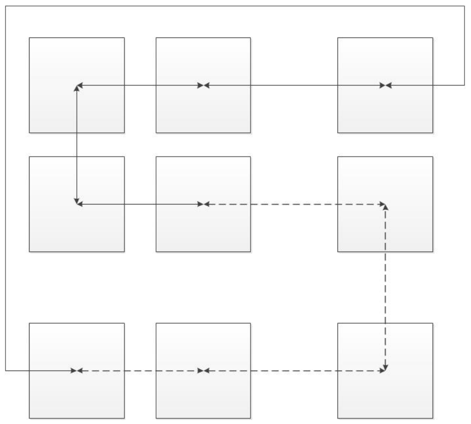
Figure 6·13: Ring-Connected 2D Matrix FPGA Cluster Network

making orthogonal connections after reaching the final node in along the primary direction. The last node is then lopped back to the first node to complete the ring.

6.3.2 Inter-FPGA Messaging

Without a network switch and with only two network links per node, inter-node communication is vastly different from the previous two architectures. The following subsections describes the messaging methods for all phases of LR computation.

Particle Information Transmission

With two bidirectional links per node, there are two viable particle information messaging. Figure 6·14 captures one of them. When a particle is mapped to one node, messages will travel along the ring in the direction requiring the least amount of hops.

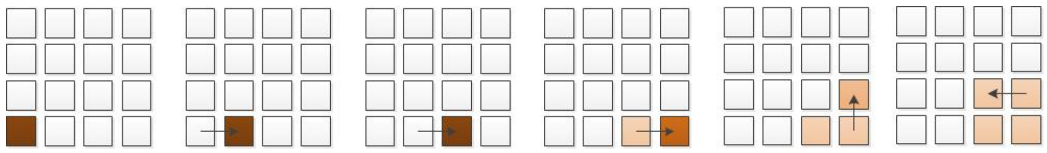
Figure 6·14: Ring Unidirectional Particle Info Messaging

The same can be said when a particle is mapped to multiple nodes. Figure 6·15 accounts for the other type of particle information messaging.

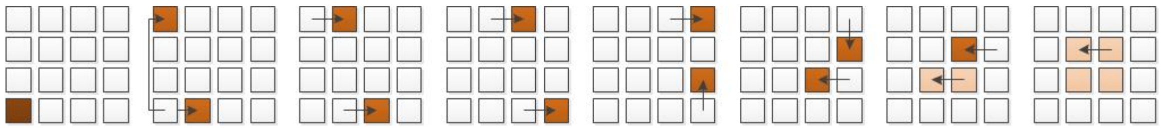

Figure 6·15: Ring Bidirectional Particle Info Messaging

With this type of message traversal it is determined that a particle maps to multiple nodes and can be mapped in the least number of hops by broadcasting that message in both directions.

FFT

Once again, FFTs in one dimension can be performed on each node simultaneously taking advantage of the grid-prism organization of grid-points mapped to each node. The 1D FFT performed along the primary dimension-traversal of node connectivity can follow the method as described in Figure 6·5. The 1D FFT performed along the dimensional orthogonal to the primary dimension-traversal of node connectivity is a bit more complex. Figure 6·16 depicts message patterns for some of the nodes in the network as they relay their data points to other nodes in a line orthogonal to the primary dimension traversal of the ring. Regardless of the node's location or messaging direction, it can to relay its FFT points to two other nodes in the same orthogonal FFT line in 8 relay stages. This will be known henceforth as a *long message*. A *short message* has only one destination requiring less than 8 relay stages. The exact number of relay stages of a *short message* is dependent on the source node's location in the ring and the direction of the messaging.

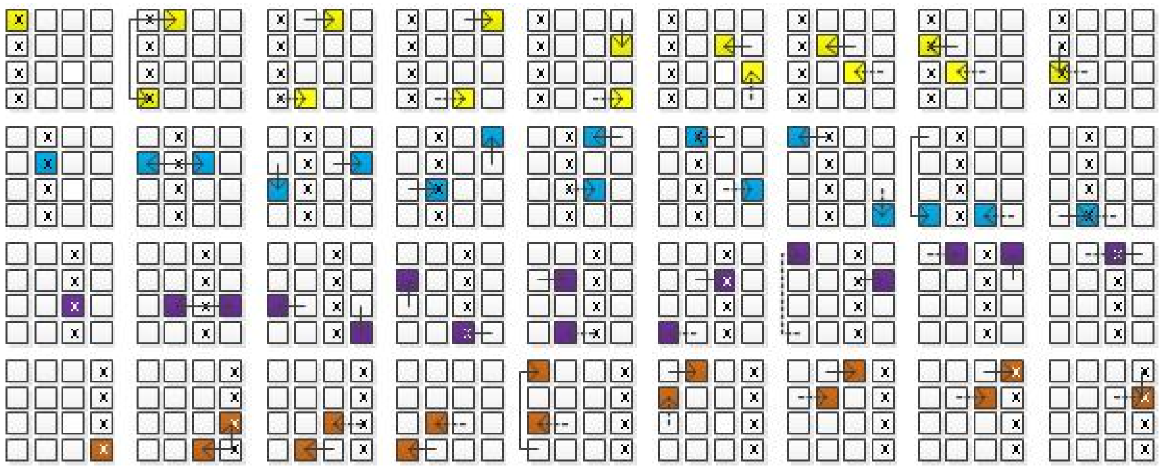
Figure 6·16: Ring Orthogonal FFT Messaging

Through simulation, it was found that by:

1. Priming each node to send long messages in one direction while sending short messages in the other.
2. Configuring each node to prioritize the relay of information from other nodes over transmitting its own information.
3. Toggling the type of message transmitted each time a new FFT point is sourced by a node (for each direction).

That the the system achieved a steady state of repetitive system-wide message patterning. Details of this patterning are depicted in Figure A·1 in the appendix. This repetitive patterning means that nodes stay relatively in sync when it comes to what FFT pencils they are actively operating over and temporarily storing. With adequate ping-pong buffering to accommodate:

1. How far ahead certain nodes can be with regards to FFT processing relative to one another.
2. The ratio of operational clock frequency between the network and the node.

It’s possible to keep the network saturated while simultaneously running FFTs on each node.

Force Computation

Figure 6·17 depicts how force computation messages traverse the network.

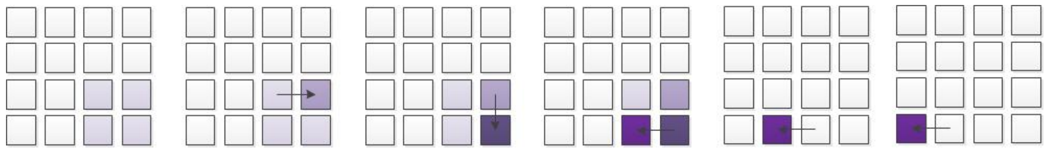

Figure 6·17: Ring Force Computation Messaging

When mapped partially to one node, the three dimensional forces are partially summed on that node and then passed to next node closest to the host-node in a snaking fashion. This inter-node partial summing is repeated until the second to last partial sum arrives at the home-node for that particle. The last summation for that particle’s forces is then performed inside the home-node and the final force values for that particle are then relayed to the host-node.

6.4 Performance Analysis

The three aforementioned networks are evaluated by mapping an LR design that accommodates a 64x64x64 grid and 64^3 particles to a 4x4 matrix of nodes. Each node utilizes the same amount of data precision as the single-FPGA LR design, that is, b_o and and b_{qf} are the same in both cases. Again, the only differences between the networks are the number of links supported by each node which lead to different interconnection strategies. Each link is assumed to support a 100Gbps serial link. These assumptions, combined with the measured performance of single-FPGA LR implementations, are then used in first order approximations of system performance.

After particle information is transmitted to all nodes, charge mapping and 1D FFT/IFFT in one dimension are identical across all three networks. To that effect, the performance of charge mapping and the 1D FFT and IFFT in that one dimension are analyzed here. The remaining sections analyze each of the three networks with respect to particle information messaging, FFTs/IFFTs in the second and third dimensions, and force information messaging.

Once the information is cached, particle mapping to intra-node clustered grid memory can begin. With 16,384 particles stored on each node, using the the same pipeline latencies as the single-FPGA LR design, and assuming a 200MHz operating frequency, mapping can be completed in 82.12μs.

FFT/IFFT in the one dimension can all be performed intra-node, but that FFT must be along the longest the longest dimension of the 16x16x64 grid-prism each node is mapped to. Making use of the equations derived for single-FPGA 1D FFTs, assuming that the latency of the FFT IP in this longest intra-node dimension increases from what it was in the 32x32x32, single-FPGA design (+64 clock cycles), and a 200MHz clock frequency, the FFT and IFFT along this long dimension each consume 2.46μs of wall-clock time.

Direct-Connect Toroid

The direct-connect toroidal network architecture defined four unique message types, thus each message must contain two bits for message type identification. The need for message identification becomes more evident considering the variability in data being sent across all message types. Now all messages require that grid-point information be transmitted for message routing, but the true data fields for particle information messaging, FFT messaging, and force messaging are 113 bits wide, 64 bits wide, and 96 bits wide, respectively.

For particle information transmission it is assumed that on average each particle message arrives at its destination after 2.5 hops, 3.5 considering the 1 hop from the host to the host-node. However, network traversal can be pipelined, therefore the time it takes to distribute all particle information is equal to the number of particles multiplied by the one-hop network latency, plus the network latency of mapping the last particle to all nodes minus the latency of one hop. With one particle message per particle, and 2 message ID bits, 18 grid-point ID bits, and 113 offset-charge-data bits per message, it takes about 348.659817μs to transmit all the particle information. Considering the potential for node contention when particles that map to multiple nodes, the link throughput can be halved to roughly account for this network contention. Doing so, doubles the particle messaging time to 697.31634μs.

FFT/IFFT in the second and third dimension are transposes of each other, but are identical in terms of run-time. FFT messages contain 2 bits for message ID, 18 bits for grid-point ID and 64 bits for complex data. With two relay stages until a message reaches all destinations and 16 messages required for all nodes to start an FFT, all data for an FFT-pencil can be delivered to all nodes in 26.88ns. With the FFT latency data captured from the single-FPGA design, it takes 1.025μs to write the FFT results back to the clustered grid memory. With 64 FFT pipelines per node, there is enough hardware to parallelize FFT computations alongside of FFT message transmission. This FFT messaging scheme is also pipelined in nature: as one message is located in a later relay stage, a newer message can be found at an earlier relay stage. Following the basic pipeline equation, the time it takes to get all messages to their destination is as follows:

$$\begin{aligned} t_{FFTMSGDCT_{23}} &= n_{nodegps} \cdot d_{1hop} + 2d_{1hop} - 1d_{1hop} \\ &= n_{nodegps} \cdot d_{1hop} + 1d_{1hop} \end{aligned} \tag{6.1}$$

or in words, equal to the latency of transmitting all messages one-hop plus the two-hop latency of the network minus one hop. Knowing that FFT computations overlap with messaging transmission, the overall 1D FFT/IFFT time needs to account for the latency of computing the last FFT/IFFT-pencil. Putting that all together results in:

$$\begin{aligned}
t_{FFTDCT_{23}} &= t_{FFTMSGDCT_{23}} + d_{inodeFFT1D_{23}} \\
&= n_{nodegps} \cdot d_{1hop} + 1d_{1hop} + d_{inodeFFT1D_{23}} \\
&= 16,384 \cdot \frac{84}{100 \cdot 10^9} + \frac{84}{100 \cdot 10^9} + 1.185 \cdot 10^{-6} \\
&= 14.9484\mu s
\end{aligned} \tag{6.2}$$

14.9634μs when accounting for the convolution following the FFT in the third dimension.

Force computation messaging is a bit more complicated to characterize. Assuming there is one particle for every floor-grid-point means that 9,216 of the 64^3 particles forces require two-staged relayed summing and 79,872 of the 64^3 particle forces sums require single-staged relayed summing (see Figure 6·6). Halving the network throughput to account for contention, 116-bit force messages require 2.32ns to complete 1 hop. With a maximum hop count of 5.5 to account for the additional summation hops and for the average message traversal from home-node to host, that results in 12.76ns for a partial sum message to become a complete sum message and make its way to the host. With each node operating at 200MHz, this summation time can be thought of as an addition of three 200MHz pipeline stages to the messaging latency per relay stage, call it ten 200MHz pipeline stages to account for additional on-node latencies to retrieve potentially stored force information. This pipelined nature of messaging allows network traversal to overlap with the each node's partial force computation. Plugging values into the classic pipeline performance equation, the force messaging run-time is calculated to be 38.07364μs.

All-to-All

The all-to-all network topology, in both the one link per node and the two link per node case, four unique message types are required. For particle information transmission, each particle message arrives at its destination after 1 hop to the home-node. However network traversal can be pipelined, therefore the time it takes to distribute all particle information is equal to the number of particles multiplied by the one-hop network latency, plus the network latency of mapping the last particle to all nodes minus the latency of one hop. With the same number of hops required to distribute the last particle message to all nodes as the direct-connect 2D toroid network, the one link per node network takes the same time to distribute all particle information, 348.65817μs, 697.31634μs, when accounting for network contention. This messaging time can be halved in the two-link-per-node case, assuming that the host can send particle information for two particles simultaneously.

In the two-link-per-node case, FFT/IFFT in the second and third dimensions is the same as with the direct-connect 2D toroidal network, 14.9484μs, 14.9634μs when accounting for the convolution following the FFT in the third dimension. The one-link-per-node case adds an additional hop to transmit pencil data to all nodes, thereby adding 840ps of to the wall-clock time of the two-link-per-node case for a total of 14.94924μs, 14.96424μ accounting for the convolution following the FFT in the third dimension.

Force computation in the all-to-all network is similar to that of the direct-connect toroid network except that each partial sum requires 3.5 hops to get to the host. This results in an overall force messaging run-time of 38.069μs. In both the one-link-per-node and two-links-per node it is assumed that each node can only output one force value per clock cycle, offering no options to further parallelize the force computation messaging.

Bidirectional Ring

The bidirectional ring network comes with the need for 7 message types, thus requiring one additional message ID bit per message.

Particle information messaging is similar to the other networks, only differing in the number of hops required to distribute the last particle message to all nodes (7) and in the number of bits per message (134). These slight differences result in a particle message run-time of 351.28234μs. With this ring network there is no possibility for network contention.

FFT/IFFT in the dimension parallel to the primary dimensional-traversal of node connectivity matches the that of the direct-connect 2D toroidal network, but requires an extra message-type bit, thus taking 15.11225μs to complete. FFT/IFFT in the dimension orthogonal to primary dimensional-traversal of node connectivity is similarly parallelized like the other two networks, but 8 hops are required to get all pencil points to all applicable nodes. That coupled with the addition of one message ID bit, FFT along this third dimension requires 15.1182μs to complete, 15.1332μs when accounting for the convolution following the FFT in the third dimension.

Force computation messaging for the ring network is again similar to that of the direct-connect 2D toroid, except that the average hop count to the host-node is 10 hops and with 117 bits in a force computation message, each network hop takes 2.34ns, resulting in an force computation messaging time of 38.41196μ.

6.5 Discussion

Figure 6·18 compares the best performing single-FPGA LR design that supports a 32x32x32 grid and 32^3 particles against the four multi-FPGA cluster LR designs that each support a 64x64x64 grid and 64^3 particles. The rest of this section highlights some of the results, discussing trends and oddities, then it goes on to discuss the cluster approach as a whole.

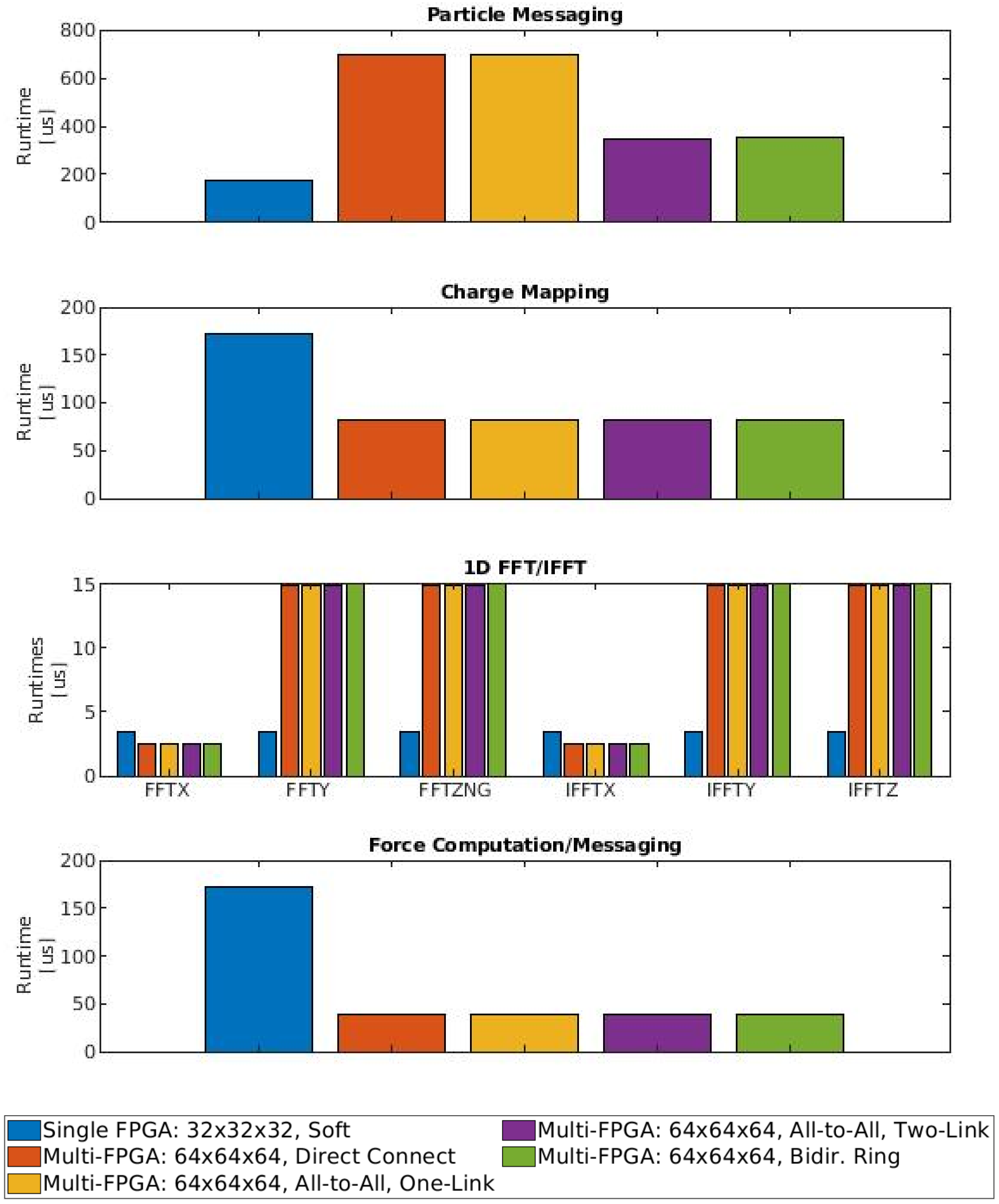

Figure 6·18: Single-FPGA and Multi-FPGA LR Design Performance Comparisons

Particle messaging was touched upon briefly when evaluating single-FPGA design, but not truly analyzed. With the prevalence of inter-node communication in the FPGA cluster deployments of LR acceleration, it's important to compare the performance messaging of particle information across all LR deployments. Particle messaging is purely based on the number of particles in the ensemble. The 32x32x32 design supports only 32^3 particles, so naturally the overall particle messaging runs in less time than the FPGA cluster designs, each of which support 64^3 particles. Even supposing that a single-FPGA design supporting a 64x64x64 grid and 64^3 particles could run at the same frequency as the best performing 32x32x32 LR design implementation, it would take 1.372ms to transfer all particle information into that single-FPGA. So as LR designs get larger, the benefits of FPGA cluster networks are obviated.

A couple additional notes on the particle messaging comparisons:

1. The all-to-all network with two-links-per node offers the best particle messaging solution because the flexibility in connectivity combined with the number of links allows for the doubling of network bandwidth.

2. The data suggests that the bidirectional ring architecture performs better than most, despite having less links and longer average messaging distances. The ring architecture removes the possibility for contention allowing for the full network bandwidth to be used in particle messaging performance approximations. The potential for contention in other networks resulted in halving the bandwidth numbers when approximating particle messaging performance. Considering the assumption of one particle mapped to each grid-point, approximately 34% of all particles to be mapped will map to multiple nodes resulting in the potential for contention. That said, the halving of bandwidth to account for contention seems overly pessimistic.

The benefits of FPGA cluster deployments are emphasized with respect to charge mapping. This LR phase is dominated by the number of particles in the system and though the analysis for FPGA clusters considers eight times the number of particles as the single-FPGA analysis, the clusters outperform. This is due to the parallelism the cluster offers: once particle information is distributed to all nodes, each node only maps 16,384 particles, dividing the single-FPGA charge mapping run-time by the number of nodes in the system.

The downside of LR designs mapped to FPGA clusters can be seen when comparing the run-times of the FFT phases. The partitioning of the 64x64x64 grid into a 4x4 matrix of 16x16x64 grid-prisms allows 1D FFTs/IFFTs in the X dimension to be parallelized without the need for inter-node communication. This parallelism coupled with the smaller geometry of the grid-prism allows for comparable run-times for 1D FFT/IFFT in the X dimension across all design deployments. FFTs/IFFTs in the other dimensions expose the real drawback, but the pipelined nature of running the FFTs/IFFTs limits the detriment.

Despite the accommodation for more particles, the FPGA cluster deployments outperform the single-FPGA deployment with regards to force computations. The distribution of the most force calculations is the driving force behind this improvement. Each cluster node performs force computations over smaller, partial particle ensembles, while the network essentially serializes all the messages generated in parallel at higher clock frequencies. On the contrary, the single-FPGA deployment needs to account for all particles in the ensemble while the messaging bandwidth is limited by the single force computation pipeline.

FPGA cluster based deployments are clearly the best strategy to accelerate LR computations. The faster network speeds coupled with the distributed nature of the network allows particle information to get transferred faster. The distribution of particles divides charge mapping run-time by the number of nodes in the network. It also divides up the force computation to almost the same degree. The only drawback is the inter-node communication requirement during FFT phases. However, the loss in performance is on the order of tens of microseconds where as the benefits in other areas are on the order of 100s of microseconds. It should be mentioned that the per-node resource consumption can be smaller when compared to single-FPGA deployments, potentially resulting in higher operational clock frequencies. The dominance of particle information on LR run-times can only be alleviated by providing more bandwidth: either higher-speed networks, more links per node, more interconnection flexibility, or a combination thereof. Force computation messaging also contributes largely to the LR run-times and are dominated by the bandwidth capabilities of each node. Additional force computations per node has the potential to divide this run-time by the number of pipelines. Outside of messaging, charge mapping is the next most dominant components of LR run-times. Parallelizing charge mapping into multiple pipelines could also reduce the run-times by similar factors.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

In this thesis LR accelerators of varying size were implemented using commercially off-the-shelf FPGAs, with some of them capable of enabling MD simulations of hundreds of nanoseconds of reality per day of wall-clock time. This was made possible by:

1. Developing a custom memory architecture and controller to transpose 3D memory access topologies with each memory access. Doing so removes the need for wholesale data transposition between the several 1D FFT/IFFT phases of LR computation.

2. Developing a novel FFT pipeline strategy that exploits the full parallelism offered by the aforementioned custom memory architecture.

This single-FPGA architecture was then extended to FPGA cluster-based architectures to overcome both the capacity constraint inherent to a single-FPGA design and the performance constraints of FPGA design in general.

While full ASIC implementations of LR accelerators easily outperform their FPGA counterparts, they are cost-prohibitive when considering the technology nodes targeted for better performance and their associated mask cost. Not to mention the engineering effort required to build custom-sized memories, floorplan the design, implement Design for Test (DFT) logic, run design rule checks, verify the implemented design against the simulated design intent, and all the tool licensing costs associated with all these ASIC design tasks. FPGAs also have the flexibility of being reprogrammed, allowing for design enhancements and re-configurations without having to spin new silicon or at the very least incur the long turnaround times typical of ASIC design flows. With FPGA clusters becoming more publicly available, the cost of mapping designs to pre-existing FPGA networks is much more viable than creating a custom network whether choosing to use ASIC-based nodes or commercially available FPGA-based nodes.

The implementation and analysis of single-FPGA LR designs exposes areas critical to both performance and resource consumption, allowing for the identification of solutions in both of these areas, with the main solution being the distribution of the computation. To be clear, the distribution of computation does result in a many-fold increase in hardware consumption. Solutions that reduce hardware consumption do so at a per-FPGA level. Moving to FPGA cluster based deployments reduces the run-time of certain LR phases via parallelization, but is also cuts down on data transmission time to and from the accelerator hardware.

Analyzing different FPGA clusters in terms of network architectures also identifies solutions for performance enhancement of distributed LR acceleration. Better operational clock frequencies and network bandwidth can come in the form of newer, smaller, faster technologies or hardware with more features becoming commercially available. Additional performance increases can come in distributing the computation over more nodes, but it's the transmission of data to and from these nodes that dominates the run-time of LR iterations. The amount of data that needs to be transmitted can only be counteracted with more bandwidth.

7.2 Future Work

Building off of what was developed in the context of this thesis, the following is a list of future work that can be performed:

1. The charge mapping hardware needs to be updated to achieve near one-particle-per-

clock-cycle mapping performance. Currently the hardware stalls the pipeline which is detrimental to performance. At the very least, the stall logic can be removed under the assumption that particles are presorted to avoid pipeline hazards when mapping.

2. The grid clearing mechanism of the LR design should be enhanced to divide the time required to clear the grid by the number of access ports on the clustered grid memory. Doing so will no longer require LR iterations to have at least as many particles as there are grid-points. Such a constraint becomes more cumbersome when considering that distributed LR computation networks could have nodes where the localized number of particles is less than the localized number of grid-points. The resultant 1:64 particle-to-grid-point ratio constraint seems reasonable. If not, then grid clearing must be somehow serialized inside LR iterations, thereby increasing computational run-times.

3. As grid sizes increase, the MUXing inside of the clustered grid memory and Green's ROM may find itself on the critical path. The one-hot MUXing employed in both those sub-modules may need to be replaced with faster MUX implementations.

4. To reduce the amount of resources consumed by the Green's ROM, the need for toroidal shift MUXIng in the X and Y dimensions should be removed. The Green's ROM is only ever accessed along the Z dimension, always in an aligned fashion.

5. The single-FPGA hardware should be integrated into a complete MD system, either on a single-FPGA (for demonstration purposes) or as part of a network of FPGAs (for more practical purposes). Doing so may result in architectural changes to ease integration or even fix real, previously unobserved issues with the design.

6. The FFT pipelining hardware should be updated to not constrain the configurability of the RTL such that the minimum grid size be the 16, four times the number of nearest-neighbors a particle has along one dimension.

7. The addition of charge mapping pipelines to the single-FPGA LR design should be explored. This has the opportunity to divide charge mapping run-times by the number of pipelines, but it does introduce issues when particles in each pipeline map to overlapping subsets of grid-points.

8. As with charge mapping pipelines, similar can be said for the force computation pipelines.

Appendix A

Ring Network Orthogonal FFT Messaging

Figure A·1 captures the results of simulating the orthogonal FFT messaging of the LR cluster described in Section 6.3. Each row in the table lists the result of the message relay in both the forward and reverse directions for all nodes in the ring. Each cell in the table contains a message identifier with the following format: $N\#M\#[L|S]$. *N#* is the source node identifier. *M#* is the message identifier respective to the source node. *L* refers to the *long message* type which gets transmitted to two nodes in the FFT line orthogonal to the primary dimension traversal of the ring. *S* refers to the *short message* type which gets transmitted to one node in the FFT line orthogonal to the primary dimension traversal of the ring.

Not shown, but implied from the third row of the table, are the types of messages each node sources initially. Messages that traverse the ring traverse the table diagonally, wrapping at node 1 in the reverse direction and 15 in the forward direction. Once the message arrives at its final destination, the destination node can then inject its next data point into the ring. After 8 message relay stages, the system reaches a steady-stage message pattern that repeats every 24 relays stages. It is observed that at the start of every repetitive pattern series, the range of message IDs across the system is 1. This maximum range limit means that no one node is contains any more incomplete FFT pencils than any other node and that each node is performing an FFT over the same message set within a couple of relay stages of each other. This relative synchronicity means that each node can be configured with the same adequately sized buffer for FFT points and that the network can be saturated with messages while each node is computing FFT results.

Node0		Node1		Node2		Node3		Node4		Node5		Node6		Node7		Node8		Node9		Node10		Node11		Node12		Node13		Node14		Node15	
Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse	Forward	Reverse
N15M0L	N01M0L	N00M0S	N02M0L	N01M0S	N03M0L	N02M0S	N04M0S	N03M0S	N05M0S	N04M0L	N06M0S	N05M0L	N07M0S	N06M0L	N08M0L	N07M0L	N09M0L	N08M0S	N10M0L	N09M0S	N11M0L	N10M0S	N12M0S	N11M0S	N13M0S	N12M0L	N14M0S	N13M0L	N15M0S	N14M0L	N00M0L
N14M0L	N02M0L	N15M0L	N03M0L	N00M0S	N03M1S	N01M0S	N05M0S	N02M0S	N06M0S	N04M1S	N07M0S	N04M0L	N08M0L	N05M0L	N09M0L	N06M0L	N10M0L	N07M0L	N11M0L	N08M0S	N11M1S	N09M0S	N13M0S	N10M0S	N14M0S	N12M1S	N15M0S	N12M0L	N00M0L	N13M0L	N01M0L
N13M0L	N03M0L	N14M0L	N03M1S	N15M0L	N05M0S	N00M0S	N06M0S	N01M0S	N07M0S	N02M0S	N08M0L	N04M1S	N09M0L	N04M0L	N10M0L	N05M0L	N11M0L	N06M0L	N11M1S	N07M0L	N13M0S	N08M0S	N14M0S	N09M0S	N15M0S	N10M0S	N00M0L	N12M1S	N01M0L	N12M0L	N02M0L
N12M0L	N03M1S	N13M0L	N02M1S	N14M0L	N06M0S	N15M0L	N07M0S	N00M0S	N08M0L	N01M0S	N09M0L	N05M1S	N10M0L	N04M1S	N11M0L	N04M0L	N11M1S	N05M0L	N10M1S	N06M0L	N14M0S	N07M0L	N15M0S	N08M0S	N00M0L	N09M0S	N01M0L	N13M1S	N02M0L	N12M1S	N03M0L
N12M1S	N02M1S	N12M0L	N06M0S	N13M0L	N07M0S	N14M0L	N08M0L	N15M0L	N09M0L	N00M0S	N10M0L	N01M0S	N11M0L	N05M1S	N11M1S	N04M1S	N10M1S	N04M0L	N14M0S	N05M0L	N15M0S	N06M0L	N00M0L	N07M0L	N01M0L	N08M0S	N02M0L	N09M0S	N03M0L	N13M1S	N03M1S
N13M1S	N01M1S	N12M1S	N07M0S	N12M0L	N08M0L	N13M0L	N09M0L	N14M0L	N10M0L	N15M0L	N11M0L	N00M0S	N11M1S	N06M1S	N10M1S	N05M1S	N09M1S	N04M1S	N15M0S	N04M0L	N00M0L	N05M0L	N01M0L	N06M0L	N02M0L	N07M0L	N03M0L	N08M0S	N03M1S	N14M1S	N02M1S
N14M1S	N07M0S	N13M1S	N08M0L	N12M1S	N09M0L	N12M0L	N10M0L	N13M0L	N11M0L	N14M0L	N11M1S	N15M0L	N10M1S	N00M0S	N09M1S	N06M1S	N15M0S	N05M1S	N00M0L	N04M1S	N01M0L	N04M0L	N02M0L	N05M0L	N03M0L	N06M0L	N03M1S	N07M0L	N02M1S	N08M0S	N01M1S
N15M1S	N08M0L	N14M1S	N09M0L	N13M1S	N10M0L	N12M1S	N11M0L	N12M0L	N11M1S	N13M0L	N10M1S	N14M0L	N09M1S	N15M0L	N08M1S	N07M1S	N00M0L	N06M1S	N01M0L	N05M1S	N02M0L	N04M1S	N03M0L	N04M0L	N03M1S	N05M0L	N02M1S	N06M0L	N01M1S	N07M0L	N00M1S
N15M2L	N01M2L	N00M1L	N02M2L	N01M1L	N03M2L	N02M1L	N04M1L	N03M1L	N05M1L	N04M2L	N06M1L	N05M2L	N07M1L	N06M2L	N08M2L	N07M2L	N09M2L	N08M1L	N10M2L	N09M1L	N11M2L	N10M1L	N12M1L	N11M1L	N13M1L	N12M2L	N14M1L	N13M2L	N15M1L	N14M2L	N00M2L
N14M2L	N02M2L	N15M2L	N03M2L	N00M1L	N04M1L	N01M1L	N05M1L	N02M1L	N06M1L	N03M1L	N07M1L	N04M2L	N08M2L	N05M2L	N09M2L	N06M2L	N10M2L	N07M2L	N11M2L	N08M1L	N12M1L	N09M1L	N13M1L	N10M1L	N14M1L	N11M1L	N15M1L	N12M2L	N00M2L	N13M2L	N01M2L
N13M2L	N03M2L	N14M2L	N04M1L	N15M2L	N05M1L	N00M1L	N06M1L	N01M1L	N07M1L	N02M1L	N08M2L	N03M1L	N09M2L	N04M2L	N10M2L	N05M2L	N11M2L	N06M2L	N12M1L	N07M2L	N13M1L	N08M1L	N14M1L	N09M1L	N15M1L	N10M1L	N00M2L	N11M1L	N01M2L	N12M2L	N02M2L
N12M2L	N04M1L	N13M2L	N05M1L	N14M2L	N06M1L	N15M2L	N07M1L	N00M1L	N08M2L	N01M1L	N09M2L	N02M1L	N10M2L	N03M1L	N11M2L	N04M2L	N12M1L	N05M2L	N13M1L	N06M2L	N14M1L	N07M2L	N15M1L	N08M1L	N00M2L	N09M1L	N01M2L	N10M1L	N02M2L	N11M1L	N03M2L
N11M1L	N05M1L	N12M2L	N06M1L	N13M2L	N07M1L	N14M2L	N08M2L	N15M2L	N09M2L	N00M1L	N10M2L	N01M1L	N11M2L	N02M1L	N12M1L	N03M1L	N13M1L	N04M2L	N14M1L	N05M2L	N15M1L	N06M2L	N00M2L	N07M2L	N01M2L	N08M1L	N02M2L	N09M1L	N03M2L	N10M1L	N04M1L
N10M1L	N06M1L	N11M1L	N07M1L	N12M2L	N08M2L	N13M2L	N09M2L	N14M2L	N10M2L	N15M2L	N11M2L	N00M1L	N12M1L	N01M1L	N13M1L	N02M1L	N14M1L	N03M1L	N15M1L	N04M2L	N00M2L	N05M2L	N01M2L	N06M2L	N02M2L	N07M2L	N03M2L	N08M1L	N04M1L	N09M1L	N05M1L
N09M1L	N07M1L	N10M1L	N08M2L	N11M1L	N09M2L	N12M2L	N10M2L	N13M2L	N11M2L	N14M2L	N12M1L	N15M2L	N13M1L	N00M1L	N14M1L	N01M1L	N15M1L	N02M1L	N00M2L	N03M1L	N01M2L	N04M2L	N02M2L	N05M2L	N03M2L	N06M2L	N04M1L	N07M2L	N05M1L	N08M1L	N06M1L
N08M1L	N08M2L	N09M1L	N09M2L	N10M1L	N10M2L	N11M1L	N11M2L	N12M2L	N12M1L	N13M2L	N13M1L	N14M2L	N14M1L	N15M2L	N15M1L	N00M1L	N00M2L	N01M1L	N01M2L	N02M1L	N02M2L	N03M1L	N03M2L	N04M2L	N04M1L	N05M2L	N05M1L	N06M2L	N06M1L	N07M2L	N07M1L
N15M3S	N01M3S	N00M2S	N02M3S	N01M2S	N03M3S	N02M2S	N04M2S	N03M2S	N05M2S	N04M3S	N06M2S	N05M3S	N07M2S	N06M3S	N08M3S	N07M3S	N09M3S	N08M2S	N10M3S	N09M2S	N11M3S	N10M2S	N12M2S	N11M2S	N13M2S	N12M3S	N14M2S	N13M3S	N15M2S	N14M3S	N00M3S
N14M3S	N02M3S	N00M3L	N03M3S	N00M2S	N03M4L	N01M2S	N05M2S	N02M2S	N06M2S	N04M4L	N07M2S	N04M3S	N07M3L	N05M3S	N09M3S	N06M3S	N10M3S	N08M3L	N11M3S	N08M2S	N11M4L	N09M2S	N13M2S	N10M2S	N14M2S	N12M4L	N15M2S	N12M3S	N15M3L	N13M3S	N01M3S
N13M3S	N03M3S	N14M3S	N03M4L	N00M3L	N05M2S	N00M2S	N06M2S	N01M2S	N07M2S	N02M2S	N07M3L	N04M4L	N09M3S	N04M3S	N10M3S	N05M3S	N11M3S	N06M3S	N11M4L	N08M3L	N13M2S	N08M2S	N14M2S	N09M2S	N15M2S	N10M2S	N15M3L	N12M4L	N01M3S	N12M3S	N02M3S
N12M3S	N03M4L	N13M3S	N02M4L	N01M3L	N06M2S	N00M3L	N07M2S	N00M2S	N07M3L	N01M2S	N06M3L	N05M4L	N10M3S	N04M4L	N11M3S	N04M3S	N11M4L	N05M3S	N10M4L	N09M3L	N14M2S	N08M3L	N15M2S	N08M2S	N15M3L	N09M2S	N14M3L	N13M4L	N02M3S	N12M4L	N03M3S
N12M4L	N02M4L	N12M3S	N06M2S	N13M3S	N07M2S	N01M3L	N07M3L	N00M3L	N06M3L	N00M2S	N10M3S	N01M2S	N11M3S	N05M4L	N11M4L	N04M4L	N10M4L	N04M3S	N14M2S	N05M3S	N15M2S	N09M3L	N15M3L	N08M3L	N14M3L	N08M2S	N02M3S	N09M2S	N03M3S	N13M4L	N03M4L
N13M4L	N01M4L	N12M4L	N07M2S	N12M3S	N07M3L	N02M3L	N06M3L	N01M3L	N05M3L	N00M3L	N11M3S	N00M2S	N11M4L	N06M4L	N10M4L	N05M4L	N09M4L	N04M4L	N15M2S	N04M3S	N15M3L	N10M3L	N14M3L	N09M3L	N13M3L	N08M3L	N03M3S	N08M2S	N03M4L	N14M4L	N02M4L
N14M4L	N07M2S	N13M4L	N07M3L	N12M4L	N06M3L	N12M3S	N05M3L	N02M3L	N11M3S	N01M3L	N11M4L	N00M3L	N10M4L	N00M2S	N09M4L	N06M4L	N15M2S	N05M4L	N15M3L	N04M4L	N14M3L	N04M3S	N13M3L	N10M3L	N03M3S	N09M3L	N03M4L	N08M3L	N02M4L	N08M2S	N01M4L
N15M4L	N07M3L	N14M4L	N06M3L	N13M4L	N05M3L	N12M4L	N04M3L	N03M3L	N11M4L	N02M3L	N10M4L	N01M3L	N09M4L	N00M3L	N08M4L	N07M4L	N15M3L	N06M4L	N14M3L	N05M4L	N13M3L	N04M4L	N12M3L	N11M3L	N03M4L	N10M3L	N02M4L	N09M3L	N01M4L	N08M3L	N00M4L
N08M3L	N06M3L	N15M4L	N05M3L	N14M4L	N04M3L	N13M4L	N11M4L	N12M4L	N10M4L	N03M3L	N09M4L	N02M3L	N08M4L	N01M3L	N15M3L	N00M3L	N14M3L	N07M4L	N13M3L	N06M4L	N12M3L	N05M4L	N03M4L	N04M4L	N02M4L	N11M3L	N01M4L	N10M3L	N00M4L	N09M3L	N07M3L
N09M3L	N05M3L	N00M4S	N04M3L	N15M4L	N03M5S	N14M4L	N10M4L	N13M4L	N09M4L	N04M5S	N08M4L	N03M3L	N07M4S	N02M3L	N14M3L	N01M3L	N13M3L	N08M4S	N12M3L	N07M4L	N11M5S	N06M4L	N02M4L	N05M4L	N01M4L	N12M5S	N00M4L	N11M3L	N15M4S	N10M3L	N06M3L
N10M3L	N04M3L	N09M3L	N03M5S	N00M4S	N10M4L	N15M4L	N09M4L	N14M4L	N08M4L	N13M4L	N07M4S	N04M5S	N14M3L	N03M3L	N13M3L	N02M3L	N12M3L	N01M3L	N11M5S	N08M4S	N02M4L	N07M4L	N01M4L	N06M4L	N00M4L	N05M4L	N15M4S	N12M5S	N06M3L	N11M3L	N05M3L
N11M3L	N03M5S	N10M3L	N02M5S	N01M4S	N09M4L	N00M4S	N08M4L	N15M4L	N07M4S	N14M4L	N06M4S	N05M5S	N13M3L	N04M5S	N12M3L	N03M3L	N11M5S	N02M3L	N10M5S	N09M4S	N01M4L	N08M4S	N00M4L	N07M4L	N15M4S	N06M4L	N14M4S	N13M5S	N05M3L	N12M5S	N04M3L
N12M5S	N02M5S	N11M3L	N09M4L	N10M3L	N08M4L	N01M4S	N07M4S	N00M4S	N06M4S	N15M4L	N13M3L	N14M4L	N12M3L	N05M5S	N11M5S	N04M5S	N10M5S	N03M3L	N01M4L	N02M3L	N00M4L	N09M4S	N15M4S	N08M4S	N14M4S	N07M4L	N05M3L	N06M4L	N04M3L	N13M5S	N03M5S
N13M5S	N01M5S	N12M5S	N08M4L	N11M3L	N07M4S	N02M4S	N06M4S	N01M4S	N05M4S	N00M4S	N12M3L	N15M4L	N11M5S	N06M5S	N10M5S	N05M5S	N09M5S	N04M5S	N00M4L	N03M3L	N15M4S	N10M4S	N14M4S	N09M4S	N13M4S	N08M4S	N04M3L	N07M4L	N03M5S	N14M5S	N02M5S
N14M5S	N08M4L	N13M5S	N07M4S	N12M5S	N06M4S	N11M3L	N05M4S	N02M4S	N12M3L	N01M4S	N11M5S	N00M4S	N10M5S	N15M4L	N09M5S	N06M5S	N00M4L	N05M5S	N15M4S	N04M5S	N14M4S	N03M3L	N13M4S	N10M4S	N04M3L	N09M4S	N03M5S	N08M4S	N02M5S	N07M4L	N01M5S
N15M5S	N07M4S	N14M5S	N06M4S	N13M5S	N05M4S	N12M5S	N04M4S	N03M4S	N11M5S	N02M4S	N10M5S	N01M4S	N09M5S	N00M4S	N08M5S	N07M5S	N15M4S	N06M5S	N14M4S	N05M5S	N13M4S	N04M5S	N12M4S	N11M4S	N03M5S	N10M4S	N02M5S	N09M4S	N01M5S	N08M4S	N00M5S
N15M6L	N01M6L	N00M5L	N02M6L	N01M5L	N03M6L	N02M5L	N04M5L	N03M5L	N05M5L	N04M6L	N06M5L	N05M6L	N07M5L	N06M6L	N08M6L	N07M6L	N09M6L	N08M5L	N10M6L	N09M5L	N11M6L	N10M5L	N12M5L	N11M5L	N13M5L	N12M6L	N14M5L	N13M6L	N15M5L	N14M6L	N00M6L
N14M6L	N02M6L	N15M6L	N03M6L	N00M5L	N04M5L	N01M5L	N05M5L	N02M5L	N06M5L	N03M5L	N07M5L	N04M6L	N08M6L	N05M6L	N09M6L	N06M6L	N10M6L	N07M6L	N11M6L	N08M5L	N12M5L	N09M5L	N13M5L	N10M5L	N14M5L	N11M5L	N15M5L	N12M6L	N00M6L	N13M6L	N01M6L
N13M6L	N03M6L	N14M6L	N04M5L	N15M6L	N05M5L	N00M5L	N06M5L	N01M5L	N07M5L	N02M5L	N08M6L	N03M5L	N09M6L	N04M6L	N10M6L	N05M6L	N11M6L	N06M6L	N12M5L	N07M6L	N13M5L	N08M5L	N14M5L	N09M5L	N15M5L	N10M5L	N00M6L	N11M5L	N01M6L	N12M6L	N02M6L
N12M6L	N04M5L	N13M6L	N05M5L	N14M6L	N06M5L	N15M6L	N07M5L	N00M5L	N08M6L	N01M5L	N09M6L	N02M5L	N10M6L	N03M5L	N11M6L	N04M6L	N12M5L	N05M6L	N13M5L	N06M6L	N14M5L	N07M6L	N15M5L	N08M5L	N00M6L	N09M5L	N01M6L	N10M5L	N02M6L	N11M5L	N03M6L
N11M5L	N05M5L	N12M6L	N06M5L	N13M6L	N07M5L	N14M6L	N08M6L	N15M6L	N09M6L	N00M5L	N10M6L	N01M5L	N11M6L	N02M5L	N12M5L	N03M5L	N13M5L	N04M6L	N14M5L	N05M6L	N15M5L	N06M6L	N00M6L	N07M6L	N01M6L	N08M5L	N02M6L	N09M5L	N03M6L	N10M5L	N04M5L
N10M5L	N06M5L	N11M5L	N07M5L	N12M6L	N08M6L	N13M6L	N09M6L	N14M6L	N10M6L	N15M6L	N11M6L	N00M5L	N12M5L	N01M5L	N13M5L	N02M5L	N14M5L	N03M5L	N15M5L	N04M6L	N00M6L	N05M6L	N01M6L	N06M6L	N02M6L	N07M6L	N03M6L	N08M5L	N04M5L	N09M5L	N05M5L
N09M5L	N07M5L	N10M5L	N08M6L	N11M5L	N09M6L	N12M6L	N10M6L	N13M6L	N11M6L	N14M6L	N12M5L	N15M6L	N13M5L	N00M5L	N14M5L	N01M5L	N15M5L	N02M5L	N00M6L	N03M5L	N01M6L	N04M6L	N02M6L	N05M6L	N03M6L	N06M6L	N04M5L	N07M6L	N05M5L	N08M5L	N06M5L
N08M5L	N08M6L	N09M5L	N09M6L	N10M5L	N10M6L	N11M5L	N11M6L	N12M6L	N12M5L	N13M6L	N13M5L	N14M6L	N14M5L	N15M6L	N15M5L	N00M5L	N00M6L	N01M5L	N01M6L	N02M5L	N02M6L	N03M5L	N03M6L	N04M6L	N04M5L	N05M6L	N05M5L	N06M6L	N06M5L	N07M6L	N07M5L
N15M7S	N01M7S	N00M6S	N02M7S	N01M6S	N03M7S	N02M6S	N04M6S	N03M6S	N05M6S	N04M7S	N06M6S	N05M7S	N07M6S	N06M7S	N08M7S	N07M7S	N09M7S	N08M6S	N10M7S	N09M6S	N11M7S	N10M6S	N12M6S	N11M6S	N13M6S	N12M7S	N14M6S	N13M7S	N15M6S	N14M7S	N00M7S
N14M7S	N02M7S	N00M7L	N03M7S	N00M6S	N03M8L	N01M6S	N05M6S	N02M6S	N06M6S	N04M8L	N07M6S	N04M7S	N07M7L	N05M7S	N09M7S	N06M7S	N10M7S	N08M7L	N11M7S	N08M6S	N11M8L	N09M6S	N13M6S	N10M6S	N14M6S	N12M8L	N15M6S	N12M7S	N15M7L	N13M7S	N01M7S
N13M7S	N03M7S	N14M7S	N03M8L	N00M7L	N05M6S	N00M6S	N06M6S	N01M6S	N07M6S	N02M6S	N07M7L	N04M8L	N09M7S	N04M7S	N10M7S	N05M7S	N11M7S	N06M7S	N11M8L	N08M7L	N13M6S	N08M6S	N14M6S	N09M6S	N15M6S	N10M6S	N15M7L	N12M8L	N01M7S	N12M7S	N02M7S
N12M7S	N03M8L	N13M7S	N02M8L	N01M7L	N06M6S	N00M7L	N07M6S	N00M6S	N07M7L	N01M6S	N06M7L	N05M8L	N10M7S	N04M8L	N11M7S	N04M7S	N11M8L	N05M7S	N10M8L	N09M7L	N14M6S	N08M7L	N15M6S	N08M6S	N15M7L	N09M6S	N14M7L	N13M8L	N02M7S	N12M8L	N03M7S
N12M8L	N02M8L	N12M7S	N06M6S	N13M7S	N07M6S	N01M7L	N07M7L	N00M7L	N06M7L	N00M6S	N10M7S	N01M6S	N11M7S	N05M8L	N11M8L	N04M8L	N10M8L	N04M7S	N14M6S	N05M7S	N15M6S	N09M7L	N15M7L	N08M7L	N14M7L	N08M6S	N02M7S	N09M6S	N03M7S	N13M8L	N03M8L
N13M8L	N01M8L	N12M8L	N07M6S	N12M7S	N07M7L	N02M7L	N06M7L	N01M7L	N05M7L	N00M7L	N11M7S	N00M6S	N11M8L	N06M8L	N10M8L	N05M8L	N09M8L	N04M8L	N15M6S	N04M7S	N15M7L	N10M7L	N14M7L	N09M7L	N13M7L	N08M7L	N03M7S	N08M6S	N03M8L	N14M8L	N02M8L
N14M8L	N07M6S	N13M8L	N07M7L	N12M8L	N06M7L	N12M7S	N05M7L	N02M7L	N11M7S	N01M7L	N11M8L	N00M7L	N10M8L	N00M6S	N09M8L	N06M8L	N15M6S	N05M8L	N15M7L	N04M8L	N14M7L	N04M7S	N13M7L	N10M7L	N03M7S	N09M7L	N03M8L	N08M7L	N02M8L	N08M6S	N01M8L
N15M8L	N07M7L	N14M8L	N06M7L	N13M8L	N05M7L	N12M8L	N04M7L	N03M7L	N11M8L	N02M7L	N10M8L	N01M7L	N09M8L	N00M7L	N08M8L	N07M8L	N15M7L	N06M8L	N14M7L	N05M8L	N13M7L	N04M8L	N12M7L	N11M7L	N03M8L	N10M7L	N02M8L	N09M7L	N01M8L	N08M7L	N00M8L

Figure A·1: FFT Messaging Table for Ring-Connected 2D Matrix FPGA Cluster

References

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

Allen, M. and Tildesley, D. (1987). *Computer Simulation of Liquids*. Oxford Science Publications, Oxford, UK.

AMD (2020). AMD High Bandwidth Memory. https://www.amd.com/en/ technologies/hbm.

Azizi, N., Kuon, I., Egier, A., Darabiha, A., and Chow, P. (2004). Reconfigurable molecular dynamics simulator. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, pages 197–206. DOI: 10.1109/FCCM.2004.48.

Benkrid, K. and Vanderbauwhede, W., editors (2013). *High Performance Computing Using FPGAs*. Springer Verlag. doi: 10.1007/978-1-4614-1791-0_4.

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

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

Chiu, M. and Herbordt, M. (2009). Efficient filtering for molecular dynamics simulations. In *2009 International Conference on Field Programmable Logic and Applications*. doi: 10.1109/ FPL15426.2009.

Chiu, M. and Herbordt, M. (2010a). Molecular dynamics simulations on high performance reconfigurable computing systems. *ACM Transactions on Reconfigurable Technology and Systems*, 3(4):1–37. doi: 10.1145/1862648.1862653.

Chiu, M. and Herbordt, M. (2010b). Towards production FPGA-accelerated molecular dynamics: Progress and challenges. In *2010 4th High Performance Reconfigurable Technology and Applications*. doi: 10.1109/HPRCTA.2010.5670800.

Chiu, M., Herbordt, M., and Langhammer, M. (2008). Performance potential of molecular dynamics simulations on high performance reconfigurable computing systems. In *2008 Second International Workshop on High-Performance Reconfigurable Computing Technology and Applications*. doi: 10.1109/ HPRCTA.2008.4745685.

Chiu, M., Khan, M., and Herbordt, M. (2011). Efficient calculation of pairwise nonbonded forces. In *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. doi: 10.1109/ FCCM.2011.34.

Cournia, Z., Allen, B., and Sherman, W. (2017). Relative binding free energy calculations in drug discovery: Recent advances and practical considerations. *Journal of Chemical Information and Modeling*, 57:2911–2937.

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

Essmann, U., Perera, L., Berkowitz, M., Darden, T., Lee, H., and Pedersen, L. (1995). A smooth particle mesh Ewald method. *The Journal of Chemical Physics*, 103:8577–8593.

Frenkel, D. and Smit, B. (2002). *Understanding Molecular Simulation*. Academic Press, New York, NY.

Geng, T., Li, A., Shi, R., Wu, C., Wang, T., Li, Y., Haghi, P., Tumeo, A., Che, S., Reinhardt, S., and Herbordt, M. (2020a). AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing. In *53rd IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

Geng, T., Wang, T., Li, A., Jin, X., and Herbordt, M. (2019a). A Scalable Framework for Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters with Weight and Workload Balancing. In *ArXiv Preprint arXiv:1901.01007*.

Geng, T., Wang, T., Sanaullah, A., Yang, C., Xuy, R., Patel, R., and Herbordt, M. (2018a). A framework for acceleration of CNN training on deeply-pipelined FPGA clusters with work and weight load balancing. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL 2018): 394–402*. doi: 10.1109/ FPL.2018. 00074.

Geng, T., Wang, T., Sanaullah, A., Yang, C., Xuy, R., Patel, R., and Herbordt, M. (2018b). FPDeep: Acceleration and Load Balancing of CNN Training on FPGA Clusters. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, page 81–84. doi: 10.1109/ FCCM.2018. 00021.

Geng, T., Wang, T., Wu, C., Li, Y., Yang, C., Wu, W., Li, A., and Herbordt, M. (2021). O3BNN-R: An Out-Of-Order Architecture for High-Performance and Regularized BNN Inference. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):199–213. doi: 10.1109/TPDS.2020.3013637.

Geng, T., Wang, T., Wu, C., Yang, C., Li, A., Song, S., and Herbordt, M. (2019b). LP-BNN: Ultra-low-Latency BNN Inference with Layer Parallelism. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, volume 2160, pages 9–16. doi: 10.1109/ASAP.2019.00-43.

Geng, T., Wang, T., Wu, C., Yang, C., Wu, W., Li, A., and Herbordt, M. (2019c). O3BNN: An Out-Of-Order Architecture for High-Performance Binarized Neural Network Inference with Fine-Grained Pruning. In *ACM International Conference on Supercomputing*, volume 2160, pages 461–472. doi: 10.1145/ 3330345. 3330386.

Geng, T., Wu, C., Tan, C., Fang, B., Li, A., and Herbordt, M. (2020b). CQNN: a CGRA-based QNN Framework. In *IEEE High Performance Extreme Computing Conference*. doi:TBD.

George, A., Herbordt, M., Lam, H., Lawande, A., Sheng, J., and Yang, C. (2016). 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), Waltham, MA*, pages 1–7. doi: 10.1109/ HPEC.2016. 7761639.

Gokhale, M. and Graham, P. (2005). *Reconfigurable Computing: Accelerating Computation with Field Programmable Gate Arrays*. Springer.

Grossman, J., Towles, B., Greskamp, B., and Shaw, D. (2015). Filtering, reductions and synchronization in the Anton 2 network. In *Proc. International Parallel and Distributed Processing Symposium*, pages 860 – 870. DOI: 10.1109/IPDPS.2015.42.

Gu, Y. and Herbordt, M. (2007). FPGA-based multigrid computations for molecular dynamics simulations. In *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 117–126. doi: 10.1109/ FCCM.2007.42.

Gu, Y., VanCourt, T., DiSabello, D., and Herbordt, M. (2005a). FPGA acceleration of molecular dynamics computations. In *IEEE Symposium on Field Programmable Custom Computing Machines*. DOI: 10.1109/FCCM.2005.54.

Gu, Y., VanCourt, T., and Herbordt, M. (2005b). Accelerating molecular dynamics simulations with configurable circuits. In *IEEE Conference on Field Programmable Logic and Applications*. DOI: 10.1109/FPL.2005.1515767.

Gu, Y., VanCourt, T., and Herbordt, M. (2006a). Accelerating molecular dynamics simulations with configurable circuits. *IEE Proceedings on Computers and Digital Technology*, 153(3):189–195. doi: 10.1049/ip-cdt:20050182.

Gu, Y., VanCourt, T., and Herbordt, M. (2006b). Improved interpolation and system integration for FPGA-based molecular dynamics simulations. In *2006 International Conference on Field Programmable Logic and Applications*, pages 21–28. doi: 10.1109/ FPL.2006.311190.

Gu, Y., VanCourt, T., and Herbordt, M. (2006c). Integrating FPGA acceleration into the ProtoMol molecular dynamics code: Preliminary report. In *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 315–316. doi: 10.1109/ FCCM.2006.52.

Gu, Y., VanCourt, T., and Herbordt, M. (2008). Explicit design of FPGA-based coprocessors for short-range force computation in molecular dynamics simulations. *Parallel Computing*, 34(4-5):261–271. doi: 10.1016/j.parco.2008.01.007.

Haghi, P., Geng, T., Guo, A., Wang, T., and Herbordt, M. (2020a). A Reconfigurable Compute-in-the-Network FPGA Assistant for High-Level Collective Support with Distributed Matrix Multiply Case Study. In *IEEE Conference on Field Programmable Technology*.

Haghi, P., Geng, T., Guo, A., Wang, T., and Herbordt, M. (2020b). FP-AMG: FPGA-Based Acceleration Framework for Algebraic Multigrid Solvers. In *28th IEEE International Symposium on Field-Programmable Custom Computing Machines*. DOI: 10.1109/ FCCM48280.2020.00028.

Haghi, P., Guo, A., Xiong, Q., Patel, R., Yang, C., Geng, T., Broaddus, J., Marshall, R., Skjellum, A., and Herbordt, M. (2020c). FPGAs in the Network and Novel Communicator Support Accelerate MPI Collectives. In *IEEE High Performance Extreme Computing Conference*.

Haile, J. (1997). *Molecular Dynamics Simulation*. Wiley, New York, NY.

Hamada, T. and Nakasato, N. (2005). Massively parallel processors generator for reconfigurable system. *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*. DOI: 10.1109/FCCM.2005.45.

Hauck, S. and DeHon, A. (2008). *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing*. Morgan Kaufmann.

Herbordt, M. (2013). Architecture/algorithm codesign of molecular dynamics processors. In *2013 Asilomar Conference on Signals, Systems, and Computers*, pages 1442–1446. doi: 10.1109/ ACSSC.2013.6810534.

Herbordt, M. (2019). Advancing OpenCL for FPGAs: Boosting Performance with Intel FPGA SDK for OpenCL Technology. In *The Parallel Universe*, pages 17–32.

Herbordt, M., Gu, Y., VanCourt, T., Model, J., Sukhwani, B., and Chiu, M. (2008a). Computing models for FPGA-based accelerators with case studies in molecular modeling. *Computing in Science and Engineering*, 10(6):35–45. doi: 10.1109/ MCSE.2008.143.

Herbordt, M., Khan, M., and Dean, T. (2009). Parallel discrete event simulation of molecular dynamics through event-based decomposition. In *In 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, Boston, MA*, pages 129–136. doi: 10.1109/ ASAP.2009.39.

Herbordt, M., Kosie, F., and Model, J. (2008b). An efficient $O(1)$ priority queue for large FPGA-based discrete event simulations of molecular dynamics. In *In 2008 16th International Symposium on Field-Programmable Custom Computing Machines*, pages 248–257. doi: 10.1109/ FCCM.2008.49.

Herbordt, M., VanCourt, T., Gu, Y., Sukhwani, B., Conti, A., Model, J., and DiSabello, D. (2007). Achieving high performance with FPGA-based computing. *IEEE Computer*, 40(3):42–49.

Humphries, B., Zhang, H., Sheng, J., Landaverde, R., and Herbordt, M. (2014). 3D FFT on a Single FPGA. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. doi: 10.1109/ FCCM.2014.28.

Intel (2017). *FFT IP Core User Guide*. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/ literature/ug/ug_fft.pdf accessed 12/2/2020.

Intel (2020a). *Floating-Point IP Cores User Guide*. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_altfp_mfug.pdf accessed 12/2/2020.

Intel (2020b). Intel® Stratix 10 MX (DRAM System-in-Package) Device Overview. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10-mx-overview.pdf.

Khan, M., Chiu, M., and Herbordt, M. (2013). FPGA-Accelerated Molecular Dynamics. In Benkrid, K. and Vanderbauwhede, W., editors, *High Performance Computing Using FPGAs*, pages 105–135. Springer Verlag. doi: 10.1007/978-1-4614-1791-0_4.

Khan, M. and Herbordt, M. (2011). Parallel discrete event simulation of molecular dynamics with speculation and in-order commitment. *Journal of Computational Physics*, 230(17):6563–6582. doi: 10.1016/j.jcp.2011.05.001.

Khan, M. and Herbordt, M. (2012). Communication requirements for FPGA-centric molecular dynamics. In *Symposium on Application Accelerators for High Performance Computing*. https:// www.bu.edu/ caadlab/saahpc12.pdf.

Kindratenko, V. and Pointer, D. (2006). A case study in porting a production scientific supercomputing application to a reconfigurable computer. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, pages 13–22. DOI: 10.1109/FCCM.2006.5.

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

Lee, S. (2005). An FPGA Implementation of the Smooth Particle Mesh Ewald Reciprocal Sum Compute Engine. Master's thesis, U. Toronto.

Li, A., Geng, T., Wang, T., Herbordt, M., Song, S., and Barker, K. (2019). BSTC: A Novel Binarized-Soft-Tensor-Core Design for Accelerating Bit-Based Approximated Neural Nets. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. doi: 10.1145/ 3295500.3356169.

Liu, Y., Sheng, J., and Herbordt, M. (2016). A Hardware Prototype for In-Brain Neural Spike-Sorting. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. doi: 10.1109/HPEC.2016.7761590.

Lyons, R. (2015). Four Ways to Compute an Inverse FFT Using the Forward FFT Algorithm. https://www.dsprelated.com/ showarticle/800.php.

Miyajima, T., Ueno, T., Koshiba, A., Huthmann, J., Sano, K., and Sato, M. (2018). High-Performance Custom Computing with FPGA Cluster as an Off-loading Engine. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. https://sc19. supercomputing.org/ proceedings/ tech_poster/ poster_files/ rpost174s2-file3.pdf.

Model, J. and Herbordt, M. (2007). Discrete event simulation of molecular dynamics with configurable logic. In *2007 International Conference on Field Programmable Logic and Applications*, pages 151–158. doi: 10.1109/FPL.2007.4380640.

NVIDIA (2017). *Molecular Dynamics (MD) on GPUs*. NVIDIA, https://images.nvidia .com/content/tesla/pdf/ Molecular-Dynamics-July-2017-MB-slides.pdf.

Ohmura, I., Morimoto, G., Ohno, Y., Hasegawa, A., and Taiji, M. (2014). MDGRAPE-4: a special purpose computer system for molecular dynamics simulations. *Philosophical Transactions of the Royal Society A*, 372(20130387).

OpenMM (2020). Documentation Resources for finding your way around OpenMM. http:// openmm.org/ documentation.html.

Pascoe, C., Lawande, A., Lam, H., George, A., Sun, Y., Farmerie, W., and Herbordt, M. (2010). Reconfigurable supercomputing with scalable systolic arrays and in-stream control for wavefront genomics processing. In *Symposium on Application Accelerators for High Performance Computing*. https: //www.researchgate.net/ publication/ 265931244_ Reconfigurable_ Supercomputing_ with_ Scalable_ Systolic_ Arrays_ and_ In-Stream _Control_ for_Wavefront_ Genomics_ Processing.

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

Phillips, J., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R., Kale, L., and Schulten, K. (2005). Scalable molecular dynamics with NAMD. *Journal Computational Chemistry*, 26:1781–1802.

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

Plimpton, S. (1995). Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19.

Procacci, P. (2009). The smooth particle mesh ewald method. http://www.chim. unifi.it/orac/MAN/node20.html.

Proctor, E., Ding, F., and Dokholyan, N. (2011). Discrete molecular dynamics. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(1):80–92.

Rapaport, D. (2004). *The Art of Molecular Dynamics Simulation*. Cambridge University Press.

Samsung (2019). Samsung Electronics Introduces New High Bandwidth Memory Technology Tailored to Data Centers, Graphic Applications, and AI. https: //www.samsung .com/semiconductor/newsroom/tech-leadership/samsung-electronics-introduces-new-high-bandwidth-memory-technology-tailored-to-data-centers-graphic-applications-and-ai.

Sanaullah, A. and Herbordt, M. (2017). OpenCL for HPC/FPGAs: Case Study with 3D FFT. In *9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, page 1–6. doi: 10.1145/3241793.3241800.

Sanaullah, A. and Herbordt, M. (2018a). An Empirically Guided Optimization Framework for FPGA OpenCL. In *2018 International Conference on Field Programmable Technology (FPT)*, pages 46–53. doi: 10.1109/FPT.2018.00018.

Sanaullah, A. and Herbordt, M. (2018b). FPGA HPC using OpenCL: Case Study in 3D FFT. In *9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, page 1–6. doi: 10.1145/3241793.3241800.

Sanaullah, A. and Herbordt, M. (2018c). Unlocking Performance-Programmability by Penetrating the Intel FPGA OpenCL Toolflow. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*. doi: 10.1109/HPEC.2018.8547646.

Sanaullah, A., Khoshparvar, A., and Herbordt, M. (2016a). FPGA-Accelerated Particle-Grid Mapping. In *IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 192–195. doi: 10.1109/ FCCM .2016.53.

Sanaullah, A., Lewis, K., and Herbordt, M. (2016b). Accelerated Particle-Grid Mapping. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. http://sc16.supercomputing.org/sc-archive/tech_poster/ poster_files/ post257s2-file3.pdf.

Sanaullah, A., Lewis, K., and Herbordt, M. (2016c). GPU Accelerated Particle-Grid Mapping. In *IEEE High Performance Extreme Computing Conference*. DOI: 10.1109/ HPEC.2016.7761599.

Sanaullah, A., Sachdeva, V., and Herbordt, M. (2018a). SimBSP: Enabling RTL Simulation for Intel FPGA OpenCL Kernels. In *Proc. Heterogeneous High Performance Reconfigurable Computing*. doi: 10.1186/s12859-018-2505-7.

Sanaullah, A., Yang, C., Alexeev, Y., Yoshii, K., and Herbordt, M. (2018b). Real-Time Data Analysis for Medical Diagnosis using FPGA Accelerated Neural Networks. *BMC Bioinformatics*, 19 Supplement 18. doi: 10.1186/s12859-018-2505-7.

Schaffner, M. and Benini, L. (2018). On the feasibility of fpga acceleration of molecular dynamics simulations. Technical report, ArXiv:1808.04201.

Schlick, T. (2002). *Molecular Modeling and Simulation: An Interdisciplinary Guide*. Springer Verlag, New York, NY.

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

Sharma, S., Ding, F., and Dokholyan, N. (2007). Multiscale modeling of nucleosome dynamics. *Biophysical Journal*, 92:1457–1470.

Shaw, D.E., et al. (2007). Anton, a special-purpose machine for molecular dynamics simulation. In *Proceedings of the International Symposium on Computer Architecture*, pages 1–12. https:// doi.org/ 10.1145/ 1364782.1364802.

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

Sheng, J., Xiong, Q., Yang, C., and Herbordt, M. (2017a). Collective Communication on FPGA Clusters with Static Scheduling. *ACM SIGARCH Computer Architecture News*, 44(4). doi: 10.1145/ 3039902.3039904.

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

Sheng, J., Yang, C., and Herbordt, M. (2015). Towards Low-Latency Communication on FPGA Clusters with 3D FFT Case Study. In *International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*. https:// pdfs.semanticscholar.org /832d/ c69145f5ba0ed6a951583201b1b20dd 2096e.pdf.

Shi, R., Dong, P., Geng, T., Ding, Y., Ma, X., So, H., Herbordt, M., Li, A., and Wang, Y. (2020). CSB-RNN: A Faster-than-Realtime RNN Acceleration Framework with Compressed Structured Blocks. In *International Conference on Supercomputing*.

Shirvanyants, D., Ding, F., Tsao, D., Ramanchandran, S., and Dokholyan, N. (2012). DMD: an efficient and versatile simulation method for fine protein characterization. *Journal of Physical Chemistry B*, 116(29):8375–8382.

Skeel, R., Tezcan, I., and Hardy, D. (2002). Multiple grid methods for classical molecular dynamics. *Journal Computational Chemistry*, 23:673–684.

Stern, J., Xiong, Q., Skjellum, A., and Herbordt, M. (2018). A Novel Approach to Supporting Communicators for In-Switch Processing of MPI Collectives. In *Workshop on Exascale MPI*. https://www.bu.edu/caadlab/ ExaMPI18a.pdf.

Stewart, L., Pascoe, C., Sherman, B., Herbordt, M., and Sachdeva, V. (2020). An OpenCL 3D FFT for Molecular Dynamics distributed across multiple FPGAs. In *ArXiv Preprint arXiv:2009.12617*.

Sukhwani, B. and Herbordt, M. (2008). Acceleration of a Production Rigid Molecule Docking Code. In *2008 International Conference on Field Programmable Logic and Applications*, pages 341–346. doi: 10.1109/ FPL.2008.4629955.

Sukhwani, B. and Herbordt, M. (2009). FPGA-Acceleration of CHARMM Energy Minimization. In *Third International Workshop on High-Performance Reconfigurable Computing Technology and Applications*, page 1–10. doi: 10.1145/ 1646461.1646462.

Sukhwani, B. and Herbordt, M. (2010). FPGA Acceleration of Rigid Molecule Docking Codes. *IET Computers and Digital Techniques*, 4(3):184–195. doi: 10.1049/ iet-cdt.2009.0013.

SVLRM (2018). IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language. *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)*, pages 1–1315.

Tajkhorshid, E., Aksimentiev, A., Balabin, I., Gao, M., Isralewitz, B., Phillips, J., Zhu, F., and Schulten, K. (2003). Large scale simulation of protein mechanics and function. *Adv. Protein Chemistry*, 66:195–247.

van der Spoel, D., Lindahl, E., Hess, B., Groenhof, G., Mark, A., and Berendsen, H. (2005). GROMACS: fast, flexible, and free. *Journal Computational Chemistry*, 26:1701–1718.

VanCourt, T., Gu, Y., and Herbordt, M. (2004). FPGA acceleration of rigid molecule interactions. In *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 300–301. doi: 10.1109/ FCCM.2004.33.

VanCourt, T. and Herbordt, M. (2005). Three dimensional template correlation: Object recognition in 3D voxel data. In *Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05)*, pages 153–158. doi: 10.1109/ CAMP.2005.52.

VanCourt, T. and Herbordt, M. (2006a). Application-dependent memory interleaving enables high performance in FPGA-based grid computations. In *IEEE Conference on Field Programmable Logic and Applications*, pages 395–401. doi: 10.1109/ FCCM.2006.25.

VanCourt, T. and Herbordt, M. (2006b). Rigid molecule docking: FPGA reconfiguration for alternative force laws. *Journal on Applied Signal Processing*, v2006:1–10. doi: 10.1155/ ASP/2006/97950.

VanCourt, T. and Herbordt, M. (2009). Elements of high performance reconfigurable computing. In Zelkowitz, M., editor, *Advances in Computers*, volume v75, pages 113–157. Elsevier. doi: 10.1016/ S0065-2458(08)00802-4.

Wang, T., Geng, T., Jin, X., and Herbordt, M. (2019a). Accelerating AP3M-Based Computational Astrophysics Simulations with Reconfigurable Clusters. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 181–184. doi: 10.1109/ ASAP.2019.000-5.

Wang, T., Geng, T., Jin, X., and Herbordt, M. (2019b). FP-AMR: A Reconfigurable Fabric Framework for Block-Structured Adaptive Mesh Refinement Applications. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 245–253. doi: 10.1109/ FCCM.2019. 00040.

Wang, T., Geng, T., Li, A., Jin, X., and Herbordt, M. (2020a). FPDeep: Scalable Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters. *IEEE Transactions on Computers*, C-69(8):1143–1158. doi: 10.1109/TC.2020.3000118.

Wang, Z., Huang, H., Zhang, J., and Alonso, G. (2020b). Benchmarking High Bandwidth Memory on FPGAs. *arXiv preprint arXiv:2005.04324*.

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

Xilinx (2020a). UltraScale+ FPGA Product Tables and Product Selection Guide. https://www.xilinx.com/ support/documentation/ selection-guides/ ultrascale-plus-fpga-product-selection-guide.pdf.

Xilinx (2020b). Versal ACAP Premium Series Product Selection Guide. https:// www.xilinx.com/support/documentation/selection-guides/ versal-premium-psg.pdf.

Xiong, Q., Bangalore, P., Skjellum, A., and Herbordt, M. (2018a). MPI Derived Datatypes: Performance and Portability Issues. In *25th European MPI Users' Group Meeting*. doi: 10.1145/ 3236367.3236378.

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

Xiong, Q., Skjellum, A., and Herbordt, M. (2018b). Accelerating MPI Message Matching Through FPGA Offload. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 191–1914. doi: 10.1109/ FPL.2018.00039.

Xiong, Q., Yang, C., Haghi, P., Skjellum, A., and Herbordt, M. (2020). Accelerating MPI Collectives with FPGAs in the Network and Novel Communicator Support. In *IEEE Symposium on Field Programmable Custom Computing Machines*.

Xiong, Q., Yang, C., Patel, R., Geng, T., Skjellum, A., and Herbordt, M. (2019). GhostSZ: A Transparent SZ Lossy Compression Framework with FPGAs. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 258–266. doi: 10.1109/FCCM.2019.00042.

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

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

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

Young, C., Bank, J., Dror, R., Grossman, J., Salmon, J., and Shaw, D. (2009). A 32x32x32, spatially distributed 3D FFT in four microseconds on Anton. In *Proceedings of the*

Conference on High Performance Computing Networking, Storage and Analysis, pages 1–11. DOI: 10.1145/1654059.1654083.

CURRICULUM VITAE

[redacted]