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
herbordt|maplegu|tvancour@bu.edu http://www.bu.edu/caadlab

ABSTRACT

Molecular Dynamics (MD) is of central importance to computational chemistry. Here we show that MD can be implemented ef£ciently on a COTS FPGA board, and that speed-ups from $31\times$ to $88\times$ over a PC implementation can be obtained. Although the amount of speed-up depends on the stability required, $46\times$ 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

Molecular Dynamics (MD) is a fundamental part of computational chemistry. In the last few years MD has become, if anything, even more critical as it has been applied to modeling molecular interactions in drug design (see e.g. [1]), and to predicting molecule structure with applications to homeland security.

MD is an iterative technique that runs in phases: the forces on each atom (bzw. 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 $O(N)$, and the other forces–which only look at 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 an obvious candidate for acceleration with special purpose hardware (see e.g. [2, 3]). In the study by Azizi, et al. [2], 2001-era FPGA technology was used to obtain performance similar to that of a 2004-era PC; this was extrapolated to a 20x speed-up by assuming hardware updates.

THIS WORK WAS SUPPORTED IN PART BY THE NIH THROUGH AWARD #RR020209-01 AND FACILITATED BY DONATIONS FROM XILINX CORPORATION.

Our work differs from previous approaches in that we combine the following: on the hardware side, that we use a COTS board; on the implementation side, that we model the Coulombic as well as the LJ term, and that we support the simultaneous modeling of multiple types of molecules. Perhaps most interesting for continued FPGA investigations, we also have investigated precision/accuracy trade-offs.

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 $31\times$ and $88\times$ depending on the stability required, and the model of the FPGA hardware used: $46\times$ can be obtained with virtually no detriment, and the upper end of the range is apparently viable in many cases. This is while using signi£cantly more detailed force and particle models.

The primary signi£cance is that a speed-up of two orders of magnitude is the oft-cited minimum for initial acceptance of non-standard computing technology. Also signi£cant is that this can be achieved using a ¤exible COTS board; that it is FPGA-based means that the hardware can ride the technology curve for commodity chips and that the con£gured algorithms can be updated as new discoveries are made. Also interesting is that use of con£gurable hardware may allow the use of precision as a design-space parameter for MD practitioners.

2. MOLECULAR DYNAMICS OVERVIEW

Molecular Dynamics simulations generally proceed in phases, alternating between force computation and motion integration. For motion integration, we use the Verlet method (described, e.g. by Frenkel and Smit [4]).

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. Because the hydrogen bond and covalent terms 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, or in the case of coprocessors based on Xilinx Virtex-Pro FPGAs, by the on-board microprocessor. 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 LJ force quickly goes to zero with distance; this is not the case with the Coulombic force. There are many ways to set the boundary conditions to deal with the long range effect; these can be classi£ed as being either non-periodic or periodic. If one of the former is chosen, the Coulombic force is computed in a manner analogous to the LJ force; if the latter, then FFT-based or multigrid methods can be used. We brie¤y describe the issues involved; 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. A recent summary by Hansson et al. [5] argues that although periodic methods are an elegant solution, the competing methods are also accurate as they show by referencing a number of recent studies.

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 or a supercomputer.* In particular, algorithms are not equally effective across computing platforms. On a supercomputer, FFT-based techniques appear usually to be preferable; on an FPGA, the problem size where transform-based techniques are preferable to direct computation may be much higher or even non-existent [6].

3. DESIGN

The high-level design is shown in Figure 1. As is common in MD hardware implementations, £xed point is used. Our use of £xed point should not be confused with the use of the integer data type, however. By appropriate initial selection of the units and by scaling the data as it ¤ows 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

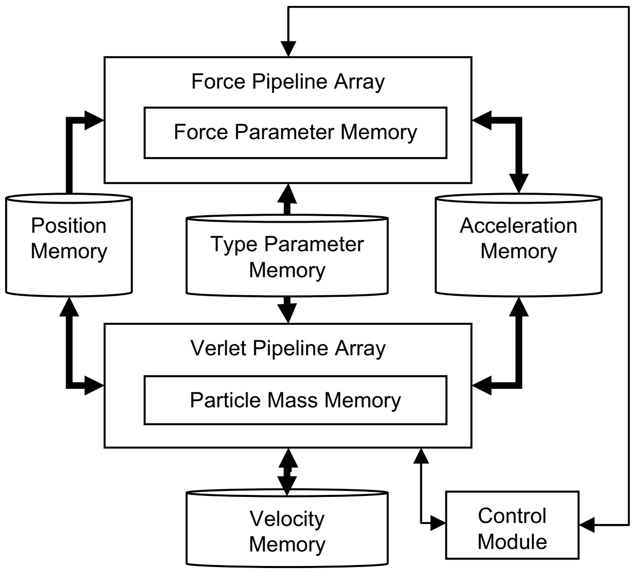

Fig. 1. Block diagram of the FPGA parts of the system.

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 (described below). 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.

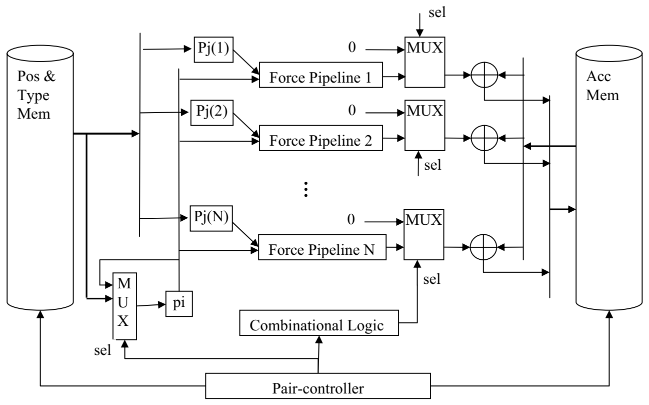

Fig. 2. Block diagram of the force pipeline array.

Details of the Force Pipeline Array are shown in Figure 2. We £rst describe how data¤ow 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.

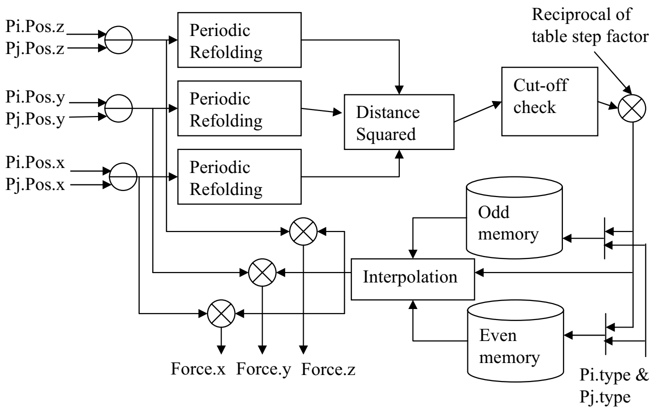

Fig. 3. Detail of a single force pipeline.

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.

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

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

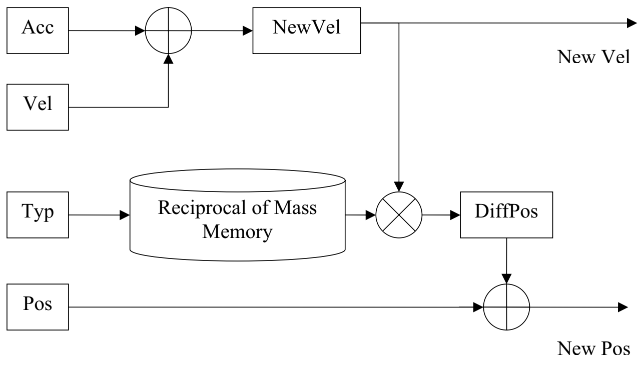

Fig. 4. Shown is a Verlet update block.

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

However, in the implementation, the mass is not taken into account until the update phase so in the £rst 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

It is well known that for particular applications, FPGA implementations can achieve speed-ups of $1000\times$ 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, researchers have avoided applications that are “canonically” double precision ¤oating point, including MD. Recent exceptions include [2, 7].

However, we believe that a central area of research in FPGA-based acceleration is analyzing applications to see whether double precision ¤oating point is actually needed, or whether it is simply used because it has little marginal performance cost on contemporary microprocessors [8, 6]. A well-known study by Amisaki, et al. [9] 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

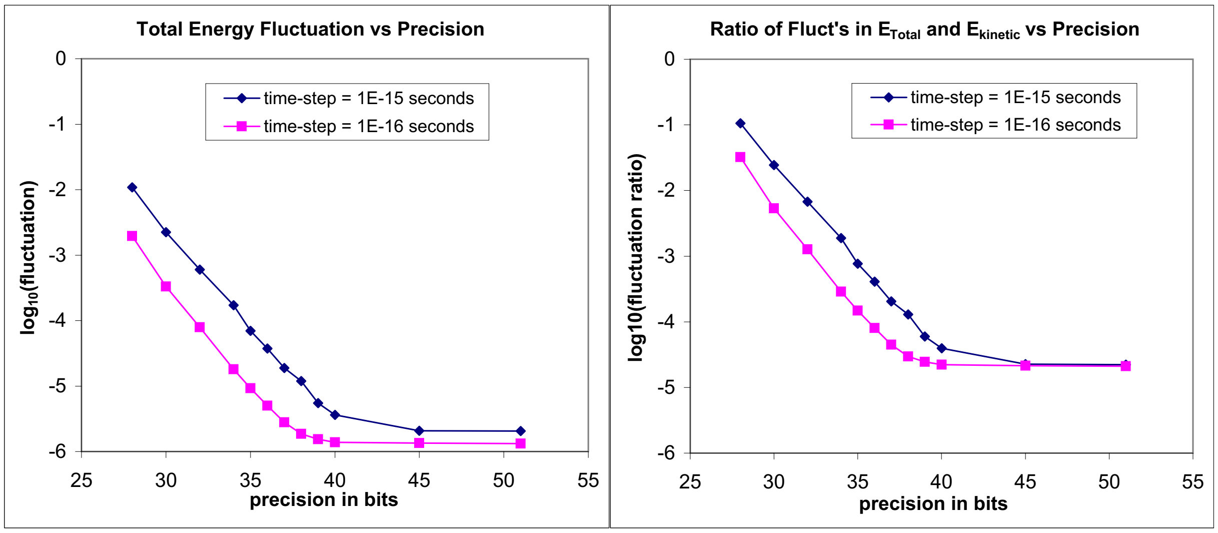

Fig. 5. Shown is the effect of precision on two metrics for simulation accuracy: (a) Fluctuation of total energy and (b) the ratio of the ¤uctuations in total and kinetic energies. Simulations were carried out with two different time-steps.

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 ¤oating point [10]. Clearly, though, this last reference is not the consensus.

However, it is also the case that 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. However, for implementations on con£gurable circuits, 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. To us, therefore, this is an important problem – we present our initial results here.

One classic check for simulation quality is to measure the ¤uctuation of physical quantities that should remain invariant, such as energy. The relative rms ¤uctuation in total energy is de£ned 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. [9]. The £rst used double precision ¤oating point, the second tracked the hardware implementation, e.g. in varying precision. When the precision of the £xed point code was set at 50 bits, the results precisely matched that of the ¤oating point code.

[1]Of course the opposite question of what to do when double precision appears to be inadequate *is* a fundamental issue (see e.g. [11]). The general solution is to increase the resolution of the time steps.

We ran a large number of experiments to £nd the relationship between energy ¤uctuation and precision. In agreement with [9], we found that the various function units can be tuned independently to derive the optimal FPGA circuits that retain minimal energy ¤uctuation. 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 ¤uctuation as a full 53-bit datapath.

However, ¤uctuation of total energy is not the only check that a system is "well-behaved." Another is the ratio of the ¤uctuations between total energy and kinetic energy $R = \Delta E_{total}/\Delta E_{kinetic}$. R should be less than .05 [12]. We plot R in the right half of Figure 5. Note that by this measure, 31 bits are suf£cient for time-steps of E-15 seconds and 30 bits are suf£cient 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 reality 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), or (2) 8 force pipelines with 35-bit precision. In the £rst implementation, behavior is equivalent to double precision ¤oating point. In the second implementation, quality depends on the metric. The 35-bit design has from a factor of 10x to 50x more energy ¤uctuation than the best case, but between 100x and 500x lower R than what has been

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	$50.8\times$
VP70 AMS	40	4	264(80%)	251(77%)	12.2	$46.4\times$
VP70 AMS	45	4	288(88%)	285(87%)	13.2	$42.7\times$
VP70 AMS	51	4	288(88%)	317(97%)	18	$31.3\times$
VP70 AMS	35	8	256(78%)	326(99%)	22.2	$51.0\times$
VP100 sim	51	4	288(65%)	317(77%)	13.6	$41.5\times$
VP100 sim	35	8	256(58%)	334(75%)	12.8	$88.5\times$

regarded as minimal to indicate "good behavior." Perhaps this is a case where there is a very 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 con£gurable circuits it is possible to do the reverse: trade off unneeded precision for computing resources. Further study will show whether 35 bits, or some other implementation dependent precision, is indeed suf£cient.

5. IMPLEMENTATION, VALIDATION, AND RESULTS

The design was implemented on a WildstarII-Pro board from Annapolis Micro Systems, which has two Xilinx Virtex-II-Pro XC2VP70 -5 FPGAs (referred to as *VP70 AMS* in Table 1). However, only one of the FPGAs is currently 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.

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 should make running these larger simulations with no slowdown straightforward. We are currently implementing this extension.

The LJ force is computed with look-up table and linear interpolation. The look-up table is indexed in three dimensions: Pi type, Pj type, and distance squared. Two memories are used, one for the even index entries, one for the odd index entries. Following the serial reference code, 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 £t on-chip. For more particle types, tables must be swapped as needed. However, since the particle types are known and the particles can be ordered *a priori*, this swapping should usually be possible without requiring stalls.

Overall, the critical resources are the hard multipliers and, in particular, the block RAMs. 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 were used in every implementation.

The design was validated against two serial reference codes, an external double precision ¤oating point code (see [13]) and our own code that tracked the hardware implementation. Against the hardware tracker, the match was exact. Against the double precision ¤oating point code there was a very close match as to be expected from the analysis in [9] and the previous section. Both reference codes ran at about 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 [2].

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 ¤oating 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 ¤oating point) requires substantially more hard multipliers, but for little bene£t.

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.

Neither the serial code, nor the FPGA codes were optimized beyond taking care to follow good design procedures. However, the serial code does perform the force computation using a table look-up, saving many ¤oating point operations over a direct implementation.

6. OPTIMIZATIONS AND EXTENSIONS

As always when measurements are done with respect to rapidly advancing technology, all numbers reported here are transient. However, FPGAs appear to be following Amdahl's law just as much as are microprocessors. For the current study, this is probably to the bene£t of the FPGA designs: increased resources can be immediately applied to the computation by adding pipelines.

Another axis of variation 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.

The most important next task is to integrate our MD implementations into a production code. We are investigating several alternatives.

An obvious extension involves using completely different algorithms, in particular those based on Ewald sums or FFT-based methods: 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 £rst. Finally, this work is part of a larger project involving the acceleration of applications in computational biochemistry (e.g. [14]) and will be integrated into that.

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

7. REFERENCES

[1] 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 *Proc. Int. Work. High Perf. Comp. Bio.*, 2004.

[2] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, "Recon£gurable molecular dynamics simulator," in *Proc. FCCM*, 2004, pp. 197–206.

[3] 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," *J. Comp. Chem.*, vol. 20, no. 2, pp. 185–199, 1999.

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

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

[6] 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.

[7] M. Gokhale, J. Frigo, C. Ahrens, J. Tripp, and R. Minnich, "Monte Carlo radiative heat transfer simulation," *Proc. Field Programmable Logic and Applications*, pp. 95–104, 2004.

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

[9] 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," *J. Comp. Chem.*, vol. 16, no. 9, pp. 1120–1130, 1995.

[10] 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.

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

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

[13] 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.

[14] T. VanCourt, Y. Gu, and M. Herbordt, "FPGA acceleration of rigid molecule interactions," in *Proc. Field Programmable Logic and Applications*, 2004.