Accelerating Molecular Dynamics Simulations With Configurable Circuits*

Yongfeng Gu Tom VanCourt Martin C. Herbordt

Department of Electrical and Computer Engineering
Boston University; Boston, MA 02215 USA

*This paper is a postprint of a paper submitted to and accepted for publication to IEE Proceedings on Computers and Digital Technology and is subject to IEE copyright [URL]. The copy of record is available at [IEE Digital Libarary URL]. This work was supported in part by the National Institues of Health through award RR020209-01 and facilitated by donations from Xilinx Corporation.

Abstract

Molecular Dynamics (MD) is of central importance to computational chemistry. Here we show that MD can be implemented efficiently on a COTS FPGA board, and that speed-ups from 31× to 88× over a PC implementation can be obtained. Although the amount of speed-up depends on the stability required, 46× can be obtained with virtually no detriment, and the upper end of the range is apparently viable in many cases. We sketch our FPGA implementations and describe the effects of precision on the trade-off between performance and quality of the MD simulation.

1 Introduction

Computer simulation is increasingly being used to extend the reach of experimental biology and chemistry; Molecular Dynamics (MD) is one of the core methods in this approach [1]. In the last few years MD has become even more critical as it has been applied to modeling molecular interactions in drug design [2, 3], and to predicting molecule structure with applications to homeland security [4]. Particularly significant is that there are certain disease processes where a *primary* method of discovery is MD-based computer simulation. One example has been in finding the basis of amyloid diseases: MD was used to show that the pathogenic species of the offending proteins is the aggregation of misfolded *intermediates*; these are difficult to observe using experimental techniques [5, 6].

MD is an iterative technique that runs in phases: the forces on each atom (or molecule) are computed, then applied using equations of motion. Although modern force computations have become highly sophisticated (with 10 or more terms in some cases), the complexity generally resides in computing the van der Waals (Lennard-Jones or LJ) and Coulombic terms. These long-range forces are $O(N^2)$ in the number of particles N, while the motion updates are $O(N)$. The other forces, which only involve bonds, are also $O(N)$. Here we describe work in accelerating MD using FPGAs. We restrict our attention to the motion updates and the $O(N^2)$ force terms.

MD is a staple of high end computing with, for example, the IBM Blue Gene/L being developed, in part, for this application [7, 8]. MD has also been accelerated with special purpose hardware. Some well-known systems are MD-GRAPE [9], MD Engine [10] and the GRAPE-based Protein Explorer [11]. In the related area of N-Body simulations, special purpose hardware includes various iterations of GRAPE processors [12].

The flexibility and cost-effectiveness of FPGAs make them well suited for MD. FPGAs have always had the advantage of adaptability to new algorithms, in contrast to the inflexibility of ASICs. This has been used to advantage by the PRO-GRAPE [13] and PROGRAPE-3P1 [14] which are N-Body implementations with configurable forces, and the AHA-GRAPE [15] and GRAPE-RACE [16], which also support smoothed particle hydrodynamics. Recently, FPGAs have become significantly more powerful

relative to both ASICs and microprocessors. One factor is that FPGAs are currently driving, rather than lagging, process technology [17]. Another is that high-end FPGAs are now in fact hybrid chips with hundreds of ASIC components (especially multipliers and independently accessible memories) in addition to the configurable circuits. The most recent work concentrating on MD using FPGAs is a study by Azizi, et al. [18]. There, 2001-era FPGA technology was used to obtain performance similar to that of a 2004-era PC; this was extrapolated to a 20× speed-up by assuming hardware updates. The LJ force and a single atom type were implemented.

In this study we address the following questions.

- How competitive is current FPGA technology for a high performance application that is generally implemented with double precision floating point?
- Can MD be implemented cost-effectively on a commercial off-the-shelf (COTS) board?
- What is the cost of implementing the Coulombic as well as the LJ force?
- What are the issues with supporting multiple atom types?
- How efficiently can the macro blocks be used?
- Since forces may not need to be generalized, is reconfigurability still an advantage for MD?
- What is the trade-off between precision and computation?

Our primary result is that FPGA-based MD acceleration is likely to be many times more effective than previously indicated. We have obtained speed-ups of between 31× and 88× depending on the stability required, and the model of the FPGA hardware used: 46× can be obtained with virtually no detriment, and the upper end of the range is apparently viable in many cases. This is while using significantly more detailed force and particle models.

The primary significance is that a speed-up of two orders of magnitude is the oft-cited minimum for acceptance of non-standard computing technology. Also significant is that this can be achieved using a flexible COTS board; that it is FPGA-based means that the hardware can ride the technology curve for commodity chips and that the configured algorithms can be updated as new discoveries are made. Also interesting is that use of configurable hardware allows the use of precision as a design-space parameter for MD practitioners.

2 Molecular Dynamics Overview

In this section we give an overview of Molecular Dynamics simulations (MD), including the choice of boundary conditions. Another issue critical in hardware implementation of MD, precision and simulation quality, is discussed in Section 4. MD is an iterative application of Newtonian mechanics to ensembles of atoms and molecules. MD simulations generally proceed in phases, alternating between

force computation and motion integration. For motion integration, we follow our external serial reference code [19] in using the Verlet method (described, e.g. by Frenkel and Smit [20]).

In general, the forces depend on the physical system being simulated and may include van der Waals (Lennard-Jones or LJ), Coulomb, hydrogen bond, and various covalent bond terms:

$$\mathbf{F}^{total} = F^{bond} + F^{angle} + F^{torsion} + F^{HBond} + F^{non-bonded}$$

Because the hydrogen bond and covalent terms (bond, angle, and torsion) affect only neighboring atoms, computing their effect is $O(N)$ in the number of particles N being simulated. In coprocessor-based systems they are therefore generally computed by the host. The LJ force for particle i can be expressed as:

$$\mathbf{F}_i^{LJ} = \sum_{j \neq i} \frac{\epsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \mathbf{r}_{ji}$$

where the ϵ_{ab} and σ_{ab} are parameters related to the types of particles, i.e. particle i is type a and particle j is type b. The Coulombic force can be expressed as:

$$\mathbf{F}_i^C = q_i \sum_{j \neq i} \left(\frac{q_j}{|\mathbf{r}_{ji}|^3} \right) \mathbf{r}_{ji}$$

We implement both Coulombic and LJ forces; we also implement multiple atom types.

The choice of boundary conditions arises because, although the LJ force quickly goes to zero with distance, this is not the case with the Coulombic force. With periodic boundary conditions, the system of interest is replicated an infinite number of times. The (now) infinite summations are again made finite either by restricting the forces through a cut-off, or through the use of transform-based methods (Ewald, PME, PPPM). With spherical (e.g. stochastic) boundary conditions, as with the various cut-off methods, the computation is usually a direct summation of the force terms. Summations are also an important part of each transform-based method.

We briefly describe some of the issues involved (see, e.g. [21, 22] for surveys and comparisons of methods); this is necessary to justify the utility of our selection of the simpler method, albeit the one with the higher asymptotic complexity. The choice of boundary condition is a tradeoff between error and speed. Generally the quality of the choice is determined by the seriousness of the simulation artifacts introduced, something to which both periodic and non-periodic methods are susceptible. In a recent survey, Hansson et al. argue that although periodic methods are an elegant solution, the competing methods are also accurate [23]. Another study indicates that cut-off, even with a small radius, can be effective [24].

The issue of boundary conditions is important in the present work because *the relative computational complexity of the methods differs when implemented on an FPGA from when implemented on a PC, an MPP, or an ASIC-based computer (e.g. [25]).* In particular, algorithms are not equally effective across computing platforms. On a PC, transform-based techniques appear usually to be preferable; on large MPPs, transforms involve high communication overhead, so cutoff-based simulations are often appropriate [26]. On an FPGA, the problem size where transform-based techniques are preferable to direct computation may be much higher or even non-existent [27].

For the current study, we examine direct application of summations only, but with periodic boundary conditions. We select the direct method because it is likely to be a major part of most FPGA-based MD simulation, including those using transform-based methods, and because of the number of issues still to be resolved. While transform-based methods (especially PPPM) are likely eventually to be a part of FPGA/MD, some other MD methods are less so. Multipole algorithms tend to require very high precision. Multigrid and tree-code methods—the latter used in astronomical computations [28]—have substantially higher overhead, and require load balancing and non-uniform data structures. Also, MD differs substantially from the N-Body problem, so it is not surprising that different methods should be used [29].

3 Design

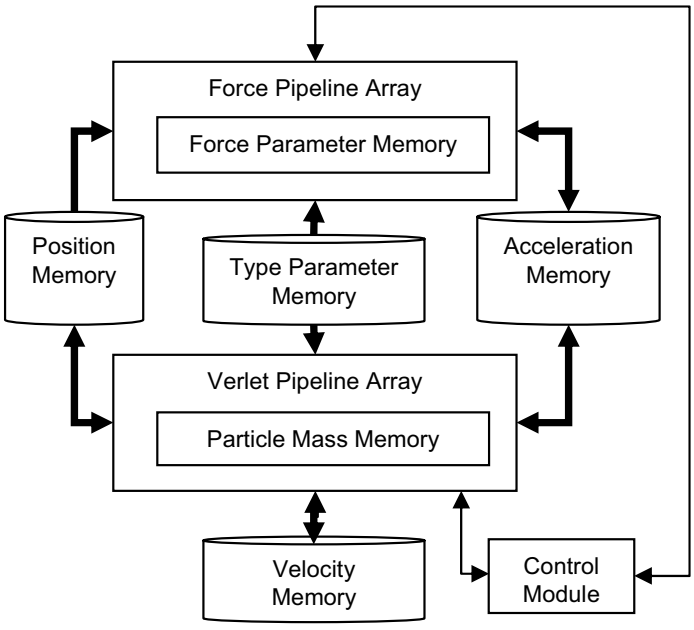

Figure 1: Block diagram of the FPGA parts of the system.

The high-level design is shown in Figure 1. As is common in MD hardware implementations, fixed

point is used. Our use of fixed point, however, should not be confused with the use of the integer data type. By appropriate initial selection of the units and by scaling the data as it flows through the hardware, the precision of the computation remains very close to the width of the datapath.

At the highest level, the computation core (shown in Figure 1) is itself wrapped by a communication layer to facilitate data transfer between the FPGA and the host PC via a PCI interface. Within the computation core, the Force Pipeline Array and the Verlet Pipeline Array are responsible for computing the forces on each particle and the motion updates, respectively. The two arrays, in turn, each contain a number of pipelines. Because of the inherent two-phase structure of the algorithm, the arrays work consecutively. There is therefore some sharing of hardware, especially multipliers, between the arrays. The various memories hold data as indicated. The Force Pipeline Array contains a pair-controller which, during each iteration, generates the addresses of the particle pairs and also accumulates the forces on each particle. On the Verlet side, each pipeline, on each iteration, takes the position, velocity, and acceleration data of one particle and outputs the updated motion parameters of another.

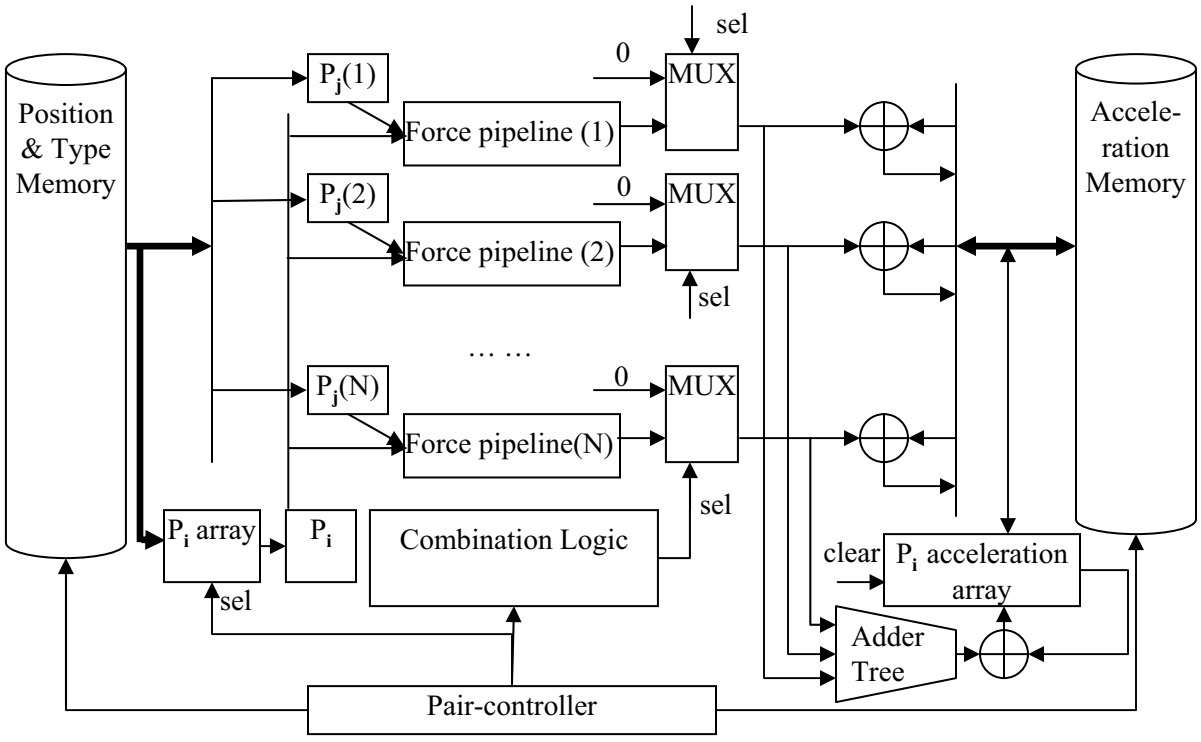

Figure 2: Block diagram of the force pipeline array.

Details of the Force Pipeline Array are shown in Figure 2. We first describe how dataflow is orchestrated and then the details of the force computation itself. The number of force pipelines N varies with the precision of the computation and the technology (as described further below) and is currently either 4 or 8. With N force pipelines, we can initiate computation for N pairs simultaneously, meaning that the data of N pairs must be fetched and stored each cycle. The data fetch of the computation

pairs can be viewed as a pair of nested loops with the inner loop unrolled N times and parallelized. The inner loop particle data are fed into the P_j registers and the outer loop data into the P_i register. The $i = j$ case is inhibited.

The forces are computed with look-up table and linear interpolation. The look-up table is indexed in three dimensions: P_i type, P_j type, and distance squared. Two memories are used, one for the even index entries, one for the odd index entries. This allows both ends of the interpolation to be fetched during a single cycle. Following the serial reference codes (described below), the table has 2K entries. The precision of the entries matches the precision of the datapath. The resolution of the table appears to be adequate, given the measurements shown in Figure 5.

Look-up tables for two particle types currently fit on-chip. For more particle types, tables are swapped as needed. However, since the particle types are known and the particles can be ordered *a priori*, this swapping is usually possible without requiring stalls.

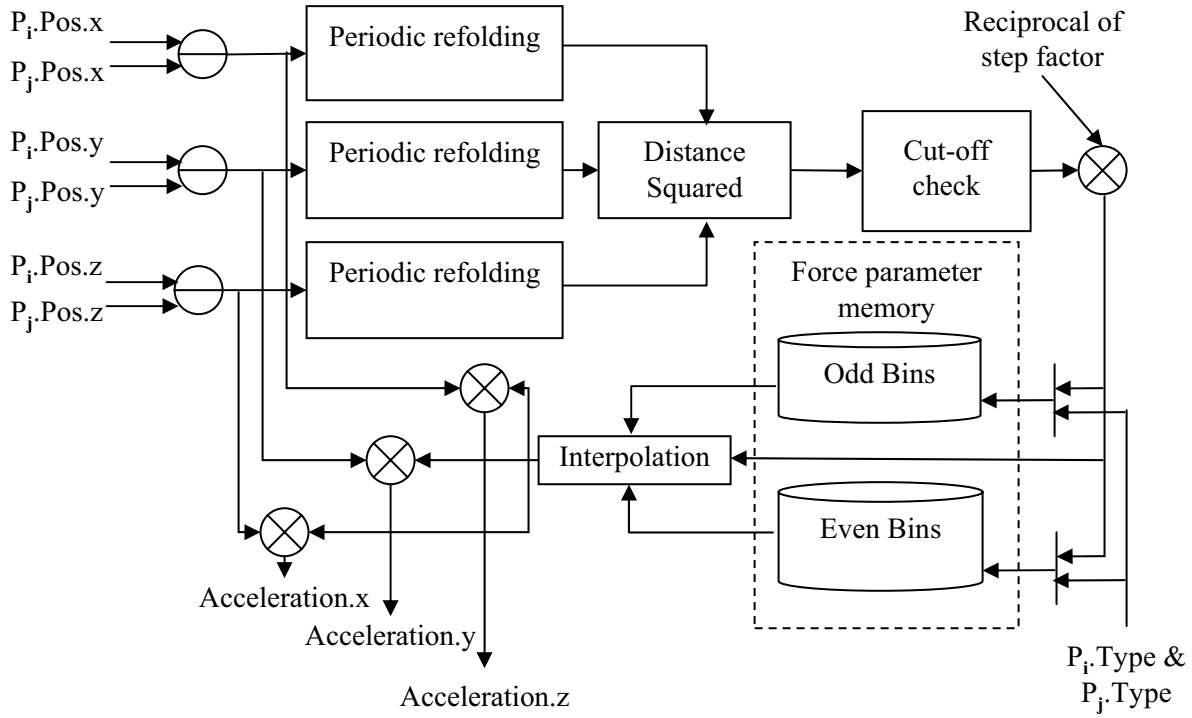

Figure 3: Detail of a single force pipeline.

Details of the individual force pipelines are shown in Figure 3. Each pipeline has 28 stages that can be grouped into 8 functions:

1. Compute the displacement in each dimension.
2. Perform periodic boundary refolding if necessary.
3. Compute the square of the distance between particle pairs.

4. Check the distance. If the distance squared is out of range, a special index is used for table lookup.
5. Divide the distance squared by the bin size to get the index for the force table. The division is done by multiplying the reciprocal of the bin size.
6. Look up the force parameter based on the types of the particles and the distance squared.
7. Do linear interpolation on the force parameter.
8. Multiply the interpolated force parameter by the displacement vector of the particle to get the force.

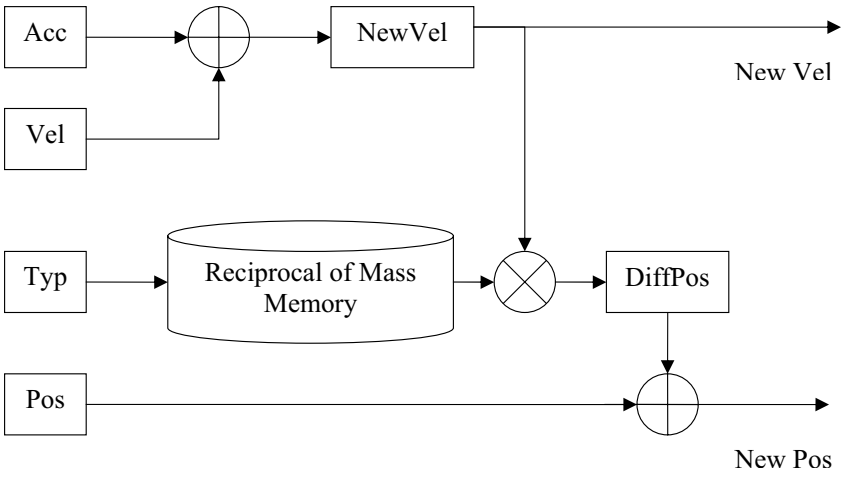

Figure 4: Shown is a Verlet update block.

The Verlet update pipeline is shown in Figure 4. For computational simplicity, the standard equations are reordered into the following:

$$vel(t+1) = vel(t) + acc(t) * dt$$

$$pos(t+1) = pos(t) + vel(t+1) * dt$$

However, in the implementation, the mass is not taken into account until the update phase so in the first equation, velocity is actually the momentum and acceleration the force. Rewriting, we obtain:

$$momentum(t+1) = momentum(t) + force(t)$$

$$dp = momentum(t+1) * 1/mass$$

$$position(t+1) = position(t) + dp$$

Since there is no interaction between particles in this phase, the implementation is straightforward with an eight stage pipeline.

4 Precision versus Quality of MD Simulations

It is well known that for particular applications, FPGA implementations can achieve speed-ups of 1000× or more. These applications are characterized by high parallelism, which can be translated into high circuit utilization. They are usually also characterized by low-precision data where the FPGA implementation can trade off datapath width for an increased number of function units. Probably for this reason, many researchers have avoided applications that are require double precision floating point, including MD.

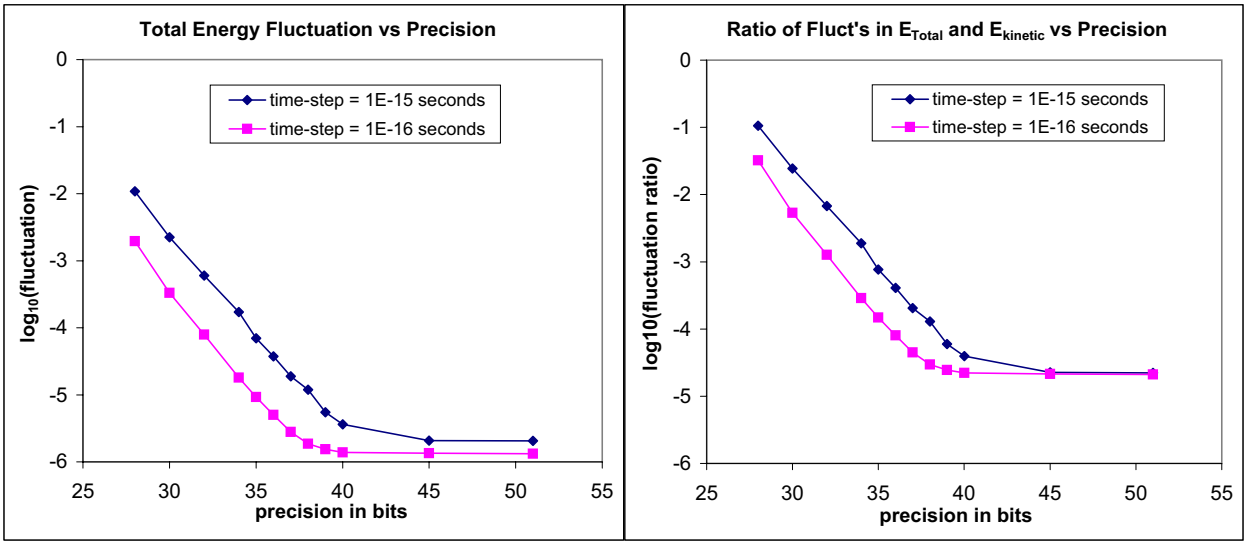

Figure 5: Shown is the effect of precision on two metrics for simulation accuracy: (a) Fluctuation of total energy and (b) the ratio of the fluctuations in total and kinetic energies. Simulations were carried out with two different time-steps.

We believe that a central area of research in FPGA-based acceleration is analyzing applications to see whether double precision floating point is actually needed, or whether it is simply used because it has little marginal performance cost on contemporary microprocessors [30, 27]. A well-known study by Amisaki, et al. [31] investigated precision required for MD; they showed that certain important measures relevant to MD simulation quality do not suffer when precisions of various intermediate data are reduced from 53 bits to 25, 29, and 48 bits, respectively. A more extreme observation was made by La Penna, et al. who write that "in our very long simulations we did not see signs of instabilities nor of any systematic drift" due to using single, rather than double precision floating point [32]. Clearly, though, this last reference is not the consensus. Now looking at this from the point of view the user, Rapaport states that "obtaining a high degree of accuracy in the trajectories is neither a realistic nor a practical goal," as "the most minute errors grow exponentially with time [33]."

The issue of exactly how much precision is required for which particular MD simulations has not

been well-studied.[1] This is precisely because MD implementations are nowadays almost universally run on machines where there is little incentive to *not* using double precision.

For implementations on configurable circuits, however, the situation is quite different. If it is possible to reduce the precision without appreciably changing the quality of the simulation, then it is possible to increase the computational resources that can be applied. This in turn should result in substantially better overall performance.

One classic check for simulation quality is to measure the fluctuation of physical quantities that should remain invariant, such as energy. The relative rms fluctuation in total energy is defined as:

$$\frac{\sqrt{|\langle E^2\rangle - \langle E\rangle^2|}}{|\langle E\rangle|}$$

We ran a set of experiments based on two versions of serial reference code, reproducing as closely as possible the experiments done by Amisaki et al. [31]. The first used double precision floating point, the second tracked the hardware implementation using varying precision. When the precision of the fixed point code was set at 50 bits, the results precisely matched that of the floating point code.

We also ran a set of experiments to find the relationship between energy fluctuation and precision. In agreement with [31], we found that the various function units can be tuned independently to derive the optimal FPGA circuits that retain minimal energy fluctuation. For simplicity, however, we present results where the precision of the entire datapath is varied in unison. We use two different simulation time scales: time steps were set to E-15 seconds and E-16 seconds, respectively. A graph showing the results from this set of experiments is shown in the left part of Figure 5. One observation is that, in this experiment, a 40-bit datapath results in a similarly low energy fluctuation as a full 53-bit datapath.

Fluctuation of total energy, however, is not the only check that a system is "well-behaved." Another is the ratio of the fluctuations between total energy and kinetic energy $R = \Delta E_{total}/\Delta E_{kinetic}$. R should be less than .05 [35]. We plot R in the right half of Figure 5. Note that by this measure, 31 bits are sufficient for time-steps of E-15 seconds and 30 bits are sufficient for time-steps of E-16 seconds. Although greater precision results in "better" behavior, that better behavior *may not be needed.*

At this point we inject into the discussion the attributes of the target technology, a high-end 2004-era FPGA. A number of implementation factors (such as the number of block RAMs and hard multipliers, indexing issues, etc.) lead to the observation that there are two sweet spots in the design space: (1) 4 force pipelines with nearly full precision (51-bit), and (2) 8 force pipelines with 35-bit precision. In the first implementation, behavior is equivalent to double precision floating point. In the

[1] Of course the opposite question of what to do when double precision appears to be inadequate *is* a fundamental issue (see e.g. [34]). The general solution is to increase the resolution of the time steps.

Table 1: Results related to various MD implementations. "VP70 AMS" refers to actual timing from the Annapolis Microsystems Wildstar board with a Xilinx Virtex-II-Pro XC2VP70 -5 FPGA. "VP100 sim" refers to timing derived from simulation only, assuming a Xilinx Virtex-II-Pro XC2VP100 -6 FPGA. Speed-up is with respect to a PC with a 2.4GHz Xeon CPU.

Platform	Precision (bits)	Pipe-lines	HW mult's used (% of usage)	Block RAMs used (% of usage)	Delay (ns)	Speed-up
VP70 AMS	35	4	176(53%)	214(65%)	11.1	50.8×
VP70 AMS	40	4	264(80%)	251(77%)	12.2	46.4×
VP70 AMS	45	4	288(88%)	285(87%)	13.2	42.7×
VP70 AMS	51	4	288(88%)	317(97%)	18	31.3×
VP70 AMS	35	8	256(78%)	326(99%)	22.2	51.0×
VP100 sim	51	4	288(65%)	317(77%)	13.6	41.5×
VP100 sim	35	8	256(58%)	334(75%)	12.8	88.5×

second implementation, quality depends on the metric. The 35-bit design has from a factor of 10× to 50× more energy fluctuation than the best case, but between 100× and 500× lower R than what has been regarded as minimal to indicate "good behavior." It appears that this may be a case where there is a large difference between "good enough" and "best possible."

Until now, MD users have almost never had the choice of precision: either double precision was good enough or it was not. If it was not, then some other quantity, such as time-step, needed to be varied. With implementations on configurable circuits it is possible to do the reverse: trade off unneeded precision for computing resources.

5 Implementation, Validation, and Results

The design was implemented on a WildstarII-Pro board from Annapolis Micro Systems. The board has two Xilinx Virtex-II-Pro XC2VP70 -5 FPGAs (referred to as *VP70 AMS* in Table 1); however, only one of the FPGAs was used. Some designs were also implemented in simulation-only on a Xilinx Virtex-II-Pro XC2VP100 -6 FPGA (referred to as *VP100 sim* in Table 1).

The critical path originally ran through the hard multipliers but this has now been optimized. For example, for the 40-bit multipliers, instead of using three hard multipliers with 25ns latency, we use nine hard multipliers with 9ns pipelined latency. The fact that 35- and 51-bit datapaths are preferred on the Virtex-II-Pro FPGAs is an artifact of the hard multiplier format (18-bit signed digit).

The VP70 implementations all hold 8K particles on chip. Larger simulations require off-chip memory access. However, the deterministic nature of the computation and the tremendous off-chip memory bandwidth of the Virtex-II-Pros makes running these larger simulations with no slowdown straightfor-

ward.

The FPGA board interacts with the host PC using DMA transfers accessed with system calls from the Annapolis Micro Systems software support library. In this study, however, the entire computation is performed by the FPGA board; that is, once the computation is initiated, no further interaction with the host is necessary. If further force terms are required (e.g., the bonded forces), then these would likely be computed on the host. In this case, motion update (also being on $O(N)$ computation) would be moved to the host as well. In this scenario, interactions between host and coprocessor would take place once per iteration. The amount of data transferred would be small, however, as only N vectors would need to be exchanged.

The critical resources on the FPGA are the hard multipliers, the registers and, in particular, the block RAMs. All block RAMs are simultaneously read from and written to on every cycle, for an aggregate memory bandwidth of 2Tb/s. The block RAM bandwidth bounds the number of pipelines for a given precision, while the block RAM size limits the number of look-up tables and particles that can be held on-chip. Harder to gauge is the relationship between slices used and design complexity. The Synplicity synthesis tool, which we used, appears to be excellent at trading off slices for performance as a large fraction of slices was used in every implementation.

The design was validated against three serial reference codes, an external double precision floating point code MD2DLJ [19], and two versions our own code (fixed and float) that tracks the hardware implementation. Validation has several parts. First, the hardware tracker matches MD2DLJ exactly when the hardware tracker has the same floating point datapath. Second, the fixed-point hardware tracker exactly matches the FPGA implementations. The missing link is the relationship between the fixed-point and floating point versions of the hardware tracker. These can only be compared indirectly, however, as is done in the previous section.

We have created several variations of these designs; three are of particular interest: 35-bit with eight pipelines, 40-bit with four pipelines, and 51-bit with four pipelines. The reasoning is as follows. Recall from the previous section that datapath sizes of 30 bits, 40 bits, and 51 bits are required to obtain adequate R, best E_{fluct}, and performance virtually indistinguishable from double precision floating point, respectively. We replace the 30-bit datapath with a 35-bit datapath because it uses virtually the same hard resources and substantially improves R and E_{fluct}. On the other hand, going from a 51-bit datapath to a 53-bit datapath (to equal the precision of double precision floating point) requires substantially more hard multipliers, but for little benefit.

Results are shown in Table 1. The 51-bit four-pipeline and the 35-bit eight-pipeline implementations are aggressive for the VP70. They use such a high fraction of the chip resources that there is a substantial reduction in operating frequency. We therefore also synthesized these for the VP100: the numbers shown

are post place and route.

For timing reference, the serial reference codes—the double precision floating point version of the hardware tracker and MD2DLJ, which is also double precision floating point—were compiled using the Microsoft VC++ compiler VC6.0 with standard optimisation settings. Both floating point reference codes ran at 9.5s per MD time-step on a PC with a 2.4GHz Xeon CPU. This is similar to the 10.8s for a 2.4GHz P4 described in [18]. Translating into force computations per cycle, the FPGA implementation completes one per cycle per pipeline, while the PC completes roughly one per 250 cycles. The large disparity is a reminder that most FPGA function units compute payload every cycle, while a large fraction of CPU cycles are spent with overhead. In particular, the MD2DLJ code computes 109 floating point operations per force computation, yielding a utilization of just over 20% of the theoretical peak of 4.8Gflops. Neither the serial code, nor the FPGA codes, were optimised beyond taking care to follow good design practices. However, both the serial codes do perform the force computation using a table look-up, saving many floating point operations over a direct implementation.

Our other serial reference code, the hardware tracker that follows the FPGA implementation, is implemented using a dedicated class library. This class implements the hybrid float/fixed arithmetic by overloading the operators, such as +,-,*, shifts, logic, and comparisons. Since the class simulates the hardware pipeline bit by bit, it runs orders of magnitude slower than the two double precision floating point reference codes.

6 Discussion, Optimizations, and Extensions

In this study we have demonstrated the competitiveness of FPGAs for molecular dynamics. The significance, beyond that of the application alone, is in showing substantial acceleration in a double precision floating point code. Other results show how simulation accuracy can be traded off for performance, and how to deal with multiple forces and atom types.

As always when measurements are done with respect to rapidly advancing technology, all numbers reported here are transient. As FPGAs appear to be following Amdahl's law just as much as are microprocessors [36, 17], these trends are likely to continue for some time. For the MD application, this is currently to the benefit of the FPGA designs: increased resources can be immediately applied to the computation by adding pipelines; the benefits for microprocessors have usually been less direct. The emergence of multicore chips should give future microprocessors the same capability as FPGAs of direct scalability, but the primary point remains unchanged.

Another axis of variation between microprocessor and FPGA implementations is design effort. Given a few months effort by experienced FPGA designers and assembly language programmers, both

FPGA and reference codes could perhaps be improved substantially. We believe, however, that this would not change the basic fact that nearly two orders of magnitude speed-up can be obtained by using an FPGA.

We have shown that an FPGA implementation of a basic molecular dynamics code can be accelerated from 31× to 88× with respect to a PC implementation, depending on the size of the FPGA and the simulation accuracy. These comparisons can be extended to estimate the effects of cost, parallel processing, and more complex software.

Currently, the part costs of high-end FPGAs and microprocessors are comparable. The cost of FPGAs packaged into plug-in boards ranges from comparable to an inexpensive server blade, to many times higher. Total cost of ownership, however, may be much lower for an FPGA/PC combination than a 32- or 64-node cluster once system administration, space, power, and other factors are considered. Low-power packaging of PC clusters [37] is addressing many of these issues, but costs remain at around $1,000/processor. On the FPGA side, low-cost multi-FPGA systems currently being developed (e.g., with the BEE project [38]), and may swing the balance back towards FPGAs.

With respect to parallel processors: In our comparisons, we have implicitly assumed perfect scalability of the serial reference codes. Given the work, e.g. on NAMD [39], this is likely to be valid up to at least several hundred processors.

Current MD codes are significantly more complex than what has been demonstrated here. One typical optimisation is to simulate the $O(N)$ forces, which have much shorter time constants than the $O(N^2)$ forces, at a similarly higher rate. As the $O(N)$ forces would likely be computed on the host, using multiple time-scales would somewhat decrease the speed-up from that presented here. But with $N = 10,000$ or more, the impact of a 10× increase in the serial portion of the compute time may not be great.

We stated earlier that MD and N-Body simulations are different enough for it to be unlikely that the same method would be preferable for both FPGA implementations. Given the configurability of the FPGA, there is no need to use a generalized N-Body implementation when a tuned MD configuration is available. The converse is also true. MD implementations, where the forces are computed using look-up tables, lend themselves to more general computations by replacing the table entries. Again, production N-Body codes are likely to rely heavily on other methods, such as tree-codes.

The most important next task is to integrate our MD implementations into production codes. Work using ProtoMol [40] is substantially complete; work using NAMD [29] is underway.

An obvious extension involves using completely different algorithms, in particular those based on Ewald sums or other transform-based methods (especially PPPM): with the current work being done on

FFTs for FPGAs, this might happen soon. However, as per our discussion above, it is far from certain that this will result in improved results. Intriguing is the possibility of using the second FPGA on our Wildstar board for this computation while retaining most of the original design on the first.

Acknowledgments

We would like to thank the anonymous reviewers for their many helpful suggestions.

References

[1] P. I. T. A. Committee, *Computational Science: Ensuring America's Competitiveness*. http://www.nitrd.gov: National Coordination Office for Information Technology Research and Development, 2005.

[2] R. Friesner and et al., "Glide: A new approach for rapid, accurate docking and scoring. 1. method and assessment of docking strategy," *Journal of Medicinal Chemistry*, vol. 47, pp. 1739–1749, 2004.

[3] M. Taufer, M. Crowley, D. Price, A. Chien, and C. B. III, "Study of a highly accurate and fast protein-ligand docking algorithm based on molecular dynamics," in *Proceedings of the International Workshop on High Performance Computational Biology*, 2004.

[4] Lawrence Livermore National Labs, "The art of protein structure prediction," *Science and Technology Review*, vol. December, 2004.

[5] M. DeMarco and V. Daggett, "From conversion to aggregation: Protofibril formation of the prion protein," *Proc. Nat. Acad. Sci.*, vol. 101, no. 8, pp. 2293–2298, 2004.

[6] ——, "Local environmental effects on the structure of the prion protein," *Comptes Rendus Biologies*, vol. Article in Press, 2005.

[7] F. Allen and et al., "Blue Gene: a vision for protein science using a petaflop supercomputer," *IBM Systems Journal*, vol. 40, no. 2, pp. 310–327, 2001.

[8] B. Fitch and et al., "Blue matter: Strong scaling of molecular dynamics on blue gene/l," IBM Research Division, Tech. Rep. RC23688 (W0508-035) Computer Science, 2005.

[9] Y. Komeiji, M. Uebayasi, R. Takata, A. Shimizu, K. Itsukashi, and M. Taiji, "Fast and accurate molecular dynamics simulation of a protein using a special-purpose computer," *J. Comp. Chem.*, vol. 18, no. 12, pp. 1546–1563, 1997.

[10] S. Toyoda, H. Mihagawa, K. Kitamura, T. Amisake, E. Hashimoto, H. Ikeda, A. Kusumi, and N. Miyakawa, "Development of md engine: High-speed accelerator with parallel processor design for molecular dynamics simulations," *Journal of Computational Chemistry*, vol. 20, no. 2, pp. 185–199, 1999.

[11] M. Taiji, T. Narumi, Y. Ohno, N. Futatsugi, A. Suenaga, N. Takada, and A. Konagaya, "Protein Explorer: A petaflops special-purpose computer system for molecular dynamics simulations," in *Supercomputing*, 2003.

[12] A. Kawai, T. Fukushige, and J. Makino, "$/mflops astrophysical n-body simulation with treecode on GRAPE-5," in *Supercomputing*, 1999.

[13] T. Hamada, T. Fukushige, A. Kawai, and J. Makino, "PROGRAPE-1: A programmable, multi-purpose computer for many-body simulations," *Publ. Astronomical Society of Japan*, vol. 52, pp. 943–954, 2000.

[14] T. Hamada and N. Nakasato, "Massively parallel processors generator for reconfigurable system," *Proceedings of the Conference on Field-programmable Custom Computing Machines*, 2005.

[15] T. Kuberka, A. Kugel, R. Manner, H. Singpiel, R. Spurzem, and R. Klessen, "AHA-GRAPE: Adaptive hydrodynamic architecture – GRAvity PipE," in *Proceedings of Field Programmable Logic and Applications*, 1999.

[16] R. Spurzem, J. Makino, T. Fukushige, G. Lienhart, A. Kugel, R. Manner, M. Wetzstein, A. Burkert, and T. Naab, "Collisional stellar dynamics, gas dynamics and special purpose computing," in *Proc. Int. Symp. on Computation Science and Engineering*, 2002.

[17] N. Tredennick, "Reconfigurable systems emerge," Keynote Talk, Int. Conf. on Field Programmable Logic and Applications, August 2004.

[18] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, "Reconfigurable molecular dynamics simulator," in *Proceedings of the Conference on Field-programmable Custom Computing Machines*, 2004, pp. 197–206.

[19] M. Bargiel, W. Dzwinel, J. Kitowski, and J. Moscinski, "C-language molecular dynamics program for the simulation of Lennard-Jones particles," *Computer Physics Communication*, vol. 64, pp. 193–205, 1991.

[20] D. Frenkel and B. Smit, *Understanding Molecular Simulation*. New York, NY: Academic Press, 2002.

[21] P. Gibbon and G. Sutmann, "Long-range interactions in many-particle simulation," in *Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms*, J. Grotenhorst, D. Marx, and A. Murmatsu, Eds. John von Neumann Institue for Computing, NIC Series, Vol. 10, 2002.

[22] C. Sagui and T. Darden, "Molecular dynamics simulations of biomolecules: Long-range electrostatic effects," *Annual Review of Biophysical and Biomolecular Structures*, vol. 28, pp. 155–179, 1999.

[23] T. Hansson, C. Oostenbrink, and W. van Gunsteren, "Molecular dynamics simulations," *Current Opinion in Structural Biology*, vol. 12, pp. 190–196, 2002.

[24] D. Beck, R. Armen, and V. Daggett, "Cutoff size need not strongly influence molecular dynamics results for solvated polypeptides," *Biochemistry*, vol. 44, pp. 609–616, 2005.

[25] T. Fukushige, J. Makino, T. Ito, S. Okumura, T. Ebisuzaki, and D. Sugimoto, "WINE-1: Special-purpose computer for n-body simulations with periodic boundary conditions," *Publ. Astronomical Society of Japan*, vol. 44, pp. 361–375, 1993.

[26] L. Kale and et al., "NAMD2: greater scalability for parallel molecular dynamics," *Journal of Computational Physics*, vol. 151, pp. 283–312, 1999.

[27] T. VanCourt, M. Herbordt, and R. Barton, "Microarray data analysis using an FPGA-based coprocessor," *Microprocessors and Microsystems*, vol. 28, no. 4, pp. 213–222, 2004.

[28] E. Athanassoula, A. Bosma, J.-C. Lamber, and J. Makino, "Performance and accuracy of a Grape-3 system for collisionless n-body simulations," *Journal of VLSI Design*, 1994.

[29] R. Brunner, J. Phillips, and L. Kale, "Scalable molecular dynamics for large biomolecular systems," in *Supercomputing*, 2000.

[30] T. VanCourt, M. Herbordt, and R. Barton, "Case study of a functional genomics application for an FPGA-based coprocessor," in *Proceedings of Field Programmable Logic and Applications*, 2003, pp. 365–374.

[31] T. Amisaki, T. Fujiwara, A. Kusumi, H. Miyagawa, and K. Kitamura, "Error evaluation in the design of a special-purpose processor that calculates nonbonded forces in molecular dynamics simulations," *Journal of Computational Chemistry*, vol. 16, no. 9, pp. 1120–1130, 1995.

[32] G. L. Penna, S. Letardi, V. Minicozzi, S. Morante, G. Rossi, and G. Salina, "Parallel computing and molecular dynamics of biological membranes," *ArXiv Physics eprints*, vol. Physics/99709024, 1997.

[33] D. Rapaport, *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 2004.

[34] J. Haile, *Molecular Dynamics Simulation*. New York, NY: Wiley, 1997.

[35] D. van der Spoel, "Gromacs exercises," CSC Course, Espo, Finland, February 2004.

[36] M. Butts, "Molecular electronics: All chips will be reconfigurable," Tutorial, 13th Int. Conf. on Field Programmable Logic and Applications, September 2003.

[37] *BioServer*, Fujitsu Computer Systems, www.fujitsu.com, 2005.

[38] C. Chang, K. Kuusilinna, B. Richards, and R. Broderson, "Implementation of BEE: A real-time large-scale hardware emulation engine," in *Proc. FPGA*, 2003.

[39] J. C. Phillips and et al., "Scalable molecular dynamics with namd," *Journal of Computational Chemistry*, vol. 26, pp. 1781–1802, 2005.

[40] T. Matthey, "ProtoMol, an object-oriented framework for prototyping novel algorithms for molecular dynamics," *ACM Transactions on Mathematical Software*, vol. 30, no. 3, pp. 237–265, 2004.