Preliminary Investigation of Advanced Electrostatics in Molecular Dynamics on Reconfigurable Computers

Ronald Scrofano
Department of Computer Science
University of Southern California
Los Angeles, CA
rscrofan@halcyon.usc.edu

Viktor K. Prasanna
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA
prasanna@halcyon.usc.edu

Abstract

Scientific computing is marked by applications with very high performance demands. As technology has improved, reconfigurable hardware has become a viable platform to provide application acceleration, even for floating-point-intensive scientific applications. Now, reconfigurable computers—computers with general purpose microprocessors, reconfigurable hardware, memory, and high performance interconnect—are emerging as platforms that allow complete applications to be partitioned into parts that execute in software and parts that are accelerated in hardware. In this paper, we study molecular dynamics simulation. Specifically, we study the use of the smooth particle mesh Ewald technique in a molecular dynamics simulation program that takes advantage of the hardware acceleration capabilities of a reconfigurable computer. We demonstrate a 2.7–2.9× speed-up over the corresponding software-only simulation program. Along the way, we note design issues and techniques related to the use of reconfigurable computers for scientific computing in general.

Keywords: Reconfigurable, FPGA, molecular dynamics, electrostatics

1 Introduction

Reconfigurable computers—computers with general purpose microprocessors (GPPs), reconfigurable hardware accelerators, memory, and high performance interconnect—have emerged as a platform for accelerating scientific computing applications. The state-of-the-art reconfigurable hardware for reconfigurable computers is field-programmable gate arrays (FPGAs). These devices can be programmed and reprogrammed with application-specific hardware designs. Until recently, FPGAs did not contain enough logic resources to implement floating-point arithmetic and were, therefore, not amenable to many scientific computing applications. Instead, they were employed with great effectiveness in fields such as signal processing, cryptography, and embedded computing. Now, large FPGAs that also possess hardware features such as dedicated multipliers and on-chip memories have become attractive platforms for accelerating kernels in scientific applications.

Scientific computing applications, however, are usually composed of a multitude of kernels. Some of these kernels may be very well-suited for execution in software. This is where reconfigurable computers can be employed: the tasks well-suited for software execution can execute on the GPPs and the tasks that are amenable to hardware acceleration can execute on the FPGAs. So far, much of the research has focused on accelerating individual scientific kernels [Underwood 2004; Hemmert and Underwood 2005; Smith et al. 2005b]. However, as will be discussed later in the paper, a careful partitioning of kernels between software and hardware is necessary to achieve performance [Gokhale et al. 2006]. That is, the focus must be on the complete application.

One particularly interesting scientific computing application to study is molecular dynamics (MD) simulation, which is a useful technique for simulating the movements of atoms in a system over time. It is applied in a wide range fields, including materials science, pharmaceuticals, and nanotechnology [Nakano et al. 2001; Tang and Xu 2002; Mao et al. 1999]. Because of its computational demands, MD simulation is most often performed on large supercomputers and clusters of commodity general purpose processors (GPPs). Recent studies into the reconfigurable-hardware acceleration of the complete MD simulation application, as opposed to its individual kernels, have mainly focused on fairly simple simulations that employ cutoff approximations

SC2006 November 2006, Tampa, Florida, USA
0-7695-2700-0/06 $20.00 ©2006 IEEE

for electrostatics calculations, if they perform them at all [Azizi et al. 2004; Gu et al. 2005; Scrofano et al. 2006]. In this paper, we investigate the use of a more advanced electrostatics calculation technique—the smooth particle mesh Ewald (SPME) technique—on a reconfigurable computer implementation of MD simulation.

In the next two subsections, we provide further introduction to FPGAs and reconfigurable computers and to MD simulations. In Section 2, we describe some related efforts in acceleration of MD simulations, including our prior work. Section 3 focuses on implementing a complete MD simulation that utilizes the SPME technique for electrostatics calculations. It presents implementation details and performance results. Section 4 presents a discussion of the results. Section 5 concludes the work and notes some areas for future work.

1.1 FPGAs and Reconfigurable Computers

Reconfigurable hardware is used to bridge the gap between flexible but performance-constrained GPPs and very high-performance but inflexible application-specific integrated circuits (ASICs). Reconfigurable hardware is programmed and reprogrammed after fabrication. Thus, using reconfigurable hardware is much less expensive and much more flexible than using ASICs is. Because reconfigurable hardware is programmed for the problem to be solved rather than to be able to solve all problems, it can achieve higher performance and greater efficiency than GPPs, especially in applications with a regular structure and/or a great deal of parallelism. The state-of-the-art devices used as reconfigurable hardware are FPGAs [Gokhale and Graham 2005]. These devices are implemented as a matrix of programmable logic cells and programmable interconnect between the cells. Modern FPGAs often have special-purpose logic such as memories (we refer to these as "on-chip memories" throughout the paper) and multipliers, embedded into the logic fabric. Increased logic density as well as improved programming environments have made FPGAs a promising technology to be exploited by the computer science and computer engineering communities.

The drawback to using FPGAs is that they have low clock frequencies—typically in the 100 MHz to 200 MHz range—when compared to GPPs. To overcome this clock frequency disadvantage, parallelism and pipelining are employed. Parallelism comes in two forms: fine-grained and coarse grained. Fine-grained parallelism involves independent operations that are involved in producing the same result and is analogous to instruction-level parallelism in computing with GPPs. Coarse-grained parallelism involves units operating independently and producing independent results. Parallelism in a design is limited by the amount of logic on the FPGA and the amount of bandwidth into the FPGA.

Pipelining is particularly important in designs employing floating-point arithmetic. Floating-point cores, especially those that conform to IEEE standard 754, are very complex and require a great deal of logic resources to implement. In order to achieve high clock frequencies, floating-point cores for FPGAs must be deeply pipelined: for example, double-precision adders and multipliers have about 10 to 20 pipeline stages while dividers and square rooters have about 40 to 60 stages [Govindu et al. 2005]. The main difficulty with deeply pipelined units—for floating-point arithmetic or otherwise—is that their use can lead to data hazards, which may cause the pipeline to stall.

1.1.1 Reconfigurable Computers

Reconfigurable computers, such as the SRC 6 MAPstation [SRC 2004], the Cray XD1 [Cra 2005], and the SGI RASC [Sil 2006] bring together GPPs and FPGAs, connecting them with a high-performance interconnect network (see Figure 1). Typically, the reconfigurable hardware has local memory banks that are on-board but off-chip. These on-board memories are typically much smaller than the RAM used by GPPs, but also have a lower access latency, in terms of cycles. Additionally, this local memory is usually divided into equally-sized banks, where the banks can be accessed in parallel.

The target architecture in this paper is the SRC 6 MAPstation. This reconfigurable computer has two 2.8 GHZ Intel Xeon microprocessors and one *MAP processor*. The MAP processor contains two Xilinx XC2V6000 FPGAs and eight banks of on-board memory. Each of these banks is 64 bits wide and can be accessed independently of the others. Six of the banks hold 4 MB of data and the other two hold 2 MB of data. Data is transferred between the GPPs and the on-board memories by DMA at a rate of 1.4 GB/s in each direction. The designs described in this paper will only make use of one of the GPPs and one of the FPGAs in the MAPstation.

The programming model employed is that which we term the *accelerated single-program, multiple data* (ASPMD) model. A high-performance reconfigurable computer or a cluster of reconfigurable computers would contain many nodes, each with processor and FPGA. Each node in the system executes the same program on different data and communicates with other nodes when data needs to be exchanged. At the node

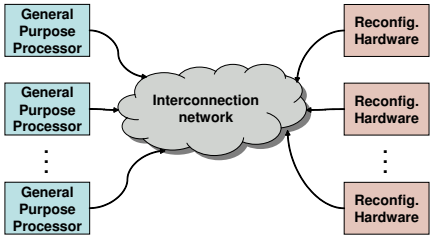

Figure 1: Model of the System Architecture

level, the application mainly runs on the GPP resource. The GPP resource uses the RH resource as an accelerator for intensive parts of the code. In this paper, we focus only on a single node.

1.2 Molecular Dynamics Background

MD is a widely used technique to simulate the trajectory of an atomic system over time. For this introduction, we mainly reference [Allen and Tildesley 1987]. A system of atoms is placed inside a simulation box. In most simulations, the system is considered to be periodic: it is assumed that the central simulation box is replicated in each direction out to infinity. Given the initial conditions of the system at time $t = 0$, the motion of the atoms in the system until time $t = t_f$ is simulated by advancing the simulation by small discrete time steps Δt. The amount of time simulated is usually on the order of nanoseconds or microseconds and the time step is usually on the order of femtoseconds. The new positions of the atoms are calculated by integrating the classical equations of motion. One popular integrator is the velocity Verlet algorithm, shown below, where $\vec{r}_i(t)$, $\vec{v}_i(t)$, and $\vec{a}_i(t)$ are the position, velocity, and acceleration of atom i at time t, respectively.

1. calculate $\vec{v}_i\left(t+\frac{\Delta t}{2}\right)$ based on $\vec{v}_i(t)$ and acceleration $\vec{a}_i(t)$;
2. calculate $\vec{r}_i(t+\Delta t)$ based on $\vec{v}_i\left(t+\frac{\Delta t}{2}\right)$;
3. calculate $\vec{a}_i(t+\Delta t)$ based on $\vec{r}_i(t+\Delta t)$ and $\vec{r}_j(t+\Delta t)$ for all atoms $j \neq i$;
4. calculate $\vec{v}_i(t+\Delta t)$ based on $\vec{a}_i(t+\Delta t)$.

It is well known that finding the acceleration on each atom (step 3) is the most time-consuming step of the simulation. Finding the acceleration requires finding the forces acting upon each atom.

1.2.1 Force Calculation

The forces calculated in MD simulations can be broken into two types: bonded and nonbonded. Bonded forces only act between atoms that are in bond groups. This is thus an $O(n)$ calculation, where n is the number of atoms in the simulation, and not usually a bottleneck.

Nonbonded forces, on the other hand, are pairwise forces that can act between any atoms in the system that are not in the same bond group. Nonbonded force calculation can further be divided into short-range and long range forces. Short range forces drop off quickly as the distance between atoms rises. For these forces, a cutoff distance is employed: if the distance between two atoms is greater than some distance r_c (often about 10 Å in practice), the atoms are assumed not to interact. Finding pairs of atoms that interact is naively an $O(n^2)$ operation. However, many techniques exist to reduce this complexity. In our work, we employ the neighbor list technique, in which every s steps, where s is user defined, a list of atoms is created and a given atom may only interact with those atoms on its list.

The force due to the Lennard-Jones potential (hereafter referred to as the Lennard-Jones force) is a common short range force. Equations 1 and 2 show the equations for the Lennard-Jones potential and force, respectively, where $\vec{r}_{ij}$ is the distance vector between atoms i and j, r_{ij} is the distance between atoms i and j, and A, B, C, and D are constants. Constants C and D are necessary to compensate for the use of the cutoff distance. This approach is called the *shifted-force* technique. Note that whenever we refer to "force calculation," in the context of the simulation, the potential energy is also calculated at that step.

$$U_{ij}^{LJ} = \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} + C_{ij} - D_{ij} r_{ij} \tag{1}$$

$$\vec{f}_{ij}^{LJ} = \left(\frac{6}{r_{ij}^{2}} \left(\frac{2A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \right) + \frac{D_{ij}}{r_{ij}} \right) \vec{r}_{ij} \tag{2}$$

Long range forces present more of a problem. They do not drop off quickly at long distances. Not only can atoms interact with any other atoms in the system, they can also interact with their images in other boxes. One solution is to use a cutoff scheme, just like for the short range forces. The long-range force used most often in MD simulations is the electrostatic or Coulomb force. A shifted-force approach can be used for the Coulomb force and potential. Equations 3 and 4 show the shifted-force Coulomb potential and force, respectively, where q_i is the charge of atom i.

$$U_{ij}^{E} = \frac{q_i q_j}{r_{ij} r_C} (r_C - r_{ij})^2 \quad (3)$$

$$\vec{f}_{ij}^{E} = \frac{q_i q_j}{r_{ij}} \left(\frac{1}{r_{ij}^2} - \frac{1}{r_c^2} \right) \vec{r}_{ij} \quad (4)$$

While simple, this technique unfortunately can lead to artifacts in the results of the simulation [Patra et al. 2003]. A more accurate technique is the Ewald summation, which breaks the potential energy calculation into a real-space (or direct-space) part, a reciprocal-space part, and a correction term. The real-space and reciprocal-space parts are given in Equations 5 and 6, where β is a convergence parameter that determines how much work is done in the real- and reciprocal-space parts, the $\vec{n}$ are vectors representing images of the central simulation box, and the $\vec{m}$ are reciprocal lattice vectors (see [Essmann et al. 1995]). The asterisk in the summation means to omit the case where $\vec{n} = 0$ and $i = j$. For the correction term, which is not computationally intensive, see [Essmann et al. 1995]. The expression for S is given in Equation 7. Forces are found by taking the negative gradient of these terms. In practice, β is often set so that the real-space part of the calculation can be done out to the same cutoff distance as the short range forces and, thus, the only value of $\vec{n}$ that matters is $\vec{n} = \vec{0}$.

$$U_{real}^{E} = \frac{1}{2} \sum_{\vec{n}}^{*} \sum_{i,j=1}^{\vec{n}} \frac{q_i q_j \mathrm{erfc}(\beta \left| \vec{r}_i - \vec{r}_j + \vec{n} \right|)}{\left| \vec{r}_i - \vec{r}_j + \vec{n} \right|} \quad (5)$$

$$U_{recip}^{E} = \frac{1}{2\pi V} \sum_{\vec{m} \neq 0} \frac{\exp(-\pi^2 \vec{m}^2 / \beta^2)}{\vec{m}^2} S(\vec{m}) S(-\vec{m}) \quad (6)$$

$$S(\vec{m}) = \sum_{i=1}^{n} q_i \exp(2\pi \vec{m} \vec{r}_i \sqrt{-1}) \quad (7)$$

At best, the Ewald summation is an $O(n^{3/2})$ calculation. The particle mesh Ewald technique and its smooth variant (SPME) reduce the complexity to $O(n \log n)$ by facilitating the use of the 3D FFT in the Ewald summation. For an in-depth description, see [Essmann et al. 1995] or [Lee 2005]. The basic idea is to approximate the S term in Equation 6. Instead of considering point charges in space, each charge contributes a certain amount to regularly spaced grid points. The amount of charge contributed is computed through interpolation. SPME uses cardinal B-splines for interpolation. The real-space part and the correction are calculated without any changes. The main steps for calculating the reciprocal-space part are

1. Find B-spline coefficients and their derivatives for each atom
2. Assign charges to grid points
3. 3D FFT the grid
4. Calculate the energy using the transformed grid and other arrays
5. Inverse FFT the product of the grid with other arrays
6. Use the result of the step 5 and the derivatives of the B-spline coefficients to find the forces

1.2.2 Benchmark Simulations

We use two real-word simulations to benchmark the performance of our implementations. One simulation is of palmitic acid in water. The simulation consists of 52558 atoms of 8 different types. The cutoff length is 10 Å, the time step is 2 fs, and the neighbor list is rebuilt once every 10 steps. The simulation box is $104 \times 84.8 \times 256$ Å^3 and the grid used in the SPME method is $128 \times 96 \times 256$. Fourth order interpolation is performed to assign charges to grid points.

The other simulation is of the CheY protein in water. This simulation consists of 32932 atoms of 17 different types. The cutoff length is 10 Å, the time step is 2 fs, and the neighbor list is rebuilt once every 10 steps. The simulation box is $73.8 \times 71.8 \times 76.8$ Å^3 and the grid used in the SPME method is $64 \times 64 \times 64$. Fourth order interpolation is performed to assign charges to grid points.

2 Related Work

In this section, we look at some other efforts to accelerate MD with hardware. One of the most successful of these efforts is the Molecular Dynamics Machine (MDM) [Narumi et al. 2001]. MDM uses two types of hardware accelerators: MD-GRAPE 2 is used to accelerate short-range force calculations and WINE-2 is used to accelerate the reciprocal-space part of the long range force calculations. By using large numbers of these hardware accelerators, MDM can achieve TFLOPS performance. The drawback to using these accelerators is the fact that they are ASICs. ASICs, while providing high performance, have high development costs and, if they are not produced in large quantities, high production costs.

[Lee 2005] is, to our knowledge, the only work implementing the reciprocal-space part of SPME on an

FPGA. The implementation is limited to fixed-point arithmetic and the author studies various fixed-point precisions to provide acceptable results. All tasks of the reciprocal-space part are executed in hardware.

Besides our own work, described in the next subsection, we are aware of three other works studying the acceleration of the complete MD application with FPGAs. In two of them, the entire velocity Verlet algorithm is moved into hardware [Azizi et al. 2004; Gu et al. 2005]. In each of these two works, fixed-point arithmetic and table lookup and interpolation are used to calculate the forces nonbonded forces. Bonded forces are not calculated. Both works achieve an impressive speed-up, but are limited to small simulations that can fit in the on-chip memory of the FPGA and both utilize the $O(n^2)$ technique that does not scale well to find interacting pairs of atoms. The speed-ups reported also come when compared to a slow MD software implementation [Bargiel et al. 1991].

The third work that accelerates MD simulations on a reconfigurable computer accelerates a modified version of the NAMD simulation package [Kindratenko and Pointer 2006]. The authors study various architectural choices and report on their performance. Notably, single-precision floating-point arithmetic and 32-bit integer arithmetic are employed and bonded force calculation is omitted. While NAMD is certainly capable of using SPME in simulations, it is unclear whether or not SPME is actually utilized in the simulations in [Kindratenko and Pointer 2006] and in any case, SPME's impact on performance is not addressed.

2.1 Our Prior Work

In our prior work, we developed an MD simulation program for the SRC 6 MAPstation [Scrofano et al. 2006]. We summarize the results here because the present MD simulation program builds upon this previously developed basic simulation program.

For our study into the acceleration of MD simulations with reconfigurable computers, we wanted to begin with a basic, yet still scientifically useful, simulation program, rather than trying to accelerate existing large-scale simulation packages such as NAMD [Kalé et al. 1999] or GROMACS [van der Spoel et al. 2005]. These large-scale simulation packages are highly optimized and parallelized and contain many features; they present rather extreme cases for studies into the appropriateness of reconfigurable hardware acceleration. At the same time, if the simulation program studied were too simplistic, no useful information would be gained from it.

The MD simulation program we developed thus supports constant energy and volume (constant-NVE) simulation with an orthogonal periodic box. The velocity Verlet integrator is employed. For bonded force calculation, bond stretch, angle bend, and dihedral torsion are supported. For nonbonded force calculation, the shifted force Lennard-Jones and shifted force Coulomb techniques are used. The Verlet neighbor list technique is used to identify the interacting pairs of atoms. The program also now implements the RATTLE algorithm for bond constraints [Allen and Tildesley 1987].

Inputs to the program are provided at run-time, and they include the cutoff distance and list cutoff distance, periodic box dimensions, number of steps in the simulation, and the length of the simulation time step. The structure of the system can be specified in an AMBER-format topology file, which is preprocessed into files in the program's internal format [AMB 2004]. Initial positions and velocities can be specified in a coordinate file in PDB format [PDB 1996], which also gets preprocessed.

We developed both a software version of the program and a hardware accelerated version. In the hardware accelerated version, the nonbonded force calculation is moved to the reconfigurable hardware while the remainder of the simulation runs in software. There is one pipeline that calculates the force and potential. It exploits the fine-grain parallelism within the operations in the force calculation. Due to area and memory bandwidth limitations, only one pipeline fits on the FPGA, so the design cannot take advantage of coarse-grained parallelism. Nonetheless, for each simulation, we obtained about a $2\times$ speed-up for the entire application by doing the hardware acceleration.

There are two main drawbacks to this implementation. The first is that it uses single-precision, rather double-precision, floating-point arithmetic. This limitation is due to limited area in the target FPGA, the limited number of on-board memory banks, and the limited bandwidth between those on-board memory banks and the FPGA. The SPME design described in the next section also has this limitation. This limitation may be overcome with the introduction of new reconfigurable computers, such as the proposed SRC 7 MAPstation, which have larger, more modern FPGAs, more on-board memory, and higher memory bandwidth [SRC 2004].

The other drawback to this basic implementation is the use of the shifted-force approximation for the Coulomb force calculation. In the next section, we describe the introduction of the more accurate SPME technique into the MD simulation program to overcome this problem. We then discuss several issues that present themselves in

any implementation of SPME on a reconfigurable computer and draw some conclusions about what these issues mean for the use of reconfigurable computers in scientific computing in general.

3 Smooth Particle Mesh Ewald on a Reconfigurable Computer

Using SPME does not change the basic structure of the simulation. The velocity Verlet algorithm, the bonded force calculation, and the Lennard-Jones portion of the nonbonded force calculation remain the same. The major changes are in the electrostatic nonbonded force calculation, as described in Section 1.2.1. Comparing the shifted force approximation to the SPME technique, we see that several changes will be required to the program. The real-space part of the potential is similar to the standard Coulomb potential (without the changes for force shifting), except a term with an $\mathrm{erfc}(x)$ operation is introduced (see Equation 5). The correction term is not of major concern as it includes only $O(n)$ calculations. The reciprocal-space calculation, however, is wholly new and requires 3D FFTs, interpolations, and the multiplication of 3D arrays. The introduction of these kernels has the potential to significantly affect the application profile, so we begin by examining the profile of a software-only implementation of MD simulation with SPME.

Following [Essmann et al. 1995] and referencing the relevant functions in NAMD and GROMACS, we developed an implementation of the SPME technique that was integrated into the simulation program described in the last section. The size of the grid and the order of the interpolation is specified by the user at run time. The computation of $\mathrm{erfc}(x)$ and its derivative are implemented with table lookup and interpolation. The FFTs are done using the appropriate functions from the Intel Math Kernel Library, version 8.1.14 [Int 2006].

3.1 Partitioning

The application profile for the two benchmark simulations, obtained using Oprofile 0.9.1 [opr], is shown in Table 1. Clearly, the real-space part of the force calculation, which includes the shifted-force Lennard-Jones force calculation, is taking the most time. In the palmitic acid simulation, the next-most time is taken by the reciprocal-space part of the SPME calculation, which includes the interpolation and both FFTs. Building the neighbor list takes roughly half as much time, and the rest of the tasks even less. So, based on the palmitic acid simulation, it seems that we should focus our efforts on the real- and reciprocal-space parts of force calculation to obtain the highest speed-up.

In the CheY simulation, the construction of the neighbor list takes the second-most time in the simulation, while the reciprocal-space part of the force calculation takes the third-most time. Note that in the CheY simulation, the FFTs take hardly any time. This occurs because the $64 \times 64 \times 64$ grid used by the CheY simulation causes very few cache misses in the GPP's 1 MB cache, leading to very high FFT performance. In general this will not be the case and the FFTs and, consequently, the reciprocal-space part of force calculation, will take a higher percentage of the time. Thus, we focus our acceleration study on the real- and reciprocal space parts of the force calculation and leave the remaining tasks, including construction of the neighbor list, in software.

As mentioned above, the real-space part of the calculation is very similar to the calculation in the shifted-force approach, with the introduction of an $\mathrm{erfc}(x)$ computation and an e^{-x^2} computation as part of the derivative of $\mathrm{erfc}(x)$. These two terms present a challenge for hardware implementation because there are no existing floating-point cores to calculate them. However, they can be calculated through table lookup and interpolation and we describe such an implementation in Section 3.2.1. Thus, without too much change, we can accelerate the real-space part in a similar fashion to the way the shifted force approach is accelerated in [Scrofano et al. 2006]. We will obtain similar performance as well, which will be a significant speed-up over the software-only implementation of the real-space part. For example, the real-space part in the palmitic acid simulation should take only about 0.34 s/step, a $4.6\times$ speed-up over the software-only version [Scrofano et al. 2006]. Similarly, the real-space part of the CheY simulation should take only about 0.19 s/step, a $4.8\times$ speed-up over the software-only version. Clearly, this task should be executed in hardware.

For the reciprocal-space part of force calculation, one option is to not accelerate it and leave it in software with the rest of the simulation. Because they are not dependent upon one another, the two parts can execute in parallel, one on the GPP and the other on the FPGA. In this case, the length of time for the two tasks to complete is equal to the maximum of the time the reciprocal-space part takes to execute in software and the real-space part takes to execute in hardware. For the palmitic acid simulation, we estimate that the two parts of the simulation will take about the same time. For the CheY simulation, the real-space part of the simulation will still be longer

Table 1: Profile of the Software Implementation for Two Benchmark Simulations

	Palmitic Acid		CheY Protein	
Task	Time (s/step)	% Computation Time	Time (s/step)	% Computation Time
Real-space	1.56	70.25	0.93	74.42
Reciprocal-space (total)	0.34	15.40	0.10	7.80
FFTs	0.16	7.03	0.01	0.93
Assign charges	0.08	3.76	0.04	3.51
Find forces	0.06	2.61	0.03	2.09
Find B-spine coeffs.	0.02	1.07	0.01	1.18
Other	0.02	0.92	< 0.01	0.10
Building Neighbor List	0.19	8.55	0.13	10.80
Other	0.13	5.79	0.09	6.96

than the reciprocal-space part, but not as much longer.

The other option is to accelerate the reciprocal-space part in hardware as well. The reciprocal-space part of the calculation consists of several subtasks: finding interpolation coefficients, assigning charges to the grid, forward and backward FFTs, and so forth. None of these kernels individually is responsible for much of the simulation's overall computation time. To effectively accelerate the entire application, then, the whole reciprocal-space part must be accelerated. The pipeline to do the real-space part of the calculation and the pipeline to do the reciprocal-space part of the calculation cannot both fit in the FPGA at the same time because of area limitations. So, only one will be able to execute on the FPGA at a time and the FPGA will have to be reconfigured each time either of the functions is called. Reconfiguration takes about 0.05 seconds, so about 0.1 seconds of overhead would be added at each call [SRC 2004]. For the CheY protein simulation, this is not a solution since the reciprocal-space part only takes 0.1 s/step. Even for the palmitic acid simulation, considering the added reconfiguration cost, the reciprocal-space part of the computation may not be significantly accelerated with a custom hardware design.

For this study, we have chosen the first partitioning scheme, that is, the real-space part executes in hardware in parallel with the reciprocal-space part executing in software. To estimate the speed-up of a hardware implementation before actually doing the implementation, we take into account the time that the application will spend in software, in hardware, and communicating data between software and hardware. We can use the profiling data to determine the amount of time that the tasks left in software will take. We know the amount of data the will be transferred between hardware and software at each step because it is based on the number of atoms in the simulation. We also know the bandwidth between the hardware and software, so we can calculate the communication cost. Finally, we need to find the hardware cost, which is determined by the hardware implementation, which we will describe in the following subsection.

Take the palmitic acid simulation as an example. The tasks left in software are estimated to take 0.66 s/step. For a 52558-atom simulation, the communication time is estimated to be only 1.2 ms, given the position data and force data being transferred and the transfer rate of the MAPstation. The hardware implementation of the real-space part is estimated to take only 0.34 s, based on our prior work, so its computation will be overlapped by the reciprocal-space part in software. Thus, it does not contribute to the overall time. The total time is, therefore, estimated to be 0.66 s/step, leading to an estimated speed-up of 3.36×.

3.2 Hardware Design for the Real-Space Part

In [Scrofano et al. 2006], we determined that the best hardware design for the nonbonded force calculation in the shifted-force technique is a *write-back* design. Since the calculations in the real-space part are very similar to those in the shifted-force technique, we employ the same basic design in this work. The algorithm for this design in shown in Figure 2. In the algorithm, `positionOBM` and `forceOBM` represent on-board memories, `forceRAM` represents an on-chip memory, and CALC_REAL represents applying the real-space force and potential calculation equations, as well as other necessary techniques, such as the minimum image convention [Allen and Tildesley 1987].

The two inner loops in the algorithm are pipelined while the outer loop is not pipelined. The inner loop beginning on line 5 is the main force calculation loop. Each atom *i*'s neighbor list is traversed. The updated force on atom *i* is accumulated. The updated force on each neigh-

```
1  foreach atom i do
2      r_i ← positionOBM[i]
3      f_i ← forceOBM[i]
4      n ← 0
5      foreach neighbor j of i do
6          if |r_i − r_j| < r_c then
7              r_j ← positionOBM[j]
8              f_ij ← CALC_REAL(r_i, r_j)
9              f_i ← f_i + f_ij
10             f_j ← forceOBM[j]
11             forceRAM[n] ← f_j − f_ij
12             n ← n + 1
13         end
14     end
15     forceOBM[i] ← f_i
16     foreach f_j in forceRAM do
17         forceOBM[j] ← f_j
18     end
19 end
```

Figure 2: Algorithm for Write-Back Design

bor atom j is stored in on-chip memory. When all the neighbors for atom i have been processed, the pipeline for the inner loop is drained. The forces stored in on-chip memory are then written back to on-board memory in a pipelined fashion (the loop beginning on line16).

The main benefit of this design is that it avoids data hazards. For example, it does not require any on-board memories to be read and written in the same clock cycle. This is important because the on-board memories in our target reconfigurable computer are single ported; trying to access the same memory from two different places within a pipeline causes the pipeline to stall, which hurts performance. Because the pipeline is drained between iterations of the outer loop, no data hazards due to the lengthy pipelines in the floating-point cores arise. Another benefit of the design is that it minimizes the number of times that the on-board memories switch between read and write modes. Such a switch often carries a penalty of multiple cycles, which slows down the pipeline.

The main drawback to this write-back design is the need to flush the inner pipelines between successive iterations of the outer loop. Pipelines made up of pipelined floating-point cores are very long, so such pipeline flushing reduces the effectiveness of the pipeline.

3.2.1 $\mathrm{erfc}(x)$ and e^{-x^2}

There are currently no libraries available for performing transcendental functions, such as $\mathrm{erfc}(x)$ and e^{-x^2} in floating-point on FPGAs. Thus, we had to develop our own. We use table lookup and linear interpolation to calculate these two functions to five digits of precision. This method works particularly well on FPGAs because the table can be stored in on-chip memory.

In MD, we are also able to bound the inputs to the two functions. That is, we know ahead of time what range of values the x in $\mathrm{erfc}(x)$ and e^{-x^2} can take on. In the case of MD, $x = \beta\left|\vec{r}_i - \vec{r}_j + n\right|$. In our simulations, we utilize the common technique of setting β such that the real-space energy is 0 at the cutoff distance, r_c. Thus, it is guaranteed that for any pair of atoms i and j, $0 \le x \le \beta r_c$. For a cutoff distance of r_c =10 Å, as is used in our simulations, it turns out that this limits the range of inputs $0 \le x \le 3.6$. Both $\mathrm{erfc}(x)$ and e^{-x^2} are well-behaved functions in this region.

The table lookup and interpolation steps are accomplished using fixed point arithmetic and shifting, which is much more area efficient than floating-point arithmetic. Any values that are larger than βr_c return $\mathrm{erfc}(\beta r_c) \approx e^{-(\beta r_c)^2} \approx 0$ and any values that are smaller than a threshold return $\mathrm{erfc}(0) = e^{-(0)^2} = 1$.

3.3 Reconfigurable Computer Implementation Details and Results

For implementation on the MAPstation, we made a few modifications to the traditional techniques. The neighbor list is usually implemented as a long list of atoms and a separate array that points to the start of a particular atom's set of neighbors. In our implementation, we instead insert the atom indices into the list right before their neighbors. We use the most significant bit of the index as a flag to denote which atoms start new sets. Also, we pack atom types and atom indices into 64-bit words in the neighbor list instead of having a separate type array. We do this to save on-board memory banks. Keeping this added information in the neighbor list does not increase data transfer costs because the transfer word on the MAPstation is 64 bits wide anyway. Similarly, rather than having a separate charge array, we pack the charge in the same array as the positions. Positions have three components that are each 32 bits but the amount of data transferred must be in multiples of 64 bits, so we have to transfer an extra 32 bits every time the positions are transferred anyway.

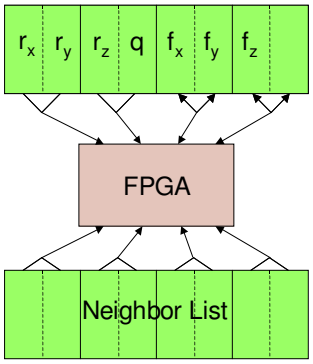

Figure 3: Data Layout in On-Board Memory

When the program runs, it executes the velocity Verlet algorithm. During the force calculation step, two threads are created using pthreads [POS 2006]. One thread calls the reciprocal-space part software function and the other calls the hardware implementation of the real-space part. The first time the real-space part is called, all the necessary constants are transferred to the FPGA and stored on-chip, either in registers or in embedded memories. These constants only need to be transferred once as they remain on-chip throughout the simulation. Each time the real-space part is called, the position/charge array is DMAed to the FPGA's on-board memory. The neighbor list is effectively streamed to the FPGA by alternating DMAs between four on-board memory banks. At the end of the calculation, the forces are DMAed back to the GPP. The data layout in on-board memory is shown in Figure 3, where r_a and f_a are the a components of the position and force, respectively, solid lines denote borders of memory banks, and dashed lines show data packed into one word. When both threads have finished, the force calculation proceeds with bonded force calculation and the force corrections.

We implemented MD with SPME for the electrostatics, as described above, on the SRC 6 MAPstation. The software code was written in C and compiled with the Intel C compiler version 8.1.

All of the hardware design, except for the table lookup and interpolation functions for $\mathrm{erfc}(x)$ and e^{-x^2}, was coded in C for the SRC Carte compiler. This compiler generates hardware designs from C code that is appropriately annotated with SRC-provided macros to perform such tasks as data movement between the GPP and the FPGA's on-board memory.

The $\mathrm{erfc}(x)$ and e^{-x^2} tables each have 2048 17-bit wide elements. Each table can fit in two embedded memories in the target FPGA. 17-bit wide values were chosen because, in addition to providing the required precision in the results, they allow each multiply done during the interpolation to use only one of the 18×18-bit embedded multipliers in the target FPGA. The values in the tables were generated by a separate software program. The thresholds for maximum and minimum values are set at 4.0 and 2^{-23}, respectively. The maximum is set to 4.0, rather than 3.6, because 4.0 can be checked simply by looking at the exponent of the floating-point number. 2^{-23} is chosen as the minimum because any number smaller than that will have all its meaningful bits shifted out when the number is converted to fixed-point. These circuits are coded in the VHDL hardware description language and synthesized using Synplicity Synplify Pro, version 8.1.

Table 2: MAPstation Performance Results

Simulation	Latency (s/step) SW only	SW + HW	Speed-up
Palmitic Acid	2.22	0.76	2.92×
CheY Protein	1.25	0.46	2.72×

The complete hardware design is mapped to the FPGA using Xilinx's ISE tools, version 7.1.04.

The results of the acceleration on the reconfigurable computer are shown in Table 2. We achieved a 2.92× speed-up and a 2.72× speed-up for the palmitic acid and CheY protein simulations, respectively, by moving the real-space part into hardware and executing it in parallel with the reciprocal-space part. The speed-up for palmitic acid is less than estimated above. A large part of this is due to a threading conflict between designs on the MAPstation and the FFTs from the Intel MKL. Thus, the FFTs execute more slowly in software when we introduce hardware acceleration for the real-space part. Also, in our estimates, we ignored overheads such as DMA start-up costs and thread creation costs.

4 Discussion

We have presented a preliminary investigation of MD simulations that include the use of SPME for advanced electrostatic calculation on reconfigurable computers. We have shown that it is possible to achieve a significant speed-up without accelerating everything in the application. The design process served to emphasize the need for profiling and performance estimation before actual implementation. By targeting our acceleration efforts to the most computationally intensive task, we are able to obtain at least a 2.7× speed-up over a software-only implementation. Had we instead tried to accelerate an-

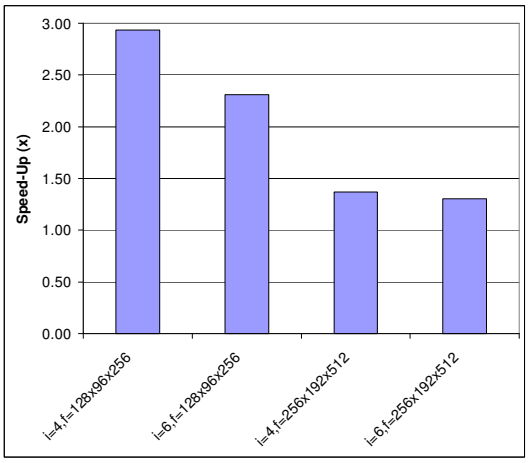

Figure 4: Speed-Ups of Palmitic Acid Simulation with Various Parameters

other part of the application, such as the FFTs within the reciprocal-space part, we could not have achieved nearly as high of a speed-up.

4.1 Partition Space

The introduction of the SPME technique to the simulation program has added tradeoffs to the design space for partitioning the MD application between hardware and software. So far, we have discussed the profile for two simulations that use fourth order interpolation and have particular FFT sizes. However, the interpolation order and the size of the FFTs used in the simulation can significantly impact the application profile.

As an example, we consider what would happen if the palmitic acid simulation used sixth-order interpolation and had a grid size of $256 \times 192 \times 512$, instead of its current configuration. In that case, the speed-up from moving the real-space part to hardware drops to $1.30\times$. The reason is that the real-space part now only accounts for 33.79% of the overall simulation time while the reciprocal-space part accounts for 58.44% of the overall simulation time. There is thus a tradeoff between interpolation order, FFT size, and speed-up. Figure 4 shows the speed-ups for various configurations of the palmitic acid simulation, where i is the interpolation order and f is the FFT size.

Another way to look at this is that changing the parameters changes the decision about which part or parts of the simulation should be moved into hardware. In the case described above in which the reciprocal-space part takes more time than the real-space part, it would be advantageous to accelerate the reciprocal-space part in hardware instead of the real-space part. Or, it may in fact be better to accelerate both parts of the force calculation in hardware.

The profile would also change if, through further optimization, the real-space part took less time to execute. For instance, GROMACS uses optimizations based upon the types of molecules interacting to reduce the number of operations in the real-space part of the force calculation. On x86 platforms, they use assembly-coded routines to increase the performance even further [van der Spoel et al. 2005]. Increasing the performance of the real-space part of the force calculation reduces its contribution to the overall computation time which in turn may make accelerating other parts of the simulation more attractive.

One more case in which we may change our decisions about what should and should not be moved to hardware is the case of a highly parallel simulation implementation. In this case, the number of atoms per node may be small because the atoms are distributed among many processors. However, it can be difficult to effectively parallelize the 3D FFT. FPGAs have demonstrated high performance in 1D FFT computations, especially large FFTs [Hemmert and Underwood 2005]. There may be a partitioning in which a small number of FPGAs perform the FFTs needed in the reciprocal-space part while the rest of the calculations are distributed among a large number of GPPs. Whether or not such a partitioning is practical will depend upon the performance of FPGAs in executing floating-point 3D FFTs, which is not well-studied at this time.

4.2 Libraries for Reconfigurable Computers

The fact that, in this application, solely accelerating the FFTs would not have greatly benefited the application as a whole serves to make a point about libraries for reconfigurable computers: they must be much more flexible than libraries for GPPs. For GPPs, the fastest library implementation for a particular kernel (FFT, for example) is best because the tasks in the application must run sequentially. For reconfigurable computers, on the other hand, there is an inherent parallelism between hardware and software. Running a kernel task more slowly in software may be beneficial if it allows another, more computationally intensive task, to run in hardware. In the embedded computing community, this idea is known as *software deceleration* [James-Roxby et al. 2004].

Thus, libraries for reconfigurable computers must do more than provide a common interface and highly op-

timized implementations for various platforms and they must be more than just wrappers around hardware implementations that always execute the hardware version if it is available. At the very least, the user should be able to give the library "hints" as to whether the software or hardware version of the library kernel should be employed. A more advanced system would allow the library to choose, at run-time, which implementation should be employed, based on parameters such as problem size (number of points in the FFT, for example). The system would provide an interface between the library's performance models and the application's performance model. An additional idea, proposed in [Smith et al. 2005a], is to fuse functions to make the library functions encompass more computation than is typically done in libraries for GPPs.

5 Conclusion

We have developed a preliminary reconfigurable-computer implementation of an MD simulation application that includes the SPME method for accurate electrostatic calculations. We were able to obtain speed-ups of almost $3\times$ by moving only the real-space part of the calculation into hardware. We have then discussed some other possible partitions and the ways in which the partitioning scheme depends upon the parameters of the simulation. Overall, our experience demonstrates that in order to achieve a speed-up for a scientific application such as MD simulation, careful partitioning between hardware and software is necessary.

There are many areas for future work. One is to develop a better understanding of the application profile when the simulation is parallelized over multiple GPPs. Insight into the behavior of the application when parallelized among many nodes will help focus our evaluation of the various partitioning schemes. Along those lines, it will also be imperative to understand the hardware performance of the various kernels in the reciprocal-space part when they are implemented in hardware with floating-point arithmetic. Another area is to investigate the possibilities for acceleration of other electrostatic techniques, such as the fast multipole method [Greengard and Rokhlin 1987].

6 Acknowledgments

This work is supported by Los Alamos National Laboratory under contract/award number 95976-001-04 3C.

We wish to thank the U. S. Army ERDC MSRC for access to its MAPstation, Frans Trouw of Los Alamos National Laboratory for providing the example simulations, and Ronald Scrofano, Sr. for his assistance with the erfc(x) and e^{-x^2} units and helpful discussions about the FFT.

References

ALLEN, M., AND TILDESLEY, D. J. 1987. *Computer Simulation of Liquids*. Oxford University Press, New York.

2004. *AMBER 8 Users' Manual*. `http://amber.scripps.edu/doc8/amber8.pdf`.

AZIZI, N., KUON, I., EGIER, A., DARABIHA, A., AND CHOW, P. 2004. Reconfigurable molecular dynamics simulator. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 197–206.

BARGIEL, M., DZWINEL, W., KITOWSKI, J., AND MOSCINSKI, J. 1991. C-language molecular dynamics program for the simulation of Lennard-Jones particles. *Computer Physics Communication 64*, 193–205.

2005. Cray, Inc. `http://www.cray.com`.

ESSMANN, U., PERERA, L., BERKOWITZ, M. L., DARDEN, T., LEE, H., AND PEDERSEN, L. G. 1995. A smooth particle mesh Ewald method. *Journal of Chemical Physics 103*, 19 (November), 8577–8593.

GOKHALE, M. B., AND GRAHAM, P. S. 2005. *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*. Springer.

GOKHALE, M. B., RICKETT, C. D., TRIPP, J. L., HSU, C. H., AND SCROFANO, R. 2006. Promises and pitfalls of reconfigurable supercomputing. In *Proceedings of the International Conference on Engineering Reconfigurable Systems and Algorithms*.

GOVINDU, G., SCROFANO, R., AND PRASANNA, V. K. 2005. A library of parameterizable floating-point cores for FPGAs and their application to scientific computing. In *Proceedings of the International Conference on Engineering Reconfigurable Systems and Algorithms*, T. Plaks, Ed.

GREENGARD, L., AND ROKHLIN, V. 1987. A fast algorithm for particle simulations. *Journal of Computational Physics 73*, 325–348.

GU, Y., VANCOURT, T., AND HERBORDT, M. C. 2005. Accelerating molecular dynamics simulations with configurable circuits. In *Proceedings of the 2005 International Conference on Field Programmable Logic and Applications*.

HEMMERT, K. S., AND UNDERWOOD, K. D. 2005. An analysis of the double-precision floating-point FFT on FPGAs. In *IEEE Symposium on Field-Programmable Custom Computing Machines*.

2006. Intel Corp. `http://www.intel.com`.

JAMES-ROXBY, P., BREBNER, G., AND BEMMANN, D. 2004. Time-critical software deceleration in an fccm. In *Proceedings of the Twelfth Annual International Symposium on Field Programmable Custom Computing Machines*.

KALÉ, L., SKEEL, R., BHANDARKAR, M., BRUNNER, R., GURSOY, A., KRAWETZ, N., PHILLIPS, J., SHINOZAKI, A., VARADARAJAN, K., AND SCHULTEN, K. 1999. NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics 151*, 283–312.

KINDRATENKO, V., AND POINTER, D. 2006. A case study in porting a production scientific supercomputing application to a reconfigurable computer. In *Proceedings of the Fourteenth Annual IEEE Symposium on Field-Programmable Custom Computing Machines*.

LEE, S. 2005. *An FPGA Implementation of the Smooth Particle Mesh Ewald Reciprocal Sum Compute Engine (RSCE)*. Master's thesis, University of Toronto.

MAO, Z., GARG, A., AND SINNOTT, S. B. 1999. Molecular dynamics simulations of the filling and decorating of carbon nanotubules. *Nanotechnology 10*, 3, 273–277.

NAKANO, A., KALIA, R. K., VASHISHTA, P., CAMPBELL, T. J., OGATA, S., SHIMOJO, F., AND SAINI, S. 2001. Scalable atomistic simulation algorithms for materials research. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, ACM Press, New York.

NARUMI, T., KAWAI, A., AND KOISHI, T. 2001. An 8.61 Tflop/s molecular dynamics simulation for nacl with a special-purpose computer: MDM. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, ACM Press, New York.

OProfile. `http://oprofile.sourceforge.net/about/`.

PATRA, M., KARTTUNEN, M., HYVNEN, M. T., FALCK, E., LINDQVIST, P., AND VATTULAINEN, I. 2003. Molecular dynamics simulations of lipid bilayers: Major artifacts due to truncating electrostatic interactions. *Biophysics Journal 84*, 3636–3645.

1996. *PDB Format Guide*. `http://www.rcsb.org/pdb/file_formats/pdb/pdbguide2.2/guide2.2_frame.htm%l`.

2006. POSIX Threads Tutorial. `http://llnl.gov/computing/tutorials/pthreads`.

SCROFANO, R., GOKHALE, M., TROUW, F., AND PRASANNA, V. K. 2006. A hardware/software approach to molecular dynamics on reconfigurable computers. In *Proceedings of the Fourteenth Annual IEEE Symposium on Field-Programmable Custom Computing Machines*.

2006. Silicon Graphics, Inc. `http://www.sgi.com`.

SMITH, M. C., VETTER, J. S., AND ALAM, S. R. 2005. Scientific computing beyond CPUs: FPGA implementations of common scientific kernels. In *Proceedings of the 8th Annual Military and Aerospace Programmable Logic Devices International Conference*.

SMITH, M. C., VETTER, J. S., AND LIANG, X. 2005. Accelerating scientific applications with the SRC-6 reconfigurable computer: Methodologies and analysis. In *Proceedings of the 2005 Reconfigurable Architectures Workshop*.

2004. SRC Computers, Inc. `http://www.srccomputers.com`.

TANG, P., AND XU, Y. 2002. Large-scale molecular dynamics simulations of general anesthetic effects on the ion channel in the fully hydrated membrane: The implication of molecular mechanisms of general anesthesia. *Proceedings of the National Academy of Sciences of the United States of America 99*, 25 (December), 16035–16040.

UNDERWOOD, K. 2004. FPGAs vs. CPUs: Trends in peak floating-point performance. In *Proceedings of the 2004 ACM/SIGDA Twelfth International Symposium on Field Programmable Gate Arrays*.

VAN DER SPOEL, D., LINDAHL, E., HESS, B., GROENHOF, G., MARK, A. E., AND BERENDSEN, H. J. C. 2005. GROMACS: Fast, flexible, and free. *Journal of Computational Chemistry 26*, 16 (December), 1701–1718.