4-22-2008

Efficient Molecular Dynamics Simulation on Reconfigurable Models with MultiGrid Method

Eunjung Cho

EFFICIENT MOLECULAR DYNAMICS SIMULATION ON RECONFIGURABLE MODELS WITH MULTIGRID METHOD

by

EUNJUNG CHO

Under the Direction of Anu G. Bourgeois

ABSTRACT

In the field of biology, MD simulations are continuously used to investigate biological studies. A Molecular Dynamics (MD) system is defined by the position and momentum of particles and their interactions. The dynamics of a system can be evaluated by an *N*-body problem and the simulation is continued until the energy reaches equilibrium. Thus, solving the dynamics numerically and evaluating the interaction is computationally expensive even for a small number of particles in the system. We are focusing on long-ranged interactions, since the calculation time is $O(N^2)$ for an *N* particle system.

In this dissertation, we are proposing two research directions for the MD simulation. First, we design a new variation of Multigrid (MG) algorithm called Multi-level charge assignment (MCA) that requires $O(N)$ time for accurate and efficient calculation of the electrostatic forces. We apply MCA and back interpolation based on the structure of molecules to enhance

the accuracy of the simulation. Our second research utilizes reconfigurable models to achieve fast calculation time. We have been working on exploiting two reconfigurable models. We design FPGA-based MD simulator implementing MCA method for Xilinx Virtex-IV. It performs about 10 to 100 times faster than software implementation depending on the simulation accuracy desired. We also design fast and scalable Reconfigurable mesh (R-Mesh) algorithms for MD simulations. This work demonstrates that the large scale biological studies can be simulated in close to real time. The R-Mesh algorithms we design highlight the feasibility of these models to evaluate potentials with faster calculation times. Specifically, we develop R-Mesh algorithms for both Direct method and Multigrid method. The Direct method evaluates exact potentials and forces, but requires $O(N^2)$ calculation time for evaluating electrostatic forces on a general purpose processor. The MG method adopts an interpolation technique to reduce calculation time to $O(N)$ for a given accuracy. However, our R-Mesh algorithms require only $O(N)$ or $O(\log N)$ time complexity for the Direct method on N linear R-Mesh and $N \times N$ R-Mesh, respectively and $O(r)+O(\log M)$ time complexity for the Multigrid method on *an* $X \times Y \times Z$ *R-Mesh*. r is N/M and $M = X \times Y \times Z$ is the number of finest grid points.

INDEX WORDS : Molecular Dynamics Simulation, Multigrid, Reconfigurable Model, Reconfigurable Mesh Algorithm, FPGA

EFFICIENT MOLECULAR DYNAMICS SIMULATION ON RECONFIGURABLE MODELS WITH MULTIGRID METHOD

by

EUNJUNG CHO

A Dissertation Submitted in Partial Fulfillment

Of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2008

EFFICIENT MOLECULAR DYNAMICS SIMULATION ON RECONFIGURABLE MODELS WITH MULTIGRID METHOD

by

EUNJUNG CHO

Committee Chair: Anu G. Bourgeois

Committee : Yi Pan
Sushil Prasad
Guantao Chen

Electronic Version Approved :

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2008

ACKNOWLEDGEMENTS

I wish to record my sincere thanks to my advisor Dr. Anu Bourgeois for her support and invaluable guidance throughout my thesis work. This dissertation would not have been possible without her help.

I would also like to express my gratitude to Dr. Sushil K. Prasad, Dr. Yi Pan and Dr. GuanTao Chen for their valuable advice and for taking time to review on the thesis document.

I also want to thank my friends Haejin, Gulsah and Feng for sharing various discussions and advice.

I wish to use this opportunity to express my thanks to my parents and my brother for their encouragement and support. And special thanks with all my heart to my husband and my daughter.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ASIC	Application-specific Integrated Circuit
BPTI	Basic Pancreatic Trypsin Inhibitor
CBI	Convolution Charge Assignment
CCA	Convolution Charge Assignment
FFP	Fast-Fourier Poisson
FFT	Fast Fourier transforms
FPGA	Field Programmable Gate Array
GBI	Gaussian Back Interpolation
GCA	Gaussian Charge Assignment
GRAPE	GRAvity PipE
LDM	Lattice Diffusion Multigrid
LGM	Lattice Gaussian Multigrid
LJ	Lennard Jones
MCA	Multilevel Charge Assignment
MD	Molecular Dynamics
MG	Multigrid
MODEL	MOlecular Dynamics processing Element
PR-Mesh	Pipelined Reconfigurable Mesh
R-Mesh	Reconfigurable Mesh
RTR	Real-Time Reconfiguration

Chapter 1
Introduction

Extensive research has been focused on the field of Molecular Dynamics (MD) over the past 20 years [2-5]. In the field of biology, MD simulations is continuously used to investigate biological studies including protein folding, enzyme catalysis, conformational changes associated with biomolecular function and molecular recognition of proteins, DNA, and biological membrane complexes. MD describes a classical particle molecular system as a function of time and has been successfully applied to understand and explain macro phenomena from micro structures, since it is in many respects similar to real experiments. An MD system is defined by the position and momentum of particles and their interactions (potential). The dynamics of a system can be evaluated by solving Newton's equation of motion, which is an *N*-body problem [6]. The classical *N*-body problem requires a numerical solution because general analytical solutions are not enough to prove it.

Solving the dynamics numerically and evaluating the interactions is computationally expensive even for a small number of particles in the system. In each iteration, MD simulation evaluates potential energy and acceleration, then updates the position of particles as changed by the forces. The simulation is continued until the energy reaches equilibrium. The interactions of particles to be evaluated are short-ranged interactions and long-ranged interactions. It takes O(*N*) time to calculate short-ranged interactions (the bonded potentials)

in a general purpose computer and $O(N^2)$ for long-ranged interactions (non-bonded potentials) in a general purpose computer. N is the number of particles in a molecular system. So we are focusing on long-ranged interactions due to the intensive computational time.

Many applications use MD for biomolecular simulations and the simulations are performed in multiscale of time and length. The simulations of the relevant scales require strong and fast computing power, but it is even beyond the reach of the current fastest supercomputers [2, 7]. Many approaches have been proposed to improve the performance of MD simulation in terms of the time required. These approaches are divided into two categories by focusing on either modifying the software or the hardware. The software approach involves developing efficient algorithms to calculate the forces. Currently many algorithms have been introduced and large scale parallel computers are used to achieve reasonable computational time. Among the algorithms, Ewald's method [8] runs in $O(N^{3/2})$ time and Particle Mesh Ewald (PME) method [3, 9] applies discrete fast Fourier transforms (FFT) to compute long-range interactions (reciprocal force) and reduce $O(N^{3/2})$ to $O(N\log N)$. The multigrid (MG) [4, 5] method requires $O(N)$ time complexity for a given accuracy on general purpose processor. Sagui and Darden [5] describe two techniques (LGM and LDM) based on MG method for classical MD simulations of biomolecules.

In this work, we propose an efficient *Multi-level Charge Assignment method (MCA)* method [10] that reduces calculation time and achieves better accuracy in LGM and LDM.

We apply multi-level charge assignment and back interpolation based on the structure of molecules to enhance the accuracy of the simulation. Managing fast calculation time and accurate simulation results is a challenge in software approaches since the two properties are usually a trade off.

The hardware approach has focused on running MD simulation in special purpose processors or developed Application-Specific Integrated Circuits (ASIC) to achieve much faster calculation times. Since MD simulations are performed for large number of atoms in a molecular system, many studies utilize supercomputing systems or parallel systems to achieve better performance. Alam *et al.* [2, 7] study the performance of supercomputing systems for running various MD simulation packages such as AMBER, NAMD and LAMMPS. NAMD and LAMMPS have been reported to scale up to a few thousand nodes, while AMBER's PME method does not scale beyond 128 processors [2, 11] due to the communication overheads. They expect that petaFLOPS-scale computing power in the near future will meet the speed requirements for biological studies[7], but not at the current time.

Special purpose processors [12, 13] and application-specific Integrated Circuits (ASIC) for MD simulation [14] require high costs, complicated processes, and long development spans. RASTRUN [12] and GRAPE-2A [13] have pipelines of digital signal processors to perform MD simulations. MODEL [14] is an ASIC machine for evaluating Lennard Jones (LJ) potential and Coulombic potential. Although the special purpose processors are very powerful,

it is much more expensive than microprocessor-based software solutions. The development of customized circuits is a very complicated process with a very long development span. In addition, it is hard to modify the circuit if the solution is changed at a later date.

Another research direction to achieve better performance is to adopt *reconfigurable models* to run large scale problems. Reconfigurable models provide the ability to customize circuits to specific problem inputs at run time and the ability to reuse the same logic resources in different configurations from one phase of a computation to the next[1]. These features enable efficient solutions to many problems, including image and video processing, cryptography, object tracking, digital signal processing, and networking[1]. Navid Azizi *et al.* [15] show the feasibility of using Field Programmable gate arrays (FPGA) to implement large-scale application-specific computations by implementing MD simulation. They design an architectural model for Lennard Jones force and simulate the model in TM3 [16]. They also suggest several factors to improve performance of the implementation. Youngfeng Gu *et al.* [17] provide an FPGA implementation for Coulombic force as well as Lennard Jones force and use Direct method [3] to calculate Coulombic force. Previous work includes a FPGA-based MD simulator that achieved faster simulation time than the simulation on a general purpose processor [15, 17]. Besides reducing the time required, another advantage is that an FPGA board is much cheaper compared to ASIC and special purpose processor or supercomputing system.

We have been working on exploiting two reconfigurable models for Molecular Dynamics simulation. First, we proposed an efficient method, Multi-level charge assignment (MCA) [10] for evaluating electrostatic potential which is the most time consuming process in MD simulation. We then design an FPGA-based MD simulator implementing the MCA method on a Xilinx Virtex-IV [18]. It performs about 10 to 100 times faster than a software implementation with Intel Core Duo T2300/1.66 GHz processor depending on simulation accuracy desired[19].

Second, we are proposing a project [20] that exploits another reconfigurable model to run MD simulations in a flexible and efficient manner. The Reconfigurable Mesh (R-Mesh) is a simple model to describe and understand since it uses a mesh topology to interconnect processors. Many published results use the R-Mesh (or variations) as the model of computation[21]. In this dissertation, we present fast and scalable R-Mesh algorithms for MD simulations and thus bring a new concept to the field of biology. This work demonstrates that the large scale biological studies can be simulated in close to real time. The R-Mesh algorithms we design highlight the feasibility of these models to evaluate potentials with faster calculation times. Specifically, we develop R-Mesh algorithms for both the Direct method and Multigrid method. The Direct method evaluates exact potentials and forces by using the equation in Chapter 2, but requires $O(N^2)$ calculation time for evaluating electrostatic forces on a general purpose processor. The Multigrid (MG) method adopts an

interpolation technique to reduce the calculation time to O(N) for a given accuracy. However, our R-Mesh algorithms require O(N) or O(log$N$) time complexity for the Direct method on N processors reconfigurable linear R-Mesh and N×N 2-dimensinal R-Mesh, respectively and O(r)+O(log$M$) time complexity for the MG method on time on an $X×Y×Z$ *3*-dimensional R-Mesh. . *r is N/M* and $M= X×Y×Z$ is the number of finest grid points applied to Multigrid method at a given parameter.

The main contribution of this work is presenting an efficient approach to solve the intensively time consuming and large scale problem of molecular dynamics simulations. Although the R-Mesh is a theoretical model, our work supports the theory that reconfigurable models are a good direction for biological studies which require high computing power.

We organize this dissertation as follows. In Chapter 2, we provide background of MD simulations and reconfigurable models, specifically the Reconfigurable Mesh (R-Mesh) model and Field Programmable Gate Arrays (FPGA). Chapter 3 provides current research directions for MD simulations. In Chapter 4, we describe our proposed algorithm, the MCA method. Chapter 5 describes the design of FPGA-based simulator implementing MCA method and Chapter 6 presents implementations of the FPGA-based simulator for MD simulation. In Chapter 7, we describe our proposed R-Mesh algorithms for MD simulations and summarized the results. Chapter 8 presents possible future directions for our research. Finally, Chapter 9 provides concluding remarks.

Chapter 2
Background

This chapter briefly describes the basics of Molecular Dynamics (MD) simulation. We also describe two reconfigurable models, Reconfigurable mesh and FPGA, that we utilize in our research for MD simulation. In sections 2.3, we describe the design process flow for implementing a new system on an FPGA.

2.1. Basic of Molecular Dynamics Simulation

In Molecular Dynamics (MD) simulation, dynamics are calculated by Newtonian mechanics [3]. MD simulation integrates acceleration to obtain position and velocity changes of atoms in the system. This process is typically continued every 1 femtosecond until the system stabilizes.

$$F = m \times a$$

There are other approaches to describe the forces of an MD system. Newton's equation of motion describes nature conserving the energy, but other approaches modify the forces to

achieve equilibrium states satisfying certain specified requirements, such as constant temperature, constant pressure or rigid molecules.

$\vec{F}_i$ represents i[th] atom's force and can be described by the potential energy:

$$\vec{F}_i = -\nabla_i U(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N) + \vec{F}_i^{extended},$$

where U is the potential, N is the number of atoms in the system and $\vec{F}_i^{extended}$ is an extended force like velocity-based friction.

The *potential U* consists of several types of forces.

$$U = U^{bonded} + U^{non-bonded} + U^{external}$$
$$U^{bonded} = U^{bond} + U^{angle} + U^{dihedral} + U^{impropor}$$
$$U^{non-bonded} = U^{electrostatic} + U^{Lennard-Jones}$$

It takes O(N) time to calculate the bonded potentials and O(N^2) for non-bonded potentials. So many researchers focus on the non-bonded interactions due to the intensive computational time. Non-bonded interactions can be divided into electrostatic potential ($U^{electrostatic}$) and Lennard-Jones ($U^{Lennard\text{-}Jones}$) potential. $U^{electrostatic}$ represents Coulomb potential and $U^{Lennard\text{-}Jones}$ represents a van der Waals attraction and a hard-core repulsion.

These potentials are pair-wise interactions and the formulation is given by

$$U_{ij}^{\ electrostatic} = \frac{1}{4\pi\varepsilon_0}\frac{q_i q_j}{\left\|\vec{x}_{ij}\right\|^2} \qquad \text{---------- (1),}$$

where π *and* ε_0 are constants and q_i and q_j are the charges of atoms *i* and *j*. $\vec{x}_{ij}$ is distance between atom *i* and *j*.

$$U_{ij}^{\ Lennard-Jones} = \frac{A_{ij}}{\left\|\vec{x}_{ij}\right\|^{12}} - \frac{B_{ij}}{\left\|\vec{x}_{ij}\right\|^{6}} \qquad \text{---------- (2),}$$

where $A_{ij} \geq 0$ and $B_{ij} \geq 0$ are the Lennard-Jones (LJ) parameters for atoms *i* and *j*. They define the energy minimum and the cross-over point, where the LJ function is zero.

The $U^{Lennard\text{-}Jones}$ can be calculated in O(*N*) time, since the LJ function decays very fast. But $U^{\ electrostatic}$ takes O(N^2) time by Equation. (1). Many methods try to reduce the time complexity while still achieving reasonable accuracy. We also propose a method to reduce the O(N^2) time to O(*N*) time, however we are able to achieve improved accuracy as compared to existing techniques.

Our R-mesh algorithm (Refer to Section 4.3 for details) implementing the Direct method uses Equation 1 to evaluate the electrostatic potential by using the equation, the Direct method provides exact results for MD simulations.

2.2. Reconfigurable Models

Reconfigurable bus-based models use dynamically alterable connections between processors to create various, changing, bus configurations. This allows efficient communication, and further, allows faster computation than on conventional *non-reconfigurable* models. Consequently, reconfigurable models have drawn considerable interest and numerous algorithms have been proposed for them [1]. Researchers have proposed a number of reconfigurable models including the Reconfigurable Mesh (R-Mesh) [22, 23], Fusing Restricted Reconfigurable Mesh (FR-Mesh) [24], Reconfigurable Network (RN) [25], Polymorphic Processor Array (PPA), Processor Array with Reconfigurable Bus System (PARBS), Reconfigurable Multiple Bus Machine (RMBM), Reconfigurable Buses with Shift Switching (REBSIS), Linear Array with Reconfigurable Pipelined Bus System (LARPBS) [26], the Pipelined Reconfigurable Mesh (PR-Mesh) [27, 28], and Field Programmable Arrays (FPGAs). Nakano [21] presented a bibliography of published research on reconfigurable models. However, we are focusing on R-Mesh and FPGA, since they can be exploited by complicated applications and are the most widely studied reconfigurable models.

2.2.1. Reconfigurable Mesh Model

An R × C Reconfigurable Mesh (R-Mesh) is a two-dimensional array of processors connected in an R × C grid [22, 23]. Each processor in the R-Mesh has Direct “external

connections" to adjacent processors through a set of four input/output ports. A processor can internally partition its set of ports so that ports in the same block of a partition are fused. These partitions, along with external connections between processors, define a global bus structure that connects the ports of processors. All ports that are part of the same bus are said to be in the same component of the R-Mesh.

The important features of an R-Mesh are [29, 30]:

1. An R × C R-Mesh is a 2-dimensional mesh connected array of processing elements (PEs). Each PE in the R-Mesh is connected to a broadcast bus which is itself constructed as an R × C grid. The PEs are connected to the bus at the intersections of the grid. Each processor has up to four bus switches that are software controlled and that can be used to reconfigure the bus into sub-buses. The ID of each PE is a pair(i, j) where i is the row index and j is the column index. The ID of the upper left corner PE is (0, 0) and that of the lower right one is (R-1, C-1).
2. The up to four switches associated with a PE are labeled E (east), W (west) S (south) and N (north). Two PEs can simultaneously set (connect, close) or unset (disconnect, open) a particular switch as long as the setting do not conflict. The broadcast bus can be subdivided into sub-buses by opening (disconnecting) some of the switches.
3. Only one processor can put data onto a given sub-bus at any time.
4. In unit time, data put on a sub-bus can be read by every PE connected to it. If a PE is

to broadcast a value in register R to all the PEs on its sub-bus, then it uses the command broadcast(R).

5. To read the content of the broadcast bus into a register R the statement R:=content(bus) is used.
6. Row buses are formed if each processor disconnects (opens) its S switch and connects (closes) its E switch. Column and connecting the S switches.

Figure 1 shows a 3 × 5 R-Mesh depicting the fifteen possible port partitions of a processor. The value written on the bus is called the bus data. The R-Mesh is a synchronous model that may change its bus configurations at each step. It also assumes negligible delay on buses [22, 23]. In this work, we assume the concurrent read and exclusive write (CREW) model.

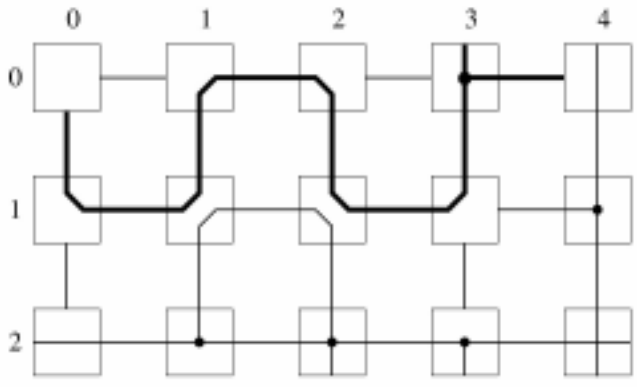


Figure 1. Internal connections of an R-Mesh

2.2.2. Field Programmable Gate Arrays (FPGAs)

A Field Programmable Gate Array (FPGA) is another reconfigurable model that has appeared in the literature [1]. Typically, these devices consist of an array of function blocks and a configurable interconnection fabric connecting them as shown in Figure 2. It is possible to configure function blocks to perform different logic functions (such as addition, multiplication, etc.) and configure the interconnection fabric to connect different function blocks.

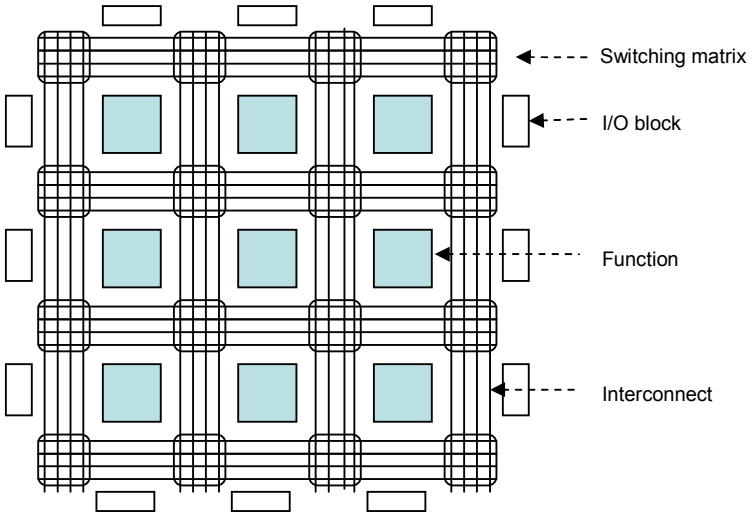


Figure 2. Simple structure of FPGAs [1]

Though other reconfigurable models and FPGA-based systems have evolved relatively independently, there is some common ground. For example, techniques for the bit-model R-Mesh can implement rudimentary arithmetic operations on FPGAs. Some Reconfigurable Multiple Bus Machine (RMBM) techniques could prove useful in configuring FPGA

interconnects. On the other hand, FPGA implementations can provide valuable clues to the direction of implementing devices with R-Mesh-type reconfiguration abilities [31].

Platform FPGAs provide a single platform that can address nearly any application. [32-34]. We can divide those applications into three categories. First, FPGAs are ideal platforms for Digital Signal Processing (DSP) applications. The 10 million gates Virtex-II architecture provides tremendous parallel processing capabilities. The result is up to 100 times higher performance than traditional programmable DSP solutions. Second, FPGAs offer a single platform for processing application. Embedded PowerPC processors, CoreConnect Buses, and related controllers and interfaces extend FPGA applications into embedded systems. Third, the flexibility of FPGAs to interconnect varying I/O standards has proven to be extremely valuable for systems integration. With the emergence of high bandwidth parallel and serial busses, IO interfaces are requiring even more complex cores and very high performance physical interface layers. Platform FPGAs provide very important connectivity and bridging between these emerging and traditional standards.

We aim to design an FPGA-based system that simulates MD with our Multi-level Charge Assignment (MCA) method. MCA method is an efficient and accurate algorithm we propose that follows the MultiGrid (MG) method. (Refer to Section 4.1 for details)

Chapter 3
Related work

In this chapter, we provide current approaches for Molecular Dynamics (MD) simulations. The approaches are divided into software approaches that have designed efficient algorithms and hardware approaches that have developed special purpose processors or reconfigurable models.

3.1. Software approaches

As mentioned in the introduction, there are many methods that try to reduce the calculation time of electrostatic potential effectively. One has designed new algorithms and exploited those to evaluate the forces. The Muligrid (MG) method is one of the efficient algorithms that evaluates the electrostatic force in O(*N*) time at certain accuracy. We will explain current researches that are based on the MG method

3.1.1. Multigrid method Background

The Multigrid (MG) method was introduced in the 1960's to solve partial differential equations (PDE). Recently it has been applied and implemented for *N*-body problems and

achieve O(*N*) time complexity at given accuracy. The basic idea of MG is to hierarchically separate the force potential into a *short range part* plus a *smooth part (slowly varying part of energy)*. MG method uses gridded interpolation for both the charges (source) and the potentials (destination) to represent its smooth (coarse) part [9]. The splitting and coarsening are applied recursively and define a grid hierarchy (Refer to Figure 3). Figure 4 presents a recursive Multigrid scheme in pseudo code.

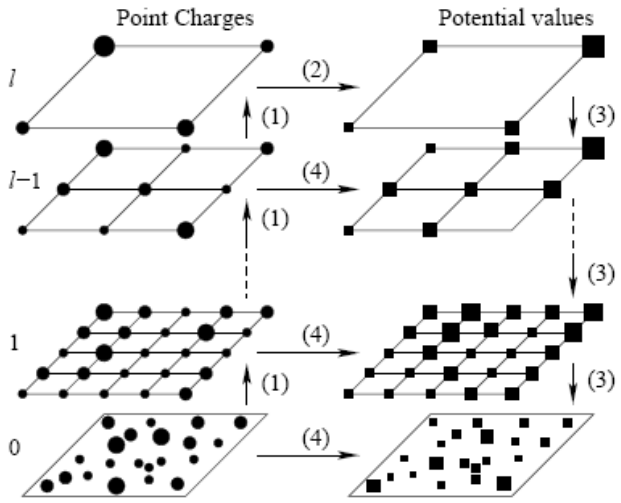


Figure 3. The multilevel scheme of Multigrid algorithm [9]
(1) Aggregate to coarser grids (2) Compute potential induced by the coarsest grid
(3) Interpolate potential values from coarser grids (4) Local corrections

The MG method is faster for a given error rate, but cannot provide better accuracy than other methods such as Ewald's method [8] and Multipole method [35]. In order to use MG method, we need to map an arbitrary distribution of discrete particles onto a grid space. There are several mapping schemes, *charge assignment* schemes, and those schemes play an

important part in MD simulation, since both accuracy and efficiency depend on the *charge assignment* scheme. That is, the charges must be spread over a fairly large number of grid points, which can become the most time-consuming part of the algorithm. Beckers *et al.* [36] presented a new implementation of P3M method by solving Poisson's equation in real space with a Successive OverRelaxation (SOR) method, but the accuracy is not good enough. So Sagui and Darden [5] proposed a modified Lattice Diffusion Multigrid (LDM) method that improved accuracy and speeds up the algorithm.

```
main:
    1. anterpolate position charges to the finest charge grid(1) - step(1) in Fig 3
    2. call multiscale(maxLevel, level 1)
    3. interpolate finest potential grid(1) to the position potentials - step(3) in Fig 3
    4. correct position potentials – step(4) in Fig 3
    5. compute forces and total energy
multiscale(maxLevel, level k):
    1. if maxLevel = k then
        (a) compute potential values on coarsest grid(maxLevel) -step(2) in Fig 3
    2. otherwise
        (a) anterpolate charge grid(k) to coarser charge grid(K+1) - step(1) in Fig 3
        (b) call multiscale(maxLevel, K+1)
        (c) interpolate coarser potential grid(K+1) to potential grid(K) - step(3) in Fig 3
        (d) correct potential grid(k) - step(4) in Fig 3
```

Figure 4. Pseudo-code of a recursive Multigrid scheme [9]

In addition, they proposed another method, Lattice Gaussian Multigrid (LGM) [5] method, which is more accurate, but more time consuming.

3.1.2. Fast-Fourier Poisson method

York and Yang [37] presented the Fast-Fourier Poisson (FFP) method, which provides an accurate solution of the Poisson's equation. They present the new form (second term in Equation (3)) to evaluate reciprocal force of electrostatic force that only needs evaluation at the lattice points. Their method is faster than Ewald summation[8] which requires to evaluate reciprocal potential($\phi_{rec}(r)$) at the point charge positions.

$$E = E_{dir} + E_{rec} + E_{self}$$
$$= \frac{1}{2}\sum_{n,i,j} q_i q_j \frac{erfc(\beta r_{ijn}/\sqrt{2})}{r_{ijn}} + \frac{1}{2}\int \rho_s(r)\phi_{rec}(r)d^3r - \frac{\beta}{\sqrt{\pi}}\sum_{i=1}^{N} q_i^2 \quad \text{-------- (3)}$$

However, they still use Fourier transform in second term in Equation (3), which has proven difficult to implement on the current parallel machine. Sagui and Darden also use Equation (3), but provide a method that calculates the second term in the real lattice. We also present our method in real space and the method could be implemented on a parallel machine.

3.1.3. Lattice Diffusion Multigrid Method

Sagui and Darden [5] introduced B-splines, which provides smooth polynomial interpolation, together with recursion formulas for their analytic derivatives. This allows computation of the

forces through analytic differentiation in real space. They applied this technique to the Lattice Diffusion Multigrid (LDM) Method. LDM has four steps

(1) Charge interpolation : Charges are interpolated onto the grid via B-splines

$$q(x) = \sum B_p(x - r_i) \quad \text{---------- (4)}$$

,where p (order of interpolation) = integer($1/\beta \cdot h_x$), x is a point on the lattice, and r_j indicates the coordinates of particle j. β is Gaussian coefficient and h_x is the size of fine grid

(2) *Charge smoothing*:

$$q(x)^{n+1} = q(x)^n + D' \sum_{NN} q(x_{NN})^n \quad \text{--------- (5)}$$

where, x_{NN} are nearest-neighbor grid points (first- , second and Nt-th nearest-neighbor, Nt = integer($1/\beta^2 h_x^2$), n stands for the time iteration. D′ is diffusion constant and adjustable parameter. Its initial value is $D' = 1/2[(1/h_x^2) + (1/h_y^2) + (1/h_z^2)]^{-1}$. The initial value is decreased slowly until it reaches a value that minimizes the force error.

(3) *Solution on the grid*: The electrostatic potential is evaluated on the mesh using an O(*N*) Multigrid technique. A Hermitian representation of Poisson's equation or seven-point representation of Laplacian can be used [5, 38].

(4) *Back interpolation* : the particle energy and force are interpolated back using the same B-splines

$$E_{rec,i} = q_i \sum_x B_p(x - r_i)\phi(x) \quad \text{---------- (6)}$$

$$F_{rec,i} = -q_i \sum_x \frac{\partial}{\partial r_i} B_p(x - r_i)\phi(x) \quad \text{---------- (7)}$$

As Beckers *et al.* [36] suggest in original diffusion scheme, D′ is an adjustable parameter. Sagui [5] find the optimal D′ by setting initial value and diminish it slowly.

However, an adjustable parameter is not a systematic factor even if it works well in some molecules. Lattice Gaussian Multigrid (LGM) method is a more systematic approach to interpolate the particle charges and smoothing the charges to give better accuracy.

3.1.4. Lattice Gaussian Multigrid Method

Sagui and Darden [5] provide a modified fast-Fourier Poisson (FFP) method to avoid the use of Fourier transforms since FFT has proven difficult on the current parallel machines. This is due to the large communication overhead associated with the global shifting of data between different processors for FFT evaluations.

The method performs all the calculations in real space, and solves Poisson's equation via Multigrid methods, and by using exact analytical expressions in real space for the forces.

(1) *Gaussian Charge assignment (GCA) & smoothing*

The charge assignment and smoothing is performed directly by Gaussian

$$\rho_s(x) = \frac{\beta^3}{\pi^{3/2}} \sum_{i=1}^{N} q_j \exp(-\beta^2 |x - r_i|^2) \quad \text{---------- (8)}$$

x is a point on the lattice, and r_j indicates the coordinates of particle j which is cutoff $R_c < |x-r_j|$. This assignment works better for relatively high βs and a smaller R_c if needed for convergence.

(2) *Solution on the grid* : The electrostatic potential is evaluated on the mesh using an O(N) Multigrid method. In LGM, Hermitian representation provides better performance. [5, 38]

(3) *Gaussian Back interpolation (GBI) :* The energy can be directly computed on the grid as

$$E_{rec,i} = \frac{1}{2} \sum_x \rho_s(x) \phi(x) h_x h_y h_z \quad \text{---------- (9)}$$

and by taking derivative of the energy with respect to the coordinated of particle i, we can get the corresponding fore as

$$F_{rec,i} = -\frac{2\beta^5 q_i}{\pi^{3/2}} \sum_x (x - r_i) \times \exp(-\beta^2 |x - r_i|^2) \phi_{rec}(x) h_x h_y h_z \quad \text{------ (10)}$$

LGM is stable and accurate method but not efficient. While LDM is very efficient but not

stable and accurate.

Banerjee *et al.* [39] proposed an approach to improve the evaluation time and keep the accuracy of LGM.

3.1.5. Convolution charge assignment in LGM

Banerjee *et al.* [39]points out that the execution time of Gaussian Charge Assignment (GCA) and Gaussian Back Interpolation (GBI) is over 70% of total execution time of LGM on single processor. Even on parallel processors, the execution time consumes the majority of time. They use convolutions in GCA and GBI to reduce the number of grid points needed to approximate a Gaussian charge distribution.

In detail the proposed charge assignment (Convolution Charge Assignment, CCA) and back interpolation (Convolution Back Interpolation, CBI) approximate Gaussian charge distribution by Gaussian basis function and calculate weights for sampled Gaussian distributions. Then the weights are added to grid distribution ρ_m and convolutions are applied to the ρ_m with the Gaussian basis function. As more sample points are used in computing the weight, the error from center of the distribution is reduced. Thus, it is more accurate to use large number of sample points because it uses more weighted Gaussian distributions and the final grid points more match GCA. But this increases evaluation time. They compare performance with Sagui's LGM [5]. It shows same accuracy level and reduces 60% total

execution time. However, the charge distribution is not consistent since it is an approximation of particles in the lattice and the distribution of particles could reduce the accuracy of most methods.

We propose a method that improves the accuracy by applying *Multi-level Charge Assignment* (MCA) method for unbalanced distribution of particles in a molecule (Refer to Chapter 4).

3.2. Hardware approaches

The number of atoms in an MD system is typically large and the MD simulation must continue until the forces reach equilibrium. Therefore, the total time required is significantly large, even if efficient algorithms are applied to the simulation. Many hardware approaches provide rapid and accurate MD simulation. In this section, we provide three approaches related with our research. First approach is exploiting supercomputing systems to run software packages for MD simulations. Software packages implement various methods and tools for analysis of results. We provide the analysis of performance for the packages running on supercomputing systems. Second approach is focusing on running the simulations on special purpose processors or developed Application-Specific Integrated Circuits (ASIC). Third approach is utilizing a reconfigurable model, Field Programmable Gate Arrays

(FPGA)-based MD simulators that currently are proposed and developed.

3.2.1. Supercomputing Systems

In the field of biology, a number of commercial and free software packages are used to evaluate Molecular Dynamics. AMBER, CHARMM, GROMACS and NAMD are used by a large community of biologists. AMBER provides various algorithms to calculate the MD simulation and consists of about 50 programs that perform diverse functions for configuration processes and analysis of results. The main module of AMBER is *sander*, which stands for simulated annealing with NMR-derived energy restraints. *Sander* is also the "main" program used for MD simulations, and is also used for replica-exchange, thermodynamic integration, and potential of mean force (PMF) calculations. Alam *et al.* [2] used *sander* to investigate the performance characteristic of MD techniques using PME and Generalized Born (GB) method and analyze the performance of AMBER on two teraflops-scale systems, IBM Blue Gene/L and Gray XT3. Blue Gene/L is the current fastest supercomputing system. They run the experiments on a 1024-node Blue Gene/L system. A single Blue Gene/L processing node consists of an ASIC, which contains the code execution logic, on-chip memory and communication logic. The total processing power is 2.8 gigaFLOP/s per processor or 5.6 gigaFLOP/s per processing node. The total memory available to an to an application is 512 megaBytes and the off-chip memory bandwidth is 5.5 gigaBytes/s. it provide two network

topologies (tree-dimensional torus network and tree network) for message passing operations The bandwidth of tree network is 2.8 gigaBytes/s and the bi-directional bandwidth is 1.4 gigabits/s in six torus directions. The Cray XT3 system builds on a single processor node or processing element (PE). Alam *et al*. used early system which contains over 5000 processing nodes XT3 system at Oak Ridge National Laboratory (ORNL) for the experiments. The XT3 uses a commodity microprocessor, the AMD Opteron and connects these processors with customized interconnect based on an ASIC called SeaStar. The ORNL XT3 uses Opteron model 150 processors and Opteron core has a 2.4 Ghz clock and the peak floating point rate of this processor is 4.8 gigaFLOP/s. Each PE has 2 gigaBytes of memory the the peak memory bandwidth per processor is 6.4 gigaBytes/s. The PE is connected toe the SeaStar chip with a 6.4 gigaBytes/s HT path. The router in SeaStar provides six high-speed network links to connect to six neighbors in the 3D torus/mesh topology. Each of the six links has a peak band width of 7.6 gigaBytes/s. The Cray XT3 bypasses communication bottlenecks, such as the PCI bus. Alam *et al.* report that AMBER's PME method does not even scale up to 128 processors on Blue Gene/L and AMBER's Generalized Born (GB) method scale up to 1024 processors on a Cray XT3.

IBM developed a new framework for MD on the Blue Gene/L called Blue Matter [40]. Its preliminary work allows scaling a small system up to 512-1024 processors. NAMD is a software framework and is reported to scale to thousands of processors. It used a

communication layer called CHARM++, which has been ported to the Blue Gene/L and XT3 systems. LAMMP [7] is also a software framework from Oak Ridge National Laboratory and provides scaling up to thousands of processors. However current version of LAMMPS does not provide the energy minimization technique and lacks many functionalities for simulations and analysis, which are provided with AMBE and CHARMM.

Currently most supercomputing systems cannot reach the computing power that is required for biological studies that include molecular systems over of 10,000 atoms. It is expected, however, but they expect that petaFLOPS-scale computing power in the near future will meet the speed for biological studies [7].

3.2.2. Special purpose machines and Application-Specific Integrated Circuits (ASIC)

Shinjiro Toyoda *et al.* [14] developed a custom processor called MODEL (MOlecular Dynamics processing ELement) for calculating *Lennard Jones force* and *Coulombic force* and a scalable plug-in machine to a workstation. The processors work in parallel and have pipeline architecture. Their MD engine system consists of 76 MODELs and is approximately 50 times faster than the equivalent software implementation on a 200 MHz Sun Ultra 2. They consider non-bonded forces (Lennard Jones force and Coulombic force) to be calculated. The magnitude of forces was determined through table lookup since computation of these forces

required greater than 99% of the CPU time in software-based simulations. The MD engine system using MODEL chips apply Ewald method [8] to calculate Coulombic forces and the method allows precise calculation of Coulombic forces. Although MODEL achieves highly improved simulation results, developing an ASIC such as MODEL not only takes much time, but also is very expensive. Most of all, it is hard to modify the circuits when the solution needs to be changed.

GRAPE(GRAvity PipE) [12, 13] is one of the ASIC machines which is originally developed for gravitational problems and currently many GRAPEs have been developed for *N*-body problem such as MD simulation. Yuto Komeiji *et al.* [12] developed MD-GRAPE (Figure 5). MD-GRAPE is one of the GRAPEs and computes force and potential efficiently. The architecture of GRAPE is simple and easy to develop. They did not use floating point arithmetic throughout as was done in MODEL. Instead, position is stored as a 40-bit fixed-point number and the force is accumulated onto an 80-bit fixed-point number. The switch to fixed-point reduces the hardware requirements substantially. However, floating-point was still used for the calculation of the forces. MD-GRAPE divides the force calculation into two part. Only the $O(N^2)$ operations were off-loaded to the hardware in this system and $O(N)$ operations were performed on a host computer. Thus, the communication time with host limits the performance of the system if the number of particles is not large enough.

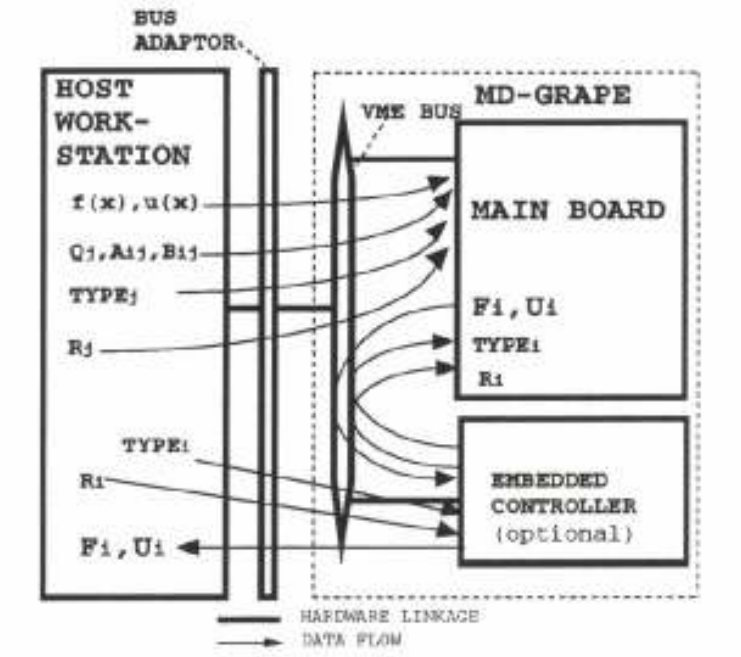


Figure 5. The hardware architecture of the MD-GRAPE system and the data flow between the machines [12]

3.2.3. FPGA-based Application Specific Processor (ASP)

While many researchers have designed ASIC machines for MD simulations [12-14] , it is difficult for them to be altered or upgraded due to a lack of flexibility in their dedicated hardware circuits. Recently, reconfigurable computing has emerged as an alternative to ASIC. It allows hardware circuits to be configured to perform the target task. Field Programmable Gate Arrays (FPGA) are semiconductor devices that processes digital information and can be reprogrammed without slowing performance. FPGA boards are cheap compared to ASICs and are very flexible due to its reprogrammable feature.

Navid Azizi *et al.* [15] exploit FPGA technology for simulating Molecular Dynamics. They

show that FPGAs is a feasible technology to develop large-scale application specific computation, such as MD simulation and developed FPGA-based ASP (Application Specific processor) to calculate Lennard Jones force of MD simulation. Their platform is TM3[16], which contains multiple interconnected FPGAs and consists of four Virtex-E 2000 devices connected to each other via 98-bit bidirectional buses. In addition, each FPGA is connected to a dedicated 256k×64 bit external SRAM, I/O connector and a nibble bus that allows communication with the host computer for download and control functions. They propose effective organization (two arrays for position) of the memory and FPGA speed (100 MHz) to improve the performance 20 times better than software implementation (2.4 GHz P4). In longer simulations, the error in potential energy remained below 1% while kinetic energy differences between hardware and software were less than 5%.

Navid Azizi *et al.* [15] introduce FPGA-based ASP for MD simulations, but they only calculate Lennard Jones force. Youngfeng Gu *et al.* [17] explore FPGA implementations for MD simulations and complete the calculation for non-bonded forces (Coulombic force and Lennard Jones force). They apply the direct method [3] to calculate the forces and used an Annapolis Microsystem Wildstar board Virtex-II XC2VP70-5 FPGA and simulate VP100, Xilinx Virtex-II Pro XC2VP100 -6 FPGA. They compare their FPGA implementation with 2.0 GHz Xeon CPU and shows it can be accelerated from 31 times to 88 times with respect to a software implementation, depending on the size of the FPGA and the simulation accuracy.

Figure 6 shows the block diagram of the FPGA parts of Youngfeng Gu [17]'s system.

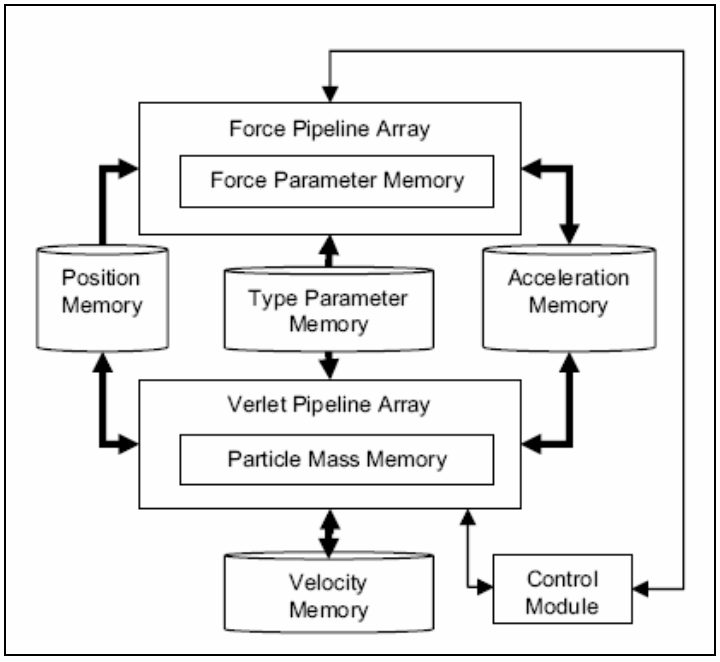


Figure 6. Block diagram of the FPGA parts of the system [17]

Chapter 4
Multi-level Charge Assignment (MCA) method

Many current techniques for the Molecular Dynamics (MD) simulation are based on the Multigrid (MG) method since this method is efficient and can be implemented on a parallel machine. However, most applications using the MG method provide inconsistent accuracy by the structure of molecules to be simulated and adjust the parameters in an unsystematic way. Since particle charges are interpolated onto the grid, the grid charge or charge distribution on a grid point cannot well represent the particle charges when the particles of the molecule are not spread evenly. In this Chapter, we propose a method to provide consistent accuracy and reduce errors even if the distribution of particles is not balanced. In Section 4.1, we describe the basic concept of our proposed Multi-level Charge Assignment (MCA) method. Section 4.2 provides the detail algorithms of the MCA. Section 4.3 provides the results and analysis of the MCA method.

4.1. Basic concept of Multi-level Charge Assignement (MCA) method

In the MG method, charge assignment and back interpolation are very important steps since accuracy and efficiency depend on these steps. We propose Multi-level Charge Assignment (MCA) [10] method which provides better accuracy with little additional cost. The main idea

of the MCA method is two fold; 1) the size of the finest grid is a factor to improve the accuracy, but the efficiency is decreased; 2) many molecules have different structures so grid points represent different number of particles.

The MCA method uses different sizes of finest grid when the finest grid represents more particles than a particular threshold value and interpolates the particle charges to the grids. The method consists of four steps: (1) calculate the density of particles for each grid point; (2) apply the MCA if the density of grid x is greater than threshold, k. Then interpolate and smooth the charges onto the grid; (3) calculate the electrostatic potential on the grid via the original MG methods; (4) back interpolate the forces and energy from the grid to the particle space.

Figure 7 shows an example that applies the MCA on grid G4. Figure 7 (a) shows the charge assignment for the original MG method. The charges of particles (solid circle; A - G) are interpolated to G4 since the particles are located within R_c (cutoff distance). Figure 7 (b) shows an example when the MCA is applied to interpolate those particles (solid circle). We would consider that the charges for particles E, F and G are interpolated to g1 and g2 which are smaller grids and have smaller cutoff distance ($R_c/2$) since particle E is closer to g1 than G4 and particle F and G is closer to g2 than G4, therefore MCA is applied. Then, g1 and g2 are interpolated to a coarser grid level G4.

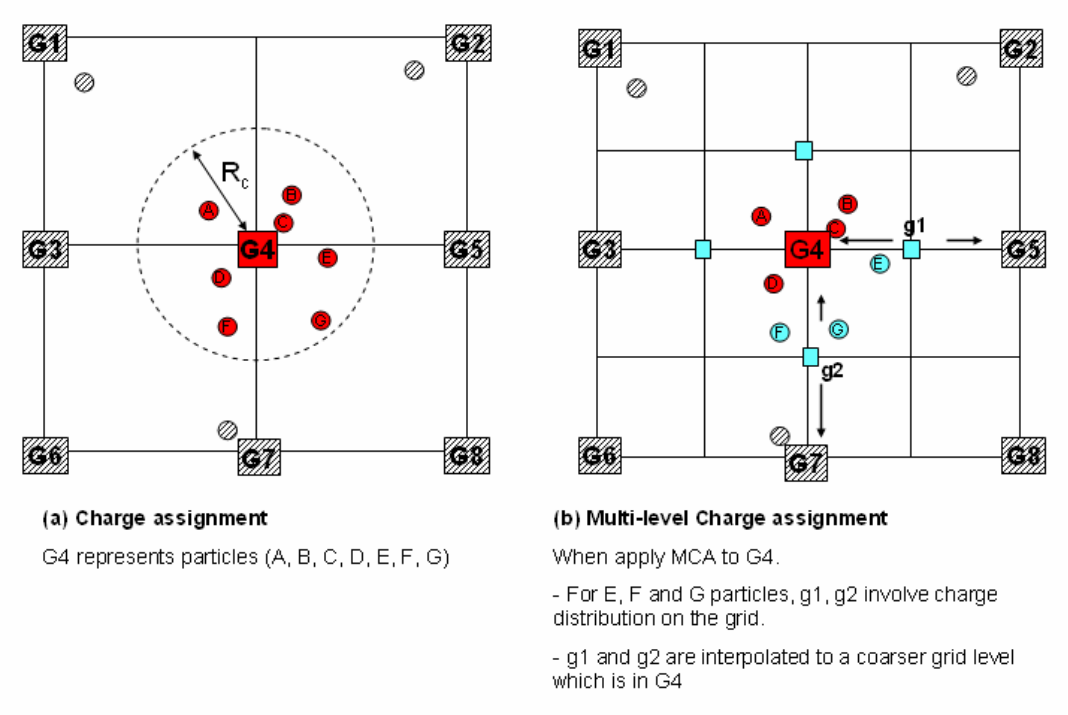


Figure 7. Multi-level Charge Assignment example

4.2. Multi-level Charge Assignment and Back Interpolation in LDM

The steps of MCA method in LDM are as follows:

(1) *Preprocessing*

DOP(x) represents density of particle for each grid point and is calculated by following equation.

$$DOP(x) = \sum_{i}^{n} (Anterpolate(x - r_i))$$

Anterpolate(y) return 1 if $|y| < R_c$ otherwise return 0, x is a finest grid in regular assignment method, R_c is cutoff distance and r_i indicates the coordinates of particle i.

(2) *Assignment*

Charge assignment is divided into two cases. Case I is where x represents the particles less than a threshold, k. Case II is where x represent the particles greater than k and needs to apply MCA to the grid, x.

Case I

If DOP(x) <= k, perform regular charge assignment

$$q(x) = \sum A_p(x - r_i)$$

A_p is interpolation function (Hermite or B-spine) and p (order of interpolation) = integer($1/\beta \cdot h_x$), β is Gaussian coefficient and h_x is size of fine grid. k is highest or second highest DOP value. If k is smaller, the method provides more accurate results but increases calculation time. x is a point on the lattice, and r_i indicates the coordinates of particle i which is $|x\text{-}r_i| > R_c$.

Case II

If DOP(x) > k, use smaller grids level (h_x /2, h_y /2, h_z /2) and interpolate the smaller grid to fine grid level(h_x, h_y, h_x).

Step 1 calculate q(y), which y is smaller grid points near x.

$$q(y) = \sum A_p(y - r_i)$$

, where r_i indicates the coordinates of particle i and particle i was assigned at x in preprocessing. In this function, cutoff is R_c/2.

Step 2 interpolate charges of y to charges of x by A_p and update ($\theta_{h_x}, \theta_{h_y}, \theta_{h_z}$) of the fine grid($h_x$, h_y, h_z) for back interpolation to obtain forces.

$$(\theta_x, \theta_y, \theta_z) = (w_s\theta_{h_x} + w_f\theta_{h_x/2}, w_s\theta_{h_y} + w_f\theta_{h_y/2}, w_s\theta_{h_z} + w_f\theta_{h_z/2})/2$$

,where w_s and w_f are currently set by value 1.

(3) *Smoothing*

This process is similar to smoothing process in LDM. (Refer to Section 3.1.3)

(4) *Solution on the grid*

The electrostatic potential is evaluated on the mesh using an O(N) Multigrid technique. A Hermitian representation of Poisson's equation or seven-point representation of Laplacian can be used [5, 38].

(4) *Back interpolation*

The particle energy and force are interpolated back using the same interpolation function A_p. However, Multi-level back interpolation should be applied. If the grid had MCA applied to it, then ($\theta_{h_x}, \theta_{h_y}, \theta_{h_z}$) were already updated and these values are used to obtain particle energy and force by back interpolation.

4.3. Results and analysis

The proposed method has been implemented in C++ and Intel Core Duo processor T7400 and 1 GB memory. We compare the accuracy and running time with a Multigrid implementation

of Protomol 2.0 [41]. Protomol 2.0 provides Simple, PME, Full Ewald and Multigrid (MG) implementation for the MD simulation and a platform for comparing various methods. Protomol implements a Lattice diffusion multigrid method and uses B-Spine and Hermite interpolation. Our tests were performed for a Calcium molecule with 309 atoms and Basic Pancreatic trypsin inhibitor (BPTI) with water that has 1101 atoms. These test data are standard and configuration files can be obtained from Protomol website or Protein Data Bank. All experiments provide comparisons for the three methods (simple, LDM and MCA). Simple method uses Equation (1) and provides exact simulation results. LDM and MCA method use 4th order Hermite interpolation and three grid level to perform MD simulations on Calcium and BPTI molecules.

Figure 8 shows the potential energy for Calcium molecule and simulates this for 10000 iterations and Figure 9 shows potential energy for BPTI molecule. Figure 8 and Figure 9 show that the simulation results of three methods provide similar plot. That is, the MCA method provides correct simulation results.

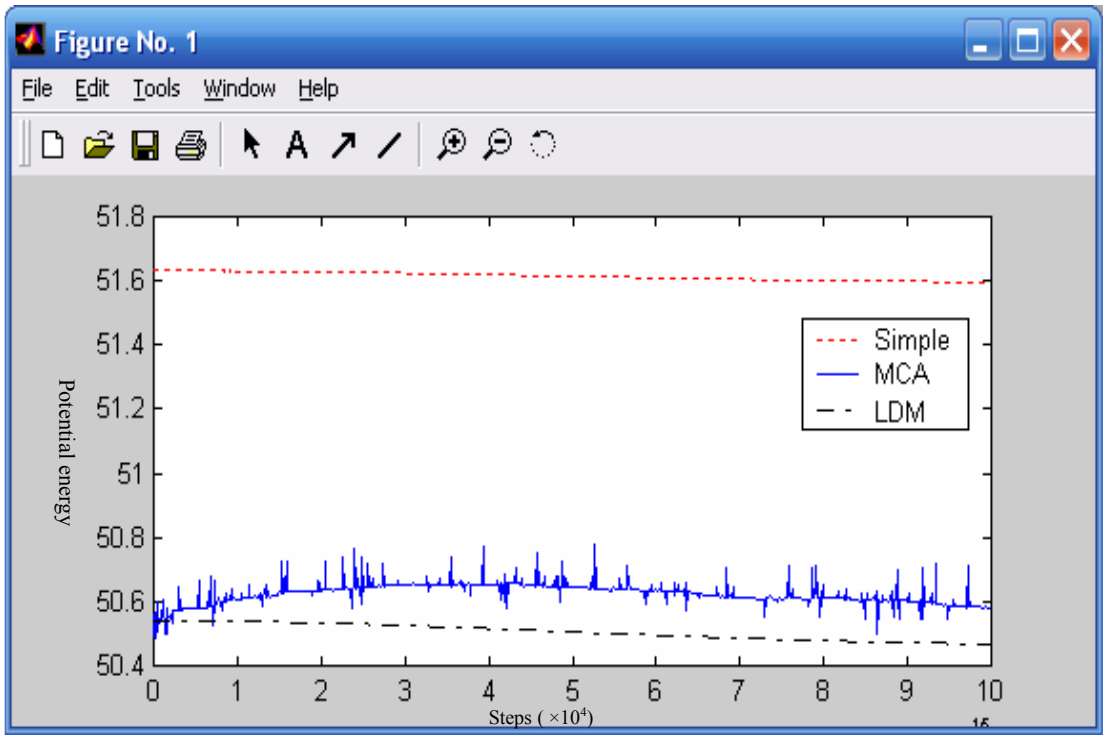


Figure 8. Potential energy for each method, Simple, LDM and MCA method for Calcium molecule

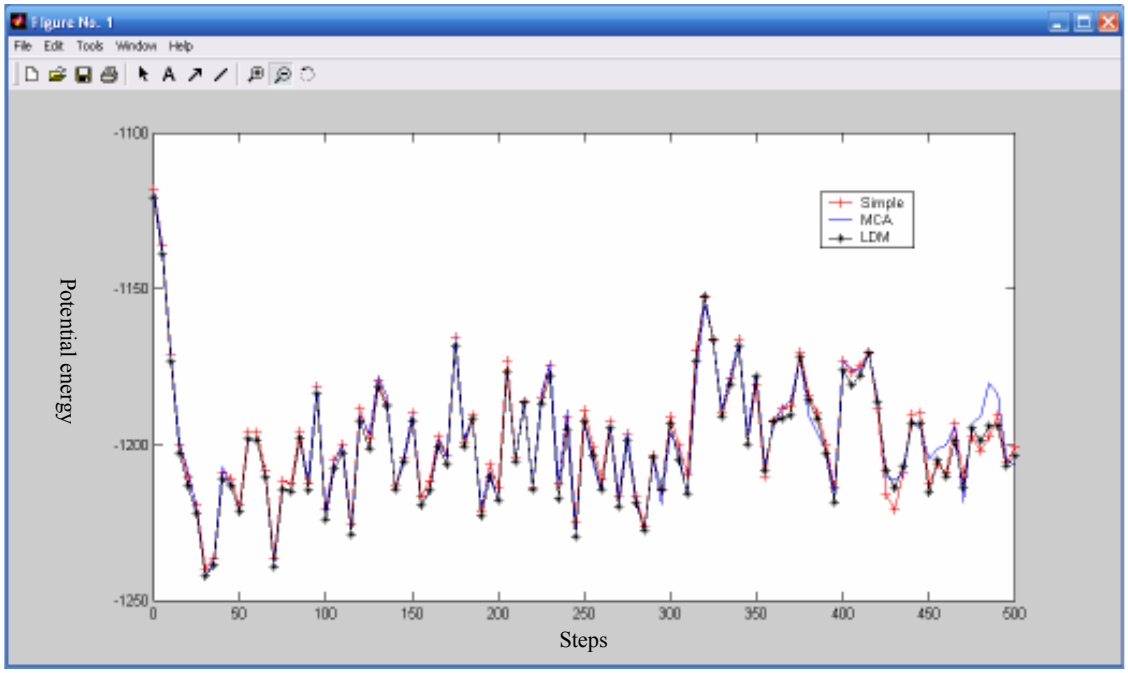


Figure 9. Potential energy for each method, Simple, LDM and MCA method for BPTI with water

To test the accuracy of LDM and MCA against the Simple method, we compared the energy relative errors (*rel.error*) defined as

$$rel.error = \frac{\left|E_{Simple} - E_m\right|}{E_{Simple}}$$

, where the subscript "Simple" refers to the exact electrostatic potential energy computed using Simple method. The subscript "m" refers to the potential energy computed using LDM or MCA method.

The simulation with Calcium molecule shows that the relative error is 0.0217 in LDM method and 0.0204 in MCA method. It is a 6% improvement on accuracy. But the total execution time is 62.78sec in LDM method and 63.90sec in MCA, resulting in minimal additional calculation time. The simulation with BPTI molecule shows that the relative error is 0.0022 in LDM method and 0.00177 in MCA method. It is a 19% improvement on accuracy. The total execution time is 118.14sec in LDM method and 119.17sec in MCA method, resulting in 1sec (0.8 %) additional calculation time.

Table I shows the simulation results and compares the proposed method with Simple and LDM method. As can be seen, both experiments show improved accuracy for the proposed method and particularly the experiments on BPTI provides much better improvement (19.5%). The reason that BPTI displays better results than the Calcium molecule is that more atoms in

BPTI molecule are applied to MCA method. As we mentioned, if the molecule is not spread evenly in grid, the MCA method provides efficient charge assignment and back interpolation.

TABLE I. Total execution time and accuracy for Simple method, LDM and MCA method on Calcium molecule and BPTI molecule

	Calcium Molecule				BPTI Molecule			
	Simple	LDM	MCA	Comparison MCA vs. LDM	Simple	LDM	MCA	Comparison MCA vs. LDM
Time (sec)	69.62	62.78	63.90	1.78% slower	159.28	118.14	119.16	0.8% slower
Accuracy	1	0.0217	0.0204	6% better accuracy	1	0.0022	0.00177	19.5% better accuracy

In summary, the MCA method achieves much better accuracy with just a little bit of cost in terms of time. The experiment of BPTI molecule in TABLE I show that the cost ratio of time to accuracy is 1:24.4.

In this chapter, we prove the MCA method improves the accuracy for the MD simulation, but it still requires the calculation time similar to the original MG method. Since it is very crucial to reduce the calculation time for the MD simulation, we study another research direction that exploit reconfigurable models, FPGA and R-Mesh, to achieve fast calculation time as well as improved accuracy. Next chapters present the research directions. Chapter 5 and 6 present our proposed FPGA-based MD simulator and Chapter 7 present R-Mesh algorithms for two method of the MD simulation.

Chapter 5
Design of FPGA-based simulator for MD simulation

It is difficult for Application-Specific Integrated Circuits (ASIC) machines to be altered or upgraded due to a lack of flexibility in their dedicated hardware circuits. Recently, reconfigurable computing has emerged as an alternative to ASIC machines. FPGAs are reconfigurable models that are semiconductor devices that processes digital information and can be reprogrammed without slowing performance [1]. Since FPGA boards are cheap and flexible compared to ASICs, many researchers have explored FPGA implementation for the MD simulation. Navid Aziz *et al.* [15] and Younfeng Gu *et al.* [17] designed FPGA implementations for calculating Lennard Jones force and non-bonded forces (Coulombic force and Lennard Jones force) of the MD simulation. They use the Direct method for Coulombic force and the calculation time of Direct method is $O(N^2)$ for an N particle system. However our proposed MCA method requires O(N) calculation time with improved accuracy and we designed an FPGA simulation implementing the MCA method. The MCA [10] method is categorized as a Multigrid(MG) method. Since the MG method is efficient and can be parallelized, many approaches for the MD simulation are based on the MG method. As we mentioned, most applications using the MG method provide inconsistent accuracy by distribution of particles in molecules to be simulated and adjust the parameters

unsystematically to get better accuracy. The MCA method applies multi-level interpolation that adjusts the grid charge and charge distribution and achieves a simulation that is more accurate without much increase in calculation time.

In this Chapter, we design an FPGA-based MD simulator that achieves fast calculation times and improved accuracy. In Section 5.1, we describe the architecture of the FPGA-based MD simulator that implements the MCA method.

5.1. System Architecture

We break down the steps to demonstrate the design. Figure 10 describes 7 steps to design an FPGA-based MD simulator. Step 0 in Figure 10 is preprocessing density of fine grid in order to apply the MCA method to the grids which represent too many particles. Step 1 in Figure 10 is MCA_Anterpolate process that maps particle charges to the finest grid level. Step 3-Step 6 in Figure 10 are similar with the procedure of the original MG method.

```
Step 0 : Preprocessing density of fine grid
Step 1 : MCA_ANTERPOLATE particle charges → C_Grid(1)
Step 2: WHILE K = 1 ..Max-1
            ANTERPOLATE C_Grid(K) → C_Grid (K+1)
Step 3 : COMPUTE energy values on C_Grid(Max)
Step 4 : WHILE K = Max .. 2
            INTERPOLATE E_Grid(K+1) → E_Grid(K)
            CORRECT E_Grid(K)
Step 5 : INTERPOLATE E_Grid(1) → particles energies
         CORRECT particles energies
Step 6 : COMPUTE forces and total energy
```

---

* K level Charge Grid : C_Grid(K)
* K level Energy Grid : E_Grid(K)
* Finest charge grid : C_Grid(1)
* Finest Energy grid : E_Grid(1)
* Coarsest charge grid : C_Grid(Max)
* Coarsest charge grid : E_Grid(Max)

Figure 10. Pseudo code of MCA method for FPGA-based MD simulator

We consider the precision of proposed architecture since the precision of system affects arithmetic accuracy of the MD simulation which is calculated by designed circuits. Amisaki *et al.* [42] examined many factors that affect the arithmetic accuracy of a dedicated accelerator that calculate non-bonded interactions. They found that the pair-wise force should be calculated with at least 29 bits of precision using coordinates that, in turn, should have at least 25. Our system uses 46 bits with 26th binary point for the pair-wise force and 46 bits with 26th binary point of precision for coordinates of particles and performs much faster MD simulations without loss of accuracy.

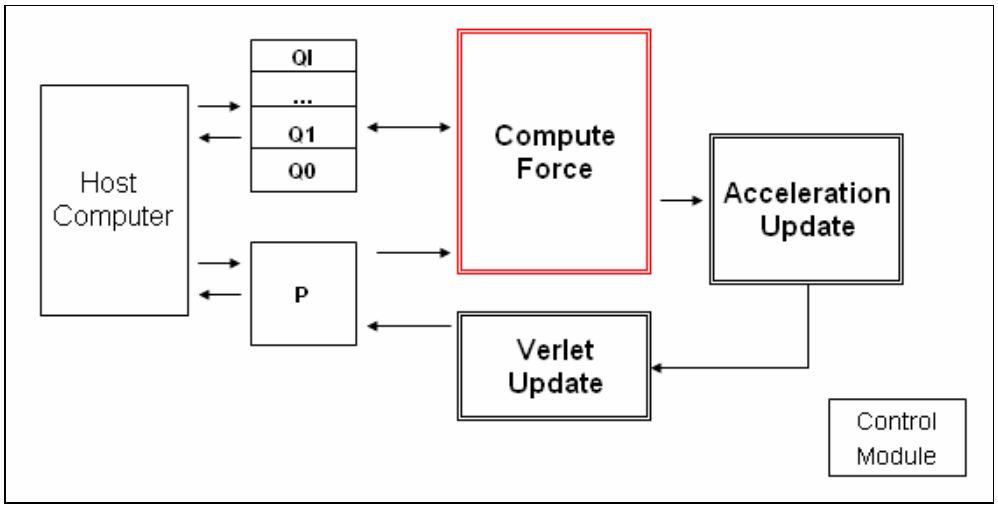


Figure 11. Block Diagram of MD Simulator

$$v(t) = v(t - \frac{\delta t}{2}) + \frac{\delta t}{2} \times a(t)$$

$$r(t + \delta t) = r(t) + \delta t \times v(t) + \frac{\delta t^2}{2} \times a(t)$$

$$v(t + \frac{\delta t}{2}) = v(t) + \frac{\delta t}{2} \times a(t)$$

Figure 12. Verlet algorithm [43]

Figure 11 shows the top level block diagram of the proposed architecture. Our system communicates with a host computer to update coordinates of particles during the MD simulation. The host computer stores coordinates and charges of particles in memory P and Q0~Ql respectively. Also, it calculates density of the finest grid as a preprocessing step of MCA method. *Compute Force* block is a main module to evaluate the forces(F0~Fl) and

potentials. Once the forces are evaluated, *Acceleration Update* block accumulates the forces and *Verlet Update* block updates position and velocity of particles. We are using Verlet algorithm [43] (Figure 12) to update the velocity.

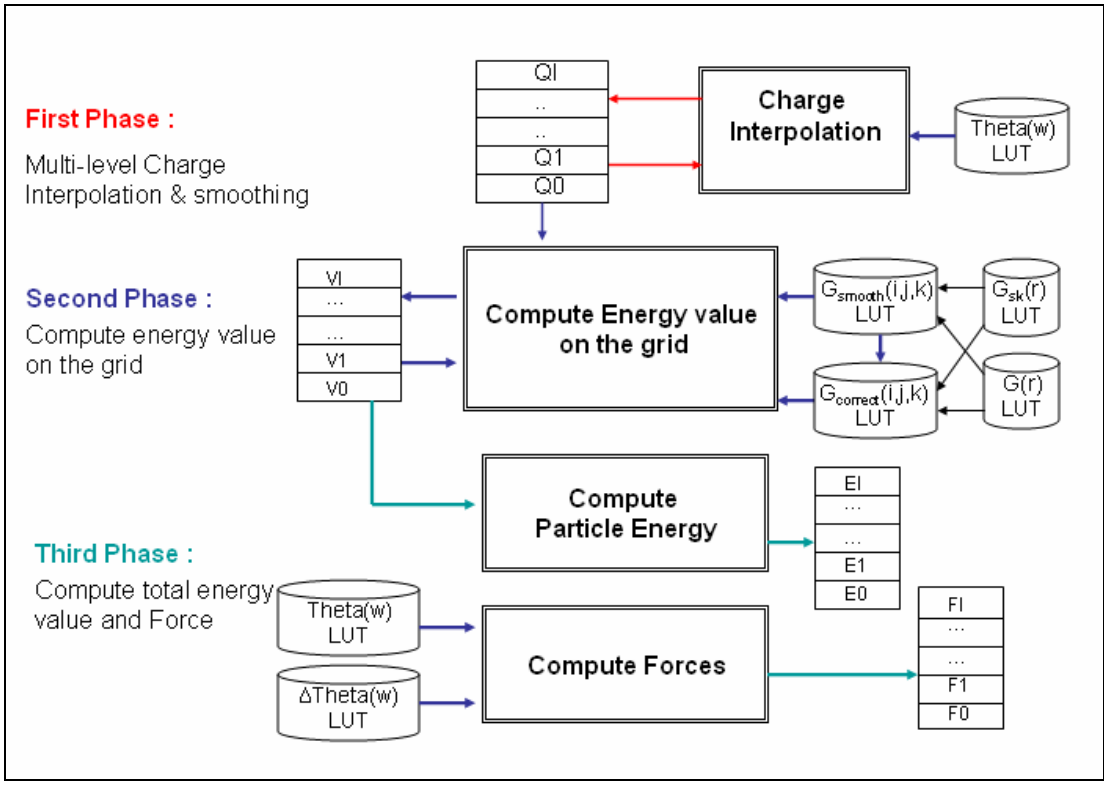


Figure 13. *Compute Force* Block of MD Simulator

Figure 13 depicts the block diagram of Compute force module shown in Figure 11. It performs three phases: 1) interpolation and smoothing which interpolate charge of particles to charge of finest grids and interpolate charge of grids to charge of coarser grid (steps 1 and 2 in Figure 10); 2) computing energy value on the grids (steps 3, 4 and 5 in Figure 10); 3) computing total energy value and force (step 6 in Figure 10).

Chapter 6
Implementing FPGA-based simulator for MD simulation

In Chapter 5, we described the process and architecture of our proposed FPGA-based simulator for the MD simulation. In this Chapter we present implementations of the FPGA-based simulator. We introduce three methods for designing a system in FPGA and provide more detail for the method that we utilize. In Section 6.1, we present two versions of the FPGA model using *SysGen* and *Simulink* on *Matlab* so that the models apply the Multigrid method to calculate non-bonded interactions. Our models are implemented on the target FPGA, Virtex-IV of Xilink Inc.

6.1. Methods for a system design in FPGA

There are three methods to design a system in Xilink FPGAs [32]. First method is using hardware description language, such as *VHDL* or *Verilog*. *VHDL* stands for VHSIC (Very High Speed Integrated Circuits) Hardware Description Language. In the 1980's the U.S. Department of Defense and the IEEE sponsored the development of this hardware description language (HDL) with the goal to develop very high-speed integrated circuit. Now it has become one of industry's standard languages used to describe digital systems. The other

widely used hardware description language is *Verilog*. Both are powerful languages that allow us to describe and simulate complex digital systems. Although these languages look similar as conventional programming languages, there are some important differences. A hardware description language is inherently parallel, i.e. commands, which correspond to logic gates, are computed in parallel, as soon as a new input arrives. An HDL program mimics the behavior of a physical, usually digital system. It also allows incorporation of timing specifications (gate delays) as well as to describe a system as an interconnection of different components[44]. This method provides complete control of the design of implementations and tradeoffs.

The second method is using pre-designed functional units, called *cores*. Xilinx provides *Core Generator* to customiz and generate these functional units. The pre-designed functional units such as adder, subtractor, multiplier or divider are available in such a way that they can be customized for a particular use. If designers can generate and customize their functional units by *Core Generator* and the cores meet their specification, the designers do not need to re-design. We describe the advantages and disadvantages of each method in Table II.

We choose to use the third method, *System Generator* to design an FPGA implementation since *System Generator* helps to reduce the gap between system designer and FPGA implementer. System designers write an algorithm in pseudo code, using filters, certain C code and certain precision. They may be familiar with Digital Signal Processing (DSP) and

Simulink models, but may not know anything about FPGAs. Not only does he not know how to target an FPGA, he does not know how to take advantage of the FPGA architecture, or how to write a design to avoid a bad FPGA implementation. For example, when he is finished with his DSP design, he may have a working model in Simulink, but he must still design the same thing in VHDL, or he gives his design to an FPGA implementer who writes the VHDL for him. But if the implementer does not know about DSP, the implementer might end up using a core that does not do exactly what the designer wants,. The FPGA implementer is just trying to translate the pseudo code that came to him into VHDL for an FPGA.

TABLE II. Three methods for design in FPGAs

Method	Advantage	Disadvantage
Full VHDL/Verilog	• Portability • Complete control of the design implementation and tradeoffs • Easier to debug and understand a code that you own	• Can be time-consuming • Do not always have control over the Synthesis tool • Need to be familiar with the algorithm and how to write it • Must be conversant with the synthesis tools to obtain optimized design
Core Generator	• Can quickly access and generate existing functions • No need to reinvent the wheel and re-design a block if it meets specifications • Intellectual Property (IP) is optimized for the specified architecture	• IP does not always do exactly what you are looking for • Need to understand signals and parameters and match them to your specification • Dealing with black box and have little information on how the function is implemented

System Generator	• Huge productivity gains through high-level modeling • Ability to simulate the complete designs at a system level • Very attractive for FPGA novices • Excellent capabilities for designing complex testbenches • Hardware Description Language (HDL) Testbench, test vector and golden data written automatically • Hardware in the loop simulation improves productivity and provides quick verification of the system functioning correctly or not	• Minor cost of abstraction: does not always give the best result from an area usage point of view • Customer may not be familiar with Simulink • Not well suited to multiple clock designs • No bi-directional bus supported

MathWorks' Simulink is a visual data flow tool and presents an alternative to using programming languages for system design. This enables designers to visualize the dynamic nature of a system while illustrating their complete system in a realistic fashion with respect to the hardware design. Most hardware design starts out with a block diagram description and specification of the system, very similar to the Simulink design[33]. The main part of Simulink is the Library browser that contains all the available building blocks to the user. This library is expandable and each block is parameterizable. Users can even create their own libraries of functions they have created.

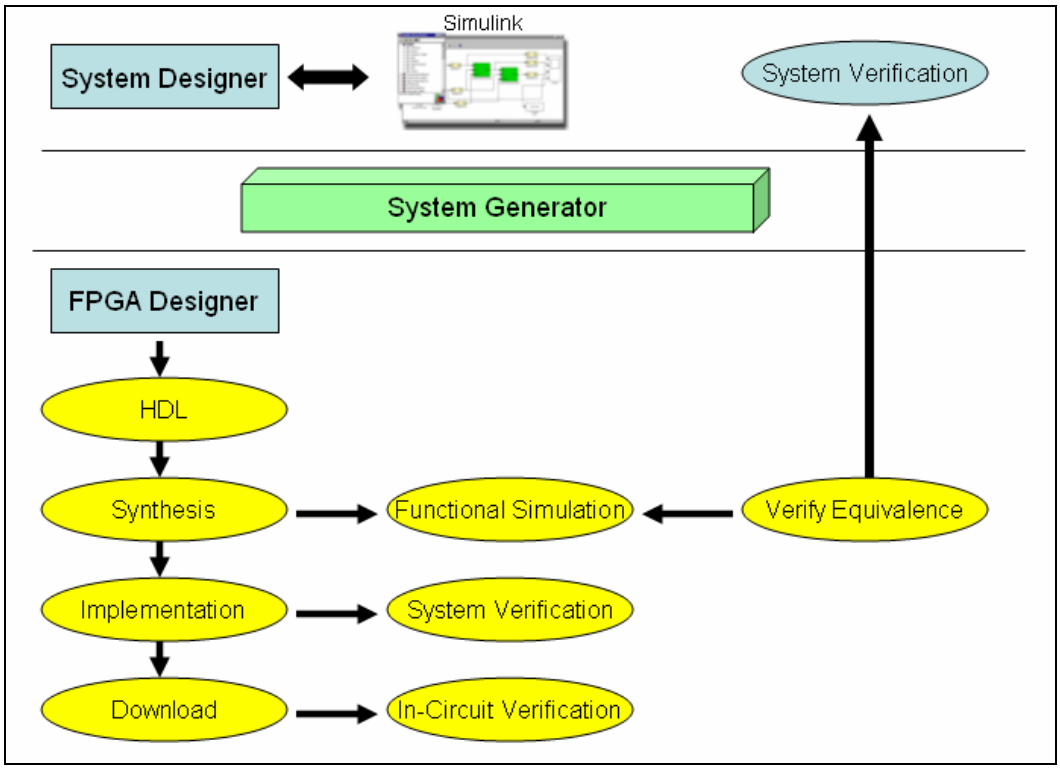


Figure 14. Design Flow in FPGAs

An important point of note about Simulink is that it can model concurrency in a system. Unlike the sequential manner of software code, the Simulink model can be seen to be executing sections of a design at the same time (in parallel). This notion is fundamental to implementing a high-performance hardware implementation.

Figure 14 shows FPGA design flow using Simulink and *System Generator*. In the Figure, *System Generator* is a plug-in to the Simulink environment, adding a Blockset to the Simulink library browser. *System generator* is an Industry's system-level design environment (IDE) for FPGAs and integrated design flow from Simulink to bit file. To provide system-level designers with a portal into the FPGA, *System Generator* taps into existing technologies

(MATLAB, Simulink, HDL synthesis, IP Core libraries, FPGA implementation tools), leveraging the MathWorks tool suite to provide the foundations for system design and the Xilinx FPGA design flow to implement the design.

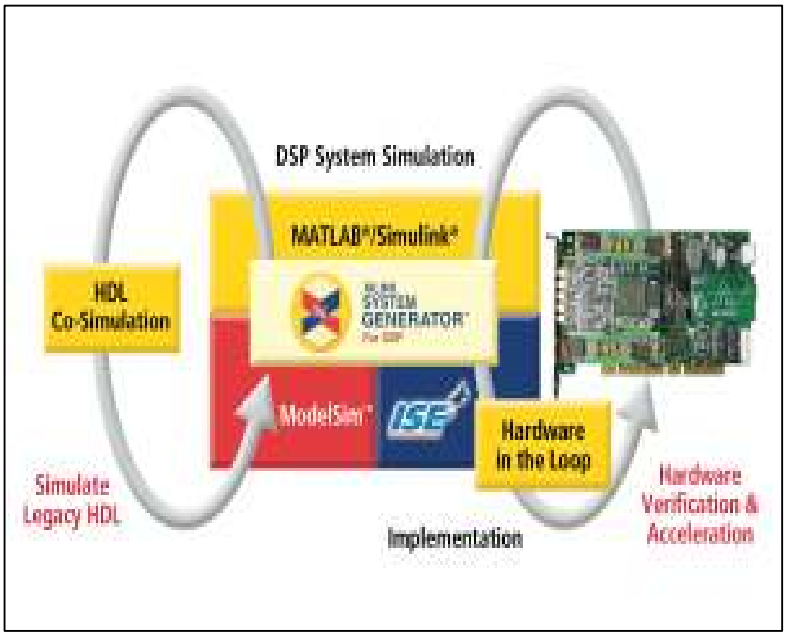


Figure 15. *System Generator* for DSP Platform Designs [32]

Figure 15 shows the process that *System Generator* performs for DSP platform design. The *System Generator* performs HDL co-simulation and Hardware in the loop simulation using black box block. The black box can be used to incorporate hardware description language (HDL) models into System Generator.

In this work, we choose the third method of the three methods to design an FPGA-based MD simulator, since it provides very productive design span by high-level modeling and quick verification of the system functioning. We use Simulink/MATLAB to design the

FPGA-based simulator and *System generator* to target FPGAs.

6.2. Implementation –Version 1

First version of the model is our preliminary and early work. Figure 14 presents an *Anterpolate* module that assigns charges of particles to charges of the finest grids. Anterpolate step is the most time consuming part of a MG method and it takes $O(K \cdot N)$ calculation time, where N is the number of atoms in the molecular system and K is a large constant that depends on the order of the interpolation function. *Anterpolate* module simulates 4th order Hermite interpolation and three grid levels. For each iteration of the MD simulation, the module consumes $128 \times N$ clock counts (each loop cycle consumes 128 sample periods).

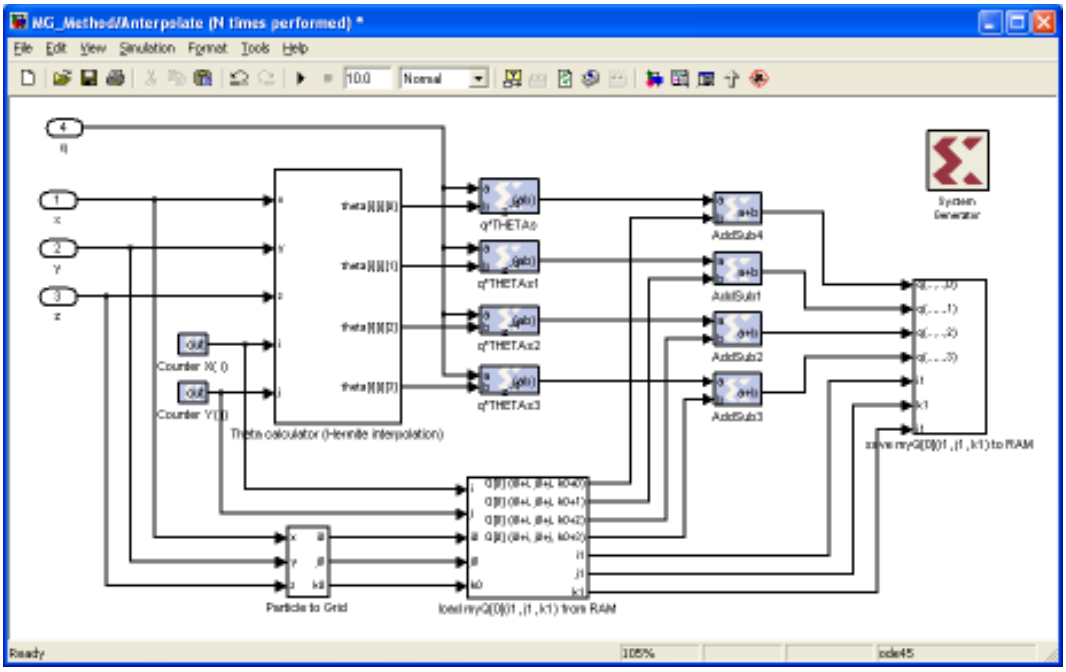


Figure 16. *Anterpolate* module

In Figure 16, we are using two counters (Counter X, Counter Y) instead of three for x, y, and z axis in order to reduce the sample periods. This results in an increase of logic units required, specifically three multipliers in the *Theta calculator* module, but it improves the performance.

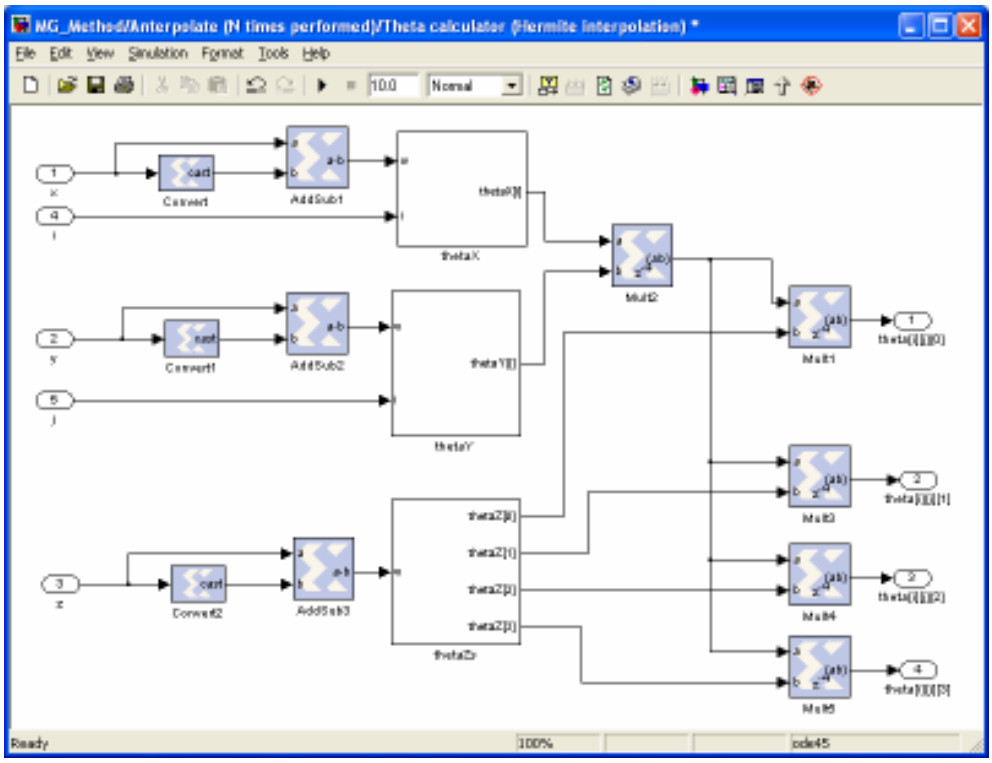


Figure 17. *Theta calculator* module in *Anterpolate* module

Figure 17 shows the *Theta calculator* module and the module provides all thetas for z axis in a subsystem (*thetaZs*) and reduces the sample period, but adds three multipliers. We could reduce *theta* calculation time to 32 × N clock counts (each loop cycle consumes 32 sample

periods) by removing one counter and add 15 more multipliers. Other modules of our first version of MD simulator are provided in Appendix.

The real calculation time of the MD simulation system depends on the speed of the processor. However, we reduce the floating point operations using table look-ups to calculate the interpolation functions and the MCA method itself improves the performance and accuracy as Table I shows.

Section 6.2 describes our improved version of the FPGA-based MD simulator and presents more detail of the modules.

6.3. Implementation - Version 2

We have optimized the first version and reduced the clock counts. Figure 18 presents an *anterpolate* module (Step 1 in Figure 10) that assigns charges of particles to charges of the finest grids. *Anterpolate* module also simulates 4th order Hermite interpolation and three grid levels. The module consumes (12 + *N)* clock counts and uses 12 look up tables to calculate *theta* values for Hermite interpolation.

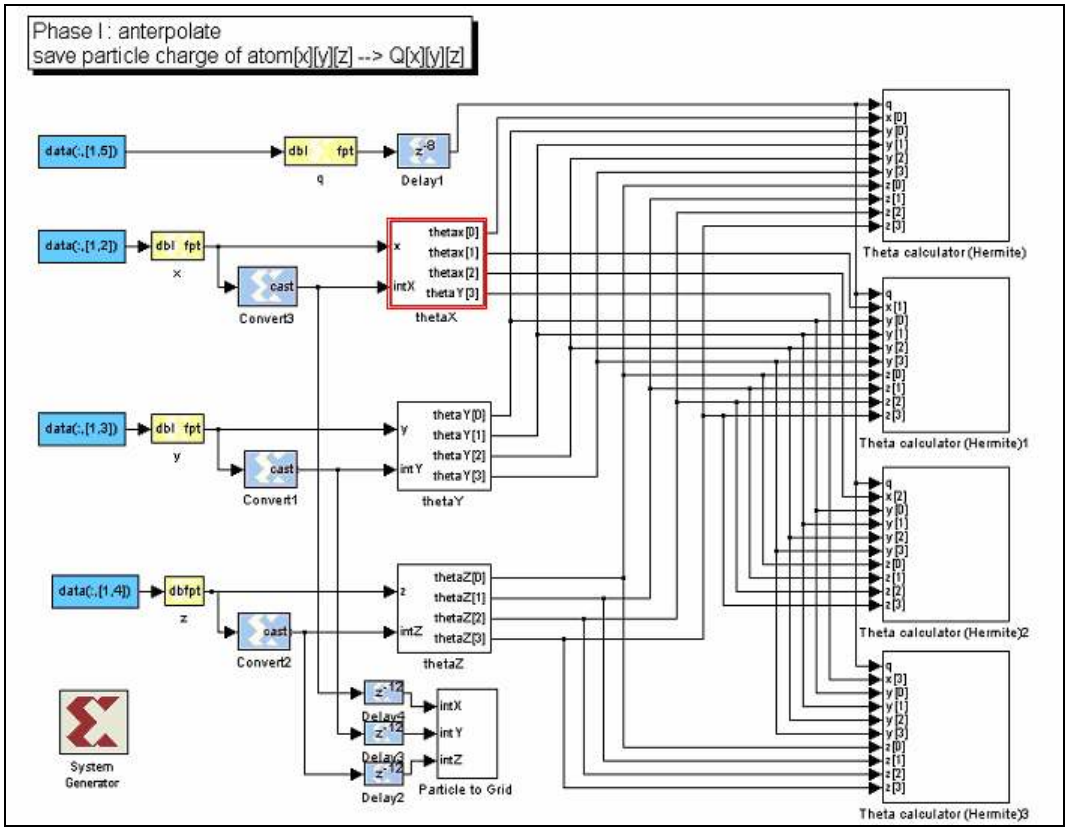


Figure 18. A*nterpolate* module

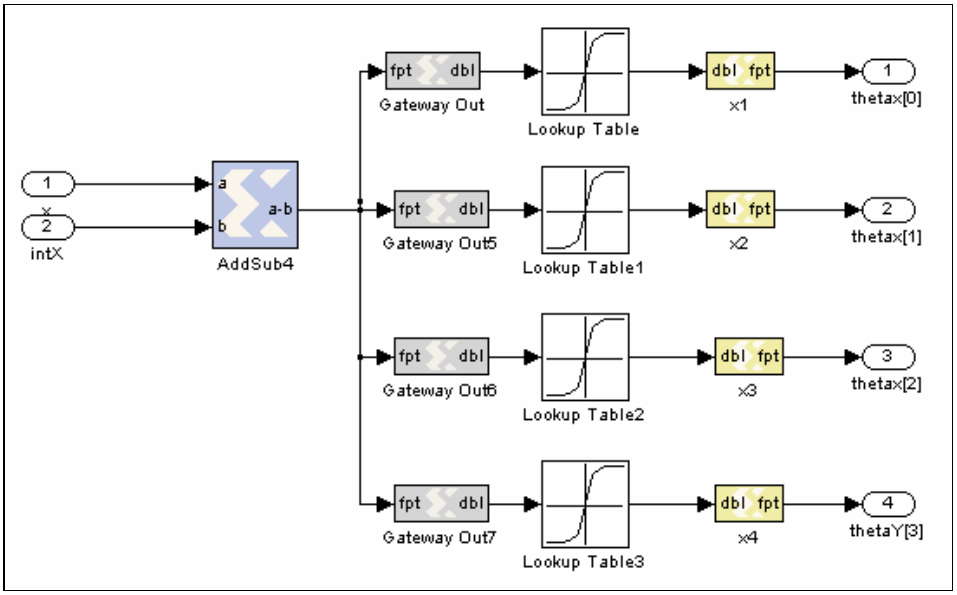


Figure 19. *thetaX* block in *anterpolate* module

Figure 19 shows *thetaX* block (red box in Figure 18) and the block provides *theta* values for X axis. This block uses four look up tables to generate theta values and the input of look up tables is weight which is distance of a grid point and particle. *thetaY* and *thetaZ* block also have similar patterns.

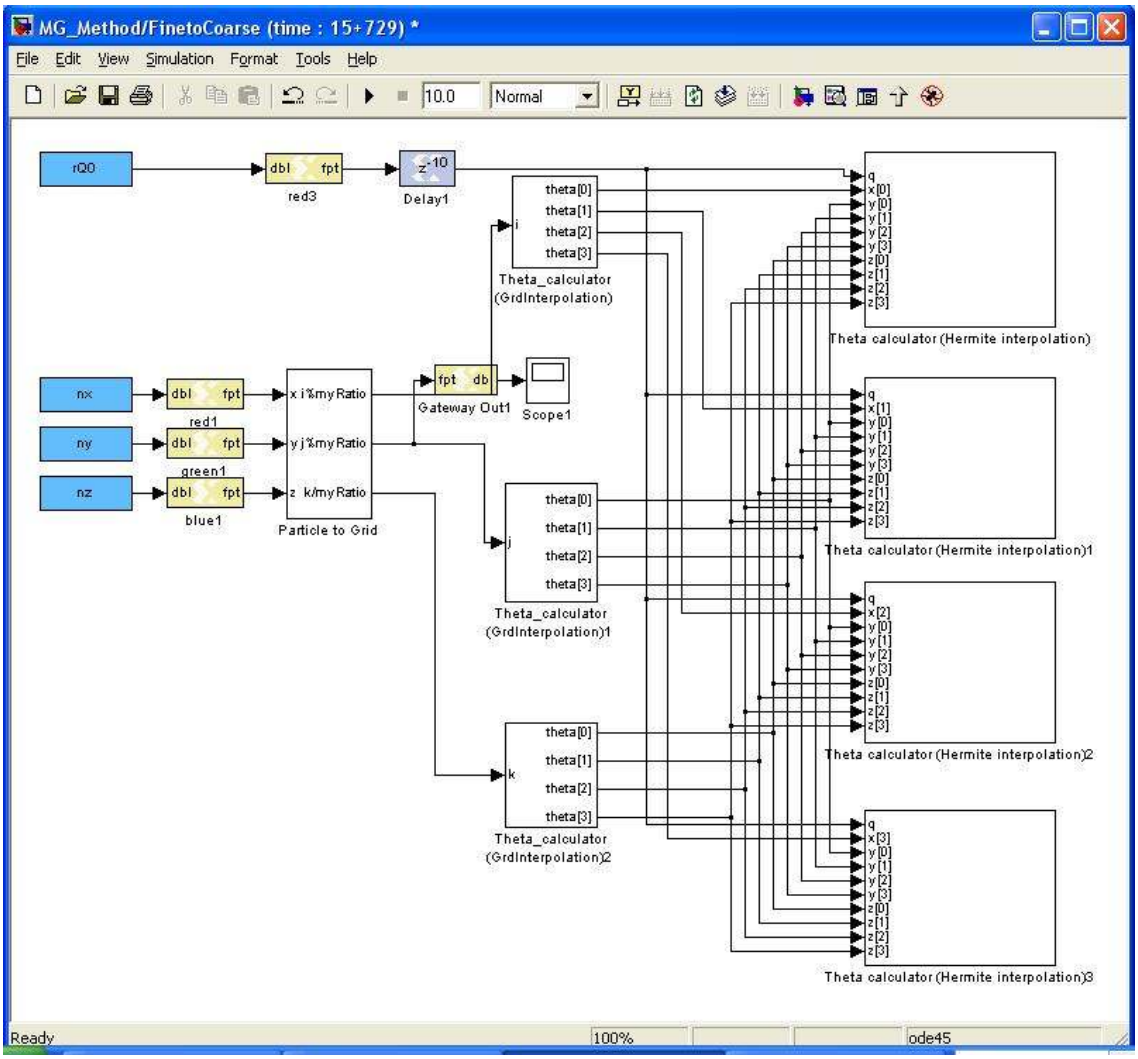


Figure 20. *FineToCoarse* module

Figure 20 presents an *FineToCoarse* module (Step 2 in Figure 10) which anterpolates

charges of grids to charges of coarser grids. A *FineToCoarse* module consumes (15 + number of grid points) clock counts at level L and the total number of clock counts for running the module is $\sum_{i}^{l-1}(15+N_x(i)\cdot N_y(i)\cdot N_z(i))$,

where Nx(i), Ny(i), Nz(i) = grid points at ith level and l = level of the MD simulation. We implement three grid levels.

The module uses 12 multiplexers to calculate theta values for Hermite interpolation. Since the module uses grid points and the position of grid points are evenly spread, we can use the multiplexers instead of look up tables. Theta Calculator (Hermite interpolation) block generates values multiplying thetas to update charges of grid points (Refer to Appendix)

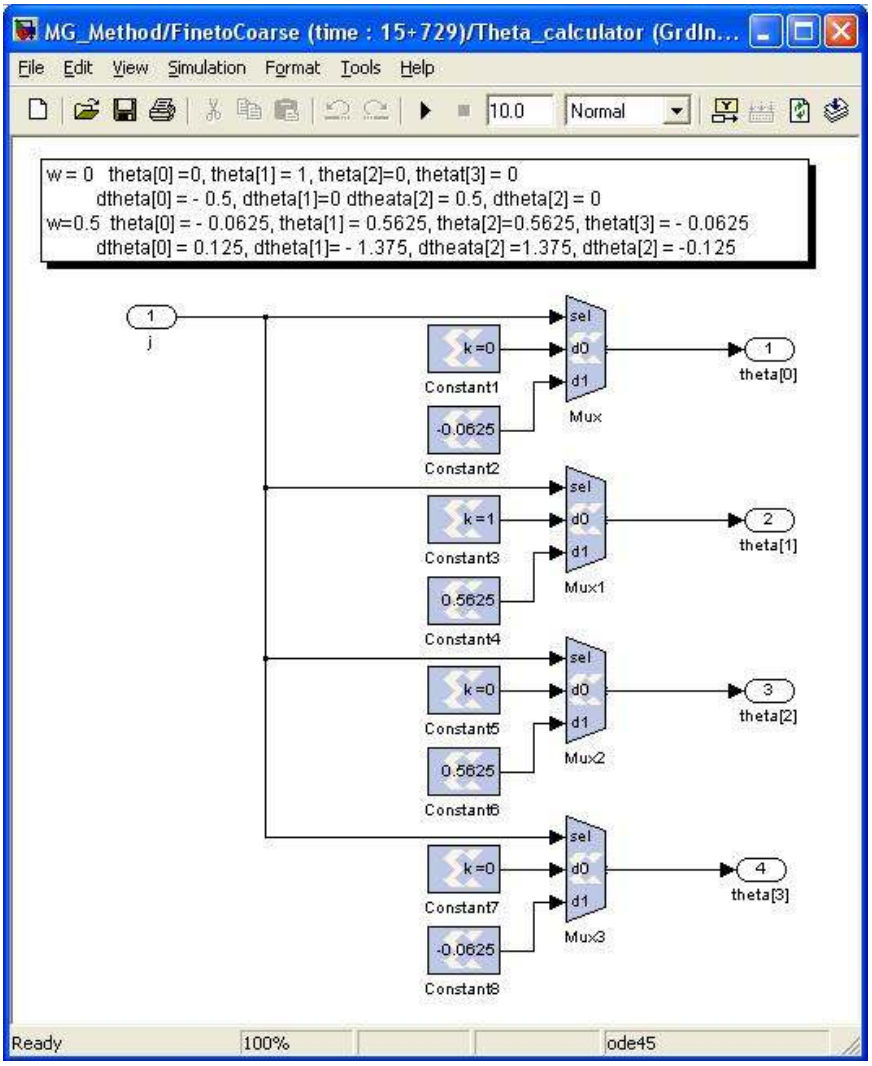


Figure 21. *Theta_calculator (GridIntepolation)* block of *FineToCoarse* module

Figure 21 shows *Theta_calculator(GridInterpolation)* block of *FineToCoarse* module (Figure 20). The block provides *theta* values using index i, which is x coordinate of grid positions and we use four multiplexers since we perform 4the interpolation function. *Theta_calculator(GridInterpolation)1* and *Theta_calculator(GridInterpolation)2* block have similar patterns.

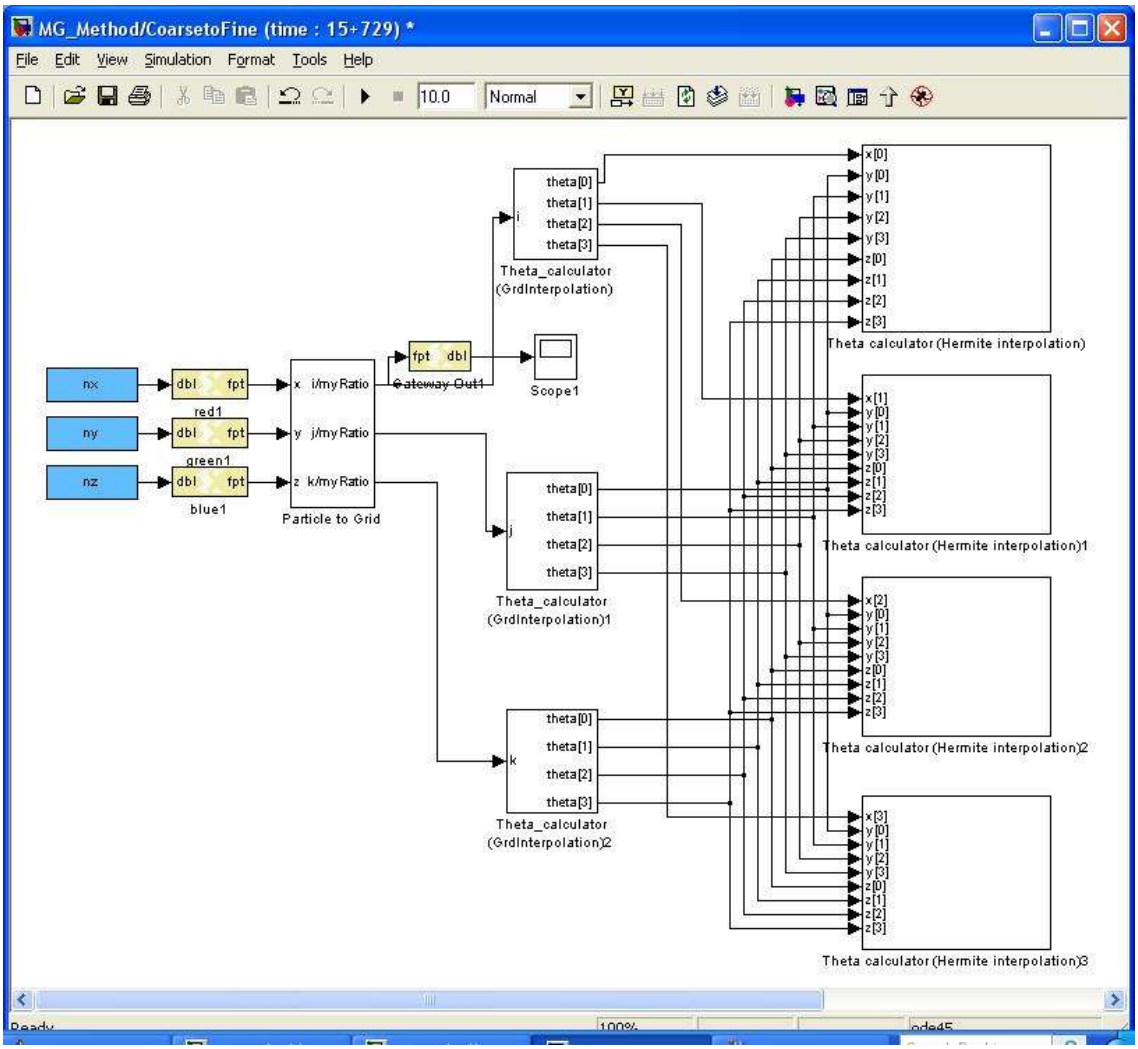


Figure 22. *CoarseToFine* module

Figure 22 presents an *CoarseToFine* module (Step 4 in Figure 10) which interpolates potentials of grids to potentials of finer grids. This module also performs 4th order Hermite interpolation and 3 grid levels. Each *CoarsToFine* FineToCoarse module consumes (15 + number of grid points) clock counts at level L and the total number of clock counts for the module is $\sum_{i}^{l-1}(15+N_x(i)\cdot N_y(i)\cdot N_z(i))$,

where Nx(i), Ny(i), Nz(i) = grid points at i^{th} level and l = level.

The module uses 12 multiplexers to calculate *theta* values for Hermite interpolation. Other sub blocks of this module are similar with blocks of *FineToCoarse* module (Refer to Appendix)

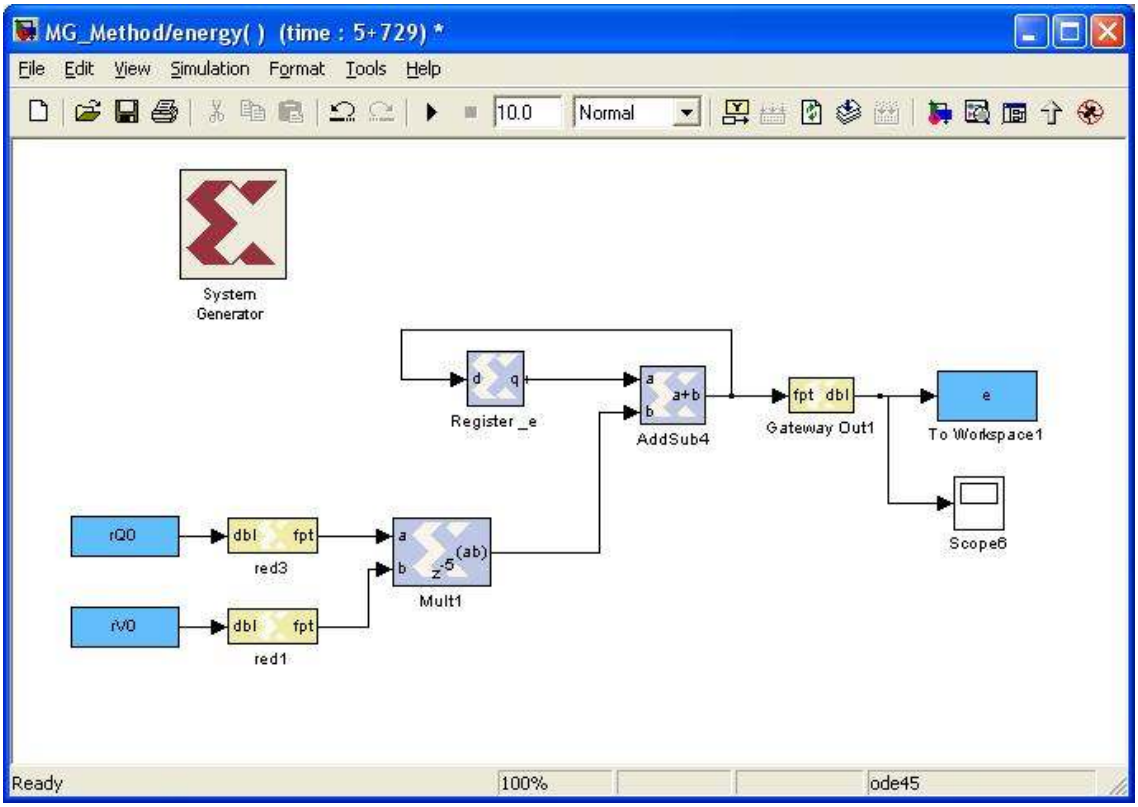


Figure 23. *energy* module

Figure 23 presents an *energy* module (Step 6 in Figure 10). The module updates total energy values by multiplying charges of finest grid (level 0) and potential of the finest grids (level 0). It consumes ($5+N_x(0)\cdot N_y(0)\cdot N_y(0)$) clock counts to complete.

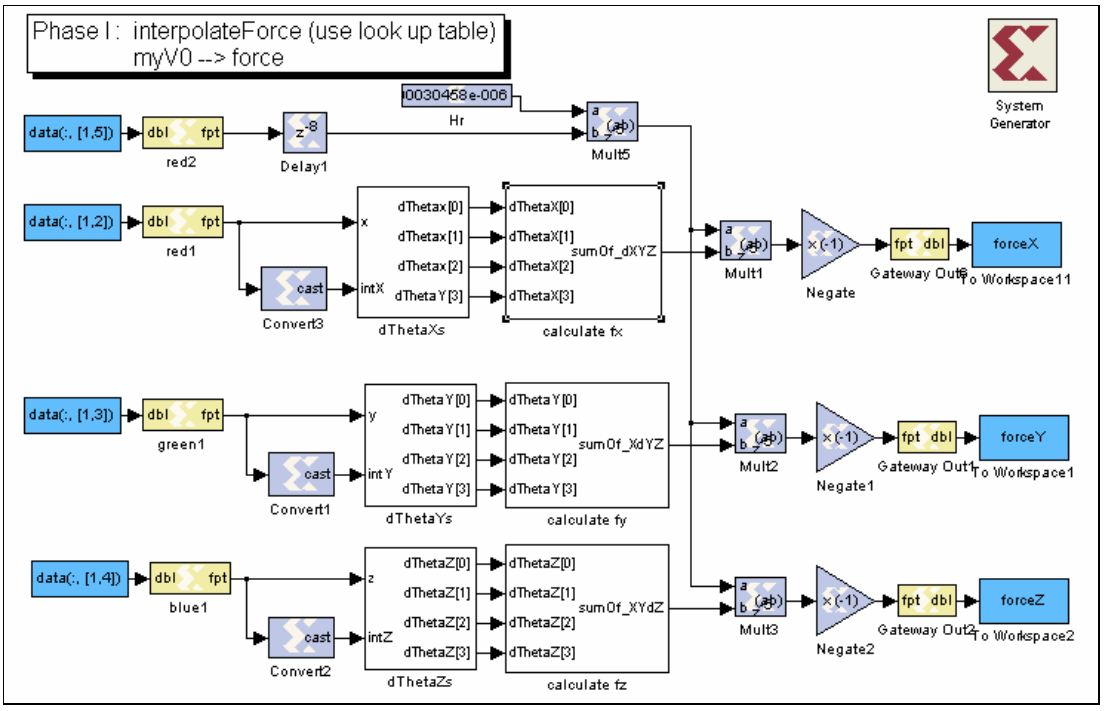


Figure 24. *Interpolate* module

Figure 24 presents the *interpolate* module (*Compute Particle energy* block in Figure 10) that evaluate forces. *Interpolate* step is also a time consuming part of the MG method like Anterpolate step. Our *interpolate* module simulates 4th order Hermite interpolation and 3 grid levels. The module consumes (16 + *N*) clock counts and uses 12 look up tables to calculate Δ*theta* values for Hermite interpolation module.

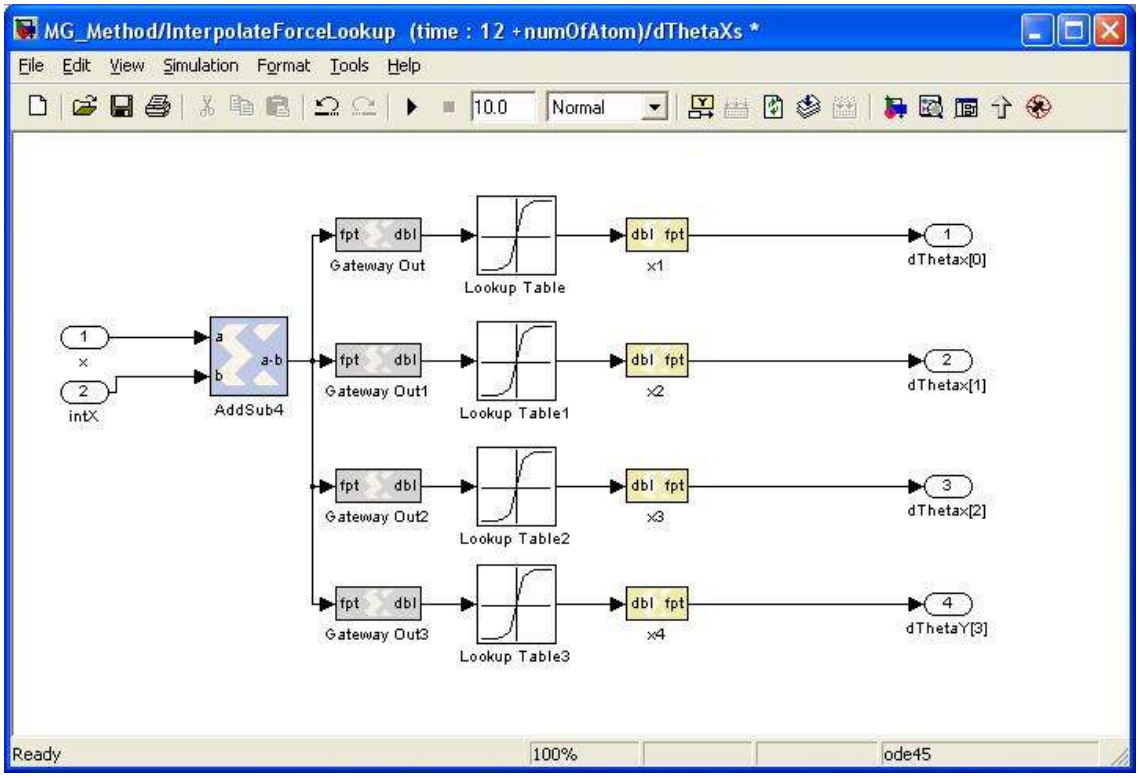


Figure 25. *dThetaXs* block of *Interpolate* module

Figure 25 shows the *dThetaXs* block and the block provides Δthetas for X axis. This block uses four look up tables to generate *theta* values and the input of look up tables is weight which is distance of a grid point and particle. *dThetaY* and *dThetaZ* block also have similar patterns.

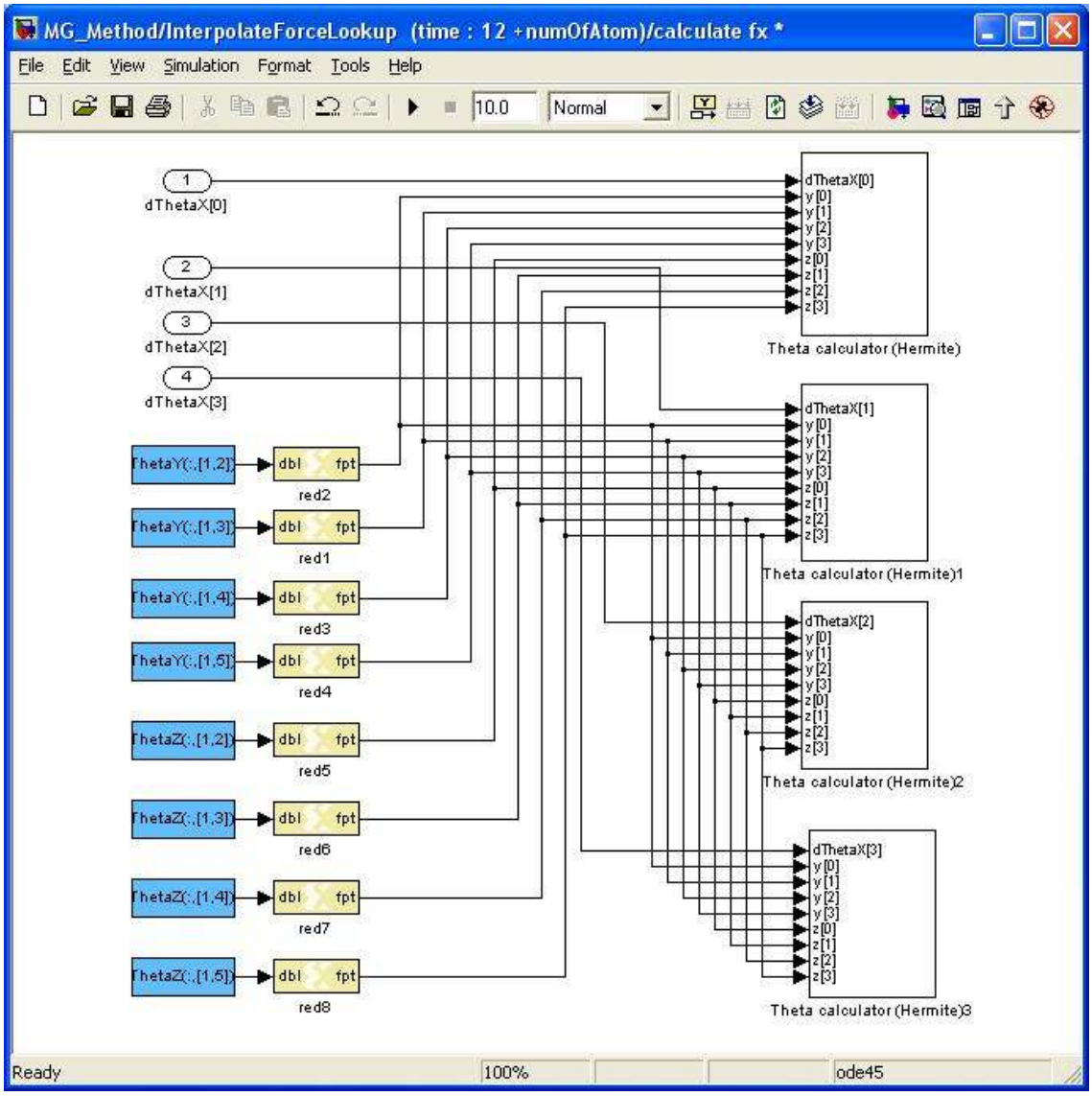


Figure 26. *CalculateFx* block of *Interpolate* module

Figure 26 present *CalculateFx* block which generates values multiplying thetas and Δthetas to update the forces of MD simulations. *Theta Calculator(Hermite)* block in Figure 26 performs the multiplication of thetas and Δthetas (Refer to Appendix). Other modules for steps in Figure 10 are computed in host computer.

6.4. Results and Analysis

In this section we describe the simulation results of a FGPA-based simulator and analyze its performance. We analyze modules of the simulator and software implementation on Protomol 2.0 [41]. Protomol 2.0 provides Simple, Particle Mesh Ewald (PME), Full Ewald and Multigrid (MG) implementation for the MD simulation and a platform for comparing various methods. Table III compares the performance of the FPGA-based simulator versus the software implementation. FPGA-based simulator (second version of the model) needs 12+N clock counts and 16+N clock counts to run steps1 and 6 in Figure 10 on the FPGA board. Steps 2 and 4 consume (1) in Table III clock counts and it is a function of the number of grid points at each level. Steps 2 and 3 in Figure 10 are performed on host computer and its running time is also a function of the number of grid points at each level. Software implementation consumes $K{\cdot}N{\cdot}\text{order}^3$ instructions to run steps1 and 6 in Figure 10. Steps 2, 3 and 4 in Figure 10 need calculation time which is a function of the number of grid points at each level.

For a large MD system, steps 1 and 6 in Figure 10 constitute the majority of the time required for the simulation. Therefore, we can focus our attention to these two steps, as the time required for the other steps are negligible in comparison. Our tests were performed for a Calcium molecule with 309 atoms and Basic Pancreatic trypsin inhibitor (BPTI) with water that has 1101 atoms and 14281 atoms. Table IV breaks down the timing for steps 1 and 6 for

both the software implementation and the FPGA implementation. It compares accuracy of software implementation versus FPGA-based simulator. The comparisons were made for multiple experiments over varying sizes of the molecular systems.

Software implementation has been implemented in C++ and runs on Intel Core Duo T2300/1.66 GHz processor. The FPGA-based simulator runs on Virtex IV which has clock speed 500MHz. Both solution use MCA method and 4th order Hermite interpolation with three grid levels. The FPGA-based simulator runs 900 times faster than software implementation to perform steps1 and 6 and about 10 to 100 times faster than software implementation depending on simulation accuracy desired without loss of accuracy. Table V shows the resources used to implement our proposed FPGA-based simulator. The simulator use 24 look up tables, 24 multiplexers and 773 multipliers.

TABLE III. Comparison of performance

N = Number of atoms, order = interpolation order, Nx(i), Ny(i), Nz(i) = grid points at ith level , l = level

	FPGA-based simulator		Software implementation (instruction)
	FPGA (clock count)	Host Computer (instruction)	
Step 1	12 + N		$K \cdot N \cdot order^3$
Step 2	finetoCoase: $\sum_{i}^{l-1}(15 + N_x(i) \cdot N_y(i) \cdot N_z(i))$ (l) Correction :	$K \cdot \sum_{i}^{l-1}(N_x(i) \cdot N_y(i) \cdot N_z(i))$ $K \cdot \sum_{i}^{l-1}(N_x(i)^2 \cdot N_y(i)^2 \cdot N_z(i)^2)$	$K \cdot \sum_{i}^{l-1}(N_x(i) \cdot N_y(i) \cdot N_z(i) \cdot order^3)$ $K \cdot \sum_{i}^{l-1}(N_x(i)^2 \cdot N_y(i)^2 \cdot N_z(i)^2)$
Step 3		$K \cdot N_x(l)^2 \cdot N_y(l)^2 \cdot N_z(l)^2$	$K \cdot N_x(l)^2 \cdot N_y(l)^2 \cdot N_z(l)^2$
Step 4	$\sum_{i}^{l-1}(15 + N_x(i) \cdot N_y(i) \cdot N_z(i))$		$K \cdot \sum_{i}^{l-1}(N_x(i) \cdot N_y(i) \cdot N_z(i) \cdot order^3)$
Step 5	$5 + N_x(0) \cdot N_y(0) \cdot N_z(0)$		$K \cdot N_x(0) \cdot N_y(0) \cdot N_z(0)$
Step 6	16 + N		$K \cdot N \cdot order^3$

TABLE IV. Time and accuracy results for the software implementation vs. FPGA-based simulator (N = Number of atoms)

		Case I (N = 309)	Case II (N = 1101)	Case III (N = 14281)
Software Implementation	Step1 (sec)	0.00030	0.0015	0.0210
	Step6 (sec)	0.00084	0.0026	0.0353
	Total (sec)	1.10e-003	4.10e-003	5.63e-002
	Accuracy	*0.0122*	*0.00177*	*1.7382 e-04*
FPGA-based simulator	Step1 (sec)	6.42 e-7	2.2260e-006	2.8586e-005
	Step6 (sec)	6.50e-7	2.2340e-006	2.8594e-005
	Total (sec)	1.2920e-006	4.4600e-006	5.7180e-005
	Accuracy	*0.0138*	*0.00183*	*1.6195 e-04*

TABLE V. Required FPGA resources

	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Total
Look up tables	12					12	24
Multiplexer		12		12			24
Multiplier	192	192		192	1	196	773

Previous FPGA simulators use the Direct method for Coulombic force and the calculation time of Direct method. However our proposed MCA method requires much less calculation time than the Direct method with improved accuracy. In addition, we provide the results of various experiments and prove that the simulator achieves better accuracy as well as better performance in terms of time.

Chapter 7
Proposed Reconfigurable Mesh Algorithms

FPGA-based simulators [15, 17, 18] lead that feasibility of exploiting Reconfigurable models on a large scale problems such as the MD simulation. Compared to supercomputing systems, the circuits of FPGAs can be customized to MD simulations and reuse the same logic resources in different configuration. They accelerate MD simulations about 100 times. But it is hard to show the scalability of FPGA-based simulator due to the lack of facilities and usually FPGA implementations demonstrate a prototype of applications.

In this Chapter, we present another reconfigurable model, Reconfigurable Mesh (R-mesh) to show not only efficiency but also scalability of reconfigurable model. Any MD simulation repeatively evaluates forces until the energy reaches equilibrium. If the function for evaluating forces requires O(N^2) time complexity such as the Direct method, the entire time complexity is $K\cdot$ O(N^2), where K is the number of iterations and is usually a large number. Therefore it is very important to reduce the time for evaluating forces. We are presenting the R-mesh algorithms for the Direct method in Section 7.1 and the Multigrid method in Section 7.2. The algorithms require much less calculation time to perform MD simulations. Section 7.3 summarizes the proposed R-mesh algorithms and provides theorems for the algorithms.

7.1. Algorithms for Direct method

The Direct method uses Equation 1 to evaluate electrostatic potential and takes O(N^2) time on a general purpose processor where N is number of atoms in a molecular system. We develop two versions (Algorithm 1 and Algorithm 2) of R-Mesh algorithms for the Direct method. Algorithms 1 and Algorithm 2 are the main modules of the MD simulations. Algorithm 1 requires K·O(N) time complexity on an N processor linear mesh. In Algorithm 1, *p(i)* and *q(i)* are local data for the position and charge of atoms. *DirectComputeForce( )* evaluates forces of each atom and is described in Algorithm 1-1. *doOneAtomPair(i, j)* in Algorithm 1-1 evaluates the potential energy between atom *i* and atom *j*. *UpdateVelocity()* and *UpdatePosition()* updates the velocity and position of atoms and takes O(1) time.

Algorithm 1 (MD simulation with direct method)
1. Model : N processors (N is # of atoms) 1-dimensional R-Mesh
2. Input: proc(i) store p(i)={p_0, p_1, …, p_{N-1}} and q(i)={q_0, q_1, …, q_{N-1}}
3. Output : proc(i) store force(i)={$force_0$, $force_1$, … $force_{N-1}$} and updated p={p_0, p_1, …, p_{N-1}} and proc(0) store total energy in E

```
begin     // K×O(N) (K is the number of iteration)
MDSimulation_Direct ( )
        while E is not changed do
                DirectComputeFoce(p, q) //   O(N)
                UpdateVelocity(pos, force, E)
                UpdatePostion(pos, force, E)
                proc(i) broadcast updated position of atom i and force to all process //   O(N)
      end_while
end
```

Algorithm 1-1 (DirectComputeForce)

1. Model : N processors (N is # of atoms) 1-dimensional R-Mesh
2. Input: proc(i) store p(i)={p_0, p_1, …, p_{N-1}} and q(i)={ q_0, q_1, …, q_{N-1}}
3. Output : proc(i) store force(i)={f_0, f_1, …, f_{N-1}} and Proc(0) store total energy in E

```
begin     // O(N)
DirectComputeForce( )
        Step 1) each proc(i)
                  for   j←1 to N-1 do
               force(i) = force(i) + doOneAtomPair(i, j)    // O(1)
                    end_for
         Step 2)   (i)   = e(i)   + Calculate_Energy(force(i))    // O(1), calculate energy for atom i
         Step 3)   compute E = e(0)+e(1) + … +e(N-1) with N R-mesh    // O(logN) [1]
               and proc(0) store E
end
```

The Direct method uses Equation 1 to evaluate electrostatic potential and takes O(N^2) time on a general purpose processor where N is number of atoms in a molecular system. We develop two versions (Algorithm 1 and Algorithm 2) of R-Mesh algorithms for the Direct method. Algorithms 1 and Algorithm 2 are the main modules of the MD simulations. Algorithm 1 requires K·O(N) time complexity on an N processor linear mesh. In Algorithm 1, *p(i)* and *q(i)* are local data for the position and charge of atoms. *DirectComputeForce( )* evaluates forces of each atom and is described in Algorithm 1-1. *doOneAtomPair(i, j)* in Algorithm 1-1 evaluates the potential energy between atom *i* and atom *j*. *UpdateVelocity()* and *UpdatePosition()* updates the velocity and position of atoms and takes O(1) time.

Algorithm 2 (MD simulation with direct method II)

1. Model : N*N processors (N is # of atoms) 2-dimensional R-Mesh
2. Input: proc(0, j) store pj and qj and proc(i, 0) store pi and qi
3. Output : proc(0, i) store force(i) and proc(0,0) store E and proc(i, j) store new p(i,j)

```
begin     // K×O(logN) (K is the number of iteration)
MDSimulation_Direct ( )
          for energy is not changed do
                    DirectComputeFoceII( ) //    O(logN)
                    proc(i, j) run UpdateVelocity(p, force, E)
                    proc(i, j) run UpdatePostion(p, force, E)
        end_for
end
```

Algorithm 2-1 describes *DirectComputeforceII()*. The module evaluates forces of each atom with $N \times N$ processors and takes O(log$N$) time complexity. Step 4 in Algorithm 2-1 describes the process that uses each row to compute force of atom i and the steps for calculating *force(i)* is log$N$. Since each row can perform the process independently, overall time complexity is O(log$N$).

Algorithm 2-1 (DirectComputeForceII)

1. Model : N*N processors (N is # of atoms) 2-dimensional R-Mesh
2. Input: proc(0, j) store p(j) and q(j) and proc(i, 0) store p(i) and q(i)
3. Output : proc(0, i) store force(i) and proc(0,0) store E and proc(i, j) store new p(i,j)

begin // O(*logN*)

DirectComputeForceII ( )

Step 1) proc(0, j) send pj and qj to column bus j

Pos_A(i, j) ← pj, Q_A(i, j) ← qj

Step 2) proc(i, 0) send pi and qi to row bus j

Pos_B(i, j) ← pj, Q_B(i, j) ← qj

Step 3) proc(i,j) run

temp(i, j) ← doOneAtomPair(Pos_A(i, j), Pos_B(i,j), Q_A(i,j), Q_B(i,j)) // O(1)

Step 4) compute force(i) = temp(i, 0)+..+temp(i, N-1) with row bus i

and proc(0, i) store force(i) //O(log*N*) [1]

Step 5) proc(0, i) store e(i) = e(i) + Calculate_Energy(force(i)) // O(1)

Step 6) compute E(i) = e(0)+..+e(N-1) with *N*N R*-Mesh and proc(0, 0) store E // O(1) [1]

end

7.2. Algorithms for Multigrid method

The Multigrid method takes O(*N*) time on a general purpose processor, where *N* is the number of atoms in a molecular system. We developed an R-Mesh algorithm for the MG method that requires O(*r*)+O(log*M*) time complexity on an $X \times Y \times Z$ 3-dimensional R-Mesh, where *r* is *N/M* and $M = X \times Y \times Z$ is the number of finest grid points applied to the MG method for a given parameter. *M* is determined by three factors, size of the finest grid, molecular system size and interpolation order. To achieve accurate simulation results, it is important to choose appropriate values for the factors. Since molecular systems have various system size

and number of atoms, it is hard to find optimal parameter values. The value of *M* is usually much smaller compared to *N* unless the molecular system to be simulated is very small. For example, MG method determines finest grid points to (13, 13, 13) for a the molecular system with *N* =309 atoms to achieve 0.0008 relative accuracy [18]. In this case M is 13×13×13 = 2,197. Large molecular system that has *N*=14,281 atoms determines finest grid points (21, 21, 21) to achieve 0.0005 relative accuracy [10]. In this case M is 21×21×21=9281 which is very small compared to the number of atoms (14,281). As mentioned earlier, MD simulations are usually performed for large molecules that include over 10,000 atoms. As the number of atoms is larger, *M* becomes much smaller as compared to *N*.

Algorithm 3 is the main module of the MD simulation and requires $K\cdot(O(r)+O(\log M))$ time complexity. The main module is similar to Algorithm 1, but with a preprocessing function (Algorithm 3.1) that distributes atoms to the nearest 64 processors. Since we are using 4^{th} hermite interpolation function, only 64 (4×4×4) processors correspond to the closest grid points to atoms. The function runs based on the flag (*CheckChangedGrid*) that is assigned by *CheckGridPoint( )*. This function checks the new position and its grid point. Usually the atoms retain their previous grid points assigned, so the calculation time of *preprocessing( )* is negligible over the entire simulation.

Algorithm 3 (MD simulation with Multigrid method)

1. Model : M processors (X×Y×Z R-Mesh, M=X×Y×Z is # of finest grid point)
2. Input: proc(i, j, k) hold store p(i,j,k)={ p_{start}, .. , $p_{start+c-1}$} and q(i,j,k)={ q_{start}, .., $q_{start+c-1}$}, which *start* = i*c +j*X*c+k*X*Y*c and *c* = N/M
3. Output : proc(i, j, k) store force(i,j,k)={f_0, f_1.. f_r}, p(i,j,k)={p_0, p_1.. ,p_r} and proc(0,0,0) hold E, *r* is number of atoms assigned in proc(i, j, k),

```
begin     // K×O(1) (K is the number of iteration)
MDSimulation_Multigrid ( )
          while energy is not changed do
                    if(CheckChangedGrid == true)
               Preprocessing( )          // O(N)
                    End_if
               MultigridComputeForce(p(i,j,k), q(i,j,k))
                    proc(i, j, k) run UpdateVelocity(p(i,j,k), force(i,j,k), E)
                    proc(i, j, k) run UpdatePostion(p(i,j,k), force(i,j,k), E)
                    proc(i, j, k) set CheckChangedGrid ← CheckGridpoint(p(i,j,k))
       end_while
end
```

Algorithm 3-1 describes *preprocessing( )* that distributes information of atoms to nearby processors. proc(*i*, *j*, *k*) represents grid point (*i, j, k*) at level 0. *calGridpoint (start+m,* $p_{start+m}$*)* returns *grid_pos* and atom *start+m* assigned to *grid_pos* to interpolate. *calThetas(grid_pos(i,j,k),* $p_{start+m}$*)* calculates *thetas* and we use 4^{th} hermite interpolation function to calculate thetas. *Anterpolate( )* module (Algorithm 3-2-1) uses this information to calculate Q_0 (charge of finest grid). This algorithm takes O(*N*) time due to the *N* broadcasting steps required.

Algorithm 3-1 (Preprocessing)

1. Model : M processors

2. Input: proc(i, j, k) hold store p(i,j,k)={ p_{start}, .. , $p_{start+c-1}$} and q(i,j,k)={ q_{start}, .., $q_{start+c-1}$}, which *start* = i*c +j*X*c+k*X*Y*c and *c* = N/M

3. Output : proc(i, j, k) store D(i,j,k) = {d_0, d_1.. d_r}, which d_m = (index, p, q thetas, grid_pos), *r* is number of atoms assigned in proc(i, j, k)

```
begin
Preprocessing ( )
        If D(i,j,k)'s grid_pos is changed          //    O(N)
            for    m ← 0 to c-1 do            // c = N/M
                    grid_pos← calGridpoint (start+m, p_start+m )
                    thetas← calThetas(grid_pos(i,j,k), p_start+m )
                    D(i,j,k).d_m = (start+m, p_start+m,, q_start+m thetas, grid_pos)
            end_for
                    send D(i,j,k).d_m to proc(D(i,j,k).d_m.grid_pos) //N broadcasting times
      else                                  //O(1)
            keep previous D(i,j,k)
      end_if
end
```

MultigridComputeForce() described in Algorithm 3-2 evaluates the forces of each atom. Each processor represents the grid points for the finest grid. It consists of 6 steps. Step 1 is *Anterpolate( )* to interpolate weights for the position of atoms and anterpolate the charge *q* onto the finest grid (level 0). Step 2 is coarsening that anterpolates the grid charge from the current *level* to *level+1*. Step 3 is computing the potential for the top grid level. Step 4 is interpolating the grid charge from *level* to *level-1*. Step 5 is computing the energy of the top grid level. Step 6 is interpolating the force from grid level 0.

Algorithm 3-2 (Multigrid method for MD simulation with n atoms)

1. Model : M processors (X×Y×Z R-Mesh, M=X×Y×Z is # of finest grid point)

2. Input: proc(i, j, k) hold D(i,j,k) = {d_0, d_1.. d_r}, which d_m = (index, p, q thetas, grid_pos), *r* is number of atoms assigned in proc(i, j, k)

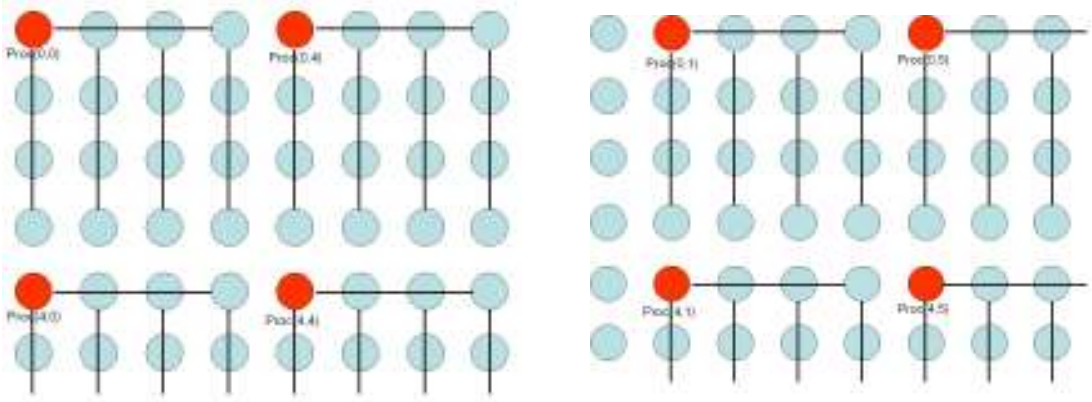
3. Output : proc(i, j, k) store force(i,j,k)={f_0, f_1.. f_r}, p(i,j,k)={p_0, p_1.. ,p_r} and proc(0,0,0) hold E, *r* is number of atoms assigned in proc(i, j, k),

```
begin
MultigridComputeForce(p, q)
        Step 1) Anterpolate( )                  // O(1)
        Step 2) for i  ← 0 to Levels-1
                    fineToCoarse ( i)           // O(1)
                    correction ( i)             // O( Nx(i)· Ny(i)· Nz(i))
                        end_for
        Step 3) direct ( )                      // O( Nx(L)· Ny(L)· Nz(L))
        Step 4) for i ← 0 to Level-1
                    coarseToFine (i)            // O(1)
                        end_for
        Step 5) energy(    )                    // O(logM)
        Step 6) interpolateForce (      )       // O(r)
end
```

Algorithm 3-2-1 describes *Anterpolate( )* that anterpolates and interpolates weights for the position of atoms and anterpolates the charge of atoms onto grid level 0. The main process of this module is *Step 1* which distributes charges of atom to grid level 0. Step 2 update temp(i,j,k) using *Cal_GridCharge(A). Cal_GridCharge(A)* function performs a equation, *A.d.q*×*A.d.theta.X*×*A.d.theta.Y*×*A.d.theta.Z.* This algorithm requires O(1) time complexity. Since each atom is interpolated to the nearest grids that are *order*×*order*×*order* grid points, broadcasting is performed on an *order*×*order*×*order* R-Mesh. The algorithm is designed so that there is no overlapping and processors can broadcast data simultaneously. The actual

number of broadcasting steps is $(order\text{-}1)^4$, where *order* is the order of the interpolation function. After broadcasting data, each processor updates $Q_0(i, j, k)$, which is the grid charges at level 0.

Figure 27 shows that processor(i, j) broadcasts data to the nearest 15 processors. Figure 27 (a) shows the first broadcasting step of processor(i, j) where i%4 ==0 and j%4 == 0. Then in the second step, the next group of nodes broadcast their data as shown in Figure 27 (b). These nodes have indices so that i%4 ==0 and (j-1)%4 == 0. This continues for a total of $(order\text{-}1)^4$ steps.



(a) (b)

Figure 27. Example of broadcasting in Algorithm 3-2-1 with *order* = 4
(a) Proc(0, 0), Proc(0, 4), proc(4, 0) and proc(4,4) broadcast data simultaneously
(b) Proc(0, 1), proc(0, 5), proc(4, 1) and proc(4,5) broadcast data simultaneously

Algorithm 3-2-1 (Anterpolate)

1. Model : M processors (X×Y×Z R-Mesh, M=X×Y×Z is # of finest grid point)
2. Input: proc(i, j, k) hold Q_0(i, j, k)=0 and hold D(i,j,k) = {d_0, d_1.. d_r}, which d_m=(index, p, q thetas, grid_pos), *r* is number of atoms assigned in proc(i, j, k)
3. Output : proc(i, j,k) update Q_0(i, j, k)

```
begin
Anterpolate ( )   // O(1)
    Step1) proc(i, j, k) broadcast D(i, j, k,) to the nearest processors
                        For rem←0 to order-1 do
                        For ix ←0 to order-1 do
                            For jx ←0 to order-1
                    For kx←0 to order-1 do
                                If  (i+ix)%order==rem  &&  (j+jx)%order==rem  &&
      (k+kx)%order==rem
                                proc(i,j,k) broadcast D(i,j,k) to proc(i, j, k)
                                    ~proc(i+order, j+order, k+order)   //O(1)
                end_if
                        end_for
                            end_for
                                end_for
        Step 2) If Proc(i, j, k) received D(i,j,k),
            update temp (i,j,k)← Cal_GridCharge(D(i,j,k))
        Step3) Q0(i, j, k)   ← Q0(i, j, k)   + temp(i,j,k)
end
```

Algorithm 3-2-2 is the coarsening process. It coarsens grid charges from *level* to *level+1* and requires O(1) time complexity. The actual number of broadcasting steps is (*order*-1)^3. *Coarsening( )* in Algorithm 3-2-2 expands broadcasting to 64 processors similar to Algorithm 3-2-1.

Algorithm 3-2-2 (FinetoCoarse(L))

1. Model : M processors (X×Y×Z R-Mesh, M=X×Y×Z is # of grid point at level L)
2. Input: proc(i, j, k) hold D(i,j,k) =(Q_L(i, j, k), theta) and L
3. Output : proc(i, j, k) update Q_{L+1}(i, j, k)

```
begin
FinetoCoarse (L) // O(1)
        Step1) proc(i, j, k) broadcast D(i, j, k,) to the nearest processors
      For ix ←0 to order-1 do
        For jx ←0 to order-1
       For kx←0 to order-1 do
                    If (i+ix)%order==0 && (j+jx)%order==0 && (k+kx)%order==0
                     Coarsening(i,j,k) //O(1)
                end_if
       end_for
         end_for
                end_for
       Step2) temp(i,j,k) ← Cal_GridCharge(D(i,j,k)) // QL(i, j, k)*theta.X*theta.Y*theta.Z
       Step3) QL+1(i, j, k) ← Q L+1 (i,j,k) + temp(i,j,k)
     end_for
end
```

Algorithm 3-2-2-1 describes *Coarsening( )* and Figure 28 provides the idea of the broadcasting with a 2-dimensional R-Mesh. Figure 28 (a) shows that first broadcasting step where i%2==0 && j%2==0. Then in the second step, the next group of nodes broadcast their data as shown in Figure 28 (b). These nodes have indices so that i%2==0 && j%2==1. Figure 28 (c)-(d) shows the similar processes. For example, Proc(0,0) represents other processors((0,1), (1,0), (1,1)). Thus these processors broadcast their information to the 15 processors that are the nearest processors to proc(0,0).

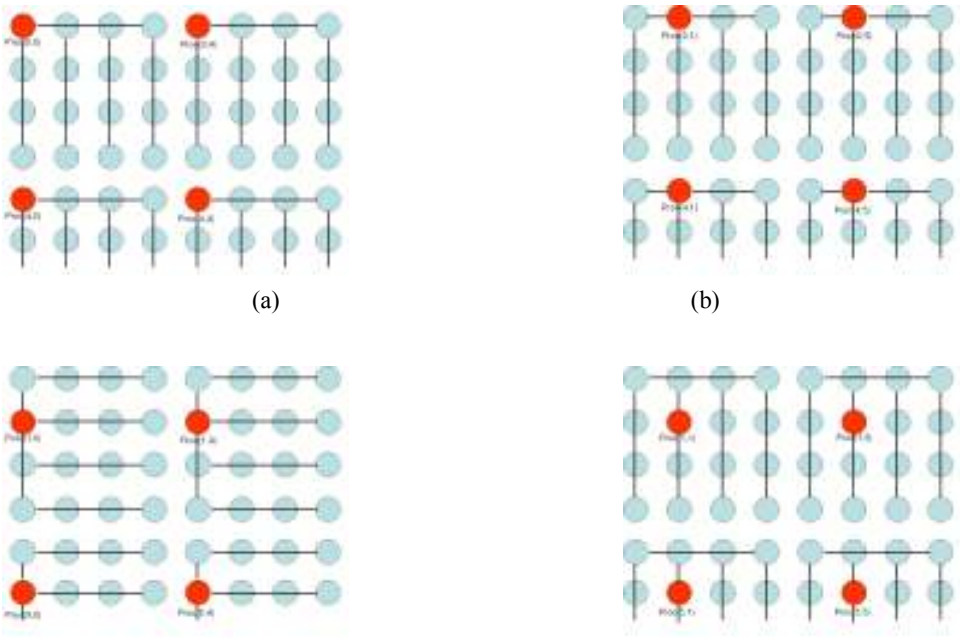
Algorithm 3-2-2-1 (Coarsening(i,j,k))

1. Model : M processors (X×Y×Z R-Mesh, M=X×Y×Z is # of grid point at level L)

2. Input: proc(i, j, k) hold D(i,j,k) =(Q_L(i, j, k), theta)

3. Output : proc(i, j, k) update D(i, j, k)

```
begin
Coarsening (i,j,k) // O(1)
        if i%2==0 && j%2==0 && k%2==0
    Broadcast data(i, j, k) to proc(i/2, j/2, k/2) ~ proc(i/2+order-1, j/2+order-1, k/2+order-1)
        end_if
        If i%2==0 && j%2==0 && k%2==1
    Broadcast data(i,j,k) to proc(i/2, j/2, k/2)~proc(i/2+order-1, j/2+order-1, (k-1)/2+order-1)
        end_if
        If i%2==0 && j%2==1 && k%2==0
    Broadcast data(i,j,k) to proc(i/2, j/2, k/2)~proc(i/2+order-1, (j-1)/2+order-1, k/2+order-1)
        end_if
        If i%2==0 && j%2==1 && k%2==1
    Broadcast data(i,j,k) to proc(i/2, j/2, k/2)~proc(i/2+order-1, (j-1)/2+order-1, (k-1)/2+order-1)
        end_if
        if i%2==1 && j%2==0 && k%2==0
    Broadcast data(i, j, k) to proc(i/2, j/2, k/2) ~ proc((i-1)/2+order-1, j/2+order-1, k/2+order-1)
        end_if
        If i%2==1 && j%2==0 && k%2==1
    Broadcast data(i,j,k) to proc(i/2, j/2, k/2)~proc((i-1)/2+order-1, j/2+order-1, (k-1)/2+order-1)
        end_if
        If i%2==1 && j%2==1 && k%2==0
    Broadcast data(i,j,k) to proc(i/2, j/2, k/2)~proc((i-1)/2+order-1, (j-1)/2+order-1, k/2+order-1)
        end_if
        If i%2==1 && j%2==1 && k%2==1
    Broadcast data(i,j,k) to proc(i/2, j/2, k/2)~proc((i-1)/2+order-1, (j-1)/2+order-1, (k-1)/2+order-1)
        end_if
end
```



(a) (b)

(c) (d)

Figure 28. Example of broadcasting in Algorithm 3-2-2 with order = 4

(a) proc(0,0), proc(0,4), proc(4,0) and proc(4,4) broadcast data simultaneously

(b) proc(0,1), proc(0,5), proc(4,1) and proc(4,5) broadcast data simultaneously and use same data bus with (a)

(c) proc(1,0), proc(1,4), proc(5,0) and proc(5,4) broadcast data simultaneously and use same data bus with (a)

(d) proc(1,1), proc(1,5), proc(5,1) and proc(5,5) broadcast data simultaneously and use same data bus with (1)

Algorithm 3-2-3 is *Correction(L)*. This module corrects grid charge(Q_L) after coarsening module (Algorithm 3-2-2) and update potential(V_L) at level *L*. The initial step generates the matrix *GCorrection(id, jd, kd)*. The number of loop iterations is $C \cdot M_L$, where $M_L = X \times Y \times Z$ which is the number of grid points at level L and *C* is constant number. As explained above, M_L is a much smaller number than *N*. Thus we could consider Correction(L) requires O(1) time complexity.

Algorithm 3-2-3 (Correction(L))

1. Model : M processors (X×Y×Z R-Mesh, M=X×Y×Z is # of grid point at level L)
2. Input: proc(i, j, k) hold data =(Q_L(i, j, k), theta) and L, GCorrDim and GCorrection
3. Output : proc(i, j, k) update V_L(i, j, k)

```
begin
Correction (L)
    Each proc(i, j, k)
        hi_l = min(i+GCorrDim.X, X);
        so_l = -GCorrDim.X+1 – min(i-GCorrDim.X+1, 0)
        lo_l = max(i-GCorrDim.X+1, 0)
        hi_m = min(j+GcorrDim.Y, Y);
        so_m = -GCorrDim.Y+1 – min(j-GCorrDim.Y+1, 0)
        lo_m = max(j-GCorrDim.Y+1, 0)
        hi_n = min(k+GCorrDim.Z, Z);
        so_n = -GCorrDim.Z+1 – min(k-GCorrDim.Z+1, 0)
        lo_n = max(k-GCorrDim.Z+1, 0)
        for l←lo_l, l2← so_l to l < hi_l do
            l0 ←l, id← abs(l2)
                for m←lo_m, m2← so_m to m < hi_m do
                m0 ←m, jd← abs(m2)
                        for n←lo_n, n2← so_n to n < hi_n do
                    n0 ←n, kd← abs(n2)
                    temp = temp + QL(l0, m0, n0) * GCorrection(id, jd, kd)
                end_for
            end_for
        end_for
        VL(i, j, k) = VL(i, j, k)+ temp*scale
end
```

Algorithm 3-2-4 describes *Step 3* in Algorithm 3-1. The module computes the potential for the top level. The module updates V_L, which is the potential of the top grid with grid charge and *GDirect* matrix. It generates the *GDirext* matrix in the initial step. The number of iterations for the loop in the module is $C \cdot M_t$, where $M_t = X_t \times Y_t \times Z_t$,which is the number of grid

points at the top level and *C* is a constant number. Since the size of grids at the top level is the biggest among the grids levels, M_t is very small and the module requires O(1) time complexity.

Algorithm 3-2-4 (Direct(L))

1. Model : N processors (X×Y×Z R-Mesh, N=X×Y×Z is # of grid point at top level)
2. Input: proc(i, j, k) hold data =(Q_L(i, j, k), V_L(i, j, k), GDriect(i, j, k))
3. Output : proc(i, j, k) update V_L(i, j, k)

```
begin
Direct (L)   // O( Nx(L)· Ny(L)· Nz(L))   i.e) Nx(i)·Ny(i)·Nz(i)   = grid points at i^th level
      Each proc(i, j, k)
         for l←i   to l < X do
                   i0←abs(i-l)
                   for m← j to m < Y do
                   j0← abs(j-m)
                            for n←k+1 to n < Z do
                        k0← abs(k-n)
                        temp = temp + Q_L(l, m, n) * GDirect(i0, j0, k0)
                        V_L(l, m, n) = V_L(l, m, n)+ Q_L(i, j, k)*GDirect(0,0,0)
                   end_for
                   n=0
            end_for
            m←0
         end_for
         V_L (i, j, k) = V_L (i, j, k) + Q_L(i, j, k)*GDirect(0,0,0) + temp
end
```

Algorithm 3-2-5 computes the energy of the top grid level and appears as step 5 in Algorithm 3-1. Each processor calculates *e(i,j,k)* by grid charge (Q_0)×potential (V_0) and the values are added to *E*. *E* is stored by processor(0,0,0). This module requires O(log*M*) time complexity, where M is $M=X\times Y\times Z$ is the number of finest grid.

Algorithm 3-2-5 (Energy)

1. Model : N processors (X×Y×Z R-Mesh, M=X×Y×Z is # of finest grid)
2. Input: proc(i, j, k) hold $Q_0(i, j, k)$ and $V_0(i, j, k)$
3. Output : proc(0, 0, 0) hold **E** (total energy)

```
begin
Energy( )    // O(logM)
        Each proc(i, j, k)
        e(i, j, k) = Q0(i, j, k) * V0(i, j, k)

            compute force(i) = temp(i, 0)+..+temp(i, M-1) with row bus i and proc(0, i) store force(i).
        add e(i, j, k) of processors with M processor into E of proc(0,0,0)          //O(logM)
end
```

Algorithm 3-2-6 interpolates grid charges from *level* to *level-1*. This algorithm updates $V_{level-1}$, which is the potential at *level-1* with *thetas*. The preprocessing module (Algorithm 3-1) generates *thetas* and the X, Y and Z field in *thetas* are an array with size *order*. *Ratio* and *order* are constant numbers and *Ratio* represents the ratio between Level *L* and *L-1*. This algorithm requires O(1) time complexity.

Algorithm 3-2-6 (CoarseToFine(L))

1. Model : M processors (X×Y×Z R-Mesh, M=X×Y×Z is # of grid point at L)
2. Input: proc(i, j, k) hold $V_{level}(i, j, k)$ and theta ={X, Y, Z}
3. Output : proc(i, j, k) update $V_{level-1}(i, j, k)$

```
begin
CoarseToFine(L)    // O(1)
        Each proc(i, j, k)
        Step 1) calculate temp using coarsened V_level and thetas
                    i1←i/Ratio, j1←j/Ratio, k1←k/Ratio
                 for i0←0 to order-1 do
                i2← i1+i0
                for j0←0 to order-1 do
                        j2← j1+j0
                        for k0←0 to order-1 do
                                k2← k1+k0
                                temp=temp+ V_level(i2,j2,k2)*theta.X[i0]*theta.Y[j0]* theta.Z[k0]
                        end_for
                end_for
        end_for
        Step2) V_level-1(i, j, k) = V_level-1(i, j, k) + temp
end
```

Algorithm 3-2-7 performs interpolating forces from grid level 0 for each atom. Its inputs are provided by *preprocessing( )* (Refer to Algorithm 3-1). *dTheta* is Δtheta and it has the X, Y and Z arrays like *theta* in Algorithm 3-2-7. Each processor stores the forces of the finest grid point (i, j, k). This algorithm requires O(*r*) time complexity, *r* is *N/M* and *M*= *X*×*Y*×*Z* is the number of finest grid points applied to Multigrid method at a given parameter.

Algorithm 3-2-7 (InterpolateForce)

1. Model : M processors (X×Y×Z R-Mesh, M=X×Y×Z is # of finest grid point)
2. Input: proc(i, j, k) hold V_0(i, j, k) Data = {d_0, d_1.. d_r}, which d = (index, p, q, theta, dtTheta, grid_pos) and *r* is number of atoms assigned in proc(i, j, k)
3. Output : proc(i, j, k) store force(i,j,k)={f_0, f_1.. f_r}

```
begin
InterpolateForce( )   // O(r)
    Each proc(i, j, k)
    For m←0 to r
        Step 1) calculate temp to update force(i,j,k)
            For i0← 0 to order do
                For j0← 0 to order do
                For k0← 0 to order do
                    Term = V0(i0+i, 0+j,k0+k)
                    fx = fx + Term* dTheta.X[i]*Theta.Y[j]*Theta.Z[k]
                    fy = fy + Term*Theta.X[i]*dTheta.Y[j]*Theta.Z[k]
                    fz = fz + Term*Theta.X[i]*Theta.Y[j]*dTheta.Z[k]
                end_for
            end_for
            end_for

        Step 2) q←Data(m).q
        Step 3) force(i.j.k).fm += Vector3D(fx*q*HXr, fy*q*HYr, fz*q*HZr);
      end_for
end
```

7.3. Results and Analysis

As explained Section 3.2.1, Cray XT3 and Blue Gene/L are only able to scale to up to a few thousands nodes due to the communication overheads [2, 7]. With this limitation, it is not possible to provide accommodating computing speed for biology activity with current

computing power. The communication overhead limits the performance and scaling on microprocessors and massively-parallel systems [45].

We support the feasibility of reconfigurable models by providing theoretical theorems with R-Mesh algorithm for MD simulation. Our results for the two versions of Direct method require O(N) time complexity with an N linear R-Mesh and O(log$N$) time complexity with an $N \times N$ 2-demensional R-Mesh. We are able to improve upon the results for the Multigrid method. While we also are able to achieve O(r)+O(log$M$) time complexity, the number of processors required are much less. The R-Mesh algorithm requires $M=X \times Y \times Z$ processors corresponding to the number of finest grid points, rather than N processors corresponding to the number of atoms in the system. For most systems M is much smaller than N, thus reducing the size of the simulating machine. This improvement is due to the natural mapping of the layout of the MD system in a grid pattern to the three-dimensional structure of the R-Mesh.

Theorem 1 *Molecular Dynamics simulation of a molecular system with N atoms can be performed in $K \cdot O(N)$ time on an N processor linear R-Mesh, when the simulation exploits the Direct method to evaluate electrostatic potential. K is the number of iterations to reach equilibrium. (Algorithm 1)*

Theorem 2 *Molecular Dynamics simulation of a molecular system with N atoms can be performed in K·O(logN) time on an N×N 2-dimensional R-Mesh, when the simulation exploits the Direct method to evaluate electrostatic potential. K is the number of iterations to reach equilibrium. (Algorithm 2)*

Theorem 3 *Molecular Dynamics simulation of a molecular system with N atoms can be performed in K·(O(r)+O(logM)) time on an X×Y×Z 3-dimensional R-Mesh, when the simulation exploits the Multigrid method to evaluate electrostatic potential. X, Y and Z are the number of finest grid points applied to Multigrid method at given parameter. K is the number of iterations to reach equilibrium. (Algorithm 3)*

Chapter 8
Future work

In this section, we suggest three possible directions for future work. The first future work is to design Pipelined Reconfigurable Mesh (PR-Mesh) Algorithms for the MD simulation. Second direction performs MD simulations for non-structured molecular system. Third direction is to improve our proposed FPGA-based simulator in parallel manner.

8.1. PR-Mesh Algorithm for MD Simulation

Many researchers have proposed several reconfigurable models employing optical buses. The optical signal transmission possesses two advantageous properties: unidirectional propagation and predictable propagation delay per unit length. These two properties allow synchronized concurrent access to an optical bus, creating a pipeline of message. Therefore, the models based on optical bus can be very efficient for parallel computation due to the high bandwidth that comes with pipelining messages [1].

The Pipelined Reconfigurable Mesh (PR-Mesh) is one of the optical reconfigurable models studies in the literature. It is a k-dimensional mesh of processors in which each processor has $2k$ ports [28]. A two-dimensional PR-Mesh is an $R \times C$ mesh of processors in which each processor has four ports. The ports connect to eight segments of buses using directional

couplers [27] as shown in Figure 29. We will extend our work with 2-dimensional PR-mesh to parallelize the proposed system. We expect it will increase the performance of proposed system highly.

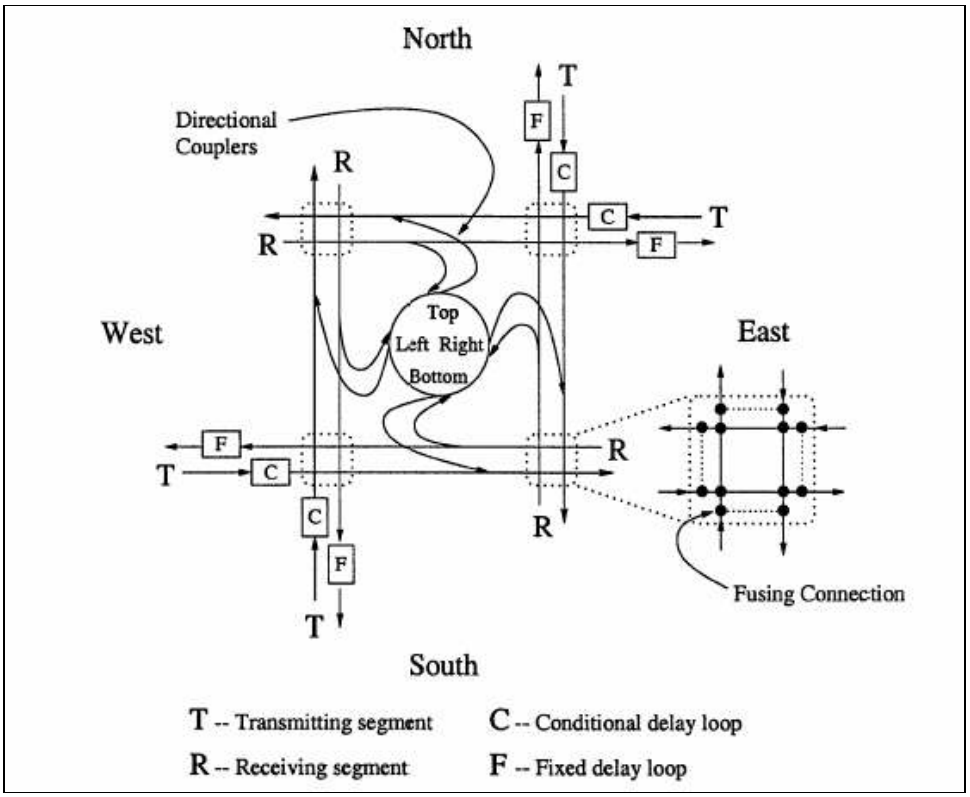


Figure 29. PR-mesh processor connections [27]

8.2. Molecular Dynamics Simulation for Non-structured Molecular System

We would continue our research to another direction that performs MD simulations for non-structured molecular system. Current methods for the MD simulation focus on a structured molecular system. However, many molecular systems are non-structured. If we consider non-structured feature to perform MD simulations, it will achieve more accuracy corresponding to

the actual structure. In this study, we will grid the system by computing *Convex Hull* and *Triangulation* algorithms and apply Multigrid method. Many graph algorithms such as *Convex Hull* and triangulation algorithm, will be exploited to evaluate MD for non-structured molecular system and are computational intensive. Sung-Ryul Kim et al. [46] proposed an O(log*N* loglog*N*) time R-Mesh algorithm for the Simple Polygon Visibility Problem. *Simple Polygon Visibility Problem* is given a simple polygon *P* with *N* vertices and a point z in the interior of the polygon finds all the boundary points of *P* that are visible from *z*. We could design R-mesh algorithms to grid unstructured molecular systems and perform MD simulations on those molecular systems with reasonable calculation time.

8.3. Parallelizing our proposed FPGA-based MD simulator

We proposed FPGA-based MD simulator that is customized for the Multigrid method. We can improve the simulator by parallelizing the time consuming tasks. The Molecular Dynamics (MD) simulation can suffer from an imbalance in load characteristic that varies at run time[47]. Due to the dynamics of particle interaction and motion during MD simulation, the task of load balancing is a non-trivial task. Potential load imbalances can significantly impart an accelerator architecture's resource utilization efficiency, especially when considering implementations based on custom architectures. Phillips *et a.l* [47] proposed architecture supporting dynamic load balancing on an FPGA for a MD algorithm.

Chapter 9
Conclusion

In the field of biology, Molecular Dynamics (MD) simulations are used continuously to study biological activities. Since the MD simulation is a large scale problem and multiscale in length and time, many approaches have been proposed to meet the speed required. In this dissertation, we have proposed two research directions. For the MD simulation we develop an efficient algorithm, Multi-level Charge Assignment (MCA) method [10] that achieves faster and accurate simulations and we also utilize Reconfigurable models to perform the MD simulation. Our MCA method is an O(N) Multigrid (MG) method for accurate and efficient calculation of the electrostatic forces. The MCA method gives consistent accuracy and reduces errors even if the distribution of particle is not balanced. We demonstrate Multigrid charge assignment scheme and back interpolation scheme which adjusts the grid charge on LDM. Using the MCA method, the MD simulation is more accurate while still requiring similar calculation time to current methods.

We support the idea that exploits Reconfigurable models to perform large scale problems such as the MD simulation. The first reconfigurable model we utilized for the MD simulation is the FPGA. We design the architecture of an FPGA-based MD simulator and the simulator employs the MCA method. The simulator is especially suitable for large scale molecules that require a considerable amount of calculation time using a software solution [18]. Using

FPGAs, we achieve speed-up the simulation with a factor 10 to 100 compared to a software implementation on Protomol without loss of accuracy [19]. We also expect more speed up if we parallelize the modules of the proposed simulator, but this would require more space and cost.

The second reconfigurable model we utilized for the MD simulation is a Reconfigurable Mesh (R-mesh). We develop R-Mesh algorithms for two MD simulation methods, Direct method and MG method. Direct method requires $O(N^2)$ time complexity for evaluating electrostatic forces and provides accurate results if executed sequentially. We develop two versions of R-Mesh algorithms that implement the Direct method. Our first version requires $O(N)$ time complexity with an N processor linear R-Mesh (Theorem 1) and the second version requires $O(\log N)$ with an $N \times N$ 2-dimensional R-Mesh (Theorem 2). We also develop an R-Mesh algorithm that implements the MG method to evaluate electrostatic forces. The MG method requires $O(N)$ calculation time at a given accuracy for a sequential implementation. However, our R-Mesh algorithm requires $O(r)+O(\log M)$ time complexity with an $X \times Y \times Z$ 3-dimensional R-Mesh (Theorem 3). This algorithm requires M processors, where M is the number of finest grid points ($M = X \times Y \times Z$). Since M is usually a much smaller number compared to N, this algorithm provides very fast simulation time with a small number of processors. In conclusion, Reconfigurable Models provide not only an efficient but also a scalable method for MD simulation. Our R-Mesh algorithm implementing the MG

method with O(r)+O(log$M$) time complexity demonstrates that the R-Mesh is a feasible choice for developing the MG method for MD simulations and likely other large scale biological problems.

As future work, we will design algorithms to utilize other reconfigurable model, Pipelined Reconfigurable Mesh (PR-Mesh) to run the MD simulation. This will simulate our proposed method widely used in reconfigurable computing. In addition, we are studying another direction that considers MD simulations for non-structured molecular system. By considering non-structured molecular system, we can expect more accurate simulation results.

Publications

1. Cho, E., A.G. Bourgeois, and J.A. Fernández-Zepeda, *Examining the feasibility of Reconfigurable Models for Molecular Dynamics Simulation*, in *International Conference on Algorithms and Architectures*. 2008: Cyprus.

2. Cho, E. and Anu G. Bourgeois, *Efficient and accurate FPGA-based simulator for Molecular Dynamics*, Reconfigurable Architectures Workshop(RAW), 2008

3. Cho, E. and A. G. Bourgeois, *Examining the Feasibility of Reconfigurable Models for Molecular Dynamics Simulation,* Parallel Computing, 2008-under review

4. Cho, E., A.G. Bourgeois, and F. Tan, *An FPGA Design to Achieve Fast and Accurate Results for Molecular Dynamics Simulations.* LECTURE NOTES IN COMPUTER SCIENCE, 2007. **4742**: p. 256.

5. Cho, E. and Anu G. Bourgeois, *Efficient Molecular Dynamics (MD) simulation on Field Programmable Gate Array (FPGA)s with MultiGrid method*, International Conference on Research in Computational Molecular Biology(RECOMB), 2007

6. Cho, E. and A. G. Bourgeois, *Efficient Molecular Dynamics Simulation on Reconfigurable Models with MultiGrid Method*, IEEE transactions on NanoBioScience, 2007

7. Cho, E. and A.G. Bourgeois, *Multi-level charge assignment for accurate and efficient Molecular Dynamics(MD) simulation*. in Proceeding of 13th AI, Simulation and Planning in High Autonomy Systems (AIS), 2006.

8. Cho, E. and K. Lee, *Security Policy Management Using Role-Based Access Control in CORBA*, Information Science Society, 1998, Vol 25.

9. Shin, H. and E. Cho, *et al*. *Search System for Document of Buddha on Internet*, Korea Information Processing Society, 1998, Vol 25.

10. XML for beginners, 2001, Kanam Publisher, Korea

Bibliography

1. Vaidyanathan, R. and J.L. Trahan, *Dynamic Reconfiguration: Architectures and Algorithms*. 2003: Plenum Pub Corp.
2. Alam, S.R., J.S. Vetter, and P.K. Agarwal, *Performance characterization of molecular dynamics techniques for biomolecular simulations.* Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming, 2006: p. 59-68.
3. Rapaport, D.C., *The Art of Molecular Dynamics Simulation*. 2004: Cambridge University Press.
4. Sagui, C. and T.A. Darden, *MOLECULAR DYNAMICS SIMULATIONS OF BIOMOLECULES: Long-Range Electrostatic Effects.* Annual Review of Biophysics and Biomolecular Structure, 1999. **28**(1): p. 155-179.
5. Sagui, C. and T. Darden, *Multigrid methods for classical molecular dynamics simulations of biomolecules.* The Journal of Chemical Physics, 2001. **114**: p. 6578.
6. Izaguirre, J.A., S.S. Hampton, and T. Matthey, *Parallel multigrid summation for the N-body problem.* Journal of Parallel and Distributed Computing, 2005. **65**(8): p. 949-962.
7. Alam, S.R. and P.K. Agarwal, *On the Path to Enable Multi-scale Biomolecular Simulations on PetaFLOPS Supercomputer with Multi-core Processors.* Sixth IEEE International Workshop on High Performance Computational Biology (HiCOMB), 2007.
8. Toukmaji, A.Y. and J.A. Board, *Ewald summation techniques in perspective: a survey.* Computer Physics Communications, 1996. **95**(2-3): p. 73-92.
9. Skeel, R.D., I. Tezcan, and D.J. Hardy, *Multiple grid methods for classical molecular dynamics.* Journal of Computational Chemistry, 2002. **23**(6): p. 673-684.
10. Cho, E. and A.G. Bourgeois. *Multi-level charge assignment for accurate and efficient Molecular Dynamics(MD) simulation*. in *International Modeling and Simulation Multiconference*. 2007. Buenos Aires, Argentina.
11. Agarwal, P.K. and S.R. Alam, *Biomolecular simulations on petascale: promises and challenges.* Journal of Physics: Conference Series, 2006. **46**(1): p. 327-333.
12. Komeiji, Y., et al., *A high performance system for molecular dynamics simulation of biomolecules using a special-purpose computer.* Pac Symp Biocomput, 1996. **472**: p. 87.
13. Komeiji, Y., et al., *Fast and accurate molecular dynamics simulation of a protein using a special-purpose computer.* Journal of Computational Chemistry, 1997. **18**(12):

p. 1546-1563.

14. Toyoda, S., et al., *Development of MD Engine: High-speed accelerator with parallel processor design for molecular dynamics simulations.* Journal of Computational Chemistry, 1999. **20**(2): p. 185-199.
15. Azizi, N., et al., *Reconfigurable molecular dynamics simulator.* Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on, 2004: p. 197-206.
16. *The Transmogrifier-3a Rapid Prototyping System*. [cited; Available from: http://www.eecg.utoronto.ca/~tm3/.
17. Gu, Y., T. VanCourt, and M.C. Herbordt, *Accelerating molecular dynamics simulations with configurable circuits.* Computers and Digital Techniques, IEE Proceedings-, 2006. **153**(3): p. 189-195.
18. Cho, E., A.G. Bourgeois, and F. Tan, *An FPGA Design to Achieve Fast and Accurate Results for Molecular Dynamics Simulations.* LECTURE NOTES IN COMPUTER SCIENCE, 2007. **4742**: p. 256.
19. Cho, E., A.G. Bourgeois, and J.A. Fernández-Zepeda, *Efficient and accurate FPGA-based simulator for Molecular Dynamics*, in *15th Reconfigurable Architectures Workshop (RAW)*. 2008: Maimi.
20. Cho, E., A.G. Bourgeois, and J.A. Fernández-Zepeda, *Examining the feasibility of Reconfigurable Models for Molecular Dynamics Simulation*, in *International Conference on Algorithms and Architectures*. 2008: Cyprus.
21. Nakano, K., *A Bibliography of Published Papers on Dynamically Reconfigurable Architectures.* Parallel Processing Letters, 1995. **5**(1): p. 111-124.
22. Li, H., et al., *Reconfigurable SIMD massively parallel computers.* Proceedings of the IEEE, 1991. **79**(4): p. 429-443.
23. Miller, R., et al., *Parallel computations on reconfigurable meshes.* Computers, IEEE Transactions on, 1993. **42**(6): p. 678-692.
24. Fernandez-Zepeda, J.A., R. Vaidyanathan, and J.L. Trahan, *Scaling simulation of the fusing-restricted reconfigurable mesh.* Parallel and Distributed Systems, IEEE Transactions on, 1998. **9**(9): p. 861-871.
25. Ben-Asher, Y., et al., *The Power of Reconfiguration.* Journal of Parallel and Distributed Computing, 1991. **13**(2): p. 139-153.
26. Bourgeois, A.G. and J.L. Trahan때, *Fault tolerant algorithms for a linear array with a reconfigurable pipelined bus system.* Parallel Algorithms and Applications, 2003. **18**(3): p. 139-153.
27. Bourgeois, A.G. and J.L. Trahan, *Relating two-dimensional reconfigurable meshes with opticallypipelined buses.* Parallel and Distributed Processing Symposium, 2000.

IPDPS 2000. Proceedings. 14th International, 2000: p. 747-752.

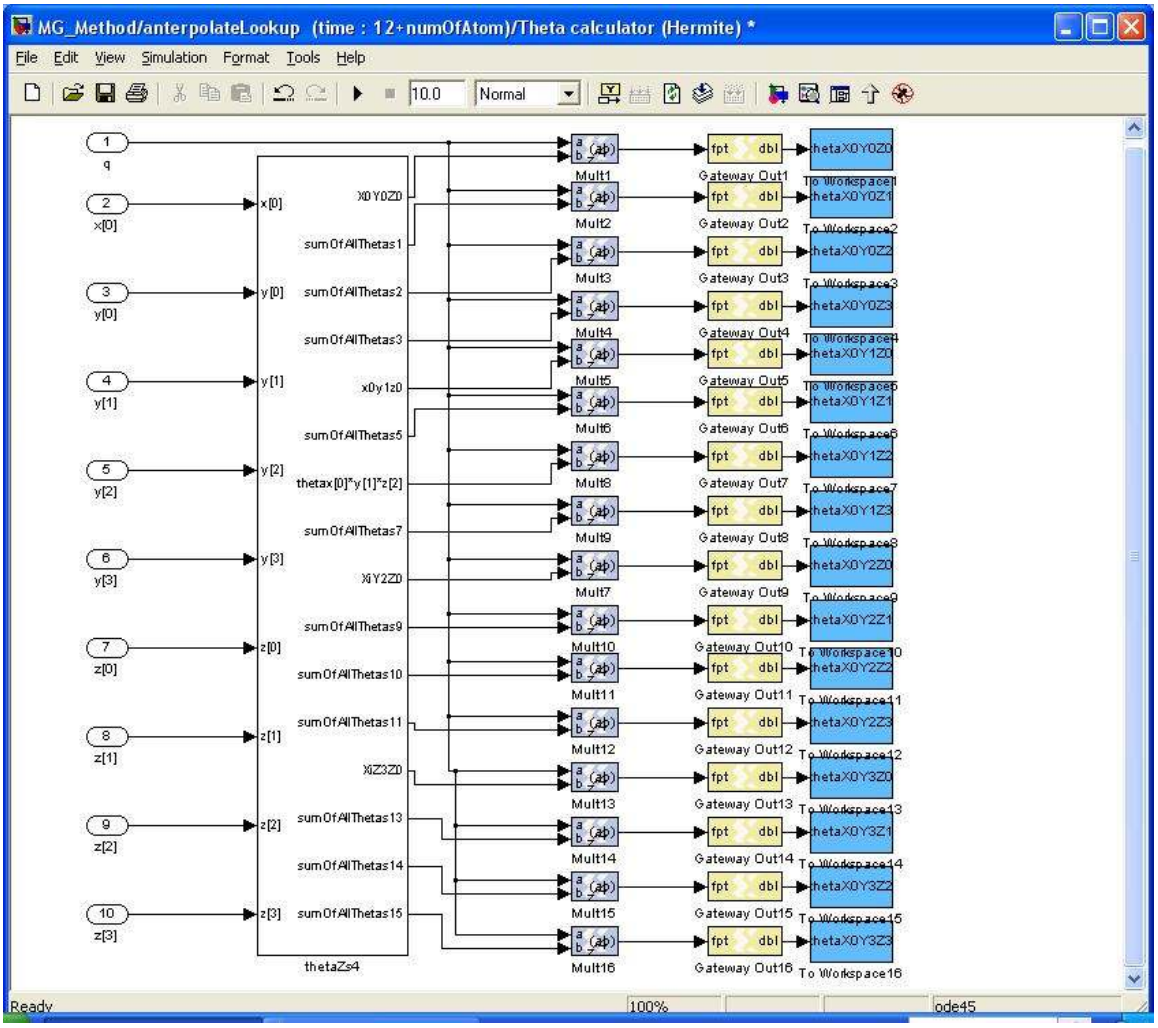
28. Trahan, J.L., A.G. Bourgeois, and R. Vaidyanathan, *Tighter and Broader Complexity Results for Reconfigurable Models.* Parallel Processing Letters, 1998. **8**(3): p. 271-282.
29. Miller, R., et al., *Meshes with reconfigurable buses.* Proceedings of the fifth MIT conference on Advanced research in VLSI table of contents, 1988: p. 163-178.
30. Jenq, J.F. and S. Sahni, *Reconfigurable mesh algorithms for image shrinking, expanding, clustering, and template matching.* Parallel Processing Symposium, 1991. Proceedings., Fifth International: p. 208-215.
31. Sato, N. and J.M. Jezequel, *Implementing and Evaluating an Efficient Dynamic Load-Balancer for Distributed Molecular Dynamics Simulation.* Proceedings ofthe 2000 International Workshops on Parallel Processing.
32. Xilink. *DSP Design Flows in FPGA*. 2003 [cited.
33. Tanurhan, Y., *DSP Design Flows in FPGAs*. 2006.
34. Goslin, G.R., *A Guide to Using Field Programmable Gate Arrays (FPGAs) for Application-Specific Digital Signal Processing Performance.* Xilinx Inc, 1995.
35. Rankin, W.T. and J.A. Board Jr, *A portable distributed implementation of the parallel multipoletree algorithm.* High Performance Distributed Computing, 1995., Proceedings of the Fourth IEEE International Symposium on, 1995: p. 17-22.
36. Beckers, J.V.L., C.P. Lowe, and S.W. De Leeuw, *An Iterative PPPM Method for Simulating Coulombic Systems on Distributed Memory Parallel Computers.* Molecular Simulation, 1998. **20**(6): p. 369-383.
37. York, D. and W. Yang, *The fast Fourier Poisson method for calculating Ewald sums.* The Journal of Chemical Physics, 1994. **101**: p. 3298.
38. Briggs, E.L., D.J. Sullivan, and J. Bernholc, *Real-space multigrid-based approach to large-scale electronic structure calculations.* Physical Review B, 1996. **54**(20): p. 14362-14375.
39. Banerjee, S. and J.A. Board Jr, *Multigrid Methods for Molecular Dynamics.* J Comput Chem, 2005. **26**: p. 957-967.
40. Fitch, B.G., et al., *Blue Matter, an application framework for molecular simulation on Blue Gene.* Journal of Parallel and Distributed Computing, 2003. **63**(7-8): p. 759-773.
41. Matthey, T., et al., *ProtoMol, an object-oriented framework for prototyping novel algorithms for molecular dynamics.* ACM Transactions on Mathematical Software, 2004. **30**(3): p. 237-265.
42. Amisaki, T., et al., *Error evaluation in the design of a special-purpose processor that calculates nonbonded forces in molecular dynamics simulations.* Journal of Computational Chemistry, 1995. **16**(9): p. 1120-1130.
43. Allen, M.P. and D. J, *Computer Simulation of Liquids*. 1989: Oxford University Press.
44. Spiegel, J.V.d.S. *VHDL Tutorial*. 2006 [cited; University of Pennsylvania,

Department of Electrical and Systems Engineering]. Available from: http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html#_Toc526061341.

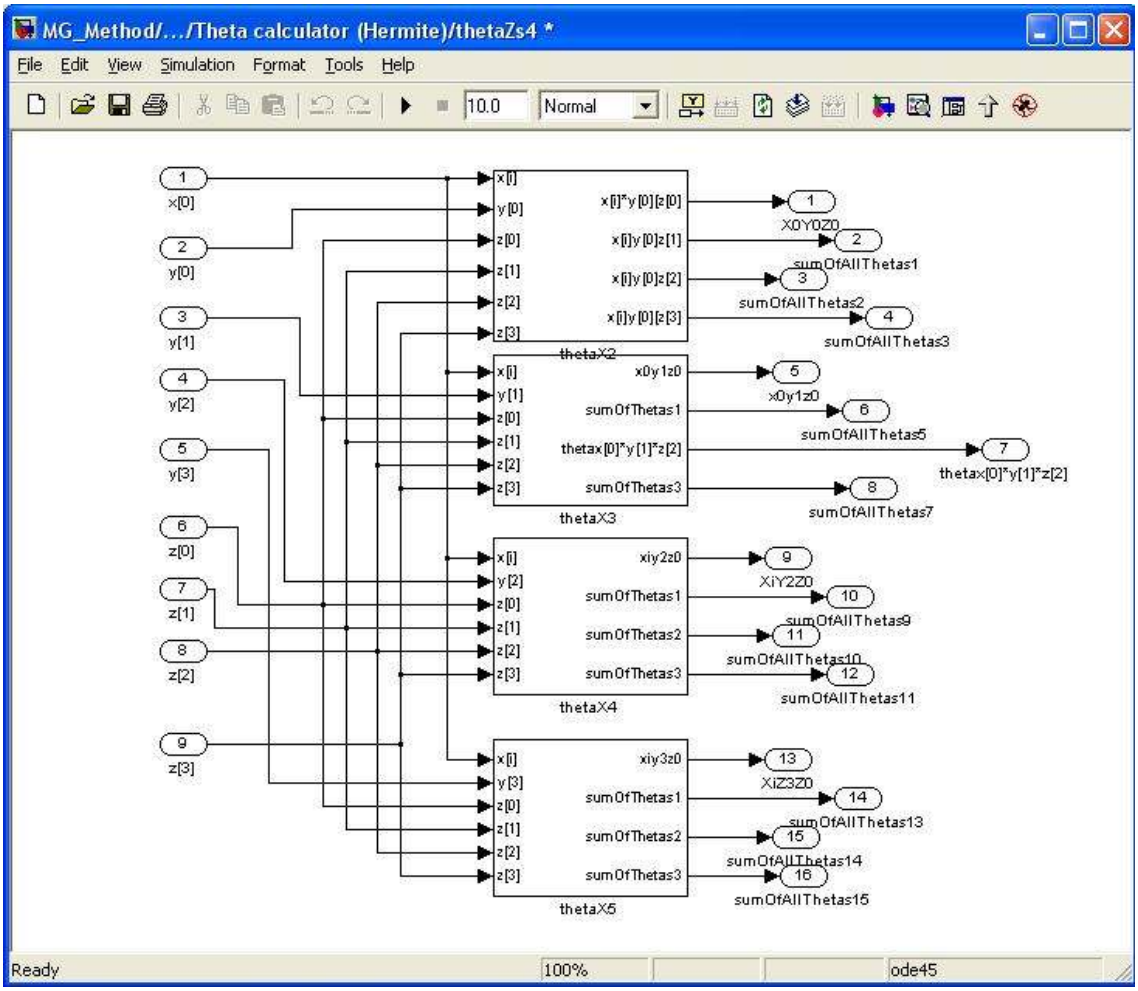
45. Crowley, M., et al., *Adventures in Improving the Scaling and Accuracy of a Parallel Molecular Dynamics Program.* The Journal of Supercomputing, 1997. **11**(3): p. 255-278.
46. Kim, S.R., K. Park, and Y.K. Cho, *An O (log N log log N) time RMESH algorithm for the simple polygonvisibility problem.* Parallel Architectures, Algorithms and Networks, 1994.(ISPAN) International Symposium on, 1994: p. 151-158.
47. Phillips, J., et al., *A Reconfigurable Load Balancing Architecture for Molecular Dynamics.* Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, 2007: p. 1-6.

Appendix

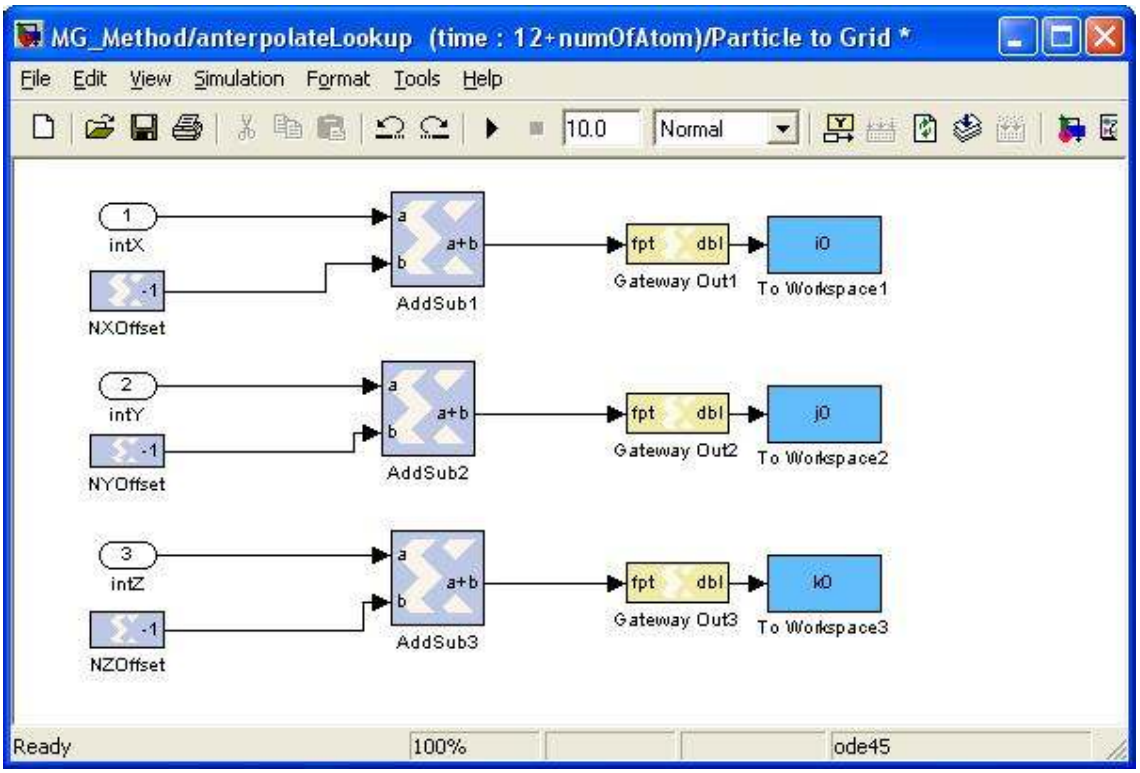
Modules of our proposed FPGA-based Simulator



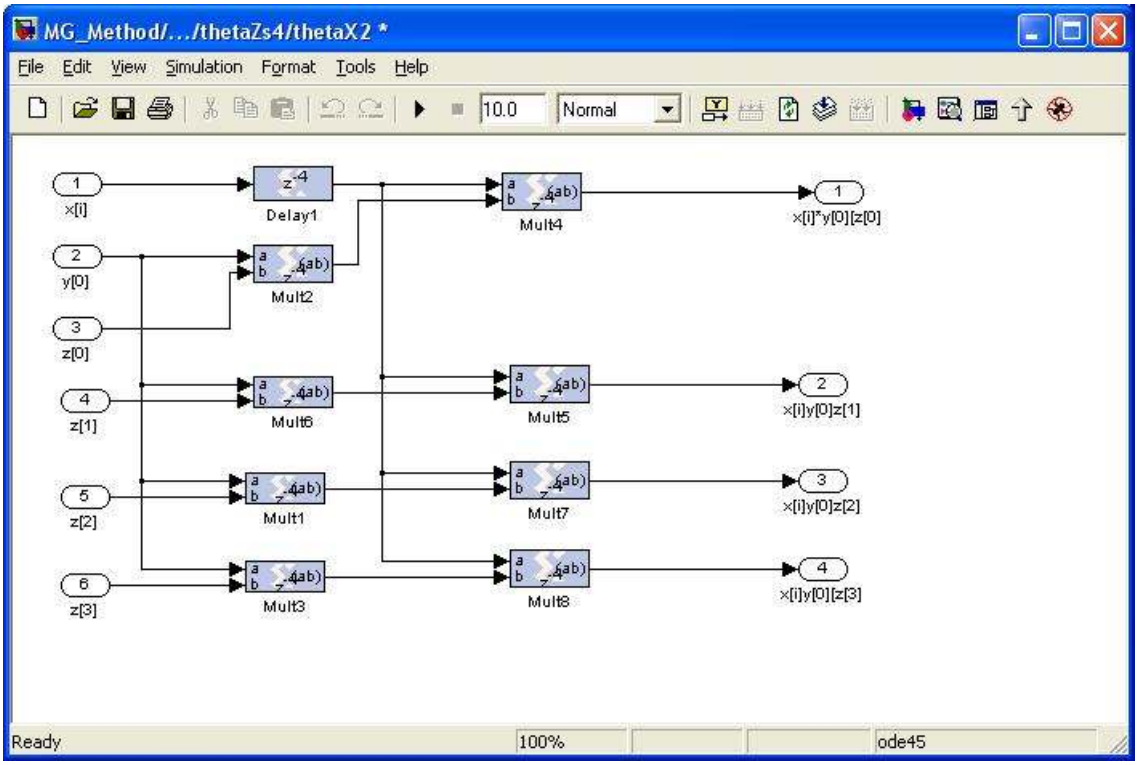
ThetaCalculator block in *anterpolate* module in figure 16



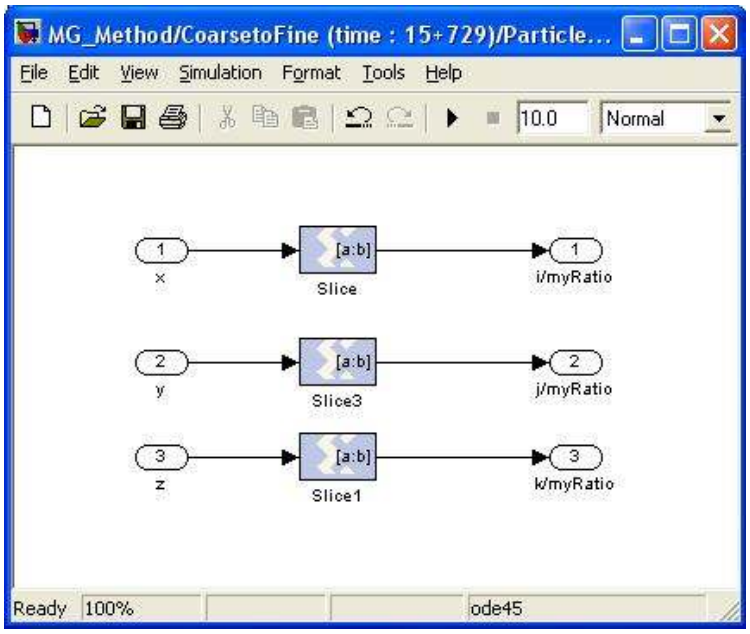
thetaZs4 block in *anterpolate* module in figure 20



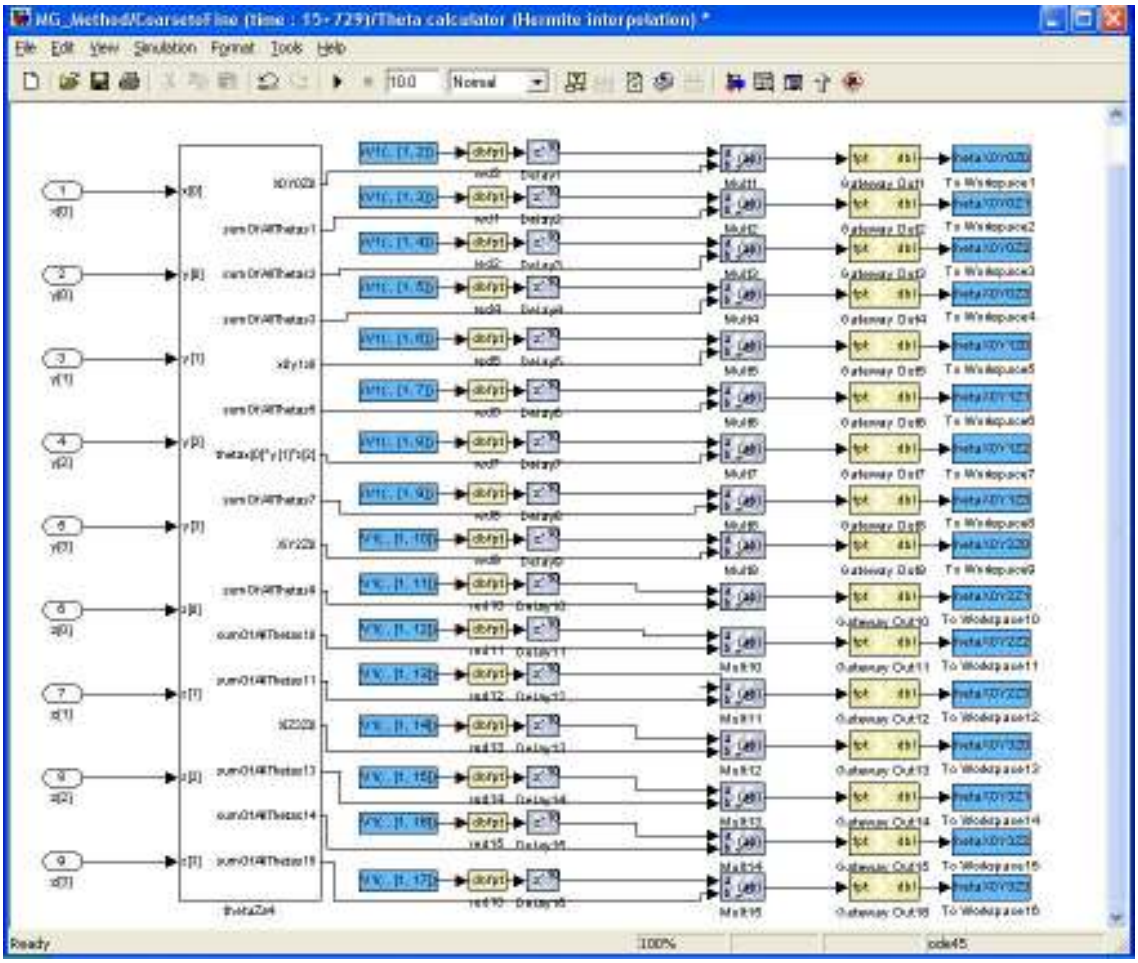
Particle to Grid block in *anterpolate* module in figure 16



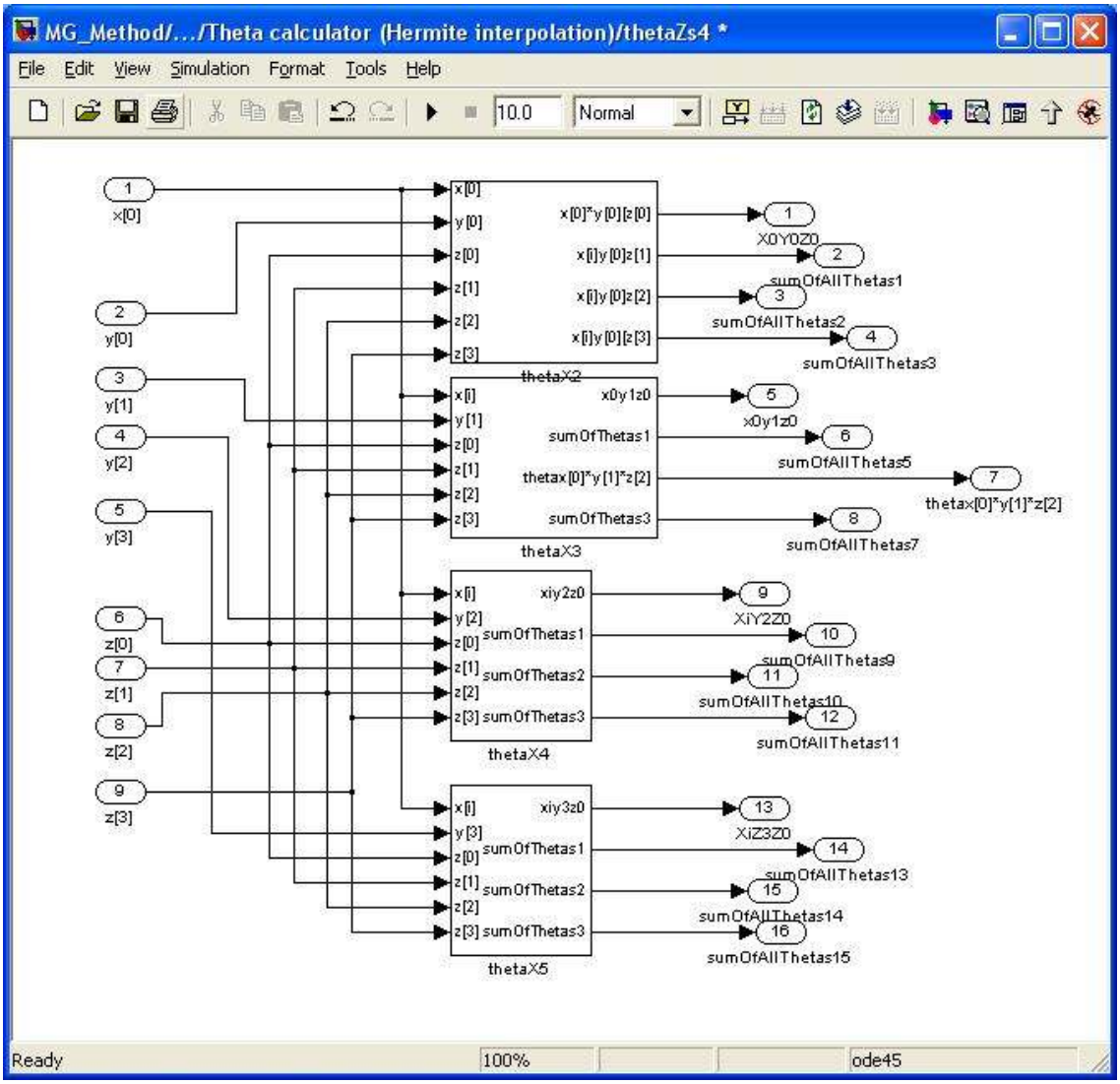
thetaX2 block in *anterpolate* module in figure 21



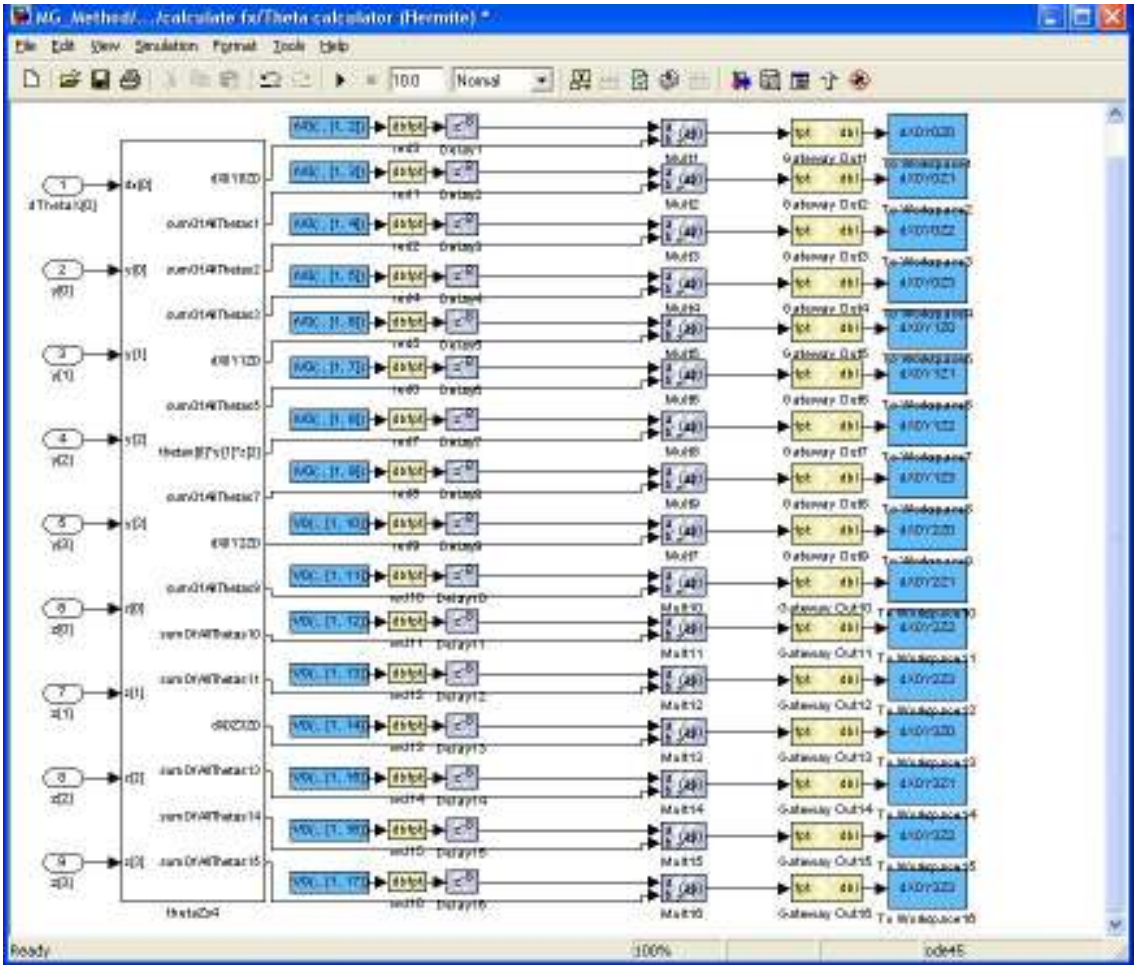
Particle to Grid block in *coarseToFine* module



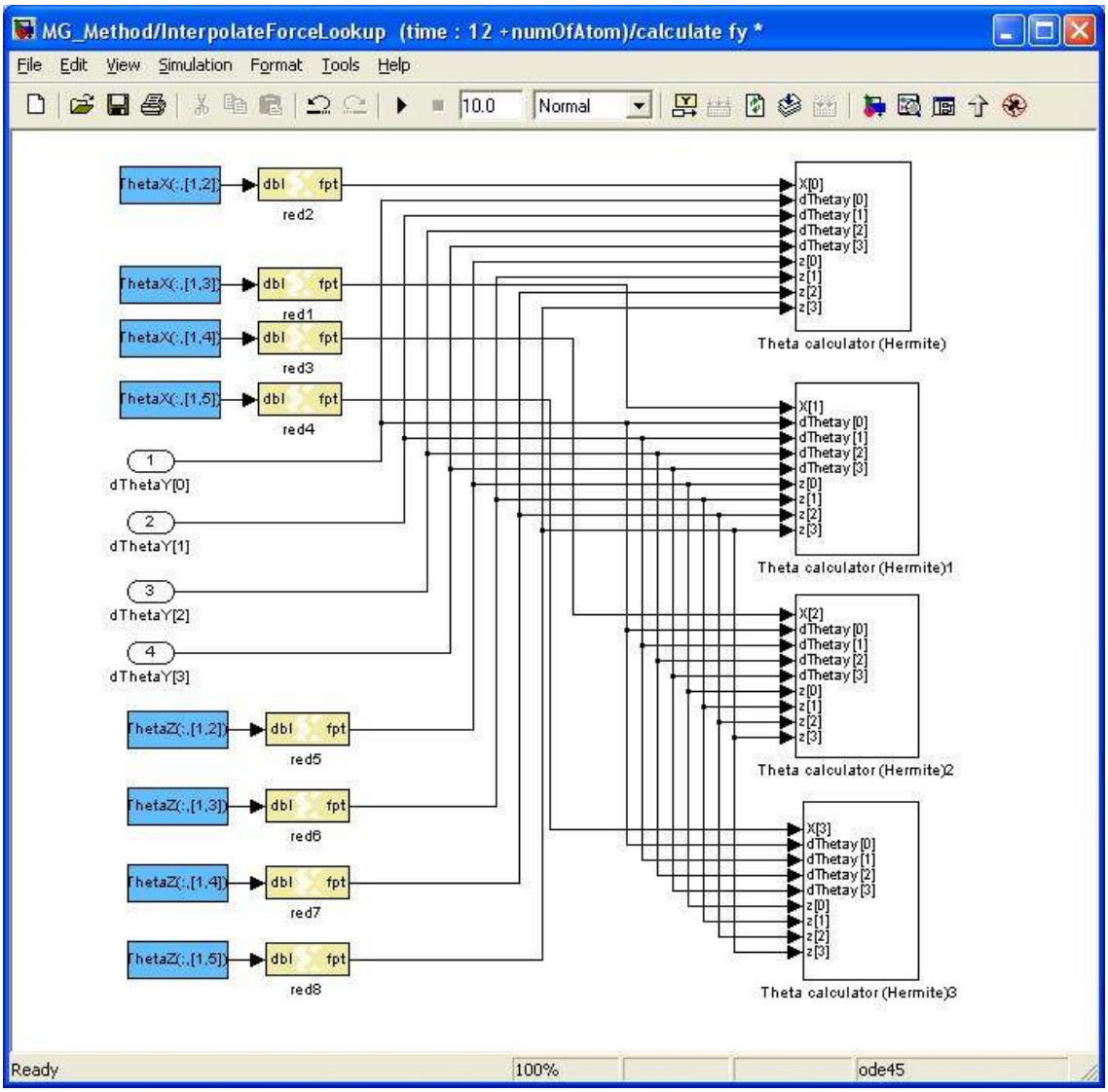
Theta Calculator block in *coarseToFine* module



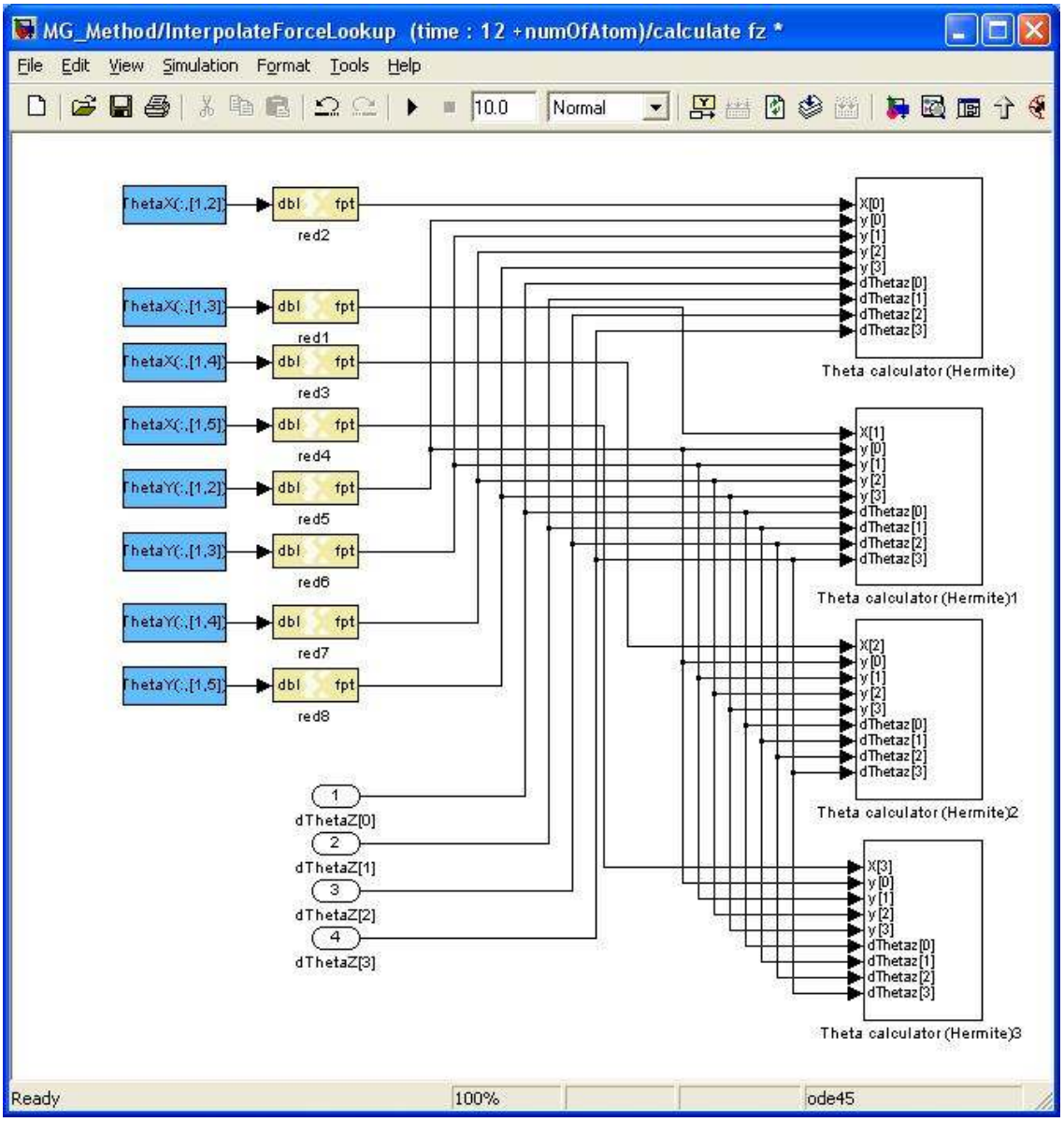
thetaZs4 block in *Theta Calculator* block



Theta Calculator block in *calculateFx* block of *InterpolateForce* module



calculateFy block in *InterpolateForce* module



calculateFz block in *InterpolateForce* module