# A Reconfigurable Load Balancing Architecture for Molecular Dynamics

Jonathan Phillips, Matthew Areno, Chris Rogers,
Aravind Dasu, and Brandon Eames

Utah State University
Dept. of Electrical and Computer Engineering
Logan, UT 84322-4120 USA
{jdphillips, matthewareno, crogers} @cc.usu.edu
{dasu, beames} @engineering.usu.edu

## Abstract

*This paper proposes a novel architecture supporting dynamic load balancing on an FPGA for a Molecular Dynamics algorithm. Load balancing is primarily achieved through the use of specialized processing units, referred to as FLEX units. FLEX units are able to switch between tasks required by a molecular dynamics algorithm as often as needed in order to cater to the nature of the input parameters. This architecture is capable of run-time performance analysis and dynamic resource allocation in order to maximize throughput. Results of a prototype of the architecture targeting an FPGA are presented.*

## 1 Introduction

Molecular Dynamics (MD) commonly refers to a set of algorithms or packages designed to model or simulate dynamic particle interaction on the molecular or atomic level. Knowledge gained from studying such interactions has led to a myriad of scientific advances across several fields, from pharmaceuticals to material sciences. While MD simulations can include a wide range of chemical and physical property calculations, our design concentrates on a simplified version, primarily concerned with total energy calculation due to particle interaction. This energy is calculated through the Lennard-Jones potential. Because particle distances beyond a certain threshold yield trivial Lennard-Jones potentials, an imaginary spherical cutoff radius is applied to any given particle. All particles outside of the threshold radius are not included in Lennard-Jones potential calculations. Thus, our MD simulation involves two primary computations: a distance calculation (DC) and a Lennard-Jones potential calculation (LJPC).

1-4244-0910-1/07/$20.00 ©2007 IEEE.

Due to the dynamics of particle interaction and motion during simulation, the task of compile-time load balancing between the two classes of computations is a non-trivial task. Potential load imbalances can significantly impact an accelerator-architecture's resource utilization efficiency, especially when considering implementations based on custom architectures.

As with other N-body problems, Molecular Dynamics can suffer from an imbalance in load characteristics that varies at run time. We propose to accelerate the process of computation and increase resource utilization efficiency through dynamic load balancing; processing units designed to switch tasks facilitate computation and prevent other processing units from waiting longer than needed to receive data required to finish their computations.

Our design introduces a FLEX unit that can efficiently, but not simultaneously, support both classes of computations employed in MD simulation. FLEX units can change between computation modes on demand, based on the algorithm needs at runtime. Our design also includes a set of fixed, mode-specific computational units, rate-balancing and load-sharing data management facilities, as well as a system controller which coordinates mode switching of the FLEX units. The subsequent sections of the paper describe the rationale behind and the design of our load balancing architecture for MD simulation. Section 2 discusses related work, section 3 discusses all included design concepts; DC, LJPC, the FLEX units, the feeder FIFO, and the inverter section. Section 4 explains preliminary results and the analysis of the design. Section 5 concludes with a discussion of future work envisioned and the direction of the project.

## 2 Related Work

Acceleration of MD simulations using FPGAs has been addressed in several recent efforts. For example,

an FPGA has been used as a floating-point coprocessor for acceleration of N-body problems (such as MD) [1]. Pipelined, complex arithmetic units using floating point adders, multipliers, dividers, and square root units were created from efficient low-level primitives. A mixture of 60 of these units was placed on a Xilinx Virtex2 FPGA. Sustained throughput of up to 3.9 GFLOPS was attained under ideal conditions. Another recent approach to solving MD problems involves the use of Application Specific Processors (ASPs) on FPGAs [2]. Unique computational units are derived, such as pair generators, Lennard-Jones potential calculators, and acceleration update blocks. A mixture of these blocks is placed upon the FPGA. Different data sets will yield different performance levels with different architectures. Difficulties arise in determining the ideal FPGA configuration for a specific data set. In [3], the Lennard-Jones and Coulombic force computations were mapped to an FPGA. Rather than using supercomputers or custom hardware, this research involves solely commercial-off-the-shelf (COTS) components. An investigation was performed on the tradeoff between accuracy and speed, by adjusting the bit-precision of floating-point numbers. Depending on the accuracy required, this system achieves speedups between 31 and 88 times over a conventional PC approach. In [4], the authors have proposed a highly optimized double precision floating point based deeply pipelined architecture that can offer 3.9 GFLOPs of performance. Another effort [5] takes the approach of coarse grain hardware software partitioning for non-bonded force evaluations on a FPGA accelerated Genereal Purpose Processor (GPP) system. They calculate the pair list (the result of successful distance calculations) on the GPP and then transfer the lists to the FPGA for evaluation of the force calculations. The calculations of next step velocities, positions and new pair list generation are again done on the GPP.

# 3 Proposed Architecture

In this section, we present an overview of our architecture developed for MD simulations. Figure 1 provides a block-level depiction of our design, illustrating both subunits and information flow. We discuss in detail each individual unit, and provide, where appropriate, a data flow graph or block diagram modeling subunit internals.

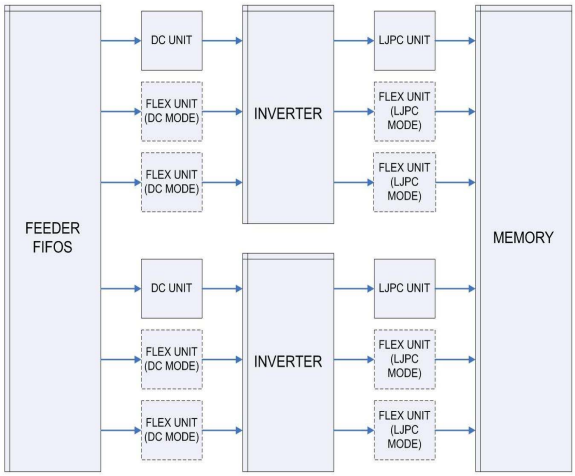


**Figure 1. Top level block diagram**

## 3.1 DC Unit

As mentioned above, distance calculation is one of two primary computations applied in MD simulations. The goal of the DC unit is to perform this distance calculation between pairs of particles. Once the distance is obtained, it is then squared in preparation for LJPC (which dictates that potential is inversely affected by the square of the distance). The integration of the distance-squaring operation into the DC unit represents a choice made in how to partition the computation load between the DC unit and LJPC. Each DC unit receives a source particle's location, designated by the triple $\langle X_s, Y_s, Z_s \rangle$. On subsequent clock cycles, a destination particle's location, of the form $\langle X_d, Y_d, Z_d \rangle$ is loaded into the DC unit and a distance calculation is initiated. Once the DC unit obtains the square of the distance between the two particles, it compares this value with the user-specified cutoff radius (CR), which is actually square in order to make the proper comparison. Particle pairs whose distance is found to be within the radius are forwarded to the inverter unit. Pairs which do not fall within the cutoff are discarded. Distance calculation is performed far more frequently than any other computation in MD simulation; hence the hardware design of our DC unit is optimized for high throughput. The DC unit is implemented in hardware as a direct instantiation of the computation data flow graph shown in Figure 2, known as a 1-to-1 mapping. This allows new data to be loaded into the DC unit at each clock cycle without incurring errors in computation or delays in waiting for necessary data to complete computation. This 1-to-1 mapping

speeds up total time necessary to compute distances, which aids in sifting through which particle distances are sent to LJPC units and which are discarded.

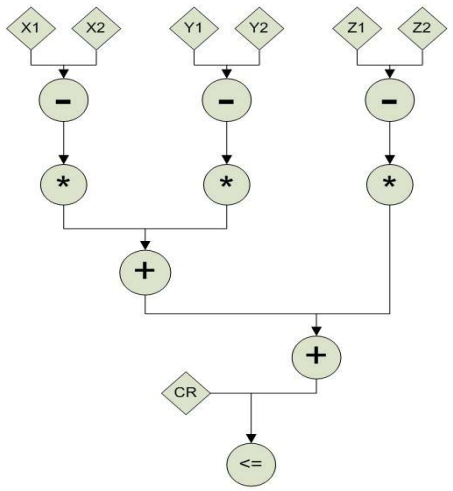


**Figure 2. DC unit data flow graph**

## 3.2 Feeder FIFO

The Feeder FIFO unit is responsible for distributing particle location information to the units performing distance calculation. The Feeder FIFO has been designed to facilitate variable-rate consumption of particles by the computation units, thereby facilitating load balancing. The Feeder FIFO consists of six small RAM modules that feed into a controller. The controller then directs traffic to six units; two DC units and four FLEX units. Because DC units will invariably empty their RAM modules before the LJPC modules, the controller needs to be able to send a free DC unit the data from different RAM modules. This has a twofold effect on load balancing: the DC unit doesn't needlessly wait for FLEX units to finish their computations, and LJPC calculation is aided by extra FLEX units that can stay in LJPC mode. The controller checks the levels of each RAM module when a DC unit becomes free, and selects the first RAM module that still has a generous amount of data to be calculated and whose corresponding FLEX unit is currently in LJPC mode. If all FLEX units are in DC mode, the FLEX unit corresponding to the fullest RAM module is switched to LJPC mode and the DC unit takes over that RAM module. This process is repeated until all RAM modules are emptied.

## 3.3 LJPC Unit

This unit is responsible for performing a portion of the Lennard-Jones potential calculation. This consists of calculating the force in all three dimensions, as well as the potential energy between the two particles. This is shown in the formula below.

$$Force = 48 * {}^{1}\!/_{r^2} * {}^{1}\!/_{r^6} * \left({}^{1}\!/_{r^6} - 0.5\right)$$

$$PotentialEnergy = 4 * {}^{1}\!/_{r^6} * \left({}^{1}\!/_{r^6} - 1\right)$$

The $1/R^2$ value in the root node of Figure 3 is supplied to the LJPC unit from the inverter module, and represents the inverse of the radius squared. The $D_X$, $D_Y$, and $D_Z$ values marked in Figure 3 represent the difference between source and destination particle positions in each dimension, as calculated during distance calculation. These values are passed, together with the inverse radius squared value, as inputs to the LJPC, and are used by LJPC in the calculation and accumulation of force per particle in each dimension, represented as $F_X$, $F_Y$, and $F_Z$. Total potential energy is also accumulated for the entire set of particles. Figure 3 shows the data flow graph for the LJPC. The implementation approach for the LJPC module varies from that used for the DC unit, in that

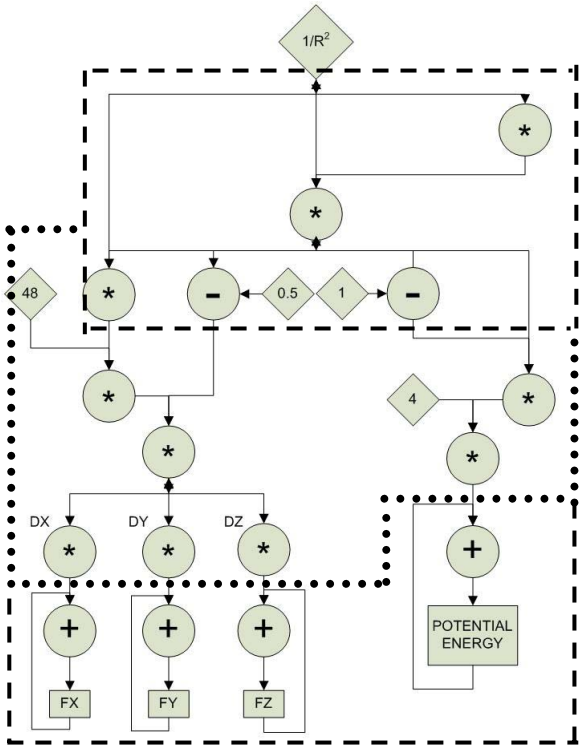


**Figure 3. LJPC unit data flow graph.**

we do not simply realize the flow graph in hardware with a 1 to 1 mapping, as was done in the DC unit. Instead, the LJPC data flow graph is broken into three parts. The three portions are shown in figure 3; distinguished by the three boxes. Each part of the data flow graph is associated with specific hardware. Thus, once data has moved from the first (top) area to the middle area, new data can be introduced to the computational units associated with the top area without any resource conflicts. If the entire LJP computation were treated as a single block, new data could only be input once old data had trickled all the way to the accumulators.

In other words, the LJP computation has been accelerated through use of a three-stage, high-level pipeline. In addition to high-level pipelining, each arithmetic unit is also pipelined. The unit with the shortest latency, the multiplier, is a 6-stage pipeline. Thus, 6 distinct sets of data can be processed at a time; assuming new data is available on each of 6 clock cycles.

### 3.4 FLEX Processor

The FLEX processor can be configured to perform the functionality of either the DC or the LJPC module. Because of this, the architecture must be general purpose enough to efficiently handle both data flow graphs. Using an iterative technique based upon force-directed scheduling, an architecture was derived which balances area consumption and throughput for both DC and LJPC computations. The result of this investigation was a FLEX unit that consists of 3 multipliers and 5 add/subtract units. The architecture is pure data flow, with delay registers inserted only at needed locations. This completely eliminates the need for register files and register file addressing of any sort whatsoever. Handling the DC operation is trivial, as there is a 1-to-1 mapping between nodes in the data flow graph and computational units in the processor. In LJPC mode, resources must be reused as there are more nodes in the data flow graph than computational units. Extensive multiplexing is used to control data flow between the computational units. The circuit is depicted in figure 4. Arithmetic units can receive data from external buffers, stored constants, or other arithmetic units. The circuit requires multiplexers on each input port of each arithmetic unit to provide this functionality. This architecture is based upon the LJPC algorithm being unrolled 7 times. If it is unrolled more than 7 times, the spatial and temporal mappings of operations to resources are disrupted. Once the bulk of the calculations are performed, four different adder trees are employed to take care of the four accumulate operations

After further analysis and debate, it was decided to change the data size from 32-bit floating point (8-bit exponent with 24-bit fraction) to 24-bit floating point (8-bit exponent with 16-bit fraction). This was done primarily to reduce the size of computational components on the FPGA to allow for everything to fit on a V4-FX60 device. We do recognize that this may impact the numerical accuracy of the result, but have not evaluated this aspect yet in detail.

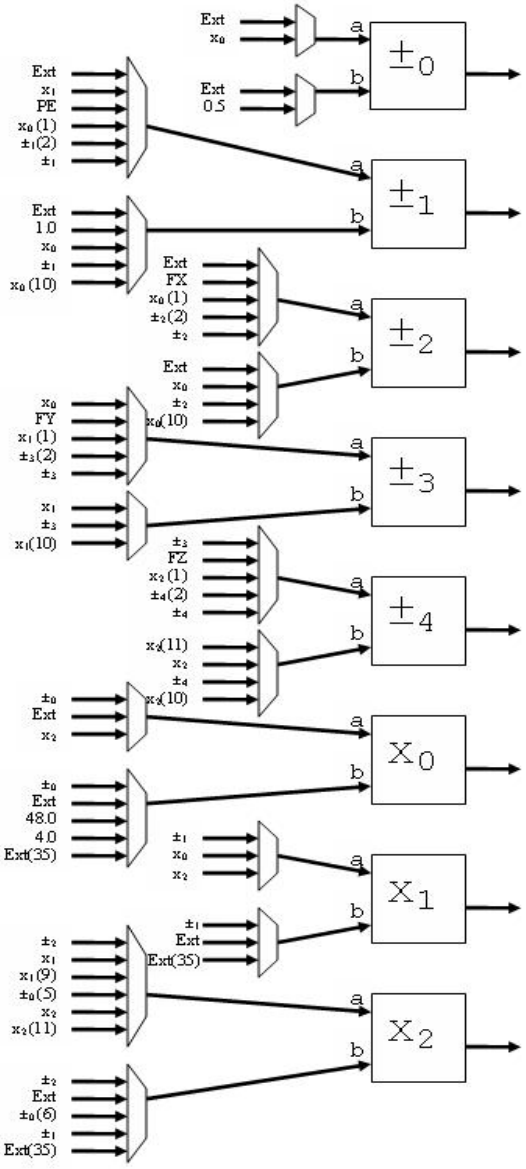


**Figure 4. FLEX architecture showing input multiplexers and computational units**

Rather than performing list scheduling on the LJPC algorithm, which results in a greedy schedule where everything is scheduled as soon as possible, more loop unrolls can be obtained if a "sub-optimal" scheduling technique is used, in which usage of specific arithmetic

units are spaced a uniform distance apart (equivalent to the shortest latency among all arithmetic units) on the first unroll. In the LJPC data flow graph, which consists of adders and multipliers, the shortest latency is 6 (multiplier). Thus, all events in the first loop unroll that are scheduled on the same resource must be scheduled a minimum of 6 cycles apart.

The FLEX processor is designed to execute one of two "programs", computing either DC results or LJPC results. An instruction consists of a set of input selects for all multiplexers on a given clock cycle and for specifying addition or subtraction for each of the add/subtract units. A 16-bit instruction word will be

sufficient to handle these options. The processor must be able to switch between programs rapidly and without extensive code loading or storage. Because of this, the FLEX program memory is constructed of individual ROMs, where each ROM is built of BRAM blocks. Each program is stored in a separate program memory. A multiplexer is used to determine which program is currently being executed. This system is shown in figure 5.

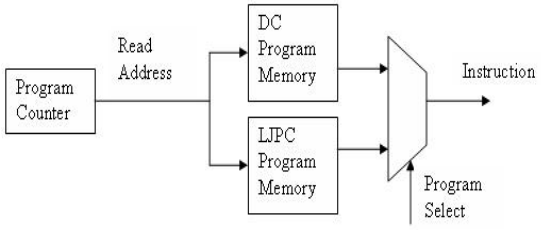


**Figure 5. FLEX program memory**

### 3.5 Inverter

The Inverter is designed to receive the squared distance value resulting from distance calculation, invert it, and then queue the result for LJPC. The Inverter design has two parallel inverters, six input FIFOs for receiving computed distances and six output FIFOs for disseminating inverted distances for LJPC. Each DC unit interfaced to the Inverter unit is assigned an Input FIFO, to which it queues its computed distance values. The Inverter unit's controller has a polling routine that checks to see if there are data in any of the Input FIFOs. There are priorities affixed to each Input FIFO, with those associated with DC units having priority over those associated with FLEX units. When valid data are retrieved from an Input FIFO, they are sent to the inverter. Once the distance has been inverted, it is sent to an Output FIFO. There, data is accumulated until the FIFO fills to at least 128 units, out of the max capacity of 1K units, where a unit consists of the inverted radius squared and the three dimensional distances values. Because the LJPC units cannot accept data on every clock cycle, Output FIFOs whose level exceeds 128 units have their contents transferred to the neighboring FLEX FIFO for processing. This provides further load-balancing among LJPC units. The controller monitors the levels of each FIFO and causes a switch from LJPC mode to DC mode when the FLEX FIFO level drops below 6 units, and from DC mode to LJPC mode when the FLEX FIFO level is above 128 units.

## 4 Results and Analysis

After synthesizing the design in Xilinx ISE 8.1i targeting a V4-FX 60 device, we simulated the behavior of the processor on a meta-data driven simulator to capture the behavior of all the modules in the system. We constructed a text file representing particle locations, and used that as input to the simulation. We then measured the time it took for each module in the program to complete and used this for our simulation results shown below. The simulation was run for 30,000 particles and with various cut-off radii to reflect numbers of particles within cut-off radii, starting from 10% of total particles falling within radius up to 90%. We noticed a plateau from 30% under radius to 70% where the amount of clock cycles necessary to complete processing remained very similar; around 14000. This is a result of load balancing and the ability of the FLEX unit to switch tasks. The results of this simulation can be seen in a plot shown in Figure 6.

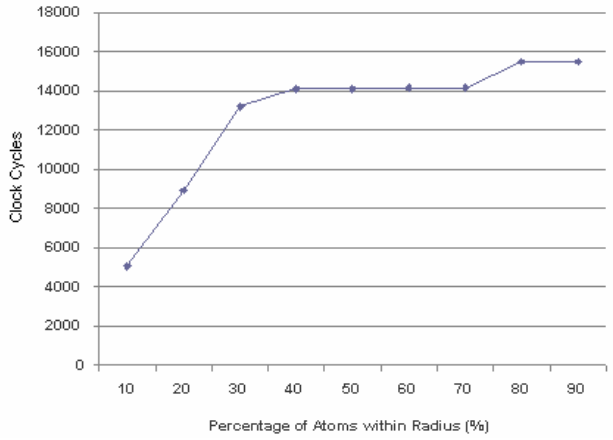


**Figure 6. Clock Cycles per Percentage of Particles in Radius**

We also monitored the Output FIFO levels (figure 7) in an effort to determine how efficiently our load-

balancing techniques were performing. This test used a 30% radius tolerance with a total of 30,000 atoms. The ability of this design to change dynamically based upon run-time conditions is clearly shown as capacity levels are indicative of the operational mode of the FLEX units.

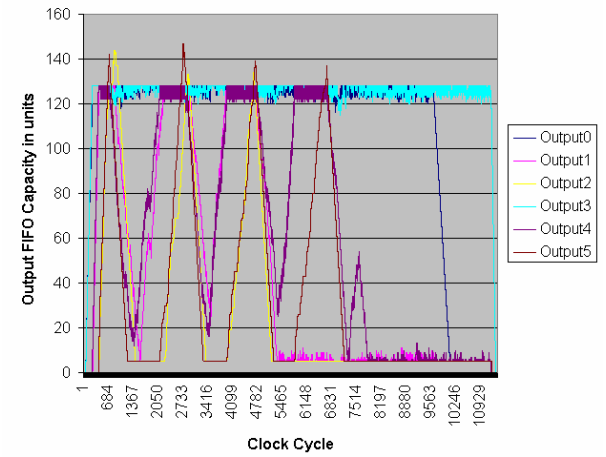


**Figure 7. Output FIFO module capacity levels**

During times were the capacities increase, the FLEX units are working in DC mode. Once the threshold is violated on an Output FIFO, the corresponding FLEX unit switches modes and you begin to see a decrease in the capacity level. This shows positive results and helps to indicate how important run-time load-balancing is in such problems as MD.

## 5 Conclusions and Future Work

We have identified three major areas for future research. Firstly we will focus on the design of the algorithms for updating velocities and positions. Secondly we will compare our method to the periodic boundary minimum image convention based population of initial data, as is adopted by GROMACS [6]. While this approach is suitable for crystal like structures, it is customized to enable parallelism on sequential machines where inter-processor communication is expensive. But at a preliminary glance it seems that our method is perhaps more generic and powerful to permit simulations of liquids, solutions etc, where introducing artificial periodic boundaries can affect the correctness of the simulation. Thirdly we will focus on streamlining the proposed methodology of data processing through the DC and LJPC units. If the FIFOs can pipe data in more efficiently and reduce the delay it takes to get the data processed, then overall efficiency will greatly increase, as FLEX units spend the majority of their time in LJPC operation mode. The logic of FIFO level monitoring can be distributed to more efficient controllers such as embedded Microblaze units. Through efficient use of these embedded processors we can introduce new functionality and likely increase performance of FLEX units. This will aid in overall load balancing. The FLEX unit's design will be closely examined, to see if there are any methods of optimizing the number of computational units. The intention is to achieve 1-1 mapping in order to increase effective use of the units that are there. This will decrease overhead and increase overall performance. We also plan to make use of our simulated results on FIFO levels in order to more optimally control FLEX modes and balance the load across all FIFOs more efficiently.

## References


[1] G. Lienhart, A. Kugel, and R. Manner, "Using floating-point arithmetic on FPGAs to accelerate scientific N-Body simulations," in *Field-Programmable Custom Computing Machines. Proceedings. 10th Annual IEEE Symposium on*, 2002, pp. 182-191.

[2] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, "Reconfigurable molecular dynamics simulator," Proc. *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. 2004, pp. 197-206.

[3] Y. Gu, T. VanCourt, and M. C. Herbordt, "Accelerating molecular dynamics simulations with configurable circuits," *Computers and Digital Techniques, IEE Proceedings-*, vol. 153, pp. 189-195, 2006.

[4] R. Scrofano and V. K. Prasanna, "Computing Lennard-Jones Potentials and Forces with Reconfigurable Hardware", Proc. *International Conference on Engineering of Reconfigurable Systems and Algorithms* 2004, pp.284-292.

[5] Scrofano, R.; Gokhale, M.; Trouw, F.; Prasanna, V.K., "Hardware/Software Approach to Molecular Dynamics on Reconfigurable Computers," *Field-Programmable Custom Computing Machines, 2006. FCCM '06. 14th Annual IEEE Symposium on* , vol., no.pp.23-34, April 2006

[6] http://www.gromacs.org