BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

ACCELERATING MOLECULAR DYNAMICS SIMULATIONS WITH HIGH PERFORMANCE RECONFIGURABLE SYSTEMS

by

SHIHCHIN CHIU

M.S., University of Southern California, 2001

B.S., National Chung-Hsing University, 1997

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2011

Approved by

First Reader

Martin C. Herbordt, Ph.D.
Professor of Electrical and Computer Engineering

Second Reader

Allyn Hubbard, Ph.D.
Professor of Electrical and Computer Engineering
Professor of Biomedical Engineering

Third Reader

Roscoe Giles, Ph.D.
Professor of Electrical and Computer Engineering

Fourth Reader

Ayse Coskun, Ph.D.
Assistant Professor of Electrical and Computer Engineering

ACCELERATING MOLECULAR DYNAMICS SIMULATIONS

WITH HIGH PERFORMANCE RECONFIGURABLE SYSTEMS

(Order No.)

SHIHCHIN CHIU

Boston University, College of Engineering, 2011

Major Professor: Martin C. Herbordt, Ph.D.,

Professor of Electrical and Computer Engineering

ABSTRACT

Molecular Dynamics (MD) simulation plays an important role in understanding functionally important phenomena of biological processes. Many of such events, however, occur on time scales beyond the capabilities of modern computers. Hence, accelerating MD simulations becomes one of critical challenges in the current study of Computational Biology and Chemistry.

For the past several years, various implementations and efficient algorithms have been developed toward this goal. Of these, accelerating MD with Field Programmable Gate Arrays (FPGAs) has been shown to be a viable candidate for improved MD cost/performance. Given the intense competition from multi-core and GPUs, there is now a question whether MD on High Performance Reconfigurable Computing (HPRC) can be competitive. The goal of this research is to create an FPGA-based MD system to achieve substantial speedup over

production MD software without compromising simulation quality.

In one part of the study, we systematically explore and evaluate the design space of the force pipeline with respect to arithmetic algorithm, arithmetic mode, precision, equation complexity, and various other optimizations. We find that the FPGAs' Block RAM (BRAM) architecture makes them well suited to support unusually fine-grained intervals. This leads to a reduction in other logic and a proportional increase in performance.

In the other part, we present the first FPGA study of the filtering of particle pairs having nearly zero mutual force, a standard optimization in MD codes. There are several innovations, including a novel partitioning of the particle space, and new methods for filtering and mapping work onto the pipelines. As a consequence, highly efficient filtering can be implemented with only a small fraction of the FPGA's resources. Overall, we find that, for an Altera Stratix-III EP3ES260, six force pipelines running at nearly 200 MHz can fit on the FPGA. This results in a 26x per core speedup for the nonbonded short-range force. The resulting integrated system is likely to make FPGAs highly competitive for MD.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Abbreviations

ALM	.....	Adaptive Logic Modules
ALUT	.....	Adaptive Look Up Table
API	.....	Application Programming Interface
ASIC	…….	Application Specific Integrated Circuit
b	…….	Bit
B	…….	Byte
BRAM	…….	Block Random Access Memory
CLB	…….	Configurable Logic Block
CPU	…….	Central Processing Unit
COTS	…….	Commercial off-the-shelf
CUDA	…….	Compute Unified Device Architecture
DDR	…….	Double Data Rate
DMA	…….	Direct Memory Access
DRAM	…….	Dynamic Random Memory Access
DSP	…….	Digital Signal Processor
FFT	…….	Fast Fourier Transform
FIFO	…….	First In First Out
FLOPs	…….	Floating point operations per second
FP	…….	Floating Point
FPC	…….	Floating Point Compiler
FPGA	…….	Field Programmable Gate Array

Gb	…….	Gigabits, 2^{30} (10^9) bits
GB	…….	Gigabytes, 2^{30} (10^9) bytes
GFLOPs	…….	Giga FLOPS, 10^9 floating point operation per second
GHz	…….	Giga Hertz, 10^9 cycles per second
GPP	…….	General Purpose Processor
GPU	…….	Graphics Processing Unit
GROMACS	…….	Groningen Machine for Chemical Simulations, an open-source molecular modeling program
HDL	…….	High Description Language
HLL	…….	High Level Language
HPC	…….	High Performance Computing
HPRC	…….	High Performance Reconfigurable Computing
IP	…….	Intellectual Property
Kb	…….	Kilobits, 2^{10} (10^3) bits
KB	…….	Kilobytes, 2^{10} (10^3) bytes
LE	…….	Logic Element
LJ	…….	Lennard-Jones
LSB	…….	Least Significant Bit
LUT	…….	Look Up Table
Mb	…….	Megabits, 2^{20} (10^6) bits
MB	…….	Megabytes, 2^{20} (10^6) bytes
MD	…….	Molecular Dynamics

MHz	…….	Mega Hertz, 10^6 cycles per second
MPP	…….	Massively Parallel Processor
MSB	…….	Most Significant Bit
MW	…….	Mega Watt
PAL	…….	Programmable Array Logic
NAMD	…….	NAnoscale Molecular Dynamics
PAR	…….	Place And Route
PC	…….	Personal Computer
PCI	…….	Peripheral Component Interconnect
PCIe	…….	Peripheral Component Interconnect Express
PE	…….	Processing Element
PME	…….	Particle Mesh Ewald
RAM	…….	Random Access Memory
RTL	…….	Register Transfer Level
SDRAM	…….	Synchronous Dynamic Random Access Memory
SIMD	…….	Single Instruction Multiple Data
SODIMM	…….	Small Outline Dual Inline Memory Module
SRAM	…….	Static Random Access Memory
TB	…….	Terabytes, 2^{40} (10^{12}) bytes
TDP	…….	Thermal Design Power
VHDL	…….	VHSIC Hardware Description Language
VLIW	…….	Very Long Instruction Word

Chapter 1 Introduction

1.1 The Problem

Driven by the rapid increase of computer power over the last 60 years, advances in computational methods have made a dramatic impact in traditional sciences. Computer simulations are now regularly used to verify theoretical assumptions, as well as to predict experimental outcomes and hence bridge the gap between theory and experiment. Simulations act as a complement to laboratory experiments, enabling us to learn something new, to understand something that cannot be found out in other ways, and to discover something difficult to observe in the laboratory [2].

Molecular Dynamics (MD) simulation is central to Computational Biology and Chemistry and commonly used in the study of biomolecules, proteins, and generally in material modeling. MD helps us understand the properties of a molecular systems in terms of the structure and microscopic molecular interactions [2, 67]. We recall a famous sentiment [28]:

Certainly no subject or field is making more progress on so many fronts at the present moment than biology, and if we were to name the most powerful assumption of all, which leads one on and on in an attempt to understand life, it is that all things are made of atoms, and that everything that living things do can be understood in terms of the jiggling and wiggling of atoms.

Richard Feynman, Lecture on Physics, vol. 1, p. 3-6 (1963)

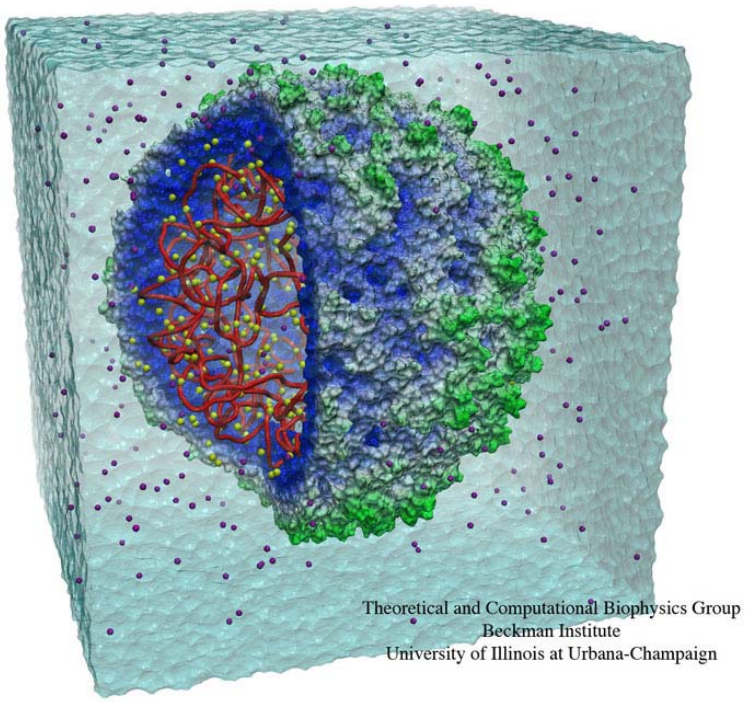

Figure 1-1: Satellite tobacco mosaic virus[i] [116]

Many important biochemical phenomena, however, usually occur on time scales that are beyond the reach of current technology [109]. Figure 1-1 illustrates a million-atom MD system consisting of a small, icosahedra plant virus, standing for satellite tobacco mosaic virus (STMV) [33, 116]. It would take a single 2006-era desktop computer over 35 years to perform 50ns simulation using NAMD, highly optimized MD package [30, 33].

Moreover, the dramatic increases of application demands have greatly outpaced the technology curve as postulated by Moore's law, which the growth of

[i] This figure was made with VMD and is owned by the Theoretical and Computational Biophysics Group, an NIH Resource for Macromolecular Modeling and Bioinformatics, at the Beckman Institute, University of Illinois at Urbana-Champaign.

new technologies still struggle to follow. Accelerating MD simulation has become a critical challenge for scientific research. As such, it has obtained much attention from various perspectives such as supercomputers, multi-cores, ASICs, and coprocessing technology using GPUs, Cell, and FPGAs. The last of these is what we focus in this work.

The problem that this dissertation explores is how to improve the computational performance and efficiency of molecular modeling applications. In particular, we address this problem by accelerating molecular dynamics simulation with the use of Field Programmable Gate Arrays or FPGAs.

1.2 Molecular Dynamics

Molecular dynamics simulation is a technique that models motions of molecular particles by applying known classical mechanics. It is an iterative process comprised of two phases: force calculation and motion update. Forces applied on each particle are computed using classical equations of motion and then the state of each particle is updated accordingly. MD simulation can serve as a computational “microscope” to observe functionally important biological processes such as folding of proteins and various types of interactions between proteins that are difficult to be observed in experiments [14]. Several open questions in the areas of biology and chemistry could be answered with aid of such simulations [109]. Many of such important events, however, usually occur on long time scales beyond the reach of modern computer capabilities.

Therefore, how to perform effective acceleration of MD simulations becomes a crucial subject in the current research of computational biology and chemistry.

Among all computations in MD, force evaluation consumes the majority of computational powers. In general, the forces depend on the physical system being simulated and may include van der Waals (approximated with the Lennard-Jones or LJ potential), electrostatic, hydrogen bond, and various covalent bond terms:

$$F_{total} = F_{bond} + F_{angle} + F_{torsion} + F_{hydrogen} + F_{non-bonded} \quad (1\text{-}1)$$

Because the hydrogen bond and covalent terms (bond, angle, and torsion) affect only neighboring atoms, computing their effect is *O*(*N*) in the number of particles *N* being simulated. The motion integration computation is also *O*(*N*). The complexity of non-bonded force evaluation is $O(N^2)$ initially and comprises the bulk of computations [46]. Several algorithms and techniques have been developed (over multiple decades) to limit the computational cost of the nonbonded forces. Each of those has its specialty and advantage for different computational platforms.

1.3 High Performance Computing with Accelerators

Since the 1970s, microprocessors have formed the backbone of all computing, and dominating high performance computing (HPC) since the early 1990s. During that time, performance of microprocessors has improved steadily and

exponentially. There have been three primary axes of improvement: increased efficiency through architectural advances, increased operating frequency, and increased number of devices per chip. The first of these appears to have topped out around 2000 when increases in Instruction Level Parallelism (ILP) began to stagnate. The second peaked in around 2004 at 4GHz and has remained steady or diminished slightly. This is due to the well-known power wall [13] that, among other problems, makes it challenging to cool devices at higher frequencies. The third component, however, continues to increase and projects to continue to do so for at least 5 more years. The focus of HPC, therefore has shifted from improving single-thread performance to increasing the number of cores for overall performance enhancement.

In November 2010, the TOP500 project confirmed that the top spot indicative of the world's fastest supercomputer had been taken by a Chinese supercomputer, the Tianhe-1A at the National Supercomputer Center in Tianjin. It demonstrated performance of 2.57 petaflop/s [124]. It consists of 14,336 multi-core CPUs and 7,168 GPUs and illustrates a good example of accelerated-based computing. The Tianhe-1A outpaces the former number one (and now number two), the Cray XT5 "Jaguar" system at the Oak Ridge National Laboratory. Jaguar contains 224,256 CPU cores and can achieve 1.75 petaflop/s [65, 124].

Besides performance, another critical issue is the energy efficiency. Although microprocessor-based clusters provide massive computational, they usually dissipate from hundreds of kilowatts to a few megawatts of power and require

cooling infrastructure. In contrast, a high-ended FPGA typically consumes just 20-30 watts. Although GPUs have high power consumption (100 - 300 watts), GPU-accelerated systems often have better performance-per-watt ratios than those with only CPUs. According to the recent release of the Green500 list on November 2010, eight of top ten slots were occupied by accelerated-based supercomputers [43]. The metric used is million floating point operations per second per watt. The potential for energy efficiency should spur the continued growth and popularity of accelerator-based computing.

As a result, high performance accelerated-computing is getting much attention from the HPC community. Some examples of accelerators are:

- Graphics Processing Units (GPUs)

  GPUs are commodity hardware that were initially designed to perform fast graphics rendering. Since CUDA was introduced to provide developers access to immense parallel computing elements in GPUs, GPUs have become a popular computing alternative for scientific applications, especially for those requiring massive numbers of floating point computations.

- Field Programmable Gate Arrays (FPGAs)

  Because of its flexible programmability and application specific characteristic, FPGAs have been used widely in design prototyping, digital signal processing, and network switches. Recent advances in FPGA

architecture such as incorporating hard multipliers and embedded processors have attracted attention and interest from other areas such as scientific computing and bioinformatics. HPC with FPGAs is often referred to as High Performance Reconfigurable Computing (HPRC).

- IBM Cell Processor

  Cell was originally designed for gaming, but its effective multi-core configuration, together with internal high speed interconnects, makes it popular and powerful for various applications and computations, including 3D FFT, video processing and cluster computing.

- Application Specific Integrated Circuits (ASICs)

  An ASIC is an integrated circuit that is customized to tackle a specific application and can be optimized to provide the best performance in terms of speed, chip density, and power. ASICs, however, often require high development costs, long time-to-market, and lack of flexibility.

Although each of these differs from the others in term of its own features and limitations, many of them share all or at least several of the following common characteristics [13]:

- Communication between the host and accelerator becomes a performance bottleneck
- High performance comes from high parallelism and utilization

- High performance requires much data reuse
- Meeting peak performance is hard
- Integer or single precision arithmetic is more favorable for achieving high performance
- Lack of software tool support
- Restructuring of algorithms and data structures is the key to success

Finding what applications are most cost-effective on which architecture is a critical problem and the answers vary with applications and problems.

1.4 High Performance Reconfigurable Computing with MD

Since firstly introduced in 1960 [26], field program gate arrays (FPGAs) have been popularly adopted for exploring new algorithms. Their tremendous flexibility and power efficiency have made them a great success in digital signal processing (DSP) where multiple small kernels are executed in parallel. Recent advances in semiconductor technology and enhanced features such as hard multipliers and individually accessible BRAMs have made FPGAs an attractive candidate for high performance computing [41]. For example, HPC using FPGAs has been extended the reach of bioinformatics and computational chemistry in such areas as biological sequence alignment analysis [50, 57, 64], molecular docking [118], and molecular dynamics simulation [11, 16, 46, 69, 105].

MD simulation is a central method in high performance computing (HPC) with applications throughout engineering and natural science. 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. As such it has received attention as a target for supercomputers [29], clusters [14], and dedicated hardware [71, 109, 123], as well as coprocessing using GPUs [96, 101], Cell [111], and FPGAs [1, 11, 46, 52, 69, 105, 128]. The last of these, MD with HPRC, is our focus here. In particular, we demonstrate that MD with HPRC is not only cost-effective, but in fact, an excellent fit. This result is surprising given the FPGA's reputation for having difficulty with floating point intensive computations.

In this research, we re-examine the short-range force computation that dominates MD. Although this problem has been addressed by many groups in the last few years, much of the design space has remained unexplored. In addition, recent advances in FPGA hardware and in compiler technology appear to have shifted some basic trade-offs.

Our study has three parts. The first part considers the force pipeline. Our goal here is to maximize throughput—operating frequency and the number of pipelines that fit on the FPGA—while maintaining simulation quality. To do this, we explore various ways to perform the arithmetic, the modes in which to execute the operations, the levels of precision, and other optimizations. Some of the choices are as follows.

- Direct computation (Direct) versus table lookup with interpolation (LUT)
- Interpolation order and the interval resolution (for LUT)
- Precision: single, double, custom
- Mode: floating point, hybrid fixed/floating point, custom
- Implementation: synthesized components, vendor cores, vendor compiler (Alter floating point datapath compiler)
- Various simulation configurations and complexity of target functions

We find that the LUT method is now preferred, and that single-precision floating point combined with higher precision fixed point leads to both excellent performance and high-quality simulations.

The second part considers filtering particle pairs. This issue emerges from the geometric mismatch between two shapes: (i) the cubes (or other polyhedrons) into which it is convenient to partition the simulation space and (ii) the spheres around each particle in which the short-range force is non-zero. If this mismatch is not addressed (e.g., only the standard cell-list method is used), then 85.5% of the particle pairs that are run through the force pipelines will be superfluous. While filtering is a critical issue, we believe that the only previously published results related to hardware implementations are from D.E. Shaw; these are with respect to their Anton processor [109]. Here, we find filtering implementation on FPGAs to provide a rich design space. Its primary components are as follows.

- Filter algorithm and precision
- Method of partitioning the cell neighborhood to balance load with respect to the Newton's 3rd law optimization
- Method of mapping particle pairs to filter pipelines
- Queuing and routing between filters and force pipelines

We present new algorithms or methods for filtering, load balancing, and mapping, and find that nearly perfect filtering can be achieved with only a fraction of the FPGA's logic.

The third part considers the integration of the new features specified by the other two parts. The particle mapping to the filter pipelines leads to changes in how cell lists are swapped on/off chip. In addition, the filter pipelines generate neighbor lists that must be fed into the force pipelines. And having multiple force pipelines (6 or more) requires accumulation of forces on the other end. We find solutions to all of these issues that have simple control, match FPGA resources, and add only little overhead.

Our basic result is that for the Stratix-III EP3SE260, and for the best (as yet un-optimized) designs, 6 force pipelines running at nearly 200 MHz can fit on the FPGA. Moreover, the force pipelines can be run at high efficiency with 90% of cycles providing payload. As a result, the short-range force for the standard 92K ApoA1 NAMD benchmark can be computed in less than 70 ms, or about a factor

of 26 faster than its per-core execution time that makes our MD design competitive and attractive.

1.5 Summary of Contributions

The contributions of this work can be classified into four main categories: (i) acceleration of MD simulation, (ii) demonstration of FPGA viability for MD (iii) the algorithms and scheme developed for MD, and (iii) the extension of the proposed algorithms and methods to other computations and applications.

The first is straightforward and immediate. Our FPGA-based accelerator shows that significant speedup can be obtained over the production software implementation while still maintaining acceptable simulation quality.

The second ensues from the outcome of the first. Given the intense competition from multi-core and GPUs, there has been a question whether MD on HPRC can be competitive. We illustrate FPGA-based computing is still a viable and highly competitive technology with our results and experiences. It, indeed, requires application-specific algorithms and schemes to map the problem well on the underlying hardware and demands careful hardware-aware implementation. We now describe those algorithms and methods in details as follows.

1.5.1 Acceleration of Molecular Dynamics Simulation

Many critically important molecular processes occur on a millisecond time scale that is beyond the reach of MD simulations with current technology capabilities

[109]. The main goal of this research is to shorten this gap by accelerating MD simulation and hence advance the progress in science and engineering. We have designed an MD accelerator that enables a significant increase in computational power and efficiency without compromising simulation accuracy. Our MD accelerator can compute the nonbonded short-range force for the ApoA1 benchmark in less than 70 ms. This represents a 26-fold per core speedup for the computational kernel. Several features of our MD accelerator include:

- **Exploration of various design implementations:** We substantially expand the exploration of the MD force-pipeline design space with respect to arithmetic algorithm, arithmetic mode, precision, and various optimizations with the goal of finding the performance limits under current technology and methods. We find that FPGAs' BRAM architecture makes them well suited to support unusually fine-grained intervals. This leads to a reduction in other logic and a proportional increase in performance.
- **Throughput enhancement of non-bonded force pipelines:** We present the first study of particle-particle filtering on FPGAs and with it a number of innovations. We find that high quality filtering can be achieved with only a small amount of logic. We present a geometric filtering scheme that is preferable for some FPGA implementations. And finally, the particle-mapping variation for mapping particle pairs to filter pipelines also appears to be new. We also describe methods for sizing components and for integrating sections of the overall processing pipeline.

- **Novel partitioning scheme**: We present a new domain partitioning method for optimizing with respect to Newton's 3rd law. It helps minimize the impact of load imbalance among filters and improves filtering efficiency. This is essential for the design presented here, but could also find application in other hardware implementations.
- **Hybrid numerical precision manipulation**: Floating-point computation was long beyond the reach of FPGAs. Recent advancements of FPGAs' architecture and processing technology now make FPGAs competitive compared to standard microprocessors [118]. The computational cost of floating point, however, is still high and the large dynamic numerical range supported by floating point arithmetic may not be necessary for all type of computations. By carefully analyzing numerical operation and accuracy and evaluating resource costs, we present a novel force pipeline that uses mixed-precision (called hybrid) arithmetic to evaluate the nonbonded short-range force. This results in performance improvement while still maintaining acceptable simulation quality.

1.5.2 General Computational Model and Algorithms

Although the algorithms and techniques developed in this work are with respect to MD simulations, they can be generalized and extended to other HPC applications. MD shares many features with those in N-body simulation and other molecular modeling applications. The data restructure and cell-list scheme

presented in this work are generic and can be applied to other parallel architecture and hardware platforms. The novel partitioning scheme that removes load imbalance from the critical path could find useful in other domain or spatially partitioned applications. Moreover, the system architecture that supports large simulations, with the management of off-chip data access, provides a generic framework for those that require off-chip memory access.

Other generic techniques include the findings of the design space exploration and the scheme of hybrid precision arithmetic. These are generic and can be extended to other computations or applications.

1.6 Organization of the Rest of this Thesis

The rest of this thesis is organized as follows. Chapter 2 describes the potential of high performance computing with FPGAs and provides a brief introduction to the target FPGA-based platforms.

Chapter 3 presents an overview of MD, including various techniques that help reduce MD computational complexity, and of several popular MD software packages and well-known hardware accelerators. A review of previous FPGA-MD work done in the CAAD Lab of Boston University is also presented.

In chapter 4, we present the work of design space exploration, specifically of examining various short-range force pipelines and determining their performance bounds and design limitations. Several design considerations and challenges are also addressed.

Chapter 5 describes the first FPGA study on the filtering of particle pairs with nearly zero mutual force, a standard optimization in MD codes. There are several innovations, including a novel partitioning of the particle space, and new methods for filtering and mapping work onto the pipelines.

In chapter 6, we describe the details of integrating our MD design into FPGA accelerated board and MD software codes. We present an overview of system architecture and flow of control. The techniques of making efficient data transfer among various components (FPGA, on-board memory, and host) are also discussed.

Chapter 7 describes the results of our FPGA accelerating system from two perspectives, performance improvement and simulation quality. We then present a preliminary study of our design on multiple FPGA scalability to demonstrate the performance potential. We also describe status of the overall implementation and implications for MD on HPRC.

Chapter 8 summarizes this thesis and provides some future directions to further extend and improve our design.

Chapter 2 High Performance Reconfiguration Computing

A common configuration of high performance computing (HPC) systems consists of a large array of conventional microprocessors that are interconnected together in a symmetrical way via high bandwidth communication channels [83]. This approach can provide good scalability that is relatively easy to manage. This generic configuration, however, may not be the best solution for various applications.

One alternative, high performance reconfigurable computing (HPRC) systems, consisting of microprocessors and field-programmable gate arrays (FPGAs), have been shown to deliver significant performance gains in terms of speed, power, and cost efficiency for different algorithmic applications [11, 46, 55, 118]. With the FPGAs' flexible programmability, an HPRC system can be tailored to fit many specific algorithms and applications. Accelerating HPC applications with FPGAs, however, still faces several serious shortcomings and challenges, such as low operating frequency and bounded hardware resources.

In this chapter, an overview of FPGA features and HPRC design considerations is presented, including recent trends in FPGA architecture and tool support. Computing models suitable for FPGA-based acceleration are suggested. In the end, several FPGA-based HPC systems are introduced briefly to illustrate various configurations of FPGA-based implementations.

2.1 Field Programmable Gate Arrays

2.1.1 Overview

A Field Programmable Gate Array (FPGA) is a semiconductor device that consists of an array of configurable logic blocks (CLBs) and interconnects. Unlike ASICs where the functionality remains unchanged during the entire lifetime, FPGAs can be programmed to perform various dedicated tasks after manufacturing [5]. Since its first introduction in 1985 [135], the FPGAs' flexible programmability together with other features, e.g. low power consumption and highly cost-effective computing, makes them attractive for many applications such as prototyping CPU designs, networking and communications, digital signal processing, embedded systems, and, more recently, in bio-informatics [11, 46, 118].

Traditionally FPGAs are best suitable for applications that contains intense integer arithmetic or streaming-type computations. FPGAs continue to evolve with the advance of semiconductor processing technology: recent FPGAs often contain built-in hard Intellectual property (IP) cores, including thousands of independent addressable memory blocks and arithmetic circuits. These enable FPGAs to meet the diverse needs of algorithms and applications while still maintaining low power and cost [5]. It is worth to note that recent FPGAs have demonstrated outstanding performance in floating point arithmetic [58, 126], which make FPGAs highly competitive for HPC applications.

Nowadays, FPGAs have been integrated to offer systems-on-a-chip (SOC) solutions. In 2010 Intel Developer Forum (IDF), Intel introduced their first configurable Atom-based processor, Stellarton, which features an Intel Atom E600 processor (formerly codenamed “Tunnel Creek”) paired with an Altera FPGA in a single multi-chip package [63, 113]. The FPGA contains more than 60,000 logic elements and can support six high-speed transceivers running up to 3.125Gbits/s. It aims to provide developers more options in their products with greater differentiation and competition, shorter time-to-market and lower cost of design prototyping.

2.1.2 FPGA Architecture

A typical FPGA architecture consists of a two-dimensional array of programmable logic blocks embedded in a network of configurable interconnects and interface I/O ports as shown in Figure 2-1 [22, 99, 103]. The logic blocks can be programmed to implement various combinational and sequential functions and are connected together via configurable interconnects. Most of the logic blocks also contain adders and flip-flops to support sequential implementations.

The basic element in and FPGA is variously called the Logic Element or LE, Adaptive Logic Module or ALM, or Slice. In general it contains 4- to 6-input Look-Up Tables (LUTs), adders, and registers [6, 136]. A LUT is commonly implemented with 2^k:1 multiplexers and 2^k configurable memory cells, where k is the number of inputs to the lookup table [103]. All possible logic values of K-input

variables are programmed and stored in memory cells and the output is selected via multiplexers to provide dedicated functionality.

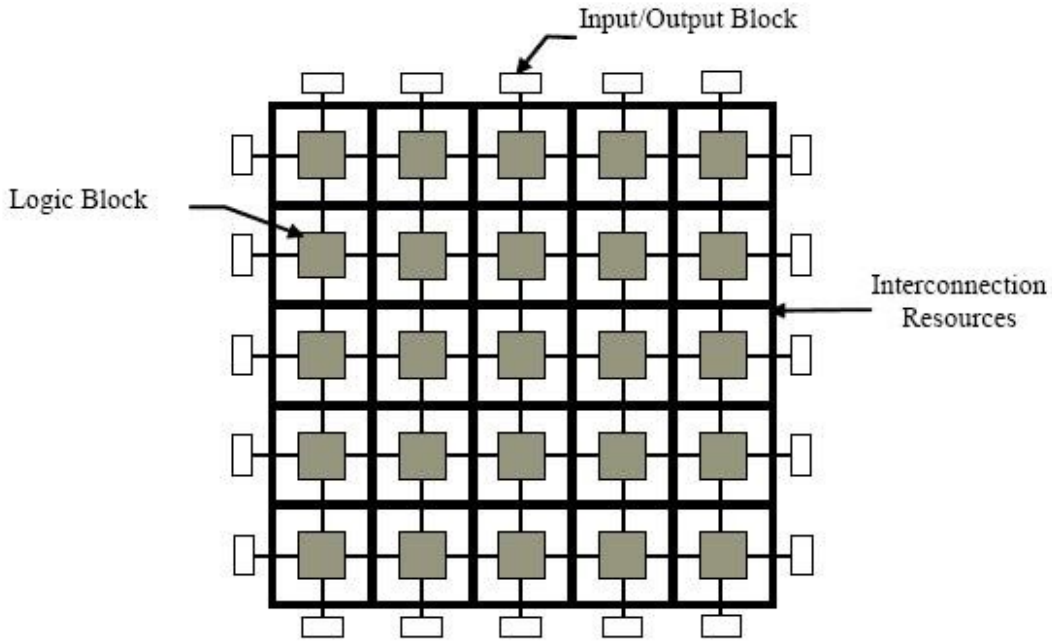

Figure 2-1: A typical FPGA structure [103]

As an example, see the block diagram of the Altera ALM shown in Figure 2-2 [6]. An ALM consists of combinational logic, adders, multiplexers, and registers. The combinational logic has eight inputs and includes a LUT that can be divided into two adaptive LUTs (ALUTs). It can support various configurations, e.g., 2 independent 4-input function, or a 5-input and a 3-input function with independent inputs, or an arbitrary six-input function [6]. Other components, i.e.

adders and registers, increase arithmetic capability and provide an option to latch LUT outputs for the support of sequential implementations and higher frequency.

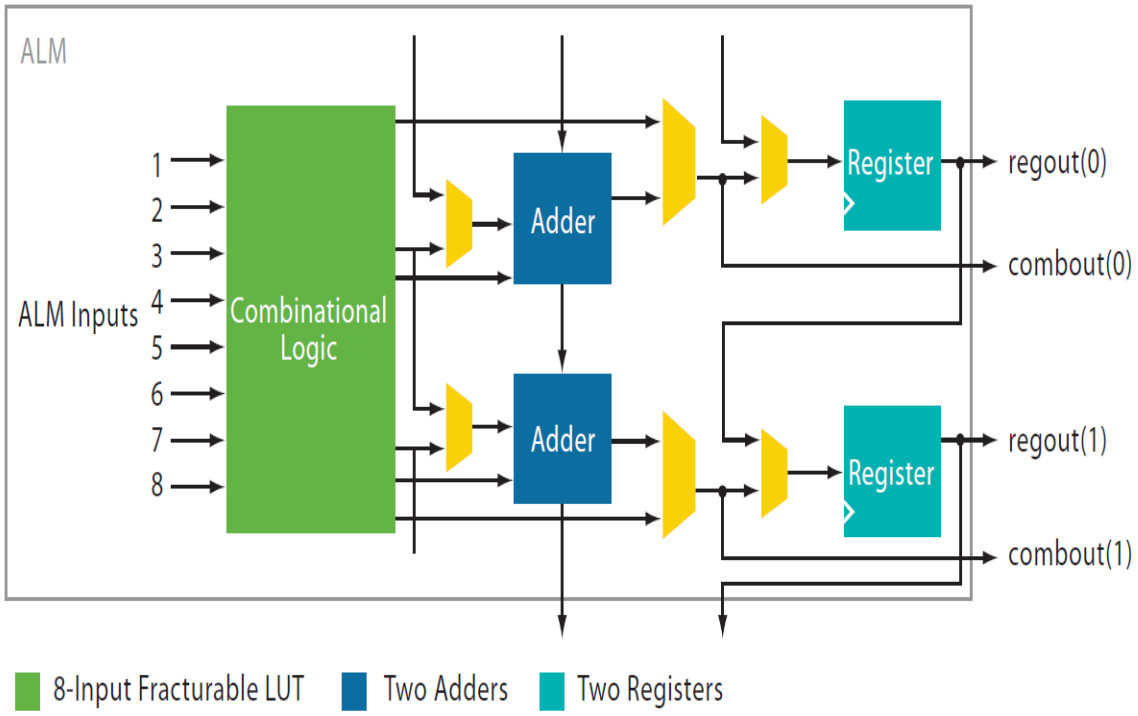

Figure 2-2: Altera Adaptive Logic Module (ALM) block diagram [6]

As described earlier, FPGAs have been successfully used in many application domains. Their success can be attributed to the following reasons [56, 118]:

- **High Parallelism**

  Millions of equivalent logic elements and hundreds of hardware components (memory blocks and arithmetic units) can be used to explore different levels of parallelism, including deep pipelining and design replication.

- **Configurable Memory Interface**

  In contrast with microprocessors where the memory interface is fixed, configurable memory blocks can be tailored to meet application-specific needs. In addition, thousands of independent addressable BRAMs enable FPGA to offer parallel single-cycle data access [118].

- **Efficient Computing**

  Since most of control logic has been embedded in designs, little or no overhead is required and high throughput can be achieved.

The great flexibility of FPGAs, however, comes at a price: diminished effective chip area and operating frequency [22]. In order to be configurable, compared to conventional ASICs with the same processing technology, the FPGA fabric is less dense and a huge amount of its resources must be preserved for routing channels. This also results in operating frequencies that are often 5 or 6 times slower than conventional ASICs.

2.1.3 Support for FPGA-based Design

Hardware Description Languages (HDLs), such as VHDL and Verilog, have been widely used in the traditional FPGA/ASIC design flow. They provide precise control over hardware implementations and synthesis results. Users can define high-level algorithms and perform low-level optimizations (gate or switch) via the same language. Productivity has been a concern, however, especially for meeting current rapid time-to-market period since highly optimized designs can

take several hours or days to synthesize. In addition, it is difficult to validate with the original software model directly, and writing testbenches to ensure the full correctness of results is very timing-consuming [61].

In the past decade, various supports from FPGA and third party tool vendors have enhanced design quality as well as development efficiency. This includes highly optimized Intellectual Property (IP) blocks and high-level programming language tools and suites, which we now briefly describe.

- **Intellectual Property (IP) Core**

  To simplify and shorten FPGA design process, highly optimized IP cores are often offered in FPGA design suites, e.g., highly efficient floating point cores such as FFT and inverse square root. Users only have to connect each component together rather than implement and optimize their own cells. This helps reduce development cycles and improve performance; at the same time, however, it reduces the design portability since those IP cores are specialized for certain FPGA architecture and devices.

- **Software Tool Support**

  In the past years, tool vendors have focused on improving design efficiency, including both core implementation and system integration. A variety of C-to-gates or C-to-FPGA compilers have been developed to convert high-level C-like software codes to gate-level HDL implementations. This enables embedded designers and software

programmers to create HPRC applications from system-level perspective without knowing the details of hardware knowledge.

Some examples are Impulse-C from Impulse Accelerated Technologies [62], MitrionC from Mitrionics [81], AutoPilot FPGA from AutoESL Design Technologies [10], and Altera Floating Point Compiler [4]. In our experience, although high-level compilers help shorten design cycles, the performance of auto-generated design is often inadequate, especially for control logic and fine-grained implementations.

In our MD work, we used the Altera Floating Point Compiler (FPC) to implement the nonbonded short-range force pipeline. We have found that this tool is effective for resource reduction and rapid design. The FPC is now described.

- Altera Floating Point Compiler

  The FPC takes C-language description of a function and translates it to a parallel datapath implementation [4, 73]. It analyzes functional expressions, determines inputs and outputs, creates a dataflow graph of the internal operations, and maps a design to FPGAs. The efficiency of the FPC gains comes from two main principles employed: a change of number representation and formats and a reduction of redundant normalizations across a group of operators [73].

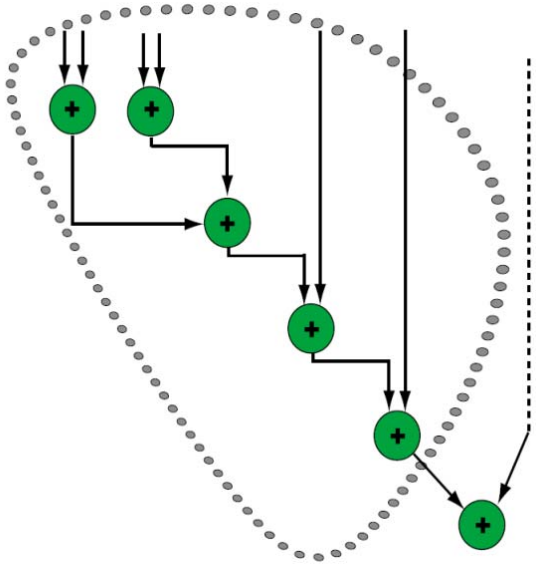

Figure 2-3: An example of local cluster scheme

An example of the local cluster scheme is shown in Figure 2-3 [4]. The FPC groups identical operations into a local cluster. Within a cluster, all intermediate results are kept unnormalized, if possible. The FPC employs several schemes to perform normalization, depending on the types of operations. In most cases, normalization is only required on the way out of a local cluster or datapath. If floating point operations can be clustered efficiently, a significant reduction of logic resources can be obtained.

As described above, although the inputs and outputs of FPC follow the IEEE 754 floating point format, the internal format of FPC are not compliant. Integer formats are used in some steps internally to improve efficiency. Together with the property of non-associativity of floating point

arithmetic, it often results in differences between FPC and serial C codes. This difference could be compounded for applications that involve iterative and high precision computations [19, 118].

2.2 HPC with FPGAs

FPGA-based systems have demonstrated great successes in digital signal processing (DSP) where multiple small kernels are executed in parallel. Recently, FPGAs have been used to accelerate biological and medical applications such as biological sequence alignment analysis [50, 57, 64], molecular dynamics simulations [11, 46, 69, 105, 106], protein docking [118, 119, 120], and positron emission tomography (PET) scanning [55].

A great example illustrating the promise of HPRC systems is the Novo-G supercomputer at the University of Florida. The Novo-G system consists of 24 computing nodes, each housing two Gidel quad-FPGA boards. Nodes communicate with one another via Gigabit Ethernet and non-blocking fabric of 20 Gb/s InfiniBnad [90]. In other words, the system contains in all 48 Gidel PROCStar III boards and 192 Altera Stratix III FPGAs. 4.25 GBs of dedicated memory is attached to each FPGA for a total of nearly 1TB. Perhaps most remarkably, it consumes just 8,000 watts, compared with conventional microprocessor-based clusters that can dissipate megawatts, e.g., Jaguar (6.95 MW) and Roadrunner (2.35 MW) [124]. In addition, Novo-G has successfully demonstrated significant performance on various applications, especially for bioinformatics research [90].

Some of the driving forces that make FPGAs plausible for HPC applications can be summarized as follows.

- **Advancing Technology**

  As with modern microprocessors, FPGAs still ride the technology curve as stated by Moore's law [46]. There are also continuing architectural advances. Current generation high-end FPGAs contain not only millions of configurable gate-equivalent logic elements but also hundreds of optimized ASIC components such as individual block RAMs (BRAMs), hard multipliers, and embedded microprocessors. Also, recent logic designs map more favorably to floating point applications.

- **Power Consumption**

  Due to its inherent low operating frequency, high-end FPGAs usually consume at most 20-30 watt; modern microprocessors consume 100-200 watts [118].

- **Flexibility**

  One of most attractive FPGA features is its reconfigurability that enables users to explore the design space to find optimal solutions for a specific target problem. FPGAs can be reprogrammed quickly, while it can take several weeks (at least) to make the same changes in an ASIC.

- **Time to Market**

  Compared with ASIC design, which usually takes several months or even more than a year, FPGAs offer much shorter design cycles. Users can implement a prototype design and verify ideas in hardware without going through the long and costly fabrication process. Incremental changes can be made and iterated on an FPGA within hours rather than weeks.

Although significant advances in FPGA chip development offer tremendous computing power, achieving a performance gain is not straightforward. It is governed by several critical factors such as:

- **Resource Limitations**

  Although modern FPGAs provide plenty of gate-equivalent logic elements and a large number of hard multipliers and BRAMs, resources are still limited. This bounds the number of processing units that can fit on a chip and thus the resulting throughput. How to efficiently utilize available resources often involves reconstruction of underlying algorithms, partition of tasks, and numerical methods.

- **Amdahl's Law**

  Efforts must be paid to accelerate the kernels that dominate the entire computation in order to improve the overall speedup. But even so, latency of serial parts, such as interaction with the host, must be minimized and hidden altogether if possible.

- **Design Expertise**

  Not all of tasks are viable candidates for FPGA acceleration. Thus, identifying algorithms that can exploit fine-grained parallelism and codes that contain intensive calculations such as excessive inner looping becomes a critical job. In order to efficiently map applications to FPGAs, certain levels of knowledge of algorithms and design expertise are often extremely helpful.

- **Programming Language**

  A common way to implement designs on FPGAs is to use a hardware-description-language (HDL) such as Verilog and VHDL. This often requires users to have understanding of logic designs and hardware behaviors. Recently, several software packages (e.g., Impulse-C and AutoPilot FPGA) and higher-level programming languages such as System-C and System Verilog have been developed to lower this barrier. This ease-of-use, however, sometimes compromises the resulting performance.

- **Arithmetic Mode**

  Numerical precision and computational mode are often related to the quality of implementations and achievable throughput. Most HPC codes are implemented with floating point arithmetic, usually double precision. The cost of performing floating point computations with FPGAs, however,

is relatively expensive compared to fixed point operations. Though vendors have made floating point computation more straightforward and cost-efficient by providing floating point IP cores and compiler tools, careful analyses of required precision and arithmetic mode are critical factors that have a direct impact on the resulting performance and quality.

- **Hardware/Software integration**

  FPGAs usually serve as coprocessors to accelerate compute intensive tasks and so cooperate with the programs executing on the host. The FPGA coprocessor also interacts with peripherals on board such as high-bandwidth interconnects and device memory. Efficiently bridging those components is another issue that designers must consider.

2.3 FPGA Computing Model

Although FPGAs have demonstrated enormous potential performance in many areas, getting tremendous speedup, however, is challenging. In particular, not all applications (or algorithms) are suitable. Therefore, identifying the appropriate computing model that can be well mapped to FPGAs becomes essential in HPRC design. In our previous work, several computing models were proposed [56, 57]:

- **Streaming**

  As its name implies, “streaming” is to pass data through arithmetic units. Streaming computing fits well for FPGAs because of their natural

properties such as multiple parallel streams, high I/O bandwidth, and flexibility of connecting different streams. Applications of streaming computing include signal and image processing, systolic array algorithms, sequence alignment, and protein docking [77, 118].

- **Associative Computing**

  This model is basic to computing with massively parallel SIMD arrays and can be characterized by the following properties: (1) broadcast, (2) parallel tag checking, (3) collective response, and (4) reduction of responses. The performance gains come from the support of hardware broadcast and reduction and fast single-cycle data access.

- **Functional Parallelism**

  Functions that take a long time in software but relatively few hardware resources would be the best candidate to be off-loaded to FPGAs, e.g., a high-quality frequently used random number generator. It only takes little chip area on an FPGA and can be fully pipelined so that the latency can be hidden.

2.4 HPRC FPGA Systems

Various Commercial Off-The-Shelf (COTS) FPGA systems have been offered by many vendors. Although each has its own goals and design considerations, it has to work with other peripherals and components such as memory modules and communication interfaces. Many of them have been used to deliver significant performance improvement in applications such as Molecular Dynamics simulations [16, 46, 69, 106], Protein Docking [118], BLAST, medical image processing and financial applications. A brief introduction of selected FPGA platforms is given in this section to illustrate different common configurations of FPGA-based coprocessor implementations.

2.4.1 Annapolis Micro Systems

The first example is a PCI-based system, the WILDSTAR-II PRO board from Annapolis Micro Systems, Inc [9]. This was used to implement our prior FPGA/MD work done by Dr. Gu [46]. Its schematic block diagram is shown in

Figure 2-4 [132]. The system consists of two Xilinx Virtex-II-Pro FPGAs, either the XC2VP70 or the XC2VP1000. Each FPGA is connected to up to 48 MB off-chip SRAM and a 128 MB DRAM. The FPGAs communicate with a host via a PCI bus interface. Two FPGA chips are connected to each other by differential and Rocket I/O pairs.

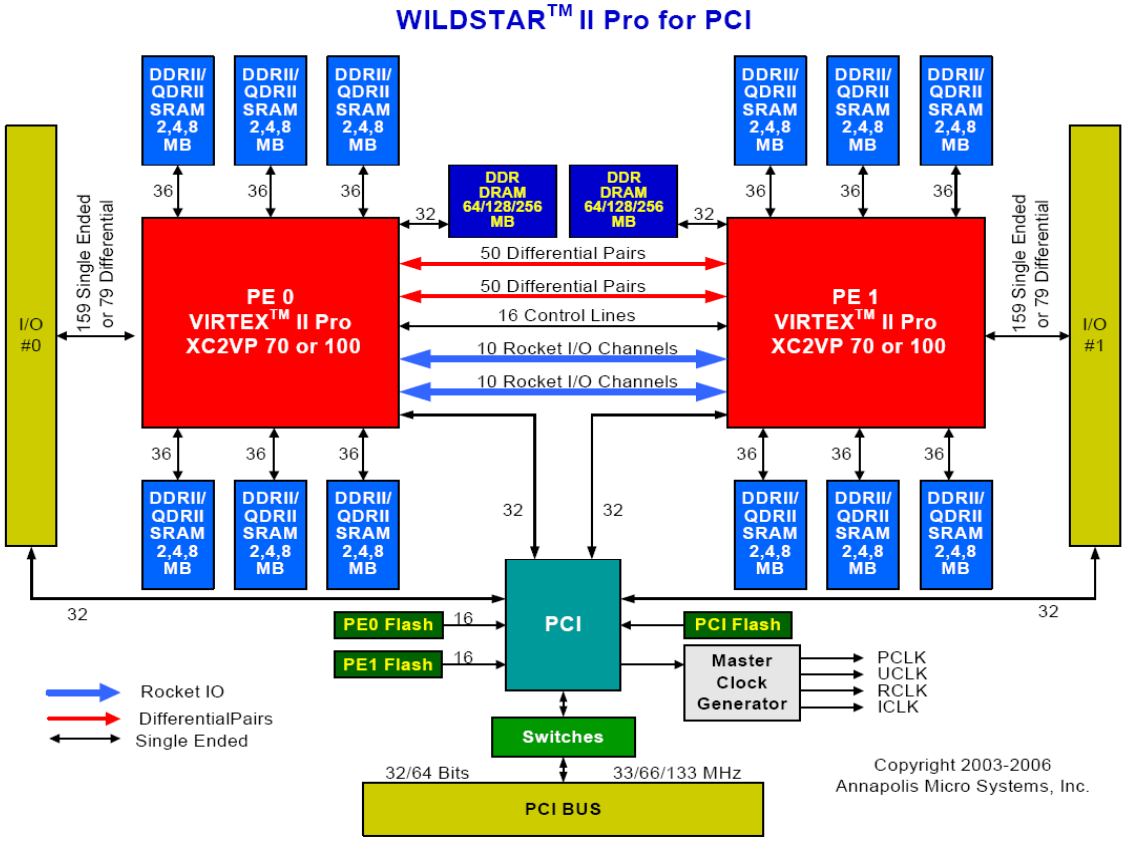

Figure 2-4: Wild-Star II Pro board block diagram [132]

2.4.2 Gidel PROCStar III Board

Our MD acceleration solution has been successfully implemented and is currently running on one FPGA of a Gidel PROCStar III board [40], a single node of Novo-G. PROCStarIII is also a PCI based system with 8-lane PCI Express (PCIe x 8) host interface and its block diagram is shown in Figure 2-5 [39].

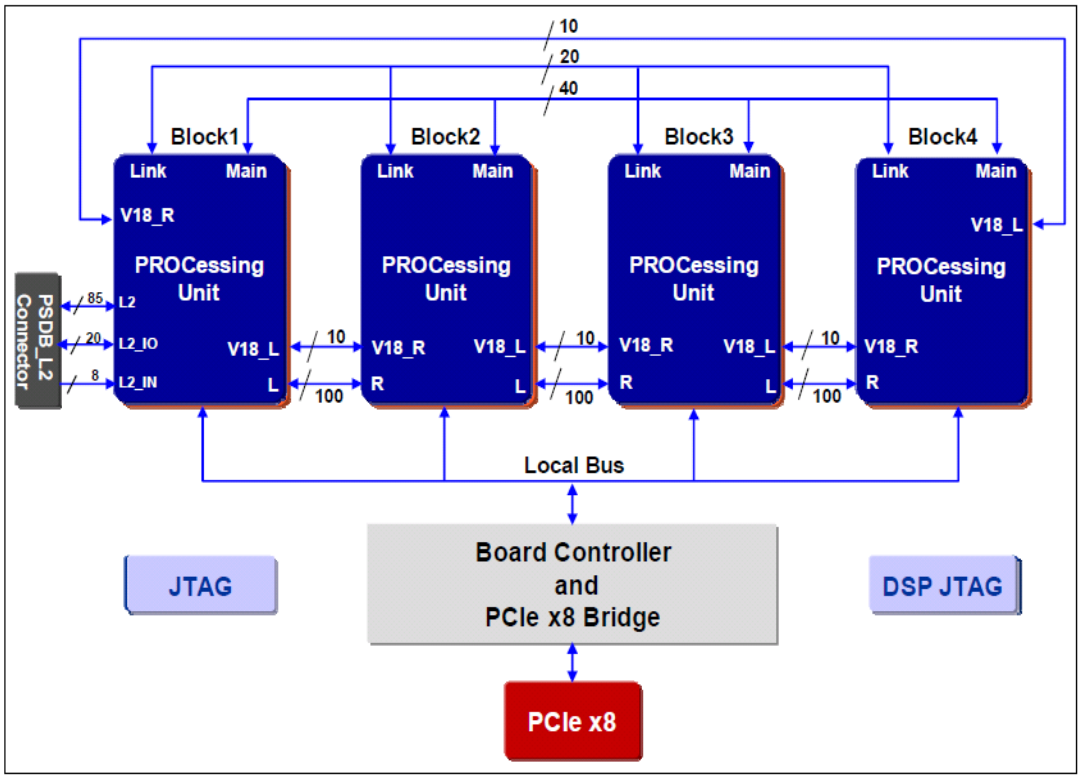

Figure 2-5: PROCStar III system overview [39]

The system consists of four processing units, Altera Stratix III SE260 FPGAs, and is capable of running at system speeds of up to 300 MHz. They communicate with a host via a PCI Express bus interface [38, 39]. Each processing unit contains the following components:

- Altera Stratix III SE260 FPGAs
- 256 MB on-board DDR II SDRAM (bank A)
- 2 x 2 GB DDR II memory (bank B and bank C) via SODIMM sockets
- 2 PSDB (PROCStar III Daughterboard) connection

The system is able to deliver high-performance FPGA solution with massive capability and throughput memory. Its memory performance is summarized in Table 2-1.

Table 2-1: PROCStarIII memory performance

	Bank A (on-board)	Bank B (SODIMM)	Bank C (SODIMM)
Capability	256 MB x 4	2 GB x 4	2 GB x 4
Performance (DDR)	667 MHz	667MHz	360 MHz
Throughput	16 GB/s	16 GB/s	8.5 GB/s

2.4.3 XtremeData XD1000

The XD1000, an FPGA coprocessor offered by XtremeData, Inc., is socket-compatible with an AMD Opteron processor [134, 137]. The XD1000 module can be directly inserted into an Opteron 940 socket by replacing a CPU and using all existing CPU peripherals on a motherboard. Such a configuration can integrate FPGA technology into a multiple-processor based system with minimum effort.

An example of applying the XD1000 module is the XD1000 development system that contains an Altera Stratix II EP2S180, 4MB off-chip SRAM and up to four 4GB DRAMs with 5.4 GB/s interface, as shown in Figure 2-6 [134]. Through the Opteron 940 socket, the FPGA can communicate with an Opteron microprocessor via HyperTransport (HT) bus that can provide up to 3.2 GB/s bandwidth. In addition, four 8MB programmable flash memory modules are available to be used for FPGA configuration files.

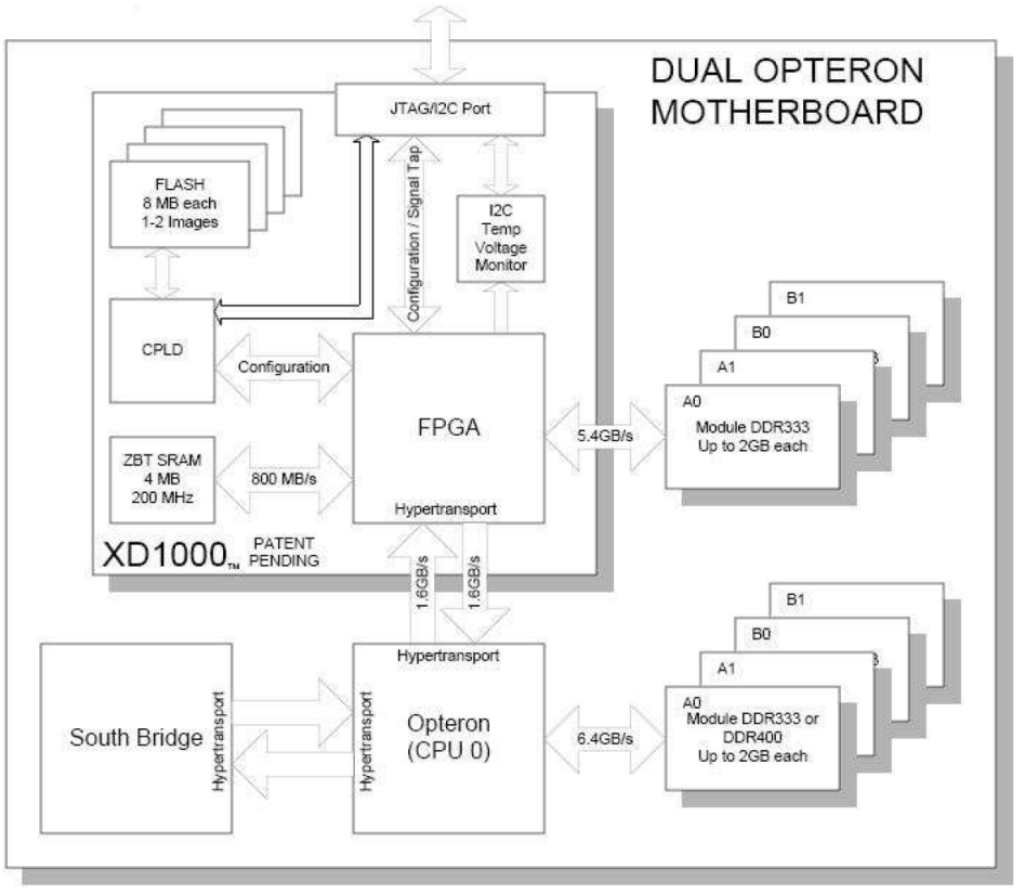

Figure 2-6: XD1000 system level block diagram [134]

Chapter 3 Molecular Dynamics

Molecular Dynamics (MD) is a type of computer simulation, which uses classical mechanics to model ensembles of atoms and molecules. It provides a projection of the laboratory experiment and acts like a "virtual experiment" [98]. MD is an example of N-body problem and inherits many of its characteristics, including the intensive computational complexity [139]. Still, it is different enough so that the basic flow of an FPGA algorithm may be very different for MD from it would be, say, for computing stellar dynamics (another classic N-body application).

Several efficient algorithms and techniques have been developed in the past decades to reduce MD computational cost. This chapter provides a brief overview of MD and those fast algorithms and techniques. Then well-known MD software packages are briefly reviewed, as well as several hardware accelerators. Lastly, FPGA works done by our and other groups are described.

3.1 MD Introduction

MD simulation is an iterative process in which the atoms and molecules interact with each other using the classical equation of Newtonian mechanics. During a simulation process, the force calculations and updates of particles' displacement and velocity are performed. It can be simply divided into two phases, i.e., force calculation and motion update, as illustrated in Figure 3-1 [46].

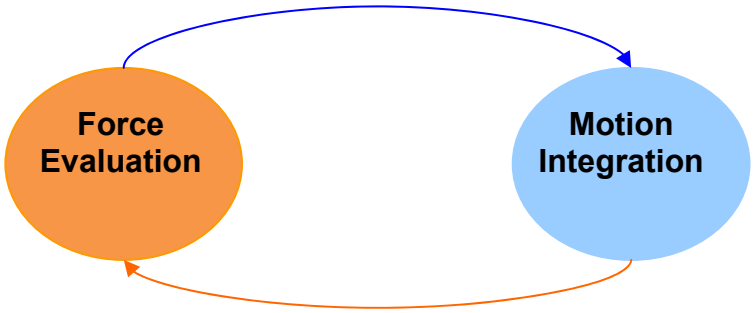

Figure 3-1: MD phase transaction

In each timestep, for a given position, the forces upon a particle are evaluated and the acceleration is calculated with Newtonian equations. This constitutes the first phase. In the second phase, the position and velocity of each particle are computed and updated and serve as inputs for the next iteration.

The forces of MD simulation come in two main categories: bonded forces and non-bonded. There are given in Equation 3-1, 3-2 and 3-3. Bonded forces contain various bonded types (bond, angle, and torsion) while non-bonded forces include Lennard-Jones (LJ), Coulomb, and hydrogen bond forces.

$$\vec{F} = \vec{F}_{bonded} + \vec{F}_{non-bonded} \quad (3\text{-}1)$$

$$\vec{F}_{bonded} = \vec{F}_{bond} + \vec{F}_{angle} + \vec{F}_{torsion} \quad (3\text{-}2)$$

$$\vec{F}_{non-bond} = \vec{F}_{LJ} + \vec{F}_{Coulomb} + \vec{F}_{hydrogen} \quad (3\text{-}3)$$

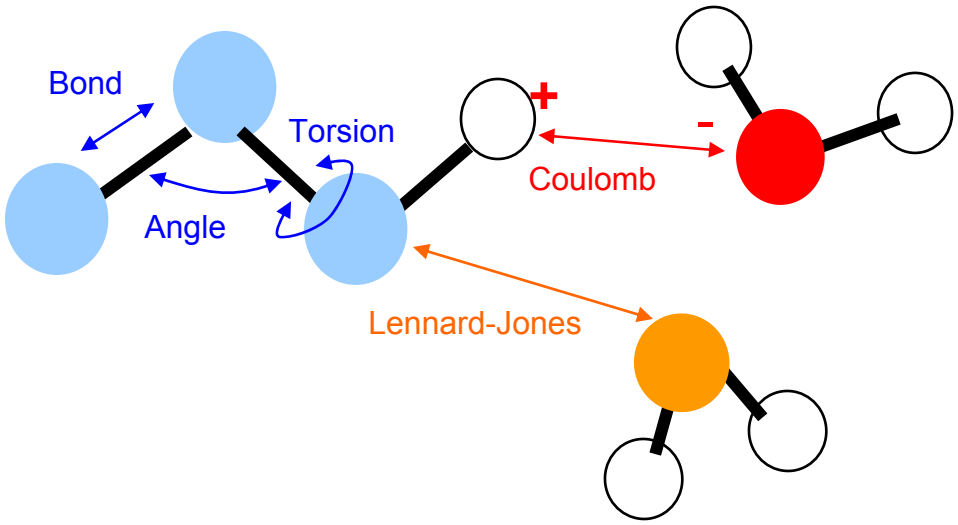

Figure 3-2: MD force field diagram

The simplified diagram of different force fields is shown in

Figure 3-2 [122]. Bonded covalent and hydrogen forces affect only neighboring particles while non-bonded forces evaluation involves interactions between all pairs of particles in the system, except those separated by covalent bonds. During a simulation, the number of bonded force evaluations scales linearly with the number of particles N. The number of non-bonded interactions, however, scales quadratically. This makes the computational complexity of non-bonded interactions $O(N^2)$ in Big O notation while the one of the bonded term is $O(N)$.

A common way to reduce the computational complexity of non-bonded forces is applying a cut-off. The Lennard-Jones (LJ) force vanishes quickly with the distance of a particle pair and is usually ignored when two particles are

separated beyond some cut-off distance, usually chosen between 8 to 12 Å [139]. The LJ force equation is given in Equation 3-4 and its potential graph is shown in Figure 3-3 [3].

$$\vec{F}_i^{\,LJ} = \sum_{j \neq i} \frac{\varepsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \cdot \vec{r}_{ji} \qquad (3\text{-}4)$$

where ε_{ab} and σ_{ad} are parameters related to particle types and r_{ij} is the relative distance between particle i and j.

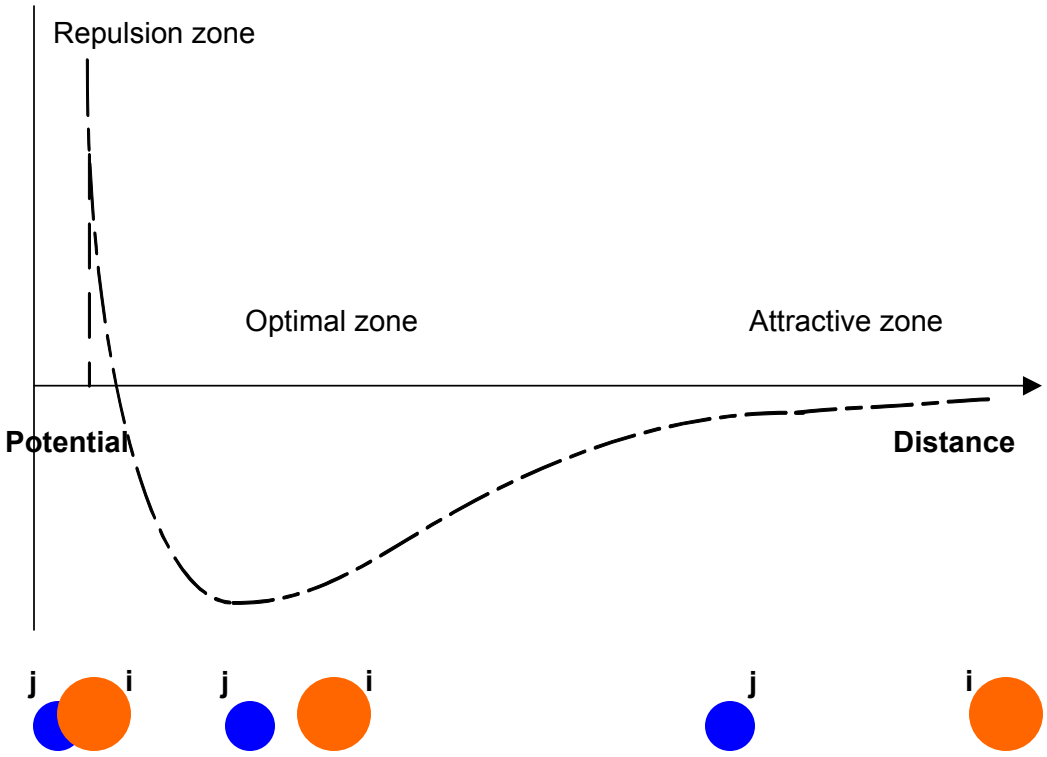

Figure 3-3: Lennard-Jones potential

When two atoms interact with each other, Pauli repulsion occurs, if the distance between interacting atoms becomes even slightly less than the sum of their contact distance. Van der Waals attraction occurs at short range, and rapidly dies off as the interacting atoms move apart. Thus, LJ force here is characterized as a type of non-bonded short-range forces. The electrostatic force, however, dies out slowly and it can affect atoms residing quite far apart. Thus

neglecting the electrostatic (Coulomb) force beyond a cut-off distance sometimes introduces serious artifacts into a simulation. The Coulomb force equation is given in Equation 3-5.

$$\vec{F}_i^{CL} = q_i \sum_{j \neq i} \left(\frac{q_j}{\left|r_{ji}\right|^3}\right)\vec{r}_{ji} \qquad (3\text{-}5)$$

where q_i and q_j are the particle charges and r_{ij} is the relative distance between particle i and j.

Although the equation itself is not complex, the computation process is time-consuming since all the particle pairs of the simulation domain must be evaluated. Several efficient algorithms have been developed to account for this non-bonded long-range force without actually interacting with all pairs of particles in the system [12, 21, 25, 112]. Those methods will be presented in the next section.

The second phase of MD simulation is motion integration which applies the computed forces to update the particle coordinates and velocities [30]. Various numerical schemes (such as the Verlet and leap-frog algorithms) are used, each with its own characteristics in terms of computational efficiency, numerical accuracy, energy conservation, and the ability to be used in long timestep integration. With regard to the last point -- it is often the case that all types of forces as well as particle coordinates and velocities are evaluated at the same frequency, say 1 femtosecond. Some advanced integrators, however, evaluate certain types of forces less frequently resulting in performance improvement.

Compared to force calculation, the motion update only occupies small amount of MD computational time. Thus it is not accelerated in this work.

3.1.1 Periodic Boundary Condition

Boundary conditions play an important role in MD simulation since they define the surroundings of the simulation system. Without any constraint upon the simulation boundary (free boundary), atoms near the boundary would have fewer neighboring atoms than those in the middle of the simulation system. In other words, those atoms surrounded by the surfaces would behave differently from the ones residing inside the simulation domain.

For a typical MD simulation, no matter how large the simulated system is, the particles N it contains would be negligible compared with the number of particles contained in a bulk system (of the order of 10^{23} particles) [100]. A common way to simulate the bulk phases and minimize the surface effect is employing periodic boundary conditions (PBC). This scheme enables an MD simulation to be performed using a relatively small number of particles in such a way that the particles experience forces as though they were in a bulk solution. A system with PBC is modeled by replicating the unit cell in all dimensions to form an infinite lattice. Thus, for a given particle, it not only interacts with the particles within the same cell but also other particles at its image cells. A two-dimension (2D) schematic representation of PBC is shown in Figure 3-4.

As a particle moves in the original unit cell, all of its images move in a consistent manner by the same amount. Since all of the images are replicants of the original unit cell with shifted coordinates, keeping only the original unit cell (the central one with shaded color in Figure 3-4) is enough to represent the entire MD system. When one particle leaves the unit cell by crossing the boundary, another cell (its image) enters the central unit cell as illustrated in Figure 3-5.

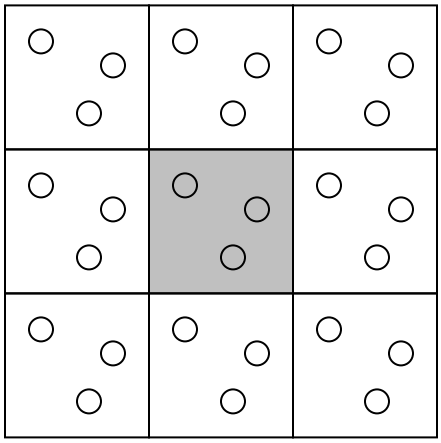

Figure 3-4: 2D schematic representation of periodic boundary conditions

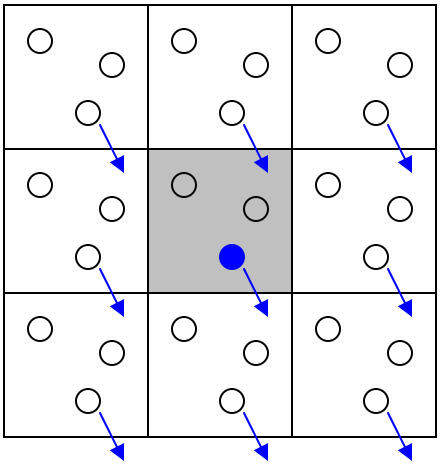

Figure 3-5: 2D representation of particles crossing the boundary under PBC

3.1.2 Energy Conservation

According to the laws of mechanics, the total energy (potential and kinetic) of an isolated biomolecular system should remain constant during a simulation. In practice, energy fluctuates on a short time period and drifts on a very long time scale because of the finite time step, Δt, used in numerical integrations. The longer the time step is chosen, the more physical time would be simulated as well as the better performance could be achieved for each computation. The time step, Δt, is determined by the fastest degrees of motion in a system (usually bond vibrations) and must be chosen short enough so that the system is stable and energy is conserved [44].

Energy conversation could also be violated because of the imperfect force evaluations performed in simulations to achieve better performance. For example,

with the cutoff scheme, an error is introduced when particles move back and forth across the cutoff radius if a perfectly smooth function is not used. Numerical errors (e.g., round-off errors or subtracting large forces in floating point arithmetic) are another source of energy drift.

Energy conservation often serves as one of quality metrics of measuring MD simulation stability. This alone is not sufficient to guarantee a realistic simulation and does not necessarily ensure the accuracy of MD simulation in reproducing physical phenomena. Even an ill-parameterized force field can still result in stable but unphysical trajectories [82].

3.2 Fast Algorithms for Computing Non-Bonded Interactions

Due to the intensive computation required, several efficient algorithms [3, 12, 21, 25, 112, 127] have been developed to accelerate the MD simulation, especially for non-bonded force calculation. In this section, we introduce two techniques to reduce the computational complexity of the non-bonded short-range force evaluation. This is followed by algorithms that improve the efficiency of the non-bonded long-range force computation. Brief descriptions of those algorithms are given in the following subsections. Some graphs in this section are drawn in two-dimensions (2D) for the convenience of illustration, but the discussions can be extended to three-dimensional (3D) systems without loss of generality.

3.2.1 Optimizing the Computation of Short-range Interactions

Neighbor Lists

For non-bonded short-range force calculations, particles only interact with each other at nearby distances and inter-particle forces are switched to zero when two particles are separated beyond some cut-off distance r_c. The basic idea of using neighbor lists is to construct a set of lists that consists of neighboring particles for each particle in the system [3]. A particle is included in the neighbor list of another particle when the distance between them is less than $r_c + r_m$ where r_c is a cut-off distance and r_m is a margin of safety. r_m should be large enough to ensure that no particle can travel into the cut-off sphere of another particle if it is not on the list of that particle before the next reconstruction of the neighboring list happens. Neighbor lists are updated periodically in a fixed time interval, or when the displacements of particles are larger than a predefined margin r_m.

The cost of constructing neighbor lists for an N-particle system scales as $O(N^2)$ with the system size. But with neighbor lists the computational cost of the force and potential evaluations can be reduced to $O(N*M)$ where M is the average length of a neighboring list [127]. M is proportional to the system density and pairlist radius and independent on the system size. As long as the neighbor list needs to be updated only rarely, this is a great savings.

Neighbor lists have proven to be efficient for small systems or those models in which particles' mobility is low, i.e., low temperature systems [130, 139]. It greatly

reduces the number of particles to be evaluated for a given particle. As shown in Figure 3-6, for particle i, all particles within its pairlist sphere are included in its neighboring list (except red-color particles and itself) and only blue-color particles residing between cutoff and pairlist sphere are overhead. Although reducing the frequency of neighbor list updates (or by increasing the margin) could reduce the computational expense of constructing neighboring lists, this would result in lower efficiency since more particles than actual need will be added into the list.

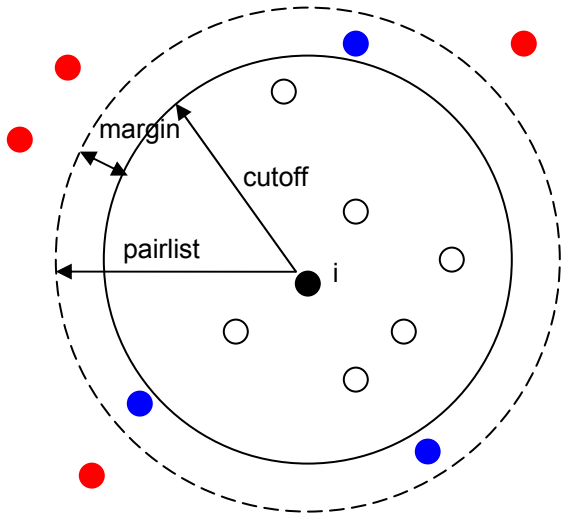

Figure 3-6: Neighboring list sphere

Cell Lists

The method of Cell Lists is another technique to improve the efficiency of non-bonded short-range force calculation [3, 139]. First, a simulation domain is partitioned into several cubic cells. The conventional length of the cell edge is equal to or larger than the cutoff distance. Particles are assigned to the

corresponding cell based on their coordinates. For example, given a particle P, it is assigned to the cell C, as shown in Figure 3-7 [46]. Thus, for particle P, it interacts only with particles of the cells to which it belongs and its 8 (for a 2D domain) or 26 (for a 3D domain) neighboring cells. If Newton's third law is used and only two-body interactions are evaluated, then only half of the neighboring cells need to be checked. This means, only five cells are required for a 2D system or 14 cells for a 3D system.

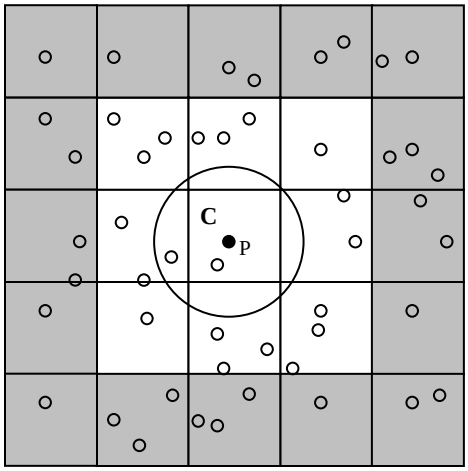

Figure 3-7: Illustration of cell-linked list algorithm

The cost of constructing cell lists scales as $O(N)$ with the number of system particles by scanning particles in the system and assigning them into the corresponding cells. Its construction effort is much less for large system compared to neighboring list method. Many more particles than needed, however, are included for the force evaluation process. Thus, these unnecessary interactions result in a negative impact on the system performance.

3.2.2 Computing Long-Range Interactions

Ewald Summation

Ewald Summation (or Ewald Sums) is an algorithm used to compute electrostatic forces in a system with periodic boundary conditions [34]. It was originally developed in 1921 to evaluate the electrostatic energy of ionic crystals [27]. Compared with direct calculation, it reduces the computational complexity from $O(N^2)$ to $O(N^{3/2})$ where N is the number of particles in a system.

For an N-particle periodic system, the Coulomb contribution to potential energy can be expressed as:

$$E_i = \frac{1}{2} \sum_{i,j=1}^{N} {\sum_{n \in Z^3}}' \frac{q_i q_j}{\left| r_{ij} + nL \right|} \qquad (3\text{-}6)$$

where the prime on the summation means that the sum is over all periodic images n and over all particles, except i = j if n = 0; L is the length of a unit cell.

The idea of Ewald Sums is to apply the Gaussian screening charge distribution with the opposite sign of a point charge into a system such that the total charges of this screening cloud exactly cancel out point charges [27]. This makes the electrostatic potential rapidly become zero due to the screening effect. The screening charge distribution determines how fast the electrostatic potential decays. The narrower charge distribution is, the more quickly the potential decays. In order to compensate for this additional screening charge distribution,

the same amount of charge distribution with the opposite of sign is also introduced into the system.

Now, the entire calculation is partitioned into two components. The first part, named the direct or real space sum, is contributed by the point charges screened by oppositely charged Gaussians distributions. The direct sum is a short-range term that converges quickly in real space, and its value can be computed with a cutoff scheme. On the other hand, the second part, named the reciprocal sum, represents the contribution of charged Gaussians. It is a smoothly varying periodic function that can be represented by a Fourier series in reciprocal space (or Fourier space). In addition, a correction term is required to exclude Coulomb self-interactions.

Coulomb potential energy E_i can be expressed as [46]:

$$E_i = E^{(r)} + E^{(k)} + E^{(s)} \tag{3-7}$$

$$E^{(r)} = \frac{1}{2}\sum_{i,j=1}^{N}{}' \sum_{n\in Z^3} q_i q_j \frac{erfc(\alpha|r_{ij}+nL|)}{|r_{ij}+nL|} \tag{3-8}$$

$$E^{(k)} = \frac{1}{2L^3}\sum_{k\neq(0,0,0)} \frac{4\pi}{k^2} e^{-\frac{k^2}{4\alpha^2}} \left|\rho(\vec{k})\right|^2, \ \rho(\vec{k}) = \sum_{j=1}^{N} q_j e^{-i\vec{k}\vec{r}_j} \tag{3-9}$$

$$E^{(s)} = -\frac{\alpha}{\sqrt{\pi}}\sum_i {q_i}^2 \tag{3-10}$$

where $E^{(r)}$ is the real space term, $E^{(k)}$ is the reciprocal space term, and $E^{(s)}$ is the self correction term and α is Ewald parameter.

With an optimal α, both computational costs of real part and reciprocal terms are $O(N^{\frac{3}{2}})$. The Optimal α can be determined by the following equation:

$$\alpha = \sqrt{\pi} \cdot \left(\frac{T_{real}}{T_{reci}} \cdot \frac{N}{V^2}\right)^{\frac{1}{6}} \tag{3-11}$$

where T_{real} and T_{reci} are time to compute the real space and reciprocal space term, respectively, and V is the volume of the simulation space. The cutoff of real space parts is:

$$r_c = \frac{\sqrt{-\ln \varepsilon}}{\alpha} \tag{3-12}$$

The cutoff of reciprocal space parts is:

$$k_c = 2\alpha\sqrt{-\ln \varepsilon} \tag{3-13}$$

where ε represents the tolerable error.

Particle Mesh Ewald (PME) Method

Particle Mesh Ewald (PME) is an efficient technique that has been widely used to evaluate the standard Ewald Sums due to its computational efficiency [21]. It approximates the reciprocal part of the Ewald Sums by a discrete convolution on an interpolation grid, using the discrete 3D Fast Fourier Transform (FFT) [97]. By choosing an appropriate splitting parameter α, the computational complexity can be reduced from $O(N^{\frac{3}{2}})$ to $O(N \log N)$.

Careful evaluation of the interpolation scheme and mesh size is required to achieve high simulation accuracy and speed. The basic procedure of is illustrated in

Figure 3-8 [46] and consists of three steps as follows.

1. Assign particles' charge to mesh points
2. Compute energy with FFT/IFFT
3. Interpolate forces back to particles

The complexity of the first and third steps is $O(N)$ while that of the second step is $O(N \log N)$ which dominates overall computations.

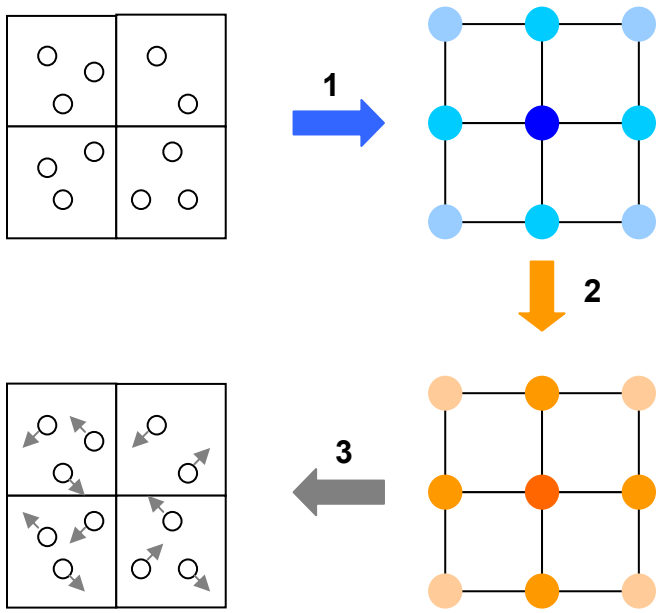

Figure 3-8: PME computational steps

The Lagrange interpolation scheme for charge assignments and force interpolation is adopted in the original PME method. However, since the Lagrangian weight function is only piecewise differentiable, energy and force

have to be evaluated separately. Most importantly, energy conversion cannot be satisfied. An improved scheme, Smooth PME (SPME), was developed to handle these difficulties [25]. A B-*spline* interpolation technique was used to replace the Lagrange-based method such that force can be obtained by directly differentiating energy and energy is thus conserved.

Multigrid Method

First introduced in the 1970s by Brandt [15], the multigrid method was originally used to solve partial differential equations (PDEs) and now has become an efficient technique to evaluate the electrostatic force. For a system which contains N particles, the computational cost of multigrid method is $O(N)$, whereas the direct calculation, the Ewald Sum, and PME are of order of $O(N^2)$, $O(N^{\frac{3}{2}})$, and $O(N \log N)$, respectively [97]. The general steps of the multigrid method are described as follows:

1. Interpolate and assign particles' charge on a grid
2. Apply multigrid method to solve Poisson's equation on the grid
3. Back Interpolate forces and energy from the grid domain to particle space

Compared with the standard Ewald Sum method and its variants such as PME and SPME, the multigrid method not only reduces the cost of force computations but also offers several advantages. These include no PBC requirement, ease of

parallelization, and no FFT. Therefore, the large communication overhead associated with the 3D FFT computation can be avoided.

Although the multigrid method helps reduce the computational complexity of force evaluation, to achieve a high-quality energy conversion, it consumes more time compared to Fourier-based schemes such as Ewald sum, PME, and SPME. For a given accuracy, it was reported that the multigrid method was 1.85 times as expensive as PME method on single processor [102] although it could become competitive with and eventually faster than the PME method for a parallel system. Most of highly tuned MD software packages employ PME or its variants to compute electrostatic forces. In order to deliver a widely acceptable MD accelerator, integrating with mainstream packages should also be taken into account.

3.3 MD Software Packages

A number of MD software packages have been developed and widely used in the community, each with its own features and goals [14, 59, 66, 76, 78, 94, 115]. Four of the most popularly used packages are briefly reviewed in this section.

3.3.1 NAMD

NAMD (NAnoscale Molecular Dynamics) is an MD simulation package that is regarded for its high scalability and parallel efficiency [94, 108]. It was written in C++ with Charm++ parallel objects and can be scaled up to hundreds of processors on a high-end parallel system [94]. It was developed by the Theoretical and Computational Biophysics Group (TCBG) of the University of Illinois, Urbana-Champaign (UIUC) and designed for high-performance simulation of large biomolecular systems [84]. NAMD uses VMD (Visualization of Molecular Dynamics) [129], which is a popular molecular graphics program, for the initial simulation setup and the visualization of trajectory analysis [60].

NAMD uses a hybrid decomposition scheme, combining the advantages of spatial and force decompositions, to achieve high scalability [94]. Particles are grouped together in a patch with the cell-list method. The length of the cell edge is extended to be slightly larger than a cut-off radius to give a margin, which allows atoms to move within a cell box between several time intervals and hydrogen atoms to reside with their parents in the same cell for faster distance checking. This could reduce the overhead of the cost of updating the cell-list. A number of computing objects are created and assigned to each processor to perform the force evaluation between neighboring patches. This allows NAMD to balance the system by dynamically distributing the computing objects to processors. The PME method [21] was adopted to compute long-range forces; it is evaluated every four time steps by default.

NAMD-Lite

Despite of NAMD's excellent reputation and popularity, extra complications may be introduced to the development due to its complex parallel structure. NAMD-Lite [53] was developed by UIUC TCBG group in 2005 to minimize design complications. It is a prototyping framework whose purpose is to simplify and smooth the development process and to provide a simpler way to examine and validate new features before integrating them into NAMD [85].

Although the performance of NAMD-Lite is not comparable to that of NAMD because of its serial implementation, it enables the integration of new algorithms and supports multiple schemes of force and energy evaluations, e.g., multiple switching and shift functions, PME, and multigrid. Thus, it makes our FPGA integration work relatively straightforward, allows us to explore new algorithms, and lessens implementation complications before porting designs into production NAMD codes.

3.3.2 GROMACS

GROMACS (GROningen MAchine for Chemical Simulations) was designed to achieve superior performance on a single processor [76]. It was originally developed at the University of Groningen, The Netherlands, in the early 1990s and written in ANSI C. It does not have its own force field, but is compatible with GROMOS, OPLS, AMBER and ENCAD. It also provides high flexibility to allow

users to add new force routines, specify tabulated functions, and customize the analysis.

A combination of many efficient algorithms have been embodied in GROMACS to accelerate its simulation, including optimization of neighbor search and inverse square root, specialized non-bonded kernels, customized techniques to avoid conditionals in the loop for PBC and force evaluations, and highly tuned hand-coded routines [115].

A new feature, the “eight shell domain decomposition” algorithm, has been implemented in the latest version (GROMACS 4) to replace the previous particle decomposition scheme. This minimizes inter-processor communications and increases scalability [59]. In addition, multiple-program multi-data (MPMD) PME parallelization, which divides domain tasks to direct and reciprocal spaces, is supported to reduce 3D FFT communication. Together with those newly added features, GROMACS now not only has extreme high performance on a single processor but also achieves high scalability on parallel machines.

3.3.3 Desmond

Desmond [14, 108] is a relatively new MD software package developed by D. E. Shaw Research (DESRES). It contains several novel features that significantly accelerate parallel MD simulations. These include a new parallel algorithm of spatial decomposition and a message passing technique that helps reduce

communication overhead. It has been shown that Desmond can deliver outstanding simulation throughput and high scalability [14].

Desmond uses both spatial and force decomposition methods to process the pairwise interactions in a way similar to other MD codes. In traditional spatial decomposition methods (Half-Shell as a typical example), the communication cost scales as $O(R^3)$, where R is a chosen cut-off distance. A new parallel algorithm specialized for the range-limited N-body problem, Neutral Territory (NT), is implemented to further reduce the amount of the data each processor has to import and scales well with the number of processors. The communication overhead of a given processor scales as $O(R^{3/2} p^{-1/2})$ where p is the number of processors [108].

Although Desmond can be configured to use either the single-precision or double-precision arithmetic mode, it is shown that the single-precision arithmetic can have much better performance by reducing the usage of memory and network bandwidth, and by allowing the usage of SIMD extension. Several numerical techniques have been applied to maintain numerical accuracy while gaining speed, such as fixed-point arithmetic for particle position updates during motion integration, and non-associativity via having a unique ordering of particles and computations.

Combining the reduction of the communication cost and the feature of single precision arithmetic computation, Desmond can achieve outstanding

performance in terms of high scalability and throughput for large commodity clusters.

3.3.4 ProtoMol

ProtoMol (PROTOtyping MOLecular dynamics) was developed by University of Notre Dame and is a high-performance object-oriented MD software package written in C++ [78]. One of its main features is the ease with which it allows prototyping of novel algorithms. This great flexibility has been demonstrated in its electrostatic force evaluation where several fast algorithms, including Ewald, PME, and Multigrid (MG) summation, have been implemented. Unlike other MD packages, where PME is typical used to evaluate the electrostatic force interaction and only applied for periodic boundary conditions, in ProtoMol MG summation can be used either in vacuum or periodic boundary conditions. This reduces the computational complexity to $O(N)$.

ProtoMol uses the method of replicated data to parallelize its computations [78]. Although this method reduces design complications, it does not scale well to large systems (with hundreds of processors) due to its high communication overhead.

3.4 MD Accelerators

Several methods have been used to accelerate MD simulation, including well-known ASIC-based systems such as MD-GRAPE [71], MD Engine [125], and Anton [109, 110]; graphic processing units (GPUs) [7, 80, 93, 96, 117]; and

FPGAs [11, 46, 47, 48, 49, 51, 69, 75, 104, 105, 106]. In this subsection, a brief review of various MD accelerators is presented.

3.4.1 Application-Specific Integrated Circuit (ASIC)

- **MDGRAPE-3**

  The MDGRAPE-3 system [23, 36, 37, 68, 71, 86, 87, 88, 121, 122, 123, 125] is a special-purpose hardware accelerator for classical MD simulations. Its architecture is similar to its predecessors, the GRAPE (GRAvity PipE) systems [71]. These were originally developed to solve gravitational N-body simulation and further extended to accelerate classical MD simulations.

  The MDGRAPE-3 system can achieve petaflops performance in the normal mode. It consists of a host computer and special-purpose MD engines. The special-purpose MD engines are only responsible for non-bonding forces evaluations, including electrostatic and intermolecular forces, which dominate most of the computations while leaving the remaining work to the host.

  The MDGRAPE-3 chip, which is able to reach 200 Gflops peak performance at 300 MHz, has 20 force calculation pipelines and can accommodate up to 32,768 particles. It uses both floating-point and fixed-point arithmetic while most of calculations are carried out by single precision mode. The complete MDGRAPE-3 system contains 4,778

dedicated MDGRAPE-3 chips, each capable of performing 200 Gflops. Its peak performance can therefore reach one petaflops in total.

- **Anton**

  Anton, a special-purpose massively parallel machine, promises to deliver the classical MD simulation through the millisecond-scale [109]. This requires improving on currently available MD simulation packages by the three orders of magnitude (500x for NAMD, 100x for Blue Matter, and 80-100x for Desmond) [109]. It was implemented by D. E. Shaw Research and became operational on late 2008 [110]

  Anton uses the same “Neutral Territory” method as the one in Desmond to reduce communication overhead and implements special-purpose logic to greatly accelerate the most time-consuming tasks of MD simulation. The initial system consists of 512 MD-specific processing nodes. Each node contains an MD computation engine on a single ASIC. Nodes interact with one another with a specialized high-speed communication network organized in an 8 x 8 x 8 toroidal mesh.

  The high throughput interaction subsystem (HTIS) is mainly responsible for the pairwise interaction as well as the charge distribution and force interpolation. It has 32 Pair-wise Point Interaction Modules (PPIMs), each containing a 26-stage force pipeline (running at 800 MHz) to compute the force between particle-pairs with the fixed-point arithmetic logic. The

flexible subsystem controls the ASIC and handles the remaining tasks, including the bonded-force computation, the FFT and force correction, and integration tasks.

3.4.2 Graphics Processing Units

With their tremendous arithmetic performance and wide availability, modern graphics processing units (GPUs) provide a compelling alternative for numerically intensive computations. It is well-known that GPUs can perform over a trillion floating point operations per second, and that they are inherently data parallel [96, 117]. Recently, with the introduction of high level programming languages such as CUDA (Compute Unified Device Architecture), GPUs have become even more appropriate for scientific computations. Another key advantage of GPUs, compared to other hardware accelerators, is their cost efficiency due to the high demand and growth of the gaming market.

With the features of massive data parallelism, high peak arithmetic and memory bandwidth, GPUs can execute small tasks in parallel to substantially outpace traditional CPU performance for certain particular applications. MD simulation is one of these appropriate candidates due to its intensive computational complexity.

- NAMD-GPU

  GPUs have been used in NAMD to perform non-bonded short-range force (both LJ and short-range part of electrostatic forces) interpolation while the

remaining tasks are left to CPU [96, 117]. Instead of neighbor lists that were commonly adopted in software codes, cell-lists are implemented and used in NAMD-GPU to avoid the long latency of GPU on-board memory access since on-chip caches are too small to accommodate whole neighbor lists. Piecewise linear interpolation is used to evaluate force and energy functions. It was demonstrated that four Tesla GPUs could outperform a cluster with 16 quad-core CPUs on the NCSA Tesla-based Lincoln cluster for the STMV (virus) benchmark that consists of one-million particles [91]. A similar technique is also used in Folding@home to accelerate the LJ force calculation in GROMACS [31, 35]. An overview of GPU general architecture is shown in Figure 3-9 [92, 93].

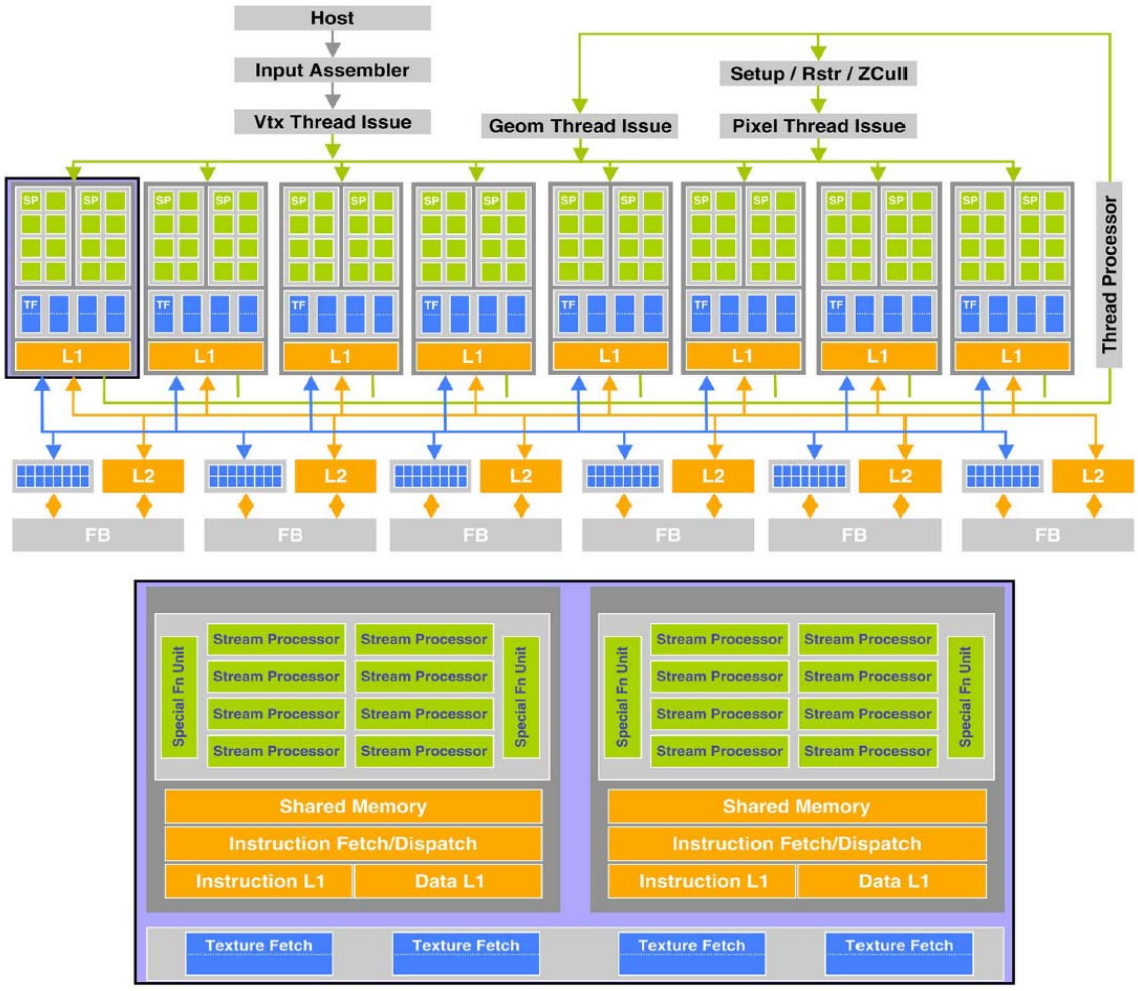

Figure 3-9: NVIDIA GeForce 8800 GTX architecture (diagram courtesy from NVIDIA)

- HOOMD/HOOMD-blue

  HOOMD [7] stands for Highly Optimized Object-oriented Many-particle Dynamics and HOOMD-blue is a direct continuation of HOOMD project. HOOMD was developed at Ames Laboratory and is able to perform general-purpose MD simulations by taking advantage of modern GPUs to attain a level of performance equivalent to dozens of processor cores on a

fast cluster. It was reported that MD simulation was fully implemented on NVIDIA GeForce 8800GTX. A speedup of up to nearly 60x was reported for Lennard-Jones liquid simulations with neighbor lists [7].

In [7], Anderson et al. presented the software re-engineering schemes of how to map MD CPU codes to GPU implementations in details. In addition, a particle reordering scheme with Hilbert curve was implemented to improve the cache-hit rate and thus increase performance.

The current release of the GPU implementation only supports the evaluations of bond, angle and non-bonded short-range force (LJ force). Electrostatic force computation is under active development. Future enhancements include support of dihedral potential and mixed precision (single and double) calculations.

3.4.3 FPGAs

FPGAs are a type of programmable logic device. They contain abundant programmable logic and interconnect fabric. In most FPGAs, the basic logic element usually contains lookup tables and flip-flops. Customized logic functions can be performed by configuring and connecting these elements.

The main advantage of the FPGA is its programming flexibility that allows users to explore novel algorithms, optimize implementations with various configurations, and quickly respond to any design change. Recently, FPGAs have increased their capability with added capacity and several advanced

features. These include hard components such as multipliers and block RAMs, but also support for a new generation of semi-hard floating point cores. With these, FPGAs have expanded their application space from traditional domains of signal and image processing to linear algebra and even scientific computing. Much work has recently been done accelerating MD simulation on FPGAs. Most of the prior studies have concentrated on the most computationally intensive tasks, e.g., the non-bonded short-range force calculation, while only few of them explored other kernels. We now provide a brief review of this FPGA work.

In 2004, N. Azizi, et al. [11] implemented a preliminary MD system on Transmogrifier 3 (TM3) system which contained four Virtex-E 2000E devices and one 256k x 64 bit external SRAM. The all-to-all LJ forces calculation and velocity Verlet algorithm were implemented. Numerical computation was carried out by fixed-point arithmetic with various scaling factors and precisions. The LJ force was computed with table look-up interpolation and the system was able to perform up to 8,192-particle simulation. The support of multiple particle types, however, was not reported. The system was validated with an academic C-based software MD simulation, MD3DLJ, and on the order of 1% RMS error was reported for both force and energy evaluations. The details of error analysis about numerical precision and scaling factors, however, were not given. The performance was reported to be 0.29 that of the original software code running on a PC with a 2.4 GHz Pentium 4 due to limited memory bandwidth and low clock speed. It was reported that a speedup of 20X could be achieved if the

implementation were scaled to more advanced FPGA devices and the memory system improved.

Several papers were presented by R. Scrofano, et al. [104, 105, 106]. In [105], an implementation of direct computation with double precision FP arithmetic was carried out to calculate LJ force and potential. A throughput of 3.9 GFLOPS was achieved with two force pipelines placed on a virtual Virtex-II Pro XC2VP125-7 chip.

In [104], a similar force pipeline with single precision FP arithmetic was implemented on a SRC 6e MAPstation. The SRC system has two Xilinx Virtex-II XC2V6000-4 FPGAs of which only one was used. The force pipeline was responsible for LJ calculations as well as Coulomb force evaluations that were approximated by the cut-off approach. Compared to more sophisticated methods like PME, 5% or less difference was estimated due to the shift-force approximation. A neighbor-list scheme was adopted to reduce the number of particle pairs that need to be evaluated. A 2x speedup was obtained; only one pipeline was implemented due to the area and memory constraints.

A modification was made in 2006 to improve the accuracy of Coulomb force calculation. Smooth Particle Ewald Sums method [25] was implemented to replace the previous cut-off shifted-force approximation for electrostatic force evaluation [106]. Only the real-space part of SPME was accelerated in hardware while the reciprocal-space part was still executed in software on the host. Direct evaluations of non-bonded short-range forces were performed with single-

precision floating point arithmetic, except $erfc(x)$ and e^{-x^2} which were approximated by table lookup interpolation. Two testbenches of 52K and 33K particles were reported and 2.7-2.9x speedup was achieved over software codes.

Another effort was made by Lee [75] to accelerate the reciprocal part of SPME on a Xilinx XC2V2000 FPGA. The computation was performed with fixed-point arithmetic that has various precisions to improve numerical accuracy. Due to the limited of logic resources and slow speed grade, the performance was sacrificed by some design choices, such as the sequential executions of the reciprocal force calculation for x, y, and z directions and slow radix-2 FFT implementation. The performance was projected to be a factor of 3x to 14x against the software implementation running in an Intel 2.4GHz Pentium 4 processor.

In [69], a simplified version of NAMD was accelerated on the SRC-6 MAPstation (MAP) platform. The modified NAMD code eliminated the bonded force calculation. Only non-bonded short-range forces were evaluated by table look-up interpolations with single precision FP arithmetic in hardware. Several design choices were analyzed and implemented. It was reported that a 1.3x speedup can be achieved against the software by using both FPGAs on a single SRC-6 MAP and 3x speedup can be obtained with a series-E MAP for a simulation of 92K particles system.

H. Guo, et al. [51] presented an FPGA-accelerated MD simulation system that used cell-list scheme and filter logic to remove extraneous pairwise interactions.

The design was implemented on the system that consisted of a 2.66 GHz Pentium 4 and Xilinx Virtex-II-Pro FPGA and it was reported that a 12x speedup was achieved over the software for small benchmarks. The LJ force was evaluated with Lagrange linear interpolation in fixed-point arithmetic. The electrostatic force computation, however, was not supported and the analysis of force accuracy and details of filter design were not presented. Compared to our filter design presented later, about 50% filter efficiency was reported and the issues of load balance and particle queuing were not addressed.

3.4.4 Previous Implementations

A high performance FPGA-based MD system was designed and implemented by Yongfeng Gu in our research group in 2007 [46, 47, 48, 49]. It was reported that the system obtained substantial (5-9x) speedup over a highly tuned MD package and supported MD simulations up to 256K particles [46]. The system was composed of a standard PC with 2004-era COTS FPGA (Xilinx Virtex II Pro VP70) in which the pipeline accelerator was implemented. Several innovations, including a novel arithmetic mode, use of cell-lists, off-chip memory management, and non-bonded force exclusion, were included in the system.

In this section, several features of the FPGA-based MD system are highlighted and reviewed briefly.

- Force Computation Schemes

  In MD simulation, a common method to compute inter-particle forces is to use table lookup and interpolation. This method can speed up the evaluation of forces that are expressed by the formulas containing transcendental functions and complex operations [133]. A loss of accuracy in the resulting trajectories of particles, however, is unavoidable. Besides, due to rapid changes of non-bonded force values, using an integer representation with fixed scaling is not sufficient to accommodate the dynamic range of force field. Moreover, the cost of floating point arithmetic is still expensive (for the 2004-era FPGA chips in which the system was implemented). A novel arithmetic mode, semi-floating point, and table lookup interpolation using the third-order orthogonal polynomial are employed to tackle these challenges.

- Interpolation Methods

  Given fixed computational resources, how to optimize a design to meet various targets is not straightforward. The number of points within each interval section, the type of polynomial, and the order of interpolation are the variables that affect computational quality. In addition, the number of pipelines that fit into a chip should also be taken into account since it greatly affects resulting acceleration speed.

In order to improve interpolation accuracy, a non-uniform separation method is used. This is based on the observation that the curve of r^{-x} changes rapidly over the range of possible interaction radii as shown in

Figure 3-10 [46]. The curve is flatter at well-behaved sections where two particles are separated much far away. In our FPGA-based system, the length of each section is twice that of the previous section; however, each section is partitioned into the same number of interpolation intervals.

Three interpolation schemes, Taylor, Hermite, and Orthogonal, were considered. They were evaluated from different perspectives: interpolation order, number of points within each interval section, and relative RMS error. Orthogonal polynomial interpolation was found to offer the best performance under the same combination of interpolation order and interval numbers [46].

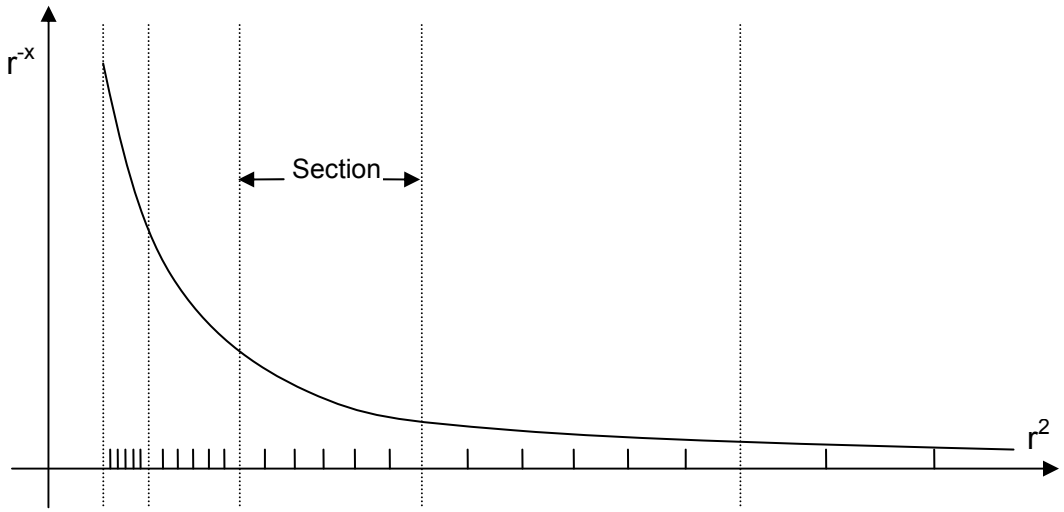

Figure 3-10: Logarithmic intervals for r^{-x} interpolation

- Semi-Floating Representation

In most MD software packages, force calculation is performed by floating point (FP) arithmetic. The expensive cost of FP operations, however, makes use of FP computations difficult. On the other hand, the dynamic range of the non-bonded force is too big to be represented by a fixed point numbering system. In order to keep good simulation quality, as well as to reduce computational costs, an efficient alternative, semi-floating point, is proposed. This method takes advantage of the characteristics of the MD force computation.

For a given input r^2, the scaling factor (exponent) at each stage of interpolation pipeline can be determined by the position of leading "1" of r^2. In addition, differences in scaling factors of addends are known. The key observation is that there are only a limited number of combinations of scaling factor pairs. This can be used to create an efficient numbering system, the semi-floating point representation. This is a fixed point numbering system whose data format is 35-bit fixed point number, but with a dynamic binary point which is able to shift based on the value of the input data (r^2). The efficient precision obtained by using semi-floating format is based on the considerations of computation resources and acceptable energy fluctuation [46]. Its use allows designs to take advantage of low latency and fewer resources of fixed point computations while yielding acceptable results.

Chapter 4 Force Pipeline Design and Optimization

Accelerating MD simulations using HPRC have been widely explored and studied. However, delivering cost-effective production applications has proved challenging. In addition, given the intensive competition from multicore and other types of hardware accelerators such as GPUs, a question arises as to whether MD using HPRC can be competitive. In this chapter, we concentrate on the MD kernel computation: determining the nonbonded short-range force between particle pairs. We examine it in detail to find the performance limits under current technology and methods. We systematically explore the design space of the force pipeline with respect to arithmetic algorithm, arithmetic mode, numerical precision, and various other optimizations. Moreover, we use Altera floating point datapath compiler to further optimize the implementations. Various designs are presented and examined. Several design considerations and challenges are also addressed.

4.1 Overview

As described in Chapter 3, forces in MD simulations can be classified into two categories: bonded and nonbonded. For an N-particle MD system, the computational complexity of nonbonded force evaluations is $O(N^2)$ whereas that of the bonded term is $O(N)$. A common way to reduce the computational complexity of nonbonded forces (LJ and Coulomb) is applying a cut-off that makes a particle interact only with its neighboring particles within the cut-off

radius. This drastically reduces the number of particle pairs that needs to be evaluated, even given the more complex bookkeeping scheme required to keep track of cell- or neighbor-lists.

However, a problem with cut-off is that, while it offers sufficient accuracy for the evaluation of the rapidly declining Lennard-Jones force, neglecting the slowly decaying Coulombic force beyond a cut-off radius often introduces intolerable errors into MD simulations. Lennard-Jones and Coulomb forces for particle i can be expressed as follows:

$$\vec{F}_i^{LJ} = \sum_{j \neq i} \frac{\varepsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \cdot \vec{r}_{ji} \qquad (4\text{-}1)$$

$$\vec{F}_i^{CL} = q_i \sum_{j \neq i} \left(\frac{q_j}{|r_{ji}|^3} \right) \vec{r}_{ji} \qquad (4\text{-}2)$$

where ε_{ab} and σ_{ad} are parameters related to particle types and r_{ij} is the relative distance between particle i and j. q_i and qj represent the charges of particle i and j.

As described in the Chapter 3, in order to reduce the expensive computational cost as well as obtain high simulation quality, several numerical schemes have been developed to solve the Poisson equation that translates charge distribution into a potential distribution. These schemes usually split the original Coulomb force curve into two parts with a smooth function $g_a(r)$, i.e., a rapidly declining short-range part, and a flat long-range part. This is shown in Equation (4-3) [46]

$$\frac{1}{r} = (\frac{1}{r} - g_a(r)) + g_a(r) \qquad (4\text{-}3)$$

where two components are illustrated in Figure 4-1.

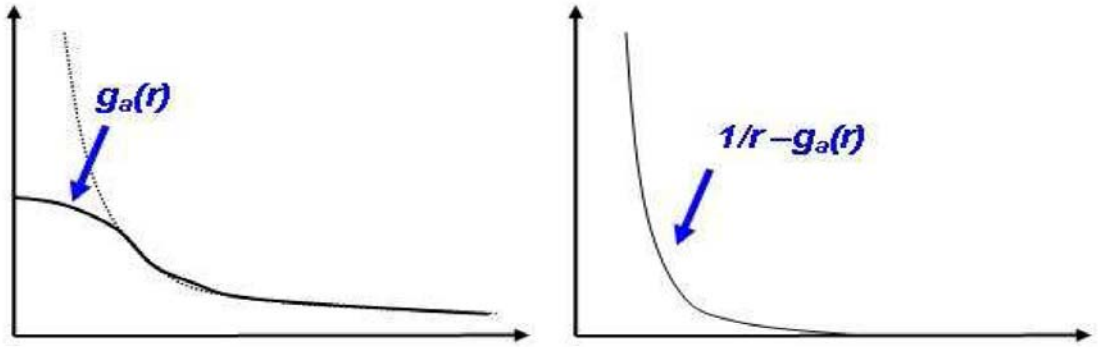

Figure 4-1: Smooth function (a) Left is the original 1/r and the smoothing function $g_a(r)$. (b) Right is 1/r - $g_a(r)$

The short-range part, $\frac{1}{r} - g_a(r)$, declines quickly enough such that it can be neglected beyond cutoff radius while $g_a(r)$ changes slowly with distance. Hence, the short-range part can be evaluated in the same way as LJ force and the entire nonbonded *short-range* force can be expressed as:

$$\frac{F_{ji}^{short}}{r_{ji}} = A_{ab} r_{ji}^{-14} - B_{ab} r_{ji}^{-8} + QQ(r_{ji}^{-3} + \frac{g_a^{/}(r)}{r}) \qquad (4\text{-}4)$$

where A_{ab} and B_{ab} are pre-computed coefficients and QQ is the product of charge of two interacting particles. The selection of the smoothing function depends on users' requirement and preference. It can be a Gaussian distribution, as the one used with Ewald Sums and its variants, or others.

4.2 Pairwise Nonbonded Force Computation

In this section, several FPGA implementations of Equations 4-1 and 4-2 will be presented. All are fully pipelined and, on every cycle, input pairwise positions and output the corresponding forces. Numerous design axes are described at the beginning of this section. The ones that visibly affect the design are follows: direct computation versus lookup table with interpolation (LUT); for LUT, order of interpolation; for direct, whether the Altera FP compiler is used or the FP cores directly; and whether fixed-point arithmetic is used for part of the computation.

The last two require further explanation. The Altera floating point datapath compiler (FPC) optimizes floating point datapaths by removing redundancy among operators and by making trade-offs in using various component types, e.g., using hard or soft components as available [72].

The second axis requiring explanation is float versus hybrid fixed/float. The problem arises from the force accumulation stage at the end of the pipeline. During force computation, the coordinates of particle i and particle j are read to compute corresponding forces, and the new partial forces are accumulated with the running total force f_i and f_j until all neighboring particles of reference particle i are processed. A floating point addition usually requires more than a single cycle, although since it is pipelined this does not necessarily change throughput. A read-after-write hazard occurs if the same particle's force is referenced on successive cycles. The hazard caused by force accumulation on particle j can be

removed without applying Newton's third law, which results in doubling the number of force evaluations [138]. Such schemes, however, still cannot reduce the hazard on the reference particle i.

There are several solutions to solve this problem: (i) stalling the pipeline, (ii) applying a single pipelined floating point unit [140], (iii) orchestrating particle processing so that hazards are avoided, (iv) accumulating the forces by using a more complex structure, such as a reduction tree, or (v) saving the force in integer format rather than floating point. In the last case, addition can be completed in a single cycle. Integer operations are also more area efficient than floating point, and if it done carefully, result in no loss of precision. The GROMACS code and the Protein Explorer, for example, both use mixed fixed/floating point [115, 123]. Moreover, another benefit of using fixed point rather than floating point is to overcome the shortcoming of non-associativity of floating point arithmetic. During the force accumulation at the end of pipeline, the range of force values could vary substantially. In the extreme case, a loss of low order bits occurs and cannot be recovered later for the floating point arithmetic.

In the rest of this section, we show how these alternatives cause the force pipeline and performance to vary.

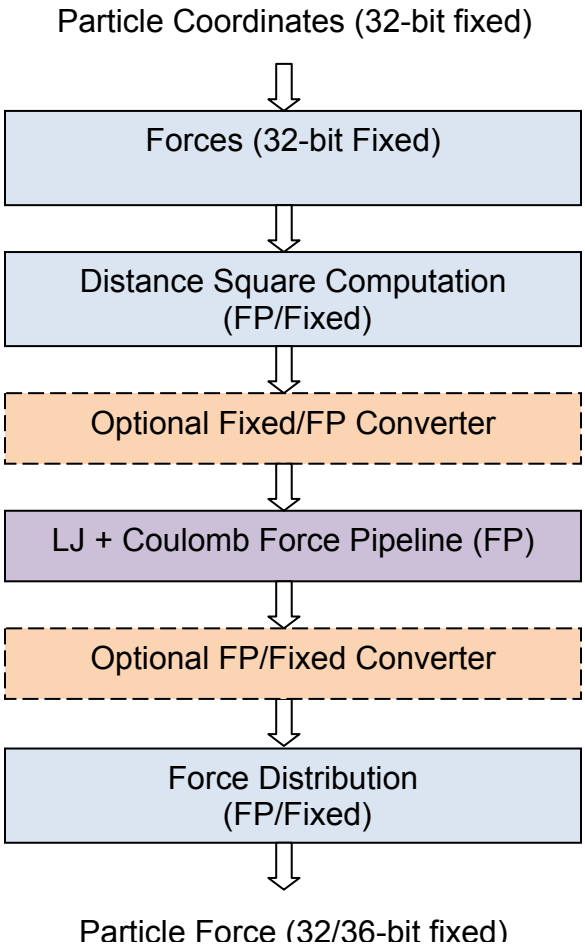

Figure 4-2: Data flow of nonbonded short-range force pipeline

4.2.1 Force Pipeline Design

Figure 4-2 shows the data flow of the nonbonded short-range force pipeline. The pipeline consists of several functional blocks that are now described.

1. **Periodic Boundary Condition Logic:** The first stage of the force pipeline calculates the distance between two particles. If periodic boundary conditions (PBC) are applied, the distance computed must be the shortest one between any mirror images of each particle. The particle distance is computed first and tested with the length of the simulation box. The operation can be performed in either fixed or floating point.
2. **Distance Squared Computation logic:** This squares the three-dimensional (3D) components of the distance between two particles. The operation can be performed in either fixed or floating point.
3. **LJ+Coulomb Force Logic:** This is the core of force computation. Several implementations are described and then examined from various perspectives such as resource utilization and simulation quality. The computation is performed with floating point arithmetic because of large dynamic numerical range required for nonbonded short-range forces. The designs can be implemented either with table lookup (with various interpolation orders and table densities) or direct computation (DC) (with the choice of using floating point (FP) cores or the FP compiler).
4. **Force Distribution Logic:** This applies the computed pseudo force to the 3D components of the distance between two particles. The operation is processed in either fixed or floating point. At the end, forces are converted to fixed point for the accumulations.

Figure 4-3 illustrates the major functional units of the force pipelines. The force function evaluators are the diamonds marked in red; these are the components that can be implemented with the various schemes. The other units remain mostly unchanged throughout the designs. The three function evaluators are for the r^{-14}, r^{-8}, and r^{-3} components of Equation 4-4, respectively. In particular, the Coulomb function uses the r^{-3} term but also includes the smooth function shown in Equation 4-3.

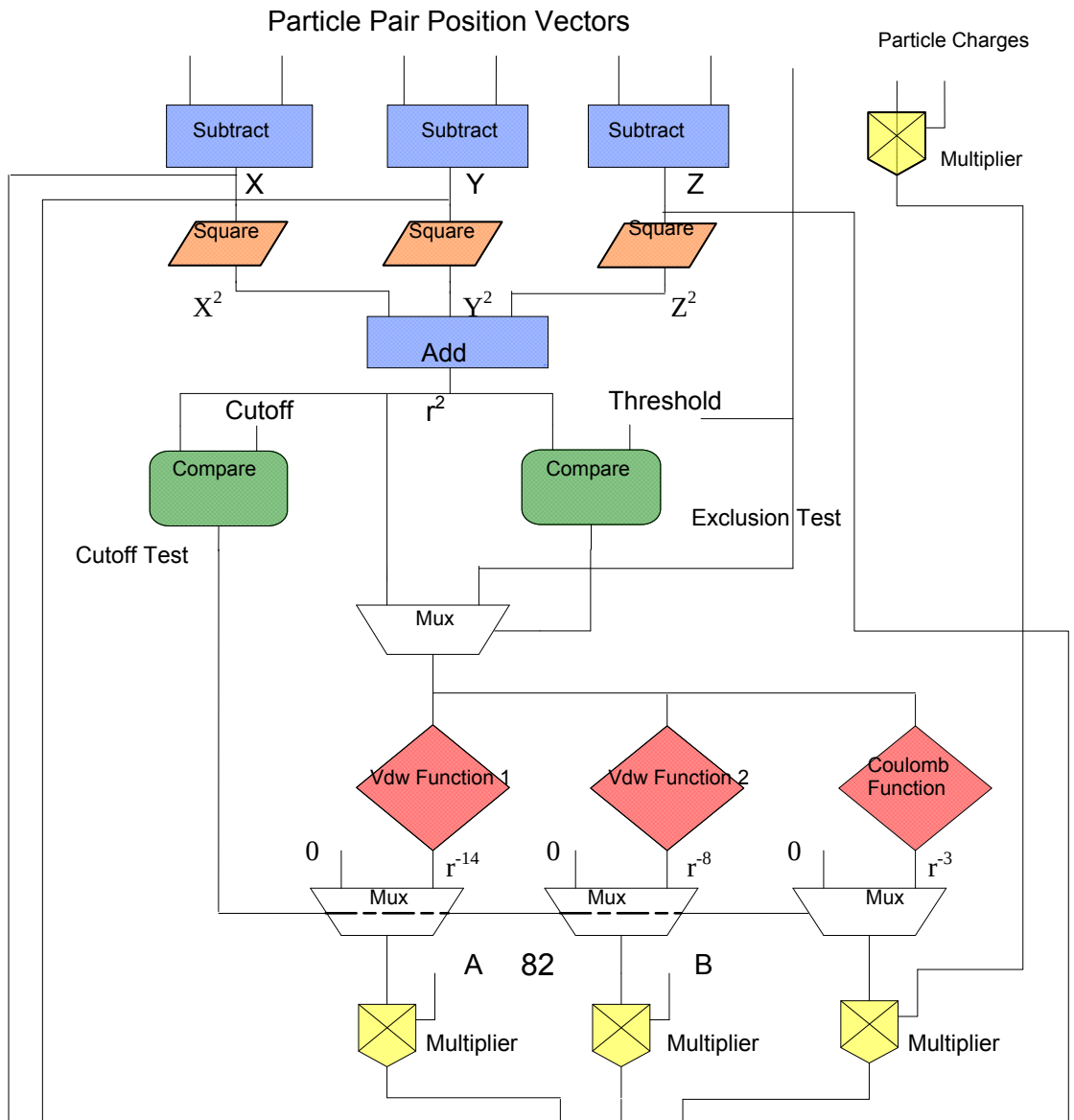
Particle Pair Position Vectors
Particle Charges
Subtract
Subtract
Subtract
Multiplier
X
Y
Z
Square
Square
Square
X2
Y2
Z2
Add
Cutoff
r2
Threshold
Compare
Compare
Exclusion Test
Cutoff Test
Mux
Vdw Function 1
Vdw Function 2
Coulomb Function
0
r-14
0
r-8
0
r-3
Mux
Mux
Mux
A
82
B
Multiplier
Multiplier
Multiplier

Figure 4-3: Force pipeline template

4.2.2 Table Look-up with Interpolation

Since nonbonded force calculations constitute the "inner loop" of MD, considerable care is taken in their implementation. A major consideration is whether to compute them directly or to use table look-up with interpolation. Interpolated solutions to function computations have been used in computing since its earliest days. Their application to MD dates at least to 1983 when Andrea et al. reported using a fifth order polynomial approximation [8]. Wolff and Rudd studied the idea in more details and proposed using first order interpolation, but also increasing the table size as necessary to ensure sufficient accuracy [133].

We now briefly describe the method. Early versions used a single table for the entire LJ force as a function of particle separation [11, 48, 133]. The index used is $\left|r_{ij}\right|^2$ rather than $\left|r_{ij}\right|$ so as to avoid the costly square-root operation. This

method is efficient for uniform gases where only a single table is required, because it is not necessary to distinguish atom types, and the lookup table is a function only of displacement. As the Lennard-Jones force depends on atom types, however, simulations of T different atom types require $T^2/2$ tables; this is prohibitive for FPGAs for many classes of simulations.

A different method, also used in most MD codes, uses multiple tables, one each for a different part of the computation [46]. For example, Equation 4-4 can be rewritten as a function of r_{ij}^2:

$$\frac{F_{ij}^{short}(|r_{ij}|^2(a,b))}{r_{ij}} = A_{ab}R_{14}(|r_{ij}|^2) + B_{ab}R_8(|r_{ij}|^2) + Q_aQ_bR_3(|r_{ij}|^2) \qquad (4\text{-}5)$$

where $R_{14}, R_8,$ and R_3 are lookup tables indexed with $|r_{ij}|^2$ and A_{ab}, B_{ab} are parameters related particle types (type “a” and “b”) and Q_a and Q_b are the particle charges.

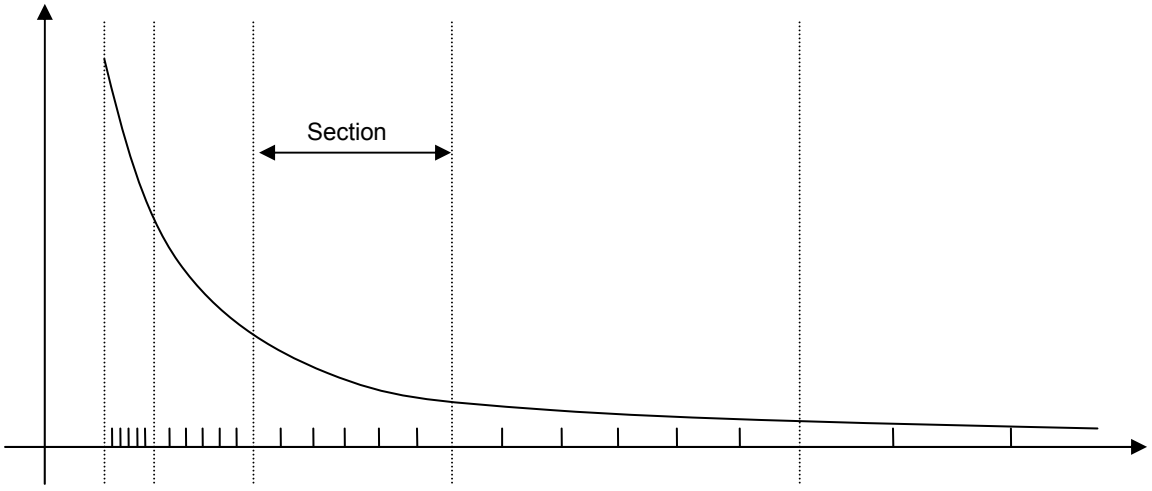

Figure 4-4: Table look-up varies in precision across r^{-x} interpolation. Each section has a fixed number of intervals.

The interval scheme used in the tables is shown in

Figure 4-4 [46]. Each curve is divided into several sections along the X-axis such that the length of each section is twice that of the previous [46]. Each section is cut into the same number of intervals N. To improve the accuracy, higher order terms can be used. When the interpolation is order M, each interval needs (M+1) coefficients and each section needs N * (M + 1) coefficients:

$$F(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \qquad (4\text{-}6)$$

Equation 4-6 shows third order interpolation with coefficients a_i. Accuracy increases with both the number of intervals per section and the interpolation order. The dataflow of a third order interpolation pipeline is illustrated in

Figure 4-5.

$$f(x) = ((A * x + B) * x + C) * x + D \qquad (4\text{-}7)$$

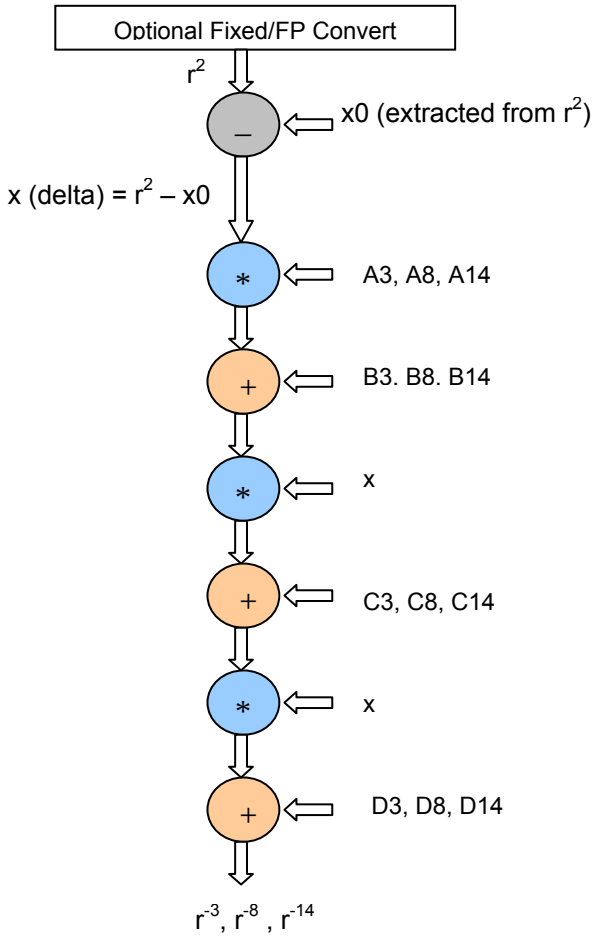

Figure 4-5: Arithmetic flow of a function evaluated with table lookup and 3rd order Interpolation.

We now present a sample of the methods of force computation adopted in widely used MD packages and systems (see Table 4-1). Clearly, there are a wide variety of parameter settings that have been chosen with regard to cache size (CPU), routing and chip area (Anton), and the availability of special features (GPU texture memory). The parameters also have an effect on simulation quality, which will be addressed later.

Table 4-1: Sample implementations of table look-up interpolation

	Order	Index	Interval density	Segment size
NAMD (CPU) [84]	2	r^2	768 (total)	Exponentially
NAMD (GPU) [117]	1	r^{-1}	512 (total)	Exponentially
CHARMM [89]	2	r^2	10-25 per Å	Uniform (1 Å)
ANTON [74]	2/3*	r^2	256 (total)	Variable
GROMACS [45]	2	r^2	500 (2000)** nm	Uniform (1nm)

* Anton used cubic polynomial to interpolate energy/force. However, no further details were specified about the interpolation order of force evaluation.

** 500 per nm for single-point floating and 2000 per nm for double precision floating.

4.3 Performance Comparison of Design Alternatives

We obtain the potential performance by multiplying the number of results obtained per cycle by the operating frequency. If the same operating frequency can be retained, fewer resource utilizations consumed by a single force pipeline represents higher performance. Hence, in the rest of this section, the space of possible solutions with respect to resource requirements will be explored. All resource utilizations are post Place and Route (P&R) with respect to the Altera Stratix III EP3SE260 FPGA.

Numerical Precision

Most of conventional MD software packages perform force evaluations in double-precision floating point arithmetic, although highly optimized MD packages, e.g., GROMACS and Desmond, use single-precision or even fixed-point to gain performance [14, 108, 115]. In contrast, MD accelerators, including those using GPUs, ASICs, and FPGAs, are usually implemented in fixed-point or single-precision. This is because of resource constraints and performance considerations [11, 46, 109, 117]. The main reason of using double precision is to avoid cumulative rounding errors in long simulations and minimize the impact of non-associativity in floating point arithmetic. As long as the divergence from the “exact” atomic trajectories is not the main concern, a system that uses single-precision arithmetic would still explore realistic configurations during the simulations [95].

As described earlier, a large dynamic range in the numerical format is required when evaluating the nonbonded short-range force kernel, especially the Lennard-Jones force. But compared with fixed-point, floating point arithmetic usually consumes substantially more hardware resources and thus reduces the overall system performance. In addition, double precision consumes 2-3x more resources than single precision. But if they are done carefully, some calculations can be still be performed in fixed-point arithmetic, especially when large dynamic numerical range is not required, e.g., the distance calculation.

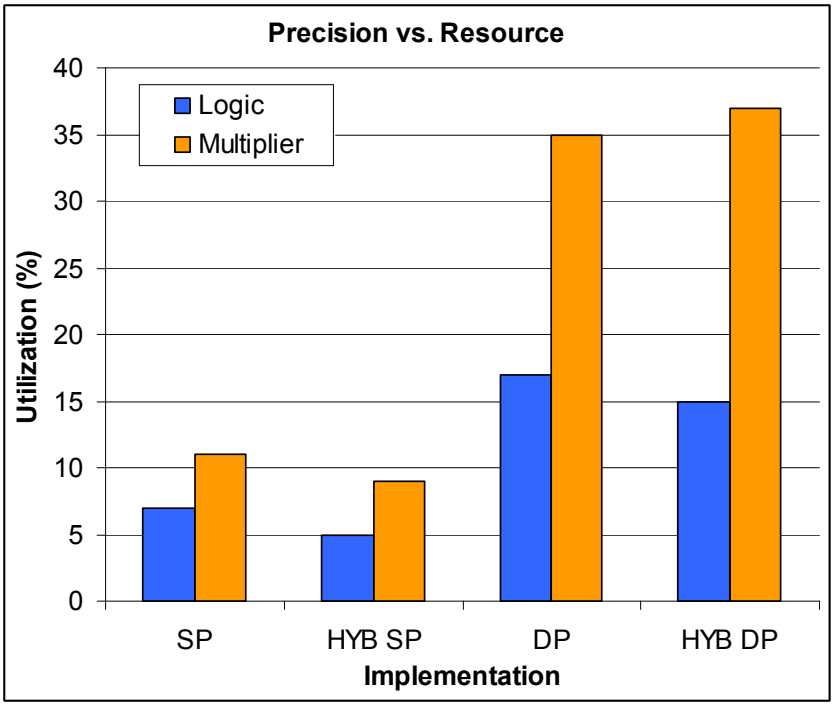

Figure 4-6: Resource utilization of various precision implementations for Stratix III.

Hence, a hybrid (HYB) method was adopted in our design. It uses 32-bit fixed point for displacement and 32/36-bit fixed point for force accumulation. LJ and Coulomb forces are evaluated in single-precision arithmetic. Figure 4-6 shows the performance comparison between direct computation (DC) and hybrid DC where mixed precision arithmetic is performed. Hybrid DC performs well in logic utilization. SP and DP refer to single-precision and double-precision floating point respectively. The reduction mainly comes from the fact that fixed-point arithmetic is employed in the displacement calculation and the force distribution (as shown in

Figure 4-2). We observe that the while Stratix-III FPGAs have substantial floating point support, this does not result in direct scaling from single to double precision. The increase in resources required is 2.5× - 3× for logic, but 4× - 4.5× for the multipliers. In addition, the operating frequency is reduced, but the quality improves.

Another finding is that more multipliers are required for the double precision case since Hybrid DC performs 64 x 64 bit multiplication for the distance squared calculation, which requires more hard-multiplier units, while DC performs 52 x 52 bit multiplication. In the HYB implementation, the square function can be implemented with either logic elements or multipliers, depending on the availability of FPGA resource. It provides designers flexibility for the pipeline implementation.

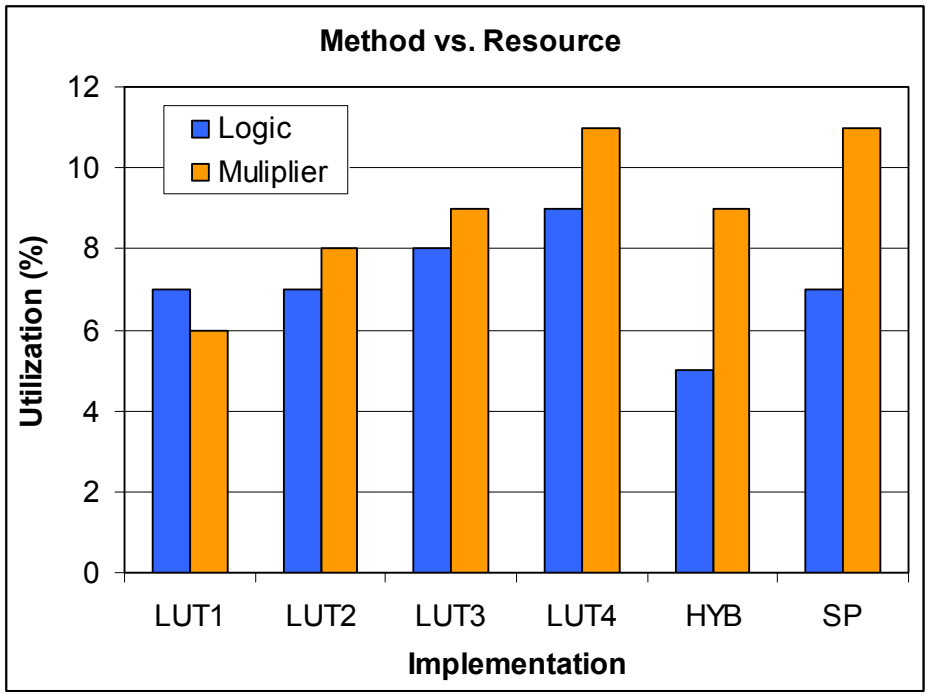

Figure 4-7: Resource utilization in logic and hard multipliers for Altera Stratix III (single pipeline, hybrid-single precision for LUT implementation).

Arithmetic Mode

Figure 4-7 shows the resource usage of various implementations but this time emphasizing single precision and the variation in interpolation order in the LookUp Table method (LUT). Hybrid direct computation uses less than 10% of the hard-multiplier units and far less of the remaining logic. The 3rd order LookUp uses a similar fraction of hard-multiplier units, but substantially more logic. Reducing the interpolation order to 2nd and 1st allows the implementation of perhaps another pipeline or two, but may result in a decrease in simulation

quality. The simulation quality of LUT interpolation order will be addressed in the next section.

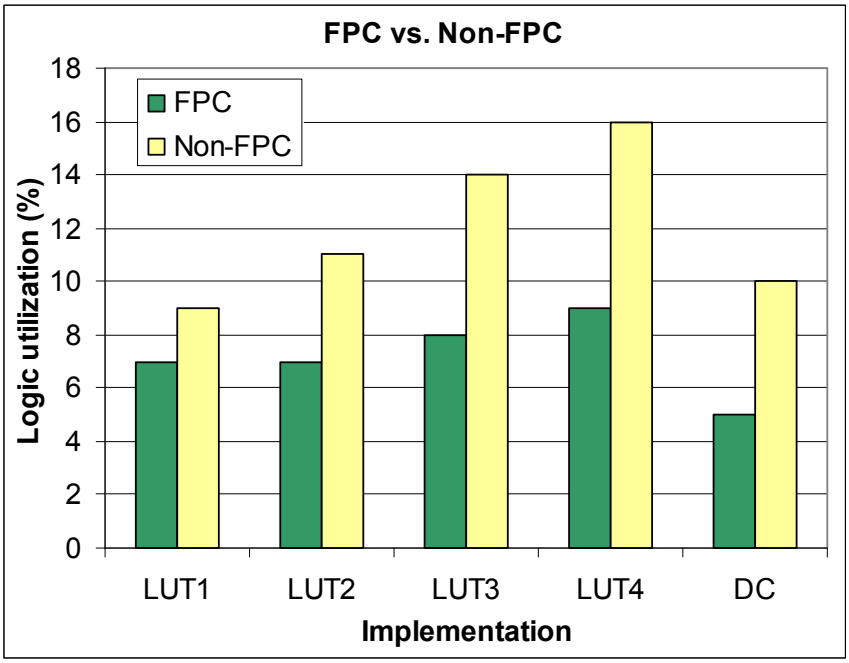

Figure 4-8: Effect of using the Altera FPC on logic utilization for Altera Stratix III (single pipeline, hybrid-single precision).

Component Implementation

The effect of using the Altera Floating Point Compiler is shown in Figure 4-8. Several LUT Implementations of various interpolation orders are presented. Force computations are performed in hybrid/single-precision arithmetic. The low-order LUTs do not take advantage of most of the compiler optimizations because the numbers of floating point operations are limited, but still result in a substantial reduction in logic resource. This is especially helpful for saving logic for the filter pipelines. The FPC does not help result in the reduction of hard-multiplier units.

Extension and Flexibility (equation complexity)

We have described the general methods involved in computing the nonbonded pairwise force and analyzed the potential performance for various design choices. Here we describe some issues in their actual implementations.

While the van der Waals term shown in Equation 4-1 converges quickly, it must still be modified for high-quality and effective MD simulations. In particular, a switching function is implemented to truncate the van der Waals force smoothly at the cutoff distance (see Equations 4-8, 4-9, and 4-10).

$$s = (cutoff^2 - r^2)^2 * (cutoff^2 + 2 * r^2 - 3 * switch\_dist^2) * denom \quad (4\text{-}8)$$

$$ds_r = 12 * (cutoff^2 - r^2) * (switch\_dist^2 - r^2) * denom \quad (4\text{-}9)$$

$$denom = \frac{1}{(cutoff^2 - switch\_dist^2)^3} \quad (4\text{-}10)$$

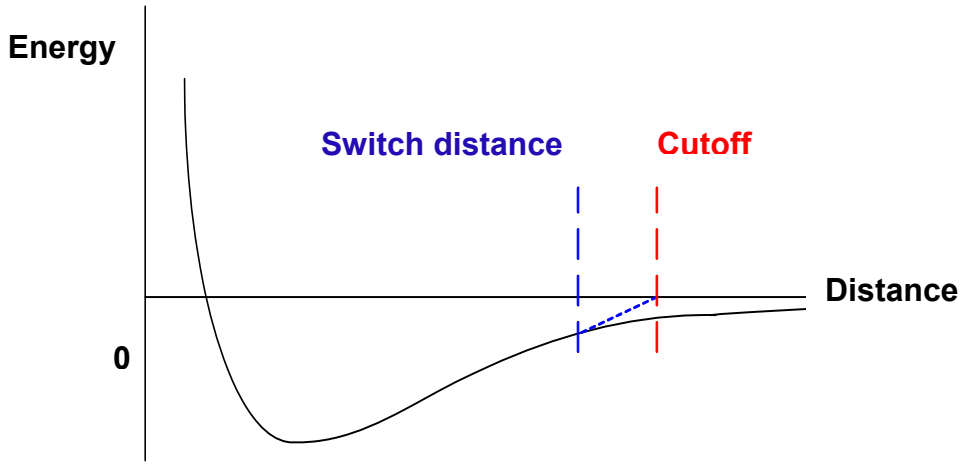

Figure 4-9: Van der Waals potential with switching smooth function

Without a switching function, the energy may not be conserved, as the force would be truncated abruptly at the cutoff distance and energy fluctuation could be large. The graph showing the van der Waals potential with the switching function is illustrated in

Figure 4-9. The van der Waals force and energy can be computed directly as shown here:

1. If $(r^2 \le switch\_dist^2)$ $U_{vdw} = U, F_{vdw} = F$

2. If $(r^2 > switch\_dist^2 \;\&\&\; r^2 < cutoff^2)$

   $U_{vdw} = U * s, F_{vdw} = F * s + U_{vdw} * ds_r$

3. If $(r^2 \ge cutoff^2)$ $U_{vdw} = 0, F_{vdw} = 0$

The switching function can be either computed directly or embedded in pre-computed interpolation coefficients.

We now examine the Coulomb term. The common algorithm supported in most MD packages of calculating the electrostatic force/energy is the Ewald method or its variants. Particle Mesh Ewald (PME) is widely used to evaluate the standard Ewald Sums due to its computational efficiency. It approximates the long-range part of the Ewald Sums by a discrete convolution on an interpolation grid; this can be performed using a discrete 3D Fast Fourier Transform (FFT). The pairwise component is

$$E_s = \sum_{i=1}^{N-1} \sum_{j>i}^{N} \frac{q_i q_j erfc(\beta r_{ij})}{r_{ij}} \qquad (4\text{-}11)$$

where r_{ij} is the distance between particle i and j. erfc(x) is the complementary error function erfc(x) = 1 – erf(x), and β is the Ewald parameter.

Since the *Es* computation contains the evaluation of the complementary error function (erfc), which is expensive in FPGA logic, we use polynomial interpolation rather than direct computation. The polynomial coefficients were pre-computed using Matlab by finding the coefficients of a polynomial p(x) of degree n that fits the data, p(x(i)) to y(i), in terms of least squares.

The problem addressed here is optimizing the computation of the pairwise nonbonded force in light of this added equation complexity. In most MD packages, simulations can be configured to meet user requirements and preference, e.g., using different switching or smooth functions, or various long-range algorithms (simple cutoff, PME or multigrid). Some functions can be evaluated directly in the FPGA; however, some, such as the erfc function, cannot. Moreover, if there were a conditional branch, e.g., the LJ switching function, extra resources would be required when evaluating equations directly. With the LUP method, the switching function can be embedded in the interpolation coefficients such that no additional resources are required.

In summary, if the number of numerical operations involved in the direct computation method is more than that in the LUT method, or if the equation cannot be evaluated directly (with reasonable costs), then the LUT scheme would be favorable. If done carefully, high simulation quality is still obtained.

4.4 Quality Comparison of Design Alternatives

Since MD is chaotic, simulation quality must be validated. While statistical error is likely to be negligible [70], systematic error can be introduced, e.g., as the motion integration algorithm generally assumes the force is continuous and differentiable [112]. True validation of simulation quality, such as through wet lab experiments, is rare: the shortest observable timescales are on the order of microseconds, but this is rarely achieved in simulation for biologically significant molecules. Another consideration is that quality required differs by application. For example, when simulation is an active process with frequent user intervention, then simplified potentials are sometimes used [70]. Quality measures can be classified as follows (see, e.g., 89, 24, 107).

1. Arithmetic error in the approximation is the deviation from the ideal (direct) computation done at high precision (double precision). A frequently used measure is the relative RMS force error, which is defined as follows:

$$\Delta E = \sqrt{\left(\frac{\sum_{i}\sum_{\alpha\in x,y,z}\left|F_{i,\alpha} - F^{*}_{i,\alpha}\right|^{2}}{\sum_{i}\sum_{\alpha\in x,y,z}\left(F^{*}_{i,\alpha}\right)^{2}}\right)} \qquad (4\text{-}12)$$

   While this can be computed precisely, it may hide effects of discontinuities in piecewise approximations [107].

2. Physical invariants should remain so in simulation. Energy can be monitored through fluctuation (e.g., in the relatively RMS value) and drift. A highly sensitive method used the shadow Hamiltonian [24]. We use the following metrics to measure the stability of MD simulation (suggested by Shan et al. [107]):

$$\Delta E = \frac{1}{N_t} \sum_{i=1}^{N_t} \left| \frac{E_0 - E_i}{E_0} \right| \qquad (4\text{-}13)$$

where E_0 is the initial value, Ni is the total number of time steps in time t, and Ei is the total energy at step (i). Acceptable numerical accuracy is achieved when $\Delta E \leq 0.003$.

As just described, direct evaluations of MD simulation quality, such as through validation with wet lab experiments, are often impractical. Thus, surrogates are often used. One type measures the errors with respect to a reference computation. Another type monitors the simulation output to confirm that a physical invariant, such as the total energy, actually is so.

The results presented below (for items 2 and 3) are for the NAMD benchmark NAMD2.6 on ApoA1. It has 92,224 particles, a bounding box of 108 Å × 108 Å × 78 Å with periodic boundary conditions, and a cut-off radius of 12 Å. The Coulomb force is evaluated with PME. The switching function is applied to smooth the LJ force when the intra-distance of particle pairs is between 10 and 12 Å. NAMD-Lite was modified to support the quality measurements.

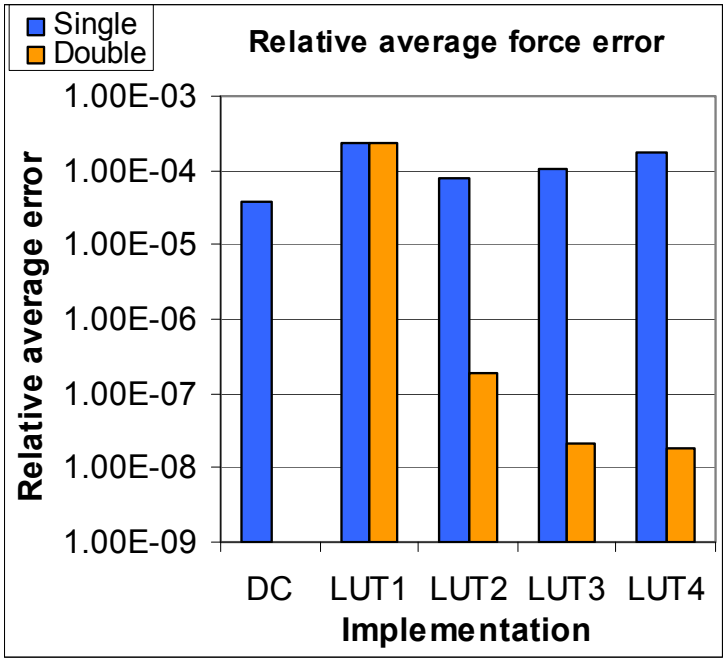

Figure 4-10: Relative average force error of the particle-particle force for various implementation and precision. DC is direct computation, LUTn refer to Look-UP of various orders.

1. **Error per individual particle-particle force computation**

Figure 4-10 shows the relative average error of pairwise nonbonded force computations for various pipeline implementations. The reference is direct computation using double precision (DC Double, error = 0). We generate the particle pairs by randomly selecting particle positions between the cut-off and exclusion radii. For the single precision LUT method, error becomes worse for higher orders. This is because of the higher precision required for those tables.

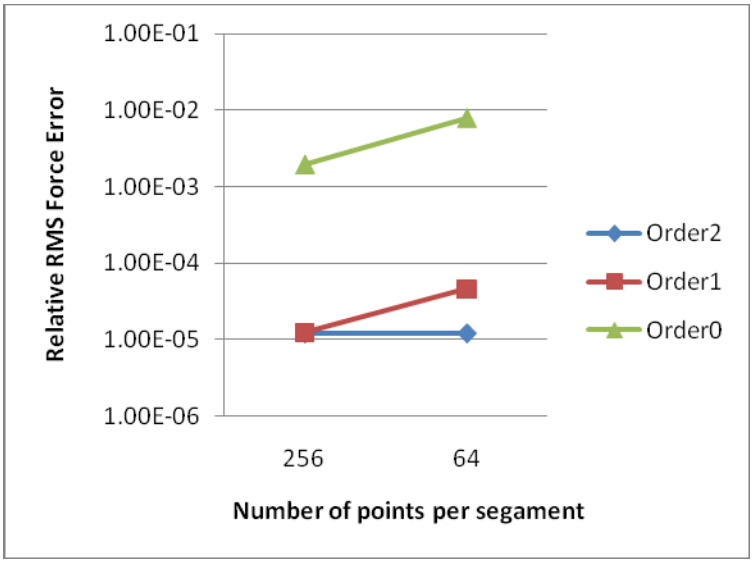

Figure 4-11: Graphs of relative RMS force error versus interpolation density per section for interpolation orders 0, 1, and 2.

2. Error per total force on a particle per iteration

The simulation was first run for 1000 timesteps using direct computation. Then in the next timestep, both direct computation and table interpolation were used to find the relative RMS force error for table interpolation. Two temporary arrays were used to save the computed forces for the two methods. Only the range-limited forces (switched vdw and short-range portion of PME) were considered. All computations were done in double precision; Equation 4-12 was used to compute the relative RMS. Results are shown in Figure 4-11. The reference is direct computation using double precision. All exceed the quality criteria given in Shan, et al. [107]. We note that 1st and 2nd order interpolation have two orders of magnitude less error than 0th order. We also note that with 256 intervals per segment

(and 12 segments) 1st and 2nd order are virtually identical. Users can choose the implementation method (1st order with higher interpolation density or 2nd order with lower interpolation density), depending on the resource availability (BRAMs versus logic and hard-multipliers) of FPGA devices.

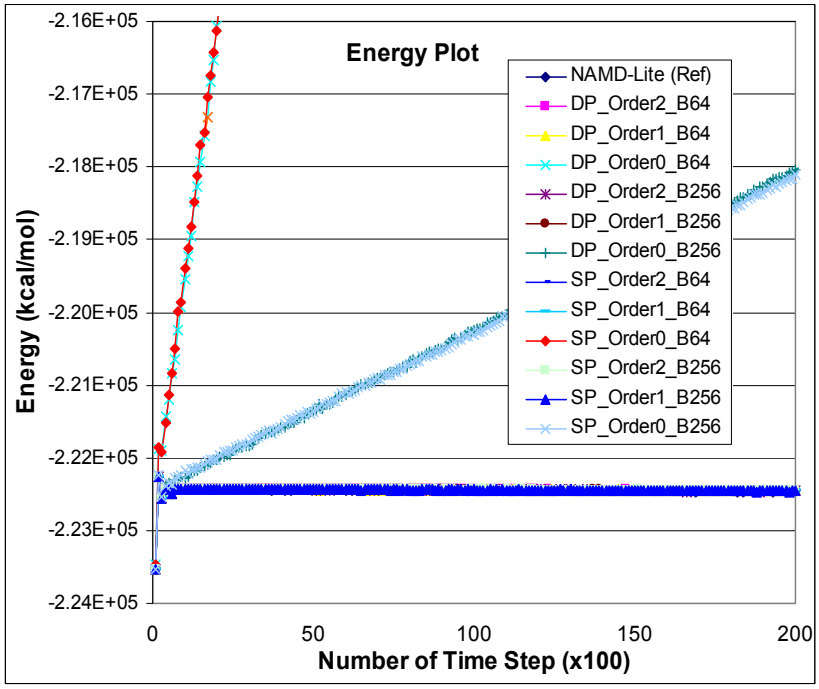

Figure 4-12: Graphs of energy for various designs run for 20,000 timesteps.

3. Energy conservation

The previous analyses examined the model accuracy statically. In order to evaluate the dynamic impact of various designs on MD simuation, we must analyze the simulation stablity of the MD system, which can be measured in term of energy conservation and fluctuation [107]. Results

with respect to energy fluctuation and drift are shown in Figure 4-12. A number of design alternatives were examined, including the original code and all combinations of the following parameters: interval density (64 and 256 per segment), interpolation order (0th, 1st, and 2nd), and single and double precision floating point. We note that all of the 0th order simulations are unacceptable, but that the others are all indistinguishable (in both energy fluctuation and drift) from the serial reference code running direct computation in double precision floating point.

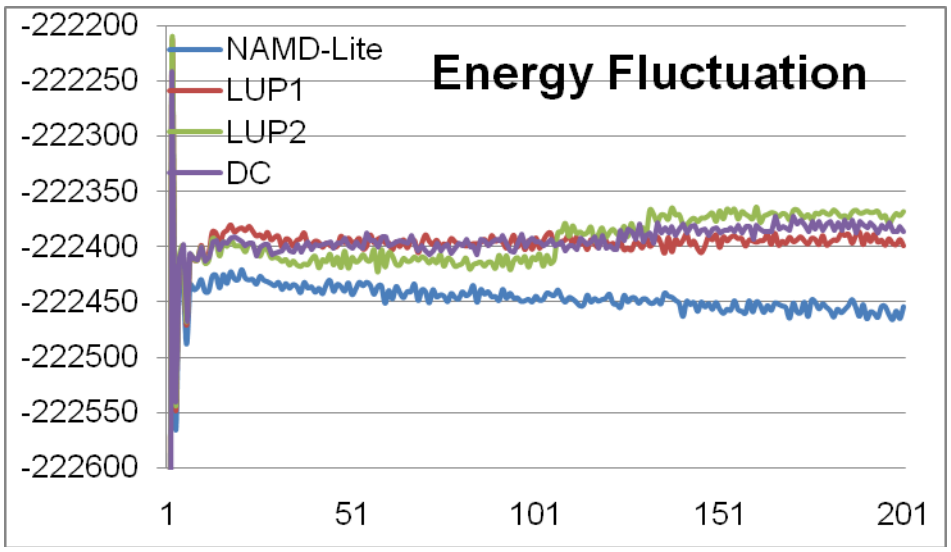

Figure 4-13: Graphs of energy for selected designs run for 100,000 timesteps

Three implementations were chosen for longer simulations (shown in Figure 4-13). Using Equation 4-13 to compute ΔE, we find that the value for the reference code is 1.1E-4 and for both of the FPGA-accelerated

codes is 1.3E-4; all are much smaller than 0.003. After 70,000 timesteps, the values for ΔE are all less than 1.5E-7.

4.5 Summary

FPGAs lend themselves to a particular rich design space of both opportunities and constraints. We have explored and evaluated this space with respect to both resource requirements and simulation quality, including numerical precision, arithmetic algorithm, datapath optimization, equation complexity and flexibility. For each design axis, general guidelines were given for implementing highly competitive MD accelerators. We find that FPGAs' BRAM architecture makes them well suited to support unusually fine-grained intervals. This leads to a reduction in other logic and a proportional increase in performance. Potential performance and simulation quality for various designs have also been examined and can serve as a guideline for FPGA MD implementations.

Although the Altera FPC helps save logic resources and simplifies the design process, special attention must be paid to how to formulate the target function so that the maximum numerical precision can be preserved and the resulting design optimized in terms of both latency and resource utilization.

In general, more accuracy requires more hardware. Optimization of performance versus quality, however, is not trivial. Accuracy only affects simulation quality indirectly and highest quality simulations may not always be needed. Direct computation (with sufficient precision) usually offers higher

simulation quality although its cost may high, especially when evaluating complex force equations or when there is a conditional branch. LUT-based methods provide flexibility and maintain a fixed hardware cost for various force equations; they are favored when the reduction in accuracy is acceptable. A mix of both schemes can also be adopted.

Chapter 5 Filter Pipeline Design and Optimization

5.1 Overview

While MD generally involves all-to-all force evaluations among particles, a cut-off is commonly applied to limit the extent of the short-range force to a fraction of the simulation space. Two methods are commonly used to take advantage of this

cutoff: cell lists and neighbor lists (as shown in Figure 5-1). With cell lists, the simulation space is typically partitioned into cubes with edge-length equal to the cutoff radius, r_c. Non-zero forces on the reference particle P can then only be applied by other particles in its home cell and in the 26 adjacent cells (the 3 x 3 x 3 cell neighborhood). We refer the second particle of the pair as the partner particle. With neighbor lists, P has associated with it a list of exactly those partner particles within r_c.

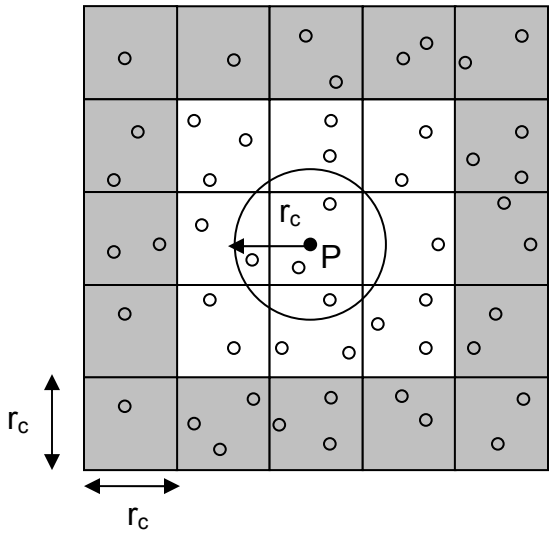

Figure 5-1: P's two dimensional cell neighborhood is shown in white; cells have edge size equal to the cut-off radius. Particles within the P's cut-off circle are in P's neighbor list [16].

We now compare these methods.

- **Efficiency:** Neighbor lists are by construction 100% efficient: only those particle pairs with non-zero mutual forces are evaluated. Cell lists as just defined are 15.5% efficient with that number being the ratio of the volumes of the cutoff sphere and the 27-cell neighborhood.

- **Storage:** With cell lists, each particle is stored in a single cell's list. With neighbor lists, each particle is typically stored in 400-1000 neighbor lists, depending on particle density and cutoff radius.
- **List creation complexity:** Computing the contents of each cell requires one pass through the particle array. Computing the contents of each neighbor list requires, naively, that each particle be examined with respect to every other particle: the distance between them is then computed and thresholded. In practice, however, it makes sense to first compute cell lists anyway. Then the neighbor lists can be computed using only the particles in each reference particle's cell neighborhood such that the work of constructing neighboring list can be reduced [139].

From the last point, it appears that the creation of neighbor lists involves not only cell lists, but also a fraction of the force computation itself. At this point, why not finish computing the forces of those particles that are within the cutoff radius? Why save the neighbor list?

Most MD codes reuse the neighbor lists for multiple iterations and so amortize the work in their creation. However, because particles move during each iterative step, particles can enter and exit the cutoff region leading to potential error. The solution is to add some margin into the neighborlist cutoff such that it is larger than the force cutoff, e.g., 13.5Å neighborlist cutoff versus 12Å force cutoff (see Figure 5-2). There is a trade-off between the increase in neighborhood size (and

thus the number of particle pairs evaluated) and the number of iterations between neighbor list updates.

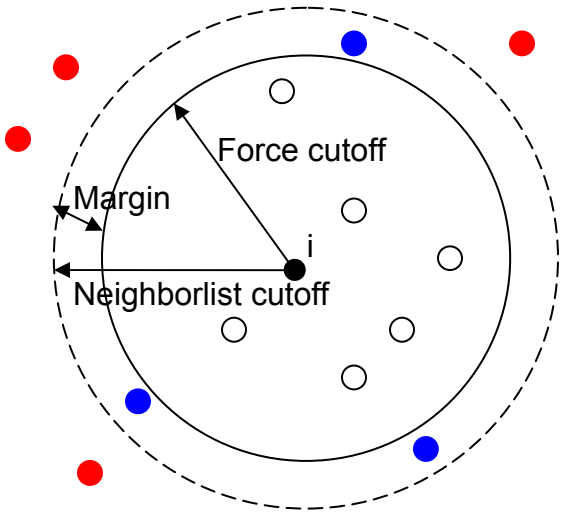

Figure 5-2: Neighborlists are often computed for a larger radius than the force cutoff.

Although neighbor lists have been proven to be efficient when particles move slowly and their construction time can be reduced with the aid of cell-list scheme [139], the shortingcomings of large storage demand and non-sequential data transfer between processors and memory still remains and becomes significant for large systems [32].

The conventional approach of evaluating pairwise interactions with the cell list method usually leads to a substantial number of superfluous computations. Several studies have made improvements on the algorithm and investigated their efficiency [3, 32, 42, 79, 131]. W. Mattson and B. M. Rice overcame this

shortcoming by further partitioning the simulation domain into cubes whose edge-length is smaller than cut-off radius [3, 133]. Another approach, proposed by P. Gonnet, reduces the number of unnecessary particle-pair calculations by first sorting the particles along the cell axis and then only evaluating particles whose inter-distance along the axis is smaller than the cutoff radius [42]. It was shown that 59.4% of the particle pairs whose inter-distance is computed and tested are actually within cutoff radius. This is almost four times better than the 16% in the conventional cell list method and more than double the 27% from using cells with edge-length equal to 0.5 * r_c [32, 42]. U. Welling and G. Guido modified Gonnet's variant of the cell list algorithm by using a reordered linked cell, rather than a plain linked cell, and adding optimal sorting to better fit a broad range of simulation setups [131].

5.2 Coprocessor Considerations

With MD coprocessing there are additional considerations. The cell list computation is very fast and the data generated small (both *O(N)*) so it is generally done on the host (along with the motion integration): the cell lists are downloaded to the coprocessor every iteration along with the new particle positions. The neighbor list computation, however, is much more expensive: if done on the host it could mitigate any advantage of coprocessing. Moreover, the size of the aggregate neighbor lists is hundreds of times that of the cell lists, which makes their transfer impractical. As a consequence, neighbor list

computation, if it is done at all, must be done on the coprocessor. But even on the coprocessor storage is still a concern.

We look first at MD with cell lists. For reference and without loss of generality we examine the NAMD benchmark NAMD2.6 on ApoA1. It has 92,224 particles, a bounding box of 108 Å × 108 Å × 78 Å, and a cutoff radius of 12 Å. This yields a simulation space of 9 × 9 × 7 cells with an average of 175 particles per cell with a uniform distribution. On the FPGA coprocessor, the working set is typically a single (home) cell and its cell neighborhood for a total of (naively) 27 cells and about 4,725 particles.

In actuality, Newton's 3rd Law (N3L) is used to reduce this number. That is, since each particle-particle interaction is mutual, it can be calculated once per particle pair and recorded for both particles. To effect the reduction in work, home cell particles are only matched with particles of a fraction of the cell neighborhood, and with, on average, half of the particles in the home cell. We refer to the subset of cells in the cell neighborhood that are processed together with (and including) the home cell as the cell set. For the 14- and 18-cell sets presented below in Chapter 5.5, the average number of particles to be examined (for each particle in the home cell) is 2,450 and 3,150, respectively. Given current FPGA technology, any of these cell sets (14, 18, or the original 27 cells) easily fits in the on-chip BRAMs.

Neighbor lists for a home cell do not fit on the FPGA. For example, the aggregate neighbor lists for 175 home cell particles is over 64,000 particles (one

half of 732 for each of the 175 particles; 732 rather than 4,725 because of increased efficiency of neighbor lists over cell lists).

The memory requirements are therefore very different for the two methods. For cell lists, we swap cells onto and off of the FPGA as needed. Because of the high level of reuse, this is easily done in the background. In contrast, neighbor list particles must be streamed from off-chip. This has worked when there are one or two force pipelines operating at 100MHz [69, 106], but is problematic for current HPRC systems. For example, the Stratix-III/Virtex-5 generation of FPGAs can support 8 force pipelines operating at 200MHz leading to a bandwidth requirement of over 20 GB/s. While high-end FPGAs support this easily, memory interfaces in commercial systems generally do not.

From this discussion, it follows that use of neighbor lists calls for an “on-FPGA” solution, but also that this itself appears to be impracticable due to memory and transfer requirements. At the same time, however, the 6x potential increase in efficiency cannot be abandoned. One way to improve efficiency is to reduce the cell size: the smaller the cell size, the finer the granularity, and the larger the fraction of the cell neighborhood volume guaranteed to be useful. With a cell edge of $r_c/2$ and a 5^3 set, efficiency increases to 26.8%. With more aggressive clipping of the corner cells, efficiency increases a bit more but so does the control complexity. More important is that reducing cell size also reduces reuse and still leaves much inefficiency. While reducing cell size is viable, there are better options.

The solution we propose is to use neighbor lists, but to compute them every iteration, generating them continuously and consuming them almost immediately. In this scenario, the use of neighbor lists can be viewed as *filtering* out the zero-force particle pairs: the filter pipelines feed the force pipelines with minimal buffering in between (see Figure 5-3). Designs and tradeoffs for this solution will be presented in the next several sections.

We now describe the execution flow. Processing is built around the home cell. Position and acceleration data of the particles in the cell set are loaded from board memory into on-chip caches, POS and ACC, respectively. When the processing of a home cell has completed, focus shifts and a neighboring cell becomes the new home cell. Its cell set is now loaded; in our current scheme this is nine new cells.

Acceleration data differs from position data in that it is read/write. That is, each particle's acceleration accumulates over this and other home cells. It is not complete for any given home cell until all 27 cells in its cell neighborhood have also been the home cell. Therefore the nine cells of acceleration data are swapped rather than just overwritten.

One design constraint is that each force pipeline should handle at most a small number of reference particles P_i at a time. This enables the total forces on the P_is to be accumulated in registers. Accumulating the mutual forces on the P_is' N3L partner particles, however, is more complex as their positions span the cell set. To prevent BRAM access contention, the following strategy is used. Partner

updates are written to BRAMs associated uniquely with each force pipeline. When processing of a home cell is completed, the partner data from the various pipeline-specific BRAMs are merged. This operation is performed during swapping out, so latency is completely hidden.

The time to process a home cell T_{proc} is generally greater than the time T_{trans} to swap cell sets with off-chip memory. Assume that a cell has edge length = r_c and contains on average N_{cell} particles. Then $T_{trans} = 324 \times (N_{cell}/B)$ (9 cells, 32-bit data, 3 dimensions, 2 reads and 1 write, and transfer bandwidth of B bytes per cycle). To compute T_{proc}, assume P pipelines and perfect efficiency. Then T_{proc} is $\sim (N_{cell})^2 \times (\pi/2P)$ cycles. This gives the following bandwidth requirement: $B > 206 \times P/N_{cell}$. For $P = 10$ and $N_{cell} = 175$, $B > 12$ bytes per cycle. For many current FPGA processor boards, B is usually larger than 16. Some factors that increase the bandwidth requirement are faster processor speeds, more pipelines, and lower particle density. A factor that reduces the bandwidth requirement is better cell reuse.

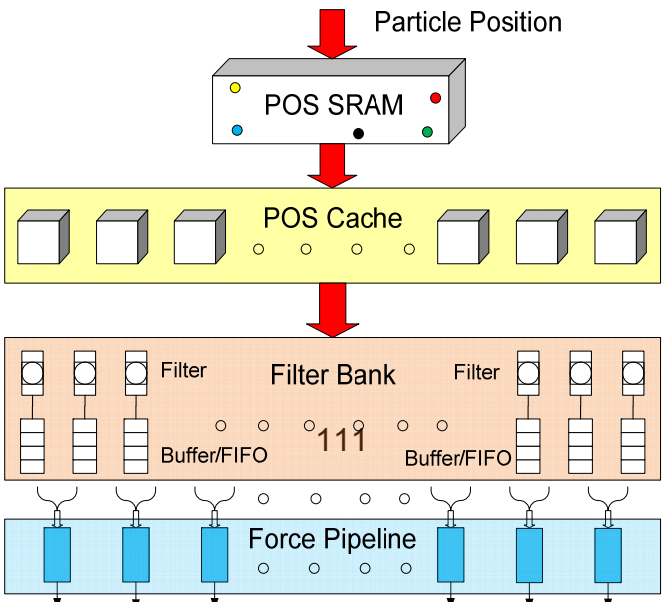

Figure 5-3: Schematic of the HPRC MD system

5.3 Filtering Algorithms

We begin by assuming cell lists with processing concentrating on one home cell at a time. With no filtering or other optimization, forces are computed between all pairs of particles i and j, where i must be in the home cell but j can be in any of the 27 cells of the cell neighborhood, including the home cell. By filtering we mean the identification of particle pairs where the mutual short-range force is zero. A perfect filter successfully removes all such pairs. The efficiency of the filter is the fraction of undesirable particle pairs removed. The extra work due to imperfection is the ratio of undesirable pairs not removed to the desirable pairs.

We evaluate three methods, two existing and one new, which trade off efficiency for hardware resources. As motivated in 0, we store particle positions in three Cartesian dimensions, each in 32-bit integer. There are two parameters, precision and geometry.

1. **Full Precision: Precision = full, Geometry = sphere**

   Computes $r^2 = x^2 + y^2 + z^2$ and compares whether $r^2 < (r_c)^2$ using full 32-bit precision. Filtering quality in this case is nearly 100%. Except for the comparison operation, this is the same computation that is performed in the force pipeline.

2. **Reduced: Precision = reduced, Geometry = sphere**

   This filter, used by D.E. Shaw [74], also computes $r^2 = x^2 + y^2 + z^2$, $r^2 < (r_c)^2$, but uses fewer bits and so substantially reduces the hardware required. Lower precision, however, means that the cut-off radius must be increased (rounded up to the next bit) so filtering efficiency goes down: for 8 bits of precision, it is 99.5%. In our reference example, each particle is now matched with about 378 particles, rather than the 366 for perfect filtering, for about 3% extra work.

3. **Planar: Precision = reduced, Geometry = planes**

   A disadvantage of the previous method is its use of multipliers, which are the critical resource in the force pipeline. This issue can be important because there are likely to be 6 to 10 filter pipelines per force pipeline. In

this method we avoid multiplication by thresholding with planes rather than a sphere (see Figure 5-4 for the two-dimensional (2D) analog). With 8 bits, this method achieves 97.5% efficiency for about 13% extra work. The formulae are as follows:

- $|x| < r_c, |y| < r_c, |z| < r_c$
- $|x| + |y| < \sqrt{2} r_c, |x| + |z| < \sqrt{2} r_c, |y| + |z| < \sqrt{2} r_c$
- $|x| + |y| + |z| < \sqrt{3} r_c$
- $|x| \geq 0$ (with Newton's 3rd law)

Table 5-1 summarizes the resource cost (LUTs, registers, and multipliers) and quality (efficiency and extra work) of the three filtering methods. Since multipliers are a critical resource, we also show the two "sphere" filters implemented entirely with logic. The cost of a force pipeline (from 0) is shown for scale.

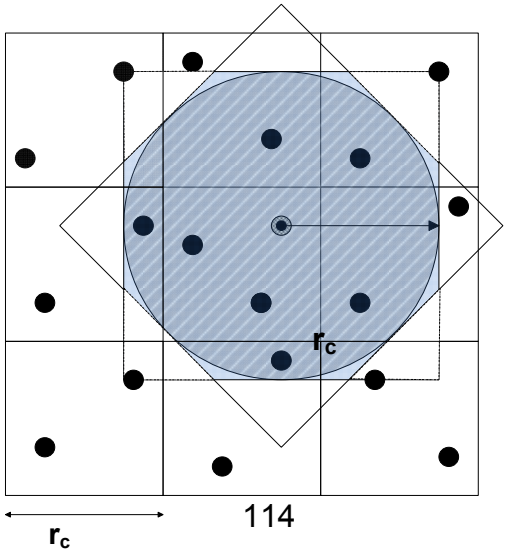

Figure 5-4: Filtering with planes rather than a sphere - 2D analogue

Table 5-1: Comparison of three filtering schemes for quality and resource usage. A force pipeline is shown for reference. Percent utilization is for the Altera Stratix-III EP3SE260.

Filtering Method	LUTs/Registers		Multipliers		Filtering Efficiency	Extra Work
Full Precision	341/881	0.43%	12	1.6%	100%	0%
(logic only)	2577/2696	1.30%	0	0.0%	100%	0%
Reduced	131/266	0.13%	3	0.4%	99.5%	3%
(logic only)	303/436	0.21%	0	0.0%	99.5%	3%
Planar	164/279	0.14%	0	0.0%	97.5%	13%
Force Pipeline	5695/7678	5.00%	70	9.1%	NA	NA

The most important result is the relative cost of the filters to the force pipeline. Depending on implementation, each force pipeline needs between 6 and 10 filters to keep it running at full utilization. We refer to that set of filters as a filter bank. Table 5-1 shows that a full precision filter bank takes from 105% and 160% of the resources of its force pipeline. The reduced (logic only) and planar filter banks, however, require only a small fraction: between 17% and 40% of the logic of the force pipeline and no multipliers at all. Since the latter is the critical resource, the conclusion is that the filtering logic itself (not including interfaces) has negligible effect on the number of force pipelines that can fit on the FPGA.

We now compare the reduced and planar filters. The “Extra Work” column in Table 5-1 shows that for a planar filter bank to obtain the same performance as logic-only-reduced, the overall design must have 13% more throughput. This translates, e.g., to having 9 force pipelines when using planar rather than 8 for reduced. The total number of filters remains constant. The choice of filter therefore depends on the FPGA’s resource mix and force pipeline implementations.

5.4 Balancing Neighboring List Sizes

For efficient control and particle-memory access, and for smooth interaction between filter and force pipelines, it is preferred to have each force pipeline handle the interactions of a single reference particle at a time. This preference becomes critical when there are a large number of force pipelines and a much larger number of filter pipelines. Moreover, it is highly desirable for all of the neighbor lists being created at any one time (by the filter banks) to be transferred to the force pipelines simultaneously (buffering mechanisms are described in Chapter 5.6). It follows that each reference particle should have a similar number of partner particles (neighbor list size).

The problem addressed in this section is that the standard method of choosing a reference particle's partner particles leads to a severe imbalance in neighbor list sizes. How this arises can be seen in Figure 5-5, which illustrates the standard method (half-shell) of optimizing for N3L. So that a force between a particle pair is computed only once, only a "half shell" of the surrounding cells is examined (in 2D, this is cells 1-4 plus Home). For forces between the reference particle and other particles in Home, if the particle identification (ID) were used to break the tie, with, e.g., the force being computed only when the ID of the reference particle is the higher. Particle B (shown in Figure 5-5b) has a much smaller neighborlist than A (shown in Figure 5-5a), especially if B has a low ID and A a high.

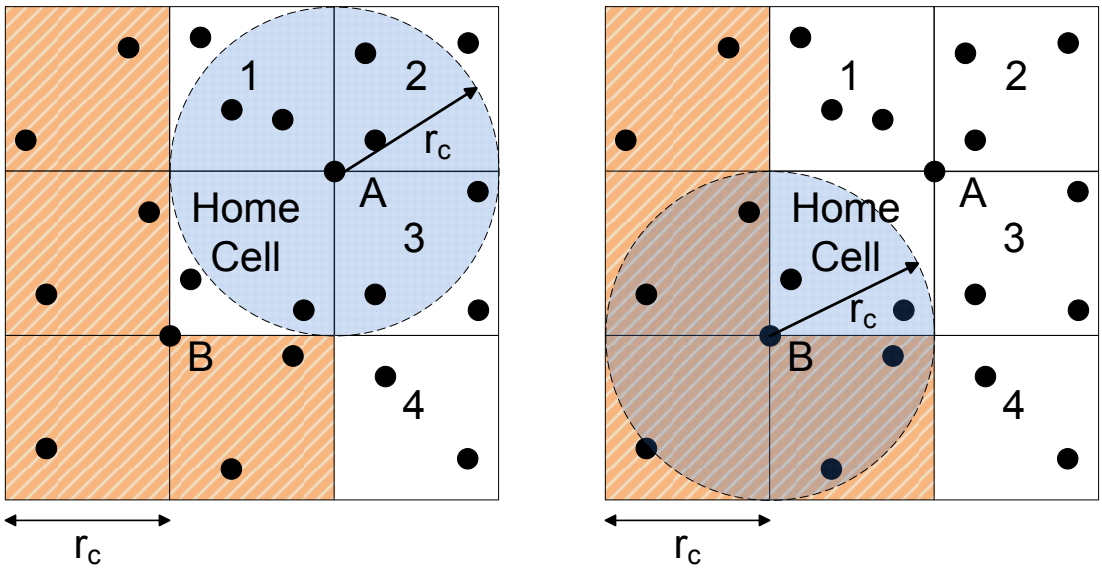

Figure 5-5: Shown is the standard partitioning scheme with Newton's 3rd law. 1-4 plus home cell are examined with a full sphere.

In fact neighborlist sizes vary from 0 to 2L, where L is the average neighborlist size. The significance is as follows. Let all force pipelines wait for the last pipeline to finish before starting work on a new reference particle. Then if that (last) pipeline's reference particle has a neighborlist of size 2L, then the latency will be double that if all neighbor lists were size L. This distribution has high variance (see Figure 5-6) meaning that neighbor list sizes greater than, say, $\frac{3}{2}$L are likely to occur. A similar situation also occurs in other MD implementations, with different architectures calling for different solutions [7, 114].

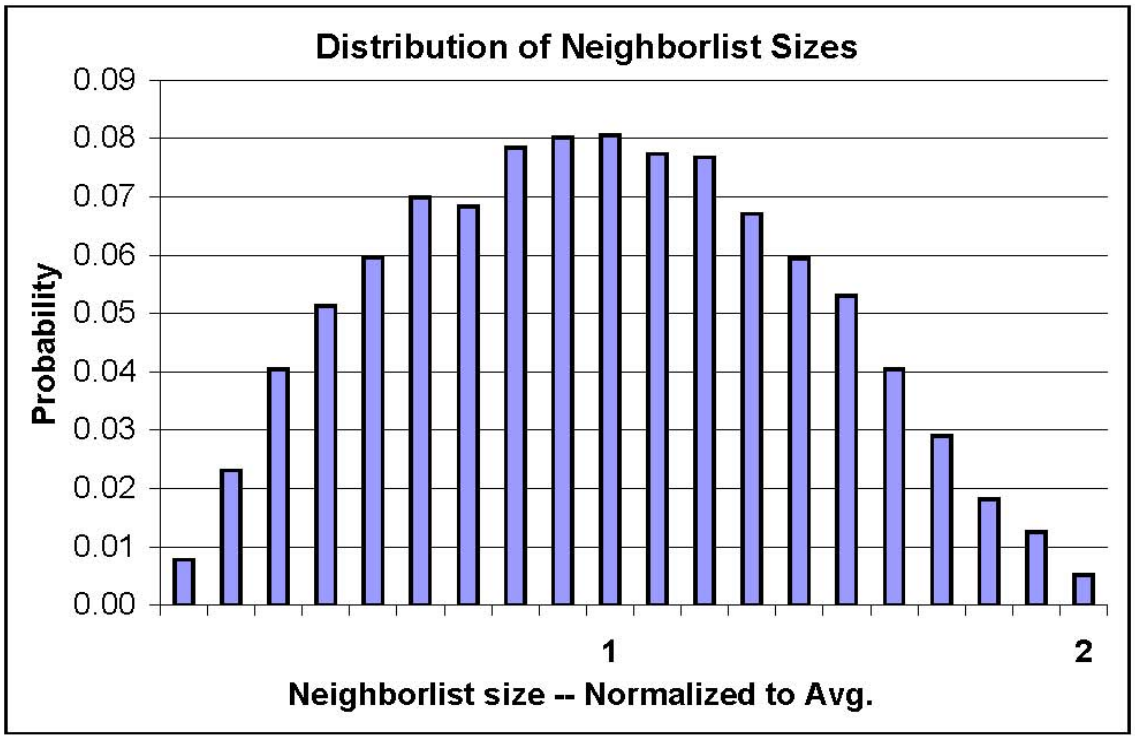

Figure 5-6: Distribution of neighborlist sizes for standard partition as derived from Monte Carlo simulations.

One way to deal with this load imbalance is to overlap the force pipelines so that they work independently. While viable, this leads much more complex control.

An alternative is to change the partitioning scheme. Our new N3L partition is shown in Figure 5-7. There are three new features. The first is that the cell set has been augmented from a half shell to a prism. In 2D this increases the cell set from 5 cells to 6; in 3D the increase is from 14 to 18. The second is that, rather than forming a neighbor list based on a cutoff sphere, a hemisphere is used instead (the “Half-Moon” in Figure 5-7). The third is that there is now no need to compare IDs of home cell particles.

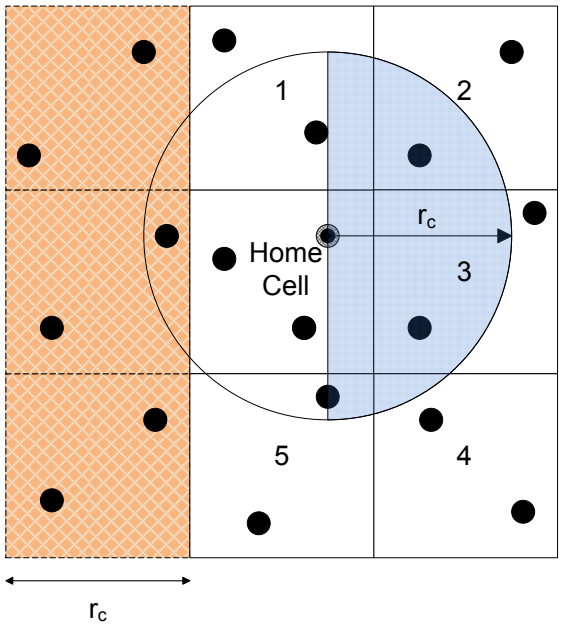

Figure 5-7: Shown is half-moon partitioning scheme for using Newton's 3rd law. 1-5 plus home cell are examined, but with a hemi-sphere (blue-shaded part of circle).

We now compare the two partitioning schemes. There are two metrics: the effect on the load imbalance and the extra resources required to prevent it.

- **Effect of load imbalance:** We assume that all of the force pipelines begin computing forces on their reference particles at the same time, and that each force pipeline waits until the last force pipeline has finished before continuing to the next reference particle. We call the set of neighbor lists that are thus processed simultaneously a cohort. With perfect load balancing, all neighbor lists in a cohort would have nearly the same size, the average. The effect of the variation in neighbor list size is the number of excess cycles—before a new cohort of reference particles can begin processing—over the number of cycles if each neighborlist were the same size. The performance cost is therefore the average number of excess cycles per cohort. This in turn is the average size of the biggest neighbor list in a cohort minus the average neighbor list size. We find that, for the standard N3L method, the average excess is nearly 50%, while for the "half-moon" method it is less than 5%.

- **Extra resources:** The extra work required to achieve load balance is proportional to the extra cells in the partition: 18 versus 14, or an extra

29%. This drops the fraction of neighbor list particles in the cell neighborhood from 15.5% to 11.6%, which in turns increases the number of filters needed to keep the force pipelines fully utilized (over-provisioned) from 7 to 9. For the reduced and planar filters, this is not likely to reduce the number of force pipelines.

5.5 Mapping Particle Pairs to Filter Pipelines

From the previous sections, we converge on an efficient design for filtering particle pairs:

- During execution, the working set (data held on the FPGA) consists of the positions and accelerations of particles in a cell set; i.e., a single home cell and its 17 neighbors (in the “half moon” scheme);
- Particles from each cell are stored in a set of BRAMs: this is currently one or two BRAMs per coordinate, depending on the cell size and particle density, for a total of 108-216;
- The N3L partition specifies 7-9 filters per force pipeline;
- FPGA resources indicate 8-10 force pipelines; and

- Force pipelines handle at most a small number of reference particles at a time (and their N3L partners).

We now address the mapping of particle pairs to filter pipelines. There are a large number of ways to do this; finding the optimal mapping is in some ways analogous to optimizing loop interchanges with respect to a cost function.

Figure 5-8 shows two possibilities. In particle mapping (a), each filter is responsible for a different reference particle. Each cycle, a single partner particle from the cell set is broadcast to all of the filters (in all of the filter banks). In cell mapping (b), each filter bank is collectively responsible for a different reference particle. Each filter within a bank processes the reference particle with respect to partners from its own subset of 2 or 3 cells. The issues are as follows.

- **Force pipeline efficiency:** Overall performance is proportional to the efficiency of the force pipelines, i.e., the fraction of cycles that they deliver "payload" (pairs with non-zero forces). Since there are no stalls, the efficiency is thus proportional to the fraction of cycles that they input (are issued) payload particle pairs from their filter banks.
- **Payload generation rate:** Given sufficient filters, a filter bank will generate payload pairs at an average rate of greater than one per cycle. The variance may be high, however, which can substantially degrade efficiency.

- **Distribution of payload particle pairs:** While the number of payload particle pairs from a given cell set—and even from any reference particle (from Chapter 5.4—has a small variance, the number and distribution of payload pairs generated by any particular filter can vary wildly. For example, in Figure 5-7, let two filters (in a bank) each handle the same reference particle, but let the partner particles be from different cells, say 3 and 5. Each filter examines the same number of pairs, but the first filter passes most of its input while the second passes almost none.

- **Queuing particle pairs:** A simple (but costly) solution is to: (i) append a large queue to each filter and (ii) implement a flexible router from these queues to the force pipeline. The two mappings lend themselves to multiple more practical queuing methods, the choice of which depends on the resources available on the FPGA.

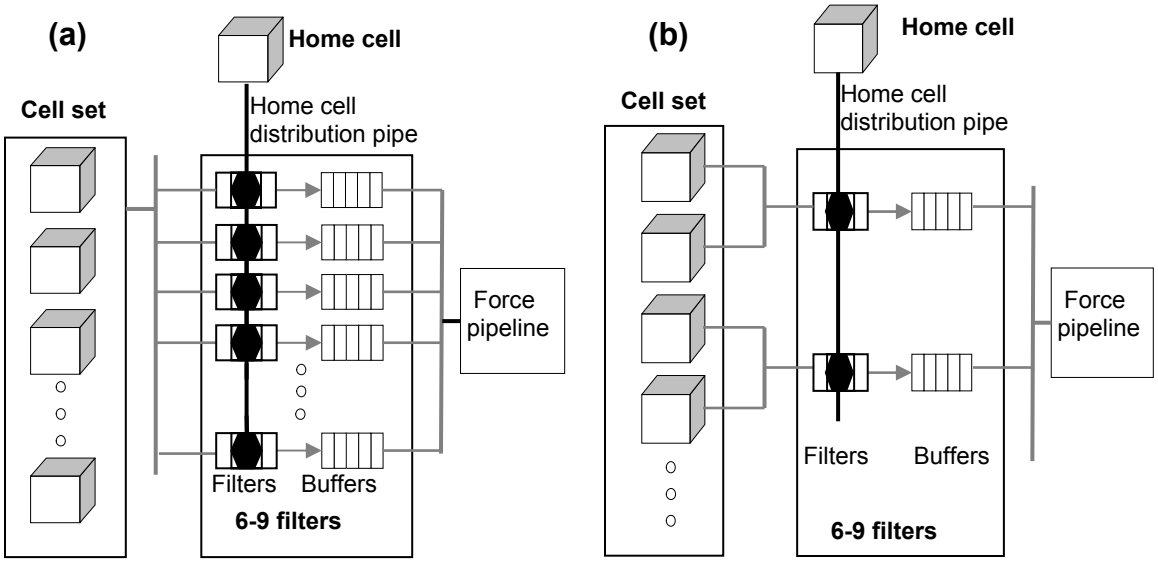

Figure 5-8: Two mappings of particle pairs onto filters. (a) Particle Mapping: Filters each hold a different reference particle. Particles in cell set are broadcast one per cycle. (b) Cell Mapping: Same reference particle held by all filters in a bank. Each filter is responsible for 2-3 cells.

5.6 Queueing and Routing Particle Pairs

In this section we present two queuing strategies, whole neighbor list and continuous. We evaluate them with respect to the two particle mapping strategies for performance (force pipeline efficiency) and hardware cost (queue size and complexity).

5.6.1 Queuing Whole Neighboring List

If there were sufficient BRAMs, then particle mapping can be used to generate neighbor lists in their entirety and consumed in the same way. Details are as follows; we assume particle mapping, but the logic is similar for cell mapping.

- A phase begins with a new and distinct reference particle being associated with each filter.
- Then, on each cycle, a single particle from the 18-cell set is broadcast to all of the filters.

- Each filter's output goes to its own set of BRAMs.
- The output of each filter is exactly the neighborlist for its associated reference particle.
- Double buffering enables neighborlists to be generated by the filters at the same time that the previous phase's neighborlists are being drained by the force pipelines.

Advantages of this method include:

- Nearly perfect load balance among the filters (from the "half-moon" partition);
- Little overhead: each phase consists of over 3000 cycles before a new set of reference particles must be loaded;
- Nearly perfect load balancing among the force pipelines: each operates successively on a single reference particle and its neighborlist; and
- Simple queuing and control: neighborlist generation is decoupled from force computation.

A disadvantage is that this queuing method requires hundreds of BRAMs. Although there are a thousand or more on some high-end FPGAs, this is still a concern.

5.6.2 Continuous Queuing

Figure 5-8 shows the basic queuing used in both mappings: Some number of filters F in a filter bank feed a single force pipeline. As described in 0, the force pipelines should be as independent as possible. This is to constrain the complexity of the routing between filter and force stages and between force stage and accelerator cache.

At a high-level, this is a typical queuing problem with F servers where each has known arrival and departure rates. An arrival is the generation of a particle pair that has passed the filter criteria; a departure is when a payload pair is consumed by the force pipeline. Also, the goal is to minimize idle time (when all queues are empty) and hardware cost. The latter includes queue size, but also complexity of the control and of the concentrator logic that routes from the filter queues to the force pipeline.

There are also a number of differences, however. These restrict the utility of stochastic analysis, but also point to implementation methods.

1. Execution proceeds in phases: For particle mapping, the filter bank processes F reference particles in a phase. For cell mapping, it processes only one.

2. Uniformity: The total number of arrivals per reference particle varies only slightly within a phase (for particle mapping) and among phases (for both mappings).

3. Non-uniformities. The F queues can have highly non-uniform departures and/or high variation in departures during a phase. Depending on the position of the reference particle in the home cell and on the cell of its partner, the a priori probability of a departure can be anything from 0 to 1.

Some design considerations are as follows. To minimize queue size, there are several mechanisms including under provisioning (by keeping F small) and throttling (when queues are full). Even if these are used, however, performance is improved by smoothing and balancing the departure rates (arrivals at the force pipelines). Here are three ways that help do this.

- **Fetch order:** Especially for particle mapping, departure rates for each filter vary widely during a phase. For example, in Figure 5-7, the departure rate for the filter of the particle shown will be near 0 when cell 4 is processed, but greater than 0.5 for cell 3. This variation can be smoothed by randomizing the order in which the partner particles are fetched from the cell set. A simple way to approximate this is to fetch particles from cells round-robin, rather than cell-at-a-time.

- **Mapping combinations of cells:** For cell mapping, different cells in the cell set vary widely in the probability that their particles will be part of a

neighbor list. For example, in Figure 5-7, the Home cell and cell 3 are much more likely to provide partner particles than the corner cells (2 and 4). Pairing cells appropriately helps sooth the arrival rates.

- **Concentrator logic:** No matter which mapping scheme (particle or cell mapping) is employed, the arrival rate of matching pairs could vary dramatically among filter queues during the run-time. This property of non-uniformity affects the size of queues as well as the efficiency of force pipelines. In order to smooth non-uniformities among filter queue arrivals, concentrator, logic that can efficiently dispatch particle pairs from queues to force pipeline, would be desired. Each filter independently enqueues particle pairs that have passed the selection criteria. An arbiter determines transfer to the force pipeline based on the following logic.

    1. First priority is given to queues that are within one of being full. This is sufficient to prevent data from being dropped. If multiple queues are nearly full, then priority is rotated round-robin.
    2. Otherwise priority is given to queues that are not empty. Again, priority is rotated round-robin.
    3. If multiple queues are nearly full, then the filters are throttled. Note that throttling by itself does not reduce efficiency; the key performance consideration is that the force pipelines always be active.

Thus, a concentrator would have the following three modes.

Normal mode: Pairs are dispatched from non-empty queues in round-robin fashion as shown in

- Figure 5-9.

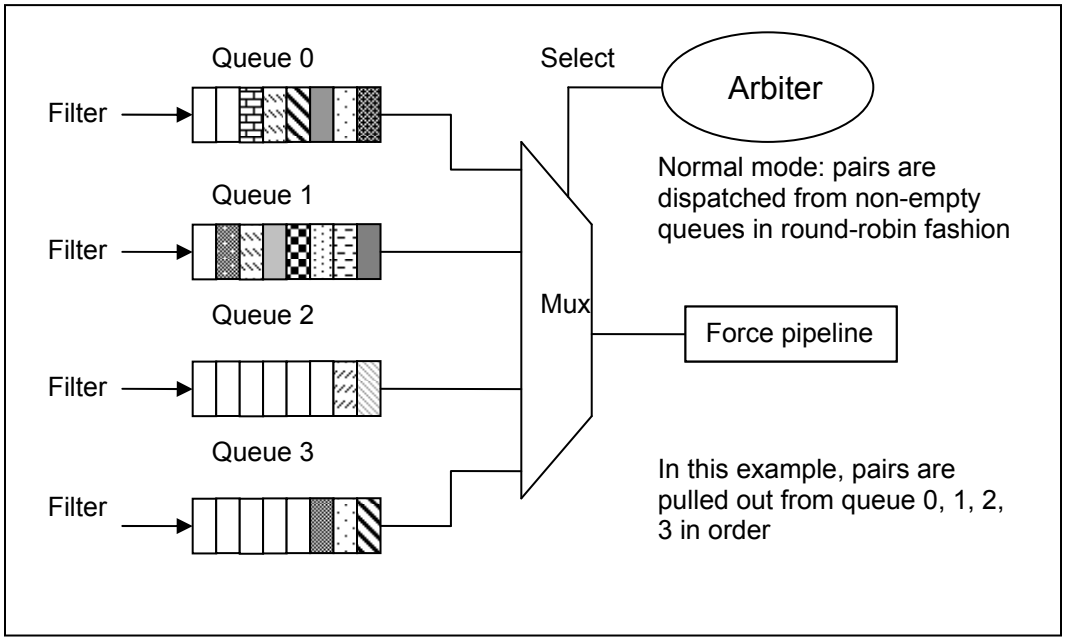

Figure 5-9: Concentrator in normal mode. Pair is dispatched from non-empty queues in round-robin fashion.

- Priority mode: When one of filter queues is full, it continuously dispatches match pairs from this queue until “full” signal is de-asserted as shown in
-
-

- Figure 5-10.

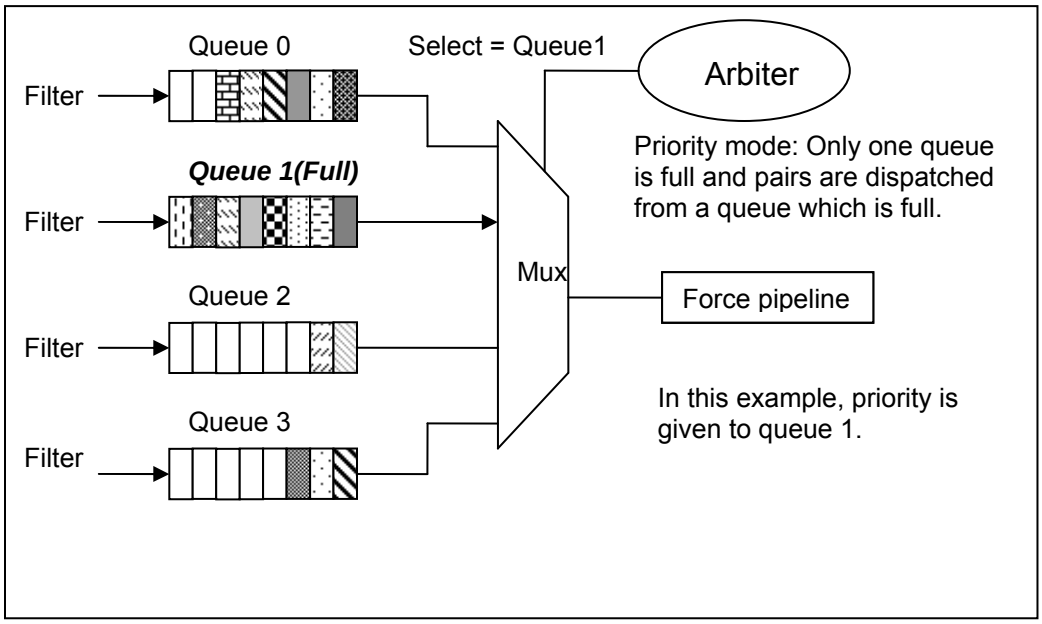

Figure 5-10: Concentrator in priority mode. Queue which is full has high priority to dispatch pairs (only one queue is full).

Throttling mode: When more than one queue is full, stall signal is asserted to halt filter pipelines and particle distribution. Priority is given to any queue which is full. An example is illustrated in

- Figure 5-11. The advantage of this scheme is that small amount of memory resources would be sufficient to maintain high throughput. However, in order to prevent the over-flow where more than one queue is full, throttling logic is required to stall filter pipelines.

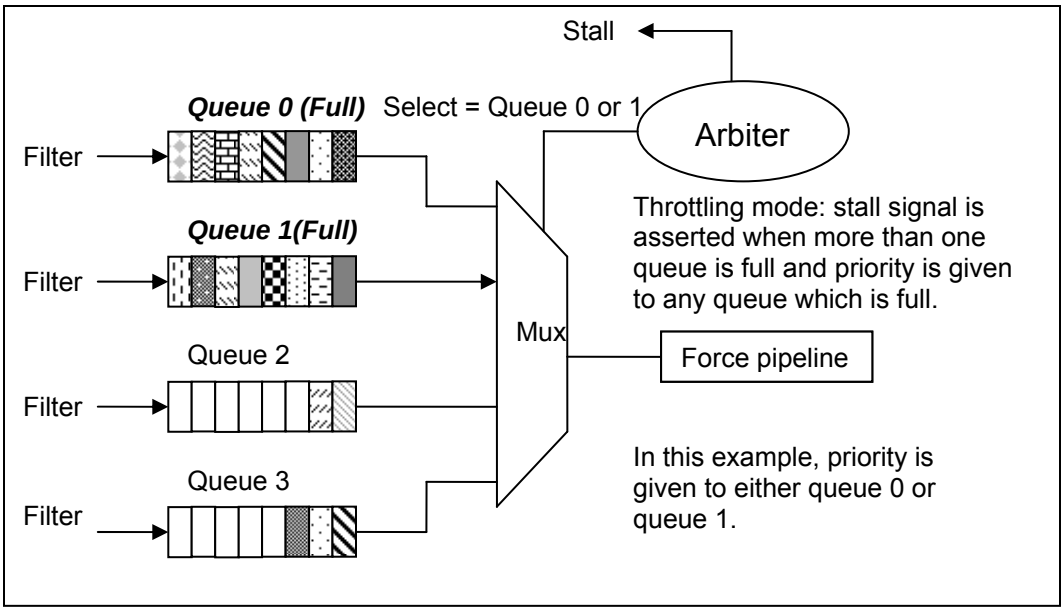

Figure 5-11: Concentrator in throttling mode. Stall signal is asserted and priority is given to any queue that is full.

Another design consideration is whether to over- or under-provision and whether to throttle filter pipeline input to reduce the queue size needed to prevent overflow. Having a smaller or larger number of filters under- or over-provisions the force pipeline. The advantage of under provisioning is that simple hardware is adequate for correct execution. The advantage of the over-provisioning is high utilization of the force pipelines: with nine or more filters in the Perfect/“half-moon” design option the force pipelines are almost always busy. In this case the design requires either larger queues or that the filters be throttled.

Table 5-2 shows various configurations with no throttling. The maximum queue size is that required to prevent overflow with very high probability. The utilization

is the average fraction of cycles that the force pipelines are busy. “Cell mapped” requires smaller queues because it has shorter phases: each filter bank processes one reference particle at a time rather than F. Even the largest queues require much less storage than the neighborlist queuing method from the previous section.

Table 5-2: Queue size requirement and utilization are show for various configurations with no throttling.

Number of filters	Particle mapping				Cell mapping	
	6	7	8	9	6	9
Queue size	10	18	36	80	6	36
Utilization	69.7%	81.2%	92.5%	99.3%	69.6%	98.3%

We now examine the effect of throttling. In this design, the filters all halt when any is in danger of overflow. Since the force pipelines consume every cycle, this happens when multiple queues are within one of full.

Figure 5-12 shows the effect of queue size, number of filters (queues) per force pipeline, and mapping on utilization. Even with over provisioning, utilization can be less than 100% because of non-uniformities in arrivals, and because of start-up and tear-down effects. The key result is that with slight over provisioning, i.e., 9 filters, particle mapped yields 99.2% utilization for a (very small) queue

size of 8. Particle mapped is slightly better than cell mapped because of its more uniform arrivals and longer phase.

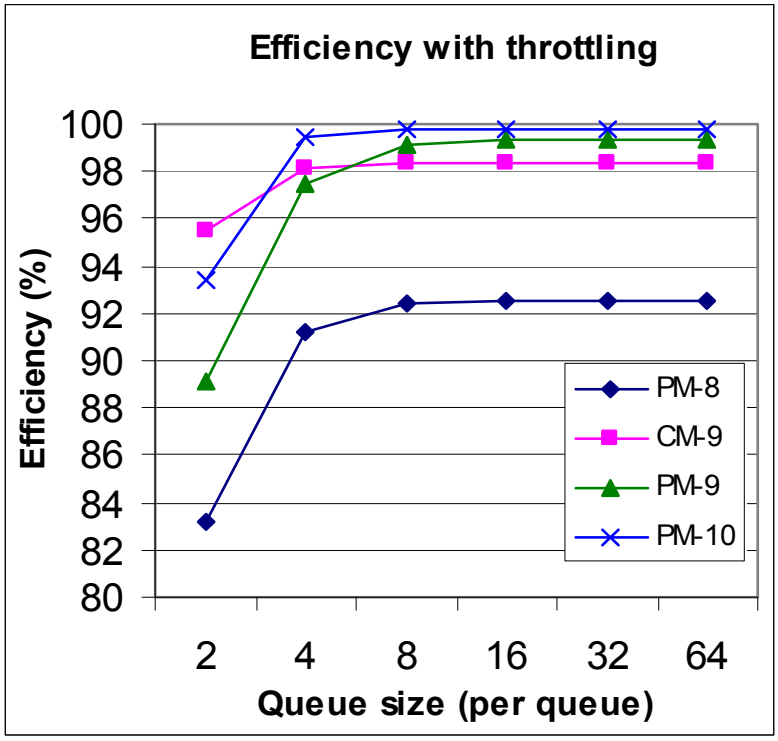

Figure 5-12: Graph shows the effect of queue size on utilization for various numbers of filters (queues) and mappings of particles onto filters. PM is particle-mapping and CM is cell-mapping.

5.7 Pipeline Throughput Analysis

Our base design uses reduced filtering, “half-moon” partitioning, and particle mapping scheme. For other FPGAs, planar filtering may be preferred. For queuing, the method depends on the balance between BRAMs on the one hand and logic and DSP units on the other. Queueing full neighbor lists is preferred in the Stratix-III SL340 (more BRAMs) while using concentrator-based queuing with throttling is preferred in the Stratix-III SE260 (more DSP blocks).

With the current implementation, the system performance is determined by the following factors as follows.

- **Pipeline efficiency:** Queueing entire neighbor lists with double buffering would provide almost 100% pipeline efficiency but costly as discussed earlier in Chapter 5.6.1. If continuous queuing method were used, the efficiency would be lowered with increased numbers of filters. This is due to the non-uniformity of match pair arrival during runtime among force pipelines. Although each force pipeline processes the same amount of particle pairs for a given reference particle on average and can operate individually, all pipelines have to be stalled once one of them is required to do so. Increasing queue size would help reduce the frequency of pipeline stall and thus improve overall pipeline performance.

- **Phase efficiency:** The number of phases necessary to process the particles in a single home cell is $\left\lceil \frac{particles - in - cell}{number - of - filters} \right\rceil$. The performance does not scale linearly with the number of filters or force pipelines. For small cells or low-density simulations, the loss of efficiency can become significant. There are, however, several reasonable solutions.
    - Instead of processing one reference cell at one time, two home cells can be fetched and processed together. This halves the phase granularity, and so the expected inefficiency, without significantly changing the amount of logic required for the distribution bus.
    - Overlap processing of two home cells. This increases the working set from 18 to 27 cells for a modest increase in number of BRAMs required. A second distribution bus may be required.
    - Another solution is to over provisioning. Increase the number of filters and further decouple neighbor list generation from consumption. However, it may impact pipeline efficiency as mentioned above if queuing FIFO size is not large enough. The reasoning is that as long as the force pipelines are busy, some inefficiency in filtering is acceptable.
- **Frequency:** Our current implementation on Stratix III runs at ~200MHz that can be improved in the later FPGA devices. FPGA operating

frequency is strongly correlated to the resource utilization. Higher resource utilizations often imply lower operating frequency. In order to achieve the optimal system performance, our goal should focus on the overall throughput rather than one single factor (e.g., the maximum number of pipelines implemented).

In summary, the time required to process all particles per step can be formulated below.

$$\frac{cell-number * number-of-phases * pairs-a-particle * filter-ratio * pipeline\_eff}{frequecncy}$$

where “pairs a particle” means the number of match pairs for a given reference particle after filtering stage and it is determined by the filtering logic efficiency, cutoff radius, and particle density. “Filter-ratio” is the number of filters within a filter bank and “cell-number” is the number of cells in MD simulation space. “Pipeline_eff” represents the pipeline efficiency, which is correlated to filter queue size as discussed earlier.

5.8 SUMMARY

We have presented a study of filtering that is the first for FPGA-based accelerators and one of only very few for hardware implementations of MD. With only a small amount of logic, high quality filtering can be achieved to improve the overall system performance. Depending on the configurations of FPGA’s hardware resources, two low-cost filtering schemes are available. A new partitioning method for optimizing with respect to Newton’s 3rd Law was also

presented. This is essential for the design presented here, but could also find application in other hardware implementations of MD. The result is that almost a 6x performance improvement can be achieved over previous FPGA-based methods.

An important comparison is with Anton, the ASIC-based MD system from D.E. Shaw [109] that is designed to support hundreds of MD processor chips. As discussed earlier in Chapter 5.3, the reduced-precision filtering scheme was implemented in Anton [74]. There are several design differences. For partitioning, rather than use either the standard “half shell” method or the “half-moon” method proposed here, Anton uses a novel “Neutral Territory” scheme. This especially minimizes interprocessor communication costs. Another consequence is that fewer filters per force pipeline are needed (normalized for throughput). For mapping particle pairs onto force pipelines, Anton uses a scheme similar to the “particle mapping” used here. The design choices in Anton are not all preferred for single chip FPGA versions (in the current FPGA chip architecture). Some of the architectural differences that lead to different design choices are as follows: single chip versus multiple chip; limitations of FPGA routing logic and its implications in control complexity; and the number and type of the FPGA’s hard components, especially BRAMs and multipliers.

Chapter 6 System Design and Integration

The primary goal of this study is to accelerate MD simulations without sacrificing simulation quality. We have presented efficient algorithms and explored the design space for improving FPGA-based designs. In this Chapter, we will illustrate how to integrate our coprocessor design into an MD software package. We will also examine the handing of various integration details, including data transfers, particle exclusion, and cell-lists.

6.1 System Architecture

Our HPRC system consists of a generic host node (e.g., a PC, workstation, or server blade) with an accelerator board plugged into a high-speed PCI Express socket. The host runs the main application program and communicates with the accelerator through function calls. The accelerator board consists of a high-end FPGA, memory blocks, and a bus interface. Besides configurable logic, the FPGA has dedicated components such as independently accessible multiport memories (e.g., 1000 x 1KB) and a similar number of multipliers. The system-level diagram is illustrated in

Figure 6-1.

The FPGA itself is divided into two main components, the user design and the vendor logic. The vendor logic is dedicated to system (non-application) functions, such as memory controllers, and occupies about 10%-15% of the FPGA's logic in our implementation. The user design contains the computational engine of our

MD accelerator, including control logic, filter banks, and force pipelines. In addition to the computational logic, the user design also includes BRAMs, to store particle data and forces, and simulation parameters.

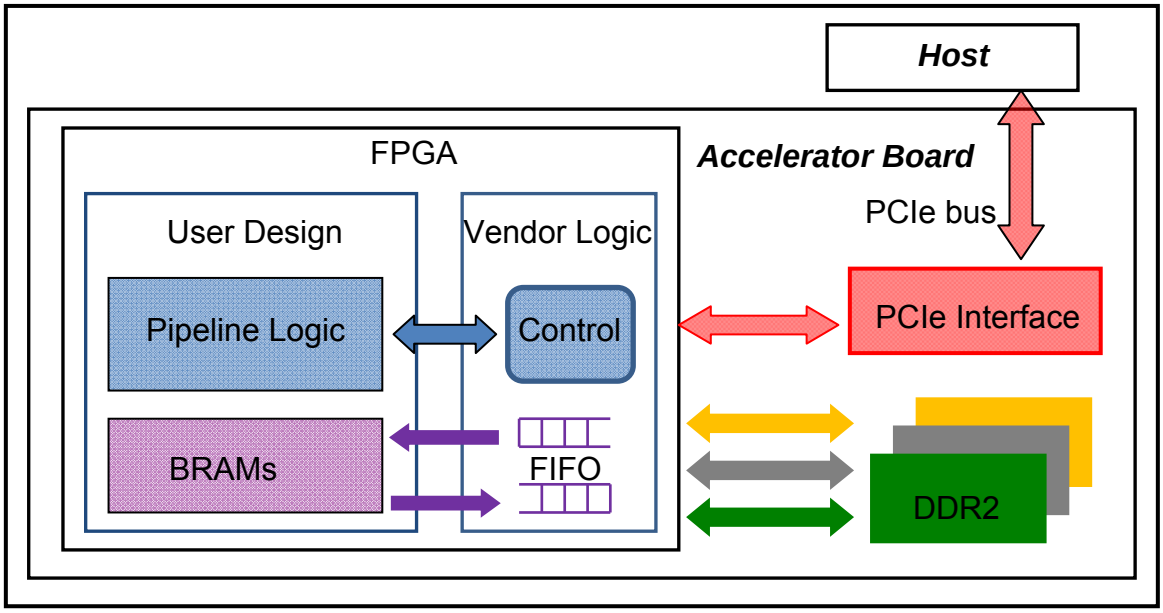

Figure 6-1: System architecture of the FPGA-based MD accelerator.

We have implemented our MD design using a Gidel PROCStar III board that has features similar to those described above. In particular, it contains three memory banks: one is 256MB (Bank A) while the other two (Banks B and C) are 2GB. Particle coordinates and charges are stored in Bank B and particle types are stored in Bank C. Bank A is used to collect the computed forces for each particle. Coordinates, charges and forces are 32-bit single precision floating point numbers. Particle types are represented in reduced precision integer; the precision required depends on how many particles are supported in the

simulation. One limitation of Bank C is that it only runs at half the frequency (167MHz) of Banks A and B (333MHz). It is therefore appropriate to store compact data (e.g., particle types) in Bank C to prevent its slow access from becoming a critical path. The PCI Express interface is responsible for protocol-level communication management between the host and accelerator.

6.2 Integration into the MD code

From a programming standpoint, NAMD-Lite integration has been straightforward. The tasks are as follows: replacement of the short-range force computation with the appropriate accelerator calls, data conversion from double precision floating point to single precision and back again, packing and unpacking the data, handling particle exclusion, and handling cell-lists.

In the following subsections, we will present the control flow between the MD software code and our FPGA-accelerated system. We will also describe changes to the original software code, including supporting particle exclusion and cell lists.

6.2.1 Control Flow

Figure 6-2 shows the control flow between the MD software code and our FPGA-accelerated system for the non-bonded short-range force kernel. The left side illustrates the procedures executed by the software, while the right side shows the steps performed on the accelerator board.

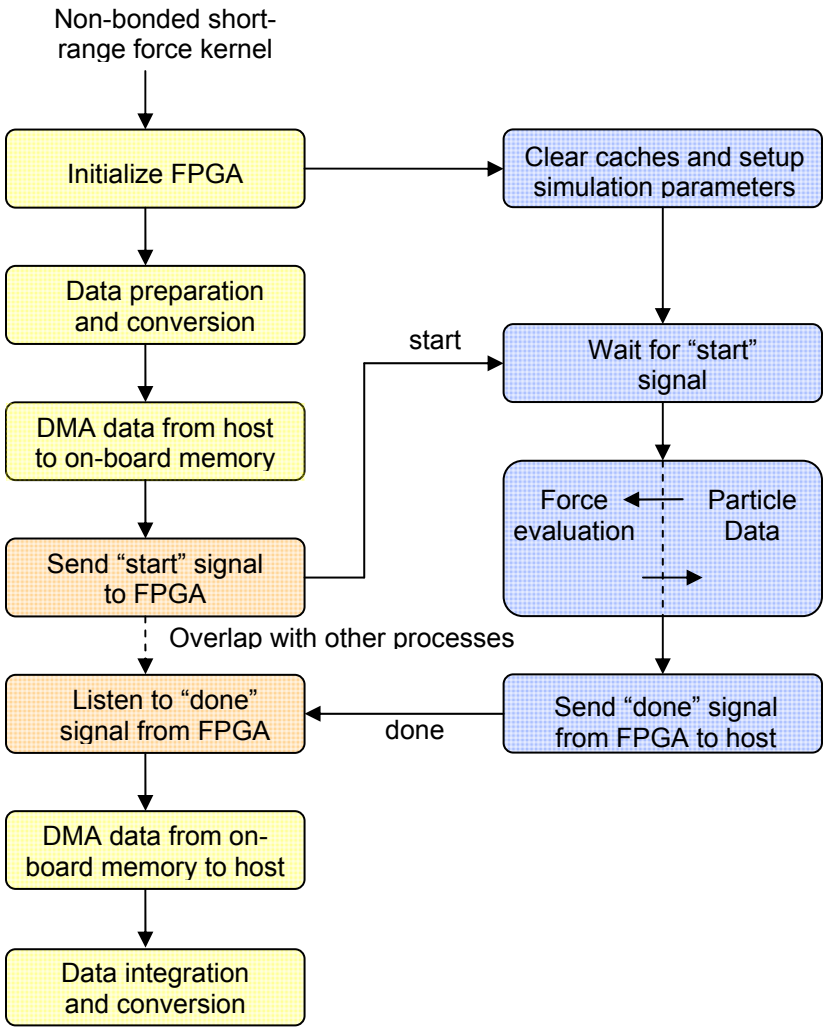

Figure 6-2: Control flow of non-bonded short-range force kernel on the FPGA-based system.

When the non-bonded short-range force kernel is invoked, the program first initializes the coprocessor by clearing on-board memory and on-chip caches. It then sets up the simulation parameters: numerical precision, cell size, and cutoff. Particle data are prepared and packed. They are then DMAed to the on-board memory banks via the PCI Express bus. After the DMA operations have completed, the host issues a “start” signal to hand over the control to the accelerator board. Once the “start” signal is received by the FPGA, the controller on the FPGA initializes the pipelines and loads data from off-chip memory to on-chip caches. Then follows the force evaluations. After all particles are evaluated, a “done” signal is sent back to the host and forces are DMAed back to the host. They are then merged with other force evaluations (e.g., for bonded forces) already computed on the host. The process of force evaluations on the FPGA (marked in blue-color) can be overlapped with those executed on the host to improve performance.

6.2.2 Cell Lists

The method of cell lists has been employed in our MD system to help minimize the number of force computations. In each timestep, particles are assigned to cells based on their coordinates. In most MD software packages, cell lists are implemented with an array of linked lists, where each list corresponds to one cell. The lists themselves consist of a series of indices pointing to the particles within the cell. When particles move across cell boundaries, the indices “move” as well and the lists are updated accordingly. The memory footprint of particle data,

however, i.e. coordinates, type, and charge, remains unchanged. This helps minimize data movement during simulations because the index is much compact than the particle data. The disadvantage of this method is that random data access is required to retrieving a series of particles of one cell: locality in space is not followed by locality in memory. This technique, there, does not align well with our FPGA-based implementation.

Our design requires that the particle data of one cell be fetchable in parallel such that they can be processed by pipelines to achieve the maximum throughput. In the worst-case scenario, particles of one cell all reside in the same memory segment. In that case, particle data must be accessed sequentially. In any case, the overhead of random data access is such that its latency could not be completely hidden.

Fortunately, there is an alternative solution that fits our needs. Instead of using lists of indices, particles' are grouped dynamically by cell and stored together in particle memory. This approach was implemented in Gu's MD design and proven to be highly effective [46].

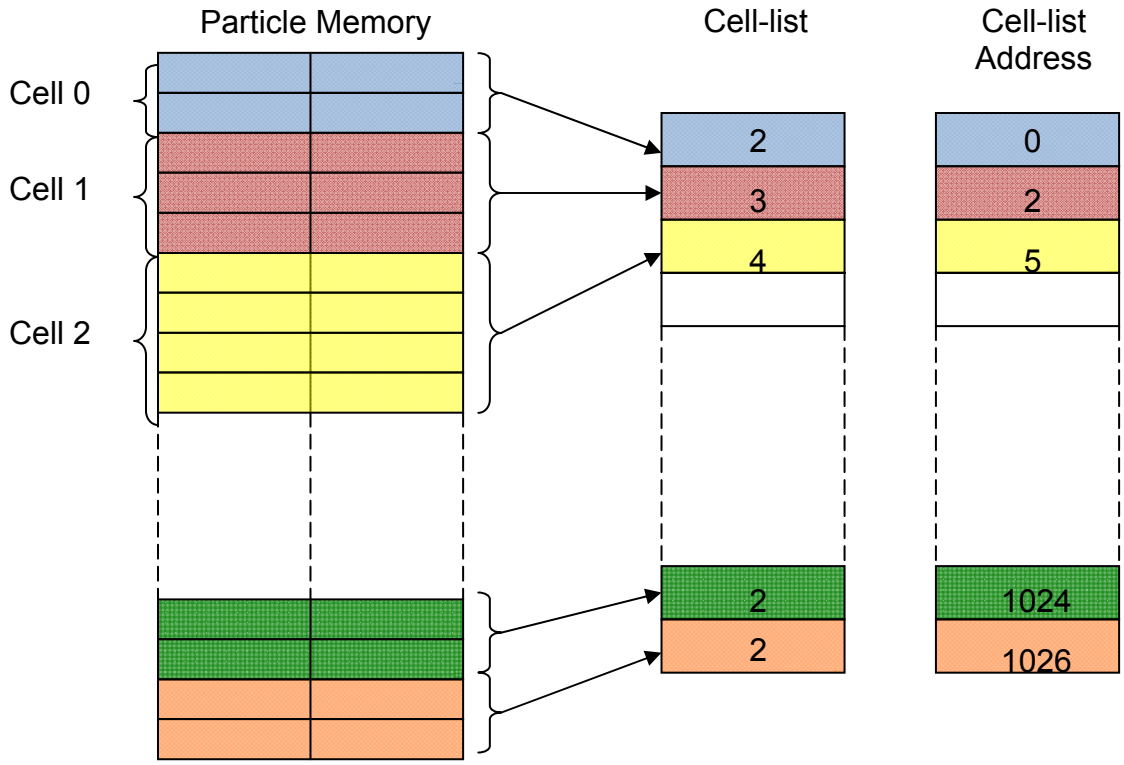

Figure 6-3: Cell list representation in FPGA-based implementation.

As shown in

Figure 6-3, particles from the same cell (marked in the same color) are grouped together in a single segment of particle memory. An element (called a *word*) of particle memory can contain more than one particle. This is to coincide with the word size of off-chip memory. In our implementation, the memory word size is 256-bit while that of particle data is 128-bit (coordinates and charge). Thus, two particles are packed together in the same word. The order of cells in particle memory is fixed *a priori*. A two-level index scheme is used to transfer particles

from on-board memory to on-chip caches and proceeds as follows. Given a particular cell, the cell-list-address is accessed to acquire the starting address of that cell in particle memory. The cell-list is also accessed to retrieve the number of particle words in that particular cell. With the starting address and the number of words, particles can be accessed and loaded from on-board memory to on-chip BRAMS for force computations. This also allows multiple particles of one cell to be fetched simultaneously for higher throughput. Dummy particles are padded at the end of the cell in the particle memory when the number of particles in one cell does not align with the word size of off-chip memory. The cell-list and cell-list address are prepared in the host and then DMAed to the coprocessor. The size of the cell-list and cell-list-address are small and therefore can be stored in on-chip BRAMs.

6.2.3 Particle Exclusion

Particle exclusion refers to the necessity of not computing the non-bonded forces for bonded particles. One common technique used in software codes is to exclude bonded particle pairs based on exclusion pair lists. That is, each particle has associated with it a pairlist that contains the particles with which it is bonded. The “pairlist” scheme is problematic, however, since it requires a fixed particle layout that our cell-list implementation tries to avoid. In addition, exclusion pairlists are needed for each particle and the size of the exclusion lists scales

linearly with the number of particles. As with neighbor lists, exclusion pairlists are not appropriate for large simulations since on-chip caches are usually not large enough to store all exclusion pairlists.

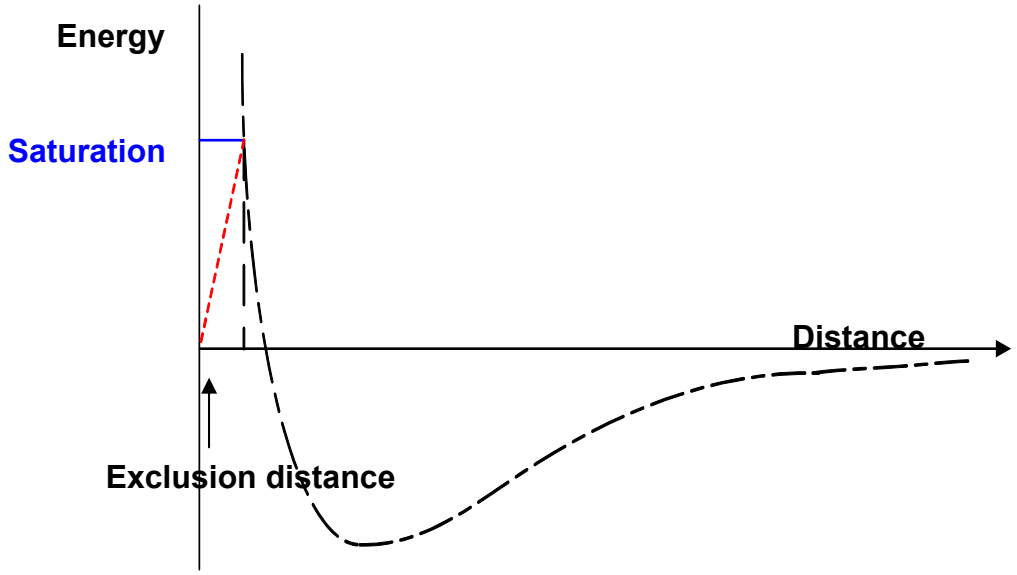

Figure 6-4: Graph shows van der Waals interaction with cutoff check with saturation force.

To support exclusion, our solution is to apply a short cut-off to the non-bonded force calculations based on the fact that two non-bonded particles generally cannot be too close to each other (the atomic radius). Therefore, two particles within a certain short distance must be bonded. The short cut-off distance can be easily calculated by solving the inequality F_{short} < range, where range is the dynamic range with a reasonable force value [46]. The left-side term of the inequality is dominated by the 14 term, $\frac{\sigma}{r^{14}}$. Multiple short cut-off values are required as this value depends on the particle type. A simple graph is shown in

Figure 6-4 to demonstrate this concept [18].

If the exclusion cutoff is chosen conservatively, then two particles would be bonded as long as their intra-distance is smaller than the exclusion distance. For bonded particle pairs whose intra-distance is larger than the exclusion cutoff, the non bonded force is subtracted in the host. There is a problem however. The exclusion distance check in the FPGA is performed in integer arithmetic while it is done in double precision in the host. An inconsistency may therefore occur when the distance between two particles is very close to the exclusion cutoff. In order to minimize the impact of this inconsistency, the

exclusion cutoff should be chosen such that it can be represented precisely in both formats. Then a saturation force is applied if the intra-distance between two particles is smaller than the exclusion cutoff, as shown by the horizontal blue line in
Figure 6-4. Another enhancement is to scale the saturation forces down with distance, as shown by the red diagonal dashed line. This can help avoid overflow in the force accumulation step and improve accuracy.

The choice of exclusion cutoff influences the precision of force accumulators. Since saturation forces often have large values, large precision would be required to avoid overflow. It also affects the simulation quality since large false forces would overwhelms the real ones. Therefore, multiple exclusion cutoffs would be required for various particle types and careful evaluation is essential.

6.3 Memory Management and Data Transfer

In this section, we describe how data are transferred between host and accelerator and between off-chip and on-chip memory. Details of transfers from stage to stage are presented in the succeeding subsections.

6.3.1 Accumulating and Combining Accelerations

In our current MD implementation, the final processing steps are accumulating and combining the accelerations generated by the force pipelines. Unlike position data, which is read-only, acceleration data is read/write. That is, during the processing of a home cell, each particle's acceleration accumulates over this and other cells in the cell set; it is not complete until all 27 cells in the neighborhood have taken a turn as the home cell. Thus for each new home cell, the running

total of accumulated accelerations of the cell set are read onto the chip in a way analogous to the position data.

One design constraint is that each force pipeline handles at most a small number of reference particles Pi at a time. This enables the total forces on the Pis to be accumulated in registers. Accumulating the mutual forces on the Pis' N3L partner particles (the Pj s), however, is more complex as their positions span the cell set. To prevent BRAM access contention, the following strategy is used. Partner updates are written to BRAMs that are associated uniquely with each force pipeline. When processing of a home cell is completed, the partner data from the various pipeline-specific BRAMs are merged [17].

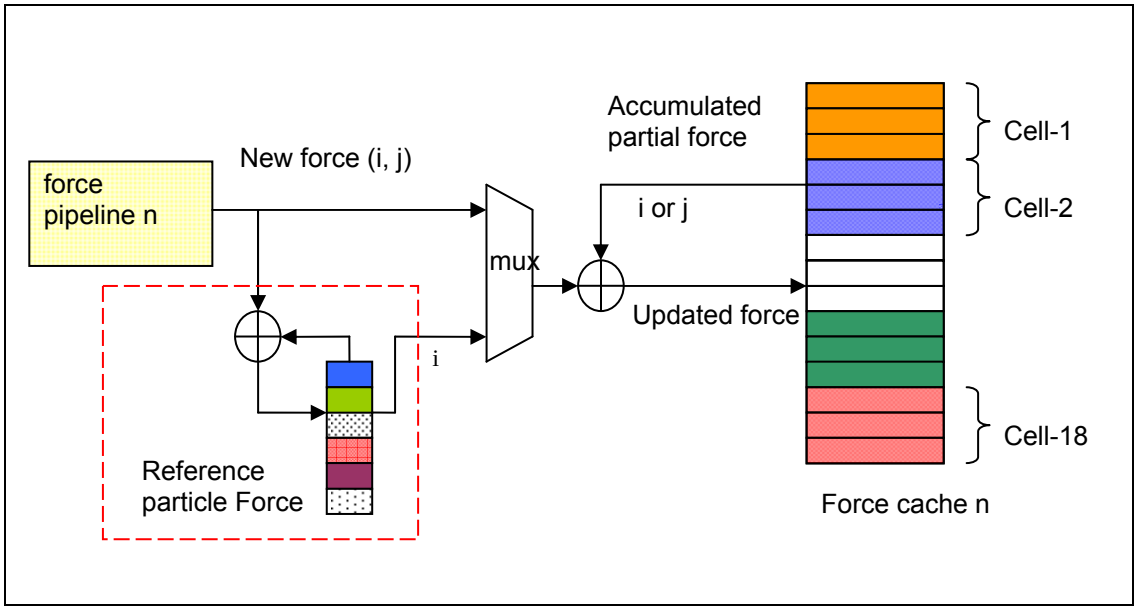

Figure 6-5: Mechanism for accumulating per-particle force. The logic of a single pipeline for both reference and partner particles is shown.

This method is depicted in

Figure 6-5. The running accumulation for a single pipeline during cell processing is shown. We describe this for particle mapping; cell mapping is analogous (see Chapter 5.5). Recall that each of the N_{force} force pipelines has $N_{filters}$ filters and that each filter processes a unique reference particle at a time. Also that reference particles are always from the home cell, but that partner particles come from the entire cell set. For each force pipeline there are $N_{filters}$ accumulators for the $N_{filters}$ reference particles being processed at a time. There are also N_{force} force caches, one for each pipeline. Each force cache has an accumulator for each particle in the entire cell set.

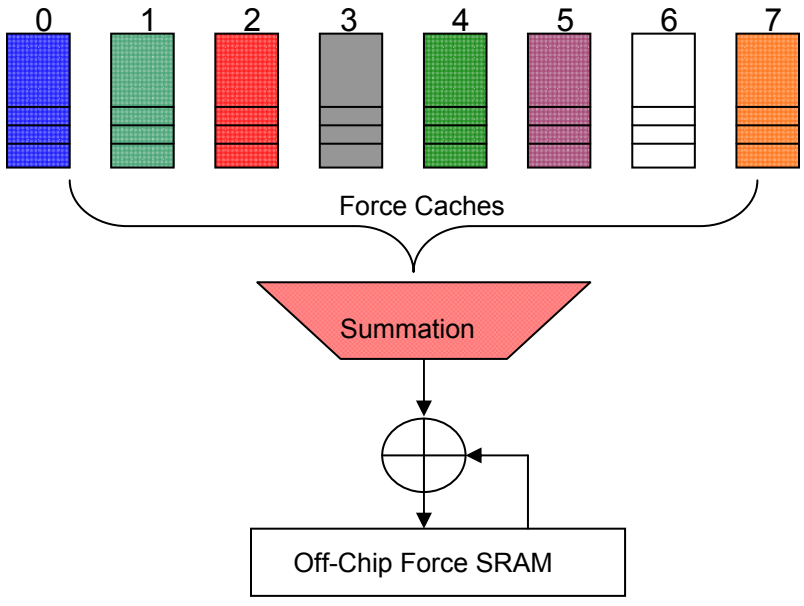

Figure 6-6: The approach of how forces are accumulated across multiple pipelines is illustrated.

Processing proceeds as follows. A new home cell and its accompanying cell set (positions and accelerations) are loaded. From the home cell, a cohort of reference particles is loaded into the filters. Forces are now computed with respect to all of the cell set particles and sent to the accumulators. Each force (for particle pair i, j) is added to both the register corresponding to reference particle i and to the jth slot in that force pipeline's force cache. The accesses to the force cache BRAMs are pipelined: the js are sent a few cycles ahead so that the current accumulated values are available "just in time." When the cohort of reference particles has been processed, the reference particle accumulators in the force array are combined with those in the force cache. When the home cell has been processed, the N_{force} force caches are combined. The concept is illustrated in

Figure **6-6**. The basic design was first appeared in Gu's design [46]. This operation is performed during swapping out, so its latency is completely hidden.

6.3.2 FPGA-Board Data Transfer

In order to support a large MD simulation where on-chip memory is not sufficient to accommodate all particle data, off-chip memory utilization is unavoidable.

Dynamic particle data--including coordinates, charges, accelerations, and types--are updated periodically and can reside in off-chip memory. Static computational parameters, such as interpolation coefficients, type-related constants, and cell lists, which are required during an entire process, can be stored in on-chip memory [46].

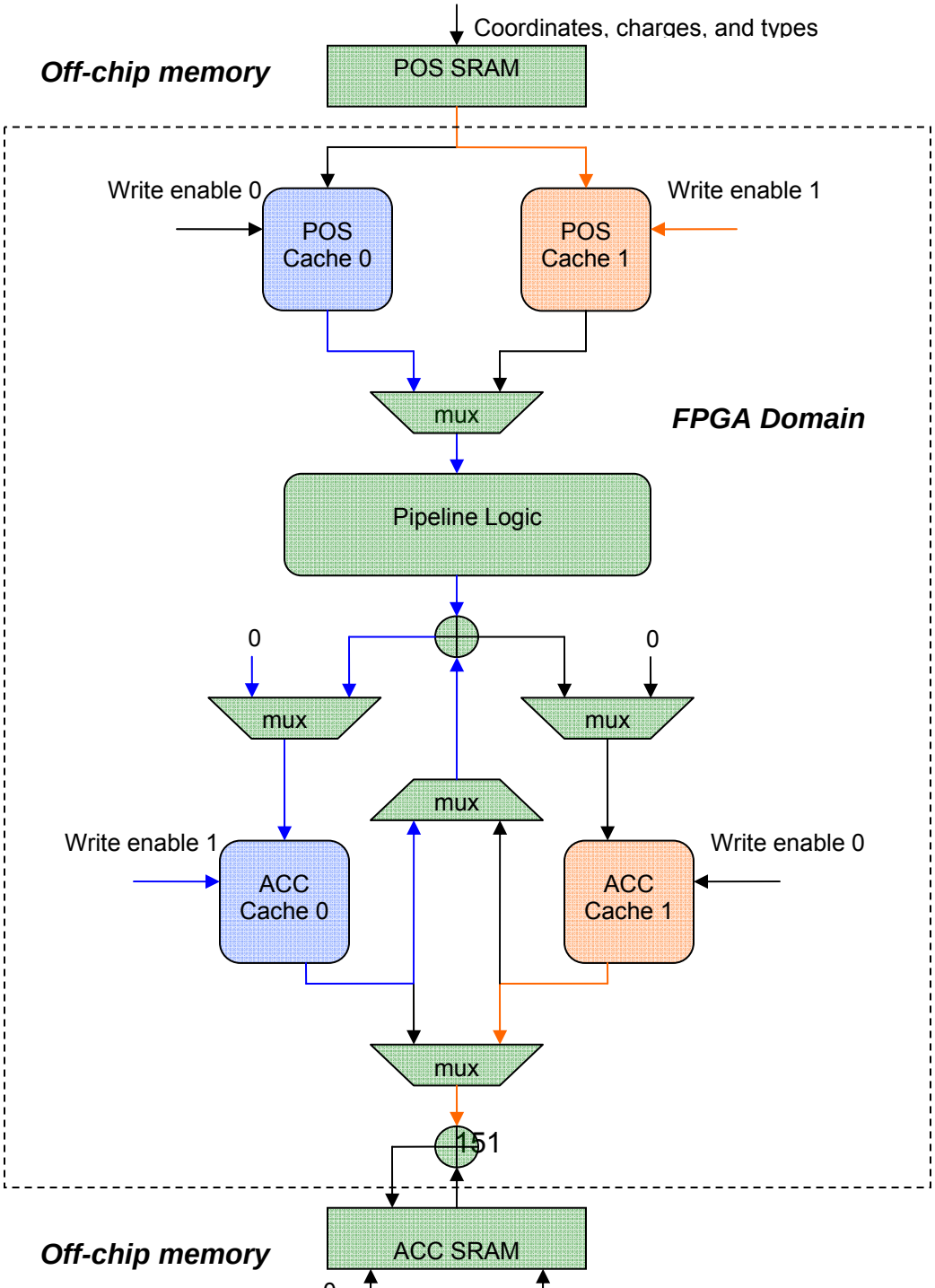

Figure 6-7: Datapaths between off-chip memories, on-chip caches and force pipelines.

The bandwidth of off-chip memory, however, may not be large enough to directly feed particle data into force pipelines. Fortunately, an efficient technique has been developed to overcome this lack by taking advantages of spatial and temporal locality. With the techniques of the cell-lists and cutoff radius, for each iteration, particles are grouped by cell and only interact with the ones residing in their neighboring cells. Hence, only small amounts of data need to be loaded into on-chip memory. Figure 6-7 shows the datapaths between on-board memories, on-chip caches, and force pipelines. Two sets of caches, cache 0 and cache 1, enable double buffering. Each set has POS and ACC caches which hold particle data (coordinates, charges, and types) and acceleration (or force) respectively. POS caches are read-only whereas ACC caches are both read and write. While one set of caches is swapping data with on-board memories, the other one is cooperating with pipeline logic to evaluate pairwise interactions.

An example is illustrated in Figure 6-7. POS cache1 (marked in orange color) loads the particle data of the next cell set, while ACC cache1 (marked in orange color) flushes the computed forces of the previous cell set and merges them with those stored in ACC on-board memory. After all of the results in ACC cache1 are

flushed, ACC cache1 is reset to zero. POS and ACC cache0 (marked in blue) work with the force pipelines to compute forces between particles. The computed forces from the pipelines are accumulated with together with the partial results stored in ACC cache0.

Another observation is that for a given system where the average number of particles per cell is N, the cost of accessing data is $O(N)$ while the cost of processing them is $O(N^2)$. The relative time ratio is about $O(N)$. Thus, a "double-buffering" scheme which overlaps data communications and force computations can hide the communication overhead and avoid pipeline stalls.

6.3.3 Host-Accelerator Data Transfers

At the highest level, processing is built around the timestep iteration and its two phases: force calculation and motion update. During each iteration, the host transfers position, type, and charge data to, and acceleration data from, the coprocessor's on-board memory (POS SRAM and ACC SRAM, respectively) via PCI Express bus. The data transfer between the host and accelerator is done through vendor DMA function calls.

The conversion of data format would be required since most of MD software implementations use double-precision floating point format while single-precision is used in our coprocessor design. The data conversion is performed in the host before transferring them to on-board memory. This results in a more compact data size and thus reduces DMA time. Another alternative method is to perform

data conversion on-the-fly while transferring data to on-board memory. This helps save the conversion time in the host but is not currently supported by our board vendor.

With 32-bit precision, 32 bytes are transferred per particle. While the phases are necessarily serial, the data transfers require only a small fraction of the processing time. For example, in Chapter 7 we described how the short-range force calculation takes about 56ms for a 92K particle benchmark and increases linearly with particle count through the memory capacity of the board. The combined data transfers of 3MB take only 6-7ms. Moreover, since simulation proceeds by cell set, processing of the force calculation phase can begin almost immediately as the data begin to arrive.

6.4 Summary

In order to impact and benefit MD user community, integrating our FPGA-accelerated design into MD software codes is essential. Although the tasks of MD software integration are relatively straightforward from the programmer point of view, it requires additional design changes to the original software codes and some detailed design considerations. Care must be taken that efficient and smooth data transfer between various interfaces can be achieved. Otherwise, the advantage of accelerator would be lost. Sometimes, design changes are not trivial and may require elaborate changes to the original code. Our ultimate goal is to align the application with our accelerator precisely for gaining the best overall performance.

Chapter 7 Results

In this Chapter, we present the results of our FPGA accelerated system from two perspectives, performance improvement and simulation quality. For performance, we concentrate on the nonbonded short-range force kernel. Computation time of various implementations is measured and compared with that of a serial reference code: NAMD. For simulation quality, the total energy of the MD system is plotted and examined to see if there is a significant divergence.

The rest of this Chapter is organized as follows: First, we evaluate the force pipelines proposed in Chapter 4. We do this first for resource utilization and then performance enhancement. Next, the total energy of various MD-FPGA systems is plotted for quality assurance. We also present a preliminary study on scalability to demonstrate performance potential of our MD system. We end by providing a brief discussion of FPGA development cost and portability.

7.1 Experiment Platforms

Our FPGA-MD accelerator has been successfully integrated with NAMD-Lite. As described in Chapter 3, the main features of NAMD-Lite are its great flexibility and ease of use for the development of new methods and algorithms. But although NAMD-Lite is sufficient to be used for examining simulation quality, it is not a proper candidate for performance comparison due to its serial implementation. To demonstrating the highly competitive capabilities of our MD

accelerator, the performance was measured by extracting timing data of nonbonded force kernel via NAMD-Lite and then comparing them with those reported by highly optimized MD packages, NAMD and NAMD-GPU.

We refer to the NAMD benchmark, NAMD2.6 on ApoA1. It has 92,224 particles, a bounding box of 108Å × 108 Å × 78 Å, and a cut-off radius of 12 Å. Coulomb force is evaluated with PME scheme. A switching function is applied to smooth out LJ force when the intra-distance of particle pairs is between 10 and 12 Å. According to a study by Stone, et al. [117], for nonbonded short-range force, this benchmark is executed in 1.78 seconds per iteration on a single core of an Intel core 2 quad-core 2.66 GHz processor.

Our base design uses reduced filtering, “half-moon” partitioning, particle mapping, and has eight filters per force pipeline. For other FPGAs, planar filtering may be preferred. For queuing, the method depends on the balance between BRAMs on the one hand and logic and DSP units on the other. For the Stratix III SL340 (more BRAMs), queuing full neighbor lists is preferred. For the Stratix III SE260, using concentrator-based queuing is preferred.

Our FPGA design was implemented on the Gidel PROCStar III board, which has four Altera Stratix III SE260 FPGAs and a total of 18 GB of on-board DDR memory. The board is housed in a Dell Precision T3400, which has an Intel Core2 Duo 2.8 GHz microprocessor and 2 GB RAM. Only one core is used to execute NAMD-Lite program. The interface is a PCIe x 8 slot. All simulations are performed in a 32-bit Window XP environment.

7.2 Performance Experiments

Resource Utilization

Performance is directly related to resources consumed. Both are shown Table 7-1. LUTn means that both the LJ and short-range part of electrostatic force were evaluated with Look-Up Table (LUT) interpolation of order n and 256 intervals per segment. DC indicates that the LJ force was computed directly while the short-range part of the electrostatic force was computed with third order LUT interpolation. The latter is due to the expensive “erfc” function as described in Chapter 4.3.

All designs have been implemented and run on an FPGA of the Gidel board. Time is per iteration. We note that the number of pipelines increases from 4 to 5 to 6 to 7 with interpolation order 2, 1, and 0, respectively. According to the quality analysis in Chapter 4.4, the six-pipeline design with 1st order interpolation is likely to be preferred. LUT0 timing is not reported since the energy drifts unacceptably during MD simulation as shown in Figure 4-12. This design increases performance by almost 50% over direct computation. The resource utilization results indicate that the limiting factor is the logic. This is used mostly for registers. An interesting observation is that the number of bins is not a major concern and could be doubled if needed to achieve better simulation quality without reducing the number of pipelines [20].

We have also synthesized the designs with respect to the Stratix IV SE530 (post place-and-route) and the results are shown in Table 7.2. After optimization, we anticipate achieving an operating frequency similar to that for the Stratix III. We expect a nearly proportional increase in performance resulting in a time per iteration of about 30 ms.

Table 7-1: Resource utilization and performance of various pipeline configurations on Stratix III SE260 (bin/segment = 256, running @ 200MHz)

	LUT0	LUT1	LUT2	DC
Multipliers	67%	63%	66%	68%
Logic (LUT/Register)	87%	88%	85%	94%
BRAM (M9K)	89%	86%	89%	62%
BRAM (M144K)	87.5%	75%	62.5%	50%
Number of Pipeline	7	6	5	4
Timing (ms)	NA	54	63	72

Table 7-2: Resource utilization and performance of various pipeline configurations on the Stratix IV SE530 (bin/segment = 256; Post Place-and-Route results reported by Altera Quartus 9.1)

	LUT0	LUT1	LUT2	DC
Multipliers	76%	87%	98%	100%
Logic (LUT/Register)	69%	75%	78%	86%
BRAM (M9K)	98%	98%	95%	67%
BRAM (M144K)	100%	100%	94%	75%
Number of Pipeline	12	11	10	8

Hardware resources for various generations of Stratix FPGAs are highlighted in Table 7-3 as well as plotted in Figure 7-1 for better illustration. For the Stratix IV, although logic elements and memory double for each process generation, the performance is limited by the number of available hard multipliers; these have not scaled with process technology. Compared with the Stratix III and IV, Stratix V

FPGAs have large variations in component resources across the family of chips. Depending on the critical component of the specific family member, the performance could vary dramatically. For example, although the Stratix V 5SEEB has abundant logic and memory resources, the performance is expected to be the same as that of Stratix III due to the small number of hard multipliers. In contrast, the Stratix V 5SGSD can fit 3 times as many pipelines as the Stratix III.

As discussed in Chapter 4, FPGA resources have a crucial influence on our implementation. One example is that the planar filtering scheme would be favorable for Stratix V 5SEEB where the performance is mainly limited by multipliers. Another design consideration is the trade-off between the interpolation order of the LUTs and the number of interval per segment; this depends on the availability of logic elements, multipliers, and BRAMs. For a given numerical precision, a LUT with high interpolation order (more logic and multipliers required) and few intervals (few BRAMs) may yield the same simulation quality as one with low order paired with large intervals. By taking advantage of FPGA BRAM architecture and providing flexible design choices, our MD design delivers promising performance on various FPGA configurations and newer chips.

Table 7-3: Altera Stratix FPGA resource overview

	Stratix III SE260 (65nm)	Stratix IV SE530 (40nm)	Stratix V 5SEEB (28nm)	Stratix V 5SGSD8 (28nm)
Equivalent Logic (K)	203	531	950	703
18 x 18 multiplier	768	1024	704	4096
Memory (Mb)	14.7	20..7	52	55

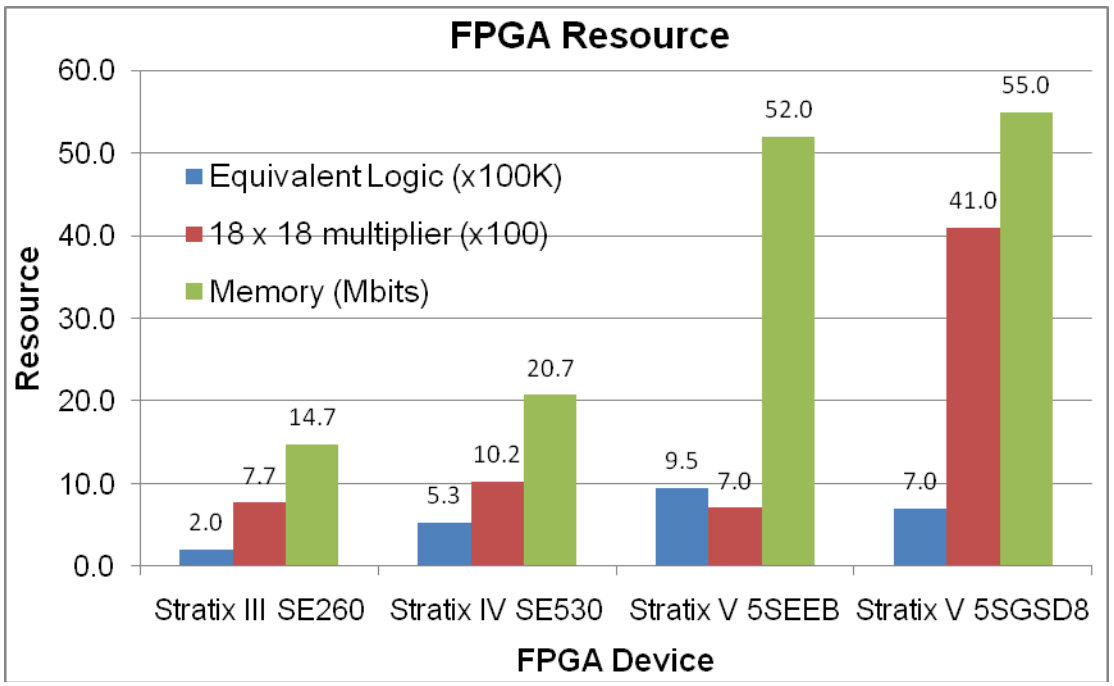

Figure 7-1: Component resources of various FPGAs

Performance Enhancement

A performance profile of our FPGA accelerator is shown in Table 7-4; the details are now described. Here, we focus on the nonbonded short-range force kernel and our base design consisting of six force pipelines on a single FPGA, each with 8 filters, and running at 200 MHz.

Table 7-4: Time profiling of FPGA design (Stratix III ES3SE260, ~200MHz)

	Data preparation	Host to FPGA	FPGA Computation (6 Pipelines)	FPGA to Host	Force Integration
Time (ms)	4.8	5.5	54	1.3	2.2

- Data preparation

  Proper data conversion and restructuring are needed before data transfer. Double precision format is used by many MD software codes, whereas single precision is used in our FPGA implementation. Thus, data conversion is required and performed on the host. An alternative is leaving data conversion to FPGAs. Although it is possible and can shorten data conversion process, it would double data transfer time and storage requirements.

  Since our MD implementation uses cell-lists, particles are grouped together based on their cell index before being transferred to the FPGA board. In addition to particle information (coordinates, charge and type), the cell-list table is also required for data retrieval.

- Communication

  Communication operations include two-way data transfer, upload (from the host to FPGA board) and download (from FPGA board to the host). Upload transfers particle data (coordinate, charges, and types) and cell-list tables to FPGA DDR memory. Download delivers the computed forces

from FPGA to host. In our experiments, download is much faster than upload for two reasons. One is the less data is required; the other is that a faster download speed is supported by the vendor.

- FPGA Computation

  This represents the time the FPGA accelerator takes.

- Force integration

  Nonbonded short-range forces evaluated by FPGAs have to be integrated with others computed on the host, i.e., bonded forces and the long-range part of electrostatic force. Since the computed force by FPGAs is single precision, data conversion is also required.

Figure 7-2 shows the performance of various FPGA implementations over NAMD running on a single core (called NAMD-CPU). The NAMD ApoA1 benchmark is used for performance evaluation. NAMD-GPU was illustrated for further comparison and its performance speedup is compared to NAMD-CPU as well. For the nonbonded short-range force kernel, Stone, et al. [117] reported that this benchmark was executed at 1.78 seconds per iteration on a single core and 0.2 seconds per iteration on a single NVIDIA GeForce 8800 GTX board. It was reported that GPU performance outpaced CPU by 9x.

NBF represents the speedups of the nonbonded force (NBF) kernel only. Total includes communication overheads, data preparation, and force integration as described above. The number on top of each column bar represents the

speedups over a single-core CPU reference model (NAMD-CPU). For our preferred MD design (LUT1), an overall speedup of 26x was obtained over single core NAMD implementation and 3x over a single GPU implementation. If the computational costs on the host (0.16 second per core) remain fixed, our single FPGA implementation can execute 0.38ns simulation per day.

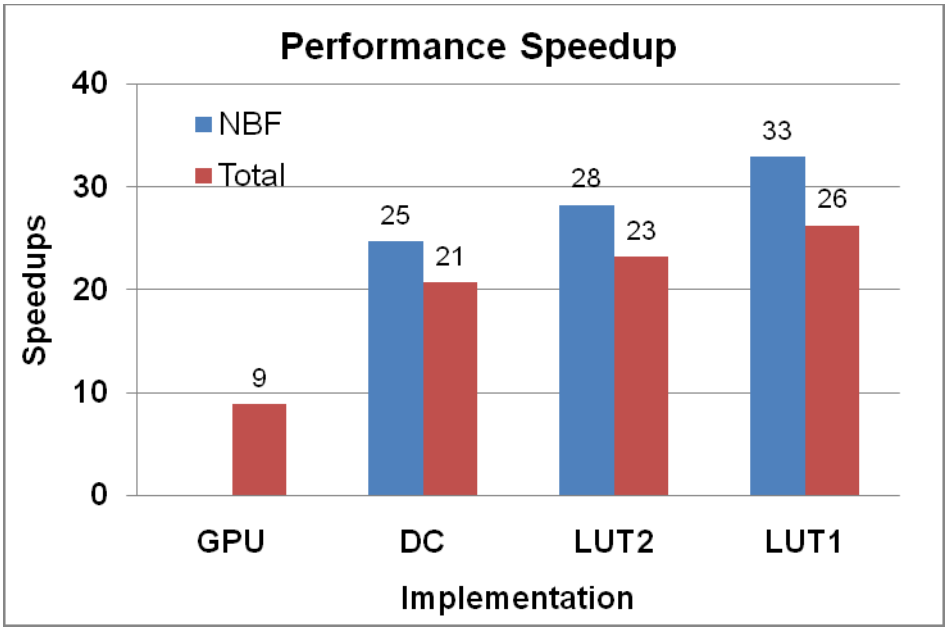

Figure 7-2: Performance speedups of various implementations

As illustrated in Figure 7-2, NBF performance scales almost linearly with the number of force pipelines. The overall performance does not due to extra overheads. One way to minimize the negative impact from overhead is to overlap the short-range force computation with host work. This code must now be parallelized to keep it off from the critical path. Also essential, as for all

accelerators, is efficient communication between host and coprocessor. For simulations of less than a few hundred thousand particles, a conventional I/O bus interfaces should be sufficient.

7.3 Simulation Quality Experiments

In order to validate and measure quality of our FPGA design, energy was plotted as a function of time (see Figure 7-3). In particular, we observe how energy is conserved for various implementations. The result labeled NAMD-Lite is from code running on the processor only and used here as a reference. The other results have the short-range forces computed on the accelerator using table look-up with polynomial interpolation with the order as shown. The time scale is in increments of 100fs. The time step is 1fs.

The main goal of examining energy over time is to ensure that there are no significant errors existed in our algorithms and implementations. As shown in Figure 7-3, there is no noticeable drift. Using Equation 4-13 to compute ΔE we find that the values for all of the FPGA-accelerated codes are smaller than1.0E-4, which is much smaller than the suggested value, 0.003 [107].

These results are preliminary and the time scale may be too short to establish final conclusions. Still we find these results promising: an implementation with 1st and 2nd order polynomial interpolation could have good energy stability. The difficulty in generating longer time-scale simulations is that NAMD-Lite is an un-

optimized serial code and so each of these graphs takes several hours or even days to generate.

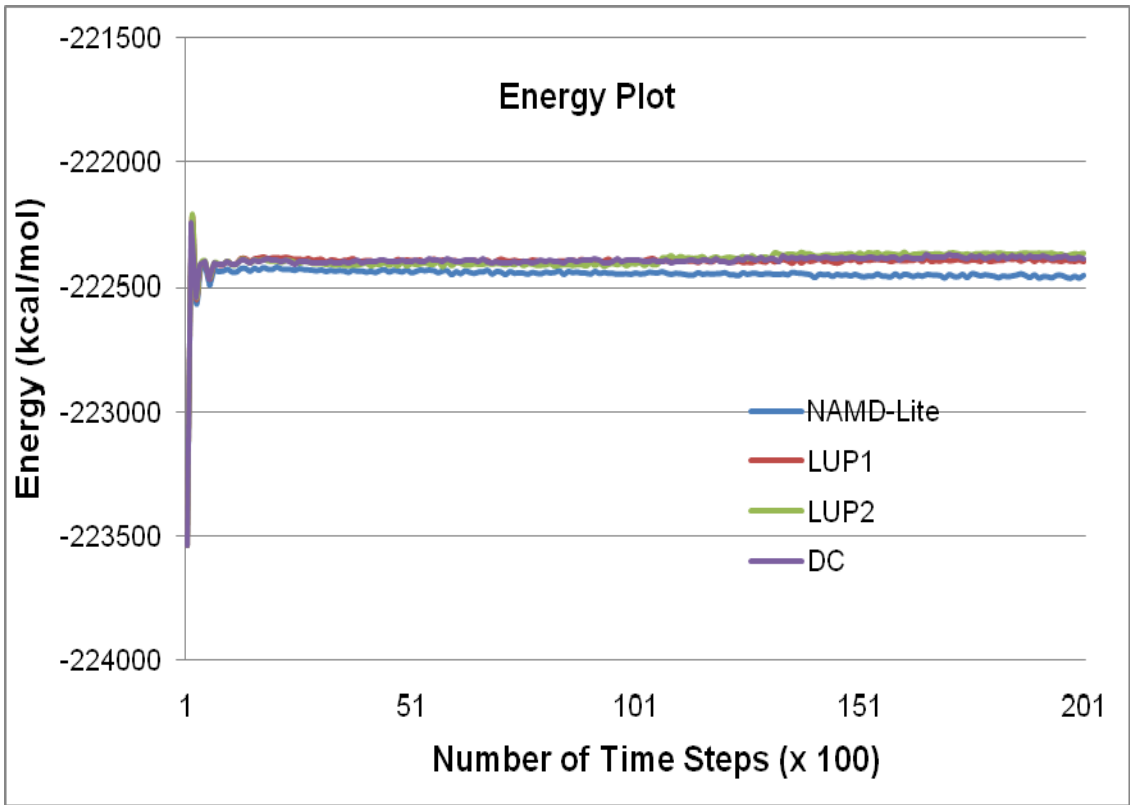

Figure 7-3: Graph of energy plot for various Implementations

Although our focus is energy conservation of various implementations, we noticed that there is a small divergence (0.02%) between FPGA implementations and NAMD-Lite as shown in Figure 7-3. We attribute it to the following reasons.

- **Numerical Precision and Arithmetic**

  Double precision is used in NAMD-Lite to avoid cumulative rounding errors in long simulations and minimize the impact of non-associativity in

floating point arithmetic. In our MD accelerator, mixed-precision manipulation (fixed-point and single precision) is adopted for resource constraint and performance. This reduction in precision, however, may cause variation from the original software.

- **Floating Point Compiler**

  As described in Chapter 4, the Floating Point Compiler may cause different results from the software codes. This is mainly because of the integer format used internally for resource reduction and performance improvement.

- **Particle Exclusion**

  Currently, a saturation force is applied to particles if the intra-distance between two particles is smaller than the exclusion cutoff. Since the exclusion cutoff has to be chosen conservatively to guarantee that two particles are bonded as long as long as their intra-distance is smaller than the exclusion distance, the saturation forces are sometimes relatively large compared to the real ones. Those large “false” forces can overwhelm the real small ones and result in the loss of precision. One enhancement is to have an accumulator with more precision.

7.4 Scalability and Extensions

The performance results reported above are referred to a single FPGA implementation. In this section, we present a preliminary study of the scalability and extensions of our MD FPGA accelerator to demonstrate the potential performance enhancement.

Scaling to Multiple FPGAs

Figure 7.4 shows the performance numbers of multiple FPGAs implementations. Our design contains six force pipelines of LUT1 implementation on an FPGA, each with 8 filters, and runs at nearly 200 MHz. All performance numbers are compared to NAMD-CPU reference model with NAMD ApoA1 benchmark. As illustrated in Figure 7.4, the performance scales linearly with the number of FPGAs. Here we only present the speedup of nonbonded short-range force (NBF) computation. Overheads are excluded. Although the overall performance improvement would be reduced when taking overhead into account, it is sufficient to demonstrate the good scalability of our MD design.

Extensions

For other MD simulations having similar particle density, the FPGA performance scales linearly with the number of particles up to the memory capacity of the FPGA board, or several tens of millions particles. For simulations having much lower density, transfer of cell sets on/off chip becomes the bottleneck. This limitation, however, is a function of current HPRC systems rather than the

FPGAs themselves. Most current HPRC board designs use only a small fraction of the FPGA's available bandwidth.

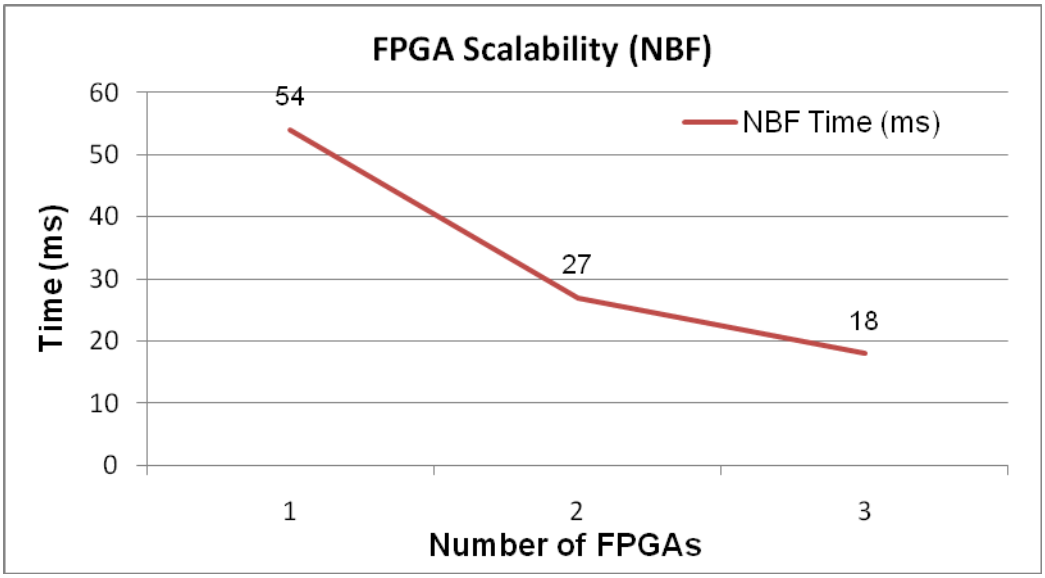

Figure 7-4: Performance speedups of multiple FPGAs (nonbonded short-range force only)

7.5 Power Performance Analysis

We have presented the performance enhancements of our MD accelerator over microprocessors and GPUs. In this section, we examine an additional metric, energy efficiency.

FPGAs are commonly regarded to be very power efficient. A high-end FPGA chip typically consumes at most 20-30 watts. Novo-G reports indicate that each FPGA of a single node consume less than 20W [90]. With the same benchmark [117], a single NVIDIA GeForce 8800 GTX dissipates about 185 watts and an Intel Core 2 Extreme QX6700 quad core CPU dissipates 130-watts.

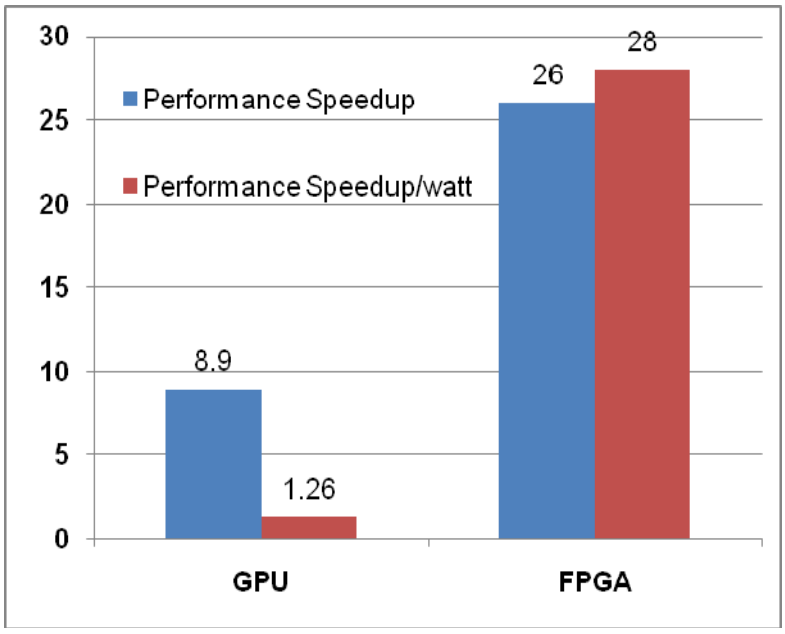

Figure 7-5: Energy-efficienct performance comparison

The performance per watt for two implementations is plotted in Figure 7-5. The performance-per-watt metric is defined as computational time times the power rating. The reference is the CPU-based model. The computational speedup is also shown as a comparison. As stated earlier in Chapter 7.2, we only focus on the non-bonded short-range kernel and omit the bonded force calculation and motion integration. Those are left to the host and only occupy a small amount of the entire computation. As illustrated in Figure 7-5, FPGA demonstrated the best performance in term of computation-to-watt. Although GPUs provide massive computational power, their comparative power-hunger lessens their advantage.

Although the results shown above are approximated and preliminary, they serve as a good indicator of the performance benefits of FPGAs over CPU or GPU-based implementations.

7.6 Development Cost and Portability

Compared with CPU and GPU implementations where highly efficient and mature tools are available and the interfaces are well defined, FPGA-based design often involves the manipulation of cumbersome low-level hardware description language and management of non-standard IO interfaces and protocols. Thus, the development cost of FPGA accelerated solution is relatively high in term of designer hours. It is also worth noting that although CPU-based implementation is relatively cost effective, developing efficient multi-core version may not be straightforward for some applications.

In order to be competitive with multi-core and GPU implementations, the design models have to be ported into the newer devices every few years. Two types of tasks, design model replication and board level integration, are involved during the porting process and are now described.

Design model replication

Porting FPGA design to newer chips mainly involves the task of maximizing the number of pipeline replications. With pipeline parallelism and multiple design options, our MD accelerating system provides easy design scaling as well as maximization of replications.

Board level integration

As long as the board vendor remains unchanged, integrating FPGA designs into a newer board would be relatively straightforward. Otherwise it is a non-trivial work. Every board vendor develops different interfaces and protocols that aim to meet various design goals. This non-standardization of interfaces results in significant effort. Changes in interface logic and software would be required to move the FPGA designs from one vendor's products to another.

From our experience, board level integration is the most timing consuming task during the development process. Due to the unfamiliarity of vendor interface logic and lack of interface simulation models, it makes our integration work challenging, especially for hardware and interface debugging.

7.7 Summary

We have summarized the results of our MD implementations and validated our designs with respect to energy conservation and fluctuation. Our MD accelerator can execute the short-range force for the ApoA1 benchmark in under 70 ms. This represents a 26-fold per core speedup for the computational kernel. Since NAMD scales well, this represents 6.5x speed-up on a quad core implementation. While this benchmark result is a little dated, its microprocessor is comparable in process technology to the Stratix-III that we use here.

We validated our designs by inspecting energy fluctuation and drift. Preliminary analysis showed that the total energy is preserved in our designs.

The development cost and challenges were also addressed briefly to reflect our views of FPGA-based implementations. Finally, we presented a discussion about how our design scales to multiple FPGAs and newer devices. Although it is still preliminary, it shows the potential of scalability and proves that our design continues being promising as the technology progresses.

CHAPTER 8 CONCLUSIONS AND FUTURE WORK

We conclude this thesis by summarizing the work performed in this study and presenting a discussion of how we plan to extend our MD-HPRC work. We also list some lessons we have learned.

8.1 Summary

In this research, we have presented a new implementation of MD for FPGA-based accelerators. We have thoroughly explored the design space of force pipeline implementations with respect to both performance and numerous measures of quality. We have presented a study of filtering that is the first for FPGAs and one of only very few for hardware implementations of MD. The results show that FPGAs are highly competitive with respect to the short-range force computation in MD simulations.

We summarize the results for the force pipeline. We found that look-up table interpolation is somewhat favorable to direct computation for supporting various simulation configurations and complex function implementations. Other results are a demonstration of the Altera Floating Point Compiler, and numerous observations with respect to datapath design parameters. The most important of these is probably that simulation quality of the single precision and hybrid (fixed point/single precision) implementations is comparable to that of full double precision.

In the filtering part of this study, we find that high quality filtering can be achieved with only a small amount of logic. We present a geometric filtering scheme that is preferable for FPGA implementation. We also present a new partitioning method for optimizing with respect to Newton's 3rd Law. This is essential for the design presented here, but could also find application in other hardware implementations of MD. And finally, the scheme of mapping particle pairs to filter pipelines also appears to be new.

We have successfully integrated our FPGA design into NAMD-Lite and this is now running on a workstation containing a single node of the Novo-G, a supercomputer that consists of 192 Stratix-III ES260 FPGAs and [90]. We also conducted performance measurement and quality evaluation. Our accelerator system demonstrated that significant performance enhancement is achieved and HPRC is promising for MD applications.

8.2 Lessons learned

Some lessons have been learned through the process of accelerating MD simulations and are summarized as follows.

- Algorithm reconstruction is essential

  Direct mapping of software serial codes to FPGAs often results unoptimized implementations. Application specific optimizations, including hardware architecture and algorithm restructure, and proper data

formatting and restructuring, is crucial for obtaining competitive performance.

- Precision management is required for high performance and quality

  Although current FPGAs have shown significant floating point computational capability, proper precision management is crucial for achieving high performance. It helps reduce hardware resource utilization and enables more coarse-level parallelism.

- High performance comes from high throughput

  Although achieving high utilization of the FPGA resources is important, the key for high performance computing is the throughput that is measured by the amount of work completed within a given time. High resource utilization often reduces the operation frequency due to routing congestion. Finding the optimal point between the resource utilization and operating frequency is an important factor in creating successful designs.

8.3 Future Directions

We now list some of possible future work.

8.3.1 Design Node Optimization

We have presented a complete MD accelerator system that delivers outstanding performance speedups. There is still room for further optimization.

- Performance

  The current FPGA system (Stratix III based board) where our system is implemented is two generations old. Porting our design into a newer device (Stratix IV or V) allows exploiting more coarse-level parallelism as well as boosting the operating frequency. For example, from the analysis shown in Table 7.2, integrating our system to the latest FPGA would nearly double performance if the same frequency is maintained.

  As described in Chapter 5.7, one potential issue of the current filter implementation is fragmentation. It will become a performance limit when the number of force pipelines is more than 12. Several solutions were proposed in Chapter 5.7 to improve filter phase efficiency.

- Quality

  The initial measurements of simulation quality indicated that our FPGA-based approach is viable, although more testing are needed. Longer simulations are essential to evaluate the overall design quality and provide more data for design optimization and turning.

- Extendibility

  Our current FPGA accelerator is currently working with NAMD-Lite, a pilot program of popular MD program, NAMD. In order to benefit the user community, the integration to NAMD is essential. The tasks mainly involve

the data restructure, generation of cell-list tables, efficient communication between the host and FPGA board, and exclusion of bonded particle pairs.

8.3.2 System Level Parallelization

We have focused on the short-range non-bonded force computation and associated overhead. We now examine our work for HPRC MD simulations as a whole. There are typically three other significant computations in MD simulations: bonded forces, long-range non-bonded forces, and motion integration. Bonded forces and motion integration are generally computed every timestep while the long-range force may be computed every fourth timestep or even less frequently. According to Amdahl's law, the overall performance of our MD system will be constrained by the kernels that are not accelerated. Hence, one reasonable solution to keep improving performance is outsourcing certain tasks to the coprocessor such that they can be removed from the critical path. For example, based on NAMD profiling, the long-range force using PME takes over 200ms [96] and can be accelerated with either GPUs or FPGAs. Hardy et al. have demonstrated a GPU version with speed-up of over 20x [54] for electrostatic energy evaluation.

Another direction of extending our work is to scale our design to multiple FPGAs and use them as coprocessors in parallel systems. Our preliminary study has shown the linear scalability of our MD accelerator for the nonbonded force kernel. Integrating MD accelerators to parallel systems would be reasonable with

some work necessary for performance tuning and system integration. One common challenge in creating parallel systems is the communication overhead. For example, the 3D FFT used in PME requires massive all-to-all data transfers. A way to overcome this problem is utilizing the overwhelming computational power of FPGAs [16, 17, 46, 49], which enables to perform large-scale simulations using relatively smaller number of nodes, resulting in reduced communication for the 3D FFT. Data transfers can be further minimized by employing direct communication among FPGAs themselves, bypassing the host CPUs.

References

[1] S. Alam, P. Agarwal, M. Smith, J. Vetter, and D. Caliga, “Using FPGA devices to accelerate biomolecular simulations,” *Computer*, 40, 3, 66-73, 2007.

[2] M. P. Allen, “Introduction to molecular dynamics simulation,” In *Computational Soft Matter - From Synthetic Polymers to Proteins, NIC Series*, 23, John von Neumann Institute for Computing, 1-28, 2004.

[3] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Oxford University Press, New York, 1989.

[4] Altera Corp., http://www.altera.com/literature/wp/wp-01050-Floating-Point-Compiler-Increasing-Performance-With-Fewer-Resources.pdf, 2007.

[5] Altera Corp., http://www.altera.com/products/fpga.html, 2011.

[6] Altera Corp., http://www.altera.com/literature/wp/wp-01003.pdf, 2011.

[7] J. A. Anderson, C. D. Lorenz, and A. Travesset, “General purpose molecular dynamics simulations fully implemented on graphics processing units,” *Journal of Computational Physics*, 227, 10, 5342-5359, 2008.

[8] T. A. Andrea, W. C. Swope, and H. C. Anderson, “The role of long ranged forces in determining the structure and properties of liquid water,” *Journal of Chemical Physics*, 79, 9, 4576-4584, 1983.

[9] Annapolis Micro Systems, Inc., http://www.annapmicro.com, 2011.

[10] AutoESL Design Technologies, Inc., http://www.autoesl.com, 2011.

[11] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, “Reconfigurable molecular dynamics simulator,” In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 197-206, 2004.

[12] J. A. Board, Jr., C. W. Humphres, C. G. Lambert, W. T. Rankin, and A. Y. Toukmaji, “Ewald and multipole methods for periodic n-body problems,” In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[13] J. Bovay, B. Henderson, H. Lin, and K. Wadleigh, “Accelerators for high performance computing investigation”, High Performance Computing Division, Hewlett-Packard Company, 2007.

[14] K. J. Bowers , E. Chow , H. Xu , R. O. Dror , M. P. Eastwood , B. A. Gregersen , J. L. Klepeis , I. Kolossvary , M. A. Moraes , F. D. Sacerdoti, J. K. Salmon , Y. Shan , and D. E. Shaw, “Scalable algorithms for molecular dynamics simulations on commodity clusters,” In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2006.

[15] A. Brandt, “Multi-level adaptive solutions to boundary-value problems,” *Mathematics of Computation*, 31, 333-390, 1977.

[16] M. Chiu and M. C. Herbordt, “Efficient particle-pair filtering for acceleration of molecular dynamics simulation,” In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2009.

[17] M. Chiu and M. C. Herbordt, “Molecular dynamics simulations on high performance reconfigurable computing systems,” *ACM Transactions on Reconfigurable Technology and Systems (ACM-TRETS)*, 3, 4, 23:1-23:37, 2010.

[18] M. Chiu and M. C. Herbordt, “Towards production FPGA-accelerated molecular dynamics: Progress and challenges,” In *Proceedings of the International Workshop on High-Performance Reconfigurable Computing Technology and Applications*, 2010.

[19] M. Chiu, M. C. Herbordt, and M. Langhammer, “Performance potential of molecular dynamics simulations on high performance reconfigurable computing systems,” In *Proceedings of the International Workshop on High-Performance Reconfigurable Computing Technology and Applications*, 2008.

[20] M. Chiu, M. A. Khan, and M. C. Herbordt, “Efficient calculation of pairwise nonbonded forces,” In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2011.

[21] T. A. Darden, D. M. York, and L. G. Pedersen, “Particle mesh Ewald: An N log N method for Ewald sums in large systems,” *Journal of Chemical Physics*, 98, 10089-10092, 1993.

[22] F. de Dinechin, “The price of routing in FPGAs,” *Journal of Universal Computer Science*, 6, 227-239, 2000.

[23] T. Ebisuzaki, J. Makino, T. Fukushige, M. Taiji, D. Sugimoto, T. Ito, and S. K. Okumura, “Grape project: An overview,” *Publications of the Astronomical Society of Japan*, 45, 269-278, 1993.

[24] R. D. Engle, R. D. Skeel, and M. Drees, “Monitoring energy drift with shadow Hamiltonians,” *Journal of Computational Physics*, 206, 432-452, 2005.

[25] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L.G. Pedersen, “A smooth particle mesh Ewald method,” *Journal of Chemical Physics*, 103, 19, 8577-8593, 1995.

[26] G. Estrin, “Organization of computer systems - the fixed plus variable structure computer,” In *Proceedings of the Western Joint Computer Conference*, 33-40, 1960.

[27] P. P. Ewald, “Die Berechnung optischer und elektrostatischer Gitterpotentiale,“ *Annalen der Physik*, 369, 3, 253-287, 1921.

[28] R. P. Feynman, R.B. Leighton, and M. Sands. In *The Feynman Lectures on Physics*, I, 3-6, 1963.

[29] B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. Giampapa, and M. C. Pitman, “Blue matter: Approaching the limits of concurrency for classical molecular dynamics,” In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2006.

[30] D. R. Flower, K. Phadwal, I. K. Macdonald, P. V. Coveney, M. N. Davies, and S. Wan, “T-cell epitope prediction and immune complex simulation using molecular dynamics: State of the art and persisting challenges,” *Immunome Research,* 6, S4, 2010.

[31] Folding@home, http://folding.stanford.edu, 2011

[32] E. S. Fomin, “Comparison of the Verlet table and cell-linked list algorithms for sequential, vectorized and multithreaded implementations,” *Electronic Journal of Numerical Methods and Programming*, 11, 299-305, 2010.

[33] P. L. Freddolino, A. S. Arkhipov, S. B. Larson, A. McPherson, and K. Schulten, “Molecular dynamics simulation of the complete satellite tobacco mosaic virus,” *Structure*, 14, 3, 437-449, 2006.

[34] D. Frenkel and B. Smit, *Understanding Molecular Simulation: from Algorithms to Applications*, Academic Press, San Diego, California, 2002.

[35] M. S. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. Legrand, A. L. Beberg, D. L. Ensign, C. M. Bruns, and V. S. Pande, "Accelerating molecular dynamic simulation on graphics processing units," *Journal of Computational* Chemistry, 30, 6, 864-872, 2009.

[36] T. Fukushige, J. Makino, T. Ito, S. K. Okumura, T. Ebisuzaki, and D. Sugimoto, "WINE-1: Special-purpose computer for n-body simulations with a periodic boundary condition," *Publications of the Astronomical Society of Japan*, 45, 361-375, 1993

[37] T. Fukushige, M. Taiji, J. Makino, T. Ebisuzaki, and D. Sugimoto, "A highly parallelized special-purpose computer for many-body simulations with an arbitrary central force: MD-GRAPE," *The Astrophysical Journal*, 468, 51-61, 1996.

[38] Gidel, PROCStar III Data Book Version 1.0, 2009.

[39] Gidel, PROCWizard User's Manual, 2009.

[40] Gidel, http://www.gidel.com, 2011.

[41] M. B. Gokhale, C. D. Rickett, J. L. Tripp, C. H. Hsu, and R. Scrofano, "Promises and pitfalls of reconfigurable supercomputing," In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2006.

[42] P. Gonnet, "A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations," *Journal of Computational Chemistry*, 28, 570-573, 2007.

[43] GREEN500, http://www.green500.org/home.php, 2011.

[44] GROMACS, http://www.gromacs.org, 2011.

[45] GROMACS Manual 4.5.3., 148, http://www.gromacs.org, 2011.

[46] Y. Gu, "FPGA acceleration of molecular dynamics simulations," In *PhD Dissertation, Boston University*, 2008.

[47] Y. Gu, T. VanCourt, and M. C. Herbordt, "Accelerating molecular dynamics simulations with configurable circuits," *IEE Proceedings on Computers and Digital Technology*, 153, 3, 189-195, 2006.

[48] Y. Gu, T. VanCourt, and M. C. Herbordt, “Improved interpolation and system integration for FPGA-based molecular dynamics simulations,” In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 21-28, 2006.

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

[50] S. A. Guccione and E. Keller, “Gene matching using JBits,” In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 1168-1171, 2002.

[51] H. Guo, L. Su, Y. Wang, and Z. Long, “FPGA-accelerated molecular dynamics simulations system,” In *Proceedings of the Eighth International Conference on Embedded Computing*, 2009.

[52] T. Hamada and N. Nakasato, “Massively parallel processors generator for reconfigurable system,” In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005.

[53] D. J. Hardy, *NAMD-Lite*, http://www.ks.uiuc.edu/Development/ MDTools/namdlite, University of Illinois at Urbana-Champaign, 2007.

[54] D. J. Hardy, J. E. Stone, and K. Schulten, “Multilevel summation of electrostatic potentials using graphics processing units,” *Journal of Parallel Computing*, 35, 3, 164-177, 2009.

[55] M. Haselman, R. Miyaoka, T. K. Lewellen, and S. Hauck, “FPGA-based data acquisition system for a positron emission tomography (PET) scanner,” In *Proceedings of the international Symposium on Field Programmable Gate Arrays*, 2008.

[56] M.C. Herbordt, Y. Gu, T. VanCourt, J. Model, B. Sukhwani, and M. Chiu, “Computing models for FPGA-based accelerators with case studies in molecular modeling,” In *Proceedings of the Reconfigurable Systems Summer Institute*, 2008.

[57] M. C. Herbordt, J. Model, B. Sukhwani, Y. Gu, and T. VanCourt, “Single pass streaming BLAST on FPGAs,” *Parallel Computing*, 33, 10-11, 741-756, 2007.

[58] M. C. Herbordt, B. Sukhwani, M. Chiu, and M.A. Khan, “Production floating point applications on FPGAs,” In *Proceedings of Symposium on Application Accelerators in High Performance Computing*, 2009.

[59] B. Hess, C. Kutzner, D. van der Spoel and E. Lindahl, “GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation,” *Journal of Chemical Theory and Computation*, 4, 3, 435-447, 2008.

[60] W. Humphrey, A. Dalke and K. Schulten, “VMD - Visual molecular dynamics,” *Journal of* Molecular Graphics, 14, 33-38, 1996.

[61] Impulse Accelerated Technologies, http://rssi.ncsa.illinois.edu/proceedings/tutorial/Impulse.pdf, 2008.

[62] Impulse Accelerated Technologies, http://www.impulseaccelerated.com, 2011.

[63] Intel Corp., http://newsroom.intel.com/community/intel_newsroom/blog/tags/e600c, 2010.

[64] A. Jacob, J. Lancaster, J. Buhler, and R. D. Chamberlain, “FPGA-accelerated seed generation in Mercury BLASTP,” In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2007

[65] Jaguar, http://www.nccs.gov/computing-resources/jaguar/, 2011.

[66] L. V. Kale, G. Zheng, C. W. Lee, and S. Kumar, “Scaling applications to massively parallel machines using projections performance analysis tool,” *Future Generation Computer Systems*, 22, 3, 347-358, 2006.

[67] M. Karplus and J. A. McCammon, “Molecular dynamics simulations of biomolecules,” *Nature Structural Biology*, 9, 9, 2002.

[68] A. Kawai, T. Fukushige, and J. Makino, “$7.0/Mflops astrophysical n-body simulation with treecode on GRAPE-5,” In *Processing of Supercomputing*, 1999.

[69] V. Kindratenko and D. Pointer, “A case study in porting a production scientific supercomputing application to a reconfigurable computer,” In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006

[70] M. G. Kiselev, B. G. Abrosimov, I. I. Vaisman, and Y. M. Kessler, “Error estimation in molecular dynamics experiments with a tabulated intermolecular interaction potential,” *Molecular Simulation*, 1, 5, 321–326, 1988.

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

[72] M. Langhammer, “Floating point datapath synthesis for FPGAs,” In *Proceedings of the IEEE Conference on Field Programmable Logic and Applications*, 355-360, 2008.

[73] M. Langhammer and T. VanCourt, “FPGA floating point datapath compiler,” In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 259-262, 2009.

[74] R. H. Larson, J. K. Salmon, R. O. Dror, M. M. Deneroff, C. Young, J. P. Grossman, Y. Shan, J. L. Klepeis, and D. E. Shaw, “High-throughput pairwise point interactions in Anton, a specialized machine for molecular dynamics simulation,” In *Proceedings of the International Symposium on High Performance Computer Architecture*, 331–342, 2008.

[75] S. Lee, “An FPGA implementation of the smooth particle mesh Ewald reciprocal sum compute engine (RSCE),”In *Master's thesis, University of Toronto*, 2005.

[76] E. Lindahl, B. Hess and D. van der Spoel, “GROMACS 3.0: A package for molecular simulation and trajectory analysis,” *Journal of Molecular Modeling,* 7, 8, 306-317, 2001.

[77] A. Mahram and M. C. Herbordt, “Fast and accurate NCBI BLASTP: Acceleration with multiphase FPGA-based prefiltering,” In *Proceedings of the International Conference on Supercomputing*, 2010.

[78] T. Matthey, T. Cickovski, S. S. Hampton, A. Ko, Q. Ma, M. Nyerges, T. Raeder, T. Slabach, and J. A. Izaguirre, “ProtoMol: An object-oriented framework for prototyping novel algorithms for molecular dynamics,” *ACM Transactions on Mathematical Software*, 30, 3, 237-265, 2004.

[79] W. Mattson and B. M. Rice, “Near-neighbor calculations using a modified cell-linked list method,” *Computer Physics Communications,* 119, 135-148, 1999.

[80] J. A. van Meel, A. Arnold, D. Frenkel, S. F. P. Zwart, and R.G. Belleman, “Harvesting graphics power for MD simulations,” *Molecular Simulation,* 34, 3*,* 259-266*,* 2008.

[81] Mitrionics, http://www.mitrionics.com, 2011.

[82] S. E. Murdock, K. Tai, M. H. Ng, S. Johnston, B. Wu, H. Fangohr, C. A. Laughton, J. W. Essex, and M. S. P. Sansom, “Quality assurance for biomolecular simulations,” *Journal of Chemical Theory Computation,* 2, 6, 1477-1481, 2006.

[83] Nallatech, http://www.nallatech.com, 2011.

[84] NAMD, http://www.ks.uiuc.edu/Research/namd, 2011.

[85] NAMD-lite, http://www.ks.uiuc.edu/Development/MDTools/namdlite, 2011.

[86] T. Narumi, R. Susukita, T. Ebisuzaki, G. McNiven, and B. Elmegreen, “Molecular dynamics machine: Special-purpose computer for molecular dynamics simulations,” *Molecular Simulation*, 21, 401 - 415, 1999.

[87] T. Narumi, R. Susukita, H. Furusawa, and T. Ebisuzaki, “46 tflops special-purpose computer for molecular dynamics simulations: Wine-2,” In *Proceedings of the International Conference on Signal Processing,* 575-582, 2000.

[88] T. Narumi, R. Susukita, T. Koishi, K. Yasuoka, H. Furusawa, A. Kawai, and T. Ebisuzaki, “1.34 tflops molecular dynamics simulation for NaCl with a special purpose computer: MDM,” In *Proceedings of the ACM/IEEE International Conference on Supercomputing*, 2000.

[89] L. Nilsson, “Efficient table lookup without inverse square roots for calculation of pair wise atomic interactions in classical simulations,” *Journal of Computational Chemistry*, 30, 9, 1490-1498, 2009.

[90] Novo-G, http://www.chrec.org/~george/Novo-G.pdf, 2010.

[91] NVIDIA, http://www.nvidia.com/object/namd_on_tesla.html, 2011.

[92] NVIDIA, http://www.nvidia.com/page/8800_tech_briefs.html, 2011.

[93] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, “GPU Computing,” In *Proceedings of the IEEE*, 96, 5, 879-899, 2008.

[94] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, “Scalable molecular dynamics with NAMD,” *Journal of Computational Chemistry*, 26, 1781-1802, 2005.

[95] L. Phillips, R. S. Sinkovits, E. S. Oran, and J. P. Boris, “The Interaction of shocks and defects in Lennard-Jones crystals,” *Journal of Physics: Condensed Matter*, 5, 35, 6357-6376, 1993.

[96] J. C. Phillips, J. E. Stone, and K. Schulten, “Adapting a message-driven parallel application to GPU-accelerated clusters,” In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2008.

[97] ProtoMol, http://protomol.sourceforge.net, 2011.

[98] B. Quentrec and C. Brot, *Journal of Computational Physics*, 13, 1975.

[99] B. Radunovic and V. M. Milutinovic, “A survey of reconfigurable computing architectures,” In *Proceedings of the International Workshop on Field-Programmable Logic and Applications*, 1998.

[100] D. C. Rapaport, *The Art of Molecular Dynamics Simulation*, Cambridge University Press, 2004.

[101] C. I. Rodrigues, D. J. Hardy, J. E. Stone, K. Schulten, and W.-M. W. Hwu, “GPU acceleration of cutoff pair potentials for molecular modeling applications,” In *Proceedings of the 5th conference on Computing Frontiers*, 2008.

[102] C. Sagui and T. A. Darden, “Molecular dynamics simulations of biomolecules: Long-range electrostatic effects,” *Annual Review of Biophysical and Biomolecular Structures*, 28, 155-179, 1999.

[103] S, Samir, “Field Programmable Gate Arrays FPGAs [Internet]”, version 18. Knol. http://knol.google.com/k/samir-s/field-programmable-gate-arrays-fpga-s/2cnzhswl7e31n/7, 2010.

[104] R. Scrofano, M. Gokhale, F. Trouw, and V. K. Prasanna, “A hardware/software approach to molecular dynamics on reconfigurable computers,” In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006.

[105] R. Scrofano and V. K. Prasanna, “Computing Lennard-Jones potentials and forces with reconfigurable hardware,” In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2004.

[106] R. Scrofano, and V. K. Prasanna, “Preliminary investigation of advanced electrostatics in molecular dynamics on reconfigurable computers,” In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2006.

[107] Y. Shan, J. L. Klepeis, M. P. Eastwood, R. O. Dror, and D. E. Shaw, “Gaussian split Ewald: A fast Ewald mesh method for molecular simulation,” *Journal of Chemical Physics*, 122, 5, 054101:1-13, 2005.

[108] D. E. Shaw, “A fast, scalable method for the parallel evaluation of distance-limited pairwise particle interactions,” *Journal of Computational Chemistry*, 26, 1318-1328, 2005.

[109] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao, M. P. Eastwood, J. Gagliardo, J. P. Grossman, C. R. Ho, D. J. Ierardi, I. Kolossváry, J. L. Klepeis, T. Layman, C. McLeavey, M. A. Moraes, R. Mueller, E. C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, and S. C. Wang, “Anton, a special-purpose machine for molecular dynamics simulation,” In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 2007

[110] D. E. Shaw, R. O. Dror, J. K. Salmon, J. P. Grossman, K. M. Mackenzie, J. A. Bank, C. Young, M. M. Deneroff, B. Batson, K. J. Bowers, E. Chow, M. P. Eastwood, D. J. Ierardi, J. L. Klepeis, J. S. Kuskin, R. H. Larson, K. Lindorff-Larsen, P. Maragakis, M. A. Moraes, S. Piana, Y. Shan, and B. Towles, “Millisecond-scale molecular dynamics simulations on Anton,” In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2009.

[111] G. Shi and V. Kindratenko, “Implementation of NAMD molecular dynamics non-bonded force-field on the cell broadband engine processor,” In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, 2008.

[112] R. D. Skeel, I. Tezcan, and D. J. Hardy, “Multiple grid methods for classical molecular dynamics,” *Journal of Computational Chemistry*, 23, 673-684, 2002.

[113] SLASH GEAR, http://www.slashgear.com/intel-stellarton-atom-e600fpga-promises-flexible-embedded-devices-14102251, 2010

[114] M. Snir, “A note on n-body computations with cutoffs,” *Theory of Computing Systems*, 37, 2, 295-318, 2004.

[115] D. van der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark and H. J. C. Berendsen, “GROMACS: Fast, flexible and free,” Journal *of Computational Chemistry*, 26, 16, 1701-1718, 2005.

[116] STMV. http://www.ks.uiuc.edu/Research/STMV

[117] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten, “Accelerating molecular modeling applications with graphics processors,” *Journal of Computational Chemistry*, 28, 2618-2640, 2007.

[118] B. Sukhwani, “Accelerating molecular docking and binding site mapping using FPGAs and GPUs,” In *PhD Dissertation, Boston University*, 2010.

[119] B. Sukhwani and M.C. Herbordt, “Acceleration of a production rigid molecule docking code,” In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2008.

[120] B. Sukhwani and M.C. Herbordt, “FPGA-based acceleration of CHARMM-potential minimization,” In *Proceedings of the International Workshop on High-Performance Reconfigurable Computing Technology and Applications*, 2009.

[121] J. V. Sumanth, D. R. Swanson, and H. Jiang, “Performance and cost effectiveness of a cluster of workstations and MD-GRAPE 2 for MD simulation,” In *Proceedings of the International Symposium on Parallel and Distributed Computing*, 2003.

[122] M. Taiji, “MDGRAPE-3 chip: A 165-Gflops application-specific LSI for molecular dynamics simulations,” *IEEE Hot Chips Symposium*, 2004. http://www.hotchips.org/archives/hc16.

[123] M. Taiji, T. Narumi, Y. Ohno, N. Futatsugi, A. Suenaga, N. Takada, and A. Konagaya, “Protein explorer: A petaflops special-purpose computer system for molecular dynamics simulations,” In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2003.

[124] TOP500, http://www.top500.org/lists/2010/11, 2010.

[125] S. Toyoda, H. Miyagawa, K. Kitamura, T. Amisaki, E. Hashimoto, H. Ikeda, A. Kusumi, and N. Miyakawa, "Development of MD engine: High-speed accelerator with parallel processor design for molecular dynamics simulations," *Journal of Computational Chemistry*, 20, 2, 185-199, 1999.

[126] K. D. Underwood, "FPGAs vs. CPUs: Trends in peak floating-point performance," In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, 171-180, 2004.

[127] L. Verlet, "Computer experiments on classical fluids. I. thermodynamical properties of Lennard-Jones molecules," *Physical Review*, 159, 1, 98-103, 1967.

[128] J. Villareal, J. Cortes, and W. Najjar, "Compiled code acceleration of NAMD on FPGAs," In *Proceedings of Reconfigurable Systems Summer Institute*, 2007.

[129] VMD. http://www.ks.uiuc.edu/Research/vmd, 2011.

[130] D. B. Wang, F. B. Hsiao, C. H. Chuang, and Y. C. Lee, "Algorithm optimization in molecular dynamics simulation," *Computer Physics Communications*, 177, 7, 551-559, 2007.

[131] U. Welling and G. Germano, "Efficiency of linked cell algorithms," *Computer Physics Communications*, 182, 3, 611-615, 2011.

[132] Wildstar II., Annapolis Micro Systems, Inc. WILDSTAR II Hardware Reference Manual, 2003.

[133] D. Wolff and W. Rudd, "Tabulated potentials in molecular dynamics simulations," *Computer Physics Communications*, 120, 1, 20-32, 1999.

[134] XD1000 Development System. http://www.xtremedata.com, 2007.

[135] Xilinx, Inc., http://www.xilinx.com/company/history.htm, 2011.

[136] Xilinx, Inc., http://www.xilinx.com/bvdocs/whitepapers/wp245.pdf, 2011.

[137] XtremeData Inc., http://www.xtremedata.com, 2010.

[138] X. Yang, S. Mou, and Y. Dou, "FPGA-accelerated molecular dynamics simulations: An overview," In *Proceedings of the International Conference on Reconfigurable Computing: Architectures, Tools and Applications*, 4419, 293-301, 2007.

[139] Z. Yao, J. Wang, G. Liu, and M. Cheng, “Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method,” *Computer Physics Communications*, 161, 1-2, 27-35, 2004.

[140] L. Zhuo, G. R. Morris, and V. K. Prasanna, “Designing scalable FPGA-based reduction circuits using pipelined floating-point cores,” In *Proceedings of the 12th Reconfigurable Architectures Workshop*, 2005.

Vita

SHIHCHIN MATTHEW CHIU

Electrical and Computer Engineering
Boston University
8 Saint Mary's Street, ECE, Boston, MA, 02215

Cell: (408) 813-3639
Email:mattchiu@bu.edu

Education

- PhD, Computer Engineering — May, 2011
  Boston University, Boston, MA — GPA: 4.0/4.0
  Dissertation Title:
  Accelerating Molecular Dynamics Simulations with High Performance Reconfigurable Systems
- M.S., Electrical Engineering — Dec., 2000
  University of Southern California, Los Angeles, CA
- B.S., Physics — Jun., 1997
  National Chung-Hsing University, Taichung, Taiwan

Professional Experience

- Teaching Fellow — Jan. 2011 – May 2011
  Department of Electrical and Computer Engineering
  Boston University, Boston, MA
- Summer Internship — Aug. 2010 – Nov. 2011
  Ultra Mobility Group
  Intel Corporation, Folsom, CA
- Research Assistant — Sep. 2007 – Jul. 2010
  Computer Architecture and Automated Design Lab
  Boston University, Boston, MA
- Research Assistant — Jan. 2007 – Aug. 2007
  VLSI and Neural Net Systems Lab
  Boston University, Boston, MA
- Teaching Fellow — Sep. 2005 – Dec. 2006
  Department of Electrical and Computer Engineering
  Boston University, Boston, MA
- Integrated Circuit Design Engineer — Feb. 2001 – Aug. 2005
  Sun Microsystems, Sunnyvale, CA

Awards and Honors (Selected)

- **Summer Fellowship**, Intel Corporation, 2010.
- **Outstanding Paper Award**, 19th International Conference on Field Programmable Logic and Applications (FPL'09), 2009. *Selected from 70 full papers and 300 overall submissions.*
- **ECE Award for outstanding research**, Department of Electrical and Computer Engineering, Boston University, 2007.
- **Outstanding Graduate Teaching Fellow Award**, College of Engineering, Boston University, 2006.
- **Graduate Teaching Fellow of the Year Award**, Department of Electrical and Computer Engineering, Boston University, 2006.

Publications (Selected)

- **M. Chiu**, Md. Ashfaquzzaman Khan, and M.C. Herbordt, "Efficient Calculation of Pairwise Nonbonded Forces," *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'11),* 2011.

- **M. Chiu** and M.C. Herbordt, "Towards Production FPGA-Accelerated Molecular Dynamics: Progress and Challenges," *Proceedings of High Performance Reconfigurable Computing Technology and Applications (HPRCTA'10)*, November, 2010.

- **M. Chiu** and M.C. Herbordt, "Molecular Dynamics Simulations on High Performance Reconfigurable Computing Systems," *ACM Transactions on Reconfigurable Technology and Systems*, 3, 4, 23:1-37, 2010.

- B. Sukhwani, **M. Chiu**, Md. Ashfaquzzaman Khan and M.C. Herbordt, "Effective Floating Point Applications on FPGAs: Examples from Molecular Modeling," *Proceedings of High Performance Embedded Computing (HPEC'09),* 2009.

- **M. Chiu** and M.C. Herbordt, "Efficient Particle-Pair Filtering for Acceleration of Molecular Dynamics Simulation," *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'09)*, August 2009 (**Winner of the Outstanding Paper Award**).

- M.C. Herbordt, B. Sukhwani, **M. Chiu** and Md. Ashfaquzzaman Khan, "Production Floating Point Applications on FPGAs," *Proceedings of the Symposium on Application Accelerators in High Performance Computing (SAAHPC'09)*, July 2009.

- **M. Chiu**, M. C. Herbordt, and M. Langhammer, “Performance Potential of Molecular Dynamics Simulations on High Performance Reconfigurable Computing Systems,” *Proceedings of the International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA’08)*, at Supercomputing ’08, November 2008.

- M. C. Herbordt, Y. Gu, T. VanCourt, J. Model, B. Sukhwani, and **M. Chiu**, “Computing Models for FPGA-Based Accelerators,” *Computing in Science & Engineering*, 10, 6, 35-45, 2008.

- M.C. Herbordt, Y. Gu, T. VanCourt, J. Model, B. Sukhwani, and **M. Chiu**, “Computing Models for FPGA-Based Accelerators with Case Studies in Molecular Modeling,” *Proceedings of the Reconfigurable Systems Summer Institute (RSSI’08)*, July 2008.