# FPGA-based Acceleration of CHARMM-potential Minimization[*][†]

Bharat Sukhwani Martin C. Herbordt

Computer Architecture and Automated Design Laboratory
Department of Electrical and Computer Engineering
Boston University, Boston, MA 02215

## ABSTRACT

Energy minimization is an important step in molecular modeling, with applications in molecular docking and in mapping binding sites. Minimization involves repeated evaluation of various bonded and non-bonded energies of a protein complex. It is a computationally expensive process, with runtimes typically being many hours on a desktop system. In the current article, we present acceleration of the energy evaluation phase of minimization using Field Programmable Gate Arrays. We project a multiple orders-of-magnitude speed-up over a single CPU core and a factor of 8 speed-up over our previous acceleration using an NVIDIA Tesla 1060 GPU.

## 1. INTRODUCTION

Molecular docking refers to the computational prediction of the least energy pose between two interacting proteins. This is a computationally demanding process, often requiring many hours to days of CPU runtime. Due to the computational complexity of the problem, most docking systems adopt a two step process. The first step docks the two proteins to find the best fit, often assuming the proteins to be rigid. It scores up to billions of poses between the two proteins and a few thousand top scoring poses are preserved for further evaluation.

The second step performs minimization of the total potential energy of the high-scoring docked complexes generated by the first step. Often, this is referred to as minimization of the CHARMM potential (or force-fields) or simply as *energy minimization*. During energy minimization, the larger molecule is generally held fixed whereas the side chain atoms of the smaller molecule (the ligand) are free to move. This accounts for the flexibility in the molecule. During each minimization iteration, the total potential energy of the complex is computed and a move to a neighboring point is made using a standard optimization technique. This is typically Newton-Raphson or quasi-Newtonian (L-BFGS). The minimization process is repeated until the energy converges to within a given threshold. Often up to a thousand iterations need to be performed per conformation, resulting in runtimes of about 30 seconds per conformation [1].

With many thousands of conformations to be minimized per protein-ligand pair, the total time for the minimization phase runs can be many hours. Moreover, drug discovery involves the docking-based screening of millions of candidate ligands for a given protein. Acceleration of energy minimization is thus highly desirable, as it would lead to faster molecular docking and quicker turn-around times in drug discovery. Some of the docking programs that use energy minimization include DOCK [2], DARWIN [3], RDOCK [4] and EADock [5].

In addition to docking, energy minimization is also used in other molecular modeling applications. One of these is *mapping*, which aims at determining the likely binding sites on a given protein [1][6][7][8]. Mapping involves docking a set of small-molecule probes using rigid docking, and performing energy minimization for each protein-probe complex. The idea arises from the observation that certain regions on protein binding sites, called hot spots, are major contributors to the binding energy and that they bind a large variety of small molecules. Thus, regions that bind a large number of probes can be indicative of sites that are likely to bind inhibitors with high affinity [9]. FTMap [1], one of these mapping algorithms, docks 16 different probes to a given protein. For each probe, it retains the 2000 top scoring conformations from rigid docking step; these are then

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*HPRCTA'09*, November 15, 2009, Portland, Oregon, USA.
Copyright 2009 ACM978-1-60558-721-9/09/11... $10.00.

[*] This work was supported in part by the NIH through award #R01-RR023168-01A1, and facilitated by donations from XtremeData, Inc., SGI, and Xilinx Corporation.

[†] 8 St. Mary's St., Boston, MA – 02215. 1-617-353-2850.
Web: www.bu.edu/caadlab. Email: {herbordt|bharats} @bu.edu

further minimized using the CHARMM potential [10].

While energy minimization can potentially benefit from acceleration, and while this computation is superficially similar to the well-studied molecular dynamics, little or no work has been done in this area. The difficulty is perhaps that there is comparatively little computation to be performed per iteration and that a substantial fraction of it is apparently serial. On the FPGA, however, these computations can be performed using deep pipelines, thus effectively performing many computations in parallel.

The focus of the current paper is accelerating the energy evaluation step of the FTMap algorithm using reconfigurable hardware. The potential energy terms that FTMap employs are the Analytic Continuum Electrostatics (ACE) and solvation energies [11], in addition to other CHARMM energy terms such as the van der Waals and bonded energies. Of these, the ACE electrostatics, solvation, and van der Waals terms represent the non-bonded (external) interactions and constitute most of the computation.

In the current paper, we present a design for mapping the evaluation of ACE electrostatics and solvation terms onto an FPGA, with an estimated speedup of two orders of magnitude on computations that constitute more than 90% of total runtime. This represents a 14x overall speedup of the FTMap program. The overall speedup is limited due to the van der Waals term being evaluated on the host. Acceleration of the van der Waals evaluation using an FPGA would result in higher overall speedup and is part of the future work.

## 2. ENERGY MINIMIZATION

Energy minimization is an iterative process which aims at computing the configuration of the atoms in a complex that corresponds to the minimum potential energy [13]. In order to evaluate the potential energy of the system rapidly, it is often represented using force-fields. A force-field represents each atom in a molecular system as a point charge and the total potential energy of the system as a sum of various two, three or four-particle interactions [14]. Various force fields have been developed, with the more popular ones being CHARMM [10] and AMBER [15]. Due to the popularity of the CHARMM force fields, energy minimization is often referred to as minimization of the CHARMM potential or simply as *CHARMM minimization*.

Minimization involves computing the potential energy of the complex at a point, updating the forces acting on the atoms, and adjusting the atom-coordinates according to the total forces acting on them. Forces acting on the atoms are obtained by differentiating the potential energy function with respect to the atom coordinates. This process of energy evaluation and of force and position updates is repeated for many iterations, until the energy of the system converges to within a threshold. During minimization, the move to the next iteration 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 functions. The choice of iteration method also affects the rate at which the energy of the system converges.

Though the underlying computation of energy minimization is superficially similar to molecular dynamics (MD), it differs from MD simulations in various ways. First, unlike molecular dynamics where the movement of the atoms is based on Newtonian dynamic laws and produces a trajectory based on kinetic energy, minimization simply adjusts the atom coordinates so as to lower the total energy of the system [10][13]. Minimization does not include the effect of temperature, and the final state of the system corresponds to the atom configurations when the temperature is approximately zero [13]. For the above reasons, the final state achieved after minimization does not depend on the initial state chosen at the start of the minimization process. Second, unlike MD where the system typically consists of millions of particles, energy minimization is often performed on a local region of the complex, resulting in only a few thousand atoms begin simulated and requiring only up to a few tens of thousands of atom-pair evaluations per iteration. Finally, even though the energy terms computed in minimization are similar to those in molecular dynamics, the actual energy expressions evaluated during minimization are quite different. For example, unlike molecular dynamics where the van der Waals term is evaluated using a 12-6 Lennard-Jones function, a minimization routine often approximates it with a sum of two or four Gaussians [16].

In energy minimization, the system to be simulated consists of a number of atoms; the total energy of the system is a sum of bonded and non-bonded energies of all the atoms. Of these, the non-bonded energy evaluation step is the most computationally intensive, requiring more than 95% of total runtime. Here, the energy of each atom is a sum of the contributions due to neighboring atoms within a cutoff distance. This computation is often arranged in a neighbor lists format, where each atom has an associated list of neighbors that contribute to its energy.

Equation 1 shows the total potential energy of a complex as a function of various bonded and non-bonded terms:

$$E^{total} = \underbrace{E^{vdw} + E^{elec}}_{non-bonded} + \underbrace{E^{bond} + E^{angle} + E^{torsion} + E^{improper}}_{bonded} \quad (1)$$

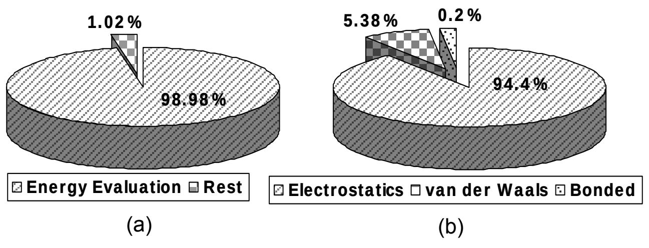

**Figure 1: Profiling of Serial FTMap program**

Energy minimization involves the repeated evaluation of this expression, once during each minimization iteration. As stated earlier, moving to the next iteration requires moving the atoms in the direction of the least energy conformation. Thus at each iteration, the total force acting on each atom is also computed and the atoms are moved in the direction of those forces. Figure 1 shows the profiling result for the FTMap program. As shown in Figure 1a, we see that more than 98% of the minimization runtime is spent in evaluating the total energy. Of this, almost 95% is spent computing the electrostatic energy (Figure 1b). This also includes the time to compute the total forces acting on the atoms. The focus of the current article is the acceleration of these computations using an FPGA.

The electrostatic energy of a solute with N charges can be decomposed into two components. The first component is a sum of N self-energy terms, each proportional to the square of the corresponding charge value. The self energy of a charge represents the electrostatic potential energy at the point where the charge is located, due to the charge itself. This effectively represents the energy required to assemble the charge. For a point charge, this energy is infinite [17][18]. Computing the self-energy, thus, requires the charge to be represented as a distributed charge. This is often done by distributing the charge uniformly over a small sphere [11][19].

The second term is the sum of the N(N-1)/2 pair-wise interaction terms, each proportional to the product of the two charges involved in the pair. Both the self-energy and the pair-wise interaction energy terms depend on the geometry of the solute [11]. The total electrostatic energy of a solute is thus given as a sum of all the self-energies, $E_i^{self}$, and the pair-wise interaction energies, $E_{ij}^{\text{int}}$ [11] (see Equation 2).

$$E^{elec} = \sum_i E_i^{self} + \sum_{i<j} E_{ij}^{\text{int}} \qquad (2)$$

For the ACE component, the self-energy of an atom is represented as a sum of its Born self-energy in the solvent plus the sum of effective pairwise interactions, $E_{ik}^{self}$, due to all the other solute atoms (see Equation 3) [11].

$$E_i^{self} = \frac{q_i^2}{2\varepsilon_s R_i} + \sum_{k \neq i} E_{ik}^{self} \qquad (3)$$

To compute the pair-wise interaction portion of the self energy, ACE defines atom charges as Gaussian distributions. $E_{ik}^{self}$ can then be computed by integrating the energy density of the electric field. For efficient computation, this is approximated as the sum of a short-range term that approximates the Gaussian and a long range term (first and second terms in Equation 4 respectively).

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

Here $q_i$ represents the charge on atom 'i', $r_{ik}$ is the distance between atoms 'i' and 'k', $\widetilde{V}_k$ is the size of the solute volume associated with atom 'k', $\omega_{ik}$ and $\sigma_{ik}$ determine the height and width of the Gaussian that approximates $E_{ik}^{self}$ and $\mu_{ik}$ is an atom-atom parameter.

The pair-wise interaction energy is given by the generalized Born (GB) equation, which is the sum of Coulomb's law in a dielectric and the Born equation [20]:

$$E_{ij}^{\text{int}} = 332\sum_{j \neq i} \frac{q_i q_j}{r_{ij}} - 166\tau \sum_{j \neq i} \frac{q_i q_j}{\sqrt{r_{ij}^2 + \alpha_i \alpha_j e^{-\left(r_{ij}^2 / 4\alpha_i \alpha_j\right)}}} \qquad (5)$$

where $\alpha_i$ and $\alpha_j$ represent the Born radius for atoms 'i' and 'j', respectively. These in turn depend on the self-energy of the atom.

Equations (4) and (5) represent the main computations that need to be performed for all atom-atom pairs to evaluate the total electrostatic energy of a given conformation. In addition, the energy gradients need to be computed to determine the forces acting on the atoms and update the atom coordinates.

## 3. ENERGY COMPUTATION: DATA STRUCTURES

In the original ACE computation, atoms are arranged in neighbor lists. Each "first atom" 'i' in the array of neighbor lists has a list of "second atoms" 'k' that contribute to its self-energy and the total interaction energy (see Figure 2).

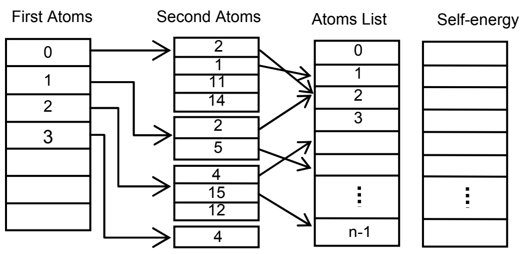

**Figure 2. Array of neighbor lists**

As the positions of the atoms change, the neighbor lists are updated. To compute the self-energies of the atoms, the original FTMap program cycles through the list of first atoms. For each first atom, it traverses the list of second atoms, updating the self-energies of both the first atom as well as the second atom. After the self-energies of all the atoms have been computed, they are accumulated and the total self-energy of the system is obtained. Similarly, the neighbor list array is again traversed to compute the pair-wise interaction energies using the generalized Born equation. Note that the neighbor lists must be traversed twice since the evaluation of pair-wise interaction energy requires that the individual total self-energy of each atom be known. The final step involves computing the gradients for each atom and updating the forces acting on the atoms.

The above computations can be divided into two FPGA pipelines, one for computing self-energies and the other for computing pair-wise interactions. These pipelines can process atom-pairs in streaming fashion, generating and updating two energy values per cycle.

| | Atom index | | Atom Type | |
|---|---|---|---|---|
| Pair # | Atom 1 | Atom 2 | Atom 1 Type | Atom 2 Type |
| 0 | 0 | 2 | T5 | T1 |
| 1 | 0 | 1 | T5 | T3 |
| 2 | 0 | 11 | T5 | T2 |
| 3 | 0 | 14 | T5 | T4 |
| 4 | 1 | 2 | T3 | T1 |
| 5 | 1 | 5 | T3 | T3 |
| 6 | 2 | 4 | T1 | T8 |
| 7 | 2 | 15 | T1 | T7 |
| 8 | 2 | 12 | T1 | T4 |
| 9 | 3 | 4 | T5 | T8 |

**Figure 3. Atom-pairs list**

We now examine the basic data structure and show how it is modified for the FPGA. As shown in Figure 2, a neighbor list essentially represents a pointer to a list of atoms. To better map this data structure to the FPGA's block RAMs, we replace the two dimensional structure of the neighbor lists with a one dimensional pairs list (see Figure 3). The pairs list, as the name suggests, is a list of atom-pairs, containing the indices of the two atoms involved in the pair and the corresponding atom types. On the FPGA, a fetch unit reads a pair of indices in each cycle and fetches the coordinates of the atoms involved. It also reads the atom types and fetches the corresponding parameters.

## 4. ELECTROSTATIC ENERGY EVALUATION ON THE FPGA

### 4.1 Basic Design

As stated earlier, we divide the evaluation of electrostatics energy into two FPGA pipelines – one for computing self-energies of the atoms (the Eself pipeline) and a second for computing the pair-wise interactions (the GB pipeline). Since the evaluation of pair-wise interactions requires the individual total self-energy of each atom, the self-energy computations must be finished before evaluation of pair-wise interactions can begin. This leads to two possible solutions. The first is to utilize the entire FPGA for self-energy pipelines and then reconfigure the FPGA into pair-wise interaction pipelines. The second scheme involves having both the pipelines present on the FPGA at the same time, but starting the pair-wise interaction pipeline only after the self-energy pipeline has finished evaluating energies of all the atoms. This results, however, in part of the FPGA resources remaining idle at all times.

Though the first scheme yields better utilization of FPGA resources and allows for more replication of pipeline instances for parallel evaluations, it is not likely to be the preferred solution. Since the time per iteration for the energy evaluation is relatively small (a few milliseconds on serial computer and a few microseconds on an FPGA), the time required for reconfiguring the FPGA for each iteration will dominate the overall compute time and nullify any performance benefits obtained by better utilization of resources. Moreover, having multiple pipelines in parallel would add to the complexity of the data path and require replication of various block RAMs for independent accesses by different pipelines. This in turn would require broadcasting to multiple BRAMs as well as combining the partial values from those BRAMs, further adding to the complexity of the system.

Thus in the current design both pipelines are present on the FPGA throughout the computation. The GB pipeline starts only after the Eself pipeline has finished processing all the atoms.

## 4.2 The Self-energy Pipeline

Figure 4 shows the FPGA design for computing the self-energies of the atoms. The self-energy pipeline computes the atom self-energies along with some other quantities for updating the forces acting on the atoms.

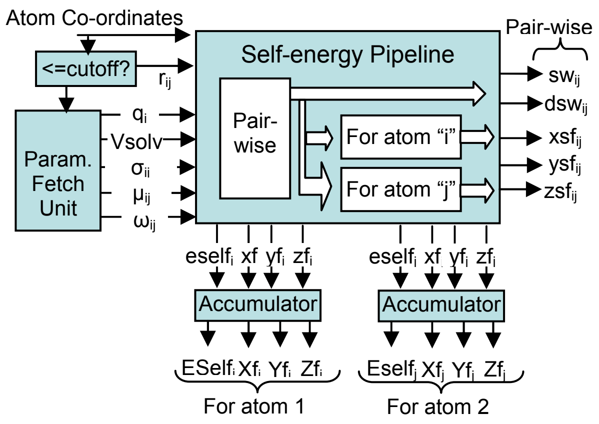

**Figure 4. Self-energy Pipeline**

The self-energy of an atom is the sum of the contributions due to all the neighboring atoms within a cutoff. At the head of the pipeline, a cut-off filter unit checks to see if the two atoms are within a cut-off distance. If so, it notifies the parameter fetch unit so that it can fetch the remaining parameters for the two atoms and pass them to the pipeline.

In each cycle, the pipeline accepts one pair of atoms and computes two sets of quantities: pair-wise values and atom-wise values (see Figure 4). Pair-wise values are those which are for a particular atom-pair and include a switching function [10] and its derivative (sw and dsw), and pair-wise force functions along each of the three axes (xsf, ysf and zsf). The switching function is used to represent the effect of distance dependent dielectric in the electrostatic energy expression. If the distance between the two atoms is larger than the cutoff, the switching function takes a value of 0, reducing the electrostatic energy between the atom-pair to zero. Similarly, the pair-wise force function along an axis depends on the switching function and its derivative as well as the difference of the atom-coordinates along that axis. The resulting forces acting on the two atoms along that axis are equal in magnitude and opposite in direction. The expressions for these various pair-wise quantities and their derivations are shown in Equations 6 – 14.

Atom-wise values are for each atom in the pair and include partial self-energy of the atom ($E_{ik}^{self}$ in Equation 3) and partial force-functions along the three axes (xf, yf, and zf) used to update the forces acting on the atom. The expressions for these atom-wise quantities, as evaluated by the self-energy pipeline, are shown in Equations 11 and 14. These partial values must be accumulated to obtain the total value for each atom. As shown in Figure 4, the self-energies and the force functions of the two atoms computed by the Eself pipeline are accumulated by the accumulators. The accumulated values are stored in block RAMs for use by the GB pipeline. Pair-wise quantities such as switching functions and force functions are also stored in block RAMs and used by the GB pipeline. The block RAM requirements for the design are discussed in Section 5.

$$sw = const\_3(nb_{cutoff}{}^2 - r^2)^2(nb_{cutoff}{}^2 - r^2 - 3(nb_{\cot}{}^2 - r^2)) \quad (6)$$

$$dsw = const\_12(nb_{cutoff}{}^2 - r^2)(nb_{\cot}{}^2 - r^2) \quad (7)$$

$$\exp = \omega_{ik} e^{-\left( r_{ik}^2 / \sigma_{ik}^2 \right)} \quad (8) \qquad T1 = \frac{\widetilde{V}_k}{8\pi}\left( \frac{r_{ik}^3}{r_{ik}^4 + \mu_{ik}^4} \right)^4 \quad (9)$$

$$T\,2 = 2\,sw\,(\exp + T\,1) \quad (10)$$

$$E^{self} = E^{self} - T2 \quad (11)$$

$$T3 = sw\left( -8T1\frac{(3\mu_{ik}^4 - r_{ik}^4)}{r_{ik}^2(r_{ik}^4 + \mu_{ik}^4)} + \frac{4\exp}{\sigma_{ik}^2} \right) - T2.dsw \quad (12)$$

$$xsf = T3(x1 - x2) \quad xsf = T3(y1 - y2) \quad zsf = T3(z1 - z2) \quad (13)$$

$$xf = xf - xsf \qquad yf = yf - ysf \qquad zf = zf - zsf \quad (14)$$

The computation of the electrostatic energy is generally performed with double precision floating point. On our FPGA Eself pipeline, we currently use single precision floating point arithmetic, with apparently little loss in accuracy when used in FTMap (see Section 6.2). The floating point operations of the pipeline are optimized using the Floating Point Compiler from Altera [21]. It generates a 105 stage pipeline running at 185 MHz.

### 4.3 The Generalized Born Pipeline

The Generalized Born pipeline is used to compute the pair-wise interactions between atom-pairs and to update the forces acting on them. As in the case of the self-energy pipeline, only those pairs whose distance is less than the cutoff need to be processed.

The Generalized Born pipeline uses atom self-energies generated by the Eself pipeline to compute the Born radius and its derivative associated with each atom (Equations 15 and 16). These in turn are used to compute the pair-wise interaction energies (Born and Coulomb energies) of the atoms (EBorn and ECoul in Figure 5). The expressions for ECoul and EBorn are given by the first and the second terms of the GB equation (Equation 5). The Born and Coulomb electrostatic energies of different atoms are then accumulated to generate the total electrostatic energy of the complex. Note that unlike self-energies, we do not need the individual total Born and Coulomb energies of each atom. Thus, the partial energy values generated by the GB pipeline are accumulated into a single total value.

$$rBorn_i = \begin{cases} \dfrac{1}{E^{Self_i}} & \text{if} \quad E^{Self_i} \geq {}^{1}\!/\!_{b_0} \\ b_0(2 - b_0.E^{Self_i}) & \text{otherwise} \end{cases} \qquad (15)$$

$$dBrdes_i = \begin{cases} \dfrac{rBorn_i}{k.E^{Self_i}} & \text{if} \quad E^{Self_i} \geq {}^{1}\!/\!_{b_0} \\ \dfrac{b_0^2}{k} & \text{otherwise} \end{cases} \qquad (16)$$

($b_0$ and $k$ are constants)

The GB pipeline also computes partial forces acting on each atom. These are computed using the pair-wise and atom-wise force functions generated by the self-energy pipeline. These partial forces must be accumulated to compute the total forces acting on each atom, which are then used to update the atom coordinates. To update the positions of the atoms, we need the individual forces acting on each atom; the accumulator thus stores the individual values into force block RAMs. The total force values, along with the energy gradients, are then used to determine the new atom positions in the complex for the next minimization iteration.

Like the Eself pipeline, the GB pipeline also uses single precision floating point arithmetic and is generated using the Altera Floating Point Compiler. The compiler generates a 125 stage pipeline running at 190 MHz. It requires about 18% of the resources of a high end Stratix III FPGA (SE110).

## 5. INTEGRATING THE PIPELINES

Since the evaluations of different atom-pairs are independent of each other, having multiple pipelines that process different pairs in parallel would yield better performance. Synthesis results indicate that we can fit two instances each of the Eself and GB pipelines on a Stratix III SE110 FPGA, allowing four energy updates per cycle. Having multiple pipelines, however, results in some added complexity in block RAM accesses and data storage.

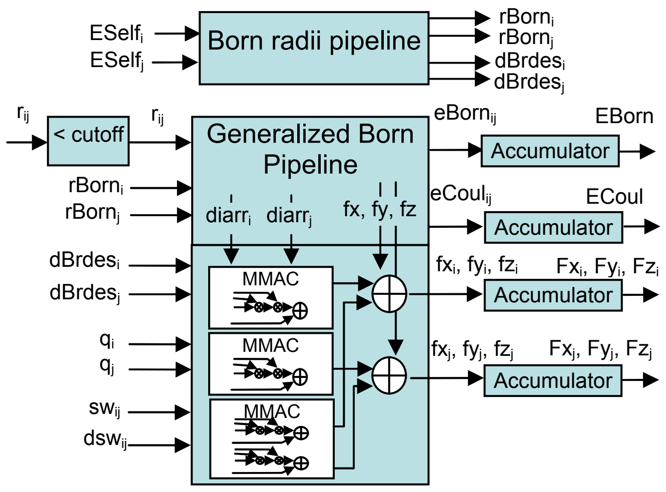

**Figure 5. Generalied Born Pipeline**

Before addressing the issues associated with block RAM accesses, we briefly discuss the issue of feeding one "valid" atom-pair to each of the pipeline instances. As stated earlier, only those atom pairs that pass the cutoff distance test need to be processed by the Eself and GB pipelines. Thus, in order to provide the pipelines with one valid atom-pair per cycle, we need multiple cutoff filter units for each energy pipeline. This results in two complications. First, each cutoff filter requires two sets of atom-coordinates per cycle. These can be read from a single dual-ported block RAM. For multiple filters, we thus need to replicate the BRAMs, one for each filter. Second, there are multiple filters and each filter can potentially output a valid atom-pair each cycle. Since the energy pipeline can only process one of those per cycle, we need to add a mechanism for storing the rest in FIFOs and possibly stalling the pipeline if the FIFOs get filled up.

In order to avoid these pipeline complications, one possible solution is to process all of the atom-pairs present in the list without filtering with respect to the cutoff distance. Note that during the initial creation of the neighbor-lists on the host, a cut-off test is performed. Only those atom-pairs which are within a cutoff are added to the list. A second cutoff test is required on the FPGA to check for atoms that might have moved out of the cutoff radius due to position updates during

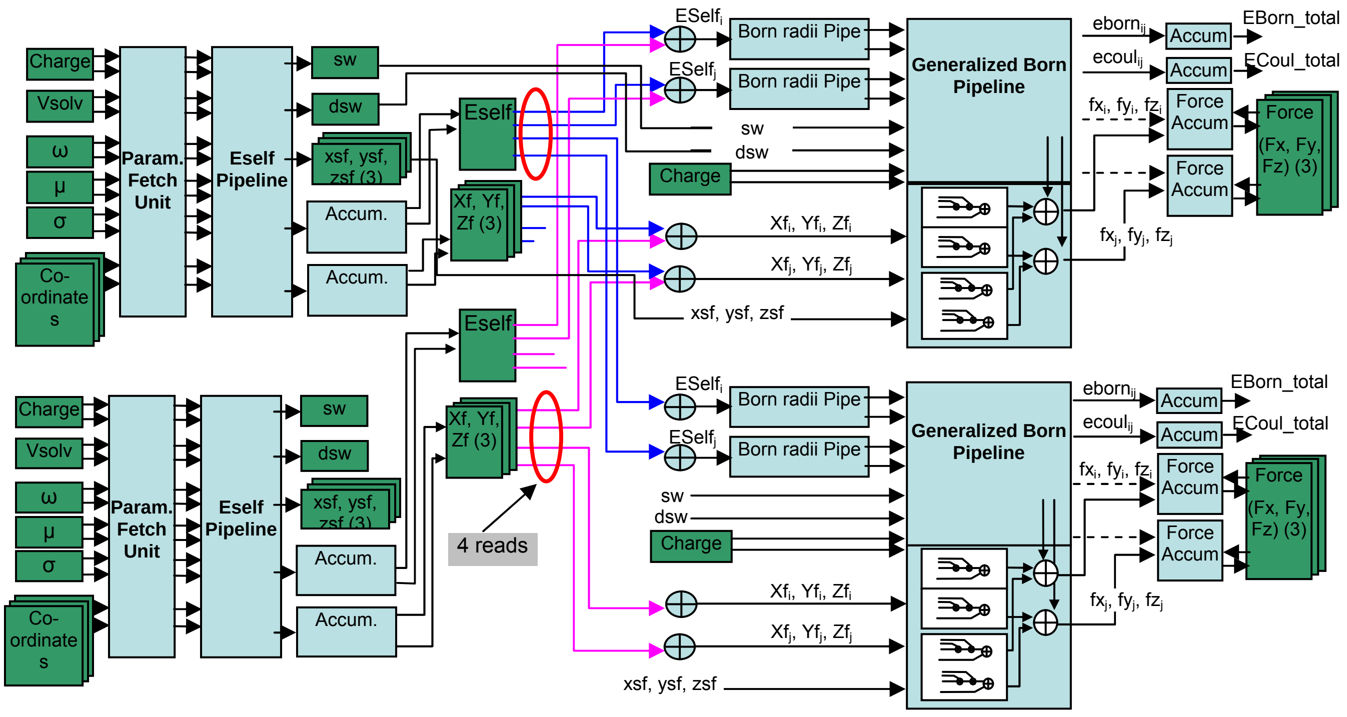

**Figure 6. Complete Electrostatic pipeline with two Eself and GB pipelines**

minimization iterations, and to check for short-range cutoff.

The change in position during minimization steps, however, is not radical [10]. Due to this, most of the atoms will likely pass the cutoff test on the FPGA. We noticed that, on average, only about 7-8% of total pairs fail the test. Moreover, since the total number of atom-pairs to be processed is only a few thousand (unlike molecular dynamics where millions of pairs are screened) unconditionally processing all the atoms does not lead to a significant performance loss. Thus we decided to unconditionally process every atom-pair. A cut-off filter is still needed at the head of the energy pipeline to invalidate the energy and force updates at the end of the pipeline if the atom-atom distance is larger than the cutoff. This can be viewed as inserting a bubble into the pipeline whenever the distance is larger than the cutoff.

Figure 6 shows the complete electrostatic energy pipeline with two instances each of the self-energy and pair-wise interaction pipelines. Also shown are the different block RAMs required in the design (dark green boxes). Two atom pairs are fetched from the pairs-list (of Figure 2) in each cycle and one pair is fed into each of the pipeline instances. As shown, the block RAMs for the atom coordinates and the other atom-wise and pair-wise parameters (atom charge, solvation volume, ω, μ, σ) are replicated to feed the two pipelines. The pair-wise parameters (ω, μ, σ) are not symmetric and hence two sets of these are required per cycle. This is done by utilizing the dual-ported capability of the block RAMs. Similarly, atom charges and coordinates are replicated into two sets of dual-ported block RAMs, one for independently feeding each of the Eself pipelines.

Each of the Eself pipeline generates and stores self-energy and force function values (Eself, Xf, Yf, Zf) into its own set of block RAMs. An independent set of BRAMs is required for each pipeline since each pipeline performs two updates per cycle and FPGA block RAMs provide only two read/write ports. Thus the values in the block RAMs of the two pipelines are partial and must be combined before being fed into the GB pipeline. As shown in Figure 6, this can be done on the fly. Partial values from the two BRAMs are fetched and added and the sum is then fed into the GB pipeline.

Each GB pipeline requires two atom-wise quantities (Eself, Xf, Yf, Zf) per cycle, each of which is a sum of values from the two sets of BRAMs. Thus, each GB pipeline needs to access two locations in each BRAM. This results in 4 reads per cycle for each of the BRAMs (as shown in Figure 6). Since four read ports are not supported by FPGA block RAMs, we implement this by further replicating each of the block RAMs into two banks (see Figure 7). Each RAM requiring four read ports is implemented by instantiating two banks of dual ported RAMs. Writes are broadcast to both the banks and two reads are provided by each of the banks.

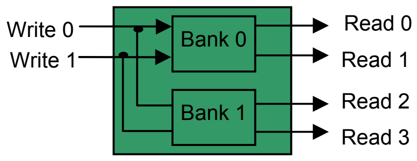

**Figure 7. 4-port RAM using 2-port banks**

For the pair-wise quantities generated by the two Eself pipelines (sw, dsw, xsf, ysf, zsf), we could use a unified BRAM since each pipeline updates only one set of values per cycle. As shown in Figure 6, we still utilize independent sets of BRAMs for each pipeline. This is done due to two reasons. First, this allows the use of simple dual-ported block RAMs. Second, even though the pair-wise quantities are stored by the two pipelines in separate BRAMs, they need not be combined before being fed to the GB pipeline. This is because these values are specific to a particular atom-pair and the distribution of pairs among the pipelines is the same in both the Eself and GB pipelines. That is, if a particular pair is processed by the first Eself pipeline, it will always be processed by the first GB pipeline. Finally, the total forces generated by the GB pipeline are stored in dual-ported block RAMs. Again, each GB pipeline has its own set of independent force BRAMs and updates two force values per cycle. These two sets of BRAMs are then combined to generate the total forces which are used to update atom positions for the next iteration.

## 6. RESULTS

### 6.1 Performance Results

Table 1 shows the resource utilizations for the self-energy and pair-wise interaction pipelines. As shown, utilization for both of the pipelines is dominated by the block multipliers. On the Stratix III SE110 FPGA, we can fit two instances of each of the pipelines, operating at 185 MHz with single precision floating point precision.

Table 2 shows the estimated speedup on FPGA for evaluation of one iteration of electrostatic energy for a protein-probe complex. The complex contains 2260 atoms and requires 9780 atom-atom evaluations per iteration. Also shown are results from our earlier work on energy minimization on GPUs [22]. As shown, the FPGA implementation results in an estimated speedup of two orders of magnitude over a single core of a quad core Intel Xeon processor (2.00 GHz). This represents an 8x improvement over our previous work on acceleration of energy minimization using NVIDIA Tesla C1060 GPU (with 240 processor cores and running at 1.3GHz).

**Table 1. Resource Utilizations on Stratix III SE110**

| *Resource* | *Self-energy Pipeline* | *Pair-wise Interaction Pipeline* | *Total with two of each pipelines* |
|---|---|---|---|
| Combinational ALUTs | 13,927 (17%) | 8,913 (11%) | 45,680 (54%) |
| Memory ALUTs | 1543 (4%) | 1126 (3%) | 5338 (13%) |
| Register Bits | 19,161 (22%) | 12,157 (14%) | 62,636 (73%) |
| Block Multipliers | 264 (29%) | 162 (18%) | 852 (95%) |
| M9K BRAMs | 104 (16%) | | |
| M144K BRAMs | 14 (88%) | | |

**Table 2. Estimated FPGA Speedups**

| *Computation* | *Serial Time* | *FPGA Time* | *GPU Time* | *Speedup over Serial* | *Speedup over GPU* |
|---|---|---|---|---|---|
| Self-energy | 6.15 ms | 0.027 ms | 0.22 ms | 225x | 8x |
| Pair-wise Interaction | 2.75 ms | 0.027 ms | 0.23 ms | 100x | 8.5x |

### 6.2 Error Analysis

As stated earlier, the original FTMap program uses double precision floating point to represent the energy and force values. In our FPGA design, we currently use single precision floating point arithmetic. To study the effects of the reduced precision on convergence, we performed a preliminary experiment on a protein-probe complex. We performed minimization using both single and double precision and compared the final coordinates of the atoms in the complex after minimization. Note that the goal of the FTMap program is to obtain the probe orientation in the protein that results in the least energy conformation. By comparing the coordinates of the atoms, we effectively compare the final orientation of the probe in the protein. The atom coordinates in FTMap are represented with a resolution of 1/1000th of an Angstrom. This is the same as required for the atom-coordinates in the Protein Data Bank (pdb) file format [23]. Our experiment indicates that out of 2260 atoms in the complex, the coordinates of 2256 atoms match precisely between the single and double precision. Of the remaining 4, the coordinates of 3 of the atoms match within 1/100th of an Angstrom, with the fourth one matching within 1/10th of an Angstrom.

Though this experiment is preliminary, it gives an indication that single precision floating point may suffice for computations in energy minimization, at least as used in mapping. We are currently performing more experiments for precision and error analysis using both single precision as well as mixed and other hybrid precision schemes.

## 7. CONCLUSION

We have presented FPGA pipelines for the electrostatic energy computation as used in energy minimization applications. The design achieves a two order-of-magnitude speedup over a single CPU core for computations that constitute more than 90% of total serial runtime. The overall speedup, however, is much modest and is currently limited to 14x. This is because the evaluation of the van der Waals energy constitutes around 5.5% of total runtime on host and has not yet been accelerated. Computations involved in evaluating the van der Waals energy are similar to those for evaluating the electrostatics energy and can benefit from FPGA acceleration. Acceleration of van der Waals evaluation would result in higher overall speedup and is now in progress.

We compared our results with our previous work of mapping the energy computation to GPUs and showed that FPGA implementation achieves an 8x improvement over the GPU version. In spite of the huge floating point capabilities of modern GPUs, the speedup on GPU is comparatively modest. This is mainly due to the limitations associated with its fixed architecture. In contrast, FPGAs provide immense flexibility both with respect to selection of the computational cores and the data communication among them. In addition, modern FPGAs boast of high floating point capabilities, with latest chips achieving up to 200 GFLOPs peak for single precision floating point. These features, combined with deeply pipelined designs and high utilization, result in very high performance.

We also performed preliminary error analysis by comparing the coordinates of the atoms in a protein-probe complex after minimization, using double and single precision floating point arithmetic. Our experiment shows that the final atom-coordinates obtained using single precision are in good agreement with the results obtained using original FTMap program that uses double precision floating point arithmetic.

## 8. REFERENCES

[1] Brenke, R. et al. (2009) Fragment-based identification of druggable ‘hot spots’ of proteins using Fourier domain correlation techniques. *Bioinformatics*, 25(5), 621-627.

[2] Meng, E. C., Gschwend, D. C., Blaney, J. M. and Kuntz, I. D. (1993) Orientational sampling and rigid-body minimization in molecular docking. *Proteins*, 17, 266–278.

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

[4] Li, L., Chen, R. and Weng, Z. (2003) RDOCK: Refinement of Rigid-body Protein Docking Predictions. *Proteins*, 53, 693-707.

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

[6] Eisen, M. B., Wiley, D. C., Karplus, M. and Hubbard , R. E. (1994) HOOK: A program for finding novel molecular architectures that satisfy the chemical and steric requirements of a macromolecule binding site. *Proteins: Structure, Function and Genetics*, 19, 199-221.

[7] Miranker, A. and Karplus, M. (1991) Functionality maps of binding sites: a multiple copy simultaneous search method. *Proteins: Structure, Function and Genetics*, 11, 29-34.

[8] Caflish, A., Miranker, A. and Karplus. M. (1993) Multiple copy simultaneous search and construction of ligands in binding sites: application to inhibitors of HIV-1 aspartic proteinase. *J.Med.Chem.*, 36, 2142-2167.

[9] Landon, M., Lancia, D., Yu, J., Thiel, S., Vadja, S. (2007) Identification of Hot Spots within Druggable Binding Regions by Computational Solvent Mapping of Proteins. *J. Med. Chem.*, 50, 1231–1240.

[10] Brooks, B.R. et al. (1983) CHARMM: a program for macromolecular energy, minimization, and dynamics calculations. *J. Comp. Chem.*, 4, 187–217.

[11] Schaefer, M. and Karplus, M. (1996) A Comprehensive Analytical Treatment of Continuum Electrostatics. *J. Phys. Chem.*, 100 (5), 1578-1599.

[12] Constanciel, R., and Contreras, R. (1984) Self consistent field theory of solvent effects representation by continuum models: Introduction of desolvation contribution. *Theoret. Chim. Acta (Berl.)*, 65, 1-11.

[13] http://en.wikipedia.org/wiki/Energy_minimization.

[14] http://www.wag.caltech.edu/publications/theses/alan/subsectional_4_0_2_1.html.

[15] Cornell, W. D. et al. (1995) A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules. *J. Am. Chem. Soc.* 117, 5179-5197.

[16] Pappu, R.V., Hart, R. K., and Ponder, J. W. (1998) Analysis and Application of Potential Energy Smoothing and Search Methods for Global Optimization. *J. Phys. Chem. B*, 102, 9725–9742.

[17] Ershov, R. E. (1970) Self-energy of a "smeared" charge. *Russian Physics Journal*, 13 (6), 813-813.

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

[19] Born, M. Z. (1920) *Physics*, 1, 45.

[20] Still, W. C., Tempczyk, A., Hawley, R. C., Hendrickson, T. (1990) Semianalytical treatment of solvation for molecular mechanics and dynamics. *J. Am. Chem. Soc.*, 112 (16), 6127-6129.

[21] Langhammer, M. Floating point datapath synthesis for FPGAs. In Proceedings of *IEEE Conference on Field Programmable Logic and Applications* (2008), pp 355-360.

[22] Sukhwani, B. and Herbordt, M. C. (2009) Accelerating Free-Energy Minimization using Graphics Processors. In Proceedings of *Symposium on Application Accelerators in High Performance Computing*.

[23] http://deposit.rcsb.org/adit/docs/pdb_atom_format.html