# Energy Minimization Accelerated with FPGAs

*Huaxin Dai*

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

**BOSTON**

**UNIVERSITY**

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Thesis

# Energy Minimization Accelerated with FPGAs

by

**Huaxin Dai**

B.S., Beijing Jiaotong University, 2008

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science

2010

Approved by

First Reader

Martin Herbordt, PhD
Associate Professor of Electrical and Computer Engineering

Second Reader

Ajay Joshi, PhD
Assistant Professor of Electrical and Computer Engineering

Third Reader

Ayse Coskun, PhD
Assistant Professor of Electrical and Computer Engineering

# Energy Minimization Accelerated with FPGAs

**Huaxin Dai**

ABSTRACT

Energy minimization is an important step in many molecular modeling applications like molecular docking and mapping binding sites. It involves repeated iterations a of evaluation of various bonded and non-bonded energies of a protein complex. Due to the extensive computation in each iteration, energy minimization is a computationally expensive process, with runtimes typically being many hours on a desktop system. We find that FPGAs are suited to accelerating energy minimization. The extensive computation time is overcome with a long processing pipeline An important contribution is the change in data structure keeps the pipeline busy. The primary result is a microarchitecture for electrostatic energy evaluation that processes the majority part of energy evaluation functions, resulting in a substantial speed-up over single core CPU implementation for that part of the computation.

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | | |
|---|---|---|
| ACE | ............. | Analytic Continuum Electrostatics |
| ASIC | ............. | Application Specific Integrated Circuit |
| CPU | ............. | Central Processing Unit |
| CHARMM | ............. | Chemistry at HARvard Macromolecular Mechanics |
| DDR | ............. | Double Data Rate |
| DFG | ............. | Data Flow Graph |
| DSP | ............. | Digital Signal Processing |
| FIFO | ............. | First In First Out |
| FPC | ............. | Floating Point Compiler |
| FPGA | ............. | Field Programmable Gate Array |
| GPU | ............. | Graphical Processing Unit |
| HPC | ............. | High-Performance Computing |
| L-BFGS | ............. | Limited memory Broyden-Fletcher-Goldfarb-Shanno method |
| MD | ............. | Molecular Dynamics |
| PCI | ............. | Peripheral Component Interconnect |
| RAM | ............. | Random Access Memory |

# Chapter 1

# Introduction

## 1.1 What is Energy Minimization and What is It Used For?

Energy minimization is an iterative process which aims at computing the minimum potential energy that can be achieved given the configuration of the atoms in a complex. In order to evaluate the potential energy of the system, the complex is often represented using force fields. A force field represents each atom in a molecular system as a point of charge, and the total potential energy of the system is the sum of various particle-particle interactions. Different kinds of force fields have been developed, with the more popular ones being CHARMM [4] and AMBER [5]. Due to the popularity of CHARMM force-field, energy minimization is often referred to as minimization of the CHARMM potential or simply as *CHARMM minimization*.

Usually, one iteration of minimization involves three steps: computing the potential energy of the complex at a point, updating forces acting on atoms in the complex, and updating atoms' coordinates based on the forces applied on them. Forces are obtained by differentiating the potential energy function with respect to atom coordinates. This procedure is generally repeated for many times until the energy of the system converges within a threshold. During minimization process, the particle moves can be made using one of many optimization approaches such as steepest descent, conjugate gradient, quasi-Newtonian, or Newton-Raphson. Depending on the method chosen, minimization requires computing the first, and in some cases the second, derivatives of the energy function. The choice of method also affects the rate at which the energy of the system converges.

Energy minimization is used in many biomolecular applications like molecular mapping and molecular docking programs. [3, 7]

## 1.2 Difficulties in Energy Minimization

The biggest problem with energy minimization is that it takes a long time to do the computations. The energy calculation consists many terms, and each of them require some amount of calculation. One of the energy terms, the electrostatic energy, requires the majority of the calculation time. It calculates the electrostatic energy and forces between atom pairs in the complex. The atom complex being evaluated usually consists of several thousand pairs, and the calculation performed is very complex with many high-order functions. These properties make energy minimization a computationally intensive and time-consuming process. In fact, energy minimization takes the majority of the runtime in many biomolecular applications like FTMap [3, 15]. Therefore if we can accelerate it, according to Amdahl's law, it could save a significant portion of runtime. In this thesis we introduce a microarchitecture to accelerate the electrostatic energy evaluation part of energy minimization. It has a demonstrated speedup of about 43x on a single last-generation FPGA.

## 1.3 Outline of the rest of thesis

The rest of this thesis is organized as follows:

- Chapter 2 introduces the energy minimization process, its terms, applications, and background information on FPGAs.
- Chapter 3 discusses issues related to porting original energy minimization program

to FPGAs, including data structure changes and FPGA architectures.

- Chapter 4 presents the implementation of main modules on FPGA.
- Chapter 5 presents the result of the system and discusses future works.

# Chapter 2

# Background

## 2.1 Biomolecular Applications

### 2.1.1 Molecular Dynamics

Molecular Dynamics(MD) is a form of computer simulation in which atoms and molecules are allowed to interact for a period of time by approximations of known physics, giving a view of the motion of the particles [6]. Figure 2·1 shows this iterative process.

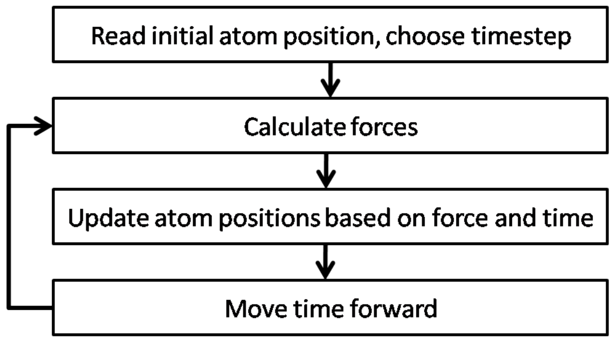
Read initial atom position, choose timestep
Calculate forces
Update atom positions based on force and time
Move time forward

**Figure 2·1:** Typical MD process

In each iteration, the forces acting on particles are evaluated based on their positions. Then the position and velocity of atoms are updated as a result of forces and used as next iteration's input. The forces evaluated here has multiple terms mainly in two categories: bonded("straight" bond , angle, and torsion) and non-bonded(Lennard-Jones, Coulomb, and hydrogen bond).

Since MD simulation often deals with systems with millions of particles, and non-bonded interaction scales quadratically, the computational complexity is usually very high. To reduce such complexity, a cut-off distance threshold is commonly applied. This works well since Lennard-Jones force reduces significantly when the distance between two particles is beyond a cut-off distance, usually between 8 Å and 12 Å. The Coulomb force is generally partitioned into a short range component that is compute along with the Lennard-Jones, and a long range component which is computationally less complex.

### 2.1.2 Molecular Docking

Molecular docking refers to the computational prediction of the pose (relative position and rotation) between two interacting proteins that has the least total energy. This is a computationally demanding process due to the complexity of the computation and the enormous size of the protein complex. Therefore, most docking systems adopt a two-step process. The first step is to find the best fit between two proteins, often assuming both proteins are rigid. During the process, it scores up to billions of poses between the two proteins and some of the top-scoring poses, often a few thousand, are preserved for further evaluation.

A typical second step is to minimize the total potential energy of the poses generated by the first step. This step is often referred to as minimization of the CHARMM potential, or *CHARMM minimization* due to the popularity of CHARMM force-field. During the step, the larger molecule of the pair(referred to as *receptor*) is generally held fixed, while the side chain atoms of the smaller molecule of the pair(referred to as *ligand*) are free to move. During each iteration, the total potential energy of the complex is calculated, and a move of the ligand to a neighboring point is made using a standard optimization technique, typically Newton-Raphson or quasi-Newtonian(L-BFGS). This procedure is repeated until the energy converges within a given threshold. Usually, several hundreds of iterations need to be performed for each conformation, resulting in about 30 seconds per conformation.

Since there are usually thousands of poses to be minimized per receptor-ligand pair, the total time for the minimization phase can run up to many hours. Moreover, in drug discovery, millions of candidate ligands are screened using docking-based methods for a given protein. Acceleration of energy minimization is thus highly desirable. Some of the docking programs that use energy minimization include DOCK [11], DARWIN [16], RDOCK [10] and EADock [7].

### 2.1.3 Molecular Mapping

Molecular mapping refers to the process that aims at finding possible binding sites on a given protein for a ligand. Mapping involves docking a set of small-molecule probes using rigid docking, and performing energy minimization for each protein-probe complex. The idea comes from observation that certain regions on protein binding sites, called "hotspots", contribute to the majority of the binding energy, and that they bind a large variety of small molecules. This leads to the idea that such regions are likely to bind inhibitors with high affinity [8].

One of the mapping algorithms, FTMap [3], utilizes energy minimization to help find binding sites that has high probability of binding 16 probe molecules. For each probe molecule, FTMap first runs rigid body docking. This is to filter the 2000 top-scoring conformations from billions of possible conformations. Then, for each of the 2000 protein-probe complex, energy minimization is run to find the minimum free energy. Then the complexes containing the same probe molecule are divided into clusters, which are ranked based on the average energy of complexes inside the cluster. The lowest-scoring six clusters are retained and divided into larger clusters, where clusters using different probes are mixed together. The cluster containing most sub-clusters is expanded to find the final binding site.

During FTMap, energy minimization is run tens of thousands of times to determine each conformation's minimum energy. Since cluster ranking uses a simple greedy algorithm and is not computationally intense, energy minimization takes the majority of the FTMap runtime. This is illustrated in section 2.3.

## 2.2 Difference between Energy Minimization and Molecular Dynamics

The underlying computation of energy minimization is superficially similar to that of molecular dynamics(MD), but has some fundamental differences. The first difference is how particle positions are updated. MD produces a trajectory based on kinetic energy and forces applied on theatom and does not care about total energy. Energy minimization adjusts the atom's coordinates only so as to lower the total energy of the system[4]. Also, minimization does not consider the effect of temperature. Therefore, the final state after minimization does not depend on the initial state at the beginning of minimization. The second difference is the number of particles in a typical complex. While MD often simulates many thousands to millions of particles, energy minimization is often performed on a local part of the complex, which consists only a few thousand atoms with tens of thousands of atom pairs. A third difference is that the actual energy expressions evaluated during minimization are different than those in MD. For example, in MD, the van der Waal energy term is evaluated using a 12-6 Lennard-Jones function, while in energy minimization it is often approximated with a sum of two or four Gaussians [12]. A fourth difference is that in MD, particularly in liquid simulations, each particle has a similar number of neighbors, typically in the hundreds. In energy minimization, most particles only have a small number of neighbors, although a few may have many more. And finally, in MD neighbor lists are updated every few iterations, in energy minimization these lists are only updated a few times over the entire computation.

## 2.3 Content of Energy Minimization Terms

In energy minimization, the total energy of the system is defined as the sum of bonded and non-bonded energies of all the atoms in the system. Equation 2.1 shows the total energy of the system as a function of various energy terms:

$$E^{system} = E^{elec} + E^{vdw} + E^{angle} + E^{bond} + E^{torsion} + E^{improper} \tag{2.1}$$

Of these different energy terms, the non-bonded energy evaluation, including electrostatic energy and van der Waals energy, is the most computationally intensive step, requiring more than 95% of the total runtime, based on previous profiling of serial FTMap program [15]. In non-bonded energy evaluation, the energy of each atom is the sum of contributions from all neighboring atoms within a certain cutoff distance. Thus, the computation is often arranged in a neighbor-list format, with each atom has a list of neighboring atoms.

Energy minimization involves evaluation of this expression every iteration. As said earlier, advancing to next iteration requires moving atoms in the direction of least energy conformation. Thus, at each iteration, the forces acting on atoms are also calculated and atoms are moved in the direction of such forces. Figure 2·2 shows the profiling of the FTMap program running on a serial processor.

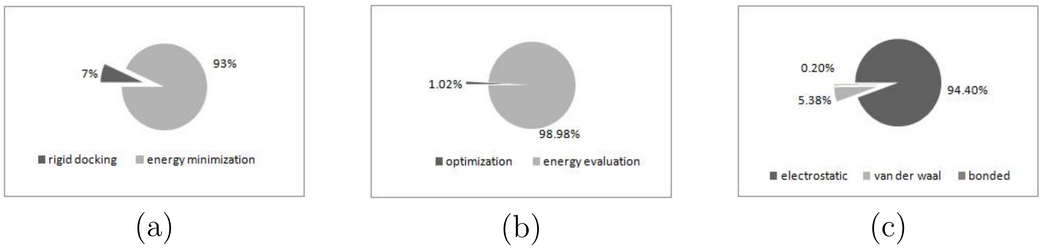

**Figure 2·2:** Runtime profiling of serial FTMap program. (a)function in one evaluation iteration; (b)function in energy minimization step; and (c)function in energy evaluation step

The electrostatic energy of a system with $N$ particles can be evaluated using equation 2.2.

$$E^{elec} = \sum_{i=0}^{N} E_i^{self} + \sum_{i<j,j<N} E_{ij}^{int} \tag{2.2}$$

The first term is the sum of $N$ self-energy terms, each of which is proportional to the square of the charge of the atom. This represents the electrostatic potential energy due to the charge itself. For a point charge, such energy is infinite [17]. Thus, computing the self energy requires the charge to be represented as distributed charge, which is often done by distributing the charge uniformly over a small sphere with radius $R_i$. [13]

The second term is the sum of pair-wise interaction terms between atoms, each of which is proportional to the product of two charges of the pair. Both self-energy and pair-wise terms depend on the geometry of the solute.

For the self-energy term, each atom's self energy is determined as a sum of its Born self energy in the solvent with dielectric constant $\epsilon_s$, and the sum of effective pairwise interactions with all other atoms, as equation 2.3 shows.

$$E_i^{self} = \frac{q_i^2}{2\epsilon_s R_i} + \sum_{i \neq j} E_{ij}^{self} \tag{2.3}$$

To compute the second term of equation 2.3, ACE defines atom charges as Gaussian distributions. $E_{ik}^{self}$ can be computed by integrating the energy density of the electric field. This is often approximated as the sum of a short-range term that approximates the Gaussian and a long range term. In equation 2.4, the first term represents the Gaussian, while the second term represents the long range term.

$$E_{ij}^{self} = \frac{\tau q_i^2}{\omega_{ij}} e^{-(\frac{r_{ij}^2}{\sigma_{ij}^2})} + \frac{\tau q_i^2 \widetilde{V}_k}{8\pi} (\frac{r_{ij}^3}{r_{ij}^4 + \mu_{ij}^4})^4 \tag{2.4}$$

Here $q_i$ is the charge on atom i, $r_{ij}$ is the distance between atom i and atom j. $\omega_{ij}$

and $\sigma_{ij}$ determines the Gaussian used to approximate $E_{ij}^{self}$ and $\mu_{ij}$ is an atom-atom parameter. $\widetilde{V}_k$ is defined as solvent-inaccessible volume, which is defined by the radius of a solvent probe sphere with radius $R_{probe}$.

The pair-wise interaction energy, $E_{ij}^{int}$, is given the generalized Born (GB) equation, which is the sum of Coulomb's law in a dielectric and the Born equation. [14]

$$E_{ij}^{int} = 332\sum_{i\neq j}\frac{q_i q_j}{r_{ij}} - 166\tau\sum_{i\neq j}\frac{q_i q_j}{\sqrt{q_{ij}^2 + \alpha_i\alpha_j e^{-(\frac{r_{ij}^2}{4\alpha_i\alpha_j})}}} \tag{2.5}$$

In equation 2.5, $\alpha_i$ and $\alpha_j$ represent the Born radii for atom i and j, respectively. Born radii values are determined by self energy values of the atoms.

Equations 2.4 and 2.5 are the main computations needed for all atom-atom pairs to evaluate the total electrostatic energy of a given complex. In addition, energy gradients need to be computed to determine the forces acting on the atoms, which in turn determine the new position of the atom.

## 2.4 Steps of Energy Minimization Process

Energy minimization is an iterative process. That is, each iteration's output becomes next iteration's input. The main steps in an iteration of energy minimization process is:

1. Check if the neighbor list needs update. If so, perform neighbor list update.
2. Perform calculations for electrostatic, van der Waals, and bonded energy as well as forces acting on each of the atoms in the complex.
3. Use energy values as well as force values to perform L-BFGS optimization on the atom complex, mainly coordinates of atoms.
4. If the total potential energy of the complex converges, finish. Otherwise, repeat from step 1.

## 2.5 Field Programmable Gate Arrays

Field Programmable Gate Arrays, or FPGAs, are pre-fabricated integrated circuits that can be prgrammed and reprogrammed to implement logic. A hardware description language, such as VHDL or Verilog HDL, is usually used to describe the logic being implemented. A compiler maps the logic to the device specified. Modern FPGAs have logic capacity equivalent to millions of gates as well as independently accessible on-chip memory and hardwired DSP blocks. Figure 2·3 shows the top-level architecture of current-generation Altera Stratix IV FPGA [1].

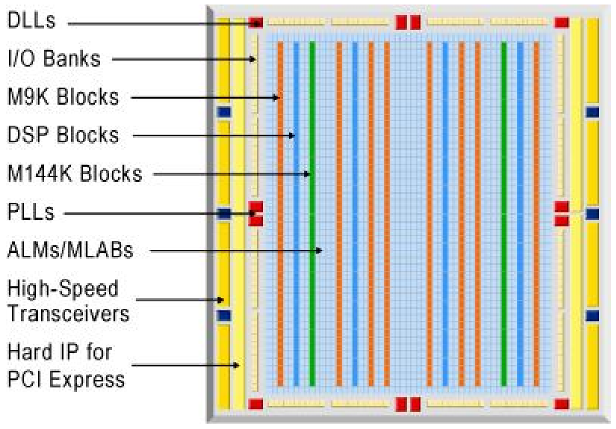

**Figure 2·3:** Stratix IV FPGA Architecture

Although FPGA performance is generally an order-of-magnitude lower than that of an equivalent ASIC, they are much cheaper to design, much less expensive(for modest quantities), as well as being versatile. Also, they often provide superior performance than a standard processor for specified applications despite running at a much lower frequency. The reason behind this is that FPGAs can be configured specifically for the application, creating customized architecture that is optimized for the application. Therefore, FPGA-based systems are often used to implement custom accelerators or co-processors for applications. Another major application domain is for prototyping new architectures. Also, FPGAs consume much less power than their generic processor counterpart, thanks to the lower operating frequency. This can make FPGA-based systems a great performance-per-watt value comparing to traditional HPC systems.

# Chapter 3

# System Architecture and Design

## 3.1 Data Structure

In the original serial code, input data of atom pairs is structured into a neighbor-list format, that is, a list contains all atoms that is the first atom in an atom pair, and each entry in that list has a pointer to another list that contains all atoms that forms a pair with this atom entry. Figure 3·1 shows the data structure.

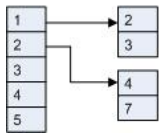

**Figure 3·1:** Neighbor list data structure

Such structure is not optimal to FPGA-based systems since the FPGA does not have special support for pointer arithmetic. However, FPGA-based systems are good at deep pipelining and custom architecture. Therefore, it's possible to change the atom pair-list structure so that its processing can be pipelined. We now introduce one data structure optimization for energy minimization on FPGA. The idea is to read every pairs in the list and put them into a long array, with each array element containing one pair. Therefore, it's able to pipeline the pair-list, thus maximizing the efficiency of FPGA. The advantaFigure 3·2 shows the transform of pair-list.

However, this approach has a drawback: when the first atom of the pair changes, it requires an additional cycle to update the partially accumulated energy or force value

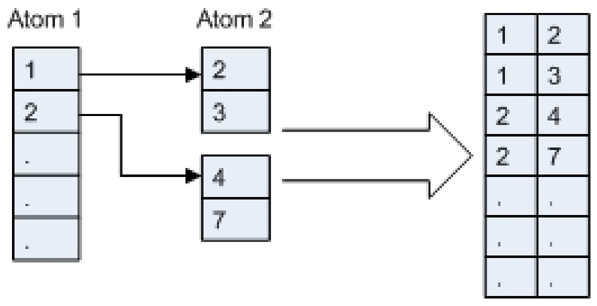

**Figure 3·2:** Data structure transformation

of the last "first" atom. For example, when the pair changes from 1/3 to 2/4, it requires an additional cycle to update partially accumulated value for atom 1. This drawback could be solved by future work discussed in Section 5.6.

## 3.2 FPGA as a Co-processor

Although the idea of using reconfigurable circuits for computing has been around since 1960's, it has only become feasible recently. One factor is that current high-end FPGAs are actually hybrid circuits that has many ASIC components(especially DSP blocks and memories) built in. Another factor is that FPGAs can be configured and reconfigured to different algorithms in a matter of milliseconds. Also FPGAs can be mapped to high-speed interface like PCI-express to off-chip devices, giving developers a chance to offload different work and data onto FPGA without losing too much performance due to the transfers.

Since FPGAs can be configured to perform almost arbitrary computations, and since the process of energy minimization is an iterative process that computes and combines different factors of forces acting on atoms(see Section 2.4), it is therefore able to offload some of the evaluation functions to FPGA while the host can calculate others, given there is no data dependency between host and FPGA. In this work, the electrostatic energy evaluation is offloaded to FPGA, and while FPGA is evaluating electrostatic energy, host machine evaluates other energy terms. At the end, results from FPGA and host are combined together to get the final force and energy values for optimization.

## 3.3 System Architecture

Figure 3·3 shows the overall system architecture.

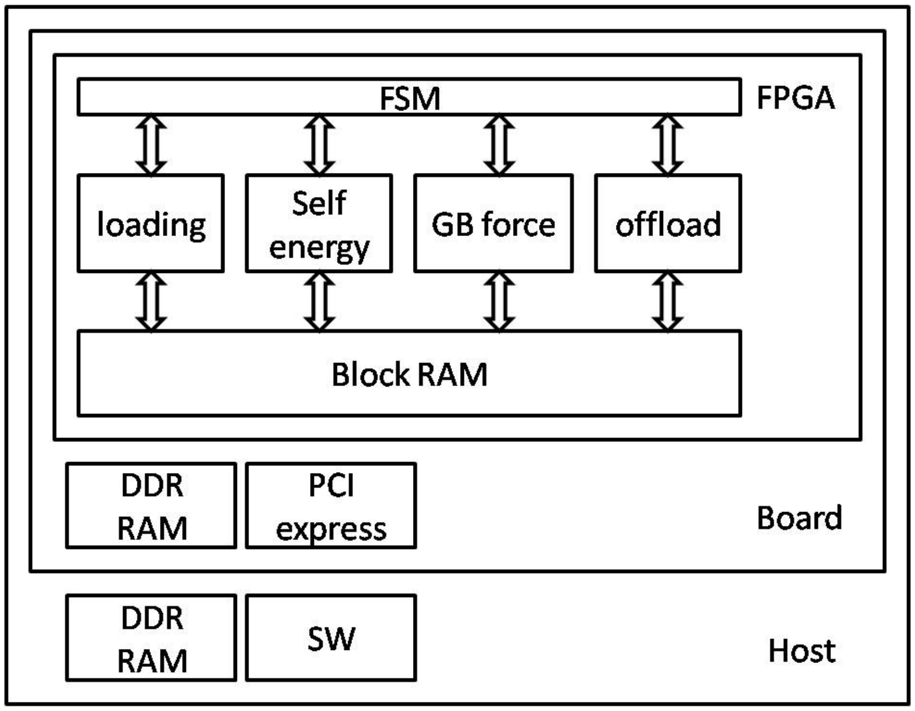

**Figure 3·3:** System Architecture

The system contains three major parts: the FPGA, the accelerator board that carries the FPGA and its external memory, and the host machine. FPGA is the main computation unit, and it stores atom's coordinates, as well as parameters used for self energy calculation. The acceleration board has a larger memory space for data structures such as the formatted neighbor list. Also the board provides a interface between FPGA and host, allowing host machine to transfer data to FPGA via PCI express bus. The host machine controls the running of the FPGA, and it calculates part of the minimization process that is not accelerated by the FPGA.

# Chapter 4

# FPGA Implementation

## 4.1 Floating Point Datapath

### 4.1.1 Floating point datapath on FPGA

Although FPGAs are well known for their customizable architecture and programmability, the computation power within FPGAs are sometimes underestimated, especially when it comes to floating point computation. Generic processors like Intel's Xeon and AMD's Operton have a piece of dedicated hardware specially optimized for floating point computation, which FPGAs usually lack. Modern FPGAs, however, are capable of doing complex computation, thanks to the increased DSP blocks being built into FPGA chip. For example, one version of the current generation of Altera's Stratix family, Stratix IV, has 1288 18x18 multipliers [2]. Creating a floating point datapath remains a problem for designers. Manually connecting IP cores is one solution, but that is inefficient and wastes resources. For example, concatenated IP FP cores have redundant normalization and denormailzation. Since the goal is the get the final computation value, there is no need to fully normalize/denormalize all the intermediate variables between stages, which is the case for manually connecting IP cores.

### 4.1.2 Floating Point Compiler

In this work, we use a C compiler designed for FPGA by Altera. The floating point compiler [9] as this product is called, translates a C function into a data flow graph(DFG), It uses the DFG, together with its internal floating point library to implement the function in VHDL. This approach saves a significant amount of area since it does not normalize/denormalize every intermediate value. Instead it uses a "fused" approach. That is, it uses a different representation of numbers, and does normalization/denormalization only after several operation stages. Based on the number of stages between normalization/denormalization, which is customizable between 2 and 16, the fused approach could save about 50% of the area and latency, comparing to sequential IEEE 754 datapath. Figure 4·1 shows the difference between two methods.

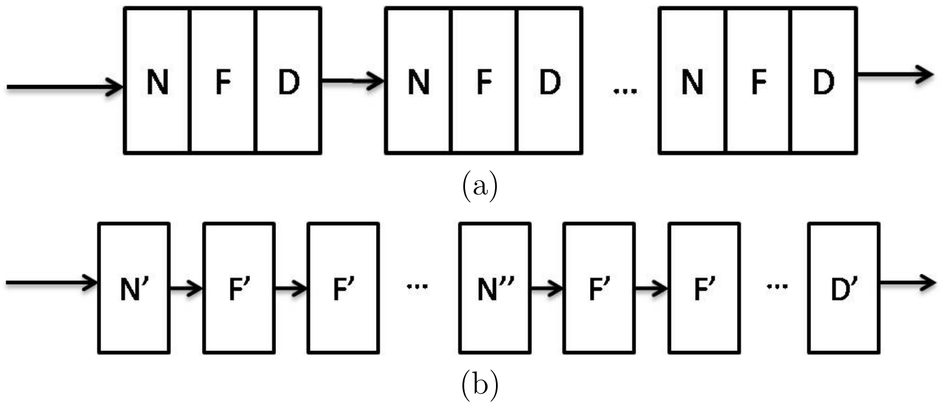

**Figure 4·1:** Floating datapath. (a) manually connecting IP; and (b) using Floating Point Compiler

In figure 4·1, N stands for normalization, F for fixed-point arithmetics, and D for denormalization. We can see that in manually built path, multiple normalization and denormalization is used through the datapath. These parts could be replaced by using another data format and flowing through modified fixed-point arithmetic block(indicated by N' and F'), with denormalization/re-normalization taking place after several stages(as indicated by N") before being denormalized back to IEEE-754 format.

#### 4.1.3 Precision in Floating Point Computation

Unlike fixed-point datapath, where the range of operand is pre-defined, floating-point datapath has to deal with data with potentially large dynamic range. Therefore overflow/underflow may happen when operands have a significant difference. Such exceptions may affect the accuracy of the computation. It is explained in [9] that it's possible to avoid such error by allowing such exception flow within a cluster before being normalized. However, the overflow/underflow is not guaranteed to be solved by doing this, and such error could propagate through the datapath, causing precision error at the output of the computation. This will be discussed further in 5.6.

## 4.2 Data pre-optimization for FPGA

The data used by FPGA includes mostly atom coordinates and parameters for self energy calculation. From profiling data and analysis of the algorithm, we found that some of the data could be pre-optimized to save resource on FPGA. The self energy calculation uses parameters related to atom types to determine the Gaussian used to evaluate energy function. From profiling it is known that the parameter matrix is sparse and could be shrunk to save on-chip memory space. In fact, by only transferring parameters that are featured in the pair list, we save 99% of the memory that would be occupied by these parameters. Also, analysis of equation 2.4 shows that the $\mu_{ij}$ and $\sigma_{ij}$ term can be pre-computed on the host before being transferred to FPGA. This save some multiplier usage and latency in the floating-point pipeline.

## 4.3 Overall design

The overall block diagram of the FPGA computation unit is shown in figure 4·2.

The GiDEL IP shown in the figure is dealing with interfacing with on board DDR

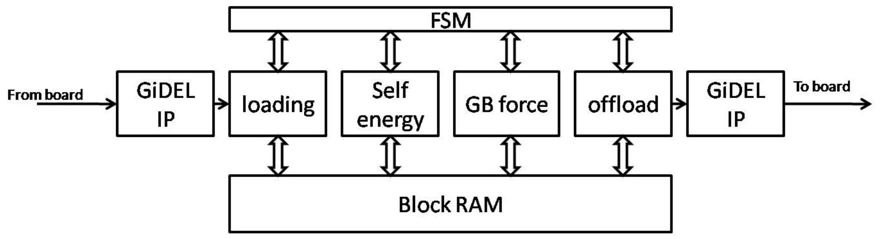

**Figure 4·2:** Block diagram of FPGA design

RAM, providing a easy-to-use interface for user design to access data. It is provided by the manufacturer of the development board. Now we discuss the function of other modules.

## 4.4 Loading module

The loading module is responsible for loading the parameters, as well as atom coordinates and initial self energy values, from on-board DDR memory and into FPGA's Block RAM(BRAM). Since the parameters used by equation 2.4 does not change during the entire application, they only need to be loaded once. Coordinates and energy values, on the other hand, change on every iteration, and therefore must be loaded every iteration. In order to speed up the loading, the loading module uses a parallel-loading scheme to write multiple values into their respective destinations at the same time. The block diagram for loading module is shown in figure 4·3.

## 4.5 Self Energy Pipeline

The self energy pipeline calculates the update on each atom's self energy, as well as the switching function, its derivative, and forces acting on each atom pair. Figure 4·4 shows the architecture of self energy pipeline.

The inputs are atom coordinates from each atom in a pair, as well as parameters related to types of the atoms. The comparators are used to determine whether the

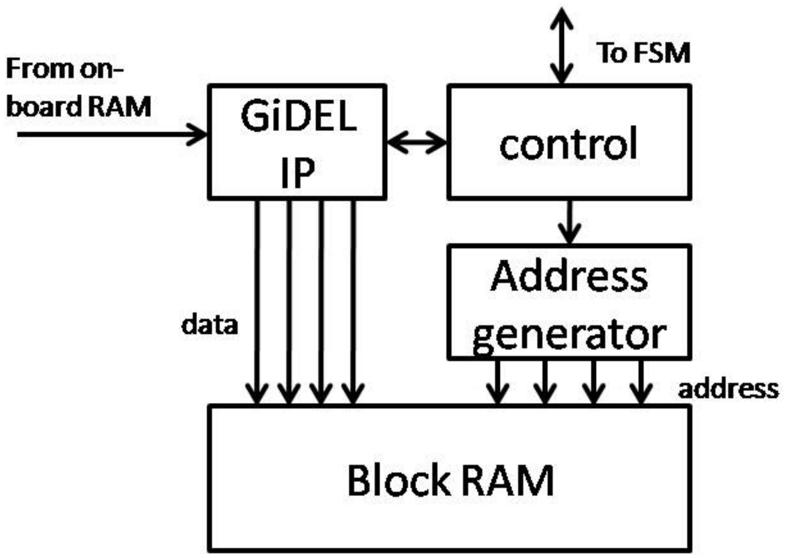

**Figure 4·3:** Block diagram of loading module

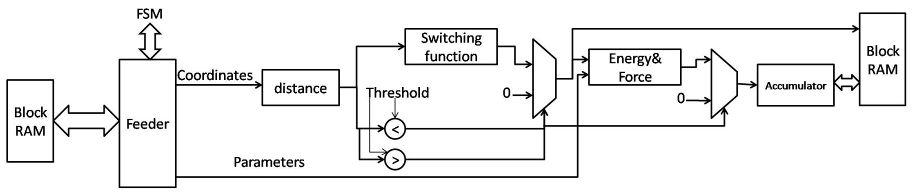

**Figure 4·4:** Self Energy Pipeline

switching function number is to be calculated or a pre-defined value is to be used. They also determine if the distance between atoms is larger than the cutoff; if so, then the self energy and force coefficient from this pair are ignored. The output goes into a ID-based fixed-point accumulator described in section 4.7 to update the self energy value stored in BRAM, while force coefficient is stored serially in another BRAM.

## 4.6 Pair-wise Energy Pipeline

The pair-wise energy pipeline finishes the calculation. It consists of two main parts to finish the calculation. The reason for using two parts is that one variable of the GB pipeline has to be accumulated before it can be used, creating a double summation which cannot be pipelined. Figure 4·5 shows the first part of the pair-wise energy pipeline.

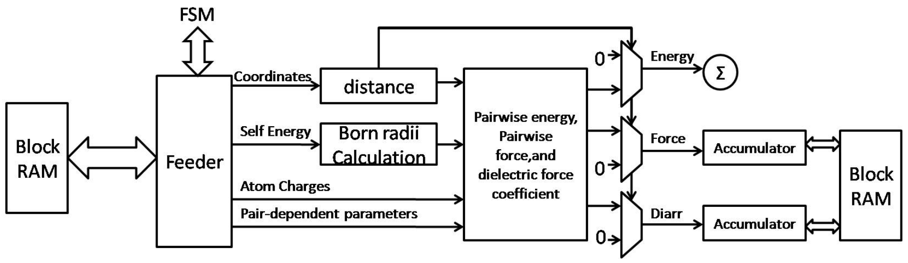

**Figure 4·5:** First part of GB Pipeline

As shown in figure 4·5, the Born radius of each atom in a pair is calculated based on atom's self energy value, and then be used to calculate forces and an intermediate value for the next stage. Both of these values are accumulated to their respective atom, and the sum is used by the next part of the pair-wise energy pipeline.

It is worth mentioning that there are three different kinds of atom pairs in the original data input, the computation performed in this part of the pipeline is slightly different among these kinds. This is addressed by adding a pair-dependent parameter to the input of the pipeline, and by slightly modifying the computation structure to use a unified computation structure.

The second part of the pair-wise energy pipeline is shown in figure 4·6.

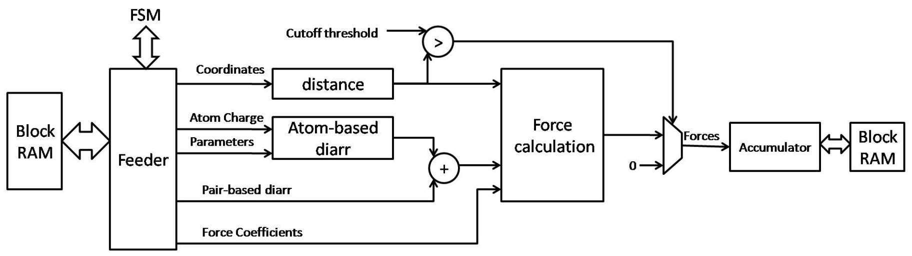

**Figure 4·6:** Second part of GB Pipeline

This part uses the intermediate value calculated in the first part of pair-wise energy pipeline and the force coefficients calculated in self-energy pipeline to update the forces acting on each atom. Note that the *diarr* value has two parts. The first part is atom-based, that is, each atom has its own *diarr* value, regardless of whether it interacts with other atoms. The second part is the sum of partial *diarr* values because of pair-wise interaction. Since the pair-wise energy pipeline is based on atom pairs, a bit vector indicating whether the atom-based diarr value has been added to the total diarr value is introduced to avoid accumulating atom-based diarr value multiple times.

## 4.7 Accumulator

For each pipeline, the output of self energy or force values comes with a corresponding atom ID. Since the data is in IEEE-754 format, the accumulation of outputs is a problem. Conventional IEEE-754 accumulators can only deal with a stream of data accumulating to one variable with latency of several clock cycles. This does not fit this application since the target of accumulation often changes, which requires frequent pre-load for the accumulator. Moreover, since every clock cycle there are 2 values for different atoms coming, it is possible that the same atom ID will show up in one stream while it is in the accumulation pipeline of the other stream. Such a situation leads to inaccurate accumulation due to a data hazard. We now introduce an accumulator that is designed for this application. The block diagram for the accumulator is shown in figure 4·7.

Given the data structure shown in figure 3·2, it is desirable to accumulate data coming from input stream 1, where the atom ID is stable for a period of time, while doing simple addition from input stream 2, where the atom ID changes every clock cycle. An ID conflict may have to be solved though. That is, when input stream 1 starts to accumulate value for atom ID 2, it needs to pre-load the current value from memory. If atom ID 2's value is not yet updated with the value from input stream 2 (two cycles ago), accumulator is pre-loading with the wrong value, causing the output to be wrong. A forwarding path is therefore provided in the accumulator to ensure that the pre-load

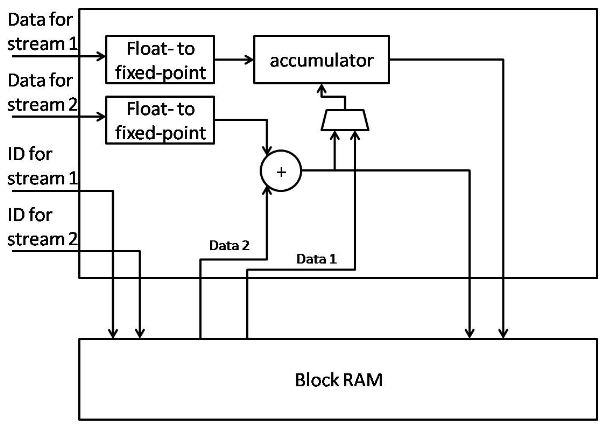

**Figure 4·7:** Block diagram of accumulator

value of the accumulator is correct.

The dual portedness of the on-chip BRAM is utilized in this accumulator. One port of the BRAM is dedicated to writing the updated value back to BRAM, while the other port is dedicated to reading the existing value from BRAM. As stated in Section 3.1, it requires 1 clock cycle to store the value in the accumulator once the atom ID from stream 1 changes. This is done on the host software by inserting a pair identical to the last pair that has the same first atom. When accumulator reads identical pairs in consecutive clock cycles, it will write the sum from accumulator to atom ID 1's address in BRAM, instead of writing the sum from the adder to atom ID 2's address in BRAM.

# Chapter 5

# Result and future work

## 5.1 Toolset and reference

The FPGA configurations are written in VHDL. The host code in C and C++. The FPGA programming files were generated using Quartus II v9.1 on a 64-bit machine running Windows XP. The reason for using a 64-bit machine is that windows XP on 32-bit machine cannot support memory usage by a single process larger than 2GB, while the compilation of the FPGA design requires more than that.

To validate the correctness and accuracy of the FPGA system, two different reference code are used. One reference code is created in C as the basis of this work. Since floating point datapath is used, Another reference code is created in MATLAB to verify FPGA system's compliance with IEEE-754 standard.

## 5.2 Test platform

The single-precision version of FPGA code runs on a GiDEL PROCe III board featuring an Altera Stratix III EP3SE260 FPGA and 4.5GB of on-board DDR RAM. The host code runs on one core of a dual-core 2.8GHz Intel Xeon CPU with 2GB RAM. The operating system is 32-bit Windows XP. The host code is compiled using Microsoft Visual Studio 2005 with /O2 option for maximizing speed. The interface between host and board is PCI Express x4. Figure 5·1 shows the top-level block diagram of GiDEL board [18]. The reference code (single precision C) runs on a single core of a 2.8GHz

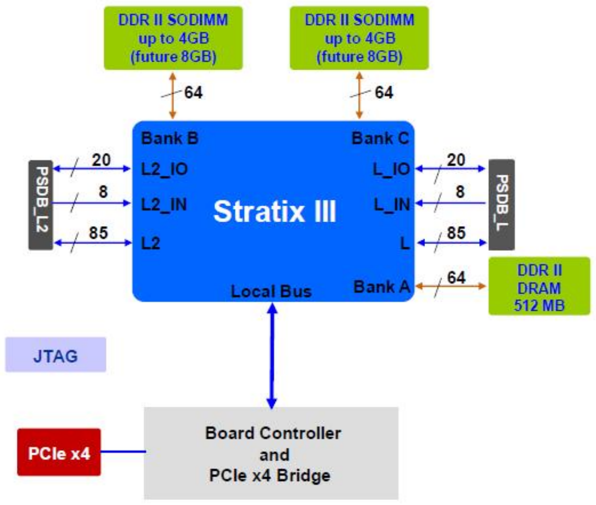

**Figure 5·1:** Block Diagram of GiDEL PROCe III board

dual-core Intel Xeon CPU with 2GB RAM. The operating system is 32-bit windows XP. Reference code is compiled using Microsoft Visual Studio 2005 with /O2 option for maximizing speed. The reference single-precision MATLAB code is running on a single core of a 3.0GHz dual-core Intel Pentium D CPU with 1GB RAM. The operating system is 64-bit Linux. MATLAB version is 7.8.0(R2009a).

The dataset used for testing contains 5 different probes, 2258 atoms in 22 different types, and approximately 9800 pairs at the beginning of the simulation.

## 5.3 Results

We now present the resuls. Table 5.1 shows the resource used by the main computation blocks and surronding modules like loading, offloading and accumulator. In total, these blocks used all but 4 of the available DSP blocks on Stratix III SE260 FPGA. DSP block number is the main limiting factor for this implementation to scale. Moving to current-generation FPGAs or multi-FPGA systems should help solve this problem. This will be discussed in section 5.6.

It is noticeable that the Altera Floating Point Compiler generates a large number of

| *resource* | *self-energy pipeline* | *pair-wise pipeline pt.1* | *pair-wise pipeline pt.2* | *loading* | *offloading* | *accumulator* |
|---|---|---|---|---|---|---|
| combinational ALUTs | 18552 | 21503 | 15994 | 452 | 625 | 651/ea |
| Registers | 57362 | 59820 | 26290 | 506 | 905 | 1063/ea |
| Multipliers | 276 | 332 | 156 | 0 | 0 | 0/ea |
| M9K BRAM | 356 | | | | | |
| M144K BRAM | 16 | | | | | |

**Table 5.1:** Resource Utilization on Stratix III SE260

computation blocks that utilize the built-in DSP block multipliers. Also, that equations 2.3 and 2.5 require a number of division. To avoid using division and to further utilize DSP blocks, we used a inverse-multiply wherever possible.

## 5.4 Accuracy

To verify the accuracy of the system, we made a number of comparisons of the accelerated and the unaccelerated systems. These are the coordinate atoms, the force values generated during electrostatic energy evaluation, and the total electrostatic energy.

### 5.4.1 Force values

The total force values calculated by FPGA has a maximum difference of $2.49\times10^{-5}$ N with respect to CPU code, and a maximum difference of $2.47\times10^{-5}$ N is observed between FPGA and MATLAB code. The main reason for such difference is that the FPGA floating-point datapath uses a different mantissa length than that of Intel processor. This difference could introduce rounding errors, since when normalizing operands for floating point calculations, mantissa of one number may be shifted too much. Although FPC uses several guard bits, it may not fully prevent such situation from happening. Also, because of the FPC's use of fused datapath, it does not denormalize the result for every stage of the datapath. Therefore rounding errors and underflows could propagate within the datapath. Although they may be canceled by subsequent operation within the cluster, they may well propagate across the cluster border and become error.

### 5.4.2 Coordinates

The coordinate values calculated by FPGA has a maximum difference of $1.24\times10^{-3}$ Å. The main reason for this difference is that the force calculation has the precision error described above. Since energy minimization is an iterative process, this error is propagated to the next iteration, therefore the final result could be random. In actual run, the final coordinates of atoms of interest have a maximum difference of 0.5 Å, with most atoms matching precisely. This is acceptable for most applications.

### 5.4.3 Total energy

The total energy values calculated by FPGA has a difference of about 0.014%. The main reason for such difference is again that FPC has some internal issues that may generate several bits of rounding error as well as the input data has too much range. However, over 25 iterations where the FPGA calculation converges for probe 0, the difference between CPU result and FPGA result remains within acceptable range.

## 5.5 Performance

Table 5.2 shows the time used by one iteration of electrostatic energy evaluation on FPGA and on host.

We managed to get the system running at 125MHz with no difficulty. The com-

| platform | host-FPGA transfer | computation | FPGA-host transfer | total |
|---|---|---|---|---|
| FPGA | 7ms | 0.46ms | 6ms | 13.46ms |
| CPU | N/A | 15.82ms | N/A | 15.82ms |
| Speedup | N/A | 34.39x | N/A | 1.17x |

**Table 5.2:** Run time per iteration, single precision

| Complex | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| FPGA SP iteration | 25 | 27 | 29 | 15 | 22 |
| FPGA SP time | 0.578 | 0.539 | 0.534 | 0.312 | 0.421 |
| CPU SP iteration | 50 | 44 | 61 | 67 | 69 |
| CPU SP time | 0.859 | 0.766 | 1.047 | 0.969 | 0.89 |

**Table 5.3:** Iteration count and end-to-end time(in seconds) for each complex

putation time for FPGA shown here reflects all five stages of the system. From table 5.2, we obtain a speed-up of about 34.39x for the computation part of the electrostatic energy evaluation. However, the transfer between host and FPGA is the bottleneck of the system, dragging down the performance to about 1.17x. The reason behind this is that the transfer between host and FPGA is through a 4-lane PCI-express 1.0 bus, and the overhead of establishing a DMA connection is much more than the time that the actual data transfer take. Also, because of the resource limit on FPGA we currently use, it can only fit the electrostatic energy part of the computation, meaning that the total electrostatic energy, as well as each atom's force values have to be transferred back to host before the computation can continue. By moving to newer generation FPGAs that have larger capacity, such transfer may be eliminated by performing other energy computation and optimization from host to FPGA. In that case, the transfer time could be reduced to twice per complex(initial coordinates and final coordinates), instead of twice per iteration.

## 5.6 Future work

There is much room for future optimization. One is to modify the accumulation module to further utilize the simultaneous 2-port access capability of on-chip BRAMs to eliminate the 1-cycle stall in the stream. Another is that the precision of floating point computation could be increased by using a later version of Floating Point Compiler or a custom/proprietary floating point library, and so giving a more accurate result. Also, although frequency is not the main roadblock for performance, moving to the current-generation FPGA e.g. Altera's Stratix V, will allow more logic to fit on the chip as well as enabling a higher operating frequency. The benefit of more logic is that for now only the electrostatic energy part of the calculation is done on FPGA. By adding the Van der Waals calculation, it could accelerate 99.8% of the total calculation. Furthermore, by using a simpler optimization method that requires less logic than L-BFGS, we could put the whole probe on FPGA, eliminating majority of the host-board transfer.

# References

[1] Altera Corp. `http://www.altera.com/products/devices/stratix-fpgas/stratix-iv/overview/architecture/stxiv-architecture.html`.

[2] Altera Corp. Stratix IV Datasheet. `http://www.altera.com/literature/hb/stratix-iv/stx4_siv51004.pdf`.

[3] R. Brenke, D. Kozakov, G.-Y. Chuang, D. Beglov, D. Hall, M. R. Landon, C. Mattos, and S. Vajda. Fragment-based identification of druggable 'hot spots' of proteins using fourier domain correlation techniques. *Bioinformatics*, 25(5):621–627, March 2009.

[4] B. Brooks, R. Bruccoleri, D. Olafson, D. States, S. Swaminathan, and M. Karplus. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4:187–217, 1983.

[5] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *J. Am. Chem. Soc.*, 117:5179–5197, 1995.

[6] D.C.Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 1996.

[7] A. Grosdidier, V. Zoete, and O. Michielin. EADock: docking of small molecules into protein active sites with a multiobjective evolutionary optimization. *Proteins*, 67(4):1010–1025, June 2007.

[8] M. R. Landon, D. R. Lancia, J. Yu, S. C. Thiel, and S. Vajda. Identification of hot spots within druggable binding regions by computational solvent mapping of proteins. *Journal of medicinal chemistry*, 50(6):1231–1240, March 2007.

[9] M. Langhammer. Floating point datapath synthesis for fpgas. In *IEEE Conference on Field Programmable Logic and Applications*, 2008.

[10] L. Li, R. Chen, and Z. Weng. RDOCK: refinement of rigid-body protein docking predictions. *Proteins*, 53(3):693–707, November 2003.

[11] E. Meng, D. Gschwend, J. Blaney, and I. Kuntx. Orientational sampling and rigid-body minimization in molecular docking. *Proteins*, 17:266–278, Nov. 1993.

[12] R. V. Pappu, R. K. Hart, and J. W. Ponder. Analysis and application of potential energy smoothing and search methods for global optimization. *The Journal of Physical Chemistry B*, 102(48):9725–9742, November 1998.

[13] M. Schaefer and M. Karplus. A comprehensive analytical treatment of continuum electrostatics. *The Journal of Physical Chemistry*, 100(5):1578–1599, January 1996.

[14] W. C. Still, A. Tempczyk, R. C. Hawley, and T. Hendrickson. Semianalytical treatment of solvation for molecular mechanics and dynamics. *Journal of the American Chemical Society*, 112(16):6127–6129, August 1990.

[15] B. Sukhwani and M. Herbordt. Acclelerating energy minimization using graphics processors. *Proceedings of Symposium on Application Accelerators in High Performance Computing*, 2009.

[16] J. S. Taylor and R. M. Burnett. DARWIN: A program for docking flexible molecules. *Proteins: Structure, Function, and Bioinformatics*, 41(2):173–191, 2000.

[17] `http://farside.ph.utexas.edu/teaching/em/lectures/node56.html`.

[18] `http://gidel.com/pdf/PROCeIIIProductBrief.pdf`.