EFFICIENT PARTICLE-PAIR FILTERING FOR ACCELERATION OF MOLECULAR DYNAMICS SIMULATION

Matt Chiu *Martin C. Herbordt*

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

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 GPUs, there has been a question whether MD on HPRC can be competitive. We concentrate here on the MD kernel computation: determining the short-range force between particle pairs. In particular, we present the first FPGA study on the filtering of particle pairs with nearly zero mutual force, a standard optimization in MD codes. There are several innovations, including a novel partitioning of the particle space, and new methods for filtering and mapping work onto the pipelines. As a consequence, highly efficient filtering can be implemented with only a small fraction of the FPGA's resources. Overall, we find that, for an Altera Stratix-III EP3ES260, 8 force pipelines running at 200MHz can fit on the FPGA, and that they can perform at 95% efficiency. This results in a 80-fold per core speed-up for the short-range force, which is likely to make FPGAs highly competitive for MD.

1. INTRODUCTION

Acceleration of MD is 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, clusters, and dedicated hardware, as well as coprocessing using GPUs, Cell, and FPGAs. The last of these is our focus here.

We concentrate on the MD kernel computation: determining the short-range force between particle pairs. In particular, we study the filtering of particle pairs with nearly zero mutual force, a standard optimization in MD codes. When coupled with our recent work on the optimization of MD force pipelines [1], this has the potential to improve performance of our previous full MD implementation [2] four-fold.

The issue of filtering particle pairs emerges from the geometric mismatch between two shapes: (i) the cubes (or other polyhedrons) into which it is convenient to partition the simulation space and (ii) the spheres around each particle in which the short-range force is non-zero. If this mismatch is not addressed (e.g., only the standard cell-list method is used), then 85.5% of the particle pairs that are run through the force pipelines will be superfluous. While filtering is a critical issue, we believe that the only previously published results related to hardware implementations are from D.E. Shaw; these are with respect to their Anton processor [3].

Here, we find filtering implementation on FPGAs to provide a rich design space. Its primary components are:

- Filter algorithm and precision,
- Method of partitioning the cell neighborhood to balance load with respect to the Newton's-3rd-Law optimization,
- Method of mapping particle pairs to filter pipelines, and
- Queueing and routing between filter and force pipelines.

We present new algorithms or methods for filtering, load balancing, and mapping, and find that nearly perfect filtering can be achieved with only a fraction of the FPGA's logic, and with a tolerable fraction of its BRAMs.

Our basic result is that for the Stratix-III EP3SE260, and for the best (as yet unoptimized) designs, 8 force pipelines running at about 200MHz can fit on the FPGA. Moreover, the force pipelines can be run at high efficiency with 95% of cycles providing payload. As a result, the short-range force for the standard 90K NAMD benchmark can be computed in under 22ms, or about a factor of 80 times faster than its per-core execution time. Contributions are two-fold: (i) demonstrating that FPGAs are highly competitive for MD and (ii) presenting the first study of particle-particle filtering on FPGAs and with it a number of innovations. The last of these may have implications to MD beyond HPRC.

The rest of this paper is organized as follows. In the next section, we review the applicable parts of MD simulation. There follows the presentation of the filtering methods, after which come results and some discussion.

2. MD PRELIMINARIES

2.1. MD review

MD simulation is an iterative application of Newtonian mechanics to ensembles of atoms and molecules. It proceeds in phases, alternating between force computation and motion integration. The forces may include van der Waals (LJ),

This work was supported in part by the NIH through award #R01-RR023168-01A1. Web: www.bu.edu/caadlab. Email: {herbordt|mattchiu} @bu.edu

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, their complexity 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, i.e., by requiring that the force on each particle is the result of only particles within the cut-off radius r_c. 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.

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 solution is to split the Coulombic force into short- and long-range components. The short-range is then computed together with the LJ to form the *short-range force*. The long-range component of the Coulombic force is the *long-range force*. Since the long-range force computation is generally a small fraction of the total [2, 4], our focus here is on the short-range force. Given a particle pair i and j:

$$\frac{\mathbf{F}_{ji}^{short}}{\mathbf{r_{ji}}} = A_{ab} r_{ji}^{-14} + B_{ab} r_{ji}^{-8} + QQ_{ab} (r_{ji}^{-3} + \frac{g_a'(r)}{r}) \quad (4)$$

where A_{ab}, B_{ab}, and QQ_{ab} are distance-independent coefficient look-up tables indexed with atom types a and b, and g' is a smoothing function for splitting the Coulombic force.

2.2. Filtering particle pairs

While MD in general involves all-to-all forces among particles, a cut-off is commonly applied to restrict the extent of the short-range force to a fraction of the simulation space. Two methods are used to take advantage of this cut-off: cell lists and neighbor lists (see Figure 1). With cell lists, the simulation space is typically partitioned into cubes with edge-length equal to r_c. Non-zero forces on the *reference particle* P can then only be applied by other particles in its *home cell* and in the 26 neighboring cells (the 3x3x3 *cell neighborhood*). We refer the second particle of the pair as the *partner particle*. With neighbor lists, P has associated with it a list of exactly those partner particles within r_c. We now compare these methods.

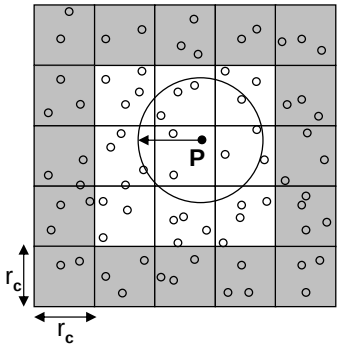

Fig. 1. P's two dimensional *cell neighborhood* is shown in white; cells have edge size equal to the cut-off radius. Particles within the P's cut-off circle are in P's neighbor list.

- **Efficiency.** Neighbor lists are 100% efficient: only those particle pairs with non-zero mutual force are evaluated. Cell lists as just defined are 15.5% efficient with that number being the ratio of the volumes of the cut-off sphere and the 27-cell neighborhood.
- **Storage.** With cell lists, each particle is stored in a single cell's list. With neighbor lists, each particle is typically stored in 400-1000 neighbor lists.
- **List creation complexity.** Computing the contents of each cell requires one pass through the particle array. Computing the contents of each neighbor list requires, naively, that each particle be examined with respect to every other particle: the distance between them is then computed and thresholded. In practice, however, it makes sense to first compute cell lists anyway. Then the neighbor lists can be computed using only the particles in each reference particle's cell neighborhood.

From this last point, it appears that the creation of neighbor lists involves not only cell lists, but also a fraction of the force computation itself. At this point, why not finish computing the forces of those particles that are within the cut-off? Why save the neighbor list?

Most MD codes reuse the neighbor lists for multiple iterations and so amortize the work in their creation. But because particles move during each iteration, particles can enter and exit the cut-off region leading to potential error. The solution is to make the neighborlist cut-off larger than the force cut-off, e.g., 13.5Å versus 12Å. There is a trade-off between the increase in neighborhood size (and thus the number of particle pairs evaluated) and the number of iterations between neighbor list updates.

3. FILTERING – HIGH LEVEL ISSUES

3.1. Coprocessor-specific preliminaries

With MD coprocessing there are additional considerations. The cell list computation is very fast and the data generated small so it is generally done on the host (along with the motion integration): the cell lists are downloaded to the coprocessor every iteration along with the new particle positions. The neighbor list computation, however, is much more expensive: if done on the host it could mitigate any advantage of coprocessing. Moreover, the size of the aggregate neighbor lists is hundreds of times that of the cell lists, which makes their transfer impractical. As a consequence, neighbor list computation, if it is done at all, must be done on the coprocessor. But even on the coprocessor storage is still a concern.

We look first at MD with cell lists. For reference and without loss of generality we examine 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 $12\mathring{A}$. This yields a simulation space of $9 \times 9 \times 7$ cells with an average of 175 particles per cell with a uniform distribution. On the FPGA, the working set is typically a single (home) cell and its cell neighborhood for a total of (naively) 27 cells and about 4,725 particles.

In actuality, Newton's 3rd Law (N3L) is used to reduce this number. That is, since each particle-particle interaction is mutual, it is calculated once per particle pair and recorded for both particles. To effect the reduction in work, home cell particles are only matched with particles of part of the cell neighborhood (see Section 5 for two options), and with, on average, half of the particles in the home cell. We refer to the subset of cells in the cell neighborhood that are processed together with (and including) the home cell as the *cell set*. For the 14- and 18-cell sets presented below, the average number of particles to be examined (for each particle in the home cell) is 2,450 and 3,150, respectively. Given current FPGA technology, any of these cell sets (14, 18, or the original 27 cells) easily fits in the on-chip BRAMs.

Neighbor lists for a home cell do not fit on the FPGA. For example, the aggregate neighbor lists for 175 home cell particles is over 64,000 particles (one half of 732 for each of the 175 particles; 732 rather than 4,725 because of increased efficiency of neighbor lists over cell lists).

The memory requirements are therefore very different for the two methods. For cell lists, we swap cells onto and off of the FPGA as needed. Because of the high level of reuse, this is easily done in the background. In contrast, neighbor list particles must be streamed from off-chip. This has worked when there are one or two force pipelines operating at 100MHz [5, 4], but is problematic for current and future high-end FPGAs. For example, the Stratix-III/Virtex-5 generation of FPGAs supports at least 8 force pipelines operating at 200MHz leading to a bandwidth requirement of over 20 GB/s.

From this discussion, it follows that use of neighbor lists calls for an "on-FPGA" solution, but that this itself appears to be impracticable due to memory and transfer requirements. At the same time, however, the 6x potential increase in efficiency cannot be abandoned.

One way to improve efficiency is to reduce the cell size: the smaller the cell size, the finer the granularity, and the larger the fraction of the cell neighborhood volume guaranteed to be useful. With a cell edge of $r_c/2$ and a 5^3 set, efficiency increases to 26.8%. With more aggressive clipping of the corner cells, efficiency increases a bit more but so does the control complexity. More important is that reducing cell size also reduces reuse and still leaves much inefficiency. While reducing cell size is viable, there are better options.

3.2. Overall design and board-level issues

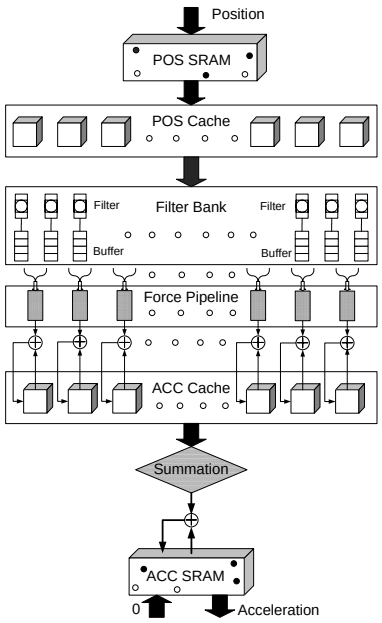

Fig. 2. Schematic of the HPRC MD system.

The solution is to use neighbor lists, but to compute them every iteration, generating them continuously and consuming them almost immediately. In this scenario, the use of neighbor lists can be viewed as *filtering* out the zero-force particle pairs: the filter pipelines feed the force pipelines with minimal buffering in between (see Figure 2). The next several sections present designs and tradeoffs for this solution.

We now describe the execution flow. Processing is built around the home cell. Position and acceleration data of the particles in the cell set are loaded from board memory into on-chip caches, POS and ACC, respectively. When the processing of a home cell has completed, focus shifts and a neighboring cell becomes the new home cell. Its cell set is now loaded; in our current scheme this is nine new cells.

Acceleration data differs from position data in that it is read/write. That is, each particle's acceleration accumulates

over this and other home cells. It is not complete for any given home cell until all 27 cells in its cell neighborhood have also been the home cell. Therefore the nine cells of acceleration data are swapped rather than just overwritten.

One design constraint is that each force pipeline should handle at most a small number of reference particles P_i at a time. This enables the total forces on the P_is to be accumulated in registers. Accumulating the mutual forces on the P_is' N3L partner particles, however, is more complex as their positions span the cell set. To prevent BRAM access contention, the following strategy is used. Partner updates are written to BRAMs associated uniquely with each force pipeline. When processing of a home cell is completed, the partner data from the various pipeline-specific BRAMs are merged. This operation is performed during swapping out, so latency is completely hidden.

The time to process a home cell T_{proc} is generally greater than the time T_{trans} to swap cell sets with off-chip memory. Assume that a cell has edge length = r_c and contains on average N_{cell} particles. Then $T_{trans} = 324 \times N_{cell}/B$ (9 cells, 32-bit data, 3 dimensions, 2 reads and 1 write, and transfer bandwidth of B bytes per cycle). To compute T_{proc}, assume P pipelines and perfect efficiency. Then $T_{proc} = N_{cell}^2 \times \pi/2P$ cycles. This gives the following bandwidth requirement: $B > 206 * P/N_{cell}$. For $P = 10$ and $N_{cell} = 175$, $B > 12$ bytes per cycle. For many current FPGA processor boards $B \geq 16$. Some factors that increase the bandwidth requirement are faster processor speeds, more pipelines, and lower particle density. A factor that reduces the bandwidth requirement is better cell reuse.

4. FILTERING METHODS

We begin by assuming cell lists with processing concentrating on one home cell at a time. With no filtering or other optimization, forces are computed between all pairs of particles i and j, where i must be in the home cell but j can be in any of the 27 cells of the cell neighborhood, including the home cell. By *filtering* we mean the identification of particle pairs where the mutual short-range force is zero. A *perfect filter* successfully removes all such pairs. The *efficiency* of the filter is the fraction of undesirable particle pairs removed. The *extra work* due to imperfection is the ratio of undesirable pairs not removed to the desirable pairs.

We evaluate three methods, two existing and one new, which trade off efficiency for hardware resources. As motivated elsewhere [1], we store particle positions in three Cartesian dimensions, each in 32-bit integer. There are two parameters, precision and geometry.

1. Full Precision: Precision = full, Geometry = sphere
Computes $r^2 = x^2 + y^2 + z^2$ and compares whether $r^2 < r_c^2$ using full 32-bit precision. Filtering quality in this case is nearly 100%. Except for the comparison operation, this is the same computation that is performed in the force pipeline.

2. Reduced: Precision = reduced, Geometry = sphere
This filter, used by D.E. Shaw [3], also computes $r^2 = x^2 + y^2 + z^2, r^2 < r_c^2$, but uses fewer bits and so substantially reduces the hardware required. Lower precision, however, means that the cut-off radius must be increased (rounded up to the next bit) so filtering efficiency goes down: for 8 bits of precision, it is 99.5%. In our reference example, each particle is now matched with about 378 particles, rather than the 366 for perfect filtering, for about 3% extra work.
3. Planar: Precision = reduced, Geometry = planes
A disadvantage of the previous method is its use of multipliers, which are the critical resource in the force pipeline. This issue can be important because there are likely to be 6 to 10 filter pipelines per force pipeline. In this method we avoid multiplication by thresholding with planes rather than a sphere. The formulae are as follows:

- $|x| < r_c, |y| < r_c, |z| < r_c$
- $|x| + |y| < \sqrt{2}r_c, |x| + |z| < \sqrt{2}r_c, |y| + |z| < \sqrt{2}r_c$
- $|x| + |y| + |z| < \sqrt{3}r_c$

With 8 bits, this method achieves 97.5% efficiency, or about 13% extra work.

Analysis
Table 1 summarizes the resource cost and quality (efficiency and extra work) of the three filtering methods. Since multipliers are a critical resource, we also show the two "sphere" filters implemented entirely with logic. The cost of a force pipeline (from [1]) is shown for scale.

The most important result is the relative cost of the filters to the force pipeline. Depending on implementation, each force pipeline needs between 6 and 10 filters to keep it running at full utilization. We refer to that set of filters as a *filter bank*. Table 1 shows that a **full precision** filter bank takes from 80% and 170% of the resources of its force pipeline. The **reduced** and **planar** filter banks, however, require only a small fraction: between 17% and 40% of the logic of the force pipeline and no multipliers at all. Since the latter is the critical resource, the conclusion is that the filtering logic itself (not including interfaces) has negligible effect on the number of force pipelines that can fit on the FPGA.

We now compare the **reduced** and **planar** filters. The Extra Work column in Table 1 shows that for a **planar** filter bank to obtain the same performance as logic-only-**reduced**, the overall design must have 13% more throughput. This translates, e.g., to having 9 force pipelines when using **planar** rather than 8 for **reduced**. The total number of filters remains constant. The choice of filter therefore depends on the FPGA's resource mix.

5. BALANCING NEIGHBOR LIST SIZES

For efficient control and particle-memory access, and for smooth interaction between filter and force pipelines, it is preferred to have each force pipeline handle the interactions of a single

Table 1. Comparison of filtering schemes for quality and resource usage. A force pipeline is shown for reference. Percent utilization is for the Altera Stratix-III EP3SE260.

Filtering Method	LUTs/ Registers	Multi-pliers	Filter Eff.	Extra Work
Full prec.	341/881 0.43%	12 1.6%	100%	0%
(logic only)	2577/2696 1.3%	0 0.0%	100%	0%
Reduced	131/266 0.13%	3 0.4%	99.5%	3%
(logic only)	303/436 0.21%	0 0.0%	99.5%	3%
Planar	164/279 0.14%	0 0.0%	97.5%	13%
Force pipe	5695/7678 5.0%	70 9.1%	NA	NA

reference particle at a time. This preference becomes critical when there are a large number of force pipelines and a much larger number of filter pipelines. Moreover, it is highly desirable for all of the neighborlists being created at any one time (by the filter banks) to be transferred to the force pipelines simultaneously (buffering mechanisms are described in Section 6). It follows that each reference particle should have a similar number of partner particles (neighbor list size).

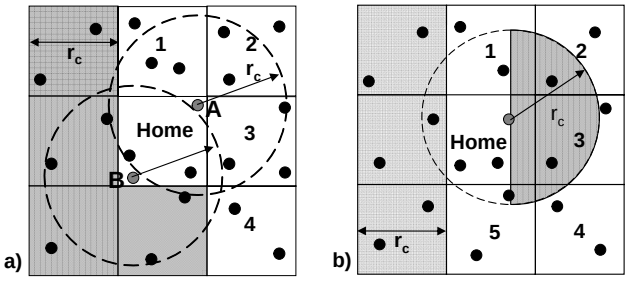

Fig. 3. Shown are two partitioning schemes for using Newton's 3rd Law. In a), 1-4 plus home are examined with a full sphere. In b), 1-5 plus home are examine, but with a hemisphere (shaded part of circle).

The problem addressed in this subsection is that the standard method of choosing a reference particle's partner particles leads to a severe imbalance in neighbor list sizes. How this arises can be seen in Figure 3a, which illustrates the standard method of optimizing for N3L. So that a force between a particle pair is computed only once, only a "half shell" of the surrounding cells is examined (in 2D, this is cells **1-4** plus **Home**). For forces between the reference particle and other particles in **Home**, the particle ID is used to break the tie, with, e.g., the force being computed only when the ID of the reference particle is the higher. In Figure 3a, particle B has a much smaller neighborlist than A, especially if B has a low ID and A a high.

In fact neighborlist sizes vary from 0 to $2L$, where L is the average neighborlist size. The significance is as follows. Let all force pipelines wait for the last pipeline to finish before starting work on a new reference particle. Then if that (last) pipeline's reference particle has a neighborlist of size $2L$, then the latency will be double that if all neighbor lists were size L. This distribution has high variance meaning that neighbor list sizes greater than, say, $\frac{3}{2}L$ are likely to occur. A similar situation also occurs in other MD implementations, with different architectures calling for different solutions [6].

One way to deal with this load imbalance is to overlap the force pipelines so that they work independently. While viable, this leads much more complex control.

An alternative is to change the partitioning scheme. Our new N3L partition is shown in Figure 3b. There are three new features. The first is that the cell set has been augmented from a half shell to a prism. In 2D this increases the cell set from 5 cells to 6; in 3D the increase is from 14 to 18. The second is that, rather than forming a neighbor list based on a cutoff sphere, a hemisphere is used instead (the "half-moons" in Figure 3b). The third is that there is now no need to compare IDs of home cell particles.

We now compare the two partitioning schemes. There are two metrics: the effect on the load imbalance and the extra resources required to prevent it.

1. Effect of load imbalance. We assume that all of the force pipelines begin computing forces on their reference particles at the same time, and that each force pipeline waits until the last force pipeline has finished before continuing to the next reference particle. We call the set of neighbor lists that are thus processed simultaneously a *cohort*. With perfect load balancing, all neighbor lists in a cohort would have nearly the same size, the average. The effect of the *variation* in neighbor list size is the number of *excess* cycles—before a new cohort of reference particles can begin processing—over the number of cycles if each neighborlist were the same size. The performance cost is therefore the average number of excess cycles per cohort. This in turn is the average size of the biggest neighbor list in a cohort minus the average neighbor list size. We find that, for the standard N3L method, the average excess is nearly 50%, while for the "half-moon" method it is less than 5%.

2. Extra resources. The extra work required to achieve load balance is proportional to the extra cells in the partition: 18 versus 14, or an extra 29%. This drops the fraction of neighbor list particles in the cell neighborhood from 15.5% to 11.6%, which in turns increases the number of filters needed to keep the force pipelines fully utilized (overprovisioned) from 7 to 9. For the reduced and planar filters, this is not likely to reduce the number of force pipelines.

6. FILTER PIPELINE DESIGN

6.1. Mapping particle pairs to filters

From the previous sections we converge on an efficient design for filtering particle pairs:

- During execution, the working set (data held on the FPGA) consists of the positions and accelerations of particles in a cell set; i.e., a single home cell and its 17 neighbors (in the "half moon" scheme);

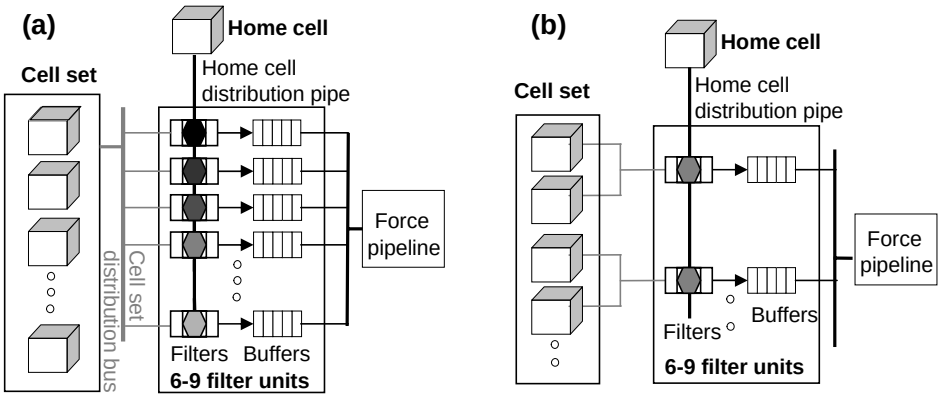

Fig. 4. (a) Particle Mapping: Filters each hold a different reference particle. Particles in cell set are broadcast one per cycle. (b) Cell Mapping: Same reference particle held by all filters in a bank. Each filter is responsible for 2-3 cells.

- Particles from each cell are stored in a set of BRAMs: this is currently one or two BRAMs per coordinate, depending on the cell size, for a total of 108-216;
- The N3L partition specifies 7-9 filters per force pipeline;
- FPGA resources indicate 8-10 force pipelines; and
- Force pipelines handle at most a small number of reference particles at a time (and their N3L partners).

We now address the mapping of particle pairs to filter pipelines. There are a large number of ways to do this; finding the optimal mapping is in some ways analogous to optimizing loop interchanges with respect to a cost function. Figure 4 shows two possibilities. In *particle mapping* (a), each *filter* is responsible for a different reference particle. Each cycle, a single partner particle from the cell set is broadcast to all of the filters (and also to the other filter banks). In *cell mapping* (b), each *filter bank* is collectively responsible for a different reference particle. Each filter within a bank processes the reference particle with respect to partners from its own subset of 2 or 3 cells. The issues are as follows.

Force pipeline efficiency. Overall performance is proportional to the efficiency of the force pipelines, i.e., the fraction of cycles that they deliver "payload" (non-zero forces). Since there are no stalls, the efficiency is proportional to the fraction of cycles that they input payload particle pairs.

Payload generation rate. Given sufficient filters, a filter bank will generate payload pairs at an average rate of greater than one per cycle. The variance may be high, however, which can substantially degrade efficiency.

Distribution of payload particle pairs. While the number of payload particle pairs from a given cell set—or a given reference particle—has small variance, the number and distribution of payload pairs generated by any particular filter can vary wildly. For example, in Figure 3b, let two filters (in a bank) each handle the same reference particle, but let the partner particles be from different cells, say 3 and 5. Each filter examines the same number of pairs, but the first filter passes most of its input while the second passes almost none.

Queueing particle pairs. A simple (but costly) solution is to: (i) append a large queue to each filter and (ii) implement a flexible router from these queues to the force pipeline.

The two mappings lend themselves to multiple queueing methods, the choice of which depends on resources available on the FPGA. In the next two subsections we present two queueing strategies, *whole neighborlist* and *continuous*. We evaluate them with respect to the two particle mapping strategies for performance (force pipeline efficiency) and hardware cost (queue size and complexity).

6.2. Queueing whole neighborlists

If there are sufficient BRAMs, then particle mapping can be used to generate neighborlists in their entirety and consumed in the same way. Details are as follows; we assume particle mapping, but the logic is similar for cell mapping.

- A phase begins with a new and distinct reference particle being associated with each filter.
- Then, on each cycle, a single particle from the 18-cell set is broadcast to all of the filters.
- Each filter's output goes to its own set of BRAMs.
- The output of each filter is exactly the neighborlist for its associated reference particle.
- Double buffering enables neighborlists to be generated by the filters at the same time that the previous phase's neighborlists are being drained by the force pipelines.

Advantages of this method include:

- Nearly perfect load balance among the filters (from the "half-moon" partition);
- Little overhead: each phase consists of over 3000 cycles before a new set of reference particles must be loaded;
- Nearly perfect load balancing among the force pipelines: each operates successively on a single reference particle and its neighborlist; and
- Simple queueing and control: neighborlist generation is decoupled from force computation.

A disadvantage is that this queueing method requires hundreds of BRAMs. Although there are a thousand or more on some high-end FPGAs, this is still a concern.

6.3. Continuous Queueing

Figure 4 shows the basic queueing used in both mappings: Some number of filters F in a filter bank feed a single force pipeline. As described in Section 3.2 the force pipelines should be as independent as possible. This is to constrain the complexity of the routing between filter and force stages and between force stage and ACC Cache.

At a high-level, this is a typical queueing problem with F servers where each has known arrival and departure rates. Also, the goal is to minimize idle time (when all queues are empty) and hardware cost. The latter includes queue size, but also complexity of the control and of the concentrator logic that routes from the filters to the force pipeline.

There are also a number of differences, however. These restrict the utility of stochastic analysis, but also point to implementation methods.

1. Execution proceeds in phases: For particle mapping, the filter bank processes F reference particles in a phase. For cell mapping, it processes 1.
2. Uniformity: The total number of arrivals per reference particle varies only slightly within a phase (for particle mapping) and among phases (for both mappings).
3. Non-uniformities. The F queues can have highly non-uniform departures and/or high variation in departures during a phase. Depending on the position of the reference particle in the home cell and on the cell of its partner, the *a priori* probability of a departure can be anything from 0 to 1.

Some design considerations are as follows. To minimize queue size, there are several mechanisms including under provisioning (by keeping F small) and throttling (when queues are full). Even if these are used, however, performance is improved by smoothing and balancing the departure rates (arrivals at the force pipelines). Here are three ways that help do this.

Fetch order. Especially for particle mapping, departure rates for each filter vary widely during a phase. For example, in Figure 3b, the departure rate for the filter of the particle shown will be near 0 when cell 4 is processed, but greater than .5 for cell 3. This variation can be smoothed by randomizing the order in which the partner particles are fetched from the cell set. A simple way to approximate this is to fetch particles from cells round-robin, rather than cell-at-a-time.

Cell mapping. For cell mapping, different cells in the cell set vary widely in the probability that their particles will be part of a neighbor list. For example, in Figure 3b, the Home cell and cell 3 are much more likely to provide partner particles than the corner cells (2 and 4). Pairing cells appropriately helps smooth the arrival rates.

Concentrator logic. Complex logic can completely smooth non-uniformities among filter queue arrivals (within a cycle) by transferring them to the queues with the most space. Logic that provides a good balance between effectiveness and hardware cost is as follows: Round-robin among queues with priority given first to those that are full and second to those that are non-empty.

Table 2. Queue size requirement and utilization are shown for various configurations with no throttling.

	particle mapped				cell mapped	
# of filters	6	7	8	9	6	9
Queue size	10	18	36	80	6	36
Utilization	69.7%	81.2%	92.5%	99.3%	69.6%	98.3%

Table 2 shows various configurations with no throttling. The maximum queue size is that required to prevent overflow with very high probability. The utilization is the average fraction of cycles that the force pipelines are busy. "Cell mapped" requires smaller queues because it has shorter phases: each filter bank processes one reference particle at a time rather than F. Even the largest queues require much less storage than the neighborlist queueing method from the previous section.

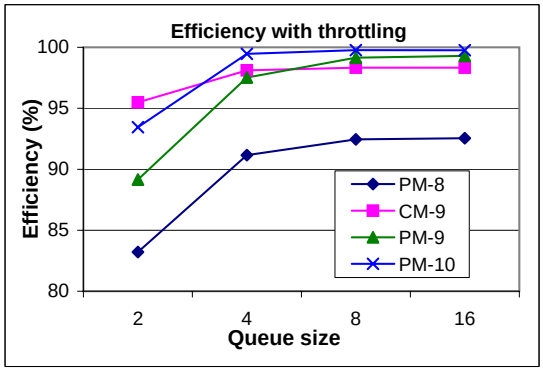

Fig. 5. Graph shows the effect of queue size on utilization for various numbers of filters (queues) and mappings of particles onto filters. PM is particle mapped, CM is cell mapped.

We now examine the effect of throttling. In this design, the filters all halt when any is in danger of overflow. Since the force pipelines consume every cycle, this happens when multiple queues are within one of full. Figure 5 shows the effect of queue size, number of filters (queues), and mapping on utilization. Even with over provisioning, utilization can be less than 100% because of non-uniformities in arrivals, and because of start-up and tear-down effects. The key result is that with slight over provisioning, i.e., 9 filters, particle mapped yields 99.2% utilization for a (very small) queue size of 8. Particle mapped is slightly better than cell mapped because of its more uniform arrivals and longer phase.

7. RESULTS

We use NAMD [7] and ProtoMol [8] as reference codes, both to determine the number of short-range particle-particle interactions computed per iteration as 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. This agrees very closely with the 33.75M computations expected from Section 3. According to a study by Stone, et al. [7], 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.

We use the force pipeline specified in a previous study [1]. It uses a mix of 32-bit integer (extended for accumulation) and single precision floating point. We find that there is little change in simulation quality over full double precision.

Our implementations are currently running in simulation. They have also been validated with serial reference codes. Results are through post place-and-route (PaR) using the standard Altera tool chain and assume the Stratix-III SE260 or SL340. PAR is sufficient to give precisely the resource usage and the number of pipelines. For operating frequency we currently achieve 200MHz; physical 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 could be realized.

Our base design uses reduced filtering, "half-moon" partitioning, particle mapping, and has 9 filters per force pipeline. For other FPGAs, planar filtering may be preferred. For queueing, the method depends on the balance between BRAMs on the one hand and logic and DSP units on the other. For the Stratix-III SL340 (more BRAMs), queueing full neighbor lists (Section 6.2) is preferred. This configuration fits 8 force pipelines. For the Stratix-III SE260 (more DSP blocks), using concentrator-based queueing (queue size = 8 with throttling) is preferred. This configuration also fits 8 force pipelines. The force pipelines run at over 95% efficiency.

This design can execute the short-range force for the ApoA1 benchmark in under 22ms. This represents an 80-fold per-core speed-up over the result shown in [7]. Since NAMD scales well, this represents a 20-fold speed-up over a full quad core implementation. While this benchmark result is a little dated, its microprocessor is comparable in process technology to the Stratix-III that we use here.

In our previous work we completely overlap the short-range force computation with host work; in [2] we found host overhead to be around 40ms on a single core. This code must now be parallelized to keep it off of the critical path. Also essential, as for all accelerators, is efficient communication between host and coprocessor. For simulations of less than a few hundred thousand particles, conventional I/O bus interfaces should be sufficient.

8. DISCUSSION AND FUTURE WORK

We have presented a study of filtering that we believe is the first for FPGAs and one of only very few for hardware implementation. The resulting designs, coupled with previous work in force pipeline design, show that FPGAs are highly competitive for MD simulation.

Specific contributions are as follows. We find that high quality filtering can be achieved with only a small amount of logic. We present a geometric filtering scheme that is preferable for some FPGA implementations. We also present a new partitioning method for optimizing with respect to Newton's 3rd Law. This is essential for the design presented here, but could also find application in other hardware implementations. And finally, the particle-mapping variation for mapping particle pairs to filter pipelines also appears to be new. We also present methods for sizing components and for integrating sections of the overall processing pipeline.

The performance results show that achieving 78-fold per core speed-ups are plausible (19-fold over a quadcore), even over optimized code on same generation multicore processors. With the Stratix-IV we can fit 16 pipelines (rather than 8) for a proportional speed-up of the short-range force.

9. REFERENCES

[1] M. Chiu, M. Herbordt, and M. Langhammer, "Performance potential of molecular dynamics simulations on high performance reconfigurable computing systems," in *Proc. HPRCTA*, 2008.

[2] Y. Gu, T. VanCourt, and M. Herbordt, "Explicit design of FPGA-based coprocessors for short-range force computation in molecular dynamics simulations," *Parallel Computing*, vol. 34, no. 4-5, pp. 261–271, 2008.

[3] R. Larson, J. Salmon, M. Deneroff, C. Young, J. Grossman, Y. Shan, J. Klepseis, and D. Shaw, "High-througput pairwise point interactions in Anton, a specialized machine for molecular dynamics simulation," in *Proc. High Performance Computer Architecture*, 2008, pp. 331–342.

[4] R. Scrofano and V. Prasanna, "Preliminary investigation of advanced electrostatics in molecular dynamics on reconfigurable computers," in *Supercomputing*, 2006.

[5] V. Kindratenko and D. Pointer, "A case study in porting a production scientific supercomputing application to a reconfigurable computer," in *Proc. FCCM*, 2006, pp. 13–22.

[6] M. Snir, "A note on N-body computations with cutoffs," *Theory of Computing Systems*, vol. 37, pp. 295–318, 2004.

[7] J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, and K. Schulten, "Accelerating molecular modeling applications with graphics processors," *JCC*, vol. 28, pp. 2618–2640, 2007.

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