Improved Interpolation and System Integration for FPGA-Based Molecular Dynamics Simulations[*]

Yongfeng Gu Tom VanCourt Martin C. Herbordt

Department of Electrical and Computer Engineering
Boston University; Boston, MA 02215
EMail: {maplegu|tvancour|herbordt}@bu.edu

Abstract: FPGA-based acceleration of molecular dynamics (MD) has been the subject of several recent studies. Here we describe a new non-bonded force computation pipeline implemented on a 2004-era COTS FPGA board and its integration into the ProtoMol MD code. There are several innovations: a novel interpolation strategy; the introduction of a "semi-floating point" format; and various issues related to system integration. As a result, we are able to model far more particle types, without relying on complex buffering, and obtain higher accuracy than previously. A two pipeline accelerator has been implemented on a Xilinx VirtexII Pro VP70, integrated into ProtoMol, and tested with an enzyme inhibitor model having 8000 particles and 26 particle types. Despite performing all $O(n)$ work on the host PC, as well as the data conversion and communication overhead, this implementation yields a 5.5x speed-up over a 2.8GHz PC, and with accuracy comparable to the serial code.

1 Introduction

Molecular dynamics (MD) simulations are increasingly important in understanding the behavior of chemical and biological systems. MD has therefore become a staple of high-end computing [7], with not only supercomputers and MPPs being applied, but also with several machines being developed (in part or entirely) for this application. The latter include the IBM Blue Gene/L [1, 5], the MD-GRAPE [9], and the GRAPE-based Protein Explorer [14]. More recently, the viability of FPGA-based MD simulation (FPGA/MD) was demonstrated [3, 6]. Despite avoiding standard double-precision floating point (DP) arithmetic, significant speed-ups were achieved with little detriment to physical significance.

Several issues remain to be solved before FPGA/MD is likely to enter production use. We address the following in this work:

- **Number of particle types simulated.** The van der Waals (Lennard-Jones or LJ) force computation depends on the types of the two interacting atoms according to atomic species (e.g. C, H, N, O) and chemical contexts (e.g. O in C=O carboxyl moieties vs. O-H hydroxyl groups). For the table-lookup method used previously ([3, 6]), a separate table was used for each combination of atom types. While viable, complex memory exchanges are required.
- **Simulation accuracy.** Until FPGAs have a number of hard-wired DP units, non-standard arithmetic is likely to yield the most competitive implementations. While previous studies have shown these to be well-behaved, this issue remains a concern with respect to general acceptance.
- **System integration.** MD codes are highly complex. Also, many MD functions (such as computing the bonded forces) are best performed on a microprocessor. FPGA implementations are therefore only viable (in the near future) when integrated into existing systems. Among the issues that must be addressed are partitioning the code, developing interfaces, and handling data translation. This must be done while retaining a performance advantage.

We address these issues as follows. To avoid the large number of look-up tables, we use one table

[*]This work was supported in part by the NIH through award #RR020209-01 and facilitated by donations from Xilinx Corporation. Web: http://www.bu.edu/caadlab.

per r^{-x} term and perform the rest of the LJ computation explicitly. With respect to non-standard arithmetic, although we still avoid DP, a number of methods are applied to improve accuracy and assure physical significance. These include non-uniform interpolation intervals, use of higher order interpolation, computing coefficients with orthogonal functions, and use of "semi floating point" arithmetic. System integration is demonstrated with ProtoMol [10]: this implementation retains a 5.5× improvement in total application performance (on 2004-era hardware) with little if any compromise in accuracy over the original code. Since the initial submission, we have also implemented cell lists. Details will be reported elsewhere, but we include here some initial results.

At least one other issue remains to be addressed in FPGA/MD beyond the scope of this work. In particular, we compute long range non-bonded forces using cell lists with switching. Although this method scales well, a full version of FPGA/MD will likely include a more rigorous algorithm (e.g. using Ewald sums). The significance of the current work is in its advance in several areas over the state-of-the-art, and because these methods will be integral to future optimized implementations. Broader significance is that it indicates that contemporary MPP/SMP FPGA nodes are likely to obtain a cost-effective speed-up.

2 Methods

2.1 MD Review

As described in previous work [6] (for more detail see, e.g., [11]), 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 the Verlet (or similar) method is typically used.

In general, the forces depend on the physical system being simulated and may include LJ, Coulomb, hydrogen bond, and various covalent bond terms:

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

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. The motion integration computation is also $O(N)$. Although some of these $O(N)$ terms are easily computed on an FPGA, their low complexity makes them likely candidates for host processing.

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 on the FPGA; we also implement multiple atom types.

2.2 New Force Computations

Because of the complexity of the LJ expression, and because it is possible in principle for any two atoms to interact this way, computing this force has been problematic. Here we propose a new method of force computation that changes what gets looked up, the shape of the table, the interpolation mechanism, and the method of computing interpolation coefficients.

Previous implementations of FPGA/MD have used table look-up for the entire force as a function of particle separation [3, 6]. This method is efficient for uniform gasses where only a single table is required [3]. As the LJ force depends on atom types, however, simulations of T different atom types require T^2 tables. Since table look-up is in the critical path, the tables must be in on-chip memory (block RAMs) for FPGA/MD to be viable. In previous work we described a latency hiding technique whereby tables are swapped as needed [6].

Here we propose a different method, used also in some MD codes. Instead of implementing the force pipeline with a single table look-up for the entire force, we use two tables, one each for r^{-14} and r^{-8}. The advantage is using two tables instead of T^2, thereby removing the need to swap tables; the disadvantage is that several additional operations must be executed to obtain $\mathbf{F}_i^{LJ}$.

We next describe an optimization to the look-up tables themselves. The r^{-x} expressions display extreme changes in behavior over the range of possible interaction radii, as shown in Figure 1. It would be an exorbitant and needless cost in table size to

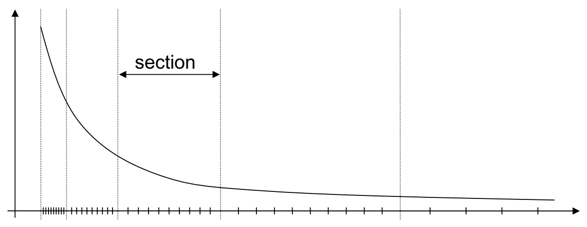


Figure 1: Table look-up varies in precision across r^{-k}.

use the same number of coefficients and the same step sizes in the well-behaved regions of the curve. Rather, as in [3], each curve is divided into several sections along the X-axis. Here, the length of each section is twice that of the previous; however, each section is cut into the same number interpolation intervals N.

To improve the accuracy of the force computation, we interpolate using higher order terms. Here we assume a Taylor expansion:

$$F(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2)}{2!} + \ldots;$$

in the next subsection we describe a more accurate alternative. Point a is the left end of the interval.

When the interpolation is order M, each interval needs $M+1$ coefficients, and each section needs $N*(M+1)$ coefficients. Since the section length increases exponentially, *extending the curve (in r) only increases the size of coefficient memory very slowly.*

Increasing M or N each improves simulation accuracy. Interestingly, on the FPGA these two numbers have a resource cost in different components: the main cost for finer intervals is in block RAMs, while the main cost for higher order interpolation is in hardware multipliers and registers. Table 1 gives a sample of the tradeoff effects. For our system configuration, $N = 128$ and $M = 3$ appears to be optimal.

2.3 Computing the Coefficients

We now show how to develop polynomial approximations to arbitrary functions so that the approximation is relatively easy to compute with economical use of FPGA resources. The FPGA will implement the approximation F of function f as a sum of terms:

$$F(x) = a_0 + a_1x + a_2x^2 + a_3x^3 \qquad (1)$$

Table 1: Shown is the trade-off between interval size (N is the number of intervals per section) and interpolation order M for r^{-14}.

N	M	Average Error	Maximum Error
32	4	2.55E-7	3.67E-6
64	4	7.35E-9	1.08E-7
64	3	3.74E-7	4.19E-6
128	3	2.56E-8	2.55E-7
128	2	2.27E-6	1.73E-5
512	2	3.32E-8	2.66E-7
512	1	1.17E-5	6.04E-5
2048	1	7.31E-7	3.76E-6

where the coefficients a_i are stored separately for a set of intervals that partition the x range of interest. This form is desirable for FPGA implementation because the low powers of x require relatively few hardware multipliers. In order to maintain accuracy in F, it is defined as a piecewise function on a set of intervals that partition the range of interest, with coefficients a_i for any interval chosen to provide the best approximation of f on that interval. The piecewise nature of F is taken for granted in the remainder of this discussion. One obvious way to create such an approximation on an interval near point p is by truncating a Taylor series expansion (as was done above). When carried to an infinite number of terms, this represents F exactly. Truncating the series causes problems in accurate approximation, however. In order to see how accuracy problems arise, consider the curves in Figure 2 illustrating the first few monomials x^i.

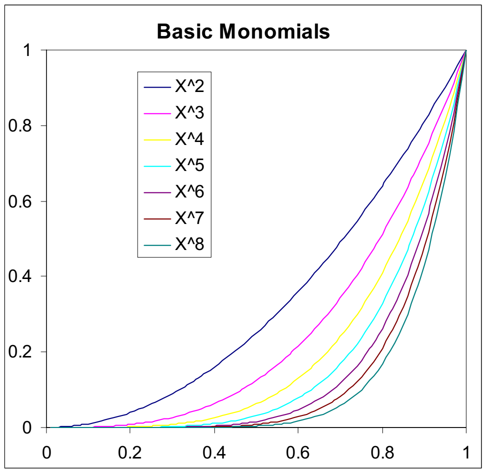
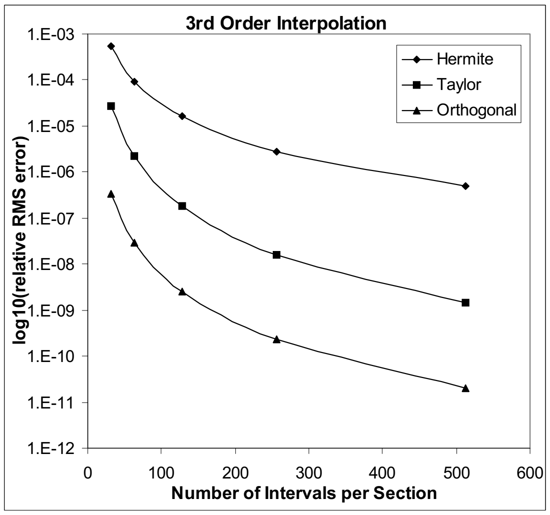


Figure 2: Basic monomials.



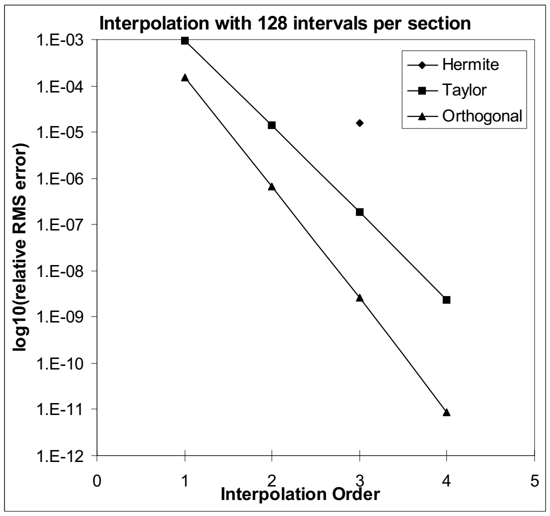


Figure 3: Interpolation comparisons.

All of the curves have the same general half-U shape, and become increasingly similar as the degree increases. In other words, the x^2 or x^3 term of a low-order approximation is important in approximations of all higher-order Taylor terms. Said differently, large coefficients of high-order Taylor terms contribute heavily to the coefficients in the low-order approximation. The $i!$ denominator in the Taylor series dominates for asymptotically large i. Still, for the functions and ranges of current interest (r^{-4} and r^{-7}, $0.1 < r < 100$), many Taylor series terms $i > 3$ have appreciably large numerators due to large binomial coefficients in $(x-p)^i$ and large derivative values $d^i f/dp^i$. As a result, the truncated Taylor series omits many large high-order terms that are important to approximation using low-order polynomials. This is especially problematic for the larger intervals where behavior may worse even than that of linear interpolation.

Instead, we propose to approximate f using a set of orthonormal functions on each interval. Following the conventions of linear algebra, orthogonal functions $p(x)$ and $q(x)$ have the property that $p(x) \cdot q(x) = 0$, for a suitable definition of dot product '$\cdot$'. Likewise, normal functions $p(x)$ have the property that $p(x) \cdot p(x) = 1$. A set of orthonormal functions is a set in which all members are normal and are orthogonal to all other members, and forms a basis set that spans some space of functions. The conventional dot product of two functions on some interval $[A, B]$ is defined as $\int_A^B p(x)q(x)dx$.

An approximation of f in terms of basis set b_i is a projection of f from some high-order function space down into the low-order space spanned by that set of basis functions. Because all terms in f are involved in the projection into that basis, this technique avoids the problem of skipping high-order terms in f that contribute significantly to low-order terms in the approximation. Because the intended FPGA implementation uses polynomials of the form in Equation 1, the set of basis functions is chosen to span the function space $\{1, x, x^2, x^3\}$. For reasons beyond the scope of current discussion, monomials are not the most convenient set of basis functions for spanning the space of cubic polynomials. Instead, a set of *orthogonal polynomials* offers many advantages. Gram-Schmidt orthogonalization is used to generate such a set of orthonormal basis functions. Initial computations show order-of-magnitude reductions in error over the truncated Taylor coefficients of the same degree.

Although orthogonal polynomial interpolation is the best local approximation in each interval, the interpolation polynomials are continuous only within the interval. When transformed into Fourier space, as is done in some long range force calculations, spurious terms may result.

Piecewise Hermite interpolation is an improved version of the original piecewise linear method. Given two points on the target curve and their derivatives $\{(x_0, f(x_0), f'(x_0)), (x_1, f(x_1), f'(x_1))\}$, a polynomial is required to go across these two points with the same derivatives Since there are four constraints, this polynomial is third order. The four coefficients can be computed by solving the following equations:

$$f(x_0) = dx_0^3 + cx_0^2 + bx_0 + a$$
$$f'(x_0) = 3dx_0^2 + 2cx_0 + b$$
$$f(x_1) = dx_1^3 + cx_1^2 + bx_1 + a$$
$$f'(x_1) = 3dx_1^2 + 2cx_1 + b$$

We compare the three higher order interpolation methods—Taylor, Orthogonal, and Hermite—by plotting their relative RMS error. In the left graph of Figure 3, the number of intervals per section varied; in the right graph, the order is varied.

2.4 Semi Floating Point

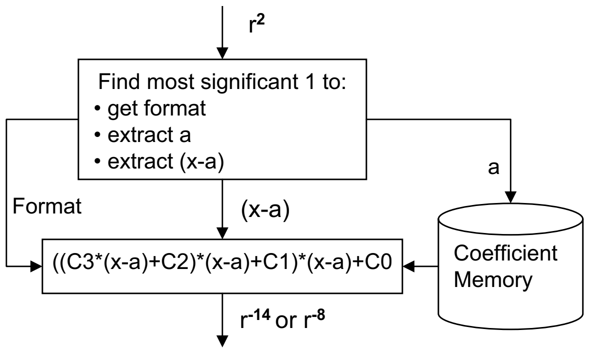


Figure 4: Position of the leading 1 determines the operand format in the interpolation pipeline.

We begin by observing that although the dynamic range of an entire r^{-x} curve is, perhaps, 100 bits, the dynamic range of a particular section could be only 9 bits. Also, the ranges of interpolation coefficients differ in different parts of the curve: they have relatively large magnitudes in the left-hand part of the curve, but lower magnitude in the right-hand regions. That implies the need for different numbers of bits to the left of the decimal point in each region of the curve, to maintain roughly constant accuracy. This optimization allows the same precision in each section, but higher coefficient values in the ill-behaved sections (with numbers of the form xxx.x) and lower values where the curve is flatter (x.xxx). We call this modified numbering system semi floating point: The binary point is able to shift, but only a few places. The data path of the interpolation pipeline is shown in Figure 4.

Once r^2 is known, its interval is determined by the position of the leading 1. The interval, in turn, indicates the format. Finally, the format is used by the adders and multipliers to align the operands. The adder is shown in Figures 5; the multiplier is analogous. The LJ pipeline uses 11 formats, the Coulomb 14; as there is no overlap in formats, the combined LJ/Coulomb pipeline uses 25 formats. Comparison with floating point is presented in the next section.

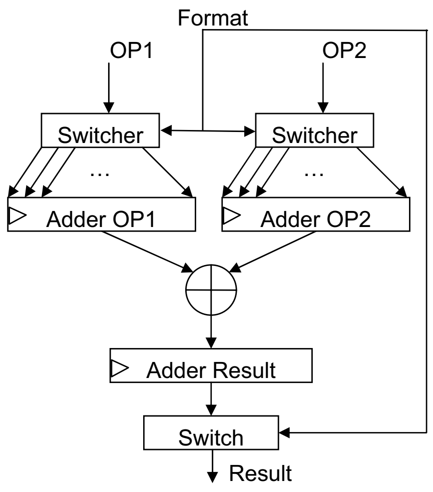


Figure 5: Semi FP adder with explicit alignment.

3 Implementation

The overall system consists of a PC with a 2.8 GHz Xeon CPU with a WildstarII-Pro PCI board from Annapolis Micro Systems (see Figure 6). The board has two Xilinx Virtex-II-Pro XC2VP70 -5 FPGAs; however, only one of the FPGAs was used. ProtoMol 2.03 was used (downloaded from the ProtoMol website). The operating system was Windows-XP; all codes were compiled using Microsoft Visual C++ .NET with performance optimization set to maximum. FPGA configurations were coded in VHDL and synthesized with Synplicity integrated into the Xilinx tool flow. Data transfer between host and coprocessor was effected with the software support library from Annapolis Microsystems. These transfer routines are highly efficient with nearly the full PCI bandwidth being used and little system overhead.

ProtoMol is a high-performance MD framework in C++ from Notre Dame University, and is designed especially for ease of experimentation. Hardware/Software partitioning was therefore simply a matter of swapping out the appropriate routines and replacing them with our own. Particle position data are down- and up-loaded in DP; fixed point conversion on both ends is done in the FPGA (the

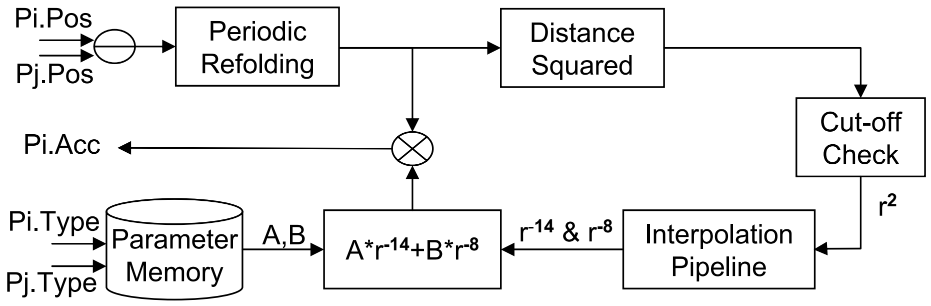


Figure 7: Force pipeline.

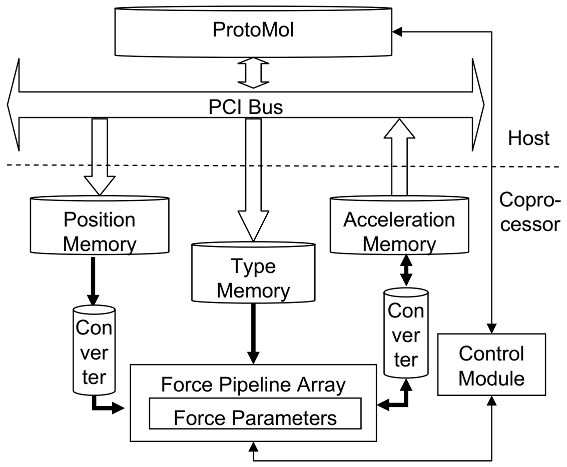


Figure 6: System block diagram.

"converters" shown in Figure 6). All motion integration, $O(N)$ force computations, and energy computations are performed with the original ProtoMol code.

The overall FPGA pipeline is shown in Figure 7; more details were presented earlier, or are available in [6]. Currently two pipelines fit on the FPGA; this is in contrast to the four pipelines previously obtained. The reduction is due to the complexity of the interpolation scheme. We have, however, achieved four pipelines in synthesis-only with the VP100; more should be possible with newer chips and further optimization. The current design supports up to 32 atom types and 11,200 atoms. The same pipeline is used for both LJ and Coulombic force computations; coefficients, however, are swapped. Following the conclusions of previous work [6], 35 bits of precision are used, but in the semi floating point representation described above.

The VP70 design currently runs at 75MHz. Nearly all the slices are used; a large fraction, however, are apparently used by the synthesis tool for optimization. In this design, 70% of the BRAMS are used; in the previous four pipeline version nearly all were.

Table 2: Shown is the resource usage in slices of various components described in this paper.

Format	Add	Mul	Pipeline Ttl
LogiCore DP	692	540	19566
LogiCore FP	329	139	6998
Semi fp 35-bit	70	390	n.a.
Integer 35-bit	18	400	n.a.
Combined semi, int	n.a.	n.a.	5624

We now compare the semi floating point to full floating point implementations. For the latter we use LogiCore Floating-point Operator v2.0 from Xilinx [15]. Table 2 gives the number of slices for various components. The final column gives the number of slices for the three versions of the entire non-bonded force pipeline. Our version uses some integer units as well as the semi floating point, but only at points in the computation where no precision can be lost. The SP and DP pipelines could perhaps also be optimized this way, but the requisite complex conversion might make this less advantageous than it is for the semi FP pipeline. A comparison with respect to register use yields similar results.

The conclusion for the VP70 is that two pipelines can also be implemented using single precision floating point, but no more, and with a loss of precision from 35 to 24 bits. As shown previously [2, 6], this difference could be critical to simulation accuracy. On the other hand, semi floating point precision could scale at least up to 40 bits (without serious optimization), with a slight reduction in operating frequency. With respect to double precision:

Table 3: Results comparing original and accelerated versions of ProtoMol.

Description	PC only time/step	PC + FPGA time/step
All-to-all no cut-off	15.1s	0.96s
all-to-all cut-off at 10A	8.0s	0.96s
Non-Bonded forces not computed	0.014s	0.022s
Cell list w/ size 5A LJ,CL cut-off 10A	0.66s	0.12s

fitting even one pipeline is problematic. This experience matches that of two other studies [8, 12].

4 Performance and Accuracy

Clearly, accuracy of the calculation is a concern when converting a floating point application to fixed point arithmetic. Because of the inherently chaotic nature of the calculation, "Obtaining a high degree of accuracy in the trajectories is neither a realistic nor a practical goal" [11]. Rather, quality assurance in MD is determined by observing fidelity of emergent physical properties. Perhaps the most common of these is energy fluctuation (see, e.g., [4, 13]). Our particular goal is therefore to achieve energy fluctuation *similar to the original calculations.*

To measure energy fluctuation, the physical model simulated was of bovine pancreatic trypsin inhibitor in water, a model of approximately 1100 particles and 26 atom types, run for 10,000 time steps. We say approximately, because the number of waters was varied within 10% between runs to obtain an ensemble. Periodic boundary conditions were used and switched cut-off at that size. The original ProtoMol was compared with ProtoMol modified as described in the previous section. After 10 runs we have found comparable energy fluctuation between the two systems, with both being close to 0.014. If anything, it appears that the FPGA version has slightly better accuracy, a phenomenon we are still investigating.

To measure performance we compared full end-to-end runs of the same model with 8192 particles and 26 atom types. Again, the modified (FPGA) ProtoMol still performs motion integration, bonded force computation, and energy computation on the host using the original code. When cell lists are used, ProtoMol computes them on the host as well. The simulation uses periodic boundary conditions with a size of 64A x 50A x 50A. Results are shown in Table 3. The first row shows LJ and Coulombic forces being computed completely all-to-all. The second is also all-to-all, but a cut-off is applied; ProtoMol optimizes for this option. In the third row, the non-bonded force computation is removed. This is shown to indicate explicitly the part of the computation that is not being accelerated with the FPGA. The additional time for the PC + FPGA version is due to the extra processing required to format the cell-lists before transfer. The final row shows the full implementation including cell-lists.

As stated previously, these simulations do not include computation of long-range forces as would usually be done, e.g. with Ewald sums. Also, the FPGA is two years old. Using a bigger contemporary FPGA with higher speed-grade would enable twice the number of pipelines and higher operating frequency. Optimizing the design should yield even greater speed-up. Together, these should double the performance of the current implementation, even accounting for the serial overhead.

5 Summary and Discussion

This work extends previous work in FPGA/MD in three major directions: complexity of phenomena modeled, accuracy of modeling compared to production MD code, and integration with production MD code. In particular we have achieved accuracy comparable to the serial version while simulating a biologically significant molecule. In the process, we have given up some of the speed-ups described previously, which had been determined by measuring acceleration of the non-bonded forces and motion integration alone. The speed-up obtained here is encouraging, especially considering the somewhat dated technology used for the FPGA.

Recently, other work has appeared that also integrates reconfigurable computing into MD codes. Kindratenko and Pointer [8] and Scrofano et al. [12] both port the non-bonded force computations of existing codes to the SRC processor. Both also use the SRC high-level language interface to implement the FPGA configurations. Kindratenko's work begins with NAMD, while that of Scrofano's begins with their own code comprised of basic MD functions. Both implement cell lists, while Kindratenko's also implements a table-lookup version of PME. A major difference between that work and ours is in precision: both Kindratenko and Scrofano use single precision floating point; Kindratenko also uses 32-bit integer. Neither reports on the effect that this difference would have on the quality of the MD simulations. Another difference is performance: Kindratenko achieves a factor of 3× speed-up (using 2 FPGAs) over the original, highly optimized NAMD code; Scrofano reports a 2× speed-up over their own reference code. An advantage of both these approaches over that presented here is the ease in which large simulations can be conducted. Augmenting our current system to support large simulations requires the implementation of off-chip (on-board) swapping of cells. This is relatively straightforward, and the latency can easily be hidden.

We are in the process of deeper evaluation of the issues related to the relationship between interpolation method and physical significance. In particular, we wish to separate out the physical significance, if any, of the various aspects of the scheme: non-uniform interpolation intervals, higher order terms, and method of coefficient computation. Clearly large ensembles of experiments are required, and on numerous quality assurance measures. While energy fluctation is the most common, others are also used and will be examined.

Acknowledgments. We would like to thank the ProtoMol group, first for providing the code under open source license (http://protomol.sourceforge.net), and also for answering questions that arose during this work. We also thank the anonymous reviewers for their many helpful suggestions.

References

[1] Allen, F., and et al. Blue Gene: a vision for protein science using a petaflop supercomputer. *IBM Systems Journal 40*, 2 (2001), 310–327.

[2] Amisaki, T., Fujiwara, T., Kusumi, A., Miyagawa, H., and Kitamura, K. Error evaluation in the design of a special-purpose processor that calculates nonbonded forces in molecular dynamics simulations. *Journal of Computational Chemistry 16*, 9 (1995), 1120–1130.

[3] Azizi, N., Kuon, I., Egier, A., Darabiha, A., and Chow, P. Reconfigurable molecular dynamics simulator. In *FCCM* (2004), pp. 197–206.

[4] Barth, E., and Schlick, T. Overcoming stability limitations in biomolecular dynamics. I. combining force splitting via extrapolation with Langevin dynamics in LN. *J. Chemical Physics 109*, 5 (1998), 1617–1632.

[5] Fitch, B., and et al. Blue Matter: Strong scaling of molecular dynamics on Blue Gene/L. Tech. Rep. RC23688 (W0508-035) Computer Science, IBM Research Division, 2005.

[6] Gu, Y., VanCourt, T., and Herbordt, M. C. Accelerating molecular dynamics simulations with configurable circuits. *IEE Proceedings on Computers and Digital Technology 153*, 3 (2006), 189–195.

[7] High-End Computing Revitalization Task Force. Federal plan for high-end computing, 2004.

[8] Kindratenko, V., and Pointer, D. A case study in porting a production scientific supercomputing application to a reconfigurable computer. In *Field-programmable Custom Computing Machines* (2006).

[9] Komeiji, Y., Uebayasi, M., Takata, R., Shimizu, A., Itsukashi, K., and Taiji, M. Fast and accurate molecular dynamics simulation of a protein using a special-purpose computer. *J. Comp. Chem. 18*, 12 (1997), 1546–1563.

[10] Matthey, T. ProtoMol, an object-oriented framework for prototyping novel algorithms for molecular dynamics. *ACM TMS 30*, 3 (2004), 237–265.

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

[12] Scrofano, R., Gokhale, M., Trouw, F., and Prasanna, V. A hardware/software approach to molecular dynamics on reconfigurable computers. In *Field-programmable Custom Computing Machines* (2006).

[13] Stuart, S., Zhou, R., and Berne, B. Molecular dynamics with multiple time scales: The selection of efficient reference system propagators. *J. Chemical Physics 105*, 4 (1996), 1426–1436.

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

[15] Xilinx, Inc. *Product Specification — Xilinx LogiCore Floating Point Operator v2.0*, 2006.