Evaluation of a Dynamic Load-Balancing Molecular Dynamics Application using Automated HW/SW Architecture Generation

Christopher Rogers
Utah State University
Logan, UT
crogers@cc.usu.edu

Matthew Areno
Sandia National Labs
Albuquerque, NM
mareno@sandia.gov

Brandon Eames
Utah State University
Logan, UT
beames@engineering.usu.edu

Abstract

The application of reconfigurable computing technology to solve high performance computing problems has received increasing attention in recent years. N-body problems such as Molecular Dynamics simulations are common candidates for such studies. The ability to perform dynamic load-balancing offers the potential of improving performance, but the implementation of hardware architectures implementing load balancing is often tedious and error prone. Further, the isolation and quantification of performance gains rendered by the integration of dynamic load balancing hardware presents a challenge for system developers. In prior work, the authors have proposed a dynamic load balancing architecture for supporting Molecular Dynamics computations. This paper presents an evaluation of the impact of dynamic load balancing on a Molecular Dynamics application by comparing several candidate architectures against a previously-developed dynamic load balancing architecture. A hardware/software generation tool has been employed to automatically create the candidate architectures and architecture simulators for comparison. The results indicate that while specialized architectures in certain circumstances can perform better than the dynamic load balancing architecture, the integration of dynamic load balancing offers a consistently efficient use of FPGA resources, independent of variations in proximity between molecules.

1 Introduction

Molecular Dynamics (MD) is a method of computing dynamic particle interaction on the molecular or atomic level. Computing platforms as large and powerful as Blue Gene [4] and as FPGAs [3] have been the target for MD simulations. MD simulations rely on knowing at the start of the simulation the mass, position, and velocity of each particle in the system. Each particle is considered as a candidate particle, and then compared with every other particle in the system to determine the net total force imposed on the candidate. This is performed using a distance calculation (DC), followed by a force calculation. Force calculations are gated by a pre-determined cutoff radius, under the assumption that only particles which are sufficiently close actually impact their respective net forces. The force due to the Lennard-Jones potential between close particles is summed up and applied to each candidate particle, resulting in changes to position and velocity. The calculation of net force is known as the Lennard-Jones Potential calculation (LJPC). When the net force for each particle has been calculated, new positions and velocities due to the Lennard-Jones potential are computed through a series of motion estimation equations that rely on Euler integration. These motion estimation equations are known as the Verlet method of integration. The process of net force calculation and Verlet integration repeats for each time step of the simulation.

The computational structure of the Molecular Dynamics algorithm presents an acceleration challenge to developers of customized hardware architectures. The variance in computational load from problem to problem between the two classes of computation lead to the development of a hardware architecture for supporting dynamic load balancing, called a FLEX processor. The FLEX processor facilitates the rapid switching between the two classes of computation required by the MD architecture, based on runtime load monitoring. The cost of this dynamic load balancing is increased controller complexity, as well as a possible decrease in resource utilization due to the need to design hardware which can compute two distinct algorithms. This architecture was introduced in a

previous work [1]. While the results of the dynamic load balancing architecture compared favorably with a standard MD simulator, the evaluation was performed on a macro-scale, comparing system level design against system level design. The question of whether dynamic load balancing is truly responsible for performance gains was not addressed in the prior work. Other similar efforts have been made in this area. Application specific processors, including DC and LJPC units in a similar study [6], yielded increases in performance but still faced the difficulty in addressing what the optimal configuration should be for processing elements. An effort was made to accelerate chemical computations in [7], where the load was distributed across as grid of processors. The demands of each processor in the gird were continually evaluated. Two processors who had interdependent data were considered neighbors. Constantly evaluating neighbors through load balancing had a considerable effect on overall timing of the computation.

This paper seeks to address the question of the impact of dynamic load balancing on the performance and efficiency of the hardware acceleration of Molecular Dynamics simulation. This is done by comparing the FLEX based MD architecture with other architectures performing similar functions, but with a static allocation of computational resources. However, due to the time intensive nature of designing hardware systems, the generation of comparative hardware designs often takes as much, if not more, time than the creation of the initial architecture itself. Thus, the problem of evaluating dynamic load balancing presented several challenges that needed to be addressed. A HW/SW architecture generator called CHARGER [5] was employed to automatically generate hardware units, as well as C++ files that could be used in a software simulator. From the files generated by CHARGER, three static architectures were created along with corresponding simulators suitable for comparing against the dynamic load balancing architecture. Results are presented comparing both the performance and efficiency of the candidate architectures.

2 Load Balancing

The N-body nature of the MD problem comes about as a result of the cutoff radius that determines if forces are imposed between nearby particles. This also results in the fact that for a given particle in the system, it is impossible to know beforehand how many particles will be close enough to the candidate particle to exert a force on it, since not all systems of particles will be uniformly distributed. Additionally, the complexity of the LJP calculation is far greater than that of a simple distance calculation, thereby causing further load-balancing issues. Consider, for example, that half of the particles in a system have 25% of their neighboring particles close enough to exert a force, while the other half of the particles have 75% of the particles close enough to exert a force. The entire system would have 50% of the particles within radius. Such a benchmark would lead a custom hardware designer to determine an allocation of hardware resources to support twice as many distance calculations as LJP calculations. However, the number of calculations due to force will greatly fluctuate from one half of the particles within system to the other. Handling such fluctuations in computations is the focus of load balancing techniques.

In order to prove that the addition of the FLEX unit was a useful and efficient addition to the MD architecture, it became an item of interest to create a set of similar designs, each with a static number of distance and force calculators and compare with the results to the FLEX-based design. The integration of the FLEX processor is not without cost, both in terms of controller complexity and footprint, as well as possible reductions in resource utilization due to the need to handle two distinct computation modes. Static architectures typically do not exhibit the same amount of overhead from controller complexity, but will often suffer from far greater reductions in resource utilization, specifically when the percentage of atoms within radius is not in the target range of the architecture. It then becomes a question of whether the dynamic load-balancing architecture can produce sufficient throughput and resource utilization to compensate for the increased area cost.

3 Simulator Functionality

There are four simulators that will be used to benchmark the MD design. The first design was developed from scratch, while the other three were integrated from CHARGER-generated modules. While CHARGER is capable of generating individual hardware modules that function as specified, the integration of those individual modules into a system still requires significant work. Using data flow graphs representations of the DC, LJPC, and Verlet units, CHARGER generates individual, pipelined modules that speed up the design, making it functionally more similar to the existing dynamic design based on the FLEX unit.

The four simulators are as follows:

1.) A – 2 DC, 4 FLEX, 2 LJPC Units
2.) B – 4 DC, 4 LJPC Units
3.) C – 6 DC, 2 LJPC Units
4.) D – 2 DC, 6 LJPC Units

All four simulators are based on a hardware design, thereby resulting in code that follows a global clock shared among units. Using CHARGER-generated modules, a main file is created that instantiates the appropriate number of each unit. Within this file, each unit is connected using STL lists that behave similar to FIFOs in hardware. Data is then feed into the inputs of the systems and flows through the simulator in a similar fashion to how it would work in hardware.

The goal of these three simulators is the represent the best solution for low, medium, and high percentages of atoms within radius. For a low percentage within radius, a 6DC/2LJPC implementation would provide the most DC units for performing DC calculations, thereby speeding the design up as much as possible, while still being able to keep up with LJPC needs. The 4DC/4LJPC implementation is intended to target the middle range percentages, while the 2DC/6LJPC targets the high percentages within radius.

4 Comparisons, Tests, and Results

All four simulators were run on the same data set. The data set is a group of randomly placed particles in three dimensional space with a pre-determined range of radii that span from 7% to 87% within radius. The total number of clock cycles necessary to execute was counted for each simulator and for each simulator run at a given cutoff radius. As the cutoff radius and percentage of particles within radius increases, it is expected that the ratio of DC:LJPC computations will even out as well.

Each simulator was run twice on the same data set, which contains 3600 total particles in the system. Figure 2 shows the results of measuring total clock cycles for each of the simulations.

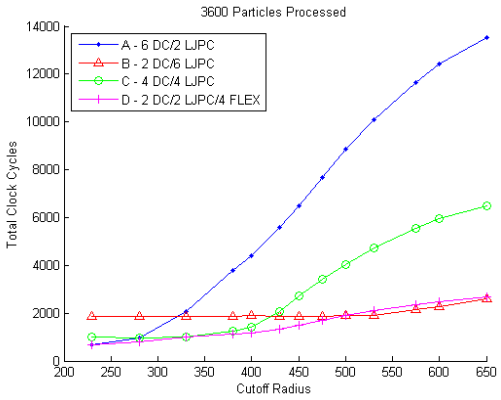

Figure 1 - Total clock cycles for 3600 particles processed.

For low amounts of particles within radius, or rather, lower cutoff radii, it is interesting to note that Simulator A performs as well as the Simulator D. This is expected; for low amounts of particles within radius, there is no need for the four FLEX units to change modes. The difference is the result of controller overhead, but is negligible for low amounts of particles in radius because controller code is never executed as the FLEX units do not change modes. Simulators A and D remain virtually identical for low cutoff radii. As would be expected as well, Simulator A performs increasingly poorly for larger numbers of particles within radius. Even Simulator C, which has an even number of DC and LJPC units, begins to exhibit the need for having more LJPC units to aid in those costly computations.

Simulators B and D have the same commonalities on the high end of particles in radius as was seen with Simulators A and D for the low end of particles in radius. They are not identical, however, because during the course of the simulation, Simulator D underwent a period of brief delay from the FLEX units switching state. However, for the purpose of proving the efficiency of having the FLEX unit, it is sufficient to see that Simulator D takes on the characteristics of Simulator A for low percentages in radius, Simulator C for roughly half the particles in radius, and Simulator B for high percentages in radius.

Throughput was calculated as the number of particles processed per second. The numbers presented in figures 2 and 3 are based upon an implementation on a Xilinx Virtex4 VFX140 running at 150MHz.

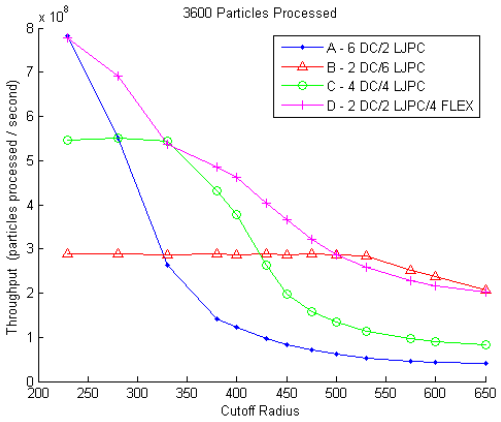

Figure 2 – Throughput results for each simulator.

The throughput values obviously go down as a function of total percentage in radius. The more particles there are in radius, the more time it takes for the architecture to complete the simulation. It is interesting to note that Simulator B has a very static

throughput curve; this is due to the fact that for the lower and middle amounts of particles within radius, the number of DC unit calculations for all total particles is what is keeping the simulator busy. Where this implementation suffers is in resource utilization. For lower and middle percentages, the DC units are consistently be used, whereas the LJPC units are rarely used and result in wasted area.

Between the simulators, Simulator A has the highest efficiency on the lower percentages in radius due to the fact that its area is the smallest. Despite having a larger area, Simulator D still remains the best option overall. The effects are observable in the efficiency plot, shown in figure 3, where efficiency is measured in throughput per unit area, or slice.

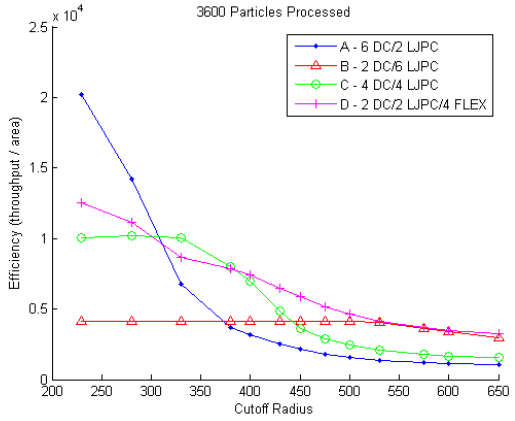

Figure 3 - Efficiency results for each simulator

7 Conclusions and Future Work

While still requiring extra effort, the CHARGER based simulators performed comparably to the FLEX based design. In addition, the amount of time required to develop the CHARGER based simulators is far less than what was required for the FLEX based architecture, requiring only about a day to implement as opposed over a week. Additional work is currently being performed to reduce this time even more by automating the integration process of multiple directed flow graphs.

It is reasonable to conclude that the presence of the FLEX processor greatly aids in reducing the total time to simulate an MD implementation. This has been illustrated by the low clock cycle count over the varying ranges of particles within radius. It is also clear that the inconsistent nature of Molecular Dynamics contributes to the difficulty in preparing an implementation to be able to handle the different numbers of particles within radius from simulation step to simulation step. The FLEX unit, while not the most efficient for all cutoff radii, is the most sensible solution to addressing dynamic load balancing.

Future work areas have been discussed, and in general attempt to use different and more current FPGA architectures to implement the FLEX units. Generation of soft core microprocessors and mapping all FLEX units to them would conceivably give more area to the rest of the chip, effectively increasing efficiency and possibly allowing for more processing elements to be put on the main chip. It is also possible to use this space on the chip to increase the size of the pipelines for each processing element. In addition to increasing overall performance, this would greatly enhance the efficiency of the design.

8 Bibliography

[1] Phillips, J.; Areno, M.; Rogers, C.; Dasu, A.; Eames, B., "A Reconfigurable Load Balancing Architecture for Molecular Dynamics," *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International* , vol., no., pp.1-6, 26-30 March 2007

[2] Model, J.; Herbordt, M.C., "Discrete Event Simulation of Molecular Dynamics with Configurable Logic," *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on* , vol., no., pp.151-158, 27-29 Aug. 2007

[3] Gu, Y.; Van Court, T.; DiSabello, D.; Herbordt, M.C., "Preliminary report: FPGA acceleration of molecular dynamics computations," *Field-Programmable Custom Computing Machines, 2005. FCCM 2005. 13th Annual IEEE Symposium on* , vol., no., pp. 269-270, 18-20 April 2005

[4] F. Allen, G. Almasi, et al, “Blue Gene: A vision for protein science using a petaflop supercomputer,” *IBM Systems Journal, Deep Computing for the Life Sciences*, Vol 40, No 2, 2001

[5] Matthew Areno, “Automated Constraint-Based Hardware Architecture Generation for Reconfigurable Computing Systems”, Master’s Thesis, Utah State University, 2007

[6] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, “Reconfigurable molecular dynamics simulator,” in 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’04), pp. 197–206. Los Alamitos, CA, USA: IEEE Computer Society, 2004.

[7] J. M. Bahi, S. Contassot-Vivier, and R. Couturier, “Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms,” IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 4, pp. 289–299, 2005.