An FPGA Implementation of the Ewald Direct Space and Lennard-Jones Compute Engines

By

David Chui

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

An FPGA Implementation of the Ewald Direct Space and Lennard-Jones Compute Engines

David Chui

Master of Applied Science, 2005
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Biomolecular simulations allow scientists to advance their understanding of biologically important molecules by providing a molecular picture of the structure and behavior of biological systems. In particular, molecular dynamics (MD) is an approach that uses classical mechanics to model the behavior of a molecular system using the Newtonian equations of motion. A biomolecular simulation using software could spend up to 99% of the total computation time in calculating the non-bonded interactions between particles. This is a significant bottleneck in attempting to achieve a long simulation to provide useful information of a biological system.

The primary motivations for this research are: (i) special-purpose computers for MD simulation have become an interesting application in recent years, and (ii) FPGA technology is becoming a viable alternative to ASIC technology to achieve high performance in system design at a low cost with a shorter development cycle. The main objective of this thesis is to design two types of FPGA-based compute engines for computing the non-bonded interactions: (i) Ewald Direct Space, and (ii) Lennard-Jones.

The results show that the hardware compute engines can achieve similar performance in arithmetic precision by using a combination of: (i) fixed-point arithmetic, (ii) function table lookup, and (iii) function interpolation, compared to computation that directly uses double precision floating point.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Paul Chow, for his continuous support, guidance, and inspirations throughout the development of my thesis. I would also like to thank Professor Régis Pomès for his invaluable suggestion and discussion in my work.

The Canadian Microelectronics Corporation has been very supportive for providing the hardware and software tools to facilitate the design implementation of this work. I am also very much appreciated the assistance from Eugenia Distefano and Peter Pereira for the computer support.

I would like to extend my special gratitude to Tat-Sum Lee who spent countless hours with me on graduate courses and our theses. I have also enjoyed the time with Chris Comis and Chris Madill for working and sharing ideas together.

Finally I would like to dedicate this thesis to my parents and Yan Kiu Chan for their love, care, and support during my graduate study.

Table of Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Biomolecular simulations provide a molecular picture of the structure and behavior of biological systems that allow scientists to advance their understanding of biologically important molecules, where the same level of information cannot be extracted from laboratory experiments. In particular, molecular dynamics (MD) is an approach that uses classical mechanics to model the behavior of a molecular system over time by integrating the Newtonian equations of motion [1].

Biological systems of interest such as enzymes, proteins, DNA strands, and membranes in the presence of a solvent have sizes ranging from a few tens of thousands to millions of atoms [2]. Performing MD simulation, such as the protein folding process described in the IBM Blue Gene project [3], involves an enormous amount of computational effort. An estimation of the computational effort required to study a typical protein system is provided in Table 1.1 [3]. This leads to an estimate of three years to simulate 100 microseconds for a petaflop/second capacity machine.

Physical time for simulation	10^{-4} seconds
Typical time-step size	10^{-15} seconds
Number of MD time-steps	10^{11}
Atoms in a typical protein and water simulation	32000
Approximate number of interactions in force calculation	10^{9}
Machine instructions per force calculation	1000
Total number of machine instructions	10^{23}

Table 1.1 Estimated simulation effort of a typical protein system

The grand challenge in the MD simulation of large scale biomolecular systems is to improve the computational speed, in particular, the non-bonded particle interaction that accounts for 99% of the total computation time. A large amount of research effort has been put into this area of study in recent years involving the biochemists and computer scientists. The ultimate goal is to design a MD simulation system that allows researchers to simulate a biomolecular system within an achievable time frame and obtain useful information of a biological system.

This thesis investigates some hardware structures for computing the more time-consuming parts of molecular simulation.

1.1 Related Work

The purpose of this section is to provide an overview of some of the research that has been done in recent years. The main target of these research projects is to speed up the most computation-intensive portion, the non-bonded interaction, in the MD simulation.

The general approach is to use: (i) multiple processors, or (ii) application specific hardware engines.

The MD-GRAPE system was developed by the University of Tokyo in 1996 [4, 5]. It was one of the early developments of specialized hardware. The project was based on the original GRAPE-2A chip that was used for *N*-body simulations in the field of theoretical cosmology. It was used to study various structures in the universe such as galaxies based on the Ewald [6] and P^3M [7] methods to calculate the gravitational force under the periodic boundary condition.

An MD-GRAPE board contains four copies of an MD chip implemented as an ASIC, and each chip contains a pipeline of the GRAPE-2A design. The MD-GRAPE board has a peak speed of 4.2 Gflops and is roughly 30 times faster than a single GRAPE-2A chip, which has a peak speed of 180 Mflops. The overall system consists of two components: (i) the MD-GRAPE board, and (ii) the host computer, that are connected to each other via a VME bus. The MD-GRAPE board has a VME bus interface unit to fetch data from the host computer and send data to the local memory unit. The particle data is then supplied to the four MD chips that calculate the potential energy and force components, and send the result back to the host computer. The MD chip uses a fixed-point format to store particle positions and the computed result, while the internal calculations are performed in 32-bit floating-point format. The system achieves speed up by offloading the most expensive part of the MD simulation to the specialized hardware, while the less expensive part, such as the time integration of particles, is performed by the host computer. The MD chip is designed to support two kinds of algorithms, Ewald and P^3M, for MD simulation.

The development of the MD Engine in 1997 [8] was also a special-purpose computer for MD simulation that involved a jointed effort of universities and industrial partners. The general system architecture of the MD Engine system is similar to that described in the MD-GRAPE system, in which the host computer is communicating with the special-purpose parallel machine that computes the non-bonded interactions in an MD simulation.

An MD Engine system consists of 76 individual processors, called MODEL, and each is implemented to compute the Lennard-Jones interaction and Ewald summation. The internal arithmetic precision is determined in accordance to the findings by Amisaki [9]. Two types of floating-point numbers are employed inside the MODEL chip. The two are slightly modified versions of the IEEE standard 754 [10]. The calculation of a pair-wise force in the MODEL chip is implemented based on table lookup followed by quadratic interpolation.

A sample simulation that involves 13,258 particles is carried out using the MD software AMBER4.1. The result shows that the simulation takes several tens of seconds to advance a time-step on a Sun Ultra-2 200 MHz machine. Most importantly, 99.8% of the simulation time was spent in computing the non-bonded interaction. The MD Engine system can perform the simulation 50 times faster than that simulated by software on the

Sun machine. Furthermore, the architecture of the MD Engine system is highly scalable allowing further acceleration with additional processors.

In 1999, IBM announced plans for the BlueGene/L Supercomputer to study the area of life science [11]. The supercomputer is targeted to operate at a peak processing power of 360 Tflops based on a massively parallel system of 65,536 nodes. Each node consists of an ASIC that is based on the PowerPC, embedded DRAM, and system-on-chip techniques that allow for integration of system functions such as cache memory and multiple high-speed interconnection networks. The overall system is designed based on the torus network topology. The project was completed in 2004 and is ranked as the fastest supercomputer [12].

The MD-GRAPE team from RIKEN, Japan's Research Institute of Physical and Chemical Research, also developed the MD-GRAPE3 special-purpose processor in 2004 [13]. The ASIC chip, running at 350 MHz, can perform 230 Gflops, and the previous version, MD-GRAPE2, running at 100 MHz that can perform 16 Gflops. Both generations of design achieve substantial improvement in performance compared to the initial MD-GRAPE project by implementing multiple pipelines.

All of the aforementioned research projects of MD simulation are implemented using ASIC technology. The hardware development could take up to several years before the application can be fully implemented. Table 1.2 shows the ASIC technology used for fabricating the different chips. With the explosive development of reconfigurable technologies, such as FPGAs, in recent years, it is possible to achieve the same hardware functionality with a substantial performance in speed. Recent academic research has attempted to implement special-purpose computers using FPGAs, which allows system prototypes to be designed with a shorter development cycle and with much less cost.

Research Project	ASIC Technology
MD-GRAPE	0.6μm
MD-Engine	0.8μm
BlueGene/L	0.13μm
MD-GRAPE3	0.13μm

Table 1.2 ASIC technology used for fabricating MD special-purpose hardware chips

A reconfigurable MD simulator [14] has been implemented on an FPGA system, the Transmogrifier 3 [15], to compute the Lennard-Jones non-bonded interaction on a system consisting of 8,192 particles and performing Verlet time integration [1]. The arithmetic core is implemented using fixed-point data types, table lookup, and function interpolation. The design achieves a speed of 26 MHz and performs one time-step of computation in 37 seconds, while the same simulation run entirely in software on a 2.4 GHz P4 takes 10.8 seconds. The system is highly scalable and parallelizable. It is estimated that a hardware system speedup of over 20 times (0.5 seconds per time-step) can be achieved compared to a state-of-the-art microprocessor by implementing the design on the latest reconfigurable hardware parts along with a better memory system, while maintaining the computational precision required.

Another configurable MD simulator [16] was recently reported using the Xilinx Virtex-II Pro XC2VP70 FPGA. The hardware implements both the Coulombic and Lennard-Jones interactions. The study extends the one-dimensional system as described in Amisaki [9] into a three-dimensional system. The computation is done using fixed-point arithmetic and a combination of table lookup and interpolation. The work has targeted two areas of interest: (i) analyzing the stability of the simulation by varying the precision of the data-path in the design, and (ii) analyzing the performance speedup by using multiple pipelines to process data simultaneously.

The RMS fluctuation of the total energy is used as a quality measure of the simulation. It is observed that the design uses precision that is less than that of double precision floating point, but has a low energy fluctuation similar to a full 53-bit datapath (double precision floating point). This finding suggests that double precision floating point provides more precision than is needed for the purpose of this MD simulation. The work also shows that the simulation can achieve a speedup of up to 88 times over the software by implementing eight pipelines with lower data-path precision, while the same simulation performed in software requires 9.5 second per time-step on a 2.4 GHz Xeon PC.

1.2 Objectives and Organization of Thesis

The primary motivations for this research are: (i) special-purpose computers for MD simulation have become an interesting application in recent years, and (ii) FPGA technology is becoming a viable alternative to ASIC technology to achieve high performance in system design at a low cost with a shorter development cycle.

The main objectives of this work are: (i) to implement the compute engines on FPGA hardware that calculate the non-bonded interactions in an MD simulation, (ii) to explore the hardware resources requirements and system architecture, (iii) to study the trade-offs between hardware resources and the computational precision, (iv) to analyze the hardware pipeline performance, (v) to analyze the arithmetic precision using a combination of fixed-point arithmetic and function lookup, and (vi) to investigate what is the minimum hardware needed to achieve the same accuracy as the computation that uses double precision floating point.

The remainder of this thesis is organized as follows:

- Chapter 2 provides the background for Molecular Dynamics in the area of non-bonded interactions: Lennard-Jones and Ewald summation.
- Chapter 3 provides the basic concept of MD simulation using multi-processors.
- Chapter 4 presents the design implementation details of the two hardware pipeline engines and the overall system architecture.
- Chapter 5 presents the error analysis and the computational precision of the hardware pipeline engines.
- Chapter 6 discusses the hardware and system performance.
- Chapter 7 gives a summary of the thesis and avenues for future research.

Chapter 2

MOLECULAR DYNAMICS BACKGROUND

This chapter provides a brief overview of molecular dynamics. It starts by describing the basic physical approach and focuses on the non-bonded interactions, which are the most computationally intensive part in MD simulation.

2.1 Interaction of Atoms

Molecular dynamics is a model for describing the structure and properties of a molecular system [1]. Each atom in the system is described as a spherical rigid body with a known position in space. An electrostatic charge quantity is also assigned to each atom and is considered as a point charge located at the center of the sphere. With these data available, simulation is able to apply the classical Newtonian approach to calculate the interactions between atoms and extract parameters such as kinetic energy, potential energy, temperature, and pressure for the system.

Atoms in a system are subjected to two categories of interaction: intramolecular and intermolecular. They are also referred to as bonded and non-bonded interactions respectively. The force of attraction that holds an individual molecule together, such as ionic and covalent bonding, is intramolecular. The bondings can be represented in classical mechanics models as angle bending, torsion, and bond stretching. In a molecular simulation, these bonded interactions must be pre-assigned to specific sets of atoms according to the structures of the molecules. On the other hand, intermolecular interaction exists between one molecule and a neighboring molecule, or between a pair of atoms that are at least greater than two bonds apart. Such atom pairs interact with each other through the Van der Waals and electrostatic attraction and repulsion. These two types of non-bonded interactions are the main focus of this work and their physical properties will be discussed with more detail in the following sections in this chapter.

In computational complexity, the number of bonded interactions is linearly proportional to the number of atoms in the system. Hence, in a system with N atoms, the complexity in calculating the bonded pair is only on the order of $O(N)$, which is not particularly of interest since this is only a small part of the overall computation. On the other hand, the non-bonded interaction is the simulation bottleneck as the amount of computation grows substantially with respect to the number of atoms in the system because it is a pair-wise interaction that is $O(N^2)$.

2.2 Energy and Force

Fundamental physics shows that a closed system has a constant total energy that is comprised of kinetic energy and potential energy. The kinetic energy is due to the motion of all the atoms within the system. As the particles are approaching and departing from each other, the total potential energy in the system is changed. The energy of each atom constantly changes between kinetic energy and potential energy.

The Van der Waals and electrostatic interaction between a pair of non-bonded particles possesses potential energy between them that has the ability to cause attraction and repulsion as a result of the force field acting on each particle. Given that the potential energy expression, $\phi(r)$, is available, the force expression can be derived according to Equation 2.1, where ∇_r represents the spatial derivative.

$$F = -\nabla_r \phi(r) \tag{2.1}$$

A positive potential energy between a pair of particles will lead to a repulsive force, and a negative potential energy will cause attraction to happen. The actions of attraction and repulsion as a result of the various forces, such as Van der Waals and electrostatic, are the principle behind molecular dynamics simulations.

2.3 Van der Waals and Lennard-Jones Potential Energy

In an average situation, an electron cloud is symmetrically distributed around a non-polar molecule such as hydrogen (H_2) and methane (CH_3), so that there does not exist any electrical distortion to produce positive or negative parts within the molecule. The lozenge-shaped diagram in Figure 2.1 shows a small symmetrical molecule such as H_2. The even shading represents an electron cloud that is evenly distributed across the molecule.

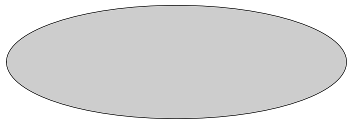


Figure 2.1 Lozenge-shaped diagram of a non-polar neutral molecule

However, quantum mechanics demonstrates that electrons exist on a probabilistic basis in space. Since the electrons are mobile, there is a chance that the electron cloud density will not be evenly distributed throughout the molecule at a given time. At any one instant the electrons might find themselves toward one end of the molecule, therefore making that end becomes slightly negative charged with value δ_-. The other end will be temporarily short of electrons and becomes charged by δ_+. Figure 2.2 shows that an uneven distribution of electrons occurs around a non-polar molecule, causing an instantaneous dipole to exist.

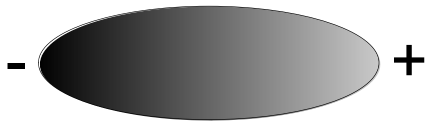


Figure 2.2 Lozenge-shaped diagram of a temporary dipole molecule

When this temporary dipole approaches another non-polar molecule nearby, the electron cloud density of this second molecule can also be redistributed. As a result, the electrons of the second molecule will gather on the side that faces a positive charge and retreat from a negative charge. Figure 2.3 shows that the instantaneous dipole on the left hand side is approached by another non-polar molecule, and the electrons will tend to be attracted by the slightly positive end of the left hand one. This process causes the electron cloud to redistribute itself and sets up an induced dipole in the approaching molecule, which is oriented in such a way that the δ_+ end of one molecule is attracted to the δ_- end of the other. Moreover, an existing polar molecule that is already a permanent dipole could also trigger this dipole induction process on a non-polar molecule nearby.

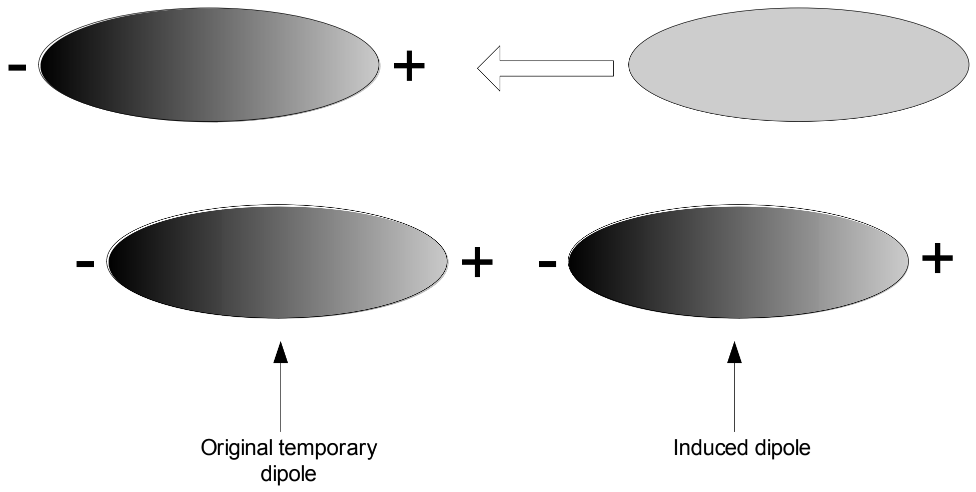


Figure 2.3 Lozenge-shaped diagram of the dipole induction process

This dipole-dipole interaction is called the Van der Waals force which is also known as the dispersion force. The Van der Waals force is a much weaker intermolecular force than others such as the ionic interaction, hydrogen bonding, and permanent dipole-dipole interactions. That is because the existence of Van der Waals force is based on probability, it only lasts for an instant of time, and most importantly, it is a short-range interaction. However, this phenomenon is the only intermolecular force present between non-polar species such as helium, nitrogen, or methane as a few examples. Without the Van der Waals force, there would be no attractive force between these molecules and they could not be obtained in liquid form.

Moreover, the Van der Waals force becomes stronger as the particle becomes larger in size. This is because there is a larger probability of creating a dispersed electron cloud, and therefore, more likely to cause the instantaneous dipole-dipole interaction. This trend can be seen from the halogens group in the periodic table where fluorine and chlorine are gases at room temperature, bromine is a liquid, and iodine is a solid. From the boiling points of the noble gases, a similar trend can also be seen as a result of the dispersion force. Table 2.1 shows the boiling points of the noble gases.

Element	Boiling Point
Helium	-269 ^oC
Neon	-246 ^oC
Argon	-186 ^oC
Krypton	-152 ^oC
Xenon	-108 ^oC
Radon	-62 ^oC

Table 2.1 Boiling point of noble gases

The reason that the boiling points increase as one go down the group is that the number of electrons increases, and so does the radius of the atom. The more electrons it has and the more space over which they can move, the larger the possible temporary dipoles can be created. Hence, it will take more heat energy to overcome the dispersion force in order to evaporate the noble gases from liquid to gas state.

On the other hand, there is a strong repulsion force between particles as they approach too close to each other. This can be explained by the fact that there is a mutual deformation of the structures of the particles as they are attempting to diffuse into each other that causes the overlapping of the electron clouds.

The overall potential resulting from these attractive and repulsive interactions as described above is called the Lennard-Jones potential energy. This phenomenon can be described by Equations 2.2 and 2.3, which are the potential and force expressions respectively.

$$U_{LJ} = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{2.2}$$

$$F_{LJ} = 24\varepsilon\frac{1}{r}\left[2\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{2.3}$$

Note that the Lennard-Jones equation is independent of the net charge, therefore, the potential energy exists between any types of particles. The ε and σ are the Lennard-Jones parameters that are determined by the types of interacting particles. The variable r is the separation distance r between the pair of particles. When the separation distance r is very small, the $1/r^{12}$ term dominates, and the potential is strongly positive. Therefore, the $1/r^{12}$ term represents the short-range repulsive potential due to the distortion of the electron clouds at small separation. In contrast, the $1/r^{6}$ term dominates when the value of r increases. Hence, the $-1/r^{6}$ term describes the mid-range to long-range attractive tail of the potential between two particles.

The Lennard-Jones potential energy equation for liquid argon is plotted in Figure 2.4. The potential-well of depth ε indicates that the strongest attractive force is at the distance of $r \approx \sigma$. As the separation distance increases, the strength decays at a rate of $-1/r^{6}$ that becomes negligible at a far distance. The depth of the potential-well in the plot is expressed in units of temperature ε/k_B, where k_B is the Boltzmann's constant. For liquid argon, the values of $\varepsilon/k_B \approx 120$K and $\sigma \approx 0.34$nm provide a reasonable agreement with the experimental properties.

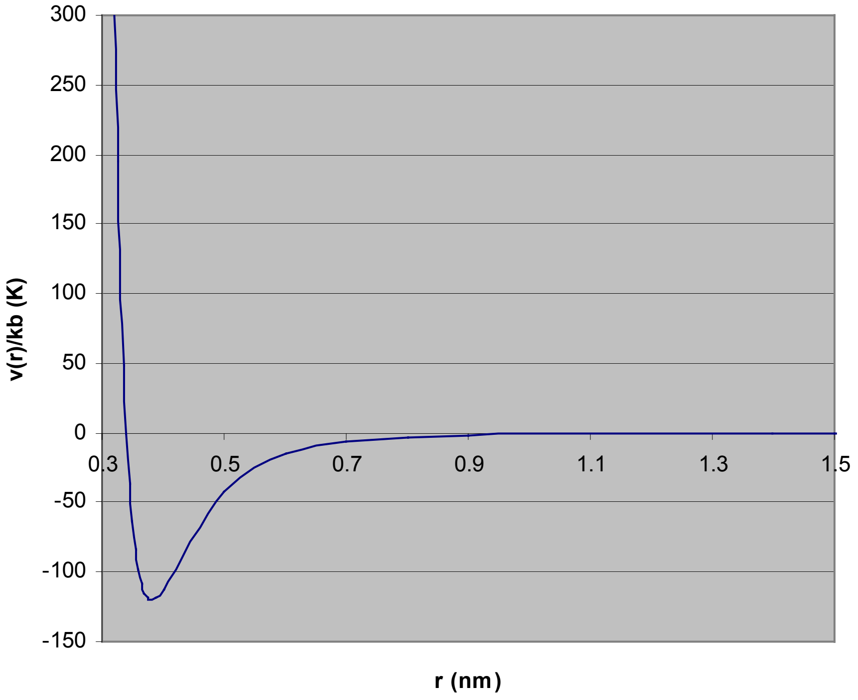


Figure 2.4 Lennard-Jones potential energy of liquid argon

The computational complexity of the Lennard-Jones interaction is $O(N^2)$ since the interaction is a pair-wise interaction within the simulation box. To reduce the computational effort, a cut-off radius is employed in simulation so that only the atoms within the cut-off radius are considered. As discussed earlier, the potential attractive tail decays at a rate of $-1/r^6$ as the separation distance increases. Using a typical cut-off radius from 12Å to 16Å [17] for the Lennard-Jones simulations will have negligible impact on the final computed energy while reducing the computational complexity closer to $O(N)$. This point will be revisited in Section 2.6, which discusses truncation of the potential field.

2.4 Electrostatic Potential Energy

The electrostatic force of attraction and repulsion is a strong intermolecular interaction between two non-bonded particles that have a net charge. In a system of particles, atoms bond with another to create ionic and covalent bonding. In ionic bonding, electrons are being transferred from one particle to another to create ions in a solution. Covalent bonding causes electrons to be shared between a pair of atoms within the molecule, leaving the electron cloud to be unevenly distributed around a non-symmetrical molecule. As a result, a net positive or negative charge exists on the ends of a molecule to create a permanent dipole.

For example, a water molecule consists of one oxygen atom and two hydrogen atoms. The covalent bonds of the structure H – O – H fill up the two vacancies in the oxygen electron structure to complete the octet. The two unshared electron pairs of the oxygen are oriented outward, away from the hydrogen atoms, to form a tetrahedral structure. This unshared electron pair can interact with a nearly naked hydrogen nucleus from a nearby water molecule to form a hydrogen bond.

In a molecular dynamics simulation, each atom in the system is assigned with a net charge quantity according to the molecular model that is determined by the shape of the molecule, the type of atom, and the type of bonding. The amount of charge assigned to each atom can be a partial charge quantity even though charge is quantized and the charge is assumed to be located at the center of the atom as a rigid spherical body. The overall charge assignment to all the atoms in a molecule is an equivalent model to represent the actual structure and property of the molecule such as the dipole moment.

The electrostatic potential energy and force are calculated using Equations 2.4 and 2.5, where the quantity ε is the permittivity constant.

$$U_{ij} = \frac{1}{4\pi\varepsilon}\frac{q_i q_j}{r} \tag{2.4}$$

$$F_{ij} = \frac{1}{4\pi\varepsilon}\frac{q_i q_j}{r^2} \tag{2.5}$$

The equations indicate that the electrostatic potential energy is a long range intermolecular force compared to the Lennard-Jones potential energy because the magnitude of the energy decays only at a rate or $-1/r$ as the distance increases. In contrast, the Lennard-Jones potential energy drops at a rate of $-1/r^6$.

The computational complexity of the electrostatic interaction is on the order of $O(N^2)$ since the interaction is also a pair-wise interaction within the simulation box. However, a similar cut-off radius cannot be directly applied to the computation of electrostatic interaction. That is because electrostatic potential energy is considered to be a long-range interaction that still has a significant impact on the total energy for atoms that are separated at a distance outside of the typical cut-off radius. Therefore, without applying a cut-off radius, the direct computation using Equations 2.4 and 2.5 will have a complexity of $O(N^2)$. The following sections will discuss an alternative methodology, Ewald summation [1, 2, 6, 18], that is used to compute the electrostatic interaction in such a way that the complexity is less than $O(N^2)$ by applying a cut-off radius.

2.5 Periodic Boundary Condition

The number of atoms in a computer simulation is usually constrained by the computation resources such as processor speed, storage, and communication overhead in a multi-processor environment. The nature of the long-range force of electrostatic potential energy requires simulation to calculate the potential energy and force for every pair of atoms in the simulation box. This default simulation methodology is executing a double loop in the program that has a computation complexity of $O(N^2)$. If the amount of computation is to be performed in a parallel fashion, the particles data will need to be duplicated in each of the processors and the communication between the host computer and the slave processors will also be increasing at a rate proportional to N^2. As a result of the limitation in the computation speed, memory consumption, and communication overhead, simulations are usually performed on small systems such that the simulation can be run enough time-steps within an achievable time. However, a small system is confined by a potential representing the container that prevents the atoms from drifting apart. The problem in such simulation is that a large percentage of the atoms are located near the surface of the simulation box. These atoms near the faces of the cube will experience a different force from those within the bulk of molecules. A periodic boundary condition is an important technique employed in molecular dynamics simulations that overcomes the problem of surface effects [1]. It is a methodology to make a simulation that consists of only thousands of atoms behaves as if it is in a system of infinite in size.

Figure 2.5, adopted from [1], illustrates the concept of periodic boundary condition [1, 2]. The simulation box in the center represents the system being simulated, while the surrounding boxes are the exact copies in every detail of the original box, that is, every particle in the simulation box has an exact duplicate in each of the surrounding cells including position and velocities. This methodology removes the surface effect since there are no walls at the boundary of the central box as the center cell is replicated throughout space to form an infinite lattice.

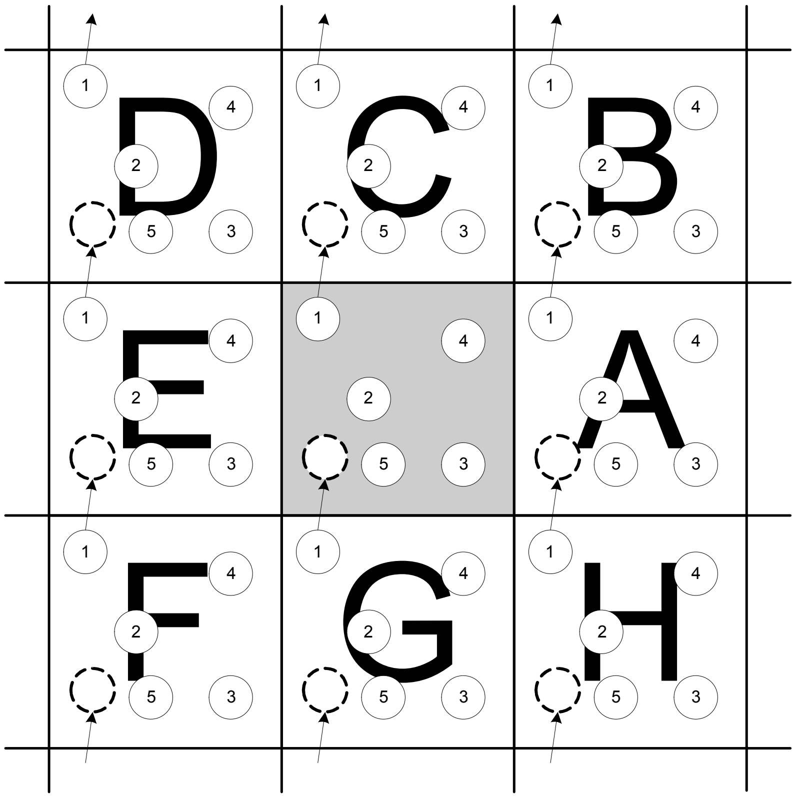


Figure 2.5 Two dimensional periodic system

During the simulation, as an atom leaves the simulation cell, it is replaced by another one with exactly the same velocity entering from the opposite cell face. Hence, the number of atoms, momentum, and energy are conserved. By referring to the two dimensional periodic system in Figure 2.5, as particle 1 moves through a boundary, its image in each of the replicated cells, 1_A, 1_B, etc., travel across the corresponding boundaries. During the program execution, it is not necessary to store the position and velocity of every atom in all the replicated boxes. It is only necessary to keep track of the particles within the original simulation box and reassign the particle position as a particle crosses the boundary. Figure 2.5 shows the infinite lattice in a two dimensional case. A real simulation in molecular dynamics is applying the same principle to a three dimensional lattice that extends to infinity in the x, y, and z directions.

2.6 Potential Truncation

The majority of the program execution time in a simulation is spent on the pair-wise computation of the potential energy and force within the system. For a particular atom in the simulation box under the periodic boundary condition, the potential acting on that atom is not only due to the other N-1 atoms within the current simulation box, but also the interactions that result from the rest of the particles in the infinite lattice. This is physically impossible to compute since the periodic boundary condition consists of an infinite number of images, hence, there are an infinite number of terms to sum up. Therefore, the overall computation complexity under the periodic boundary condition has become $O(N^2+)$. For a short-range intermolecular interaction, it is desirable to apply some methods of truncation to restrict this summation by making an approximation without introducing an error that is beyond an acceptable level of tolerance.

Consider the periodic boundary condition in Figure 2.6, adopted from [1]. Particle 1 is located at the center of the virtual simulation box where it is interacting with the rest of the other N-1 particles. This virtual square region has the same size and shape as the basic simulation box and this method is referred as the 'minimum image convention' [1]. For example, particle 1 interacts with particle 2 within the original simulation box, 3_D, 4_E, and 5_C from the surrounding images. The application of the 'minimum image convention' as a method of truncation simplifies the problem from summing up an infinite number of terms in the periodic boundary condition into a problem of summing up the pair-wise interactions within the virtual region, which essentially computes the potential for all the particles in the system. The problem is still challenging since the complexity of the computation remains $O(N^2)$.

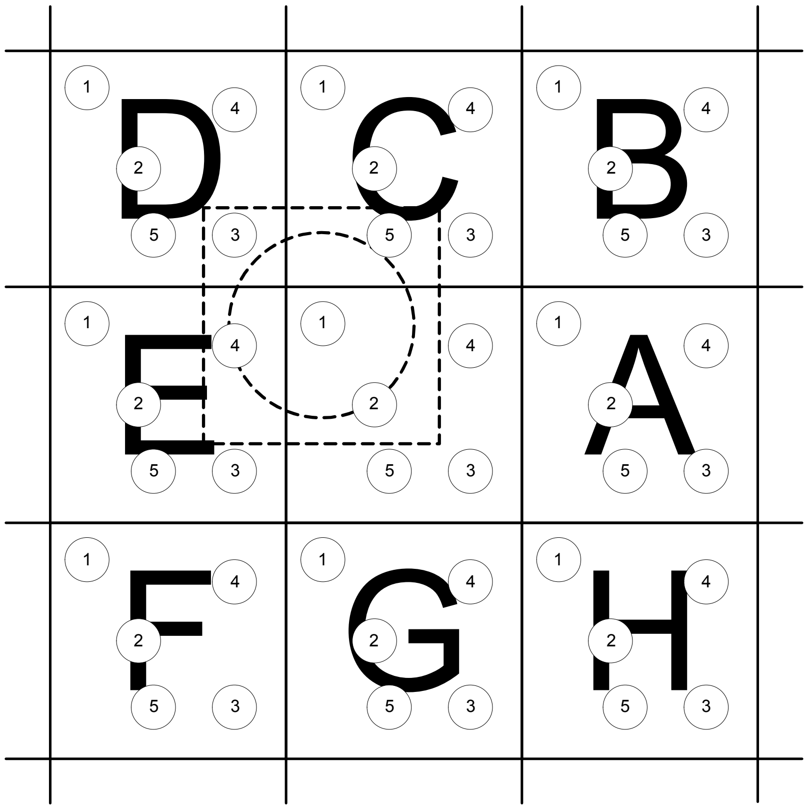


Figure 2.6 Minimum image convention in a two dimensional periodic system

It is desirable to apply further approximation to significantly reduce the computation. The largest contribution to the potential energy and force comes from neighbours that are close to the atom of interest. It also makes physical sense to apply a spherical cut-off radius instead of a cubic 'minimum image convention' as those particles near the corner of the cube are further away from the center. The question is what factors determine the cut-off radius? The spherical cut-off radius can be pre-determined before the simulation takes place. For example, a simulation that only involves Lennard-Jones potential energy, such as the liquid argon system in Figure 2.4, selects a cut-off radius of $r_c=2.5\sigma$. The potential energy is roughly 1.6% of the potential-well depth and the potential energy is considered to be zero beyond this cut-off radius. This cut-off radius is selected based on the specific Lennard-Jones potential function, as well as the acceptable level of error tolerance of the simulation, that is 1.6% of the potential-well depth in this liquid argon example. Moreover, the cut-off radius must not be larger than $L/2$, where L is the simulation box width, for consistency with the minimum image convention. If the cut-off radius is selected to be larger than $L/2$, then the sphere is covering a diameter that is greater than L. In such case, it is possible that a particular particle is considered twice during the computation as a result of the replicated image.

Note that by applying the spherical cut-off radius as discussed above, the simulation performance would not be improved much if the sphere is selected to be just inscribing the cube. This is because a high percentage of the area (or volume in the three dimensional case) is still being covered, hence, the complexity remains $O(N^2)$. The main goal of applying a spherical cut-off is to select a radius that is much smaller than the box size L, such that only a small region within the cube is considered.

The Lennard-Jones potential energy computation can leverage from the use of a small cut-off radius because it is a weak intermolecular interaction. In a typical molecular dynamics simulation, the simulation box size is usually in the range from 64Å to 128Å [17], while the cut-off radius is normally selected to be from 12Å to 16Å as mentioned in Section 2.3. It can be seen that the cut-off radius employed by Lennard-Jones simulation is much less than the $L/2$ restriction imposed by the minimum image convention. Such a cut-off radius would only cover a small region relative to the entire cube, hence, only a small number of particles in the surrounding space will be interacting with the particle at the center of the cut-off sphere. Therefore, the computational complexity of Lennard-Jones can approach $O(N)$ by applying a cut-off radius.

To provide a sense of how well the spherical truncation scheme works, the previous example of the liquid argon system can give some numerical insight. By applying the experimental values of $\varepsilon/k_B \approx 120\text{K}$ and $\sigma \approx 0.34\text{nm}$, the Lennard-Jones potential energy is roughly 0.21% (at a distance of 12Å) and 0.037% (at a distance of 16Å) relative to the potential-well depth. The magnitude of the potential-well depth represents the maximum attractive force of that particular Lennard-Jones function and the percentage obtained above is the error tolerance.

Another issue to be emphasized is that the periodic boundary condition is in fact not a critical concern in Lennard-Jones simulation. Since the Lennard-Jones function tail decays at a rate of $1/r^6$, applying a cut-off radius that is much smaller than the box size, $r_c << L$, already yields a very small percentage error given that the computational complexity approaching $O(N)$ is achieved. It is then not necessary to consider the replicated image beyond the virtual simulation box boundary as shown in Figure 2.6.

The electrostatic potential energy is a relatively strong intermolecular interaction. Neither the application of the spherical cut-off, nor the use of minimum image convention would satisfy a desired level of error tolerance. Since the electrostatic potential energy function only decays at a rate of $1/r$, the magnitude of the potential energy remains significant at a distance that is beyond the minimum image. Therefore, the total energy and force of the system will experience a much larger error that will eventually affect the stability of the system energy and the physical structure of the molecular system if one of the previous truncation schemes is applied.

The next section will discuss the Ewald summation, which is an alternative computation methodology for calculating the equivalent electrostatic potential energy under the periodic boundary condition by decomposing the energy into a short-range and a long-range interaction.

2.7 Standard Ewald Summation

The previous section introduces the use of the periodic boundary condition that replicates the simulation box into an infinite lattice to minimize the surface effect. The potential truncation methods are employed to improve the computational efficiency by discarding interactions that are outside of a certain region. However, the use of truncation has been shown to introduce significant error and artificial behaviour in simulations involving electrostatic interactions. Ewald summation [1, 2, 6, 18] is a method for evaluating the electrostatic potential energy interaction under the periodic boundary condition efficiently, while maintaining the accuracy of the simulation result at a satisfactory level.

Under the periodic boundary condition, the total coulombic energy of a system with N particles in a simulation box of size L can be expressed by Equation 2.6.

$$U = \frac{1}{2}\sum_{n}{}' \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{q_i q_j}{r_{ij,n}} \tag{2.6}$$

In the infinite lattice of cells, the cell-coordinate is represented by the vector $\boldsymbol{n} = (n_1, n_2, n_3) = n_1L_x + n_2L_y + n_3L_z$. The origin cell is located at $\boldsymbol{n} = (0, 0, 0)$. Figure 2.7 illustrates the coordinate system of a two dimensional infinite lattice.

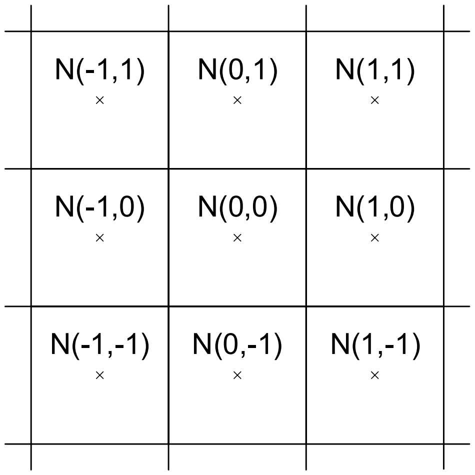


Figure 2.7 Coordinate system of a two dimensional periodic system

The double summation over *i* and *j* in Equation 2.6 represents the pair-wise interactions of the *N* particles within the original simulation box. The left-most summation is to represent the summation of potential energy across all the replicated cells as ***n*** goes to infinity. The first summation is primed to indicate that the terms with $i = j$ are omitted when ***n*** = (0, 0, 0). When ***n*** ≠ (0, 0, 0), particle *i* will interact with the rest of the particles, *j*, in the current cell, as well as interacting with all other particles in the replicated images including *i* itself in the replicated cells. The triple summation term is conditionally convergent [19, 20, 21], that is, the result will depend on the order of how the terms are added up. The technique employed by Ewald summation is to split up Equation 2.6, a single slowly and conditionally convergent series, into the sum of two rapidly converging series and a constant term as in Equation 2.7.

$$U^{Ewald} = U^r + U^m + U^o \tag{2.7}$$

The Ewald sum is then expressed in three parts, the real space energy (U^r), the reciprocal space energy (U^m), and the constant self-term energy (U^o). The break down of the total energy is to decompose the coulombic interaction into short-range (real-space) and the long-range (reciprocal space) interactions. The purpose of this break down is to enable the real space term to be calculated with the use of truncation since it is a short-range interaction, such that the potential is negligible outside of the origin simulation cell ***n*** = (0, 0, 0). The three terms are given by Equations 2.8, 2.9, and 2.10.

$$U^r = \frac{1}{2}\sum_n{}' \sum_{ij}^{N} q_i q_j \frac{erfc(\alpha r_{ij,n})}{r_{ij,n}} \tag{2.8}$$

$$U^{m} = \frac{1}{2\pi V}\sum_{ij}^{N} q_i q_j \sum_{m\neq 0} \frac{\exp(-(\pi m/\alpha)^2 + 2\pi i m\cdot(r_i - r_j))}{m^2} \tag{2.9}$$

$$U^{0} = \frac{-\alpha}{\sqrt{\pi}}\sum_{i=1}^{N} q_i^{\,2} \tag{2.10}$$

The function erfc() is the complimentary error function that decreases monotonically as *x* increases and is defined by Equation 2.11.

$$erfc(x) = 1 - erf(x) = 1 - \frac{2}{\sqrt{\pi}}\int_0^x e^{-u^2}\,du \tag{2.11}$$

The variable *V* is the volume of the simulation box and ***m*** = (*i*, *j*, *k*) is the reciprocal-space vector and ***n*** is defined earlier. The plot of the function erfc() is given in Figure 2.8. The graph shows that the function decays very fast at the beginning and the curve approaches zero as *x* goes to infinity.

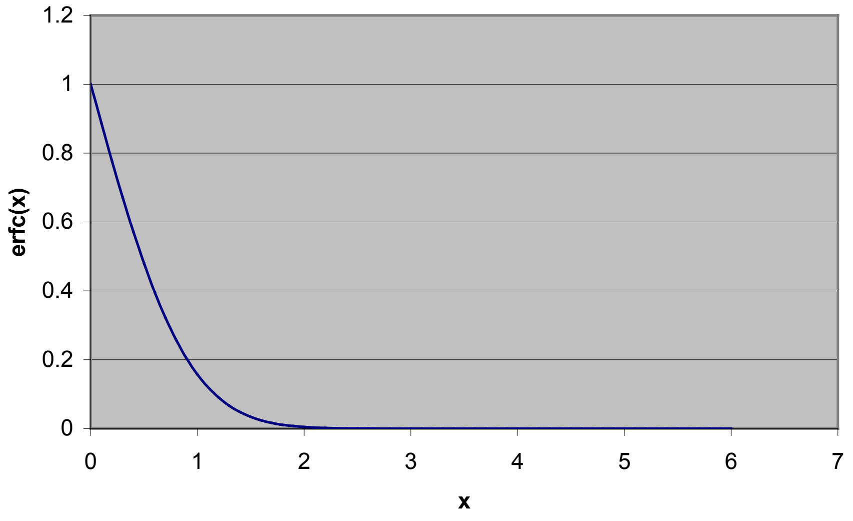


Figure 2.8 Complimentary error function

For the detailed derivation of these equations, interested readers should refer to the original derivation by P.P. Ewald [6], and other sources that discuss the convergence of infinite series [19, 20, 21] and a survey on Ewald summation [2]. However, a brief detail of the derivation of Equation 2.8 is given in Appendix A, and a further simplification of Equation 2.9 is given in Appendix B.

The physical interpretation of the Ewald decomposition is that each point charge in the system is viewed as being surrounded by an additional Gaussian charge distribution of equal magnitude and opposite sign [2]. The charge distribution is conveniently selected to be Gaussian and is given by Equation 2.12.

$$p_i(r) = q_i \alpha^3 \exp(-\alpha^2 r^2) / \sqrt{\pi^3} \qquad (2.12)$$

The variable α is a positive parameter that determines the width of the distribution and the variable r is the distance relative to the center of the distribution. Figure 2.9 shows the graphical representation of the Ewald decomposition [2].

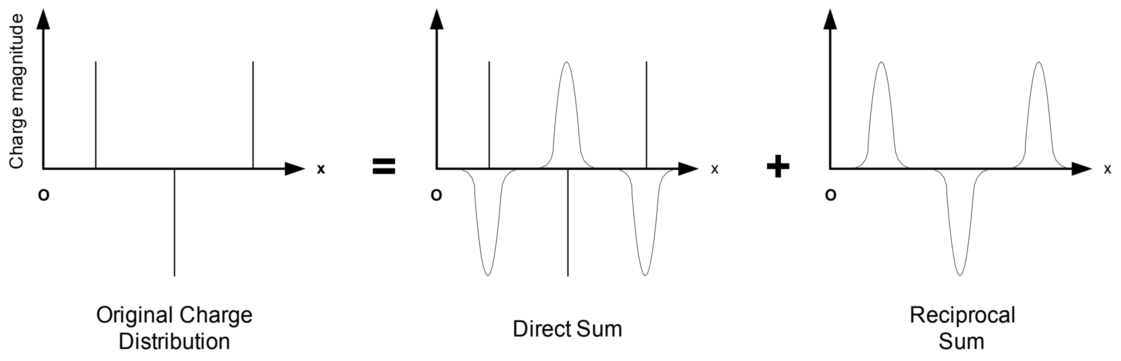


Figure 2.9 Ewald decomposition

For the Ewald direct sum, the original point charge and the additional Gaussian charge distribution of opposite magnitude can be viewed as cancelling each other at a far distance x from the point O because the Gaussian distribution looks narrow enough from a distance. The total potential energy acting on a particular point in space, for example the point O, will become effectively limited to short range. Consequently, the sum over all charges and their images in real space converges rapidly. By comparing with Equation 2.6, Equation 2.8 has an additional factor of erfc( ) that represents the interaction between the two charges, q_{i} and q_{j}. This diminishes to zero as the separation distance increases according to Figure 2.8. This corresponds to the fact that the point charge is cancelled out by the Gaussian distribution as viewed from a far distance in Figure 2.9. To counteract this induced Gaussian distribution, a second Gaussian charge distribution of the same sign and magnitude as the original distribution is added for each point charge. This time the sum is performed in the reciprocal space using Fourier transforms to solve the resulting Poisson's equation. The derivation of the reciprocal sum equation is not discussed here because it is beyond the scope of this thesis.

2.8 Particle Mesh Ewald

From Equations A.10 in Appendix A and B.7 in Appendix B, there are three parameters that determine the workload in the real space and the reciprocal space. The integer n_{max} determines the number of real space vectors $\boldsymbol{n}$ needed to have convergence of Equation A.10. The integer m_{max} determines the number of reciprocal space vectors $\boldsymbol{m}$ needed to have convergence of Equation B.7. The relative rate of convergence between the real and reciprocal sums is controlled by α. The Gaussian distribution becomes narrower with a large value of α causing the real-space sum to converge faster. That is, as $\alpha \to \infty$, the erfc(αx) $\rightarrow$ 0. Therefore, a small number of $\boldsymbol{n}$-vectors (small $n_{\max}$) is sufficient to give a rapid convergence in Equation A.10. On the other hand, a small α in Equation B.7 causes the reciprocal space sum to converge faster since as $\alpha \rightarrow 0$, the exp($-x/\alpha$) $\rightarrow$ 0. Hence, a small number of $\boldsymbol{m}$-vectors (small $m_{\max}$) are sufficient for Equation B.7 to rapidly converge.

It can be seen that by altering the value of α, the effect is to shift the workload between the real space and the reciprocal space. The number of $\boldsymbol{n}$-vectors or $\boldsymbol{m}$-vectors that are required to sum will depend on the desired level of convergence of the series. Note that the values of $n_{\max}$ and $m_{\max}$ represent the number of real-space images and reciprocal space images to be summed up respectively. In practice, the α is chosen so that the real space computation extends no further than the original cell, thus limiting the computational complexity of the direct sum to $O(N^2)$, while maintaining the reciprocal sum complexity at $O(N)$ in the standard Ewald summation [2].

The Particle Mesh Ewald (PME) method [22] is based on the Hockney and Eastwood particle-particle particle-mesh method (PPPM) [7]. PME splits the total potential energy using the same method as the standard Ewald, that is, by dividing the total energy into direct sum and reciprocal sum, and uses the conventional Gaussian charge distribution. In other words, Equations A.10 and B.7 can be reapplied here for PME. The difference between the standard Ewald and PME is that the Ewald parameter α is chosen large enough in PME such that a cut-off radius (that is much smaller than the size of the box L) can be applied to limit the number of pair-wise interactions in the system. This reduces the computational complexity of the direct sum from $O(N^2)$ closer to $O(N)$. To compensate for the truncation in evaluating the direct sum, the number of reciprocal vectors, $\boldsymbol{m}$, is increased proportionally to N. From Equation B.7, the computational complexity of the reciprocal sum becomes $O(N^2)$ as $m_{\max}$ grows at a rate of N. In PME, the reciprocal sum is then computed using a 3D-FFT such that the complexity becomes $N\log(N)$. Therefore, the overall computational complexity of PME is $N\log(N)$. For completeness, a brief summary of the PME equations of the reciprocal space is given in Appendix C.

2.9 Summary

This chapter has introduced the two types of intermolecular interactions: Lennard-Jones and electrostatic potential energy. Lennard-Jones is considered to be a short-range interaction as a result of the $-1/r^6$ rate of decay of the potential tail in Equation 2.2. The standard Ewald summation and its improved version, Particle Mesh Ewald, are used to split the long-range electrostatic interaction into the direct sum (short-range) and reciprocal sum (long-range).

Since the Lennard-Jones and the direct space of PME are considered to be short-range interactions, a spherical cut-off radius can be applied to both computations reducing the complexity to approach $O(N)$. In fact, molecular dynamics simulations, in general, will select a common cut-off radius such that these two short-range interactions can be computed together in the software program.

For the reciprocal space of PME, the 3D-FFT is applied to perform the computation with a complexity of $N\log(N)$. Therefore, the overall non-bonded calculation becomes $N\log(N)$ as compared to the original method of summing up the pair-wise interactions in the infinite lattice, which has a complexity of $O(N^2+)$.

Chapter 3

MULTI-PROCESSOR COMPUTATION

This chapter reviews a technique for parallelizing the Particle Mesh Ewald and Lennard-Jones algorithms. The approach allows the algorithms to be decomposed and executed on multiple processors to achieve speed up in molecular dynamics simulations.

3.1 Distributed Particle Mesh Ewald

Chapter 2 has reviewed the standard Ewald summation approach that computes the long-range electrostatic interaction under the periodic boundary condition. It has a computational complexity of $O(N^2)$ that converts a single slowly and conditionally convergent series, into the sum of two rapidly converging series: (i) the real space, and (ii) the reciprocal space. The standard Ewald summation is reformulated to become the Particle Mesh Ewald by explicitly using a cut-off radius for the direct space sum, while the reciprocal space sum is approximated via a convolution that is performed using the 3-D FFT. The computational complexity of this improved method is reduced to

$O(N\log(N))$ compared to the traditional method. The purpose of this section is to analyze the PME equations and review a method for parallelizing the algorithm based on the NAMD software [23, 24], such that it can be executed in a multi-processor environment.

In PME, the value of α is selected such that the condition expressed in Equation 3.1 is satisfied.

$$\frac{erfc(\alpha r_{cutoff})}{r_{cutoff}} = Tolerance \tag{3.1}$$

This fraction term is extracted from Equation 2.8, the Ewald Direct Space electrostatic potential energy function. This constraint means that with an appropriate choice of α, the interaction between the two particles drops to a factor of *Tolerance* at exactly the distance of r_{cutoff}. Figure 3.1 shows the relationship between these two parameters at an arbitrary value of α.

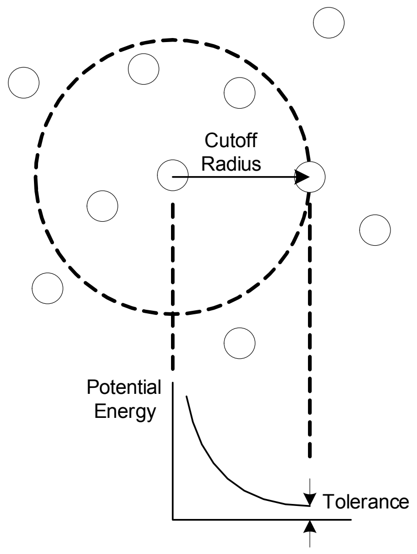


Figure 3.1 PME cut-off radius and tolerance

In a molecular dynamics simulation that adopts PME, the choice of these two parameters will affect the simulation accuracy and the computational load balancing. Figure 3.1 shows that the pair-wise interactions that are outside of the selected r_{cutoff} will be excluded from consideration and these excluded terms could have values as large as the defined *Tolerance*. Thus, as the simulation sums up the total system energy, the size of the error in the total energy directly depends on the choice of *Tolerance*, and subsequently affects the energy fluctuation and stability of the molecular system. The choice of r_{cutoff} will affect the amount of work that needs to be done on the direct space side, since a larger cut-off radius encapsulates more atoms within the sphere.

The workload split between the real space and reciprocal space are inversely related to each other. If a simulation chooses to decrease the workload on the real space side by selecting a small cut-off radius for a specific value of *Tolerance*, this requires the value of α to be increased, and subsequently causes the reciprocal space side to work harder to reach convergence. Therefore, the workload in the two spaces can be adjusted by selecting an appropriate value of α. In a molecular dynamics simulation, the typical value of *Tolerance* is between 10^{-6} to 10^{-10}, and the r_{cutoff} is between 10Å to 16Å [17].

Another important characteristic of the PME is that the direct space sum computation can be performed in parallel since the interactions are limited to be within the cut-off radius. Figure 3.2 shows that the simulation box is subdivided into a large set of cubic cells, and each one has a length slightly larger than the cut-off radius [7, 25]. Particles are sorted into these cells according to their coordinates. To evaluate the interaction, each particle calculates its interactions with other particles within its own cell, and all of the particles in the immediate neighboring cells. There would be no interactions that extend beyond one neighboring cell because the cell size is set to be slightly larger than the cut-off radius. This spatial decomposition approach allows particles that reside in a particular region in the simulation box to compute their interactions independently from others in a parallel fashion.

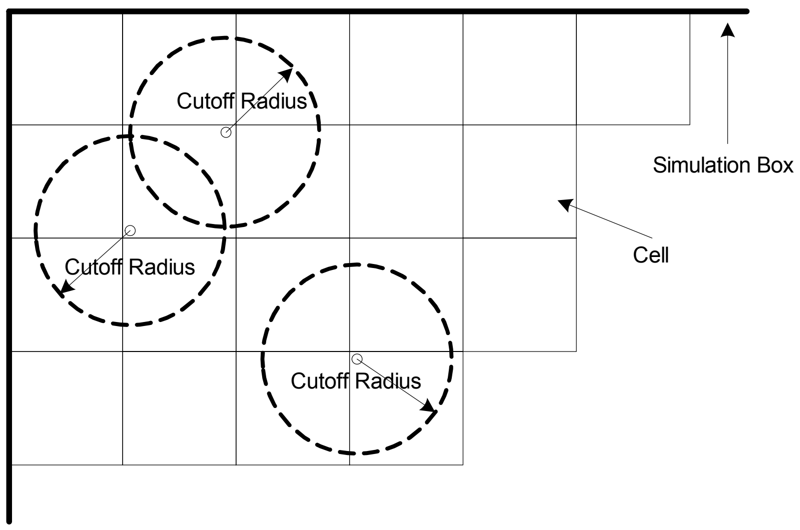


Figure 3.2 Spatial decomposition of PME direct space sum

The Distributed Particle Mesh Ewald, DPME [26], is introduced based on the two aforementioned characteristics of PME: (i) load balancing between the real space and reciprocal space, and (ii) spatial decomposition of the real space sum. The basic approach of performing DPME computation is to setup a master-slave environment. A host computer controls the execution flow of the program and is also responsible for computing the relatively inexpensive reciprocal sum. The host computer communicates with a set of slave processors that collectively evaluate the direct space sum and send the results back to the host computer.

The host computer has the capability to determine the Ewald coefficient α that affects the load sharing between the host computer itself and all the slave processors. In this spatial decomposition setup, each slave processor is supplied with a neighbor-set that is comprised of particle data of the cells that belong to a particular region in space. The master-slave approach is a generic way to parallelize algorithms such as DPME.

3.2 Lennard-Jones

As discussed in Section 2.3, the use of a cut-off radius is adequate for the computation of Lennard-Jones interaction since it is a short range non-bonded interaction. Therefore, the method of spatial decomposition as shown in Figure 3.2 can also be applied to the Lennard-Jones computation.

The next section will discuss the NAMD software, a molecular dynamics simulation tool that applies the spatial decomposition method to compute the non-bonded interactions, DPME and Lennard-Jones, in a multi-processor environment.

3.3 NAMD Software

By applying the PME approach with the use of a cut-off radius, the computation on the order of $O(N\log(N))$ would still represent a substantial amount of work if the software program is executed sequentially. NAMD is a high performance software program for large biomolecular system simulations that employs the DPME approach as described in the previous section. Its object-oriented structure, multi-threading, and the use of the prioritized message-driven execution capabilities of the Charm++/Converse [23] library allows NAMD to be executed efficiently on both massively parallel supercomputers and workstation clusters.

The particles in the simulation box are partitioned into cells according to the spatial decomposition method of DPME. Since the cells are setup to be slightly larger than the cut-off radius, only the interactions as a result of those particles in the immediate neighboring cells are to be examined. Figure 3.3 shows that each cell inside the simulation box is interacting with the immediate neighboring cells around it.

In this case, Cell 5 in the middle is interacting with all the neighboring cells in the eight different directions as joined by the arrows. To calculate the non-bonded interactions for the particles in Cell 5, NAMD assigns two types of compute objects: (i) to compute non-bonded interactions within Cell 5, and (ii) to compute non-bonded interactions between a pair of neighboring cells. A compute object in NAMD software is a single software thread that executes a piece of code.

Each cell in the simulation box has eight neighbors, however, for Cell 5 in this two dimensional example, it is only necessary to assign five (8/2 + 1) compute objects to compute the non-bonded interactions within Cell 5 (one compute object) and those interacting with Cells 2, 3, 6, and 9 (four compute objects). Interactions with the other neighbors will be considered when Cells 1, 4, 7, and 8 are in focus as indicated by the direction of the arrows. Based on the Newton's 3rd Law, it is not necessary to duplicate the compute object to calculate the interaction of particle *A* in one cell acting on particle

B in a different cell and then the reverse interaction. Therefore, in the three dimensional case, the total number of compute objects required is equal to fourteen (26/2 + 1 self-interaction) times the number of cells [23].

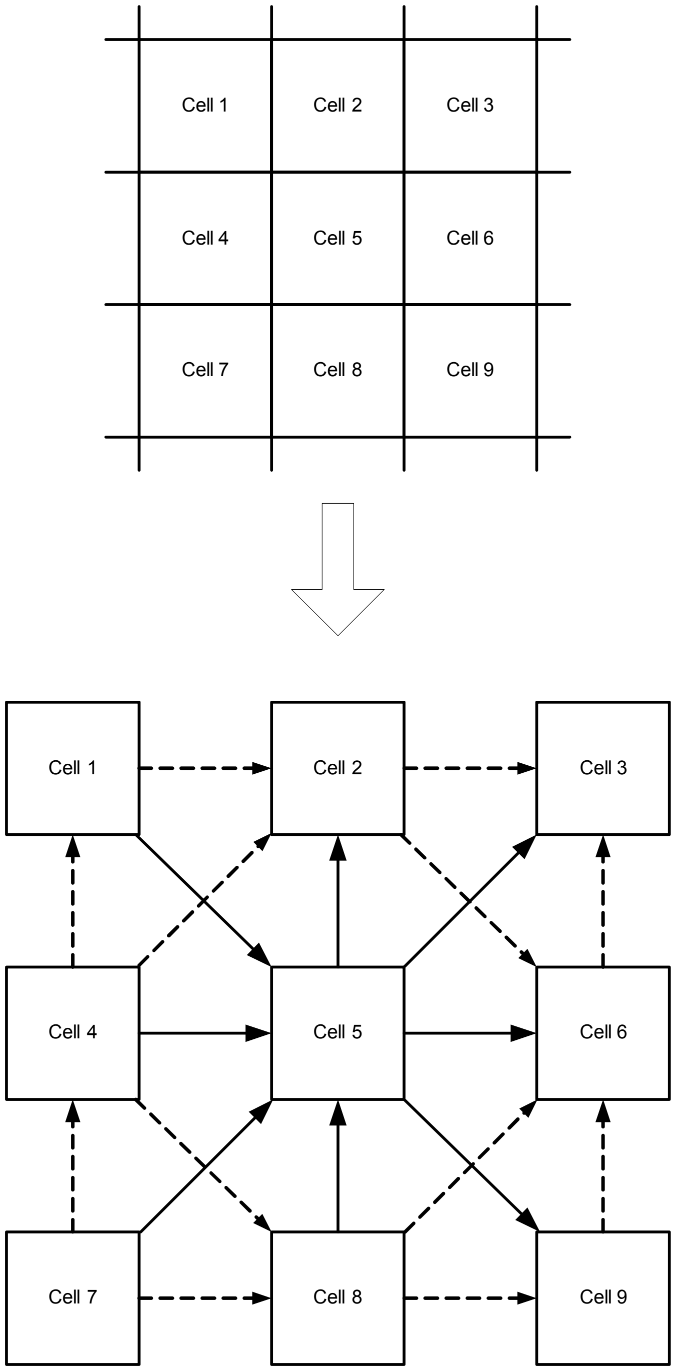


Figure 3.3 Cell interactions in spatial decomposition

The compute objects described above can be spawned off to individual processors available in the NAMD simulation since the computation can be performed independently from others. Each of the cells in Figure 3.3 is also represented in NAMD by an object called a home patch. The home patch is responsible for storing and distributing particle data, retrieving forces from the associated compute objects, and integrating the equations of motion for all the atoms owned by the patch. Figure 3.4 shows that the home patch of a particular cell is linked to its own compute objects, the self compute object and pair compute object, as described above. Figure 3.4 also demonstrates that for a compute object to access particle data from another cell, it has to go through a proxy that also exists as an object in the NAMD software.

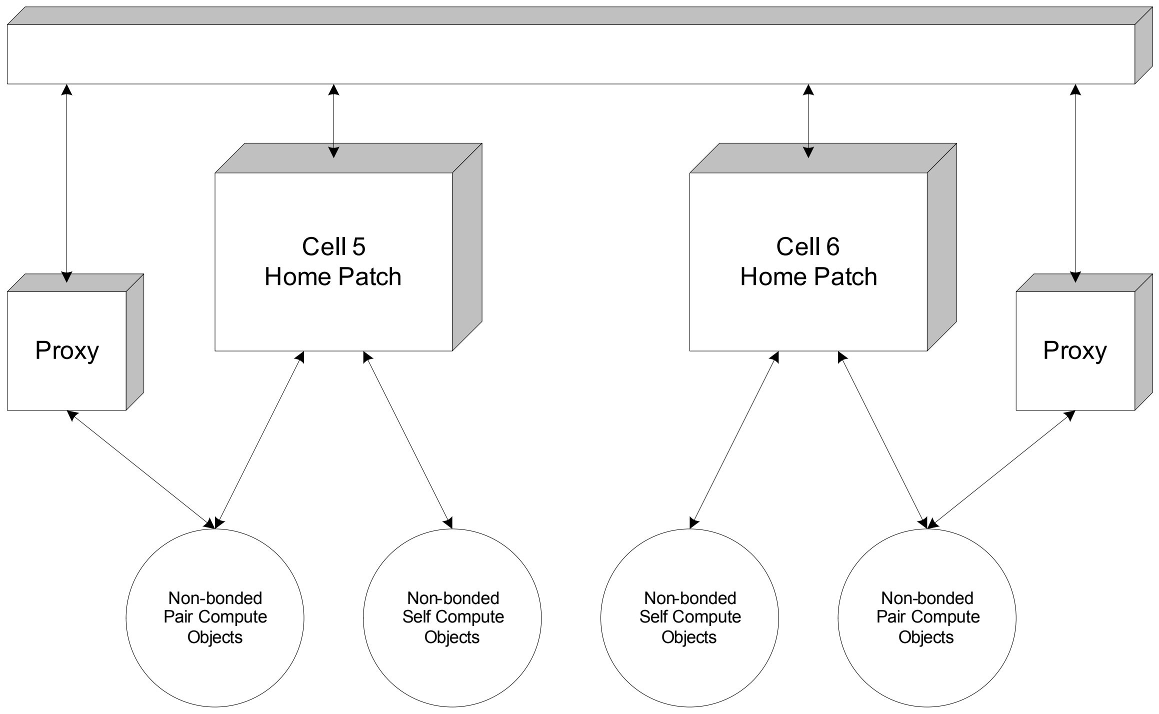

Cell 5
Home Patch
Cell 6
Home Patch
Proxy
Proxy
Non-bonded Pair Compute Objects
Non-bonded Self Compute Objects
Non-bonded Self Compute Objects
Non-bonded Pair Compute Objects

Figure 3.4 NAMD objects architecture

Since the method of spatial decomposition is used by both the Ewald Direct Space and Lennard-Jones computations, a common cut-off radius can be used in a simulation. In addition to that, their algorithms are implemented within the same compute object in the NAMD software.

The general software architecture of NAMD is to create different types of software objects: (i) home patch, (ii) compute object, and (iii) proxy. These objects represent the cell structure of the spatial decomposition, computation node, and communication interface respectively. All the objects exist as software threads and are mapped to different processors for execution. The underlying communication architecture is achieved by employing the Charm++/Converse library that facilitates the exchange of information, load balancing, as well as the scalability of the system. For more detailed description of the software architecture and multi-processor environment, readers should refer to [23].

3.4 Summary

This chapter has reviewed a method for parallelizing the PME and Lennard-Jones algorithms according to the spatial decomposition approach, such that the algorithm can be executed in a multi-processor environment. The general computation and communication architecture of the NAMD software is also discussed. In particular, the software creates individual compute objects to handle computation of the different types of interactions, and these objects are spawned off to different processors for execution to achieve parallelization.

Chapter 4

DESIGN IMPLEMENTATION

This chapter describes the two types of hardware engines that compute the Lennard-Jones and Ewald Direct Space non-bonded interactions. It starts by describing the configuration for testing the hardware. Then, the arithmetic architecture of the hardware engines will be discussed in detail.

4.1 Hardware Testing

Chapter 3 has described how the computation of the non-bonded interactions, Lennard-Jones and Ewald Direct Space sum, are performed within the individual compute objects. The goal of this work is to design dedicated hardware that performs the same task as these compute objects do and integrate the hardware into the NAMD software. As the MD simulation is executed, the software function that corresponds to the compute object is called and the particle information is provided for computation. Instead of performing the computation in the software domain, the goal is to forward the particle data to the hardware through a communication interface, trigger the computation, and retrieve the result back.

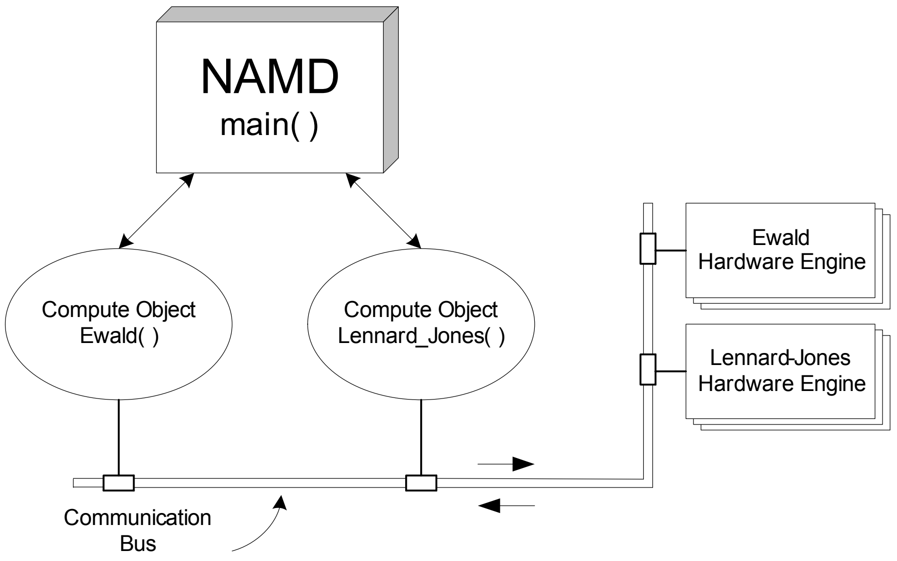


Figure 4.1 Overall system architecture

Figure 4.1 shows the configuration for testing the hardware engines. It uses the NAMD software compute objects to act as software driver code for interfacing with the hardware engines. The actual configuration is to connect a Linux workstation to a single Xilinx FPGA development board via a serial link as shown in Figure 4.2.

The NAMD 2.2 software is selected for integration with the hardware and to perform MD simulation. The reason for choosing this particular release is because the compute objects that compute Lennard-Jones and Ewald Direct Space can be easily extracted for interfacing with the hardware.

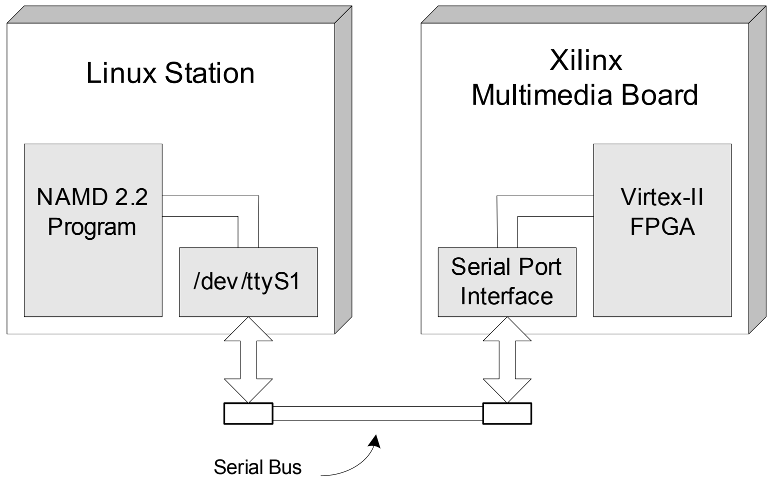


Figure 4.2 Serial bus communications

4.2 Hardware Resources

The Virtex-II XC-2V2000 FPGA on the Xilinx Multimedia Board [27] is the target device to be used for the compute engine implementation. The device comes with a list of features that are suitable for the purpose of arithmetic operations, data lookup, and temporary storage. The following highlights the major features of the device and the multimedia board:

1. Xilinx MicroBlaze 32-bit soft processor

- User C program can be compiled into assembly code, downloaded, and executed on the embedded soft processor.

2. Multiplier Blocks

- 56 dedicated multiplier blocks are provided. Each of the multiplier blocks is an 18-bit by 18-bit 2's complement signed multiplier. A larger multiplier can be built by concatenating the primitive multiplier blocks together.

3. Block RAMs

- 56 dedicated SelectRAM blocks are provided. Each of the SelectRAM blocks is an 18-Kb dual-port memory that can be configured into various depths and widths. A larger block RAM can be built by concatenating the primitive SelectRAM blocks together.

4. ZBT External Memory

- The multimedia board supports five independent banks of 512K x 36-bits ZBT RAM. Each can be run at a speed of 130 MHz with byte-write capability.

5. Serial Communication Interface

- The multimedia board provides a connection via an RS-232 port. This interface is used to communicate with the software running on a Linux workstation.

For more detail about usage and features of the various FPGA resources, please refer to Xilinx Virtex-II data sheets [28].

4.3 Communication Interface

The NAMD 2.2 program is configured to run on a single processor under the Linux operating system. The information is sent to the hardware compute engine through the serial communication port of the Linux workstation as shown in Figure 4.2. On the hardware side, the board has a serial port interface that can be accessed from the Virtex-II FPGA. This end-to-end connection enables the communication between the software compute object and the hardware compute engine.

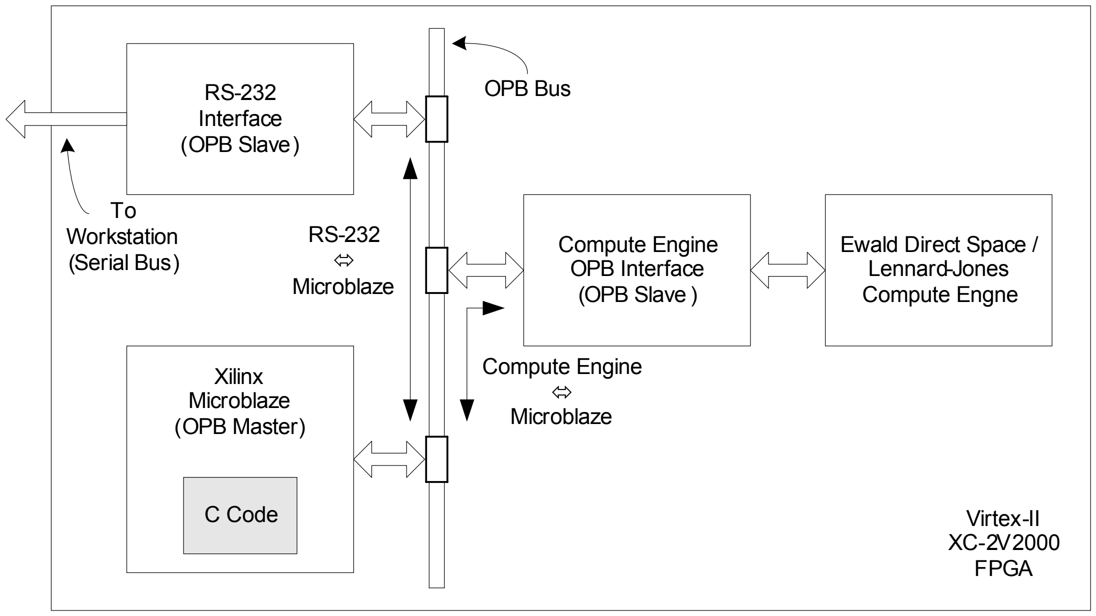
Each side of the serial communication link has a limited buffer size to store incoming data. The buffer sizes on the Linux station and the hardware board are 4 bytes and 16 bytes respectively. If the data packets are arriving at a rate faster than the data can be retrieved from the buffer, the buffer will be flooded and the additional incoming data will be dropped, causing the loss of data packets. To avoid this problem, a hand-shaking protocol is defined between the master (Linux workstation) and the slave (hardware board). The method is to put the sender into a halt state until an acknowledgement is received from the receiving end. The maximum achievable baud rate on the serial bus is 57600 bps. It is important to note that the actual data rate is also limited by the bus turn-around time since: (i) an acknowledgement scheme is used that introduces additional waiting time, and (ii) the buffer sizes on the two ends are relatively shallow increasing the frequency of waits for acknowledgement.

4.3.1 Linux Workstation Software Driver

The software running on the Linux workstation can have access to the serial bus by opening the terminal file /dev/ttyS0 as a file descriptor in C. The write( ) and read( ) function calls are used to send and receive data respectively from the serial bus buffer. The lowest layer of the software, the serial bus driver, handles the hand-shaking protocol between the master and slave such as the data sending and acknowledgement. A higher level of the software is responsible for data conversion between double precision and fixed-point data types, as well as keeping track of how many bytes of particle information are sent to the compute engine and how many bytes of the force and energy result are to be retrieved back. A sample C code of the serial bus driver is provided in Appendix D.

4.3.2 Hardware Compute Engine Interface

The Xilinx design environment is equipped with different hardware peripherals that allow the designer to build a base system and interact with an application-specific block defined by the user. Figure 4.3 shows the block diagram of the interface logic between the serial communication bus and the MD compute engine. The interface module includes the Xilinx Microblaze soft processor, RS-232 interface, compute engine interface, and the on-chip peripheral bus (OPB). The Microblaze processor is acting as the master device that talks to other OPB slave devices connected to the bus that are referenced by a 32-bit address.



OPB Bus
RS-232 Interface (OPB Slave)
To Workstation (Serial Bus)
RS-232 ⇔ Microblaze
Compute Engine OPB Interface (OPB Slave)
Ewald Direct Space / Lennard-Jones Compute Engne
Compute Engine ⇔ Microblaze
Xilinx Microblaze (OPB Master)
C Code
Virtex-II XC-2V2000 FPGA

Figure 4.3 Compute engine interface and OPB structure

The C code running on the processor is acting as driver code that serves three purposes: (i) to send and receive data from the RS-232 interface block, (ii) handling the hand-shaking protocol with the C code running on the other end of the serial bus, and (iii) to deliver particle information, parameters, and memory content to the compute engine, and retrieve results back. Since the two OPB slave devices are unable to communicate to each other directly, the main responsibility of the master device is to serve as a two-way transceiver to bridge between the RS-232 and the user hardware. A sample C code of the OPB driver is provided in Appendix D.

4.4 Arithmetic Structure

The IEEE double precision data type is commonly used in software programming such as the NAMD software for storing particle information and system parameters. It is possible to carry out the same computation in double precision on the FPGA compute engine but the high use of hardware resources and the complexity is not desirable. Instead, the arithmetic unit of the compute engine is built by using a combination of fixed-point arithmetic and table lookup. The primary advantages of using these two methods are: (i) to reduce the hardware usage, and (ii) to reduce the number of arithmetic operations, which improves the precision of the final result. Further discussion of each of the items above is provided below. In Chapter 5, a study is provided to analyze the accuracy of the computed result versus floating point using different arithmetic precisions.

4.4.1 Fixed-Point Arithmetic

In a molecular dynamics simulation, there is a list of parameters that define the characteristic of the system such as the simulation box size and cut-off radius. The computation unit is also supplied with a list of particle information that includes particle positions, charge quantity, and other constant values. The purpose of the hardware compute engine is to compute the energy and force between a given pair of particles based on these system constraints. It is expected that only a limited range of input values would need to be supported by the compute engine. Therefore, the arithmetic operations of the compute engine could be performed in a fixed-point format rather than a floating-point format. Compared to the floating-point engine, the primary advantages of designing the arithmetic unit in the fixed-point domain are the saving in hardware space, improvement in speed, and lower power consumption.

Figure 4.4 shows the format of a fixed-point data type that consists of the integer part and the fractional part. For the rest of the discussion, the notation $\{X, Y\}$ is used to represent a fixed-point data type with X bits for the integer parts and Y bits for the fractional parts. The maximum value that can be stored in a $\{X, Y\}$ data is $(2^X - 1) + (1 - 2^{-Y})$.

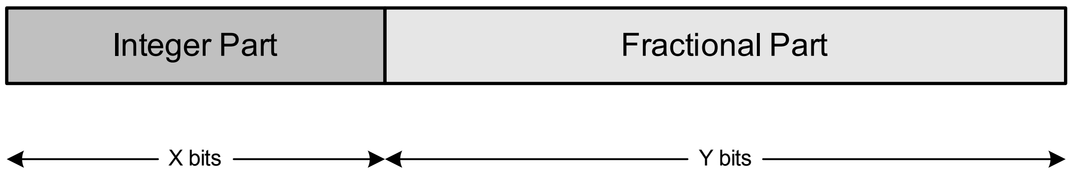


Figure 4.4 Fixed-point data type

Similar to floating-point operations, there are also arithmetic exceptions to be handled in the fixed-point domain such as overflow and underflow. The challenges in avoiding overflow are: (i) to allocate enough significant bits for the intermediate arithmetic results given the limited amount of hardware resource available, and (ii) predict and define the system input parameters such that the arithmetic operations can be performed within a permissible range. Underflow is less harmful because truncation of the result would only come with a penalty of the loss in precision. It is also important to reduce the number of addition and multiplication steps in the force and energy computations because every arithmetic step injects error into the intermediate result and it is carried forward to the final result. It is especially severe for multiplication because the error is magnified by the operation. More precision issues will be addressed in Section 4.5.

4.4.2 Memory Lookup Table

The purpose of the compute engines is to evaluate the force and energy given a pair of particles. For Lennard-Jones, the energy and force equations are expressed in Equations 2.2 and 2.3. For the Ewald Direct Space, the energy and force equations are expressed in A.10 and A.11 respectively. These equations involve complicated arithmetic operations that include division, square root, high order of powers, exponentials, the error function erfc( ), and also numerous steps of addition and multiplication. The disadvantages of performing a direct evaluation include: (i) too many arithmetic steps accumulate a substantial degree of error in the final result, (ii) implementation requires a lot of hardware resource, and (iii) increase in clock latency of the hardware pipeline.

A closer look at these force and energy equations shows that the entire equation or part of the equation can be expressed as a function of *r*, the separation distance between the pair of particles.

An alternative method for evaluating the equations is to perform memory lookup, followed by interpolation given that the *r* is known. The value and the slope of an equation at discrete points is pre-computed using double precision in a C program, converted into fixed-point format, and loaded into the memory prior to starting the simulation. After obtaining the memory lookup result, the final energy and force equations can be calculated with a few extra steps of addition and multiplication performed in the fixed-point domain.

Figure 4.5 shows a function curve that is sampled into points. The value and slope at these discrete points are mapped into separate memory banks that contain the lookup table for the function.

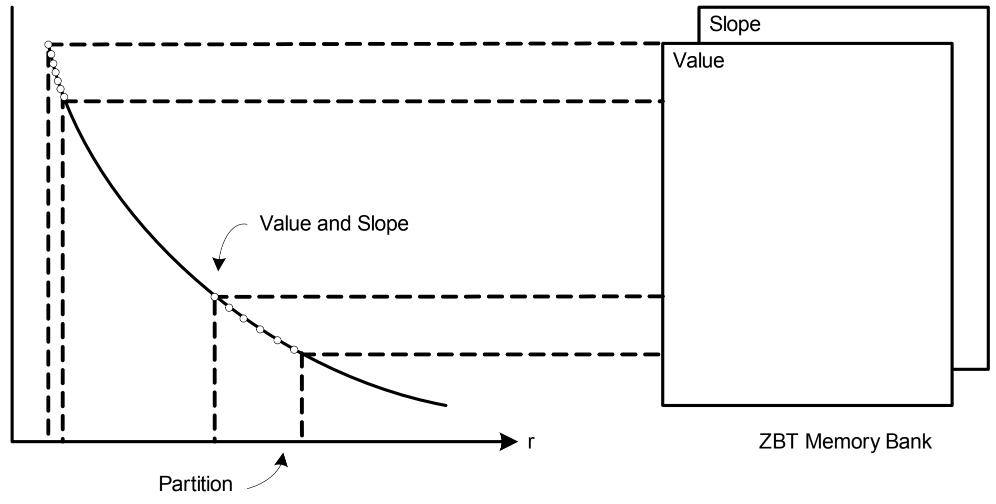


Figure 4.5 Function curve sampling and lookup table construction

With the function value and slope available at the discrete points, any intermediate points on the curve can be obtained by interpolation. The requirement in the function partitioning process is to maintain the relative error of the interpolated result at a uniform level across the entire table. The precision of the final interpolated result is based on factors such as the density of the lookup points and the type of interpolation. Figure 4.5 demonstrates the general rule that more lookup points should be stored for a region that is steeper in slope versus the flatter region at the tail of the function curve. The interpolation method adopted in this implementation is linear interpolation that consumes two words of memory space, the function value and slope, per sample point. Higher orders of interpolation, such as cubic spline interpolation [29], can provide better results, however, it requires more memory space for the table to store higher order coefficients. Section 4.5 will discuss how the lookup table methodology is employed to evaluate the Ewald Direct Space and Lennard-Jones force and energy equations. For other methods of function evaluation by table lookup, an extensive resource can be found in [30].

4.5 Hardware Compute Engine

The previous sections have introduced the overall system architecture that provides an idea of where the compute engine is positioned within the entire system, how it interacts with the software, and its purpose. The goal of this section is to discuss the internal hardware architecture of the Ewald Direct Space and Lennard-Jones compute engines. Several design aspects will be described such as the arithmetic units, memory consumption, data flow, and pipeline architecture. The following highlights the common design goal of the compute engines:

- Modularize the design to improve reusability of blocks for different compute engines such as the memory controller, function lookup, and function interpolation.
- Pipeline the design into stages to allow one pair of particles to be processed every clock cycle to maximize data throughput.
- Pipeline the combinational logic where hardware resources are available to reduce the propagation delay, and therefore, maximize the operating clock frequency.
- Maximize the precision of outputs from each stage of logic based on the available hardware resources in subsequent stages and system constraints.

Since the computation unit is based on fixed-point arithmetic with limited resources on the targeted FPGA device, it is necessary to define some system constraints of what the hardware compute engine can support. Table 4.1 summarizes a list of parameters that should be considered in the design. The actual hardware unit that relates to each of these parameters will then be discussed.

Parameter	Description
Simulation Box Size	Cubic simulation box with maximum length of 128Å. This is adequate for small systems in NAMD simulation [17].
Cut-off Radius	Typical value for the Ewald Direct Space cut-off radius is from 12Å to 16Å [17]. The same cut-off radius can be applied to Lennard-Jones computation because its potential function decays much more rapidly than the Ewald Direct Space counterpart.
Memory Lookup Data	The on-board ZBT memory only has 32 bits of the data bus connected to the FPGA. Thus, data stored in ZBT memory has a maximum width of 32 bits.
Particle Charge	A partial charge quantity is assigned to each atom in the system. The absolute value of the allowable charge is within the range of [0, 4) Coulomb (in fixed-point format {2.29})
Particle Separation Distance	The minimum separation distance of non-bonded atoms under the influence of Lennard-Jones and Coulombic force is roughly 1Å to 1.5Å [17].

Table 4.1 System parameters and constraints

4.5.1 Ewald Direct Space Compute Engine

The derivation of the Ewald Direct Space potential and force functions is described in Section 2.7 and the equations are shown in A.10 and A.11 respectively. These equations are simplified and rearranged to reflect the structure of the arithmetic core inside the compute engine. The simplifications applied to the equations are: (i) the constant term is removed because it can be easily performed in software, and (ii) the summation sign is removed because the hardware arithmetic unit is calculating potential and force for a single pair of particles. The final potential and force functions implemented in the compute engine are shown in Equations 4.1 and 4.2 respectively. The subscript *CE* stands for compute engine. The algebraic expressions within the square brackets in Equations 4.1 and 4.2 are represented by [*A*] and [*B*] respectively.

$$|P_{CE}| = q_i q_j \times \left[\frac{erfc(\alpha r_{ij})}{r_{ij}} \right] = q_i q_j \times [A] \tag{4.1}$$

$$\vec{F}_{CE} = \vec{r}_{ij} \times q_i q_j \times \left[\left(erfc(\alpha r_{ij}) + \frac{2\alpha}{\sqrt{\pi}} r_{ij} \exp(-(\alpha r_{ij})^2) \right) \frac{1}{r_{ij}^3} \right] = \vec{r}_{ij} \times q_i q_j \times [B] \tag{4.2}$$

The term within the square brackets will be evaluated by table lookup as discussed in Section 4.4.2. The pre-computed values are loaded into the ZBT memory and each entry is indexed by the single variable $|r_{ij}|^2$. The rest of the computation involves only additions and multiplications in fixed-point arithmetic. Based on the partitioning of Equations 4.1 and 4.2, the equations are mapped into the arithmetic core as shown in Figure 4.6. The figure highlights the key control logic, the precision of the input and intermediate variables in fixed-point format, and the data flow.

4.5.1.1 Input Variables

The input to the arithmetic unit includes: (i) the coordinates of the pair of particles, (ii) the charge quantity and polarity of the pair of particles, and (iii) the constant value of the cut-off radius and simulation box size. The specific information of the particles, coordinates and charge quantity, is represented in software as 32-bit unsigned integers and then is loaded into the hardware temporary storage. The input coordinates are used to represent points in space within the simulation box that has a maximum size of 128Å as defined in Table 4.1. Thus, to represent a position within the range of [0, 128), it is required to allocate 7 bits for the integer part and the rest of the 25 bits for the fraction. The position will not reach 128.0 because the box wraps around under the periodic boundary condition, thus, the particle exits the current box at 128.0 while its image enters at 0.0. For the particle charge, 1 bit is allocated to represent the polarity of the charge and the rest of the 31 bits are used to store the charge quantity within the range of [0, 4) Coulomb as defined in Table 4.1. Table 4.2 summarizes the precision in fixed-point format for all the input variables to the Ewald compute engine.

Input	Sign	Integer	Fraction	Fixed-Point	Range
Coordinate	-	7	25	{7.25}	[0, 128)
Charge	1	2	29	{2.29}	(-4, 4)

Table 4.2 Input variables precision

4.5.1.2. Hardware Pipeline

The first stage of the pipeline engine is to evaluate the delta position of each of the *x*, *y*, and *z* components for a pair of particles concurrently in three separate instances of hardware as shown in Figure 4.6. The delta position is computed under the periodic boundary condition as shown in Figure 2.6. A pair of particles within the cut-off radius can be found either within the original simulation box or in adjacent image cells.

As an example, in Figure 2.6 the logic has the capability to distinguish that Particle 4 is outside the cut-off radius of Particle 1 while Particle 4E is inside the cut-off radius. There are two control signals output from the first stage to represent: (i) the direction of the interaction between the pair of particles, and (ii) whether the pair of particles is within the cut-off radius. The direction control bit is used in a later pipeline stage to compute the polarity of the force vector, which is determined in this stage based on the relative particle locations within the simulation box. The other control bit is asserted if the pair of particles is within the defined cut-off radius.

The computed delta position has a range of [0, 16), that is, up to the maximum value of the cut-off radius as defined in Table 4.1. The cut-off control bit is used to represent an overflow condition that indicates whether the pair of particles is outside of the cut-off radius. When the cut-off control bit is asserted, there is no overflow condition. Thus, the computed delta position can be stored and forwarded to the second stage with a maximum data width of {4.25}. On the other hand, the pair of particles will be discarded if any of the three cut-off control bits is set to zero, indicating that the separation distance is larger than the cut-off radius. The maximum data width of {4.25} also limits the size of the multiplier in the second stage to 29 bits by 29 bits.

The first stage in the pipeline also evaluates the product term q_i x q_j which is used for both Equations 4.1 and 4.2. The intermediate result is expressed in {4.58} immediately after the multiplication, and is then truncated down to {4.29} based on the input precision. The polarity of the q_i x q_j product term is the xor-ed result of the charge polarity of the two particles.

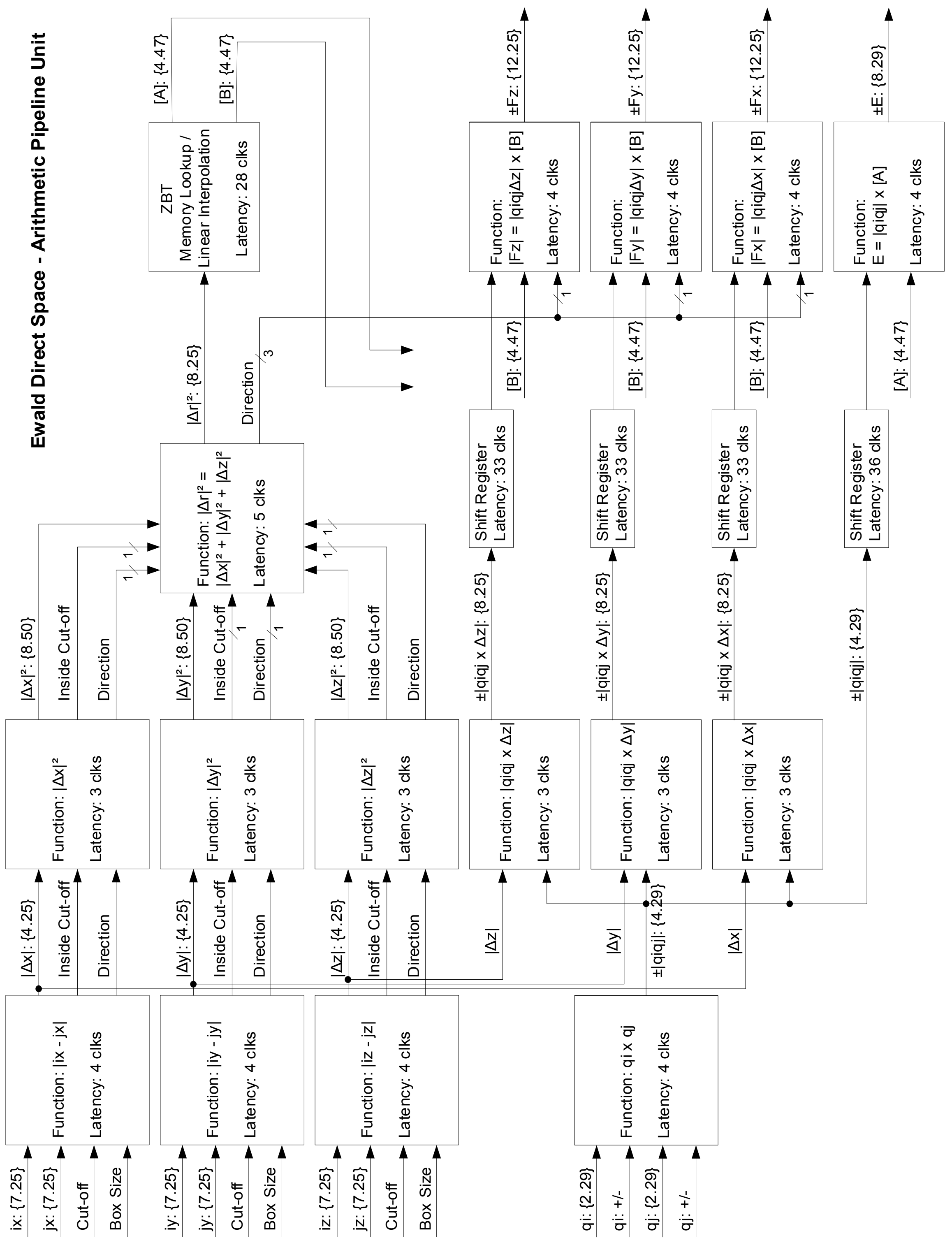


Figure 4.6 Ewald Direct Space arithmetic pipeline unit

In the upper portion of Figure 4.6, the purpose of the second and third pipeline stage is to evaluate the $|\Delta r|^2$ such that it can be used as an input to the memory lookup block to obtain the $[A]$ and $[B]$ terms in Equations 4.1 and 4.2 respectively. The second stage is simply to input the delta position with the precision of {4.25} into the two ports of the multiplier to obtain the squared result with output precision of {8.50}. Each of the three separate results, $|\Delta x|^2$, $|\Delta y|^2$, and $|\Delta z|^2$, has a range of [0, 256). They are forwarded to the third stage without truncation along with the direction and cut-off control bit.

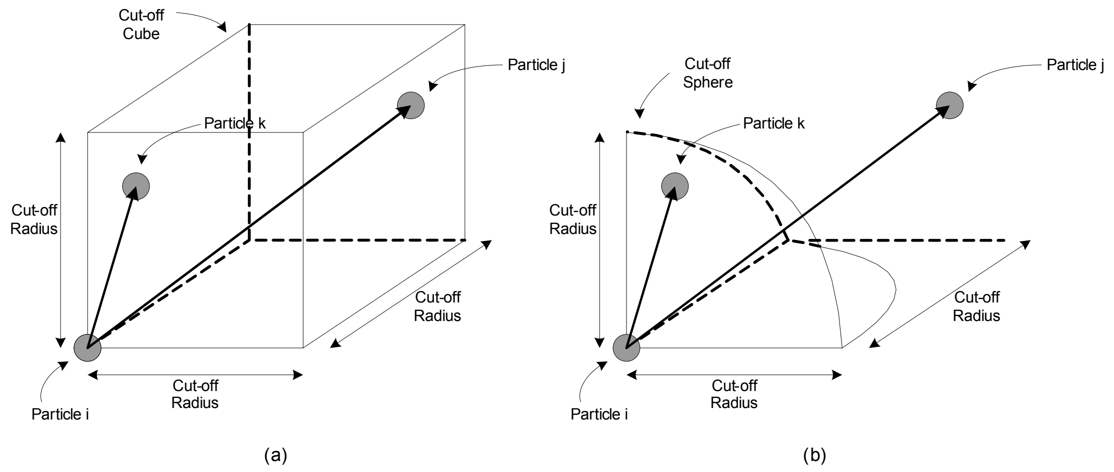


Figure 4.7 Cut-off cube and cut-off sphere of a pair of particles

Inside the third pipeline stage, the $|\Delta x|^2$, $|\Delta y|^2$, and $|\Delta z|^2$ are summed up together to obtain an intermediate result with precision of {10.50} that can hold the value of 3 x 256 = 768 $> |\Delta r|^2$. The three cut-off control bits forwarded from the first pipeline stage are used here to determine if the pair of particles is within the cut-off cube as shown in Figure 4.7a. If any of the three cut-off control bits is not asserted, it means that the pair of particles cannot be within the cut-off cube. Thus, the pair is discarded for memory lookup and computation in subsequent pipeline stages. The result for that particular pair of particles coming out at the end of the pipeline will be marked as invalid. On the other hand, if the pair of particles is within the cut-off cube, the computed value of $|\Delta r|^2$ is then used to check if it is inside the cut-off sphere as shown in Figure 4.7b, that is, whether $|\Delta r|^2$ is less than $|\text{cut-off radius}|^2$. Figure 4.7a shows an example where Particles j and k are both within the cut-off cube with respect to Particle i. The value of $|\Delta r|^2$ is then used to determine that Particle j is too far away from the origin (Particle i), and Particle k is inside the cut-off sphere which is inside the radius of non-bonded interaction.

The main purpose of using the cut-off control bits from the first stage in the pipeline is to reduce: (i) the data width, (ii) the multiplier size in the second stage, (iii) the adder and comparator hardware usage in the third stage, and (iv) the delay of the combinational logic. The final output of $|\Delta r|^2$ from the third stage has a precision of {8.25} that represents a range of [0, 256). This value is then used for the memory lookup block, which will be discussed in the Section 4.5.1.3.

In the lower portion of Figure 4.6, the purpose is to compute the $\vec{r}_{ij} \times q_i q_j$ term in Equation 4.2 by multiplying the product term $|q_i q_j|$ by Δx, Δy, and Δz in three separate instances of hardware. The multiplication outputs generate intermediate results with precision of {8.54} and truncated down to {8.25} based on the input significant values. The output data $|q_i q_j|$ from the first stage and $|q_i q_j \text{ x } \Delta|$ from the second stage are pipelined accordingly through the shift registers to line up with the interpolated result returned from the memory lookup block. The final stage of the pipeline engine is simply to multiply the terms [A] and [B] from the lookup tables by the previously generated partial products to produce the final result of the non-bonded energy and the force components between the pair of particles. Note that the sign of the result is based on the charge polarity and direction bits generated in an early stage of the pipeline. The final computed non-bonded energy has a precision of {8.29} and the non-bonded force components have a precision of {12.25}.

4.5.1.3 Memory Lookup and Function Interpolation

The output from the third pipeline stage in Figure 4.6, $|\Delta r|^2$ of the pair of particles, is used as the input to the memory lookup module to obtain the terms [*A*] and [*B*] as defined in Equations 4.1 and 4.2 by using first order linear interpolation. The terms [*A*] and [*B*] are repeated here as Equations 4.3 and 4.4.

$$[A] = \left[\frac{erfc(\alpha r_{ij})}{r_{ij}} \right] \tag{4.3}$$

$$[B] = \left[\left(erfc(\alpha r_{ij}) + \frac{2\alpha}{\sqrt{\pi}} r_{ij} \exp(-(\alpha r_{ij})^2) \right) \frac{1}{r_{ij}^3} \right] \tag{4.4}$$

The function values and slopes of [*A*] and [*B*] at discrete points are pre-computed in double precision in a C program, truncated and converted into fixed-point format, and loaded into the ZBT memory through the serial port communication interface. The table construction should involve the following characteristics and design considerations:

- The lookup table range is bounded by the minimum separation distance of non-bonded atoms in MD simulations and the maximum cut-off radius defined by the user.

- The input to the lookup table is $|\Delta r|^2$ instead of $|\Delta r|$ such that a square root procedure is avoided.
- The lookup table is partitioned into groups within the domain of [1/4, 256) that corresponds to the minimum separation distance and maximum cut-off radius.
- Both functions [*A*] and [*B*] are very steep at small values of *r* and decay exponentially as *r* increases. More discrete points are stored in memory at the steeper slopes to achieve higher lookup density and therefore, better interpolation results. The functions [*A*] and [*B*] are similar to the erfc( ) plot as shown in Figure 2.8.
- The lookup table is divided into ten partitions: [1/4, 1/2), [1/2, 1), [1, 2), …, [64, 128), [128, 256). Each partition has half the lookup density of the previous partition as $|\Delta r|^2$ increases.
- Each of the functions [*A*] and [*B*] requires two banks of ZBT memory to separately store the function value and slope, *m*, to allow the data to be retrieved concurrently in one clock cycle in the pipeline. The compute engine consumes a total of four banks of ZBT memory for storing lookup data.
- The values of *m* are the slopes of two adjacent points on the piecewise linear functions [*A*] and [*B*].
- The lookup table is constructed using the block floating point memory lookup approach. The fixed-point data stored in each partition has a different binary point offset. This approach allows data to be stored as precisely as possible by removing some of the leading zeros and maximizing the usage of the 32 bits in a memory word to represent the non-zero portion of the fixed-point number.

- Since the block floating point memory lookup approach is used, a separate set of parameters are required to represent the binary point offset, the lookup size, and the ZBT base address of each partition.
- The lookup data of the functions [*A*] and [*B*] are computed for one specific value of Ewald coefficient α as discussed in Section 3.1. Using a different coefficient for another simulation requires the memory content to be reloaded.

The memory lookup and linear interpolation block shown in Figure 4.6 is presented in more detail in Figures 4.9 and 4.12. The rest of this section will provide an overview of the structure and data flow of the memory lookup module.

The data $|\Delta r|^2$ with precision of {8.25} is input to the partition selector in the first stage of the memory lookup module in Figure 4.9. The purpose of the partition selector is to determine which partition is to be used based on the case-logic in Figure 4.8. The partition selector only looks at the most-significant bits of $|\Delta r|^2$ and the rest are "don't care" bits.

<--------- $\|\Delta r\|^2$ {8.25} --------->	Partition	Lookup Range
00000000.01xxxxxxxxxxxxxxxxxxxxxxx	1	[0.25, 0.5)
00000000.1xxxxxxxxxxxxxxxxxxxxxxxx	2	[0.5, 1.0)
00000001.xxxxxxxxxxxxxxxxxxxxxxxxx	3	[1.0, 2.0)
0000001x.xxxxxxxxxxxxxxxxxxxxxxxxx	4	[2.0, 4.0)
000001xx.xxxxxxxxxxxxxxxxxxxxxxxxx	5	[4.0, 8.0)
00001xxx.xxxxxxxxxxxxxxxxxxxxxxxxx	6	[8.0, 16.0)
0001xxxx.xxxxxxxxxxxxxxxxxxxxxxxxx	7	[16.0, 32.0)
001xxxxx.xxxxxxxxxxxxxxxxxxxxxxxxx	8	[32.0, 64.0)
01xxxxxx.xxxxxxxxxxxxxxxxxxxxxxxxx	9	[64.0, 128.0)
1xxxxxxx.xxxxxxxxxxxxxxxxxxxxxxxxx	10	[128.0, 256.0)

Figure 4.8 Partition selector case-logic

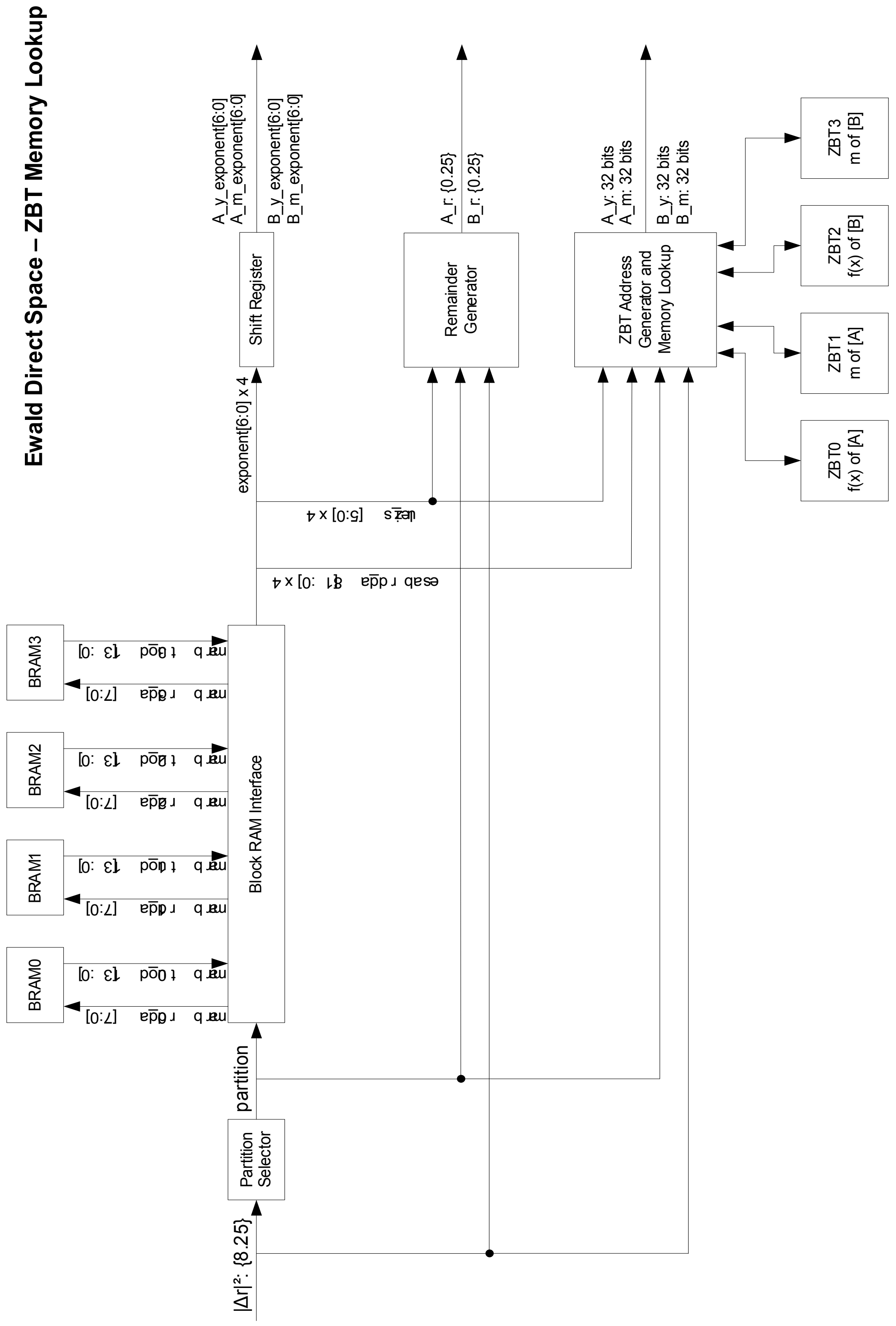


Figure 4.9 Ewald Direct Space ZBT memory lookup module

The second stage consists of four independent block RAMs that contain parameters describing the lookup tables stored inside the respective ZBT memory banks. Figure 4.10 shows one of the four block RAMs. It has ten entries of data that correspond to the ten partitions of a lookup table. Each entry contains parameters that describe the corresponding partition such as: (i) the exponent (binary point offset) of the fixed-point numbers, (ii) the lookup size, and (iii) the ZBT base address. The example given in Figure 4.10 is assumed to use a lookup size of 16k discrete points and an arbitrary offset value.

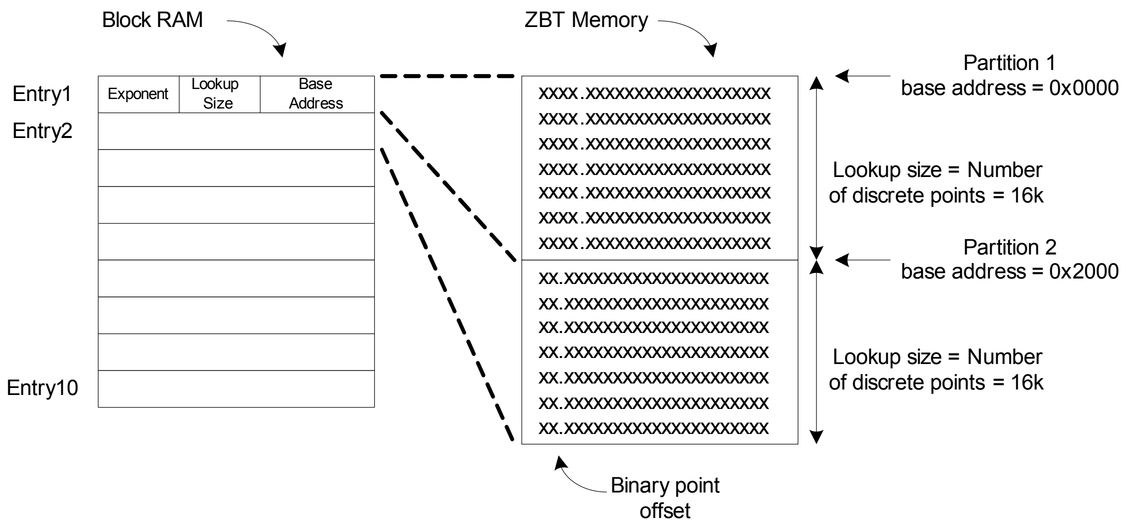


Figure 4.10 Block RAMs parameters

The partition number obtained earlier is used by the block RAM interface in Figure 4.9 as an index to retrieve the parameters from the corresponding block RAM entries. Once the parameters are returned from the block RAMs, the exponent values are pipelined through the shift register for later use, and the lookup size and base address are forwarded to the third stage of logic in the module.

Figure 4.11 shows an example where the value of $|\Delta r|^2$ falls inside the range of Partition 5 and the size of the partition is 16k. Since the partition contains 16k points (or intervals), 14 bits (the dark highlight) are used as the interval offset from the base address of the partition. The remaining bits (the grey highlight) represent the position of the point between two adjacent points in the lookup table, and its quantity is represented by ∂_r.

<--------- $\|\Delta r\|^2$ {8.25} --------->	Partition	Lookup Range	Size
00000000.01xxxxxxxxxxxxxx xxxxxxxxx	1	[0.25, 0.5)	16k
00000000.1xxxxxxxxxxxxxx xxxxxxxxxx	2	[0.5, 1.0)	16k
00000001.xxxxxxxxxxxxxx xxxxxxxxxxx	3	[1.0, 2.0)	16k
0000001x.xxxxxxxxxxxxx xxxxxxxxxxxx	4	[2.0, 4.0)	16k
000001xx.xxxxxxxxxxxx xxxxxxxxxxxxx	5	[4.0, 8.0)	16k
00001xxx.xxxxxxxxxxx xxxxxxxxxxxxxx	6	[8.0, 16.0)	16k
0001xxxx.xxxxxxxxxx xxxxxxxxxxxxxxx	7	[16.0, 32.0)	16k
001xxxxx.xxxxxxxxx xxxxxxxxxxxxxxxx	8	[32.0, 64.0)	16k
01xxxxxx.xxxxxxxx xxxxxxxxxxxxxxxxx	9	[64.0, 128.0)	16k
1xxxxxxx.xxxxxxx xxxxxxxxxxxxxxxxxx	10	[128.0, 256.0)	16k

▩ x - Interval Offset
▨ x - Remainder Bits

Figure 4.11 Interval offset and remainder bits

The purpose of the third stage of logic is to: (i) extract the interval offset and remainder bits of $|\Delta r|^2$, ∂_r, as shown in Figure 4.11, (ii) calculate the ZBT lookup absolute address by adding the interval offset and the partition base address together, and (iii) perform a ZBT memory read access through the ZBT memory interface.

The remainder bits have a precision of {0.25} and all the data returned from the ZBT memory are 32 bits wide with default precision of {1.31}. The scaling of these data to their actual values depends on the corresponding exponent (binary point offset) values retrieved from the block RAMs. The data returned from the ZBT memory lookup include the function and slope values of [*A*] and [*B*] at the point of interest.

The function interpolation module takes all the output data from the memory lookup module to perform linear interpolation ($y + m\partial_r$) to obtain the values of [A] and [B]. The block diagram of the function interpolation module is shown in Figure 4.12.

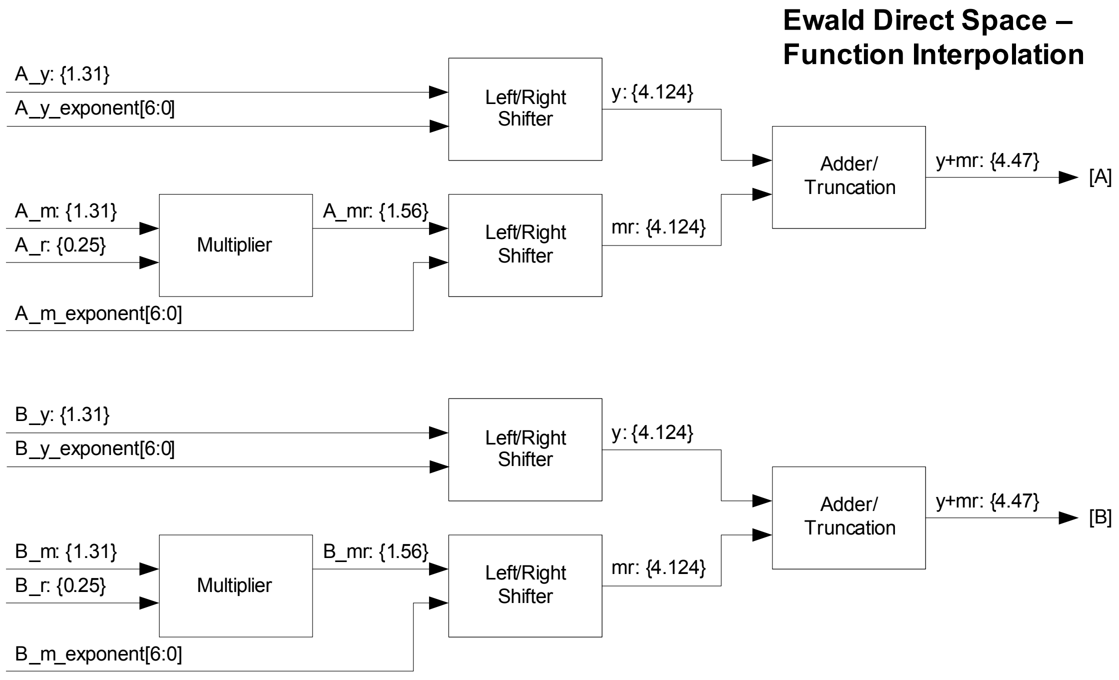


Figure 4.12 Ewald Direct Space function interpolation

As mentioned earlier, all the data stored inside the ZBT memory has default precision of {1.31}. The data has to go through left/right shifting based on the scaling value to align the binary point. The output from the shifter has precision of {4.124} to ensure the largest and smallest actual quantity can be represented. The last stage performs an addition and truncation to obtain the final value of [A] and [B] with precision of {4.47}.

Referring back to Figure 4.6, the values of [A] and [B] obtained here are input to the final multiplication stage to obtain the force components and energy term. The reason for maintaining a precision of {4.47} for [*A*] and [*B*] is to keep enough bits to be input to the largest available multipliers in the final stage in Figure 4.6. The ZBT lookup module and function interpolation module take 13 and 15 clocks respectively. A total of 28 clocks are required to obtain the value of [*A*] and [*B*] as shown earlier in Figure 4.6.

4.5.1.4 High-Level Data Flow and Operation

The top level view of the entire compute engine is briefly summarized in Figure 4.13. The signal names and bus widths are not shown in the figure.

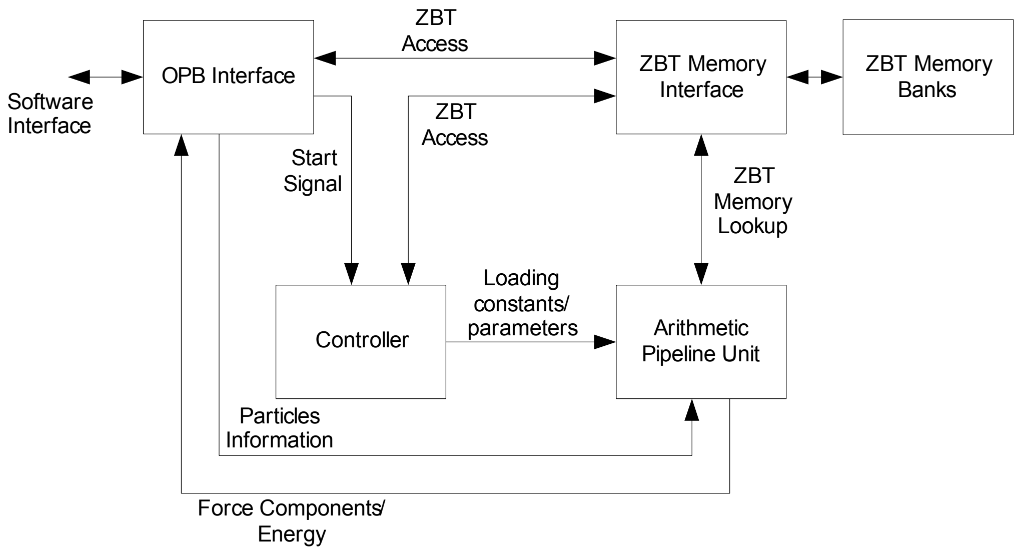


Figure 4.13 Compute engine top level view

Before computation takes place, the ZBT memory banks are initialized with the lookup tables by the software driver through the OPB interface. The data loaded into the ZBT memory can also be read back by the software for validation purposes to ensure that the tables are filled correctly. ZBT bank 0 to bank 3 are used for storing lookup tables and ZBT bank 4 is loaded with constants and parameters used during simulation. Once the ZBT memory banks are filled, the software driver will trigger a start command to the controller to indicate that the ZBT memory banks are ready for use. The controller will retrieve all the constants such as cut-off radius and box size from bank 4 of the ZBT memory and drive the interface signals of the arithmetic pipeline unit shown in Figure 4.6. The controller will also retrieve all the parameters and load them into the block RAMs inside the arithmetic pipeline unit shown in Figure 4.10. After these initialization steps are completed, the software driver is allowed to send particle information to the arithmetic pipeline unit through the OPB interface to perform computation. The force components and energy results are sent back to the OPB interface and then read back by the software driver.

4.5.1.5 Hardware Utilization

The design is implemented on a Virtex-II XC-2V2000 FPGA on the Xilinx Multimedia Board. The device utilization summary of the Ewald Direct Space compute engine design is given in Table 4.3.

Device Components	Logic Utilization	
External IO Buffer	317 / 624	50%
Block RAMs 18Kbit	24 / 56	42%
Multiplier 18x18	55 / 56	98%
Slices	8648 / 10752	80%

Table 4.3 Ewald Direct Space compute engine device utilization

The multiplier component is particularly important in the compute engine design because: (i) there are a lot of multiplication steps involved, and (ii) the arithmetic precision depends on the size of the multipliers. A detailed breakdown of the multiplier usage is show in Appendix E.

4.5.2 Lennard-Jones Compute Engine

The nature of the Lennard-Jones potential is discussed in Section 2.3 and the potential and force functions are shown in Equations 2.2 and 2.3 respectively. The memory lookup process in the Lennard-Jones compute engine works differently from the Ewald compute engine because it is not possible to isolate a single term from each of the potential and force functions that is expressed as a function of *r*. The σ in the equations is also a variable that depends on the pair of particles in the computation. The potential and force functions are rearranged in Equations 4.5 and 4.6 by factoring out σ^6.

$$U_{LJ} = 4\varepsilon\sigma^6\left[\frac{\sigma^{12}/\sigma^6}{r^{12}} - \frac{1}{r^6}\right] \tag{4.5}$$

$$F_{LJ} = 24\varepsilon\sigma^6\frac{1}{r}\left[2\frac{\sigma^{12}/\sigma^6}{r^{12}} - \frac{1}{r^6}\right] \tag{4.6}$$

Further simplifications applied to the equations are: (i) the constant term is removed because it can be easily performed in software, and (ii) the σ^{12} and σ^6 terms are provided to the hardware compute engine by the software driver as if they are constant values. The final potential and force functions to be implemented by the compute engine are shown in Equations 4.7 and 4.8 respectively. The subscript CE stands for compute engine. The terms [*C*] and [*D*] are $1/r^6$ and $1/r^2$ respectively.

$$\left|P_{CE}\right| = \frac{sig12\_6}{r^{12}} - \frac{1}{r^6} = sig12\_6 \times [C]^2 - [C] \tag{4.7}$$

$$\vec{F}_{CE} = \vec{r}_{ij} \times \left(2\frac{sig12\_6}{r^{12}} - \frac{1}{r^6}\right)\frac{1}{r^2} = \vec{r}_{ij} \times \left(2 \times sig12\_6 \times [C]^2 - [C]\right) \times [D] \tag{4.8}$$

Unlike the Ewald Direct Space compute engine, the terms within the square brackets being looked up in this case are used by both the potential and force equations. The pre-computed values of [*C*] and [*D*] are loaded into the ZBT memory and each entry is indexed by the single variable r_{ij}. Based on the partitioning of Equations 4.7 and 4.8, the equations are mapped into the arithmetic core as shown in Figures 4.14 and 4.15.

4.5.2.1 Input Variables

The input to the arithmetic unit is slightly different for the Lennard-Jones compute engine. It includes: (i) the coordinates of the pair of particles, (ii) the σ^{12}/σ^{6} term provided by software, and (iii) the constant value of the cut-off radius and simulation box size. The input coordinates are used to represent points in space within the simulation box that has a maximum size of 128Å as in the previous case. The σ^{12}/σ^{6} term should be generated by table lookup given the types of particles are known. Since there are not enough hardware resources, the term is pre-computed by software for this design and supplied as an input to the compute engine. Its range of values is determined experimentally by running selected NAMD simulations. Table 4.4 summarizes the precision in fixed-point format for all the input variables to the Lennard-Jones compute engine.

Input	Integer	Fraction	Fixed-Point	Range
Coordinate	7	25	{7.25}	[0, 128)
sig12_6	13	19	{13.19}	[0, 8192)

Table 4.4 Input variable precision

The σ^{12}/σ^{6} term lookup is implemented in the NAMD software as a two-dimensional table indexed by the types of particles in computation. The size of the lookup table is found experimentally to have less than 1K entries. With the available hardware resources, the table can be constructed in the ZBT memory loaded with pre-computed σ^{12}/σ^{6} values. The table lookup index can be determined by the particle types, which is provided by the software driver along with the particle position coordinates.

4.5.2.2 Hardware Pipeline

Figure 4.14 shows the first part of the pipeline engine. The upper portion is identical to the Ewald Direct Space counterpart that calculates the $|\Delta r|^2$ given the coordinates of a pair of particles. Equations 4.7 and 4.8 show that the terms [C] and [D] are required before any further computation can proceed. Hence, the first part of the pipeline engine is mainly to perform the ZBT memory lookup to obtain the values of [C] and [D]. The intermediate values of $|\Delta x|$, $|\Delta y|$, and $|\Delta z|$ with precision of {4.25} in the lower portion go through the shift register to line up with the data returned from the ZBT lookup module.

The upper portion in the second part of the pipeline engine in Figure 4.15 is to evaluate the partial products: (i) sig12_6 x $[C]^2$, and (ii) 2 x sig12_6 x $[C]^2$. The input [C] is squared and truncated to obtain an intermediate result with precision of {8.47}. It is then multiplied by the sig12_6 term with precision of {13.19} that is provided by software to obtain the two partial products above with precision of {22.66}. These results go through the subtraction stage to obtain the sums: (i) sig12_6 x $[C]^2 - [C]$, and (ii) 2 x sig12_6 x $[C]^2 - [C]$. The precision of the sum sig12_6 x $[C]^2 - [C]$ is truncated from {22.66} down to {22.42} as the final non-bonded energy output. The precision of the sum 2 x sig12_6 x $[C]^2 - [C]$ is truncated from {22.66} down to {22.40} that is determined by the maximum available input precision of the multiplier in the final logic stage.

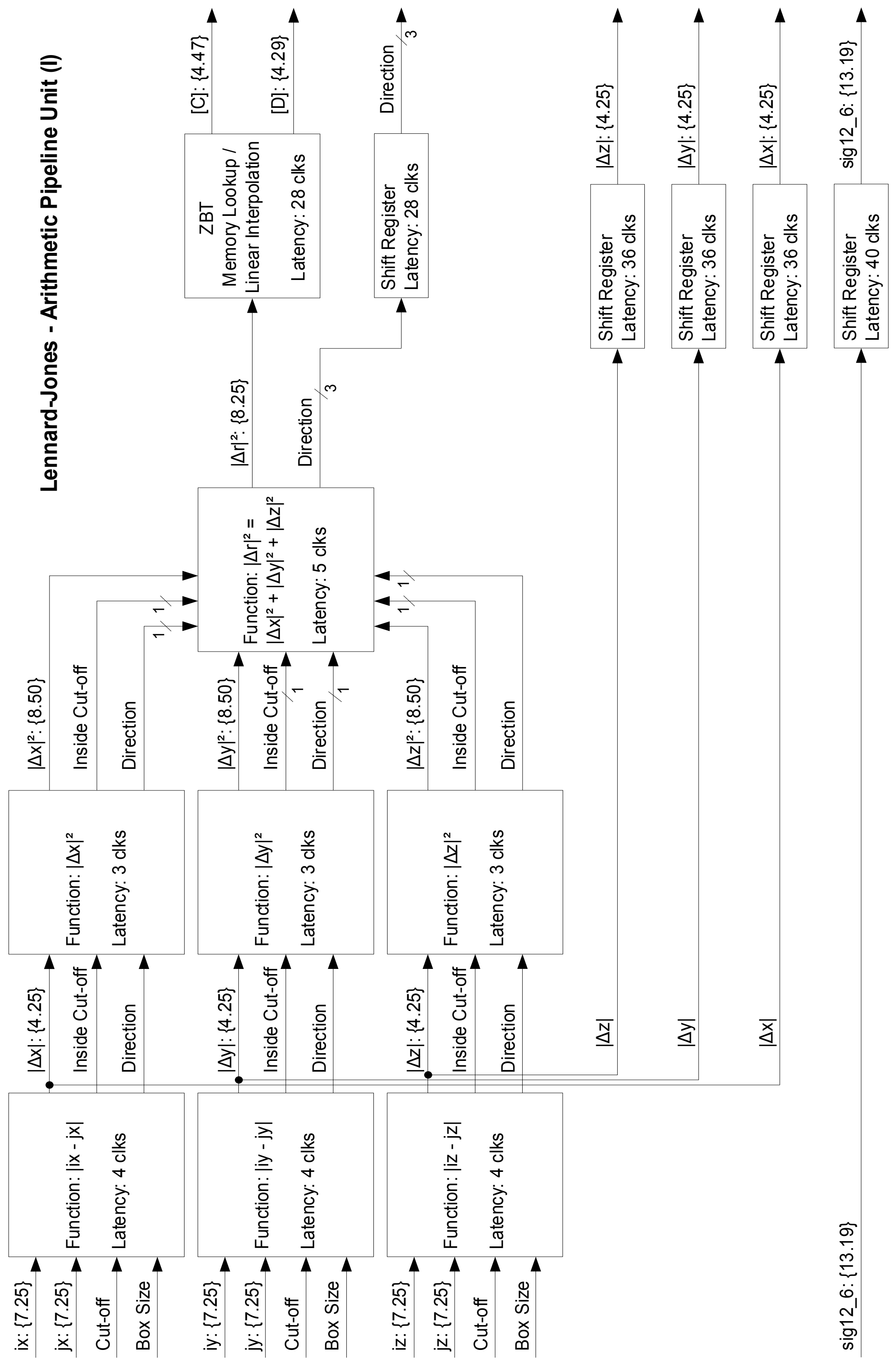


Figure 4.14 Lennard-Jones arithmetic pipeline unit (I)

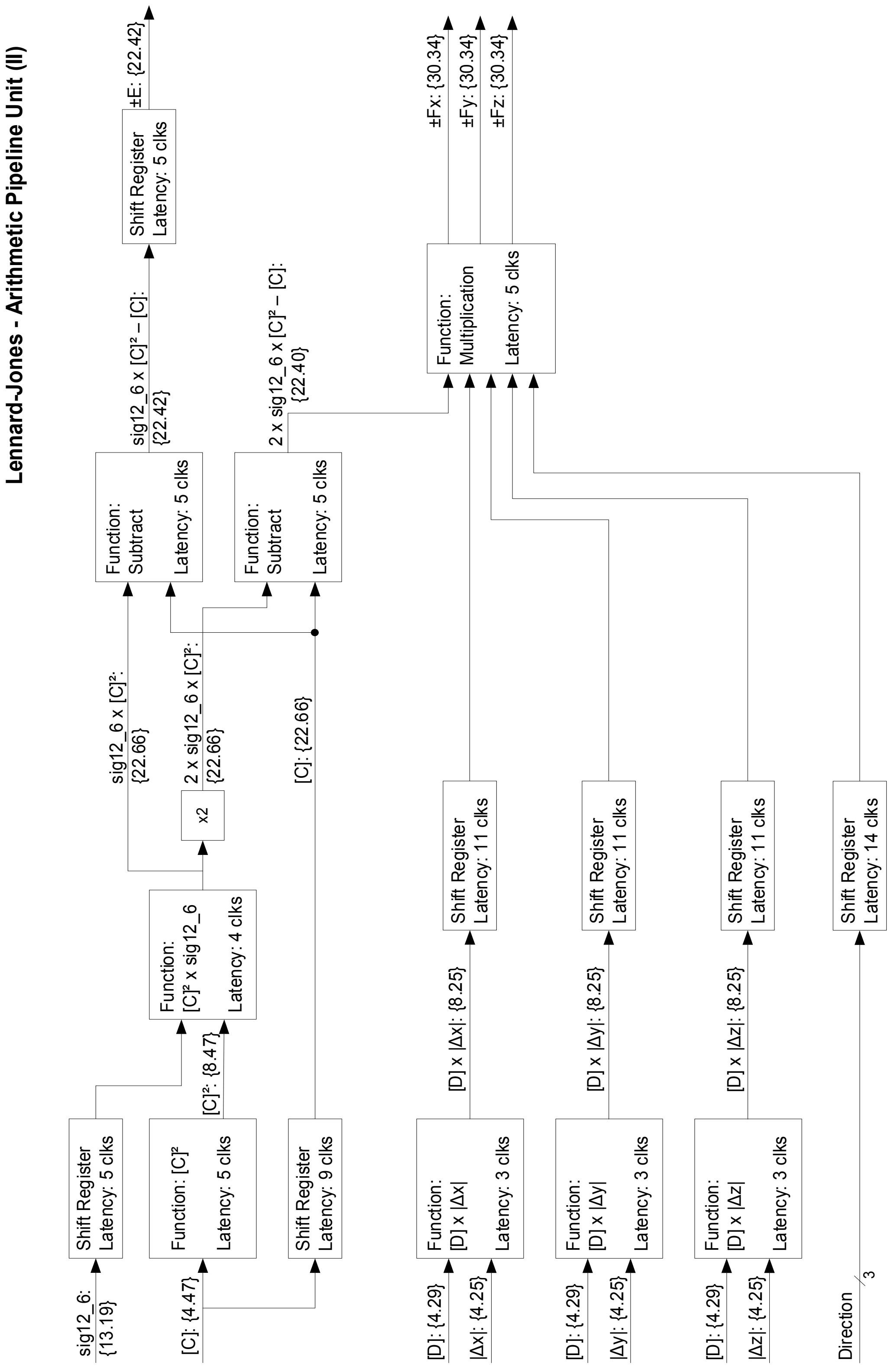

Lennard-Jones - Arithmetic Pipeline Unit (II)
sig12_6: {13.19}
Shift Register Latency: 5 clks
[C]: {4.47}
Function: [C]² Latency: 5 clks
[C]²: {8.47}
Shift Register Latency: 9 clks
Function: [C]² x sig12_6 Latency: 4 clks
x2
sig12_6 x [C]²: {22.66}
2 x sig12_6 x [C]²: {22.66}
[C]: {22.66}
Function: Subtract Latency: 5 clks
Function: Subtract Latency: 5 clks
sig12_6 x [C]² – [C]: {22.42}
2 x sig12_6 x [C]² – [C]: {22.40}
Shift Register Latency: 5 clks
±E: {22.42}
[D]: {4.29}
|Δx|: {4.25}
Function: [D] x |Δx| Latency: 3 clks
[D] x |Δx|: {8.25}
Shift Register Latency: 11 clks
[D]: {4.29}
|Δy|: {4.25}
Function: [D] x |Δy| Latency: 3 clks
[D] x |Δy|: {8.25}
Shift Register Latency: 11 clks
[D]: {4.29}
|Δz|: {4.25}
Function: [D] x |Δz| Latency: 3 clks
[D] x |Δz|: {8.25}
Shift Register Latency: 11 clks
Direction
3
Shift Register Latency: 14 clks
Function: Multiplication Latency: 5 clks
±Fx: {30.34}
±Fy: {30.34}
±Fz: {30.34}

Figure 4.15 Lennard-Jones arithmetic pipeline unit (II)

The lower portion corresponds to the $\vec{r}_{ij} \times [D]$ in Equation 4.8. The result [D] returned from the ZBT memory lookup is multiplied by $|\Delta x|$, $|\Delta y|$, and $|\Delta z|$ in three separate instances of hardware. The multiplications generate intermediate results and are truncated to {8.25} based on the input precision, and then go through shift registers to line up with the result 2 x sig12_6 x $[C]^2$ from the upper portion.

For the multiplication in the final stage in the pipeline engine, there is only one instance of multiplier implemented given the limited hardware resources. This multiplier is shared among the three separate inputs to produce the force components of the final result. Hence, the pipeline can only accept an input every three clock cycles as a maximum processing rate. Future designs using a larger FPGA device should have this multiplication implemented in three separate instances such that the pipeline throughput can be maximized.

The final computed non-bonded energy has a precision of {22.42} and the non-bonded force components have a precision of {30.34}. All results are 64 bits wide and can be read back by the software driver in two words.

4.5.2.3 Memory Lookup and Function Interpolation

Similar to the Ewald Direct Space compute engine, the output from the third pipeline stage in Figure 4.14, $|\Delta r|^2$ of the pair of particles, is used as the input to the memory lookup module to obtain the terms [C] and [D], as defined in Equations 4.7 and 4.8, by using first order linear interpolation. The terms [C] and [D] are repeated here as Equations 4.9 and 4.10.

$$[C] = \left[\frac{1}{r_{ij}^{6}}\right] \tag{4.9}$$

$$[D] = \left[\frac{1}{r_{ij}^{2}}\right] \tag{4.10}$$

The table construction in this case is slightly different from the Ewald Direct Space counterpart. The differences are outlined in the following design considerations:

- The lookup table is partitioned into groups within the domain of [1/2, 256) that corresponds to the minimum separation distance and maximum cut-off radius.
- The lookup table is divided into nine partitions: [1/4, 1/2), [1/2, 1), [1, 2), …, [64, 128), [128, 256). Each partition has half the lookup density of the previous partition as $|\Delta r|^2$ increases. This partitioning is the same as in the case of Ewald Direct Space except for the range of [1/4, 1/2).
- The partition that covers [1/4, 1/2) is not supported by the compute engine because some values stored inside the lookup table are too large resulting in overflow in the current implementation with limited hardware resources. For example, if $|\Delta r|^2$ equals 1/4, the lookup value of the term [*C*] in Equation 4.9 is 64. This creates an overflow since the data in the pipeline that holds the value of [*C*] only has a precision of {4.47} in the current implementation.
- Any values of $|\Delta r|^2$ that fall within the range of [1/4, 1/2) are treated as an exception and the computation will be performed by the software instead. A sample NAMD simulation shows that there are no situations that require lookup in the range of [1/4, 1/2) and there are approximately 0.5% of the computations that are in the range of [1/2, 1). Therefore, it is reasonable to define the exceptional case to offload the computation from the hardware.

- The lookup data of the functions [C] and [D] are fixed values for any type of simulation. They are unaffected by any variable such as the Ewald coefficient α.

The memory lookup and linear interpolation block in Figure 4.14 is shown in more detail in Figures 4.16 and 4.17 respectively. The overall hardware implementation of these two blocks is identical to the Ewald Direct Space compute engine counterpart. Refer to Section 4.5.1.3 for more details. The ZBT lookup module and function interpolation module take 13 and 15 clocks respectively. A total of 28 clocks are required to obtain the value of [C] and [D] as shown in Figure 4.14.

4.5.2.4 High-Level Data Flow and Operation

The overall data flow and operation of the Lennard-Jones compute engine is identical to the Ewald Direct Space compute engine. Figure 4.13 in the previous section has provided the picture of the hardware data flow and operation. The only difference is that the OPB interface expects a different number of inputs for each pair of particles, that is, the charge quantity is not required for Lennard-Jones computation but it requires the software computed term σ^{12}/σ^{6}. As discussed earlier, the values of the σ^{12}/σ^{6} term should be pre-computed and stored in a hardware lookup table if hardware resources are available. The values can be loaded into the ZBT memory and indexed by the types of particles in computation.

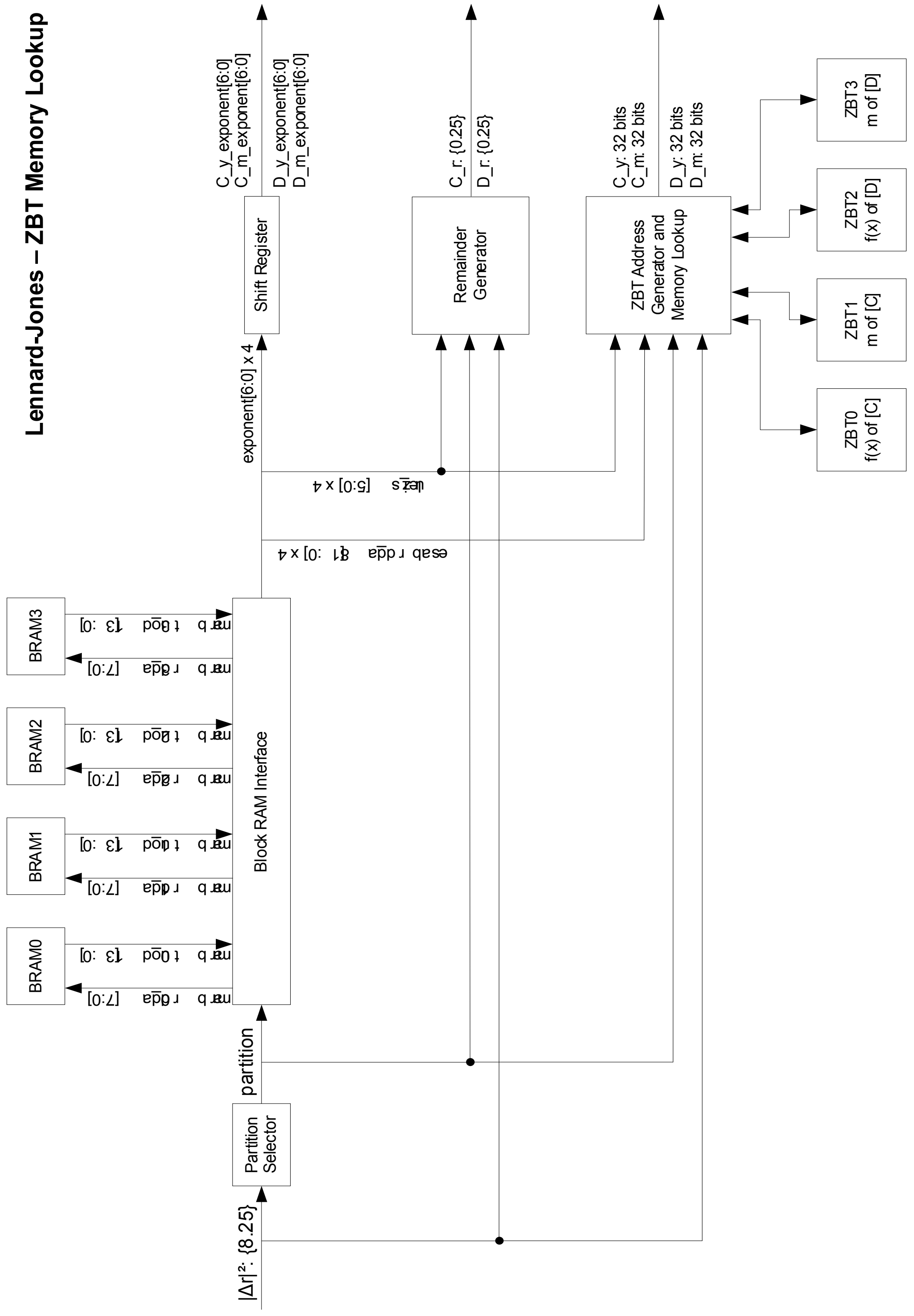


Figure 4.16 Lennard-Jones ZBT memory lookup module

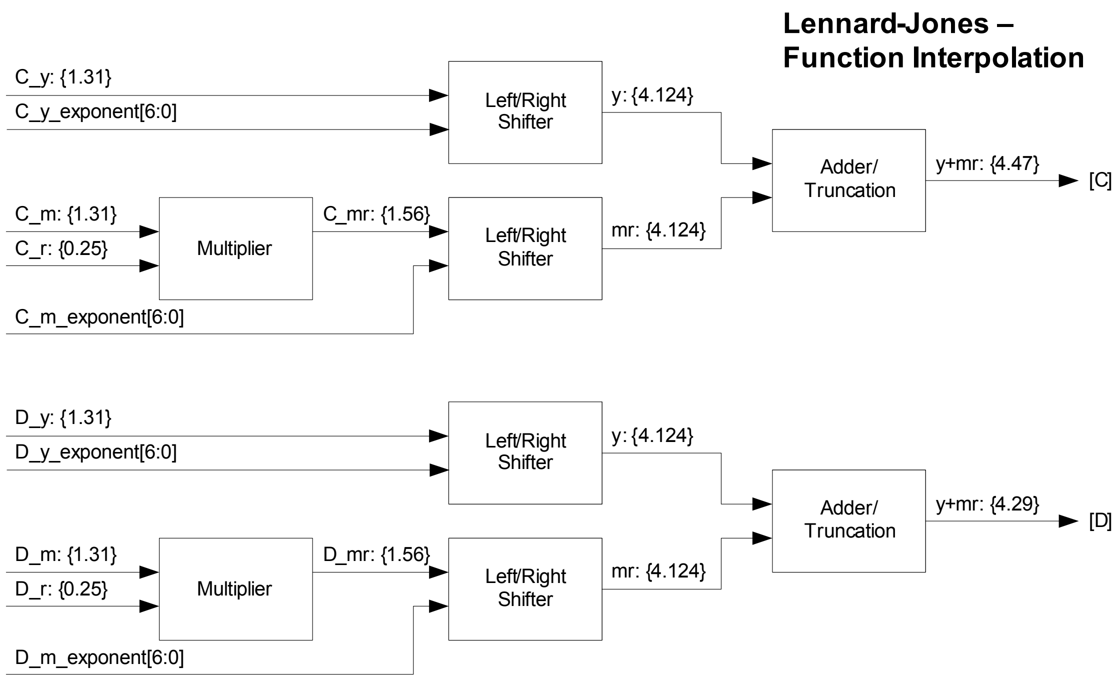


Figure 4.17 Lennard-Jones function interpolation

4.5.2.5 Hardware Utilization

The design is implemented on a Virtex-II XC-2V2000 FPGA on the Xilinx Multimedia Board. The device utilization summary of the Lennard-Jones compute engine design is given in Table 4.5. A detailed breakdown of the multiplier usage is show in Appendix E.

Device Components	Logic Utilization	
External IO Buffer	317 / 624	50%
Block RAMs 18Kbit	24 / 56	42%
Multiplier 18x18	52 / 56	92%
Slices	9258 / 10752	86%

Table 4.5 Lennard-Jones compute engine device utilization

4.6 Summary

This chapter has covered the hardware implementation details of both the Ewald Direct Space and Lennard-Jones compute engines. The designs are realized on the Virtex-II XC-2V2000 FPGA on the Xilinx Multimedia Board. The main hardware features on the FPGA that facilitate the design goal include: (i) the Microblaze soft processor provides an interface between the software driver and hardware pipeline engine, and (ii) the built-in multipliers and block RAMs used for the arithmetic unit. At the time this design is implemented, the Xilinx multimedia board is the only off-the-shelf device board that provides the amount of SRAM banks required to perform the function table lookups.

Future design should be implemented on a larger FPGA device that provides more multipliers and logic slices for the arithmetic operations to improve the precision. More built-in primitive multipliers allow larger multipliers to be built by concatenating individual ones together to accept wider data inputs, and subsequently improve the precision of intermediate results.

Further study should be done to determine the required lookup range needed to support the smallest possible value of $|\Delta r|^2$ (particle separation distance) in a molecular system. As discussed in Section 4.5.2.3, an overflow condition could result if the data precision in the compute engine is unable to hold the interpolated function value. On the other hand, hardware resources are wasted if additional bits are used to hold a larger interpolated function value for a particular lookup range that is not possible in a physical molecular system. The lookup range affects both the data precision in the compute engine and the size of the lookup table.

In the Ewald Direct Space compute engine, some multipliers are built using LUT logic because the 56 primitive multipliers are insufficient. As in the case of the Lennard-Jones compute engine, some multipliers are also built by LUT logic, one multiplier is being shared by three separate inputs, and some inputs to multipliers are forced to truncate the precision since the multipliers do not have a very wide input data width.

The ZBT memory banks provide 5 x 512k of entries and each entry is 32 bits wide. For a function with a 16k lookup table with ten partitions, only 31.25% of the memory is consumed. Hence, the ZBT memory still has unused resources to store a larger lookup table.

Moreover, the σ^{12}/σ^{6} input of the Lennard-Jones compute engine should be obtained from a hardware lookup table to reduce the workload on the software side as much as possible.

Chapter 5

HARDWARE MODELING AND SIMULATION

This chapter describes an approach for analyzing the arithmetic error of the hardware compute engines. It is followed by a sensitivity analysis to demonstrate how different magnitudes of computation error affect the total energy of a molecular system.

5.1 Numerical Analysis Overview

The hardware implementation of the compute engine is based on fixed-point arithmetic that includes: (i) multiplication, (ii) addition/subtraction, (iii) memory lookup of pre-computed fixed-point values, and (iv) truncation. Each of the arithmetic procedures injects error into the intermediate value and is subsequently carried forward to the final result. The purpose of this section is to provide an introductory level of numerical analysis to demonstrate how arithmetic error is generated [31].

When particle information is sent from the MD simulation software to the compute engine, a conversion from double precision floating point to a fixed-point format results in an error that is proportional to the precision being stored. In the compute engine design, the particle position is stored in the fixed-point precision of {7.25}. If double precision floating point data is truncated into a {7.25} fixed-point data, a maximum error of 2^{-25} occurs in the worst case if all the truncated bits are 1's. This error can be expressed as β and is defined as unit of last place (ulps). On the other hand, if the double precision floating point data is rounded off into {7.25} fixed-point data, a maximum error of $\beta/2$ would be injected.

Performing addition or subtraction on two numbers that already have an error of $\beta/2$ would have a combined effect of producing a maximum error of β. It can be concluded that to minimize the error in addition, subtraction, truncation, and memory lookup value storage, the general rule is to keep as many fractional bits as possible to reduce the value of β, that is, the ulps.

However, multiplication is slightly different and it can be demonstrated by an example. Suppose two numbers x and y are multiplied together and each has an error of β_x and β_y respectively after a truncation step. The multiplication can be expressed in Equation 5.1 and is expanded into Equation 5.2 using simple algebra.

$$(x+\beta_x)\times(y+\beta_y) \tag{5.1}$$

$$xy + x\beta_y + y\beta_x + \beta_x\beta_y \tag{5.2}$$

Equation 5.2 shows that the error incurred in the first step as a result of truncation is being magnified in the multiplication step. The second and third terms in Equation 5.2 are the dominant error term, and the last term is negligible. Therefore, the error being injected into the result is much larger in this case versus addition and subtraction. Again, the general rule is to reduce the value of β by reserving more fractional bits during the data type conversion.

This type of error analysis can be applied to the Ewald Direct Space and Lennard-Jones potential energy and force equations, Equations 4.1, 4.2, 4.7, and 4.8, to predict the upper bound error of a computation as a result of using fixed-point arithmetic and function table lookup. However, the algebraic procedures become tedious if the equations involve numerous steps of truncation, addition, multiplication, and table lookup. An alternative method to analyze the error is by a software simulation approach, which is discussed in the following section.

5.2 Hardware Modeling

The design implementation discussed in Chapter 4 has shown that the inputs to the compute engine have a fixed and restricted size as a result of limited hardware resources on the Virtex-II XC-2V2000 FPGA on the Xilinx Multimedia Board. The precision of the input variables in Tables 4.2 and 4.4 are presented here again in Table 5.1.

Input	Sign	Integer	Fraction	Fixed-Point
Coordinate	-	7	25	{7.25}
Charge	1	2	29	{2.29}
sig12_6	-	13	19	{13.19}

Table 5.1 Compute engine input variables

The simulation approach is to model the hardware functionality as a software program instead of performing simulation directly on the real hardware in FPGA. The software model is written in the SystemC hardware language, which is a collection of C++ libraries with hardware specific features such as the fixed-point data type. It also allows data conversion such as from double precision floating point to fixed-point format with the options of rounding and truncation. The randomization feature also facilitates a simulation that allows the programmer to generate millions of random input data with a specific constraint.

As a performance measure of the hardware, it is also of value to simulate and obtain the error with respect to different input precision and memory lookup table size. To perform such simulations, it is not feasible to re-design the compute engine in RTL and perform simulation with the real hardware because: (i) the existing FPGA has limited resources that do not allow a compute engine to consume any more multipliers and LUT logic, (ii) re-designing RTL with higher precision involves a substantial amount of work such as changing the pipeline stages, timing, and area, and (iii) performing simulation in real hardware would be slow because of the serial communication link of the current test configuration. Therefore, simulating the hardware performance using a software model is the better choice in terms of cost, development cycle, and efficiency.

5.3 Simulation

The purpose of this section is to perform simulation on the two compute engines, the Ewald Direct Space and Lennard-Jones, to obtain the computational error as a result of using a fixed-point format versus a double precision floating point data type. The setup of the simulation is to model the compute engines using the SystemC language as described in Section 5.2. The potential energy and force equations to be analyzed are: (i) Equations 4.1 and 4.2 for the Ewald Direct Space compute engine, and (ii) Equations 4.7 and 4.8 for the Lennard-Jones compute engine.

The software model has a particle randomizer that randomly produces particle positions in space in each of the ten partitions, [1/4, 1/2), [1/2, 1), …, and [128, 256). The potential and force equations are evaluated in each of the ten partitions with 500,000 pairs of particles generated by the randomizer using a default seed value. The error is obtained by comparing the result generated by the software model to that computed directly using a double precision floating point data type. The software model keeps track of the maximum absolute error in each partition and reports the result at the end of the simulation.

The software model implements the exact functionality of the hardware but the precision of the input variables can be selected by the simulation. For the potential energy and force equations, only the precisions of the particle coordinates and the memory lookup table size are varied. These variables are selected because the particle coordinates (x, y, and z) are used to calculate the term $|r^2|$ that is subsequently used as the index to the memory lookup table. It is of value to analyze how the precision of the particle coordinates and/or the memory table lookup size should be varied such that the accuracy of the final result can be maximized without allocating the hardware resources wastefully. The simulation results also demonstrate at what level the error saturates with different parameter settings as a result of the function lookup and interpolation.

The other input variables in Table 5.1, the charge quantity and sig12_6, are considered as constant values that have no effect on the accuracy of the function lookup and interpolation of the terms [*A*], [*B*], [*C*], and [*D*]. Therefore, the precision of the constant variables are not parameterized in the simulation. However, they would still have an effect on the final result of Equations 4.1, 4.2, 4.7, and 4.8 based on the discussion in Section 5.1. The choice of precisions of these constant inputs would depend on which of the error terms in the equation is selected to be the dominant error term. This analysis is not performed.

The following sub-sections will provide the simulation results of the various equations of the compute engines. For the rest of the discussion, the memory lookup size refers to the number of points per partition.

5.3.1 Particle Separation Distance

Figure 5.1 shows the absolute error of the particle separation distance $|r|^2$ with different precision. The compute engine uses the default precision of {7.25} to represent the particle coordinates. The software model increases the fractional precision by an additional 4, 8 and 16 bits and simulates the error based on the same set of randomized particles in each case.

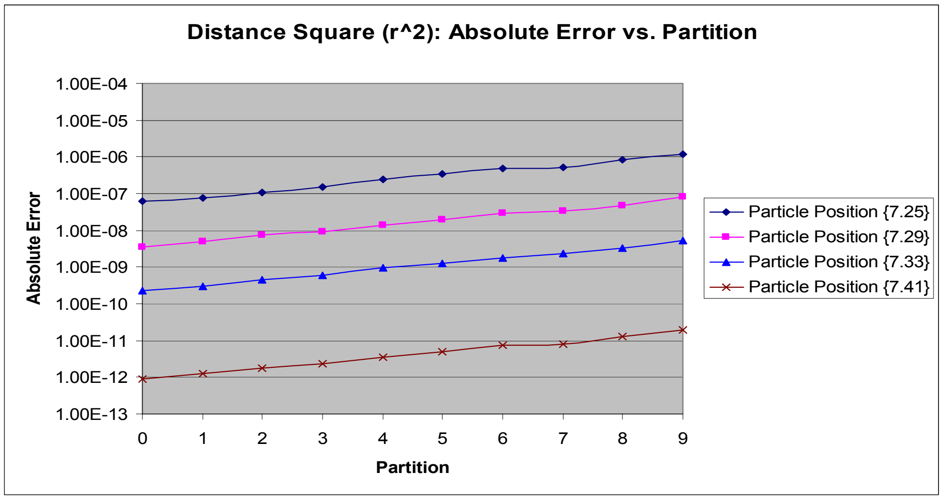


Figure 5.1 Absolute error of particle separation distance $|r|^2$

Figure 5.1 shows that the absolute error increases along with the separation distance based on Equation 5.2 showing that the error β is being magnified by the multiplications ($|r|^2 = |x|^2 + |y|^2 + |z|^2$). The graph also shows that the absolute error can be reduced roughly by a factor of 10^2 with every additional 8 bits allocated to the fractional part of the position coordinates.

The result generated by this simulation is a good reference guide for the process of constructing the function lookup table. For example, the uncertainty (error) in representing $|r|^2$ should be less than the spacing between two adjacent lookup points inside a particular interval. If the lookup table density is too high such that the uncertainty is larger than the distance between two adjacent lookup points, the interpolation step would become meaningless, or there is no need to interpolate. On the other hand, if the uncertainty of $|r|^2$ is relatively small between two adjacent points, the linear interpolation might not be able to produce a good approximation of the function.

In that case, the lookup point density should be increased. The simulation results in the following sections will address this issue further.

5.3.2 Ewald Direct Space Compute Engine

The potential energy and force equations of the Ewald Direct Space compute engine are simulated with coordinate precisions of {7.25}, {7.29}, {7.33}, and {7.41} for each of the memory lookup sizes of 1K, 4K, and 16K points. The absolute error of the potential energy equation is plotted with respect to the different precisions and table lookup sizes, and they are shown in Figures 5.2 to 5.8. The simulation results show that the absolute error of the force equation has the same trend as in the case of potential energy, therefore, the results are not shown here.

In Figures 5.2 to 5.5, each plot presents the error with respect to different memory lookup sizes for a fixed particle coordinate precision. In Figure 5.2, it shows that increasing the table lookup size from 1K to 4K or 16K points has little improvement in reducing the error for the first few partitions using a precision of {7.25}. It is of value to reduce the error in the first few partitions because these regions represent the small separation distances between particles where the potential energy is the strongest, and therefore, representing large values of potential energy.

Using the precision of {7.29} in Figure 5.3 shows that the error can be reduced approximately ten fold when the memory lookup table is increased from 1K to 4K points. Further increasing the memory lookup table to 16K points has little contribution to reducing the error in the overall potential energy.

Using the precision of {7.33} in Figure 5.4 shows that another ten fold error reduction can be achieved by using a memory lookup size of 16K points. By comparing the results in Figures 5.4 and 5.5, it shows that the error has negligible improvement when the precision is increased to {7.41}. That is, the error saturates if the particle coordinate precision is increased without the corresponding increase in memory lookup table size.

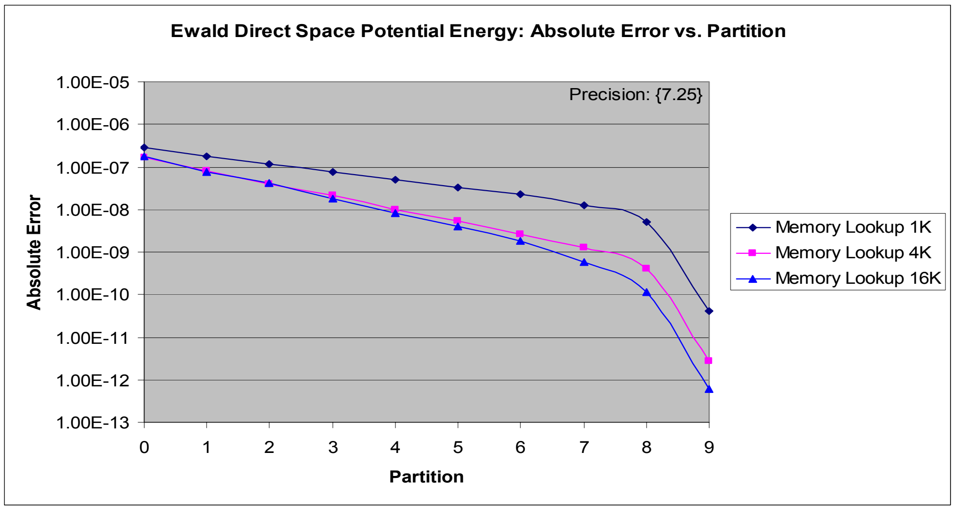


Figure 5.2 Absolute error of the Ewald Direct Space potential energy in {7.25} precision

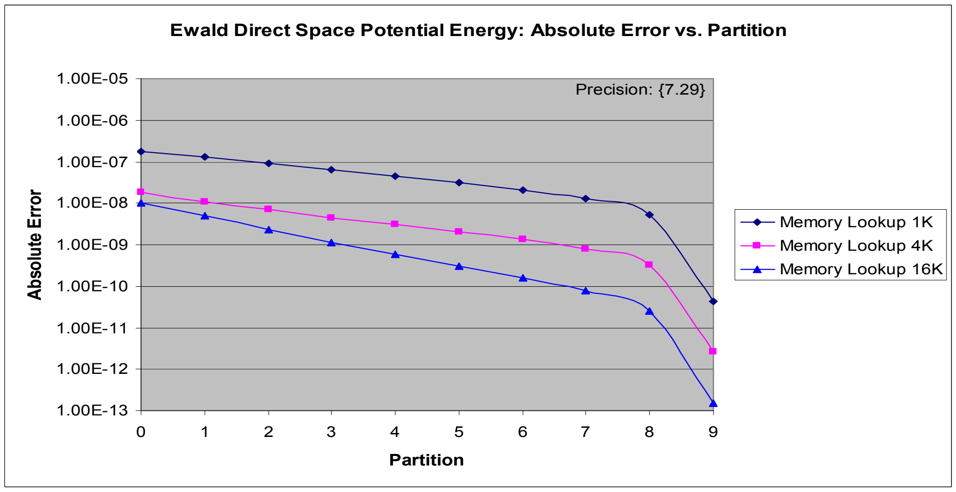


Figure 5.3 Absolute error of the Ewald Direct Space potential energy in {7.29} precision

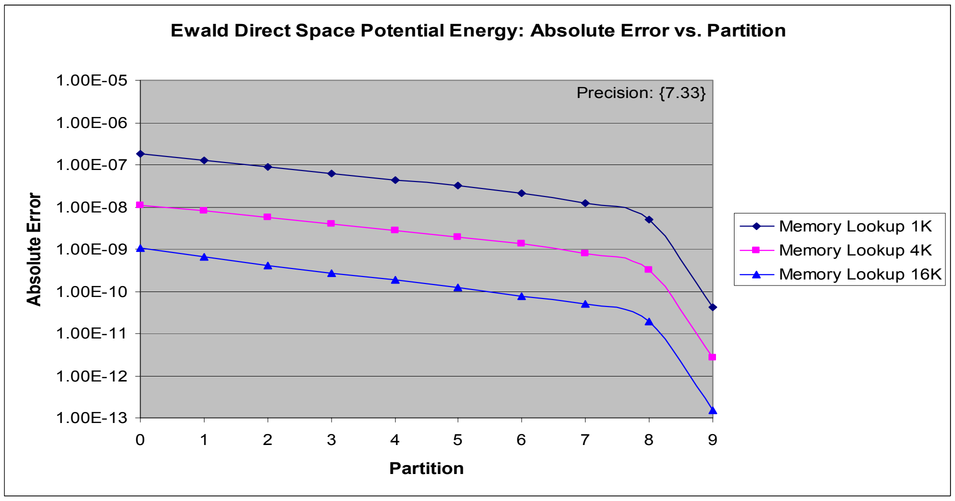


Figure 5.4 Absolute error of the Ewald Direct Space potential energy in {7.33} precision

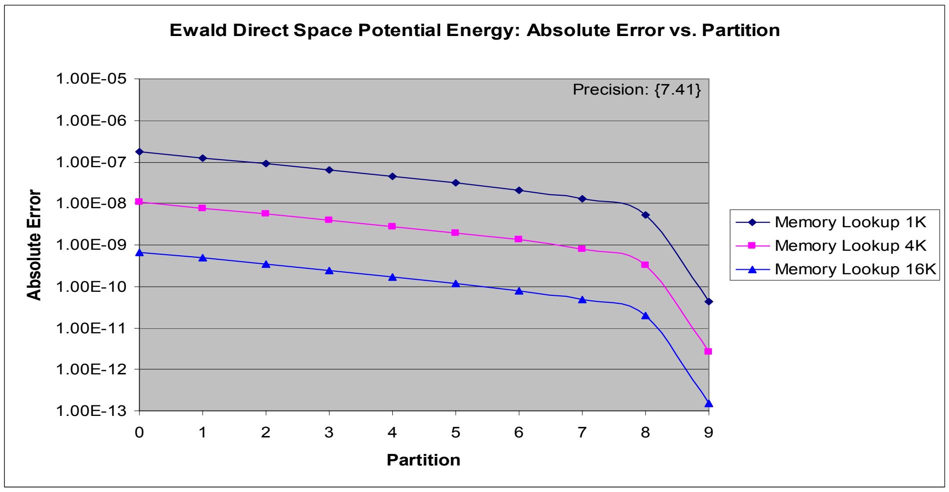


Figure 5.5 Absolute error of the Ewald Direct Space potential energy in {7.41} precision

Figures 5.6 to 5.8 present the same set of data with respect to the different particle coordinate precisions for a fixed memory lookup size. This uses a different point of view to demonstrate what particle coordinate precision should be chosen when the memory lookup table size is the system constraint. Figure 5.6 shows that any precisions beyond {7.25} provide insignificant improvement if the memory lookup size of 1K points is used. Figures 5.7 and 5.8 also demonstrate a similar trend that the error reduction is negligible when the precision is beyond a certain point for a particular memory lookup size. This indicates the fact that for a fixed lookup table, the interpolation step will not reduce the error any further even if the lookup index, $|r|^2$, is represented more precisely.

The overall simulation results shown in Figures 5.2 to 5.8 demonstrate that the error can be reduced roughly by a factor of ten consistently in the entire lookup table for: (i) every 4 bits of increase in the precision of the particle coordinates, and (ii) a four times increase in the memory lookup table size.

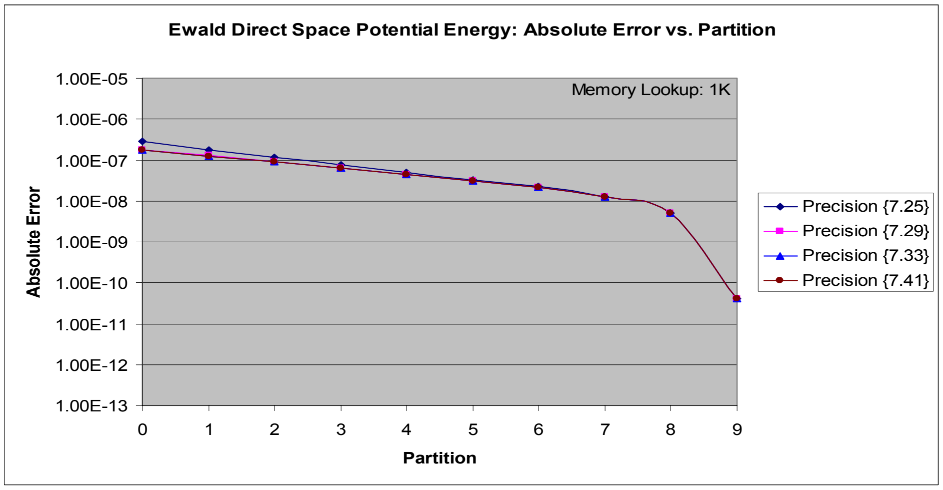


Figure 5.6 Absolute error of the Ewald Direct Space potential energy in 1K lookup

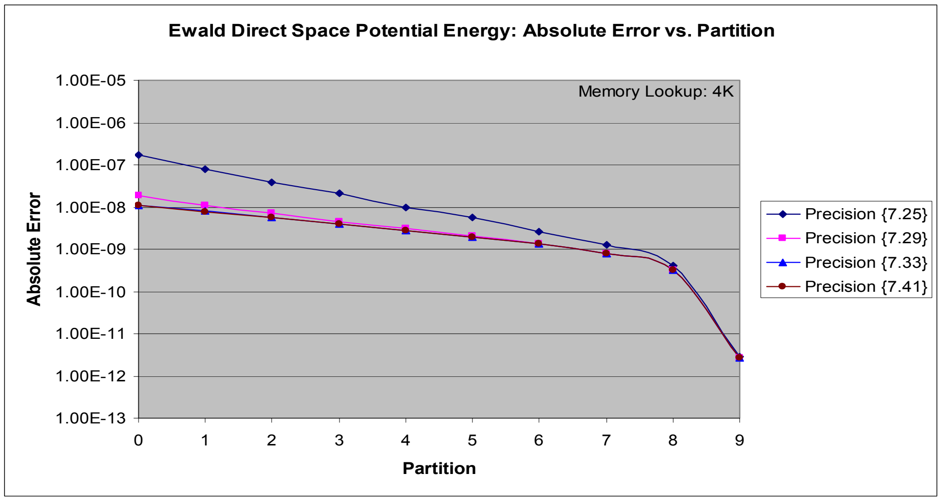


Figure 5.7 Absolute error of the Ewald Direct Space potential energy in 4K lookup

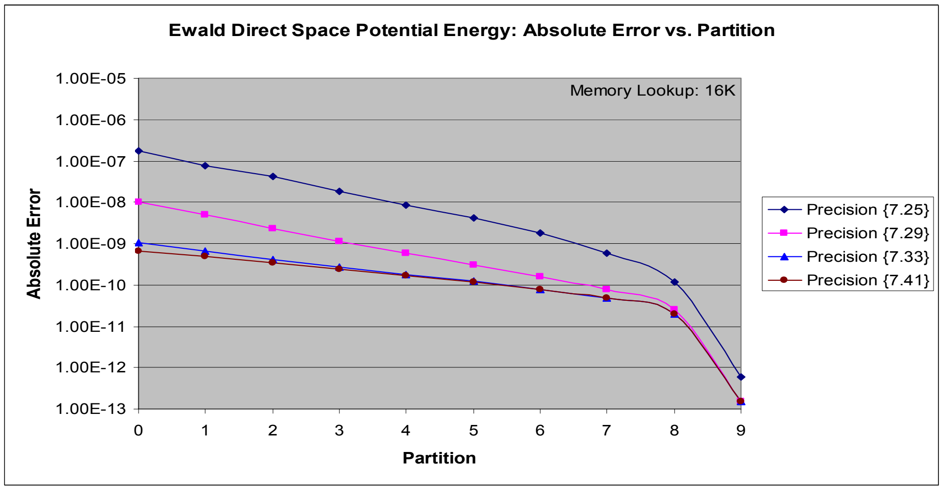


Figure 5.8 Absolute error of the Ewald Direct Space potential energy in 16K lookup

5.3.3 Lennard-Jones Compute Engine

The simulation for the Lennard-Jones compute engine is performed in a similar way as in the previous section. The same set of coordinate precisions and memory sizes are simulated and the absolute error of both the potential energy and force are obtained. The absolute error of the potential energy equation is plotted with respect to the different precisions and table lookup sizes, and they are shown in Figures 5.9 to 5.15. The simulation results also show that the absolute error of the force equation has the same trend as in the case of potential energy since the force is the derivative of the potential energy.

In Figures 5.9 to 5.12, each plot presents the error with respect to different memory lookup sizes for a fixed particle coordinate precision. Similar to the previous simulation, it can be seen in Figures 5.9 to 5.11 that the error is reduced by ten fold for every four extra bits used to represent the coordinate precision.

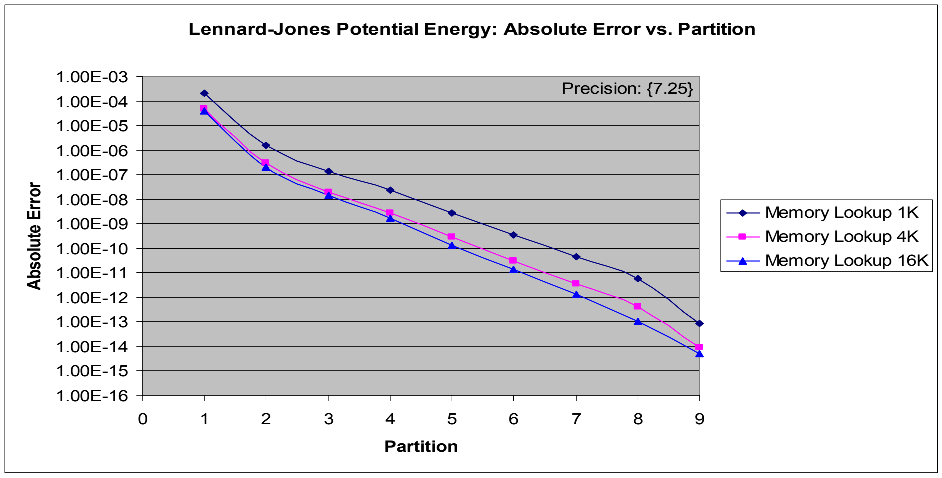


Figure 5.9 Absolute error of the Lennard-Jones potential energy in {7.25} precision

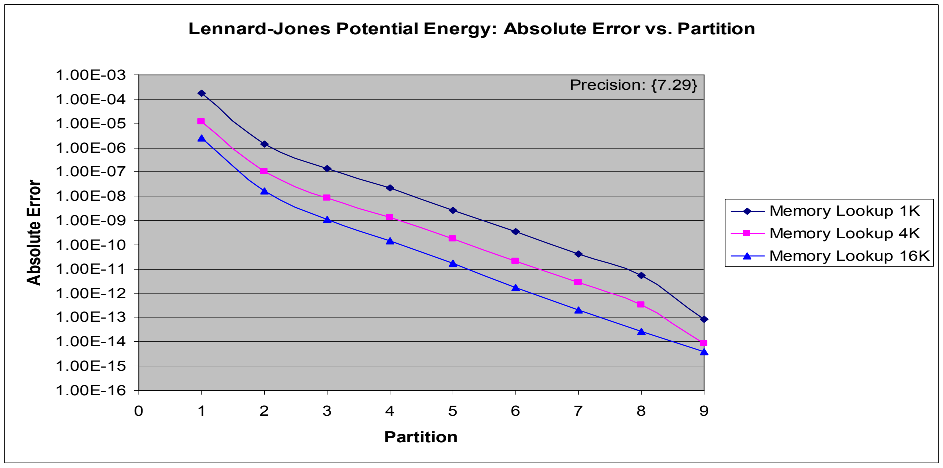


Figure 5.10 Absolute error of the Lennard-Jones potential energy in {7.29} precision

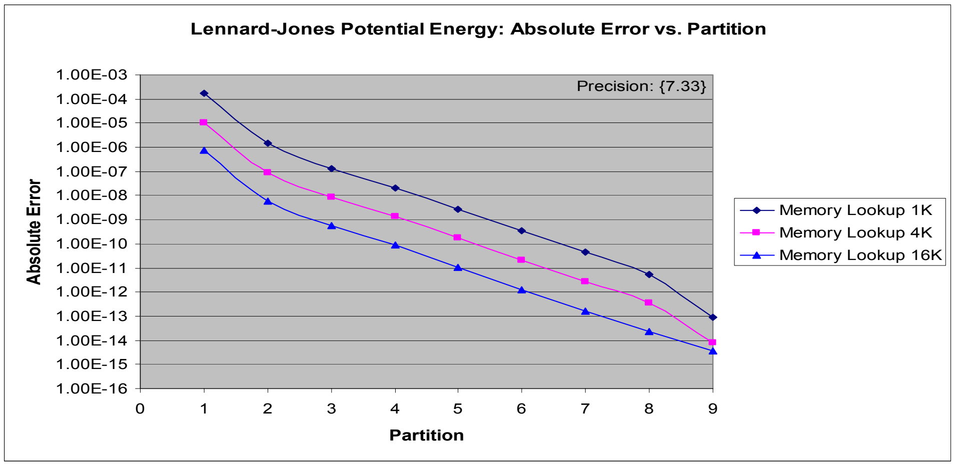


Figure 5.11 Absolute error of the Lennard-Jones potential energy in {7.33} precision

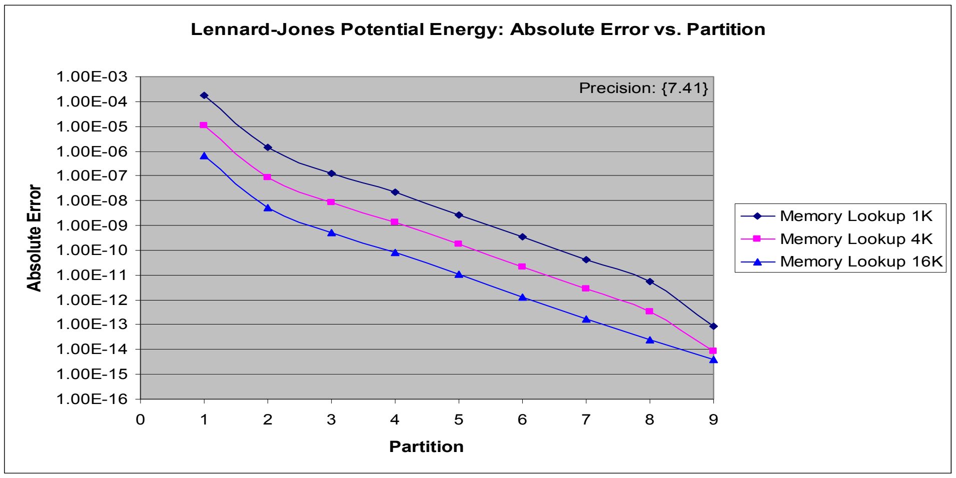


Figure 5.12 Absolute error of the Lennard-Jones potential energy in {7.41} precision

Figures 5.13 to 5.15 plot the error with respect to the different particle coordinate precisions for a fixed memory lookup size. The result obtained here is similar to the previous section showing that the error tends to be saturated beyond a certain precision for any particular memory size. As shown in Figure 5.13, the error obtained for different precisions have no significant difference from one another for the memory lookup size of 1K points. This result suggests that both the coordinate precision and memory lookup size should be increased accordingly to produce a more accurate result.

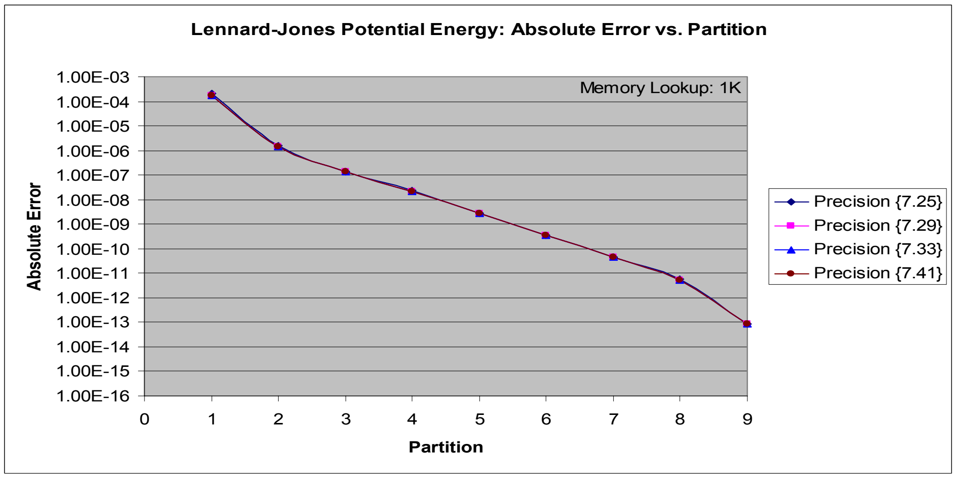


Figure 5.13 Absolute error of the Lennard-Jones potential energy in 1K lookup

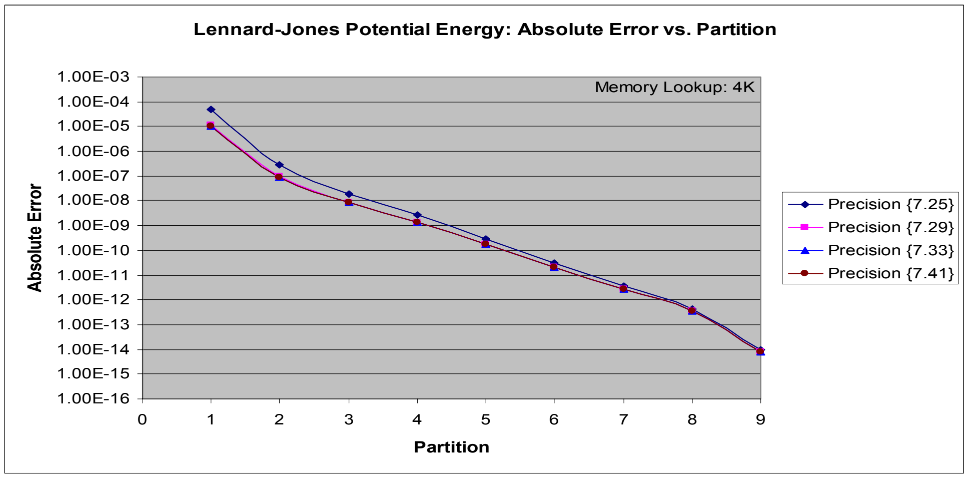


Figure 5.14 Absolute error of the Lennard-Jones potential energy in 4K lookup

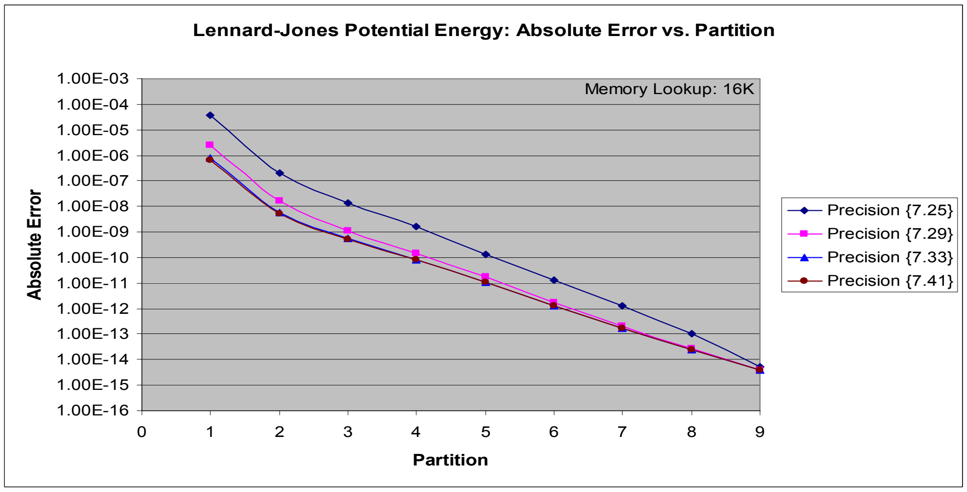


Figure 5.15 Absolute error of the Lennard-Jones potential energy in 16K lookup

Figure 5.14 shows that the error can be reduced approximately by a factor of five when the memory lookup size is increased to 4K points along with an increase in coordinate precision to {7.29}. The result shows no further improvement in error if the coordinate precision continues to increase. A similar trend can be seen in Figure 5.15 that the error can be reduced by a maximum of fifty times by using a memory lookup size of 16K and coordinate precision of {7.33}.

5.4 Sensitivity Analysis

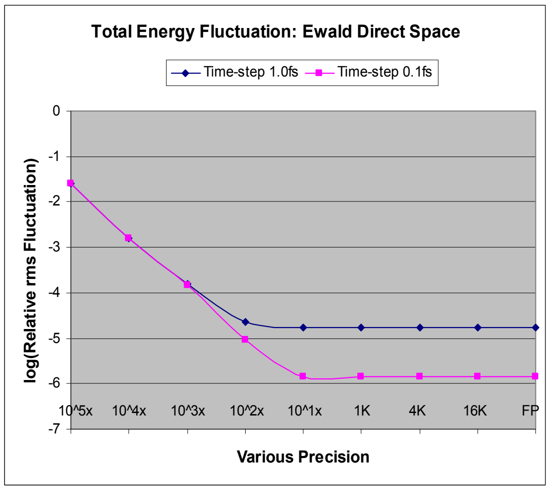
The purpose of this section is to analyze how the total energy of a MD simulation is affected as a result of the error generated by the compute engines discussed in Sections 5.3.2 and 5.3.3. A sample NAMD simulation is selected for performing this analysis. The configurations of the sample system are summarized in Appendix F. This is not a thorough analysis as it only uses a single small system for the study. However, the results suggest that this aspect should be investigated further.

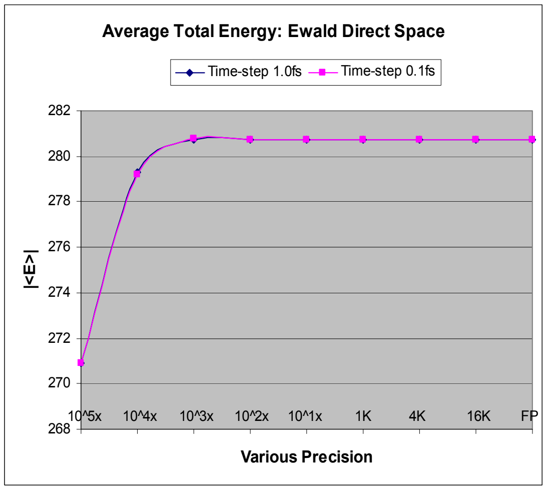
The error shown in Figures 5.5 and 5.12, which correspond to a particle coordinate precision of {7.41} and table lookup sizes of 1K, 4K, and 16K, are injected into the NAMD simulation as if the computation error is produced by the hardware engine. This is done by computing the $|r|^2$ term in the NAMD simulation and then approximating the error of the energy and force components based on the function value of the particular partition that includes $|r|^2$ in Figure 5.5 and 5.12. The polarity of the injected error is randomly chosen. Additional errors of different magnitudes are also extrapolated from the plots and applied to the simulation to provide an insight of how the overall simulation is further affected in the case that a very large error is produced by the hardware engine.

The stability of a MD simulation is evaluated in terms of the magnitude of the relative root mean square fluctuation in the total energy given by Equation 5.3 [9].

$$\frac{\sqrt{\left|\langle E^2 \rangle - \langle E \rangle^2\right|}}{\left|\langle E \rangle\right|} \qquad (5.3)$$

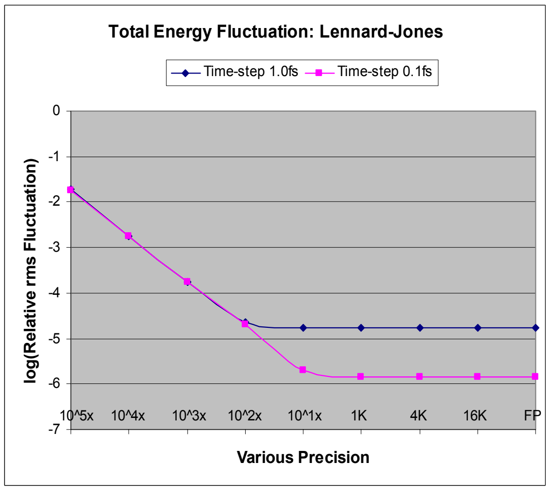
The NAMD simulation is run for 5,000 and 50,000 steps using a time-step size of 1fs and 0.1fs respectively. The results are presented in Figures 5.16 and 5.17.

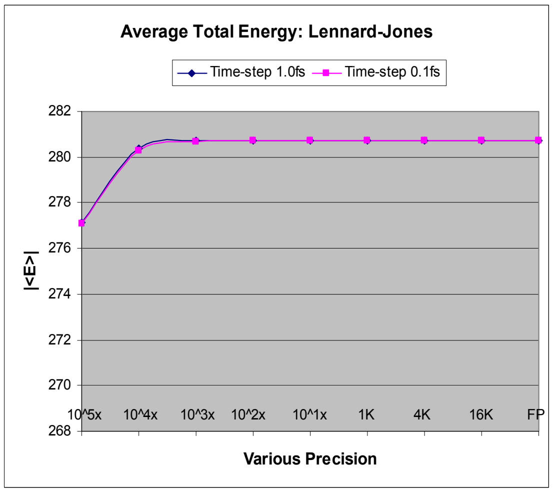




(a) (b)

Figure 5.16 System total energy with Ewald Direct Space computation error





(a) (b)

Figure 5.17 System total energy with Lennard-Jones computation error

Figure 5.16a shows the relative rms fluctuation in the total energy of the NAMD simulation by injecting error into the Ewald Direct Space computation. The point on the right hand side corresponds to the result obtained by simulation using double precision floating point, therefore, it represents the best possible accuracy. The next three points to the left correspond to particle coordinate precisions of {7.41} and table lookup size of 1K, 4K, and 16K. The rest of the points on the left hand side are obtained from simulations by injecting error that is 10 to 100,000 times of the error that corresponds to the 1K table lookup.

The results show that running an MD simulation using a 0.1fs time-step produces a simulation with less energy fluctuation. However, as the computation error increases, the relative rms fluctuation of the total energy is converging into the same magnitude disregarding the time-step used. Another important point observed is that lookup table sizes of 1K, 4K, or 16K are able to maintain the relative rms value of the simulation as low as that obtained by using double precision floating point. This can be observed from Figure 5.16a where the relative rms value decreases to the base value of -4.77 and -5.85 using a time-step of 1.0fs and 0.1fs respectively. This suggests that the magnitude of error injected is not large enough to have any significant impact on the overall simulation. Figure 5.16b shows the absolute average total energy deviates from the actual value as the computation error is increased when it should remain constant. The same trend can be observed from the NAMD simulation result, as shown in Figures 5.17a and 5.17b, by injecting error into the Lennard-Jones computation.

The results summarized in Figures 5.16 and 5.17 are based on injecting error with random polarity. This is to model the fact that the hardware compute engines could possibly produce both positive and negative error. It is important to note that using a different error injection scheme will produce a slightly different result. A separate simulation was performed to inject error that always has the same polarity as the computed energy and force components. Although the results show that the plots behave differently in the region with very large injected error, the same level of relative rms and average total energy is achieved with table lookup sizes of 1K, 4K, and 16K. Therefore, based on the above simulations, it can be summarized that the error injected by using the table lookup sizes of 1K, 4K, and 16K are negligible enough and would not cause any significant impact on the system total energy.

The overall simulation result with error injection suggests that additional error in computations acts as an external force that produces negative impacts on the system. From the result, it is observed that: (i) the rms fluctuation and the absolute average total energy of the system deviates from the actual value as the error increases, (ii) the different precisions adopted in the analysis in Sections 5.3.2 and 5.3.3 are adequate for this particular NAMD simulation to produce a result that is close to the original result generated by using double precision floating point, and (iii) the rms fluctuation begins to climb when the error is roughly ten times the error generated from using a 1K table lookup size. Therefore, the results conclude that even a 1K lookup table is adequate for simulation of the sample system.

The same behaviour in the total energy fluctuation is reported by Gu [16]. Their experiment shows that a 40-bit datapath in the design results in a similarly low energy fluctuation as a full 53-bit datapath. The total energy fluctuation begins to rise as the hardware datapath uses less precision. While the same behaviour in the total energy fluctuation is also observed in Figures 5.16a and 5.17a, it is not possible to compare the precision performance of this design with the one in Gu [16] because the molecular systems being simulated are different. However, the trends are very similar.

5.5 Summary

This chapter has covered the basic concept of numerical analysis that provides a basic idea to the designer of how error occurs and how error can be minimized in fixed-point arithmetic such as addition and multiplication. The simulation results for both the Ewald Direct Space and Lennard-Jones compute engines are also presented to show how the absolute error of the potential energy varies with respect to the different coordinate precisions and memory lookup sizes. The current hardware system, the Xilinx multimedia board, has provided five banks of ZBT memory with each bank of 512K in size. The memory size is therefore not a limiting factor based on the simulation discussed in this chapter.

For the particle coordinate precision, it would be convenient to select {7.25} if the design is to be implemented in the current system. If additional coordinate precision is required, future designs should leverage from the 36 bits of data available in the existing ZBT memory banks.

The overall simulation result shows that the error can be minimized effectively only if both the precision and table lookup size are increased accordingly.

The simulation of a sample molecular system shows that precisions less than that of double precision floating point can also achieve an energy fluctuation similar to the computation that uses double precision floating point. The sensitivity analysis that was performed suggests that further reducing the error in the compute engines is not necessary. This should be investigated more thoroughly.

Chapter 6

HARDWARE RESULTS

This chapter presents some of the performance parameters of a NAMD simulation and the hardware compute engines. The data will be summarized along with a discussion of areas of improvement.

6.1 Hardware Validation

The two hardware compute engine designs, Ewald Direct Space and Lennard-Jones, have been successfully integrated into the NAMD software through the serial communication bus as show in Figure 4.2. Simulations run at a very slow speed because the serial bus connection can only achieve a maximum speed of 57600 bps. Table 6.1 shows the approximate wall-clock time required to run one time-step of the sample NAMD simulation in different system configurations. The sample molecular system that was simulated is described in Appendix F.

System Description	Wall-clock Time Per Time-step (second)
Default NAMD simulation that is run entirely on software code.	1.2
Ewald Direct Space compute engine integrated into NAMD software.	445.2
Lennard-Jones compute engine integrated into NAMD software.	398.5

Table 6.1 NAMD simulation performance

The NAMD simulation that uses the Ewald Direct Space compute engine takes slightly more time because there are more input data being sent through the serial bus, compared to the case where the Lennard-Jones compute engine is used. The results show that it is impractical to run a simulation for thousands of steps with the hardware compute engines in the current test configuration. However, the system setup allows the hardware functionality to be validated.

6.2 Improving the Test Environment

The NAMD software is modified and integrated with the hardware compute engines as shown in Figure 4.2. The current method of communication is that a pair of particles is sent to the hardware for computation and the results are read back before the next pair of particles can be transmitted. This substantially reduces the hardware throughput because the hardware only handles one pair of particles at a time, and the processing speed is limited by the transmission speed of the serial communication port. Moreover, the pipelining capability is not utilized.

To improve the system throughput in this testing environment, both the software driver and hardware compute engine should be modified as follows:

- A cache memory is required between the OPB interface and the pipeline engine to store particle information and the computed energy and force results.
- A handshaking protocol should be defined to send data from the software driver to the cache memory without overflowing the particle information memory cache.
- The software driver should have the knowledge of the size of the memory cache and retrieve results as soon as possible to avoid possible conditions such as loss of data or causing the compute engine to stall.
- Additional control logic is required in the compute engine to fetch particle information from the cache memory when it is available and write results into a separate memory space.

6.3 Pipeline Improvement

The hardware utilization of the Ewald Direct Space and Lennard-Jones compute engine designs are provided in Sections 4.5.1.5 and 4.5.2.5 respectively. The place-and-route reports from the Xilinx EDK tool are shown in Table 6.2.

Compute Engine	Maximum Frequency (MHz)	Minimum Period (ns)
Ewald Direct Space	47.5	21.0
Lennard-Jones	43.6	22.9

Table 6.2 Compute engine operating frequency

The Xilinx Timing Analyzer shows the critical path in each of the designs involve the Microblaze soft processor and OPB bus interface, which are designed to run at the system clock frequency of 27 MHz. Therefore, these two components become the bottleneck in the overall compute engine operating frequency.

The place-and-route process was done separately on the isolated compute engine cores that exclude the Microblaze soft processor and OPB bus to show the performance of the computational units by themselves. Table 6.3 shows that the operating frequency of the compute engine cores can operate at approximately 80 MHz. Achieving 100 MHz with newer FPGAs should be quite feasible.

Compute Engine Core	Maximum Frequency (MHz)	Minimum Period (ns)
Ewald Direct Space	82.2	12.2
Lennard-Jones	80.0	12.5

Table 6.3 Compute engine core operating frequency

The timing reports show that the critical paths involve addition and comparator operations of very wide data buses. Tables 6.4 and 6.5 show the critical paths and the components of delay in each of the paths.

Source	top_lu_Mshreq_rsq_d4<25>_280
Destination	top_lu_for_y_zbt_addr_17
Data Path Delay	5.5 ns (Logic) + 6.6 ns (Routing) = 12.1 ns
Clock Path Skew	-0.1 ns

Table 6.4 Critical path of the Ewald Direct Space compute engine core

Source	top_accumulator_r_12_a_b/BU1600
Destination	top_accumulator_use_32
Data Path Delay	7.5 ns (Logic) + 5.0 ns (Routing) = 12.5 ns
Clock Path Skew	0.0 ns

Table 6.5 Critical path of the Lennard-Jones compute engine core

The results suggest that the compute engine cores can operate at a much higher clock frequency by decoupling the cores from the system components, such as the Microblaze soft processor and OPB bus. For future designs, it is desirable to implement a clock crossing boundary to separate the system components from the compute engine cores, such that the computations can be performed at a higher clock rate to increase the pipeline throughput, and utilize the available ZBT memory bandwidth, which can operate at a maximum clock speed of 130 MHz. To further improve the operating frequency of the compute engine cores, it is recommended that: (i) place-and-route constraints should be defined to reduce the routing delay, and (ii) combinational logic including the blocks created by the Xilinx CORE Generator should maximize the number of pipeline stages if hardware resources are available to reduce the logic delay. Therefore, when additional hardware resources become available in a larger FPGA device, it should be possible to improve the operating speed of the cores to run at a clock rate beyond 100 MHz. For example, the cores can be driven by a 108 MHz (27 MHz x 4) clock that can be conveniently generated by a Xilinx Digital Clock Manager (DCM).

The current architecture of the Ewald Direct Space compute engine can accept one pair of particles per clock cycle because the data path and ZBT memory lookup can process data on every clock edge. For the Lennard-Jones compute engine, as discussed in Section 4.5.2.2, the multiplier that computes the force components is shared among the three separate inputs due to limited hardware resources. Therefore, the pipeline can only support one pair of particles for every three clock cycles. Table 6.6 summarizes the performance of the pipeline engines.

Compute Engine	Latency (clocks)	Maximum Throughput (%): Current pipeline design / Pipeline design without hardware constraints
Ewald Direct Space	44	100 / 100
Lennard-Jones	59	33.33 / 100

Table 6.6 Latency and maximum throughput of the compute engine designs

6.4 Performance Analysis

In the envisioned complete MD system, the compute engines would be more directly coupled to the rest of the system, rather than to the NAMD software running on a processor. In such a design architecture, full pipeline throughput should then be achievable. Therefore, it is of value to analyze the pipelining architecture of the current design and compare its performance with previous implementations.

The Lennard-Jones pipeline unit in the current design is compared with the one implemented in the reconfigurable MD simulator [14] done at University of Toronto because there are more implementation details available. Table 6.7 presents the pipeline performance of the two designs.

Lennard-Jones Pipeline Unit	Latency (clocks)	Operating Frequency (MHz)
Implemented on the Transmogrifier 3 FPGA system	11	26.0
Implemented on the Xilinx Multimedia Board Virtex-II FPGA	59	80.0

Table 6.7 Performance of the Lennard-Jones pipeline units

Both designs are able to achieve the goal of processing one pair of particles per clock cycle. However, the performance data shows that the pipeline latency of the current design has increased significantly along with the improvement in the operating frequency and some additional design features over the previous implementation.

The Lennard-Jones pipeline unit implemented on the Transmogrifier 3 FPGA system requires 4 clock cycles to compute the $|r|^2$ as an index for the memory lookup. An additional 7 clock cycles are used to perform the memory lookup and interpolation to compute the potential and force components. In that design, all the addition and multiplication procedures and memory access are performed in a single clock cycle, that is, without pipelining.

The current Lennard-Jones pipeline design is shown in Figures 4.14 to 4.17. The design is different in a number of aspects:

- The first stage of the design uses 12 clock cycles to compute the $|r|^2$ term using pipelined adders and multipliers.
- The memory lookup and interpolation uses 28 clock cycles to evaluate the $[C]$ and $[D]$ terms. To support block floating point memory, the function lookup and interpolation require additional hardware such as: (i) internal block RAM lookup to retrieve parameters that associate with the different ZBT memory partitions, (ii) generation of the ZBT memory lookup address based on the partition size returned from the block RAM, (iii) ZBT memory access through the internal memory interface unit, and (iv) the interpolation stage uses pipelined adders and multipliers, and it uses a shifter to adjust the binary point of the values returned from the ZBT memory.
- The final stage that computes the potential and force components uses 19 clock cycles since it involves the σ terms to support the interactions of different types of particles. Several pipelined adders and multipliers are used in this stage.

It can also be seen that the current Lennard-Jones pipeline design achieves its speedup in the operating frequency mainly by pipelining the hardware logic, rather than leveraging from the improvement of the semiconductor technology of the FPGA.

The Transmogrifier 3 FPGA system did not implement the Ewald summation, however, the hardware architecture of the Ewald Direct Space calculation is very similar to that described in the Lennard-Jones design. Therefore, analyzing the Lennard-Jones pipeline unit is sufficient.

6.5 Summary

This chapter has provided several performance measures and estimations for the compute engine designs. More hardware resources would allow further pipelining of the combinational logic to increase the operating clock frequency. They would also avoid design limitations such as a multiplier that is being shared among different input sources, and causing a reduction in the pipeline throughput. Additional hardware along with design constraints should be used to improve the logic placement and routing to reduce path delay.

It is also demonstrated that an improvement in the operating frequency can be realized by pipelining the hardware logic, however, it comes with a negative impact of increasing the latency of the computations. It is of value for future research to study how the latency of the pipeline engine affects the overall system performance.

Chapter 7

CONCLUSIONS AND FUTURE WORK

This chapter summarizes the work that has been done in this thesis and provides the avenues for future research.

7.1 Conclusions

Molecular dynamics simulation using special-purpose computers has been an increasingly interesting research topic in the last decade for improving the understanding of biomolecular systems. Early research projects were implemented using ASIC technology. The explosive development in FPGA technology in recent years allows these special-purpose computers to be implemented with a shorter development cycle and at much less cost.

In this research work, the Ewald Direct Space and Lennard-Jones compute engines are implemented using the Xilinx Virtex-II XC-2V2000 FPGA on the Multimedia Board.

The hardware compute engines are successfully integrated with the NAMD simulation software.

It is shown that using the combination of fixed-point arithmetic and table lookup method can achieve high precision compared to computations that directly use double precision floating point. The computation error has been analyzed with different precisions and lookup table sizes. The sample NAMD simulations show that a similarly low relative rms fluctuation in the total energy can be achieved by using a coordinate precision of {7.41} with a table lookup size of 1K. The results also show that using a lookup table size beyond 1K is excessive in delivering computational accuracy.

The compute engine designs demonstrate that the block floating point memory is achievable. The approach of using block floating point memory for partitioning the lookup table has shown that the precision of the data stored in the memory can be improved enough to sufficient accuracy. It is also demonstrated that this methodology allows the memory usage to be optimized for the different types of functions.

The Xilinx design tools report that the Ewald Direct Space and Lennard-Jones hardware compute engines can operate at 47.5 MHz and 43.6 MHz respectively. The compute engine cores can operate at 82.2 MHz and 80.0 MHz respectively by decoupling them from the Microblaze soft processor and OPB bus. Achieving 100 MHz should be feasible with newer FPGAs.

7.2 Future Work

There are a number of areas of this work that require further research in the future. It is of value to analyze the computation error arising from various coordinate precisions, table lookup sizes, and interpolation orders. It is also important to further study how different types of NAMD simulations will respond and tolerate different magnitudes of computation error, which can lead to reduced hardware requirements. In particular, more thorough study should be done to confirm the results of the sensitivity analysis.

The current lookup tables in the ZBT memory are implemented for first order interpolation and a large amount of memory space is required. Future designs should investigate the computational precision using higher order interpolation such as the Chebyshev polynomial. It is also of value to examine the possibility of storing the lookup tables in the block RAMs instead of the external ZBT memory, which may be feasible with a higher order interpolation because the interpolation intervals do not have to be so fine.

Place-and-route results show that the current hardware compute engine cores can operate at about 80 MHz. It is estimated that the compute engine cores can be improved to operate at a minimum clock rate of 108 MHz with additional hardware resources and place-and-route constraints. With such design performance, the available ZBT memory bandwidth can be utilized further.

APPENDIX A – Standard Ewald Direct Space

This appendix shows the derivation of the Ewald Direct Space electrostatic potential energy equation in an infinite lattice system with *N* particles [18].

The potential energy at a particular point in space is the interaction as a result of the Gaussian charge cloud. The potential energy can be derived from the following Poisson's equation using the spherical symmetry of the Gaussian charge cloud:

$$-\frac{1}{r}\frac{\partial^2 r\phi_{Gauss}(r)}{\partial r^2} = 4\pi\rho_{Gauss}(r) \tag{A.1}$$

$$p_{Gauss}(r) = q_i\alpha^3\exp(-\alpha^2 r^2)/\sqrt{\pi^3} \tag{A.2}$$

The first partial integration yields:

$$-\frac{\partial r\phi_{Gauss}(r)}{\partial r} = \int_{\infty}^{r} 4\pi r\rho_{Gauss}(r)dr \tag{A.3}$$

$$= -2\pi q_i\left(\frac{\alpha^2}{\pi}\right)^{3/2}\int_{r}^{\infty} r\exp(-\alpha^2 r^2)dr \tag{A.4}$$

$$= -2q_i\left(\frac{\alpha^2}{\pi}\right)^{1/2}\exp(-\alpha^2 r^2) \tag{A.5}$$

The second partial integration yields:

$$r\phi_{Gauss}(r) = 2q_i\left(\frac{\alpha^2}{\pi}\right)^{1/2}\int_{0}^{r}\exp(-\alpha^2 r^2)dr \tag{A.6}$$

$$\phi_{Gauss}(r) = \frac{q_i}{r} erf(\alpha r) \quad \text{(A.7)}$$

where erf(x) is the error function.

Finally, the electrostatic potential can be derived as a result of the point charge q_i surrounded by a Gaussian distribution with net charge of $-q_i$:

$$\phi_{short-range}(r) = \frac{q_i}{r} - \frac{q_i}{r} erf(\alpha r) \quad \text{(A.8)}$$

$$= \frac{q_i}{r} erfc(\alpha r) \quad \text{(A.9)}$$

Therefore, the total electrostatic potential energy in a system of N particles over the infinite lattice is given by Equation 2.8 and it is also shown in Equation A.10:

$$U^r = \frac{1}{2} \sum_n{}' \sum_{ij}^{N} q_i q_j \frac{erfc(\alpha r_{ij,n})}{r_{ij,n}} \quad \text{(A.10)}$$

The corresponding force equation is shown in Equation A.11:

$$F_p^r(i) = q_i \sum_{j=1,j\neq i}^{N} \sum_n q_j \frac{(r_{ij,n})_p}{r_{ij,n}^3} \left\{ erfc(\alpha r_{ij,n}) + \frac{2\alpha}{\sqrt{\pi}} r_{ij,n} \exp(-(\alpha r_{ij,n})^2) \right\} \quad \text{(A.11)}$$

APPENDIX B – Standard Ewald Reciprocal Space

The purpose of this appendix is to further simplify the Ewald Reciprocal Space electrostatic potential energy, Equation 2.9, to transform the double loop in the reciprocal sum into a single sum [2]. Equation 2.9 is also shown in Equation B.1:

$$U^m = \frac{1}{2\pi V}\sum_{ij}^{N} q_i q_j \sum_{m\neq 0} \frac{\exp(-(\pi m/\alpha)^2 + 2\pi i m\cdot(r_i - r_j))}{m^2} \tag{B.1}$$

Expanding the term and simplifying yields:

$$U^m = \frac{1}{2\pi V}\sum_{m\neq 0} \frac{\exp(-(\pi m/\alpha)^2)}{m^2} \sum_{ij}^{N} q_i q_j \cos(2\pi m\cdot r_{ij}) \tag{B.2}$$

Equation B.2 can be further simplified using a trigonometric identity:

$$U^m = \begin{array}{l} \frac{1}{2\pi V}\sum_{m\neq 0} \frac{\exp(-(\pi m/\alpha)^2)}{m^2} \times \\ \sum_{ij}^{N} q_i q_j \{\cos(2\pi m\cdot r_i)\cos(2\pi m\cdot r_j) + \sin(2\pi m\cdot r_i)\sin(2\pi m\cdot r_j)\} \end{array} \tag{B.3}$$

$$U^m = \begin{array}{l} \frac{1}{2\pi V}\sum_{m\neq 0} \frac{\exp(-(\pi m/\alpha)^2)}{m^2} \times \\ \left\{ \left[\sum_{i=1}^{N} q_i \cos(2\pi m\cdot r_i)\right]^2 + \left[\sum_{i=1}^{N} q_i \sin(2\pi m\cdot r_i)\right]^2 \right\} \end{array} \tag{B.4}$$

And finally by defining:

$$S(m) = \sum_{k=1}^{N} q_k \exp(2\pi i m \cdot r_k) \tag{B.5}$$

The Ewald reciprocal sum can be expressed as Equation B.6:

$$U^m = \frac{1}{2\pi V} \sum_{m \neq 0} \frac{\exp(-(\pi m / \alpha)^2)}{m^2} S(m) S(-m) \tag{B.6}$$

$$U^m = \frac{1}{2\pi V} \sum_{m \neq 0} \frac{\exp(-(\pi m / \alpha)^2)}{m^2} \left| S(m) \right|^2 \tag{B.7}$$

Therefore, the above algebraic procedures have transformed the double sum over *i* and *j* of order $O(N^2)$ into two single sums of order $O(N)$, and finally into a single sum as in Equation B.7, that is more efficient. The corresponding force equation is shown in Equation B.8:

$$F_p^m = \frac{2q_i}{L} \sum_{j=1, j \neq i}^{N} \sum_{m \neq 0} \frac{m_p}{m^2} \exp\left(-\left(\frac{\pi m}{\alpha L} \right)^2 \right) \sin\left(\frac{2\pi}{L} m \cdot r_{ij} \right) \tag{B.8}$$

APPENDIX C – Particle Mesh Ewald

The standard Ewald Direct Space Equation A.10 is reused for the PME method with the use of a cut-off radius. The standard Ewald Reciprocal Space Equation B.7 is computed using 3D-FFT as discussed in [22], and is shown in the following:

$$U^{m} = \frac{1}{2\pi V}\sum_{m\neq 0}\frac{\exp(-(\pi m/\alpha)^2)}{m^2}S(m)S(-m) \tag{C.1}$$

where $S(m)$ is the structure factor defined in Equation B.5. This structure factor can be approximated by:

$$\begin{aligned} S(m) \approx \tilde{S}(m) &= \sum_{k_1,k_2,k_3} Q(k_1,k_2,k_3)\exp\left(2\pi i\left(\frac{m_1k_1}{K_1}+\frac{m_2k_2}{K_2}+\frac{m_3k_3}{K_3}\right)\right) \\ &= F(Q)(m_1,m_2,m_3) \end{aligned} \tag{C.2}$$

where $F(Q)$ is the 3-D FFT of the charge matrix Q. The simulation box is partitioned into uniform grids of dimension $K_1\times K_2\times K_3$, and the Q matrix is a three dimensional matrix obtained by interpolating the point charges on that grid. The reciprocal energy can be approximated by Equation C.3:

$$\tilde{U}^{m} = \frac{1}{2\pi V}\sum_{m\neq 0}\frac{\exp(-(\pi m/\alpha)^2)}{m^2}F(Q)(m)F(Q)(-m) \tag{C.3}$$

It is then expressed as a convolution after some algebraic procedures:

$$\tilde{U}^{m} = \frac{1}{2}\sum_{m_1=0}^{K_1-1}\sum_{m_2=0}^{K_2-1}\sum_{m_3=0}^{K_3-1} Q(m_1,m_2,m_3)(\psi_{rec} * Q)(m_1,m_2,m_3) \qquad \text{(C.4)}$$

where Ψ_{rec} is the reciprocal pair potential and the $*$ indicates a convolution. The PME direct sum uses cut-off radius to limit the computation with order of $O(N)$, and the reciprocal sum is computed using 3D-FFT with an overhead that grows at $N\log(N)$. Thus, the overall PME method is on the order of $O(N\log(N))$.

APPENDIX D – Software Driver C Code

The MD simulation software running on the Linux station is communicating with the Microblaze soft processor through the serial bus interface as shown in Figure 4.2. The software driver C code running on the Linux station and the Microblaze soft processor are shown in Figures D.1 and D.2 respectively.

```
// ----------- //
// serialcon.h //
// ----------- //
#ifndef SERIALCON_H
#define SERIALCON_H

#define UINT unsigned int

class serialcon
{
    public:
        serialcon(char * deviceFile);
        ~serialcon();
        void sendUINT(UINT &data);
        void recvUINT(UINT &data);
        void waitForAck();
        void sendAck();

    private:
        int fileDesc;
};

#endif

// ------------- //
// serialcon.cpp //
// ------------- //
#include "serialcon.h"
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>

serialcon::serialcon(char * deviceFile)
{
    fileDesc = open(deviceFile, O_RDWR | O_NOCTTY);

    if(fileDesc == -1)
    throw;

    fcntl(fileDesc, F_SETFL, 0);
    struct termios options;
    tcgetattr(fileDesc, &options);
    cfsetispeed(&options, B57600); //Input speed
    cfsetospeed(&options, B57600); //Output speed
    options.c_cflag |= (CLOCAL | CREAD);
    options.c_cflag &= ~PARENB;
    options.c_cflag &= ~CSTOPB;
    options.c_cflag &= ~CSIZE;
    options.c_cflag |= CS8;
    options.c_cflag &= ~CRTSCTS;
    options.c_iflag &= ~(IXON | IXOFF | IXANY);
    cfmakeraw(&options);
    tcsetattr(fileDesc, TCSANOW, &options);
}

serialcon::~serialcon()
{
    close(fileDesc);
}

void serialcon::sendUINT(UINT &data)
{
    if(write(fileDesc, &data, sizeof(UINT)) <= 0)
    throw;
}

void serialcon::recvUINT(UINT &data)
{
    if(read(fileDesc, &data, sizeof(UINT)) <= 0)
    throw;
}

void serialcon::waitForAck()
{
    char aChar = '\0';

    while(aChar != (char)0xFF)
    {
        if(read(fileDesc, &aChar, 1) <= 0)
        throw;
    }
}

void serialcon::sendAck()
{
    char aChar = (char)0xFF;

    if(write(fileDesc, &aChar, 1) <= 0)
    throw;
}
```

Figure D.1 Linux station software driver C code

```
// ----------- //
// serialcon.h //
// ----------- //
```

```
#ifndef SERIALCON_H
#define SERIALCON_H

#define UINT unsigned int

void sendUINT(UINT *data);
void recvUINT(UINT *data);
void waitForAck();
void sendAck();

#endif

// ------------- //
// serialcon.cpp //
// ------------- //
#include "xuartlite_l.h"
#include "xparameters.h"
#include "serialcon.h"

void sendUINT(UINT *data)
{
    char * byte;

    for(byte = ((char*)data) + 3; byte >= (char*)data; byte--)
    {
        XUartLite_SendByte(STDOUT_BASEADDRESS, *byte);
    }
}

void recvUINT(UINT *data)
{
    char * byte;

    for(byte = ((char*)data) + 3; byte >= (char*)data; byte--)
    {
        *byte = XUartLite_RecvByte(STDIN_BASEADDRESS);
    }
}

void waitForAck()
{
    char aChar = '\0';

    while(aChar != (char)0xFF)
    {
        aChar = XUartLite_RecvByte(STDIN_BASEADDRESS);
    }
}

void sendAck()
{
    XUartLite_SendByte(STDOUT_BASEADDRESS, (char)0xFF);
}
```

Figure D.2 Microblaze soft processor driver C code

APPENDIX E – Hardware Utilization

Table E.1 summarizes the multiplier usage of the Ewald Direct Space compute engine. Some of the multipliers are built by LUTs because there are not enough built-in multipliers to achieve the design goal. The corresponding functionality can be found in Figures 4.6 and 4.12, and the Microblaze soft processor also consumes three multipliers. The total number of built-in multipliers consumed is 55 out of the 56 available in the device.

Functionality	# of Instances	Multiplier Size (bits)	# of Multipliers per Instance	Total # of Multipliers
$\lvert\Delta x\rvert^2$ $\lvert\Delta y\rvert^2$ $\lvert\Delta z\rvert^2$	3	29 x 29	4	12
$\lvert q_i \times q_j\rvert$	1	31 x 31	4	4
$\lvert q_i q_j \times \Delta x\rvert$ $\lvert q_i q_j \times \Delta y\rvert$ $\lvert q_i q_j \times \Delta z\rvert$	3	33 x 29	4	12
$\lvert q_i q_j\rvert \times [A]$ $\lvert q_i q_j \Delta x\rvert \times [B]$ $\lvert q_i q_j \Delta y\rvert \times [B]$ $\lvert q_i q_j \Delta z\rvert \times [B]$	4	33 x 51	6	24
Microblaze	1	18 x 18	3	3
$m \times r$ (interpolation)	2	25 x 32	LUTs	LUTs

Table E.1 Multipliers usage of the Ewald Direct Space compute engine

Table E.2 summarizes the multiplier usage of the Lennard-Jones compute engine. Some of the multipliers are also built by LUTs since there is a limitation in hardware resources. The corresponding functionality can be found in Figures 4.14, 4.15, and 4.16. The total number of built-in multipliers consumed is 52 out of the 56 available in the device.

Functionality	# of Instances	Multiplier Size (bits)	# of Multipliers per Instance	Total # of Multipliers
$\lvert\Delta x\rvert^2$ $\lvert\Delta y\rvert^2$ $\lvert\Delta z\rvert^2$	3	29 x 29	4	12
$[C]^2$	1	51 x 51	9	9
$[C]^2$ x sig12_6	1	55 x 32	8	8
$[D]$ x $\lvert\Delta x\rvert$ $[D]$ x $\lvert\Delta y\rvert$ $[D]$ x $\lvert\Delta z\rvert$	3	33 x 29	4	12
(2 x sig12_6 x $[C]^2$ – $[C]$) x $[D]$ x $\lvert\Delta\rvert$	1	33 x 62	8	8
Microblaze	1	18 x 18	3	3
m x r (interpolation)	2	25 x 32	LUTs	LUTs

Table E.2 Multipliers usage of the Lennard-Jones compute engine

APPENDIX F – NAMD Simulation System

A sample NAMD simulation is provided as part of the software code for testing purpose. This system is used for the sensitivity analysis in Section 5.4 and also for the hardware validation discussed in Section 6.1. Table F.1 summarizes the configurations of this sample simulation that are related to the course of this work.

Parameter	Configuration
Number of Particles	66
PME Tolerance (refer to Equation 3.1)	1.0×10^{-6}
PME Cut-off Radius (Å)	16
Lennard-Jones Cut-off Radius (Å)	16
Simulation Box Size (Å)	100 x 100 x 100

Table F.1 Configurations of the NAMD sample simulation

REFERENCE

[1] M. P. Allen, D. J. Tildesley, *Computer Simulation of Liquids*, Oxford University Press, 1987.

[2] A. Y. Toukmaji, J. A. Board Jr., *Ewald summation techniques in perspective: a survey*, Computer Physics Communications, p.73-92, 1996.

[3] F. Allen et al., *Blue Gene: A vision for protein science using a petaflop supercomputer*, IBM Systems Journal, Volume 40, No.2, p.310-327, 2001.

[4] F. Toshiyuki, T. Makoto, M. Junichiro, E. Toshikazu, S. Duiichiro, *A Highly Parallelized Special-Purpose Computer For Many-Body Simulations with an Arbitrary Central Force: MD-GRAPE*, Astrophysical Journal, v.468, p.51-61, 1996.

[5] Y. Komeiji, M. Uebayasi, R. Takata, A. Shimizu, K. Itsukashi, M. Taiji, *Fast and Accurate Molecular Dynamics Simulation of a Protein Using a Special-Purpose Computer*, Journal of Computational Chemistry, Volume 18, No.12, p.1546-1563, 1997.

[6] P. P. Ewald, *Die Berechnung optischer und elektrostatischer Gitterpotentiale*, Annalen der Physik, v.64, p.253-287, 1921.

[7] R. W. Hockney, J. W. Eastwood, *Computer Simulation Using Particles*, McGraw-Hill, 1981.

[8] S. Toyoda, H. Miyagawa, K. Kitamura, T. Amisaki, E. Hashimoto, H. Ikeda, A. Kusumi, N. Miyakawa, *Development of MD Engine: High-Speed Accelerator with Parallel Processor Design for Molecular Dynamics Simulations*, Journal of Computational Chemistry, Volume 20, No.2, p.185-199, 1999.

[9] T. Amisaki, T. Fujiwara, A. Kusumi, H. Miyagawa, K. Kitamura, *Error Evaluation in the Design of a Special-Purpose Processor That Calculates Nonbonded Forces in Molecular Dynamics Simulations*, Journal of Computational Chemistry, Volume 16, No.9, p.1120-1130, 1995.

[10] *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std.754-1985, IEEE: New York, 1985.

[11] IBM BlueGene/L Team, *An Overview of the BlueGene/L Supercomputer*, Proc. ACM Supercomputing Conference, 2002.

[12] Top 500 Supercomputer Sites, November 2004.
http://www.top500.org/lists/2004/11
[Online accessed June 2005]

[13] *Japan Designers Shoot for Supercomputer on a Chip*, August 2004. http://news.com.com/Japan+designers+shoot+for+supercomputer+on+a+chip/2100-7337_3-5322558.html
[Online accessed June 2005]

[14] N. Azizi, I. Kuon, A. Egier, A. Darabiha, P. Chow, *Reconfigurable Molecular Dynamics Simulator*, IEEE Symposium on Field-Programmable Custom Computing Machines, p.197-206, April 2004.

[15] M. van Ierssel, D. Galloway, P. Chow, J. Rose, *The Transmogrifier-3a: Hardware and Software for a 3 Million Gate Rapid Prototyping System*, Micronet Annual Workshop, April 2001.

[16] Y. Gu, T. VanCourt, M. C. Herbordt, *FPGA Acceleration of Molecular Dynamics Computations*, To appear: Proceedings of Field Programmable Logic and Applications, August 2005.

[17] R. Pomès, Private communication, Department of Biochemistry, University of Toronto, May 2004.

[18] D. Frenkel, B. Smit, *Understanding Molecular Simulation*, Academic Press, 1996.

[19] S. W. de Leeuw, J. W. Perram, E. R. Smith, *Simulation of Electrostatic Systems in Periodic Boundary Conditions I*, Preceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences, Volume 373, Issue 1752, p.27-56, October 1980.

[20] S. W. de Leeuw, J. W. Perram, E. R. Smith, *Simulation of Electrostatic Systems in Periodic Boundary Conditions II*, Preceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences, Volume 373, Issue 1752, p.57-66, October 1980.

[21] S. W. de Leeuw, J. W. Perram, E. R. Smith, *Simulation of Electrostatic Systems in Periodic Boundary Conditions III*, Preceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences, Volume 388, Issue 1794, p.177-193, July 1983.

[22] T. Darden, D. York, L. Pedersen, *Particle mesh Ewald: An Nlog(N) method for Ewald sums in large systems*, Journal of Chemical Physics, Volume 98, No. 12, June 1993.

[23] J. C. Phillips, G. Zheng, S. Kumar, L. V. Kale, *NAMD: Biomolecular Simulation on Thousands of Processors*, Proceedings of SC 2002, September 2002.

[24] L. Kale, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Philips, A. Shinozaki, K. Varadarajan, K. Schulten, *NAMD2: Greater Scalability for*

Parallel Molecular Dynamics, Journal of Computational Physics, Volume 151, p.283-312, 1999.

[25] D. Fincham, *Optimization of the Ewald Sum for Large Systems,* Molecular Simulation, Volume 13, p.1-9, 1994.

[26] A. Toukmaji, D. Paul, J. A. Board Jr, *Distributed Particle-Mesh Ewald: A Parallel Ewald Summation Method*, Technical Report 96-002, Department of Electrical and Computer Engineering, Duke University, August 1996.
http://www.ee.duke.edu/~ayt/parallel_ewaldpaper/parallel_ewaldpaper.html
[Online accessed June 2005]

[27] *Xilinx Multimedia Board*
http://www.xilinx.com/products/boards/multimedia/
[Online accessed June 2005]

[28] *Xilinx Virtex-II Complete Data Sheet*
http://direct.xilinx.com/bvdocs/publications/ds031.pdf
[Online accessed June 2005]

[29] P. Henrici, *Essentials of Numerical Analysis*, Wiley, 1982.

[30] M. D. Ercegovac, T. Lang, *Digital Arithmetic*, Morgan Kaufmann Publishers, 2004.

[31] A. Ralston, *A First Course in Numerical Analysis*, McGraw-Hill, 1978.