Performance Potential of Molecular Dynamics Simulations on High Performance Reconfigurable Computing Systems *

Matt Chiu **Martin C. Herbordt** **Martin Langhammer**

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

Altera UK Ltd
Buckinghamshire UK
EMail: mlangham@altera.com

Abstract: The acceleration of molecular dynamics (MD) simulations using high performance reconfigurable computing (HPRC) has been much studied. Given the intense competition from multicore, and from other types of accelerators, there is now a question whether MD on HPRC can be competitive. We concentrate here on the MD kernel computation–determining the force between short-range particle pairs–and examine it in detail to find the performance limits under current technology and methods. We systematically explore the design space of the force pipeline with respect to arithmetic algorithm, arithmetic mode, precision, and various other optimizations. We examine simplifications that are possible if the end-user is willing to trade off simulation quality for performance. And we use the new Altera floating point cores and compiler to further optimize the designs. We find that for the Stratix-III, and for the best (as yet unoptimized) single precision designs, 11 pipelines running at 250MHz can fit on the FPGA. If a significant fraction of this potential performance can be maintained in a full implementation, then HPRC MD should be highly competitive.

1 Introduction

Molecular dynamics simulation (MD) is a central method in high performance computing (HPC) with applications throughout engineering and natural science. Acceleration of MD is recognized as a critical problem with a many order-of-magnitude gap between the largest current simulations and the potential physical systems to be studied. As such it has received attention as a target for supercomputers [6], clusters [4], and dedicated hardware [15, 23, 27], as well as coprocessing using GPUs [20], Cell [24], and FPGAs [1, 3, 9, 11, 13, 21, 30]. The last of these, MD with High Performance Reconfigurable Computing (HPRC) is our focus here. In particular, we seek to establish bounds on possible performance of MD on HPRC with the goal of finding whether current methods and technologies can provide the basis for cost-effective implementations.

While HPRC MD has been widely studied, delivering cost-effective production applications has proved challenging. Although there are many difficulties, the basic problem is perhaps that MD codes often rely on double precision floating point (DP). There are two reasons why this is problematic for HPRC. First, FPGAs are not highly optimized for floating point. And second, DP is high precision: this does not play to the FPGAs' strength, which is configurability into large numbers of low precision arithmetic units.

Our method of determining the current viability of HPRC/MD is as follows. We use as our basic metric throughput of short-range force computations between particle pairs. In other words, how many of these force computing pipelines fit on a target FPGA and how fast do they run? For comparison, we use published results with respect to standard benchmarks.

Although a simple metric, we believe that it captures the essence of MD – short-range force computation is the bulk of the work; it may be possible to offload or otherwise hide other parts of the computation. Another advantage is that, although far from trivial to determine, it is at least somewhat independent of the way the rest of the application is implemented. To use a higher level metric, say, examining complete HPRC/MD systems (many of which exist only in prototype), would both be intractable and not answer the basic question. After all, for any given HPRC/MD system, it may be possible to drastically improve performance through continued optimization. On the other hand, a lower level metric, say, peak FLOPs, would not allow for arithmetic-level optimizations (which are also a focus of this study). Stating our goal another way, we want to be able to say, "even if everything else about the design is perfect (e.g., keeping the pipelines full), we are limited by the particle-particle force computation to performance of no better than N-times a single

*This work was supported in part by the NIH through award #R01-RR023168-01, by IBM through a Faculty Award, and facilitated by donations from Altera Corporation. Web: http://www.bu.edu/caadlab.

978-1-4244-2826-7/08/$25.00

processor core." Our goal is similar to that of Koehler, et al. [14], but our methods take full advantage of our application knowledge.

This simplified metric leaves a substantial design space. There are various ways to perform the arithmetic, modes in which to perform the operations, levels of precision, and other optimizations if the user can tolerate reduced quality simulations:

- Direct computation versus table lookup with interpolation (TLwI),
- If TLwI, what order interpolation?
- Precision: single, double, other,
- Mode: floating point, hybrid fixed/floating point, other,
- Implementation: synthesized components, vendor cores, vendor compiler, (such as the floating point datapath synthesis tool from Altera [16]),
- Target FPGA, and
- Various arithmetic reorderings.

We find that for the Stratix-III, and for the best (as yet unoptimized) single precision designs, 11 pipelines running at about 250MHz can fit on the FPGA. If a significant fraction of this potential performance can be maintained in a full implementation, then HPRC MD should remain highly competitive.

Significance is at three levels. The first is demonstrating the continued competitiveness of HPRC/MD. The second is (as far as we know) the first systematic exploration of the MD datapath design space for FPGAs. We find that direct computation, rather than table lookup, appears to be preferred. The third is a case study of the Altera Floating Point Compiler. Although this is not an ideal computation, we still find a substantial reduction in non-DSP logic.

The rest of this paper is organized as follows. In the next section, we review the applicable parts of MD simulation. There follows descriptions of the FPGA implementations, after which comes the results obtained so far and some discussion.

2 MD Computation

2.1 MD Review

MD is an iterative application of Newtonian mechanics to ensembles of atoms and molecules (see, e.g., [19] for details). MD simulations generally proceed in phases, alternating between force computation and motion integration. In general, the forces depend on the physical system being simulated and may include LJ, Coulomb, hydrogen bond, and various covalent bond terms:

$$\mathbf{F}^{total} = F^{bond} + F^{angle} + F^{torsion} + F^{HBond} + F^{non-bonded} \quad (1)$$

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, which is what we assume here. The LJ force for particle i can be expressed as:

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

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:

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

A standard way of computing the non-bonded forces (LJ and Coulombic) is by applying a cut-off. Then the force on each particle is the result of only particles within the cut-off radius. Since this radius is typically less than a tenth of the size per dimension of the system under study, the savings are tremendous, even given the more complex bookkeeping required to keep track of cell- or neighbor-lists.

The problem with cut-off is that, while it may be sufficiently accurate for the rapidly decreasing LJ force, the error introduced in the slowly declining Coulombic force may be unacceptable. A number of methods have been developed to address this issue with some of the most popular being based on Ewald Sums (see, e.g., [5]) and multigrid (see, e.g., [12, 25]). Here we use the standard convention of calling *short-range* the LJ force and the Coulombic force generated within a cut-off radius. We refer to the Coulombic force generated outside the cut-off radius as *long-range*. Since the long-range force computation is generally a small fraction of the total (see, e.g., [7, 21]), and because of the high-level goal of this particular paper, we ignore it here.

2.2 Short-Range Force Computation

As just described, the short-range force computation has two parts, the LJ force and the rapidly converging part of the Coulomb force. The LJ force is often computed with the so-called 6-12 approximation given in Equation 2. This has two terms, the repulsive Pauli exclusion and the van der Waals attraction. Both require coefficients specific to the component particles of the particle pair whose interaction is being evaluated. These can be combined with the other constants (physical and scaling) and stored in coefficient look-up tables.

Thus the LJ force can be expressed as

$$\frac{\mathbf{F}_{ji}^{LJ}(r_{ji}(a,b))}{\mathbf{r_{ji}}} = A_{ab}|r_{ji}|^{-14} + B_{ab}|r_{ji}|^{-8} \quad (4)$$

where A_{ab} and B_{ab} are distance-independent coefficient look-up tables indexed with atom types a and b.

Returning now to the Coulomb force computation, we begin by rewriting equation 3 as

$$\frac{\mathbf{F}_{ji}^{CL}(r_{ji}(a,b))}{\mathbf{r_{ji}}} = QQ_{ab}|r_{ji}|^{-3}, \quad (5)$$

where QQ_{ab} is a precomputed parameter (analogous to A_{ab} and B_{ab}). Because applying a cut-off here often causes unacceptable error, and also because the all-to-all direct computation is too expensive for large simulations, various numerical methods are applied to solve the Poisson equation that translates charge distribution to potential distribution. To improve approximation quality and efficiency, these methods split the original Coulomb force curve in two parts (with a smoothing function $g_a(r)$): a fast declining short range part and a flat long range part. For example:

$$\frac{1}{r} = (\frac{1}{r} - g_a(r)) + g_a(r). \quad (6)$$

The short range component can be computed together with Lennard-Jones force using a third look-up table (for QQ_{ab}). The entire short range force to be computed is:

$$\frac{\mathbf{F}_{ji}^{short}}{\mathbf{r_{ji}}} = A_{ab}r_{ji}^{-14} + B_{ab}r_{ji}^{-8} + QQ_{ab}(r_{ji}^{-3} + \frac{g_a'(r)}{r}). \quad (7)$$

2.3 Computing Short-Range Forces with Table Look-up

Since these calculations are in the "inner loop," considerable care is taken in their implementation: even in serial codes, the LJ equation is often not evaluated directly, but rather with table look-up and interpolation (TLwI). Previous implementations of FPGA/MD have used table look-up for the entire LJ force as a function of particle separation [3, 8]. The index used is $|r_{ji}|^2$ rather than $|r_{ji}|$ so as to avoid the costly square-root operation. This method is efficient for uniform gases where only a single table is required [3], but is less likely to be preferred in more general cases.

In more recent work [10], we use a different method: Instead of implementing the force pipeline with a single table lookup, we use three, one each for r^{-14}, r^{-8} and $r_{ji}^{-3} + \frac{g_a'(r)}{r}$. Equation 7 can be rewritten as a function of r_{ji}^2:

$$\frac{\mathbf{F}_{ji}^{short}(|r_{ji}|^2(a,b))}{\mathbf{r_{ji}}} =$$

$$A_{ab}R_{14}(|r_{ji}|^2) + B_{ab}R_8(|r_{ji}|^2) + QQ_{ab}R_3(|r_{ji}|^2), \quad (8)$$

where R_{14}, R_8, and R_3 are lookup tables indexed with $|r_{ji}|^2$.

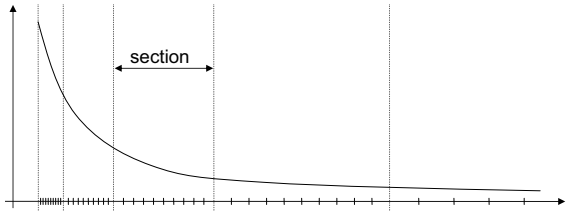

Figure 1: Table look-up varies in precision across r^{-k}. Each section has a fixed number of *intervals*.

The intervals in the tables are represented in Figure 1. Each curve is divided into several sections along the X-axis such that the length of each section is twice that of the previous. Each section, however, is cut into the same number of intervals N. To improve the accuracy, higher order terms can be used. When the interpolation is order M, each interval needs $M+1$ coefficients, and each section needs $N*(M+1)$ coefficients:

$$F(x) = a_0 + a_1x + a_2x^2 + a_3x^3 \quad (9)$$

shows third order with coefficients a_i. Naturally, accuracy increases with both the number of intervals per section and the interpolation order as shown in Figure 2. These issues are discussed in detail in [10].

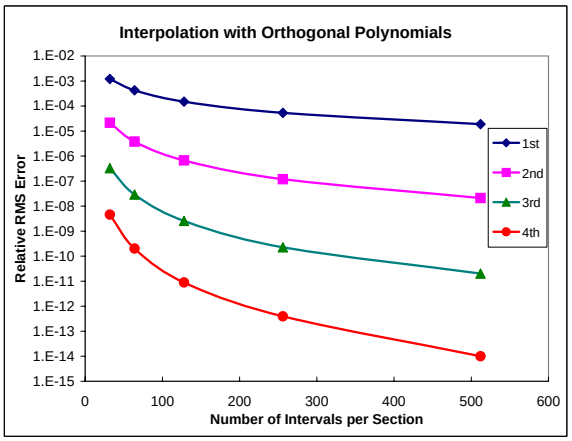

Figure 2: Comparisons of various orthogonal interpolations. Varied are the order and the number of intervals per section.

3 FPGA Implementations

3.1 Overview

In this section we describe FPGA implementations of Equation 7. All are pipelined and, on every cycle, input particle pair positions and output corresponding

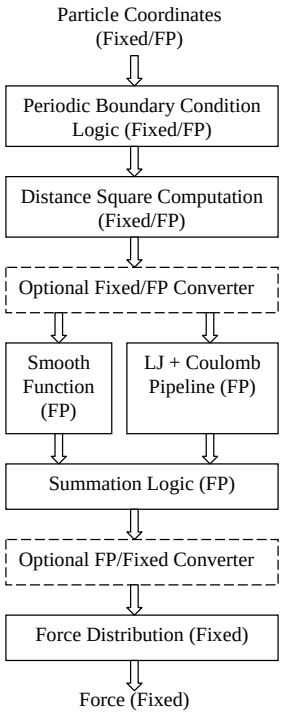

Figure 3: Shown is the functional block diagram of the short-range particle-particle datapath.

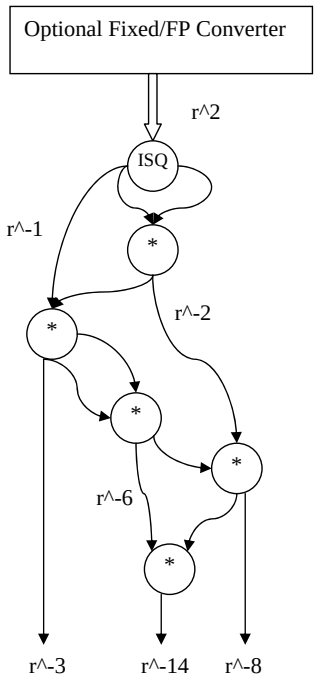

Figure 5: Shown is the Coulomb + LJ block (from Figure 4) modified with respect to the FP Compiler optimization.

forces. There are numerous design axes as described in Section 1. The ones that visibly affect the design are as follows: direct computation versus table lookup with interpolation (TLwI); for TLwI, order of interpolation; for direct, whether the Altera FP Compiler is used or the FP cores directly; and whether integer is used for part of the computation.

The last two require further explanation. The Altera floating point compiler optimizes floating point datapaths. As described in [16], there are two sources of performance gain. The first is that expressions are examined for functional redundancy among operators: since only the end result must comply with the FP standard, intermediate operations can often be removed. For example, with several consecutive FPADDs some normalizations can probably be avoided. The second is that the compiler makes trade-offs in using various component types, e.g., using hard or soft components as available. Here, the use of the compiler results in a different datapath being optimal.

The second axis requiring explanation is float versus hybrid fixed/float. The problem arises in the final force accumulation at the end of the pipeline. The new force component for each particle needs to be added to the running total. The floating point addition requires more than one cycle. Since this is pipelined, this does not change throughput. But if the same particle's force is referenced on successive cycles a hazard results. There are several solutions. Either the pipeline can stall, the particles can be orchestrated so that this does not happen, the forces can be combined in a longer structure, or the force can be saved in integer. In this last case, addition takes only a single cycle. Integer operations are also more efficient than floating point, and if done carefully, result in no loss of precision. The GROMACS code, for example, uses mixed fixed/floating point [29].

3.2 Direct Computation

The direct computation flow is shown in Figure 3 and detail in Figure 4. Note that the computation through obtaining r^2 can be done in fixed point. In that case, it must be converted to floating point before being combined with the coefficients (on the smoothing side) and divided (on the Coulomb + LJ side). The conversion at the output is similar.

Figure 5 shows the Coulomb plus LJ block that should be replaced when using the compiler. This is because there is a highly efficient inverse square root, which replaces the square root and the divide. This is because there are good convergence algorithms for inverse square root, but not square root, so this can be implemented with multipliers rather than logic. This both saves logic and increases speed.

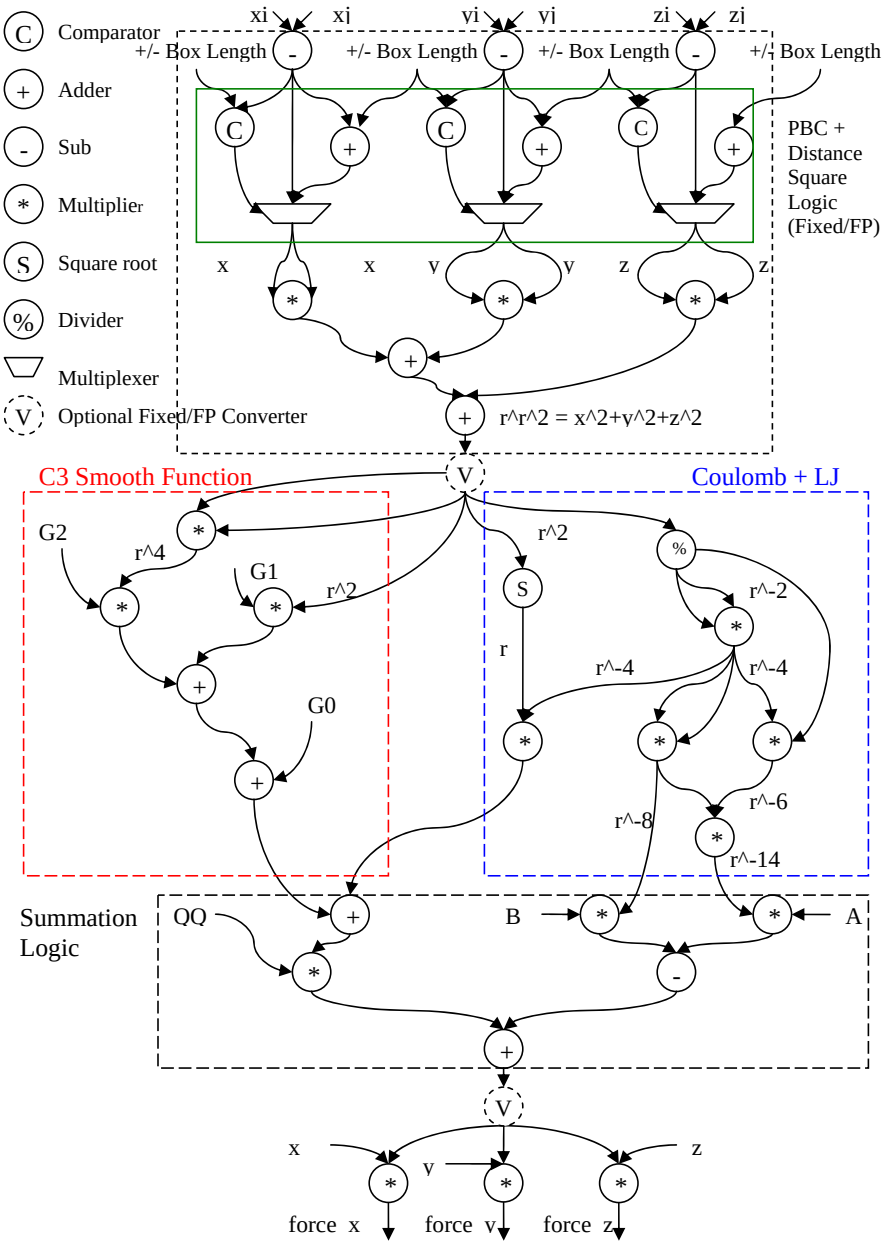

Figure 4: Shown is the detailed datapath for the direct computation. For hybrid integer/floating point, the computation is the same, but with data before the first conversion and after the second being integer.

3.3 Table Lookup with Interpolation

We now describe the interpolation pipeline (see Figure 6). Given that the interpolation function is third order, it necessarily has the format

$$F(x) = ((C_3(x-a)+C_2)(x-a)+C_1)(x-a)+C_0,$$

where $x \equiv r^2$ = input, a = the index of the interval from the beginning of the section (see Figure 1), and $x-a$ = the offset into the interval. The coefficients $C_0, \ldots, C_3$ are unique to each interval, and are retrieved by determining the section and interval. Proper encoding makes trivial the extraction of the section, interval, and offset.

Figure 7 contains the replacement in Figure 4 needed to implement TLwI. For lower order interpolation, fewer stages are needed.

4 Results

4.1 Reference Codes

We use NAMD [18] and ProtoMol [17] as reference codes both to determine the number of short-range particle-particle interactions computed per iteration as

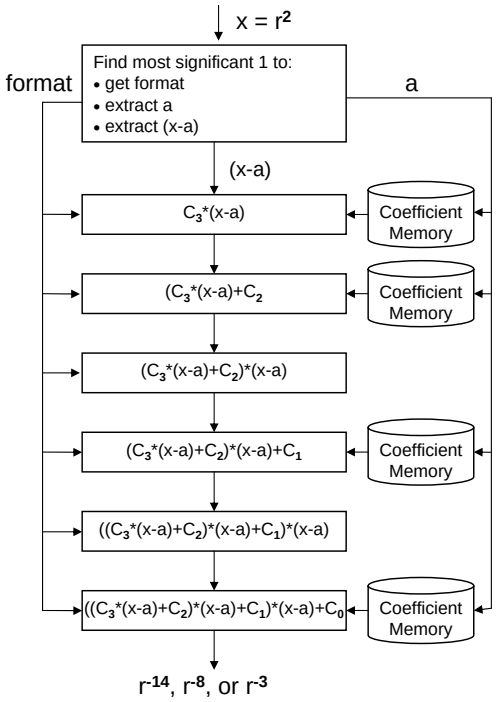

Figure 6: Position of the leading 1 determines the operand format in the interpolation pipeline.

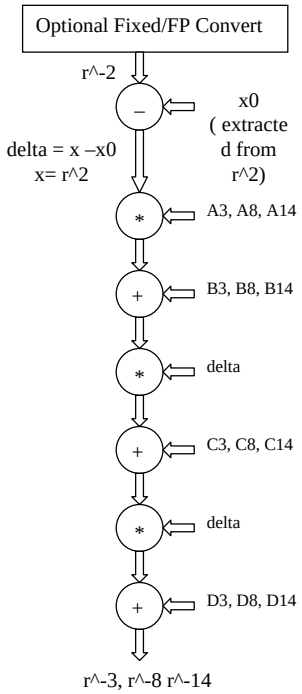

Figure 7: Shown is the Coulomb + LJ block (from Figure 4) modified for Table Lookup with Interpolation. Interpolation is 3rd order. Represented are three computations being done at once.

well as the time per iteration per core. NAMD scales well with multiple cores and multiple processors up to hundreds of processors.

We refer to the NAMD benchmark NAMD2.6 on ApoA1. It has 92,224 particles, a bounding box of $108\mathring{A} \times 108\mathring{A} \times 78\mathring{A}$, and a cut-off radius of $12\mathring{A}$. By instrumenting the codes, we determine that on average 33.4M non-trivial particle-particle computations are performed per iteration. According to a study by Stone, et al. [26], this benchmark is executed at 1.78 seconds per iteration on a single core of an Intel core 2 quad-core 2.66 GHz processor.

4.2 Performance Results

The two metrics for evaluating our designs are potential performance, described in this subsection, and simulation quality, described in the next.

We obtain the potential performance by multiplying the number of results obtained per cycle by the operating frequency. Since pipelines each generate a result per cycle, their number is the same as the results per cycle. Results are through post place-and-route (PaR) using the standard Altera tool chain. This method is sufficient to give precisely the number of pipelines. For operating frequency, true implementations are often slightly lower. On the other hand, the floating point cores (and code compiled using the Altera Floating Point Compiler or FPC) are specified to run at more than 250MHz, so with some optimization higher performance than that shown should be realized.

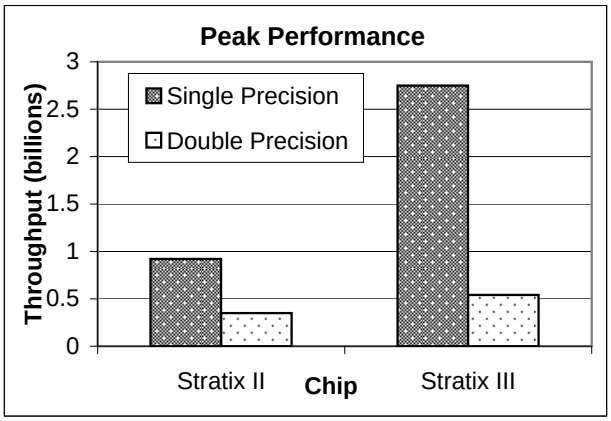

Figure 8: Peak performance in throughput of particle-particle force computations using direct computation.

1. Highest performing configuration. The results from the highest performing configuration are shown in Figure 8. Eleven pipelines run at 250MHz for a potential peak performance of 2.75G particle-particle force computations per second. The design configuration is as follows:

- Direct computation (no table look-up)
- Single precision floating point
- Generated using FPC
- Stratix-III EP3SE260

For the same configuration but with double precision floating point, 3 pipelines run at 180MHz. For the Stratix-II EP2S180, the capacity is 5 pipelines running at 184MHz for single precision and 2 pipelines running at 177MHz for double precision.

Table 1: Results for short-range particle processing. Potential speed-up is for single precision floating point on the Stratix-III versus a single core running NAMD. Efficiency refers to filtering method used by FPGA, and ranges from the straight-forward to the challenging.

Precision	cell size = cut-off (14% effic.)	cell size = cut-off/2 (24% effic.)	perfect filtering (100% effic.)
Single	20 ×	35 ×	146 ×
Double	4 ×	7 ×	28 ×

We now compare with the NAMD result just referenced, again stating that our goal is to find the bounds of peak performance (using throughput of short-range force computations as a metric), and not a comparison of complete systems.

A primary consideration in evaluating the FPGA performance is the control structure that ensures that computations entering the pipeline are performing useful work, i.e., are for particle pairs within the cut-off radius. MD codes generally do this in two parts. First, for each particle, a set of neighborhood particles is determined using either cell or neighbor lists. This reduces the computation by 3-4 orders of magnitude; there remain, however, 5-10 times as many particles as are within the cut-off radius. For serial codes, a simple distance computation can be used to determine whether further processing is necessary. For the benchmark referred to above, there remain an average of 33.4M short-range force computations per iteration.

On FPGAs, filtering (removing useless computations) is potentially costly: we already have highly parallel structures, and additional routing can take significant chip area. In Table 1 we illustrate three possible scenarios. In the first column, we show the potential acceleration using a cell size equal to the cut-off radius and no further filtering. This method is straightforward and was used by Gu [10]. The second column shows the potential acceleration if smaller cell sizes can be implemented efficiently. The third column shows the potential acceleration if perfect filtering can be implemented.

In the rest of this subsection, we examine the effect of varying, one at a time, a particular design parameter. All other parameters remain as in the highest performing configuration.

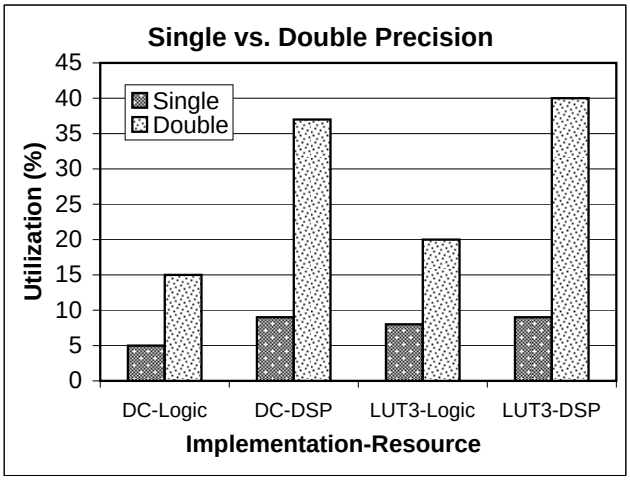

Figure 9: Precision comparison on the Stratix-III.

2: Single precision versus double precision. A comparison between single and double precision is shown in Figure 9. Although the latest Stratix-III chips have substantial floating point support, this does not yet result in direct scaling from single to double precision. The increased resources required is 2.5× - 3× for logic, but 4× - 4.5× for the multipliers. Also, the operating frequency is reduced, but the quality improves.

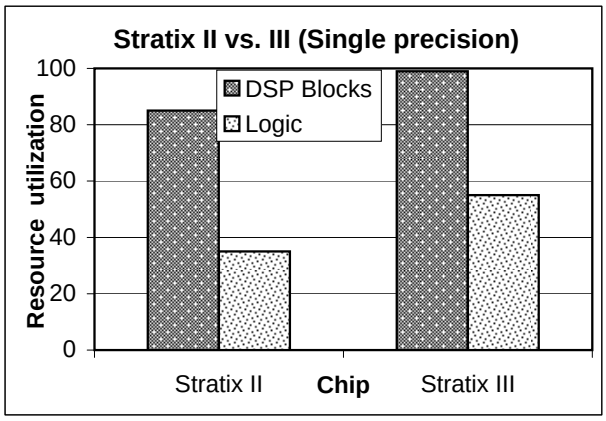

Figure 10: FPGA version comparison.

3: Stratix-II versus Stratix-III. A more detailed comparison of the Stratix-II and Stratix-III is shown in Figure 10. Interesting is that a greater fraction of non-DSP logic is used on the Stratix-III.

4: Effect of arithmetic implementation. Figure 11 shows the resource usage of various implementations. Direct computation (DC) uses less than 10% of the DSP units and far less of the remaining logic. The 3rd

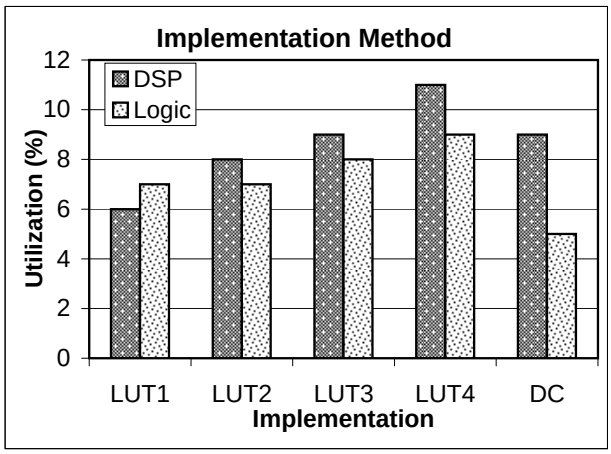

Figure 11: Comparison of implementation costs of various pipeline implementations. LUT1 to LUT4 are TLwI of orders 1 to 4, DC is direction computation. All are single precision on the Stratix-III.

order TLwI uses a similar fraction of DSP units, but substantially more logic. Going to 2nd and 1st order allow the implementation of perhaps another pipeline or two, but are not likely to be worth the decrease in simulation quality. Overall, this is a surprising result. In previous studies we found that TLwI was superior. We attribute the change to advances in floating point support and compilation in new generation FPGAs.

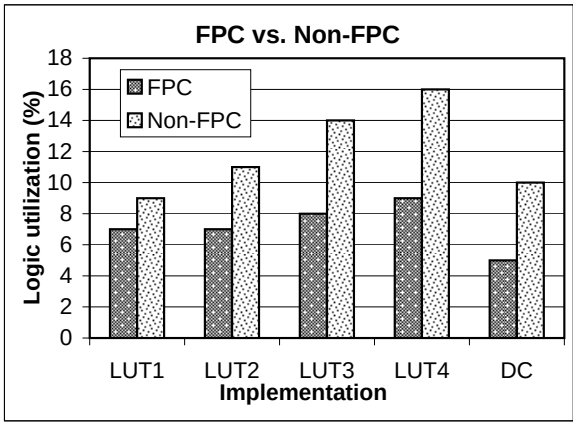

Figure 12: Effect of using the Altera FPC on logic utilization. For single pipeline, single precision, Stratix-III. Same configurations as in Figure 11.

5: Floating Point Compiler versus core only. The effect of using the Altera Floating Point Compiler is shown in Figure 12. This computation (short-range force pipeline) does not take advantage of most of the compiler optimizations, but still results in a substantial reduction in non-DSP logic. This is likely to be critical for implementing more aggressive filtering. In particular, note that in Figure 10 nearly 100% of the DSP blocks are used, but less than 60% of the remaining logic. Without the FPC, the latter would also be close to 100% resulting in an unroutable design.

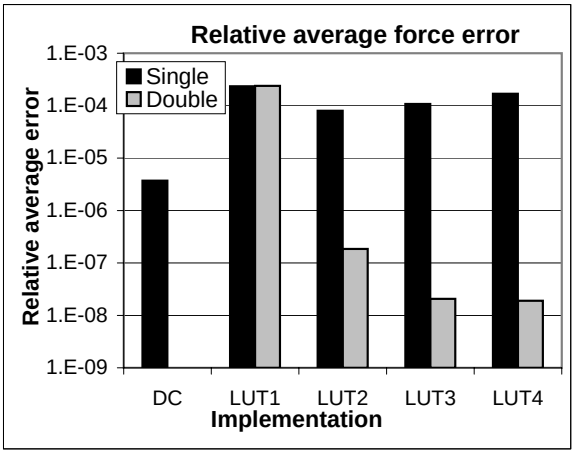

Figure 13: Relative average force error for the particle-particle force computation for various implementations and precisions. DC is direction computation, LUTn refers to TLwI of various orders.

4.3 Quality Results

Direct evaluations of MD simulation quality, such as through wet-lab experiments, are often impractical. Thus surrogates are often used. One type is to measure the errors with respect to a reference computation. Another type monitors the simulation output, e.g., to confirm that a physical invariant, such as the total energy, actually is so. Here we use two of each type.

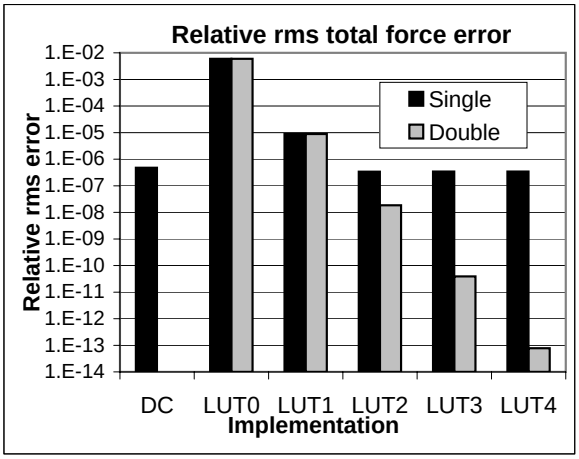

Figure 14: Relative average force error for the total force on a particle per iteration for various implementations and precisions. DC is direction computation, LUTn refers to TLwI of various orders.

1. Error per individual particle-particle force computation. Figure 13 shows the relative average error for the individual particle-particle force computations for the various pipeline implementations. The

reference is direct computation using double precision (DC Double, error = 0). We generate the particle pairs by randomly selecting particle positions between the cut-off and exclusion radii. For single precision TLwI, error becomes *worse* for higher orders. This is because of the higher precision required for those tables.

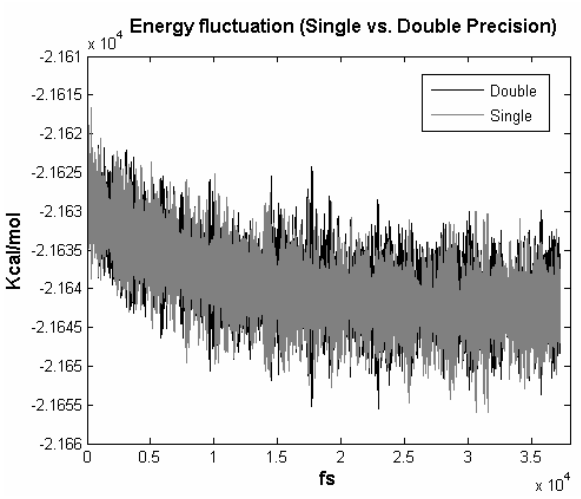

Figure 15: Energy profile of BPTI with 14K particles.

2. Error per total force on a particle per iteration. Figure 14 shows the relative rms force error of the total force on a particle (see, e.g., [22, 25, 31]). The reference is direct computation using double precision (DC Double, error = 0). LUT0 refers to TLwI with no interpolation (as in [31]). All except for LUT0 appear to exceed the quality criteria in Shaw, et al. [23].

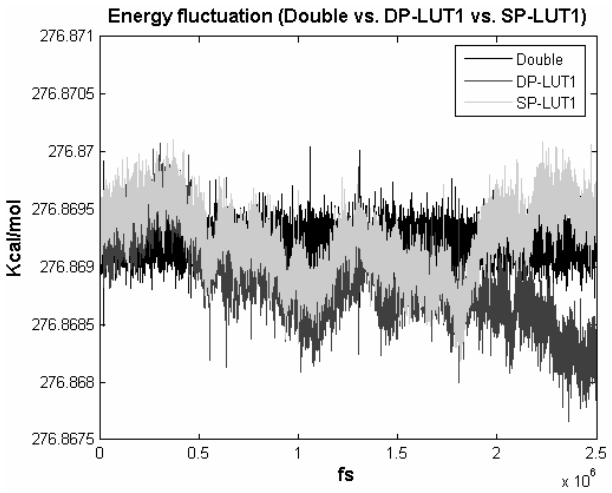

Figure 16: Energy profile of argon with 400 particles.

3. Energy fluctuation. We simulated BPTI with 14K particles and 26 particle types. After 37K time steps (see Figure 15), the energy fluctuations [2] for direct force computation are 2.48×10^{-4} and 2.62×10^{-4} for double precision and single precision, respectively. The ratios of the fluctuations between total energy and kinetic energy are 0.0402 and 0.0422; both surpass the .05 suggested in [28].

4. Energy drift. The final quality measure we refer to as "energy drift." We simulated 400 Argon atoms (as per the ProtoMol test case); the results are shown in Figure 16. Note that while the double precision direct computation remains stable, both simulations with 1st order TLwI appear to drift. That is, not only is there an envelope within which the energy fluctuates, the envelope itself fluctuates. This behavior may be less likely to be acceptable.

5 Discussion and Future Work

From Figure 11 we see that direct computation is somewhat favorable to table lookup with interpolation, except when 1st order is used. The reduction in accuracy, however, may not be acceptable (see Figure 16). Other results are a demonstration of the Altera floating point compiler, and numerous observations with respect to datapath design parameters. The most important of these is probably that the reduction in simulation quality (by standard measures) of going from double to single precision appears to be negligible.

Our initial goal was to bound possible performance gain of the HPRC/MD over core-based MD. We have found that HPRC/MD continues to be highly competitive. Each Stratix-III pipeline can provide performance, by the metric of this paper, of about 13× a core. The best implementation (other than the low order table lookup) supports 11 pipelines for a total potential speed-up of about 143×. The corresponding generation of microprocessor has four cores, so a chip to chip comparison would be about 35×. Of course achieving this throughput requires (at least) highly efficient routing and filtering.

Again, none of these results are comparisons. Our purpose was to continue exploring the HPRC/MD design space, and to determine whether yet further work in this direction is warranted.

Acknowledgments. We thank the anonymous reviewers for their many helpful comments. We also thank Yongfeng Gu for being so generous with his time in answering numerous questions.

References

[1] Alam, S., Agarwal, P., Smith, M., Vetter, J., and Caliga, D. Using FPGA devices to accelerate biomolecular simulations. *Computer 40*, 3 (2007), 66–73.

[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 *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines* (2004), pp. 197–206.

[4] Bowers, K.J., et al. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *Proceedings of Supercomputing* (2006).

[5] Darden, T., York, D., and Pedersen, L. Particle Mesh Ewald: an $N\log(N)$ method for Ewald sums in large systems. *J. Chemical Physics 98* (1993), 10089–10092.

[6] Fitch, B.G, et al. Blue Matter: Approaching the limits of concurrency for classical molecular dynamics. In *Proceedigs of Supercomputing* (2006).

[7] Gu, Y., and Herbordt, M. High performance molecular dynamics simulations with FPGA coprocessors. In *Proceedings of the Reconfigurable Systems Summer Institute* (2007).

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

[9] Gu, Y., VanCourt, T., and Herbordt, M. Improved interpolation and system integration for FPGA-based molecular dynamics simulations. In *Proceedings of the IEEE Conference on Field Programmable Logic and Applications* (2006), pp. 21–28.

[10] Gu, Y., VanCourt, T., and Herbordt, M. Explicit design of FPGA-based coprocessors for short-range force computation in molecular dynamics simulations. *Parallel Computing 34*, 4-5 (2008), 261–271.

[11] Hamada, T., and Nakasato, N. Massively parallel processors generator for reconfigurable system. *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines* (2005).

[12] Izaguirre, J., Hampton, S., and Matthey, T. Parallel multigrid summation for the n-body problem. *Journal of Parallel and Distributed Computing 65* (2005), 949–962.

[13] Kindratenko, V., and Pointer, D. A case study in porting a production scientific supercomputing application to a reconfigurable computer. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines* (2006), pp. 13–22.

[14] Koeher, S., Curreri, J., and George, A. Performance analysis challenges and framework for high performance reconfigurable computing. *Parallel Computing 34*, 4-5 (2008), 217–230.

[15] 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. *Journal of Computational Chemistry 18*, 12 (1997), 1546–1563.

[16] Langhammer, M. Floating point datapath synthesis for FPGAs. In *Proceedings of the IEEE Conference on Field Programmable Logic and Applications* (2008), pp. 355–360.

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

[18] Phillips, J., Zheng, G., and Kale, L. NAMD: biomolecular simulation on thousands of processors. In *Supercomputing* (2002).

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

[20] Rodrigues, C., Hardy, D., Stone, J., Schulten, K., and Hwu, W.-M. GPU acceleration of cutoff pair potentials for molecular modeling applications. In *Proc. ACM Int. Conf. on Computing Frontiers* (2008).

[21] Scrofano, R., and Prasanna, V. Preliminary investigation of advanced electrostatics in molecular dynamics on reconfigurable computers. In *Supercomputing* (2006).

[22] Shan, Y., Klepeis, J., Eastwood, M., Dror, R., and Shaw, D. Gaussian split Ewald: A fast Ewald mesh method for molecular simulation. *Journal of Chemical Physics 122*, 4 (2005).

[23] Shaw, D.E., et al. Anton, a special-purpose machine for molecular dynamics simulation. In *Proceedings of the International Symposium on Computer Architecture* (2007), pp. 1–12.

[24] Shi, G., and Kindratenko, V. Implementation of NAMD molecular dynamics non-bonded force-field on the Cell Broadband Engine processor. In *Proc. 9th Int. IEEE Workshop on Parallel and Distributed Scientific and Engineering Computing* (2008).

[25] Skeel, R., Tezcan, I., and Hardy, D. Multiple grid methods for classical molecular dynamics. *Journal of Computational Chemistry 23* (2002), 673–684.

[26] Stone, J., Phillips, J., Freddolino, P., Hardy, D., Trabuco, L., and Schulten, K. Accelerating molecular modeling applications with graphics processors. *Journal of Computational Chemistry 28* (2007), 2618–2640.

[27] 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).

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

[29] van der Spoel, D., Lindahl, E., Hess, B., Groenhof, G., Mark, A., and Berendsen, H. GROMACS: fast, flexible, and free. *Journal of Computational Chemistry 26* (2005), 1701–1718.

[30] Villareal, J., Cortes, J., and Najjar, W. Compiled code acceleration of NAMD on FPGAs. In *Proceedings of the Reconfigurable Systems Summer Institute* (2007).

[31] Wolff, D., and Rudd, W. Tabulated potentials in molecular dynamics simulations. *Computer Physics Communications 120* (1999), 20–32.