FPGA-Accelerated Molecular Dynamics

M.A. Khan, M. Chiu, and M.C. Herbordt

Abstract Molecular dynamics simulation (MD) is one of the most important applications in computational science and engineering. Despite its widespread use, there exists a many order-of-magnitude gap between the demand and the performance currently achieved. Acceleration of MD has therefore received much attention. In this chapter, we discuss the progress made in accelerating MD using Field-Programmable Gate Arrays (FPGAs). We first introduce the algorithms and computational methods used in MD and describe the general issues in accelerating MD. In the core of this chapter, we show how to design an efficient force computation pipeline for the range-limited force computation, the most time-consuming part of MD and the most mature topic in FPGA acceleration of MD. We discuss computational techniques and simulation quality and present efficient filtering and mapping schemes. We also discuss overall design, host–accelerator interaction and other board-level issues. We conclude with future challenges and the potential of production FPGA-accelerated MD.

1 Introduction to Molecular Dynamics

Molecular dynamics simulations (MD) are based on the application of classical mechanics models to ensembles of particles and are used to study the behavior of physical systems at an atomic level of detail [39]. MD simulations act as virtual experiments and provide a projection of laboratory experiments with potentially greater detail. MD is one of the most widely used computational tools in biomedical research and industry and has so far provided many important insights into understanding the functionality of biological systems (see, e.g., [1, 25, 31]). MD models have been developed and refined over many years and are validated through

M.A. Khan (✉) • M. Chiu • M.C. Herbordt
Boston University, 8 Saint Mary's Street, Boston, MA 02215, USA
e-mail: azkhan@bu.edu; mattchiu@bu.edu; herbordt@bu.edu

fitting models to experimental and quantum data. Although classical MD simulation is inherently an approximation, it is dramatically faster than a direct solution to the full set of quantum mechanical equations.

But while the use of classical rather than quantum models results in orders-of-magnitude higher throughput, MD remains extremely time consuming. For example, the simulation of even a comparatively simple biological entity such as the STM virus (a million-atom system) for 100 ns would take 70 years if run on a single CPU core [14]. Fortunately MD scales well for simulations of this size or greater. The widely used MD packages, e.g., AMBER [6], CHARMM [5], Desmond [4], GROMACS [21], LAMMPS [37], NAMD [34], can take full advantage of scalability [27]. But typical MD executions still end up taking month-long runtime, even on supercomputers [45].

To make matters worse, many interesting biological phenomena occur only on far longer timescales. For example, protein folding, the process by which a linear chain of amino acids folds into a three-dimensional functional protein, is estimated to take at least a microsecond [12]. The exact mechanism of such phenomena remains beyond the reach of the current computational capabilities [44]. Longer simulations are also critical to facilitate comparison with physically observable processes, which (again) tend to be at least in the microsecond range. With stagnant CPU clock frequency and no remarkable breakthrough in the underlying algorithms for a decade, MD faces great challenges to meet the ever-increasing demand for larger and longer simulations.

Hardware acceleration of MD has therefore received much attention. ASIC-based systems such as Anton [43] and MD-Grape [32] have shown remarkable results, but their non-recurring cost remains high. GPU-based systems with their low cost and ease of use also show great potential. But GPUs are power hungry and, perhaps more significantly, are vulnerable to data communication bottlenecks [16, 48].

FPGAs, on the other hand, have a flexible architecture and are energy efficient. They bridge the programmability of CPUs and the custom design of ASICs. Although developing an FPGA-based design takes significantly longer than a GPU-based system, because it requires both software and hardware development, the effort should be cost-effective due to the relatively long life-cycle of MD packages. Moreover, improvements in fabrication process generally translate to performance increases for FPGA-based systems (mostly in the form of direct replication of additional computation units). And perhaps most significantly for emerging systems, FPGAs are fundamentally communication switches and so can avoid communication bottlenecks and form the basis of accelerator-centric high-performance computing systems.

This chapter discusses the current state of FPGA acceleration of MD-based primarily on the work done at Boston University [8, 9, 18]. The remainder of this section gives an extended introduction to MD. This is necessary because while MD is nearly trivial to define, there are a number of subtle issues which have a great impact on acceleration method. In the next section we present the issues universal to MD acceleration. After that we describe in depth the state-of-the-art in FPGA

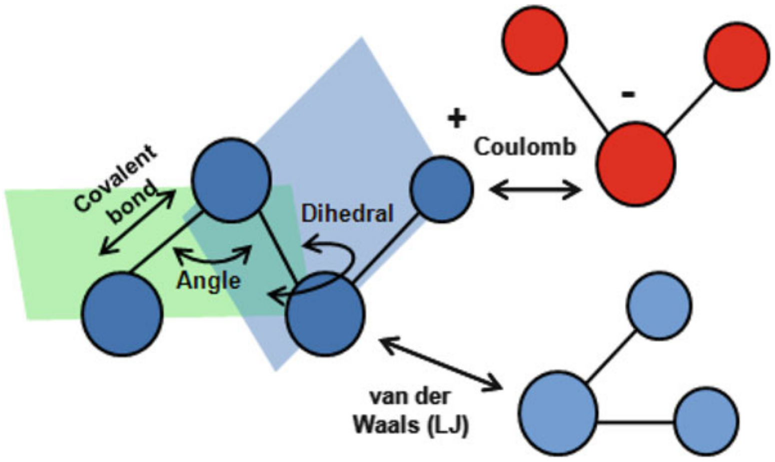

Fig. 1 MD Forces computed by MD include several bonded (covalent, angle, and dihedral) and nonbonded (van der Waals and Coulomb)

MD acceleration focusing on the range-limited force. Finally, we summarize future challenges and potential especially in the creation of parallel FPGA-based MD systems.

1.1 Overview of Molecular Dynamics Simulation

MD is an iterative process that models dynamics of molecules by applying classical mechanics [39]. The user provides the initial state (position, velocity, etc.), the force model, other properties of the physical system, and some simulation parameters such as simulation type and output frequency. Simulation advances by timestep where each timestep has two phases: force computation and motion update. The duration of the timesteps is determined by the vibration of particles and typically corresponds to one or a few femtoseconds (fs) of real time. In typical CPU implementations, executing a single timestep of a modest 100 K particle simulation (a protein in water) takes over a second on a single core. This means that the 10^6–10^9 timesteps needed to simulate reasonable timescales result in long runtimes.

There are many publicly available and widely used MD packages including NAMD [34], LAMMPS [37], AMBER [6], GROMACS [21], and Desmond [4]. They support various force fields (e.g., AMBER [38] and CHARMM [30]) and simulation types. But regardless of the specific package or force field model, force computation in MD involves computing contributions of van der Waals, electrostatic (Coulomb), and various bonded terms (see Fig. 1 and (1)).

$$F_{\text{total}} = F_{\text{bond}} + F_{\text{angle}} + F_{\text{dihedral}} + F_{\text{hydrogen}} + F_{\text{vanderWaals}} + F_{\text{electrostatic}}. \tag{1}$$

van der Waals and electrostatic forces are *non-bonded* forces, the others *bonded*. Non-bonded forces can be further divided into two types: the *range-limited* force that consists of the van der Waals and the short-range part of the electrostatic force and the *long-range* force that consists of the long-range part of the electrostatic force.

Since bonded forces affect only a few neighboring atoms, they can be computed in $O(N)$ time, where N is the total number of particles in the system. Non-bonded terms in the naive implementation have complexity of $O(N^2)$, but several algorithms and techniques exist to reduce their complexity; these will be described in later subsections. In practice, the complexity of the range-limited force computation is reduced to $O(N)$ and that of the long-range force computation to $N\log(N)$. Motion update and other simulation management tasks are also $O(N)$. In a typical MD run on a single CPU core, most of the time is spent computing non-bonded forces. For parallel MD, inter-node data communication becomes an increasingly dominant factor as the number of computing nodes increases, especially for small to medium sized physical systems. Sample timing profiles for both serial and parallel runs of MD are presented in Sect. 2.

The Van der Waals (VdW) force is approximated by the Lenard-Jones (LJ) potential as shown in (2):

$$\overrightarrow{F}_i(LJ) = \sum_{i \neq j} \frac{\varepsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \overrightarrow{r_{ij}}, \tag{2}$$

where ε_{ab} and σ_{ab} are parameters related to particle types and r_{ij} is the relative distance between particle i and particle j.

A complete evaluation of VdW or LJ force requires evaluation of interactions between all particle pairs in the system. The computational complexity is therefore $O(N^2)$, where N is the number of particles in the system. A common way to reduce this complexity is applying a cutoff. Since the LJ force vanishes quickly with the separation of a particle pair it is usually ignored when two particles are separated beyond 8–16$\mathring{A}$. To ensure a smooth transition at cutoff, an additional switching function is often used. Using a cutoff distance alone does not reduce the complexity of the LJ force computation because all particle pairs must still be checked to see if they are within the cutoff distance. The complexity is reduced to $O(N)$ by combining this with techniques like the cell-list and neighbor-list methods, which will be described in Sect. 1.2.

The electrostatic or Coulomb force works between two charged particles and is given by (3):

$$\overrightarrow{F}_i(CL) = q_i \sum_{i \neq j} \left(\frac{q_j}{|r_{ij}|^3} \right) \overrightarrow{r_{ij}}, \tag{3}$$

where q_i and q_j are the particle charges and r_{ij} is the separation distance between particles i and j.

Unlike the van der Waals force, the Coulomb force does not fall off sufficiently quickly to immediately allow the general application of a cutoff. The Coulomb force

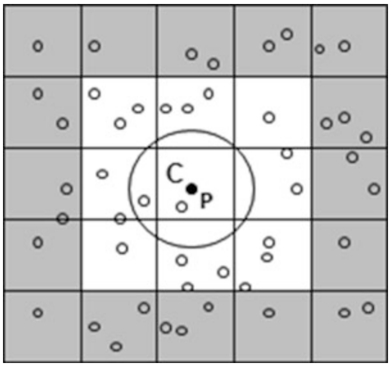

Fig. 2 2D Illustration of cell and neighbor lists. In the range-limited force, particles only interact with those in the cell neighborhood. Neighbor lists are constructed by including for each particle only those particles within the cutoff radius C (shown for P)

is therefore often split into two components: a range-limited part that goes to zero in the neighborhood of the LJ cutoff and a long-range part that can be computed using efficient electrostatic methods, the most popular being based on Ewald Sums [11] or Multigrid [46]. For example, one can split the original Coulomb force curve into two parts (with a smoothing function $g_a(r)$):

$$\frac{1}{r} = \left(\frac{1}{r} - g_a(r)\right) + g_a(r). \tag{4}$$

The short-range component can be computed together with the Lennard–Jones force using particle indexed lookup tables A_{ab}, B_{ab}, and QQ_{ab}. Then the entire short-range force to be computed is:

$$\frac{\mathbf{F}_{ji}^{\text{short}}}{\mathbf{r_{ji}}} = A_{ab} r_{ji}^{-14} + B_{ab} r_{ji}^{-8} + QQ_{ab}\left(r_{ji}^{-3} + \frac{g_a'(r)}{r}\right). \tag{5}$$

In addition to the non-bonded forces, bonded interactions (e.g., bond, angle, and dihedral in Fig. 1) must also be computed every timestep. They have $O(N)$ complexity and take a relatively small part of the total time. Bonded pairs are generally excluded from non-bonded force computation, but if for any reason (e.g., to avoid a branch instruction in an inner loop) a non-bonded force computation includes bonded pairs, then those forces must be subtracted accordingly. Because the long-range force varies less quickly than the other force components, it is often computed only every 2–4 timesteps.

1.2 Cell Lists and Neighbor Lists

We now present two methods of reducing the naive complexity of $O(N^2)$ to $O(N)$. In the cell-list method [22, 40] a simulation box is first partitioned into several cells, often cubic in shape (see Fig. 2 for a 2D depiction). Each dimension is typically chosen to be slightly larger than the cutoff distance. This means, for a 3D system, that traversing through the particles of the home cell and 26 adjacent cells suffices, independent of the overall simulation size. If Newton's third law is used, then only half of the neighboring cells need to be checked. If the cell dimension is less than cutoff distance, then more number of cells need to be checked. The cost of constructing cell lists scales linearly with the number of particles but reduces the complexity of the force evaluation to $O(N)$.

Using cell lists still results in checking many more particles than necessary. For a particle in the center of a home cell, we only need to check its surrounding volume of $(4/3) * 3.14 * R_c^3$, where R_c is the cutoff radius. But in the cell-list method we end up checking a volume of $27 * R_c^3$, which is roughly 6 times larger than needed. This can be improved using neighbor lists [49]. In this method, a list of possible neighboring particles is maintained for each particle and only this list is checked for force evaluation. A particle is included in the neighbor list of another particle if the distance between them is less than $R_c + R_m$, where R_m is a small buffer margin. R_m is chosen such that the neighbor-list also contains the particles which are not yet within the cutoff range but might enter the cutoff range before the list is updated next. In every timestep, the validity of each pair in a neighbor list is checked before it is actually used in force evaluation. Neighbor lists are usually updated periodically in a fixed number of timesteps or when displacements of particles exceed a predetermined value.

Although neighbor lists can be constructed for all particles in $O(N)$ time (using cell-lists), it is far more costly as many particles must still be checked for each reference particle. But as long as the neighbor lists are not updated too frequently, which is the case generally, this method reduces the range-limited force evaluation time significantly. The savings in runtime comes at the cost of extra storage required to save the neighbor-list of each particle. For most high-end CPUs, this is not a major issue.

1.3 Direct Computation vs. Table Interpolation

The most time-consuming part of an MD simulation is typically the evaluation of range-limited forces. One of the major optimizations is the use of table lookup in place of direct computation. This avoids expensive square roots and $erfc$ evaluations. This method not only saves computation time but is also robust in incorporating small changes such as the incorporation of a switching function.

Typically the square of the inter-particle distance (r^2) is used as the index. The possible range of r^2 is divided into several sections or segments and each section is further divided into intervals or bins as shown in Fig. 3. For an M order interpolation, each interval needs $M + 1$ coefficients and each section needs $N * (M + 1)$ coefficients, where N is the number of bins in the section. Accuracy increases with both the number of intervals per section and the interpolation order. Generally the rapidly changing regions are assigned relatively higher number of bins, and relatively stable regions are assigned fewer bins. Equation (6) shows a third order interpolation.

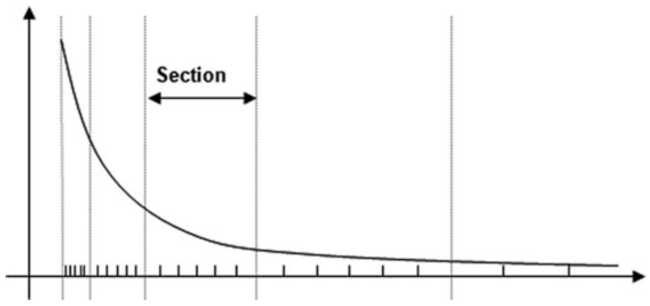

Fig. 3 In MD interpolation, function values are typically computed by *Section* with each having a constant number of bins, but varying in size with distance

$$F(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \tag{6}$$

For reference, here we present a sample of table interpolation parameters used in widely known MD packages and systems.

- NAMD (CPU)—[34] and Source code of NAMD2.7
  Order = 2   bins/segment = 64   Index: r^2
  Segments: 12—segment size increases exponentially, starting from 0.0625
- NAMD (GPU)—[48] and Source code of NAMD2.7
  Order = 0   bins/segment = 64   Index: $1/\sqrt{r^2}$
  Segments: 12—segment size increases exponentially
- CHARMM—[5]
  Order = 2   bins/segment = 10–25   Index: r^2
  Segments: Uniform segment size of $1\AA^2$ is used which results in relatively more precise values near cut-off
- ANTON—[28]
  Force Table Order = Says 3 but that may be for energy only. Value for force may be smaller.
  # of bins = 256   Index: r^2
  Segments: Segments are of different widths, but values not available, nor whether the number of bins is the total or per segment.
- GROMACS—[21] and GROMACS Manual 4.5.3, page 148
  Order = 2   bins = 500 (2000) per nm for single (double) precision
  Segments: 1   Index: r^2
  Comment: Allows user-defined tables.

Clearly there are a wide variety of parameter settings. These have been chosen with regard to cache size (CPU), routing and chip area (Anton), and the availability of special features (GPU texture memory). These parameters also have an effect on simulation quality, which we discuss next.

1.4 Simulation Quality: Numeric Precision and Validation

Although most widely used MD packages use double-precision floating point (DP) for force evaluation, studies have shown that it is possible to achieve acceptable quality of simulation using single-precision floating point (SP) or even using fixed point arithmetic, as long as the exact atomic trajectory is not the main concern [36, 41, 43]. Since floating point (FP) arithmetic requires more area and has longer latency, a hardware implementation would always prefer fixed point arithmetic. Care must be taken, however, to ensure that the quality of the simulation remains acceptable. Therefore a critical issue in all MD implementations is the trade-off between precision and simulation quality.

Quality measures can be classified as follows (see, e.g., [13, 33, 43]).

1. Arithmetic error here is the deviation from the ideal (direct) computation done at high precision (e.g., double-precision). A frequently used measure is the relative RMS force error, which is defined as follows [42]:

$$\Delta F = \sqrt{\left(\frac{\sum_i \sum_{\alpha\in x,y,z}[F_{i,\alpha} - F^*_{i,\alpha}]^2}{\sum_i \sum_{\alpha\in x,y,z}[F^*_{i,\alpha}]^2}\right)}. \tag{7}$$

2. Physical invariants should remain so in simulation. Energy can be monitored through fluctuation (e.g., in the relative RMS value) and drift. Total fluctuation of energy can be determined using the following expression (suggested by Shan et al. [42]):

$$\Delta E = \frac{1}{N_t}\sum_{i=1}^{N_t}\left|\frac{E_0 - E_i}{E_0}\right|, \tag{8}$$

where E_0 is the initial value, N_i is the total number of time steps in time t, and E_i is the total energy at step i. Acceptable numerical accuracy is achieved when $\Delta E \leq 0.003$.

2 Basic Issues with Acceleration and Parallelization

2.1 Profile

The maximum speed-up achievable by any hardware accelerator is limited by Amdahl's law. It is therefore important to profile the software to identify potential targets of acceleration. As discussed in Sect. 1.1, a timestep in MD consists of two parts, force computation and motion integration. The major tasks in force computation are computing range-limited forces, computing long-range forces, and computing bonded forces. Table 1 shows the timing profile of a timestep using the GROMACS MD package on a single CPU core [21]. These results are typical;

Table 1 Timing profile of an MD run from a GROMACS study [21]

Step	Task	% execution time
Force computation	Range-limited force	60
	FFT, Fourier-space computation, IFFT	17
	Charge spreading and force interpolation	13
	Other forces	5
Motion integration	Position and velocity updates	2
	Others	3

see, e.g., [43]. As we can see the range-limited force computation dominates and consumes 60% of the total runtime. The next major task is the long-range force computation, which can be further divided into two tasks, charge-spreading/force-interpolation and FFT-based computation. FFT, Fourier-space computation, and inverse FFT take 17% of the total runtime while charge spreading and force interpolation take 13% of the total runtime. Computing other forces takes only 5% of the total runtime. Unlike the force computation, motion integration is a straightforward process and takes only 2% of the total runtime. Other remaining computations take 3% of the total runtime. In addition to serial runtime, data communication becomes a limiting factor in parallel and accelerated version. We discuss this in Sect. 2.3.

2.2 Handling Exclusion

While combining various forces before computing acceleration is a straightforward process of linear summation, careful consideration is required for bonded pairs, especially when using hardware accelerators. In particular, covalently bonded pairs need to be excluded from non-bonded force computation. One way to ensure this is to check whether two particles are bonded before evaluating their non-bonded forces. This is expensive because it requires on-the-fly check for bonds. Another way is to use separate neighbor lists for bonded and non-bonded neighbors. Both of these methods are problematic for hardware acceleration: one requires implementing a branch instruction while the other forces the use of neighbor-lists, which may be impractical for hardware implementation (see Sect. 3.2).

A way that is often the preferred for accelerators is to compute non-bonded forces for all particle-pairs within the cutoff distance, but later subtract those for bonded pairs in a separate stage. This method does not need either on-the-fly bond checking or neighbor-lists. There is a different problem here though. The r^{14} term of the LJ force (2) can be very large for bonded particles because they tend to be much closer than non-bonded pairs. Adding and subtracting such large scale values can overwhelm real but small force values. Therefore, care needs to be taken so that the actual force values are not saturated. For example, an inner as well as an outer cutoff can be applied.

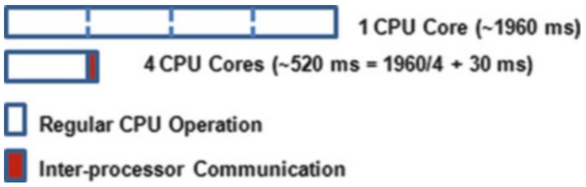

Fig. 4 Apoa1 benchmark runtime/timestep using NAMD showing overhead in a small-scale parallel simulation

2.3 Data Transfer and Communication Overhead

Accelerators are typically connected to the host CPU via some shared interface, e.g., the PCI or PCIe bus. For efficient computation on the accelerator, frequent data transfers between the main memory of the CPU and accelerator must be avoided. Input data need to be transferred to the accelerator before the computation starts and results need to be sent back to the host CPU. Although this is usually done using DMA, it may still consume a significant amount of time that was not required in a CPU-only version. It is preferred that the CPU remains engaged in other useful tasks while data transfer and accelerated computation take place, allowing efficient overlap of computation and communication, as well as parallel utilization of the CPU and the accelerator. Our studies show that host-accelerator data transfer takes around 5–10% of the accelerated runtime for MD (see Sect. 3.2).

In addition to intra-node (host-accelerator) data transfer, inter-node data communication may become a bottleneck, especially for accelerated versions of MD. MD is a highly parallel application and typically runs on multiple compute nodes. Parallelism is achieved in MD by first decomposing the simulation space spatially and assigning one or more of such decomposed sections to a compute node (see, e.g., [34]). Particles in different sections then need to compute their pairwise interaction forces (both non-bonded and bonded) which requires inter-node data communication between node-pairs. In addition to that, long-range force computation requires all-to-all communication [50]. Thus, in addition to the serial runtime, inter-node communication plays an important role in parallel MD. Figure 4 shows an example of inter-processor communication time as the number of processors increases from 1 to 4. We performed this experiment using Apoa1 benchmark and NAMD2.8 [34] on a quad-core Intel CPU (2 core2-duo) of 2.0 GHz each. For a CPU-only version the proportion may be acceptable. For accelerated versions, however, the proportion increases and becomes a major problem [35].

2.4 Newton's 3rd Law

Newton's 3rd law (N3L) allows computing forces between a pair of particles only once and uses the result to update both particles. This provides opportunities for certain optimizations. For example, when using the cell-list method, each cell now only needs to check half of its neighboring cells. Some ordering needs to be established to make sure that all required cell-pairs are considered, but this is a trivial problem.

The issue of whether to use N3L or not becomes more interesting in parallel and accelerated version of MD. It plays an important role in the amount and pattern of inter-node data communication for parallel runs, and successive accumulation of forces in multi-pipelined hardware accelerators (see discussion on accumulation in Sect. 3.1). For example, assume a parallel version of MD where particles x and y are assigned to compute nodes X and Y, respectively. If N3L is not used, we need to send particle data of y from Y to X and particle data of x from X to Y before the force computation of a timestep can take place. But no further inter-node communication will be required for that timestep as particle data will be updated locally. In contrast, if N3L is used, particle data of y need to be sent from Y to X before the computation and results need to be sent from X to Y. Depending on the available computation and communication capability, these two may result in different efficiency. Similar, but more fine-grained, issues exist for hardware accelerators too.

2.5 Partitioning and Routing

Parallel MD requires partitioning of the problem and routing data every timestep. Although there are various ways of partitioning computations in MD, practically all widely used MD packages use some variation of spatial decomposition (e.g., recursive bisection, neutral territory, half shell, or midpoint [4, 23]). In such a method, each compute node or process is responsible for updating particles in a certain region of the simulation space. In other words, it owns the particles in that region. Particle data such as position and charge need to be routed to the node that will compute forces for that particle. Depending on the partitioning scheme, computation may take place on a node that owns at least one of the particles involved in the force computation, or it may take place on a node that does not own any of the particles involved in the force computation. Computation results may also need to be routed back to the owner node. This also depends on several choices such as the partitioning scheme and the use of N3L.

For an accelerated version of MD, partitioning and routing may cause additional overhead. Because hardware accelerators typically require a chunk of data to work on at a time in order to avoid frequent data communication with the host CPU. This means fine-grained overlapping of computation and communication, which is possible in a CPU-only version, becomes challenging.

3 FPGA Acceleration Methods

Several papers have been published from CAAD Lab at Boston University describing a highly efficient FPGA kernel for the range-limited force computation [7–9]. The kernel was integrated into NAMD-lite [19], a serial MD package developed at UIUC to provide a simpler way to examine and validate new features before integrating them into NAMD [34]. The FPGA kernel itself was implemented on an Altera Stratix-III SE260 FPGA of Gidel ProcStar-III board [15]. The board contains four such FPGAs and is capable of running at a system speed of up to 300 MHz. The FPGAs communicate with the host CPU via a PCIe bus interface. Each FPGA is individually equipped with over 4GB of memory.

The runtime of the kernel was 26× faster over the end-to-end runtime of NAMD, for Apoa1, a benchmark consisting of 92,224 atoms [10]. The electrostatic force was computed every cycle using PME and both LJ and short-rage portion of electrostatic force were computed on the FPGAs. Particle data along with cell-lists and particle types are sent to the FPGA every timestep, while force data are received from the FPGA and then integrated on the host. A direct end-to-end comparison with the software-only version was not done since the host software itself (NAMD-lite) is not optimized for performance. In the next three subsections we discuss the key contributions of this work in depth. In the following two subsections we describe some preliminary work in the FPGA-acceleration of the long-range force and in mapping MD to multi-FPGA systems.

3.1 Force Pipeline

In Sect. 1.1 we described the general methods in computing the range-limited forces (see (5)). Here we present their actual implementation emphasizing compatibility with NAMD.

While the van der Waals term shown in (2) converges quickly, it must still be modified for effective MD simulations. In particular, a switching function is implemented to truncate van der Waals force smoothly at the cutoff distance (see (9)–(11)).

$$s = (cutoff^2 - r^2)^2 * (cutoff^2 + 2 * r^2 - 3 * switch\_dist^2) * denom \tag{9}$$

$$ds_r = 12 * (cutoff^2) * (switch\_dist^2 - r^2) * denom \tag{10}$$

$$denom = 1/(cutoff^2 - switch\_dist^2)^3. \tag{11}$$

Without a switching/smoothing function, the energy may not be conserved as the force would be truncated abruptly at the cutoff distance. The graph of van der Waals potential with the switching/smoothing function is illustrated in Fig. 5. The van der Waals force and energy can be computed directly as shown here:

IF ($r^2 \leq switch\_dist^2$) $U_{vdW} = U, F_{vdW} = F$
IF ($r^2 > switch\_dist^2$ && $r^2 < cutoff^2$) $U_{vdW} * s, F_{vdW} = F * s + U_{vdw} * ds_r$
IF ($r^2 \geq cutoff^2$) $U_{vdW} = 0, F_{vdW} = 0.$

For the Coulomb term the most flexible method used by NAMD for calculating the electrostatic force/energy is Particle Mesh Ewald (PME). The following is the pairwise component:

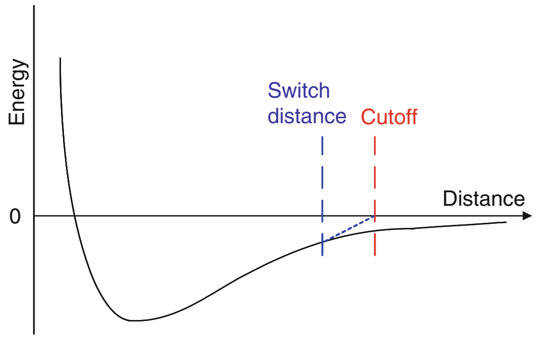

Fig. 5 Graph shows the van der Waals potential with switching/smoothing function (*dashed line*)

$$Es = \frac{1}{4\pi\varepsilon_0}\frac{1}{2}\sum_n\sum_{i=1}^{N}\sum_{i=0}^{n}\frac{q_iq_j}{|r_i - r_j + nL|}erfc\left(\frac{|r_i - r_j + nL|}{\sqrt{2}\sigma}\right). \tag{12}$$

To avoid computing these complex equations explicitly, software often employs table lookup with interpolation (Sect. 1.3). Equation (5) can be rewritten as follows:

$$\frac{\mathbf{F}_{ji}^{\text{short}}(|r_{ji}|^2(a,b))}{\mathbf{r_{ji}}} = A_{ab}R_{14}(|r_{ji}|^2) + B_{ab}R_8(|r_{ji}|^2) + QQ_{ab}R_3(|r_{ji}|^2), \tag{13}$$

where R_{14}, R_8, and R_3 are three tables indexed with $|r_{ji}|^2$ (rather than $|r_{ji}|$, to avoid the square root operation).

Designing a force computation pipeline on FPGA to accurately perform these tasks requires careful consideration of several issues. Figure 6 illustrates the major functional units of the force pipelines. The force function evaluators are the diamonds marked in red; these are the components which can be implemented with the various schemes. The other units remain mostly unchanged. The three function evaluators are for the R_{14}, R_8, and R_3 components of (13), respectively. In particular, *Vdw Function 1* and *Vdw Function 2* are the R_{14} and R_8 terms but also include the cutoff shown in (9)–(11). *Coulomb Function* is the R_3 term but also includes the correction shown in (12).

For the actual implementation we use a combination of fixed and floating point. Floating point has far more dynamic range, while fixed point is much more efficient and offers somewhat higher precision. Fixed point is especially advantageous for use as an index (r^2) and for accumulation. We therefore perform the following conversions: float to fixed as data arrive on the FPGA; to float for interpolation; to fixed for accumulation; and to float for transfer back to the host.

A significant issue is determining the minimum interpolation order, precision, and number of intervals without sacrificing simulation quality. For this we use two methods both of which use a modified NAMD-lite to generate the appropriate data. The first method uses (7) to compute the relative RMS error with respect to the reference code. The simulation was first run for 1,000 timesteps using direct computation. Then in the next timestep both direct computation and table lookup

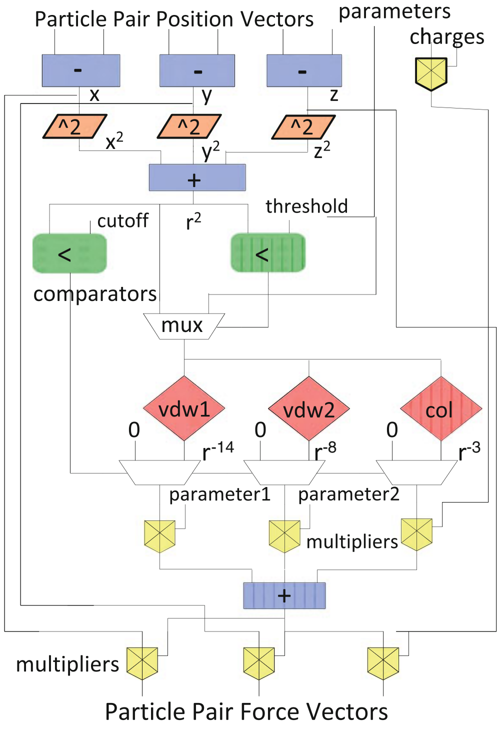

Fig. 6 Logic for computing the range-limited force. *Red diamonds* indicate respective table lookups for the two van der Waals force components and the Coulombic force

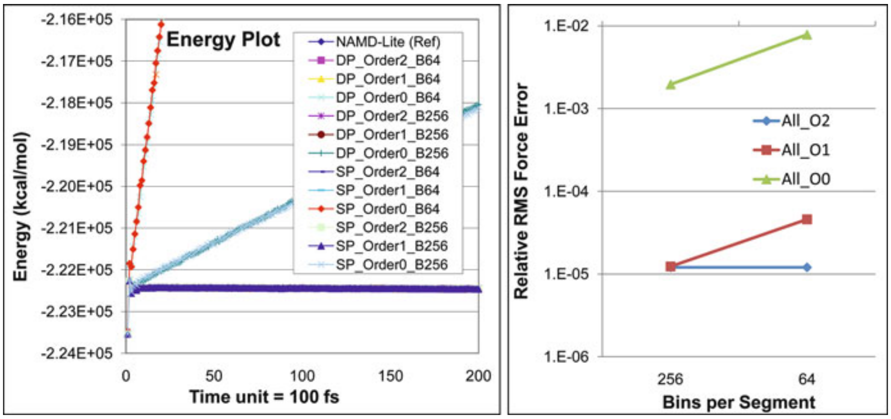

Fig. 7 *Right graph* shows relative RMS force error versus bin density for interpolation orders 0, 1, and 2. *Left graph* shows energy for various designs run for 20,000 timesteps. Except for 0-order, plots are indistinguishable from the reference code

with interpolation were used to find the relative RMS force error for the various lookup parameters. Only the range-limited forces (switched VdW and short-range part of PME) were considered. All computations were done in double-precision. Results are shown in the right half of Fig. 7. We note that 1st and 2nd order

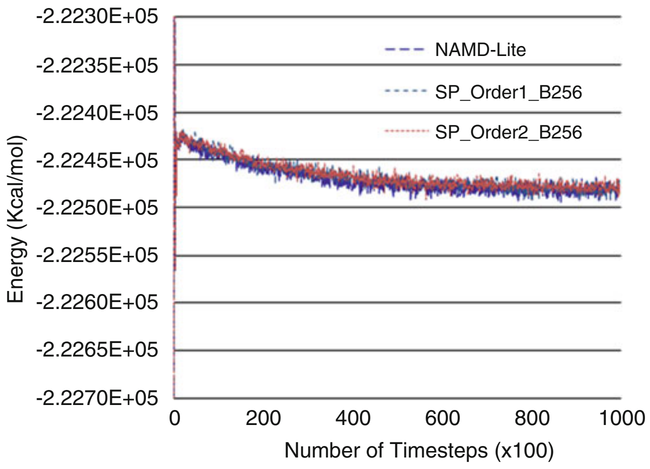

Fig. 8 Reference code and two designs run for 100,000 timesteps

interpolation have two orders of magnitude less error than 0th order. We also note that with 256 bins per section (and 12 sections), 1st and 2nd order are virtually identical.

The second method was to measure energy fluctuation and drift. Results are presented for the NAMD benchmark ApoA1. It has 92,224 particles, a bounding box of $108Å \times 108Å \times 78Å$, and a cutoff radius of $12Å$. The Coulomb force is evaluated with PME. A switching function is applied to smooth the LJ force when the intra-distance of particle pairs is between 10 and $12Å$. Preliminary results are shown in the left side of Fig. 7. A number of design alternatives were examined, including the original code and all combinations of the following parameters: bin density (64 and 256 per section or segment), interpolation order (0th, 1st, and 2nd), and single and double-precision floating point. We note that all of the 0th order simulations are unacceptable, but that the others are all indistinguishable (in both energy fluctuation and drift) from the serial reference code running direct computation in double-precision floating point.

To validate the most promising candidate designs, longer runs were conducted. An energy plot for 100,000 timesteps is provided in Fig. 8. The graphs depict the original reference code and two FPGA designs. Both are single precision with 256 bins per interval; one is first order and the other second order. Good energy conservation is seen in the FPGA-accelerated versions. Only a small divergence of 0.02% was observed compared to the software-only version. The ΔE values, using (8), for all accelerated versions were found to be much smaller than 0.003.

One of the interesting contributions of this work was with respect to the utilization of Block RAM (BRAM) architecture of the FPGAs for interpolation. MD packages typically choose the interval such that the table is small enough to fit in L1 cache. This is compensated by the use of higher order of interpolation, second order being a common choice for force computation [9]. FPGAs, however, can afford having finer intervals because of the availability of on-chip BRAMs. It was found that, by doubling the number of bins per section, first order interpolation can achieve similar simulation quality as the second order interpolation (see Fig. 7). This saves logic and multipliers and increases the number of force pipelines that can fit in a single FPGA.

3.2 Filtering and Mapping Scheme

The performance of an FPGA kernel is directly dependent on the efficiency of the force computation pipelines. The more useful work pipelines do every cycle, the better the performance is. This in turn requires that the force pipelines be fed, as much as possible, with particle pairs that are within cutoff distance. Section 1.2 described two efficient methods for finding particle-pairs within cutoff distance. But for MD accelerators, this requires additional considerations. The cell list computation is very fast and the data generated are small, so it is generally done on the host. The results are downloaded to the FPGA every iteration. The neighbor-list method, on the other hand, is problematic if the lists are computed on the host. The size of the aggregate neighbor-lists is hundreds of times that of the cell lists, which makes their transfer to FPGA impractical. As a consequence, neighbor-list computation, if it is done at all, must be done on the FPGA.

This work first looks 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 $108\AA \times 108\AA \times 78\AA$, and a cutoff radius of $12\AA$. This roughly 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. Using Newton's third law (N3L), home cell particles are only matched with particles of part of the cell neighborhood, and with, on average, half of the particles in the home cell. For the 14- and 18-cell configurations (see later discussion on mapping scheme), the number of particles to be examined is 2,450 and 3,150, respectively. Given current FPGA technology, any of these cell neighborhoods (14, 18, or even 27) easily fits in the on-chip BRAMs.

On the other hand, neighbor-lists for a home cell do not fit on the FPGA. The aggregate neighbor-lists for 175 home cell particles is over 64,000 particles (one half of 732 per particle—732 rather than 4,725 because of increased efficiency).

The memory requirements are therefore very different. Cell-lists can be swapped back and forth between the FPGA and the DDR memory, 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 as they are needed. This has worked when there are one or two force pipelines operating at 100 MHz [26, 41], but is problematic for current and future high-end FPGAs. For example, the Stratix-III/Virtex-5 generation of FPGAs can support up to 8 force pipelines operating at 200 MHz leading to a bandwidth requirement of over 20 GB/s.

The solution proposed in this work is to use neighbor-lists, but to compute them every iteration, generating them continuously and consuming them almost immediately. There are three major issues that are addressed in this work, which we discuss next.

1. How should the filter be computed?
2. What cell neighborhood organization best takes advantage of N3L?
3. How should particle pairs be mapped to filter pipelines?

3.2.1 Filter Pipeline Design and Optimization

For a cell-list-based system where one home cell is processed 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. *Filtering* here means 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 ratio of undesirable particle pairs that were removed to the original number of undesirable particle pairs. The *extra work* due to imperfection is the ratio of undesirable pairs not removed to the desirable pairs.

Three methods are evaluated, two existing and one new, which trade off filter efficiency for hardware resources. As described in Sect. 3.1, particle positions are stored in three Cartesian dimensions, each in 32-bit integer. Filter designs have two parameters, precision and geometry.

1. *Full precision:* Precision = full, Geometry = sphere
   This filter computes $r^2 = x^2 + y^2 + z^2$ and compares whether $r^2 < r_c^2$ using full 32-bit precision. Filtering efficiency 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 [28], 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 cutoff radius must be increased (rounded up to the next bit) so filtering efficiency goes down: for 8 bits of precision, it is 99.5 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 6–10 filter pipelines per force pipeline. In this method we avoid multiplication by thresholding with planes rather than a sphere (see Fig. 9 for the 2D analog). The formulas are as follows:

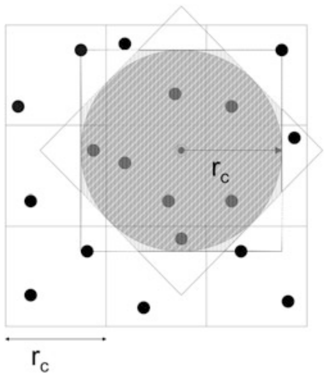

Fig. 9 Filtering with planes rather than a sphere—2D analogue

Table 2 Comparison of three filtering schemes with respect to quality and resource usage

Filtering Method	LUTs/registers		Multipliers		Filter eff.	Extra work
Full precision	341/881	0.43%	12	1.6%	100%	0%
Full prec.—logic only muls	2577/2696	1.3%	0	0.0%	100%	0%
Reduced precision	131/266	0.13%	3	0.4%	99.5%	3%
Reduced prec.—logic only muls	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

A force pipeline is shown for reference. Percent utilization is with respect to the Altera Stratix-III EP3SE260

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

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

Table 2 summarizes the cost (LUTs, registers, and multipliers) 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 Sect. 3.1) is shown for scale.

The most important result is the relative cost of the filters to the force pipeline. Depending on implementation and load balancing method (see later discussion on mapping scheme), each force pipeline needs between 6 and 9 filters to keep it running at full utilization. We refer to that set of filters as a *filter bank*. Table 2 shows that a *full precision* filter bank takes from 80 to 170% of the resources of its force pipeline. The *reduced (logic only)* and *planar* filter banks, however, require only a 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 a minor effect on the number of force pipelines that can fit on the FPGA.

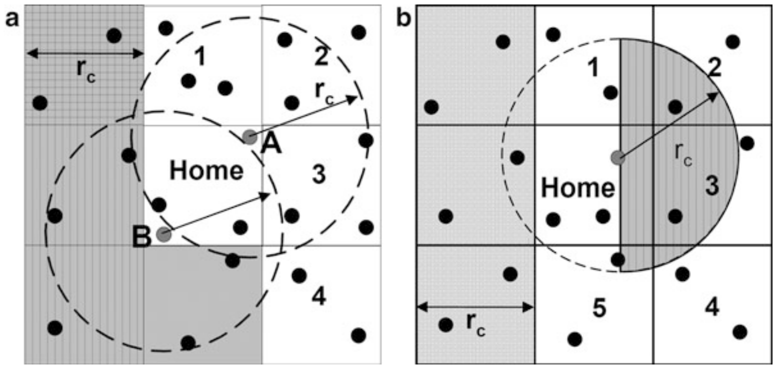

Fig. 10 Shown are two partitioning schemes for using Newton's 3rd law. In (**a**), 1–4 plus home are examined with a full sphere. In (**b**), 1–5 plus home are examined, but with a hemisphere (*shaded part of circle*)

We now compare the *reduced* and *planar* filters. The Extra Work column in Table 2 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.

3.2.2 Cell Neighborhood Organization

For efficient access of particle memory and control, and for smooth interaction between filter and force pipelines, it is preferred to have each force pipeline handle the interactions of a single reference particle (and its partner particles) 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 neighbor-lists being created at any one time (by the filter banks) to be transferred to the force pipelines simultaneously. It follows that each reference particle should have a similar number of partner particles (neighbor-list size).

The problem addressed here 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 Fig. 10a, 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 Fig. 10a, particle *B* has a much smaller neighbor-list than *A*, especially if *B* has a low ID and *A* a high.

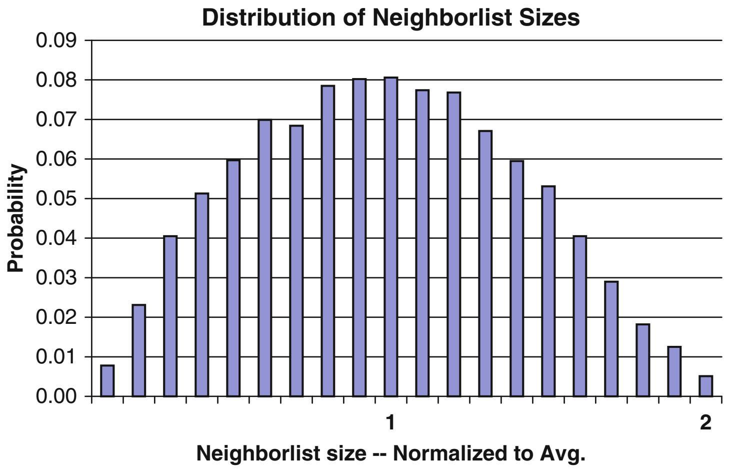

Fig. 11 Distribution of neighbor-list sizes for standard partition as derived from Monte Carlo simulations

In fact neighbor-list sizes vary from 0 to $2L$, where L is the average neighbor-list 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 neighbor-list of size $2L$, then the latency will be double that if all neighbor-lists were size L. This distribution has high variance (see Fig. 11), 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 [2, 47].

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

An alternative is to change the partitioning scheme. Our new N3L partition is shown in Fig. 10b. 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 Fig. 10b). 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 the same size, the average L. The effect of the

variation in neighbor-list size is in the number of *excess* cycles—before a new cohort of reference particles can begin processing—over the number of cycles if each neighbor-list 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. It is found 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 cell set: 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.

3.2.3 Mapping Particle Pairs to Filter Pipelines

From the previous sections an efficient design for filtering and mapping particles follows.

- During execution, the input working set (data held on the FPGA) consists of the positions of particles in a single home cell and in its 17 neighbors;
- Particles in each cell are mapped to a set of BRAMs, currently one or two per dimension, depending on the cell size;
- The N3L algorithm specifies 8 filter pipelines per force pipeline; and
- FPGA resources indicate around 6–8 force pipelines.

The problem we address in this subsection is the mapping of particle pairs to filter pipelines. There are a (perhaps surprisingly) 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. For example, one mapping maps one reference particle at a time to a bank of filter pipelines and relates each cell with one filter pipeline. The advantage of this method is that the outputs of this (8-way) filter bank can then be routed directly to a force pipeline. This mapping, however, leads to a number of load balancing, queuing, and combining problems.

A preferred mapping is shown in Fig. 12. The key idea is to associate each filter pipeline with a single reference particle (at a time) rather than a cell. Details are as follows. By "particle" we mean "position data of that particle during this iteration."

- 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 neighborhood is broadcast to all of the filter.
- Each filters output goes to a single set of BRAMs.

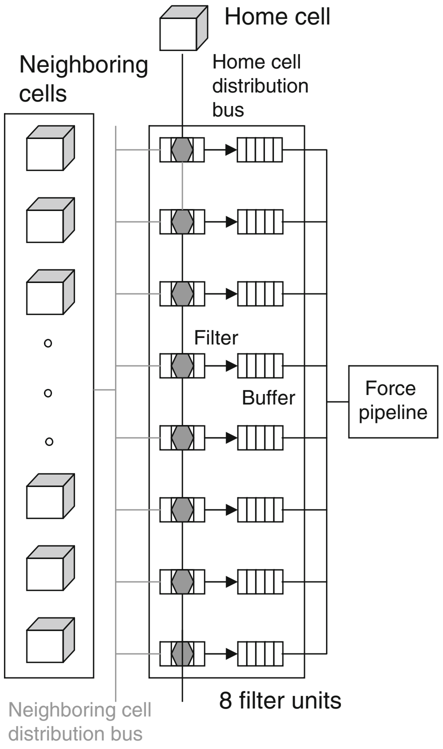

Fig. 12 A preferred mapping of particle pairs onto filter pipelines. Each filter is used to compute all interactions for a single reference particle for an entire cell neighborhood

- The output of each filter is exactly the neighbor-list for its associated reference particle.
- Double buffering enables neighbor-lists to be generated by the filters at the same time they are drained by the force pipelines.

Advantages of this method include:

- Perfect load balance among the filters;
- Little overhead: Each phase consists of 3,150 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 neighbor-list; and
- Simple queueing and control: Neighbor-list generation is decoupled from force computation.

This mapping does require larger queues than mappings where the filter outputs feed more directly into the force pipelines. But since there are BRAMs to spare, this is not likely to have an impact on performance.

A more substantial concern is the granularity of the processing phases. The number of phases necessary to process the particles in a single home cell is $\lceil |\text{particles-in-home-cell}| / |\text{filters}| \rceil$. For small cells the loss of efficiency can become significant. There are several possible solutions.

- Increase the number of filters and further decouple neighbor-list generation from consumption. The reasoning is that as long as the force pipelines are busy, some inefficiency in filtering is tolerable.
- Overlap processing of two home cells. This increases the working set from 18 to 27 cells for a modest increase in number of BRAMs required. One way to implement this is to add a second distribution bus.
- Another way to overlap processing of two home cells is to split the filters among them. This halves the phase granularity and so the expected inefficiency without significantly changing the amount of logic required for the distribution bus.

3.3 Overall Design and Board-Level Issues

In this subsection we describe the overall design (see Fig. 13), especially how data are transferred between host and accelerator and between off-chip and on-chip memory. The reference design assumes an implementation of 8 force and 72 filter pipelines.

1. *Host-Accelerator data transfers:* At the highest level, processing is built around the timestep iteration and its two phases: force calculation and motion update. During each iteration, the host transfers position data to, and acceleration data from, the coprocessor's on-board memory (POS SRAM and ACC SRAM, respectively). With 32-bit precision, 12 bytes are transferred per particle. While the phases are necessarily serial, the data transfers require only a small fraction of the processing time. For example, while the short-range force calculation takes about 55 ms for 100 K particles and increases linearly with particle count through the memory capacity of the board, the combined data transfers of 2.4 MB take only 2–3 ms. Moreover, since simulation proceeds by cell set, processing of the force calculation phase can begin almost immediately as the data begin to arrive.
2. *Board-level data transfers:* Force calculation is built around the processing of successive home cells. 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, ACC data are written back. Focus shifts and a neighboring cell becomes the new home cell. Its cell set is now loaded; in the current scheme this is usually nine cells per shift. The transfers are double buffered to hide latency. 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. Let a cell contain an average of N_{cell} particles. Then $T_{\text{trans}} = 324 \times N_{\text{cell}}/B$ (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 $T_{\text{proc}} = N_{\text{cell}}^2 \times 2\pi/3P$ cycles. This gives the following bandwidth requirement: $B > 155 * P/N_{\text{cell}}$. For $P = 8$ and $N_{\text{cell}} = 175$, $B > 7.1$ bytes per cycle. For many current FPGA processor boards $B \geq 24$.

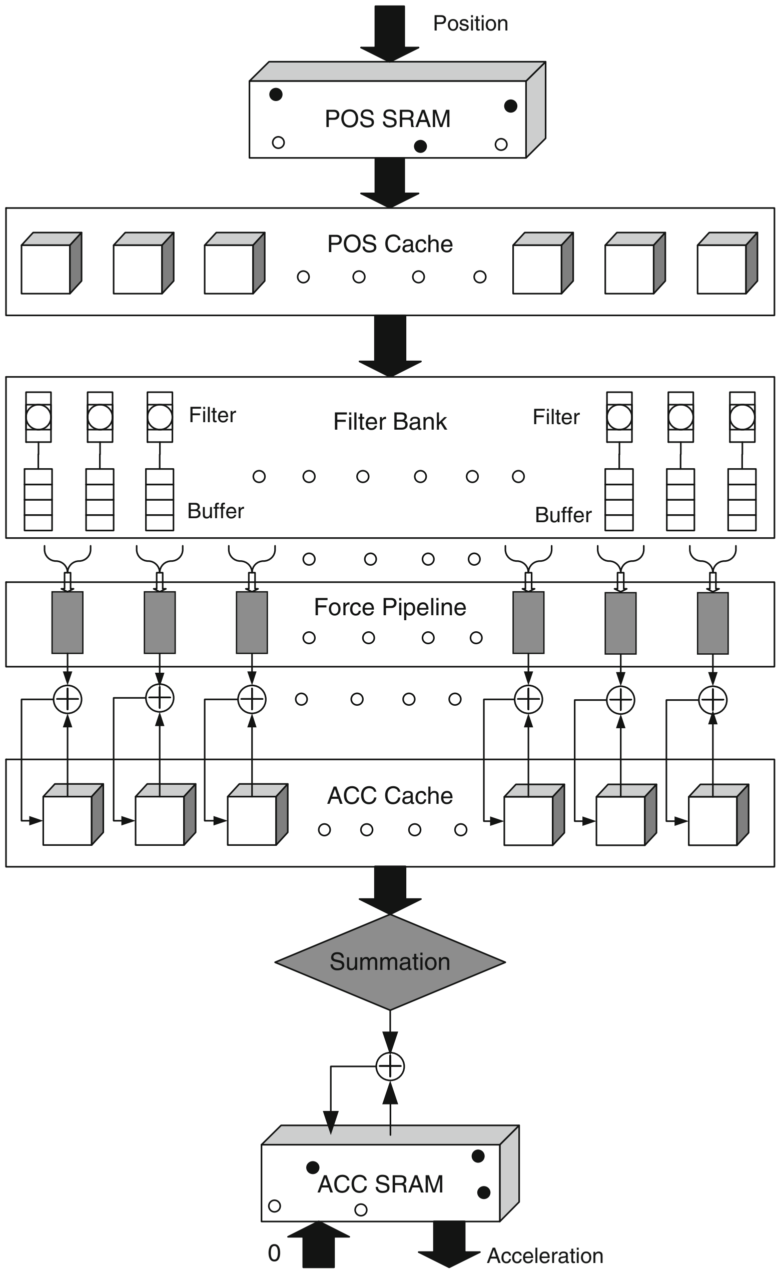

Fig. 13 Schematic of the HPRC MD system

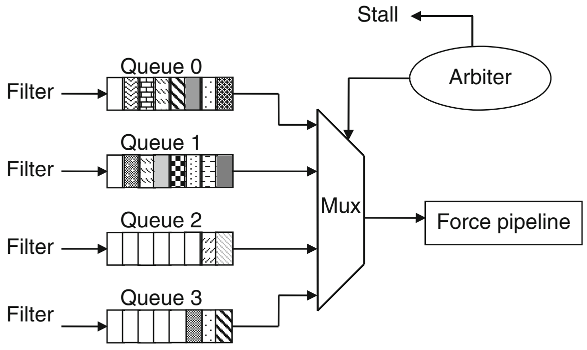

Fig. 14 Concentrator logic between filters and force pipeline

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.

3. *On-chip data transfers:* Force computation has three parts, filtering particle pairs, computing the forces themselves, and combining the accumulated accelerations. In the design of the on-chip data transfers, the goals are simplicity of control and minimization of memory and routing resources. Processing of a home cell proceeds in *cohorts* of reference particles that are processed simultaneously, either 8 or 72 at a time (either one per filter bank or one per force pipeline). This allows a control with a single state machine, minimizes memory contention, and simplifies accumulation. For this scheme to run at high efficiency, two types of load-balancing are required: (1) the work done by various filter banks must be similar and (2) filter banks must generate particle pairs having nontrivial interactions on nearly every cycle.
4. *POS cache to filter pipelines:* Cell set positions are stored in 54–108 BRAMS, i.e., 1–2 BRAMs per dimension per cell. This number depends on the BRAM size, cell size, and particle density. Reference particles are always from the home cell, partner particles can come from anywhere in the cell set.
5. *Filter pipelines to force pipelines:* A concentrator logic is used to feed the output of multiple filters to a pipeline (Fig. 14). Various strategies were discussed in [8].
6. *Force pipelines to ACC cache:* To support N3L, two copies are made of each computed force. One is accumulated with the current reference particle. The other is stored by index in one of the large BRAMs on the Stratix-III. Figure 15 shows the design of the accumulator.

3.4 Preliminary Work in Long-Range Force Computation

In 2005, Prof. Paul Chow's group at the University of Toronto made an effort to accelerate the reciprocal part of SPME on a Xilinx XC2V2000 FPGA [29]. The computation was performed with fixed-point arithmetic that has various precisions

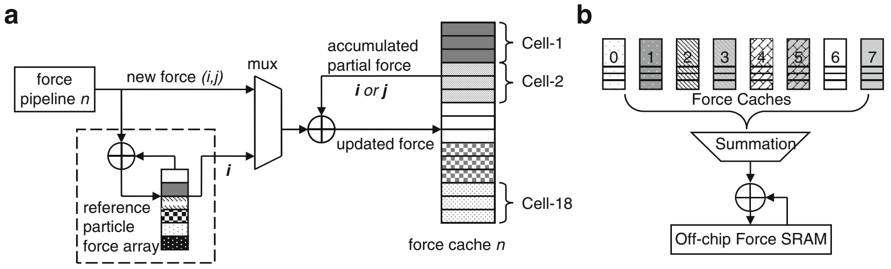

Fig. 15 Mechanism for accumulating per particle forces. (**a**) shows the logic for a single pipeline for both the reference and partner particles. (**b**) shows how forces are accumulated across multiple pipelines

to improve numerical accuracy. Due to the limited logic resources and slow speed grade, the performance was sacrificed by some design choices, such as the sequential executions of the reciprocal force calculation for x, y, and z directions and slow radix-2 FFT implementation. The performance was projected to be a factor of $3\times$ to $14\times$ over the software implementation running in an Intel 2.4GHz Pentium 4 processor. At Boston University the long-range electrostatic force was implemented using Multigrid [17] with a factor of $5\times$ to $7\times$ speed-up reported.

3.5 Preliminary Work in Parallel MD

Maxwell is an FPGA-based computing cluster developed by the FHPCA (FPGA High Performance Computing Alliance) project at EPCC (Edinburgh Parallel Computing Centre) at the University of Edinburgh [3]. The architecture of Maxwell comprises 32 blades housed in an IBM Blade Center. Each blade consists of one Xeon processor and 2 Virtex-4 FX-100 FPGAs. The FPGAs are connected by a fast communication subsystem which enables the total of 64 FPGAs to be connected together in an 8×8 torus. Each FPGA also has four 256 MB DDR2 SDRAMs. The FPGAs are connected with the host via a PCI bus.

In 2011, an FPGA-accelerated version of LAMMPS was reported to be implemented on Maxwell [24, 37]. Only range-limited non-bonded forces (including potential and virial) were computed on the FPGAs with 4 identical pipelines/FPGA. A speed-up of up to $14\times$ was reported for the kernel (excluding data communication) on two or more nodes of the Maxwell machine, although the end-to-end performance was worse than the software-only version.

This work essentially implemented the inner-loop of a neighbor-list-based force computation as the FPGA kernel. Every time a particle and its neighbor-list would be sent to the FPGAs from the host and then corresponding forces would be computed on the FPGAs. This incurred tremendous amount of data communication which ultimately resulted in the slowdown of the FPGA-accelerated version.

They simulated a Rhodopsin protein in solvated lipid bilayer with LJ forces and PPPM method. The 32 K system was replicated to simulate larger systems. This work, however, to the best of our knowledge, is the first to integrate an FPGA MD kernel to a full-parallel MD package.

4 Future Challenges and Opportunities

The future of FPGA-accelerated MD vastly depends on the cooperation and collaboration among computational biologists, computer architects, and board/EDA tool vendors. In the face of the high bar set by GPU implementations, researchers and engineers from all of these three sectors must come together to make this a success. The bit-level programmability and fast data communication capability, together with their power efficiency, do make FPGAs seem like the best candidate for MD accelerator. But to realize the potential, computer architects will have to work with the computational biologists to understand the characteristics of the existing MD packages and develop FPGA kernels accordingly. The board and EDA tool vendors will have to make FPGA devices much easier to deploy. Currently FPGA kernels are mostly designed and managed by hardware engineers. A CUDA-like breakthrough here would make FPGAs accessible to a much broader audience.

Below, we discuss some of the specific challenges that need to be addressed in order to achieve the full potential of FPGAs in accelerating MD. These challenges provide researchers with great opportunities for inventions and advancements that are very likely to be applicable to other similar computational problems, e.g., N-body simulations.

4.1 Integration into Full-Parallel Production MD Packages

After a decade of research on FPGA-accelerated MD, with many individual pieces of work here and there, none of the widely used MD packages have an FPGA-accelerated version. Part of this is because FPGA developers have only focused on individual sections of the computation. But another significant reason is the lack of understanding of how these highly optimized MD packages work and what needs to be done to get the best out of FPGAs, without breaking the structure of the original packages. Researchers need to take a top-down approach and focus on the need of the software. Certain optimizations on the CPUs may need to be revisited, because we may have more efficient solutions on FPGAs, e.g. table-interpolation using BRAM as described in Sect. 3.1. Also, more effort must be given on overlapping computation and communication.

4.2 Use of FPGAs for Inter-Node Communication

While CPU-only MD remains compute-bound for at least a few hundred compute nodes, that is not the case for accelerated versions. It should be evident from the GPU experience that communication among compute nodes will become a bottleneck even for small systems. The need for fast data communication is especially crucial in evaluating the long-range portion of electrostatic force, which is often based on the 3D FFT and requires all-to-all communication during a timestep. Without substantial improvement in such inter-node communication, FPGA-acceleration will be limited to only a few times of speed-up. This presents a highly promising area of research where FPGAs can be used directly for communication between compute nodes. FPGAs are already used in network routers and seem like a natural fit for this purpose [20].

4.3 Building an Entirely FPGA-Centric MD Engine

As Moore's law continues, FPGAs are equipped with more functionality than ever. It is possible to have embedded processors on FPGAs, either soft or hard, which makes it feasible to create an entirely FPGA-centric MD engine. In such an engine, overall control and simple software tasks will be done on the embedded processors while the heavy work like the non-bonded force computations will be implemented on the remaining logic. Data communication can also be performed using the FPGAs, completely eliminating general purpose CPUs from the scene. Such a system is likely to be highly efficient, in terms of both computational performance and energy consumption.

4.4 Validating Simulation Quality

While MD packages typically use double-precision floating point for most of the computation, most FPGA work used fixed, semi-floating or a mixture of fixed and floating point for various stages of MD. Although some of these studies verified accuracy through various metrics, none of the FPGA-accelerated MD work presented results of significantly long (e.g., month-long) runs of MD. Thus it is important to address this issue of accuracy. This may mean revisiting precision and interpolation order in the force pipelines.

Acknowledgments This work was supported in part by the NIH through award #R01-RR023168-01A1 and by the MGHPCC.

References

1. S.A. Adcock, J.A. McCammon, Molecular dynamics: survey of methods for simulating the activity of proteins. Chem. Rev. **106**(5), 1589–1615 (2006)
2. J.A. Anderson, C.D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units. J. Comput. Phys. **227**(10), 5342–5359 (2008)
3. R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. McCormick, G. Smart, R. Smart, A. Cantle, R. Chamberlain, G. Genest, Maxwell - a 64 FPGA supercomputer, in *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS)* (2007), IEEE Computer Society, Washington, DC, USA, pp. 287–294
4. K.J. Bowers, E. Chow, H. Xu, R.O. Dror, M.P. Eastwood, B.A. Gregersen, J.L. Klepeis, I. Kolossvary, M.A. Moraes, F.D. Sacerdoti, J.K. Salmon, Y. Shan, D.E. Shaw, Scalable algorithms for molecular dynamics simulations on commodity clusters, in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC)* (2006), ACM New York, NY, USA, pp. 84:1–84:13
5. B.R. Brooks, C.L. Brooks III, A.D. Mackerell Jr., L. Nilsson, R.J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A.R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R.W. Pastor, C.B. Post, J.Z. Pu, M. Schaefer, B. Tidor, R.M. Venable, H.L. Woodcock, X. Wu, W. Yang, D.M. York, M. Karplus, CHARMM: the biomolecular simulation program. J. Comput. Chem. **30**(10, Sp. Iss. SI), 1545–1614 (2009)
6. D.A. Case, T.E. Cheatham, T. Darden, H. Gohlke, R. Luo, K.M. Merz Jr., A. Onufriev, C. Simmerling, B. Wang, R.J. Woods, The Amber biomolecular simulation programs. J. Comput. Chem. **26**(16), 1668–1688 (2005)
7. M. Chiu, M.C. Herbordt, Efficient particle-pair filtering for acceleration of molecular dynamics simulation, in *International Conference on Field Programmable Logic and Applications (FPL)* (2009), ACM New York, NY, USA, pp. 345–352
8. M. Chiu, M.C. Herbordt, Molecular dynamics simulations on high-performance reconfigurable computing systems. ACM Trans. Reconfigurable Tech. Syst. (TRETS) **3**(4), 23:1–23:37 (2010)
9. M. Chiu, M.A. Khan, M.C. Herbordt, Efficient calculation of pairwise nonbonded forces, in *The 19th Annual International IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2011), IEEE Computer Society Washington, DC, USA, pp. 73–76
10. S. Chiu, Accelerating molecular dynamics simulations with high-performance reconfigurable systems, PhD dissertation, Boston University, USA, 2011
11. T. Darden, D. York, L. Pedersen, Particle mesh Ewald: an N.log (N) method for Ewald sums in large systems. J. Chem. Phys. **98**(12), 10089–10092 (1993)
12. W.A. Eaton, V. Muñoz, P.A. Thompson, C.K. Chan, J. Hofrichter, Submillisecond kinetics of protein folding. Curr. Opin. Struct. Biol. **7**(1), 10–14 (1997)
13. R.D. Engle, R.D. Skeel, M. Drees, Monitoring energy drift with shadow Hamiltonians. J. Comput. Phys. **206**(2), 432–452 (2005)
14. P.L. Freddolino, A.S. Arkhipov, S.B. Larson, A. McPherson, K. Schulten, Molecular dynamics simulations of the complete satellite tobacco mosaic virus. Structure **14**(3), 437–449 (2006)
15. Gidel, Gidel website (2009), http://www.gidel.com. Accessed 17 April 2012
16. GROMACS, GROMACS installation instructions for GPUs (2012), http://www.gromacs.org/Downloads/Installation_Instructions/GPUs. Accessed 17 April 2012
17. Y. Gu, M.C. Herbordt, FPGA-based multigrid computation for molecular dynamics simulations, in *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2007), pp. 117–126
18. Y. Gu, T. Vancourt, M.C. Herbordt, Explicit design of FPGA-based coprocessors for short-range force computations in molecular dynamics simulations. Parallel Comput. **34**(4–5), 261–277 (2008)
19. D.J. Hardy, NAMD-Lite (2007), http://www.ks.uiuc.edu/Development/MDTools/namdlite/. University of Illinois at Urbana-Champaign. Accessed 17 April 2012

20. M. Herbordt, M. Khan, Communication requirements of fpga-centric molecular dynamics, in *Proceedings of the Symposium on Application Accelerators for High Performance Computing* (2012)
21. B. Hess, C. Kutzner, D. van der Spoel, E. Lindahl, GROMACS 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. J. Chem. Theor. Comput. **4**(3), 435–447 (2008)
22. R. Hockney, S. Goel, J. Eastwood, Quiet high-resolution computer models of a plasma. J. Comput. Phys. **14**(2), 148–158 (1974)
23. L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, K. Schulten, NAMD2: Greater scalability for parallel molecular dynamics. J. Comput. Phys. **151**, 283–312 (1999)
24. S. Kasap, K. Benkrid, A high performance implementation for molecular dynamics simulations on a FPGA supercomputer, in *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)* (2011), IEEE Computer Society Washington, DC, USA, pp. 375–382
25. F. Khalili-Araghi, E. Tajkhorshid, K. Schulten, Dynamics of K+ ion conduction through Kv1.2. Biophys. J. **91**(6), 72–76 (2006)
26. V. Kindratenko, D. Pointer, A case study in porting a production scientific supercomputing application to a reconfigurable computer, in *14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2006), IEEE Computer Society Washington, DC, USA, pp. 13–22
27. S. Kumar, C. Huang, G. Zheng, E. Bohm, A. Bhatele, J.C. Phillips, H. Yu, L.V. Kalé, Scalable molecular dynamics with NAMD on the IBM Blue Gene/L system. IBM J. Res. Dev. **52**(1–2), 177–188 (2008)
28. R. Larson, J. Salmon, R. Dror, M. Deneroff, C. Young, J. Grossman, Y. Shan, J. Klepeis, D. Shaw, High-throughput pairwise point interactions in Anton, a specialized machine for molecular dynamics simulation, in *IEEE 14th International Symposium on High Performance Computer Architecture (HPCA)* (2008), IEEE Computer Society Washington, DC, USA, pp. 331–342
29. S. Lee, An FPGA implementation of the Smooth Particle Mesh Ewald reciprocal sum compute engine, Master's thesis, The University of Toronto, Canada, 2005
30. A.D. MacKerell, N. Banavali, N. Foloppe, Development and current status of the CHARMM force field for nucleic acids. Biopolymers **56**(4), 257–265 (2000)
31. G. Moraitakis, A.G. Purkiss, J.M. Goodfellow, Simulated dynamics and biological macromolecules. Rep. Progr. Phys. **66**(3), 383 (2003)
32. T. Narumi, Y. Ohno, N. Futatsugi, N. Okimoto, A. Suenaga, R. Yanai, M. Taiji, A high-speed special-purpose computer for molecular dynamics simulations: MDGRAPE-3. NIC Workshop, From Computational Biophysics to Systems Biology, NIC Series, vol. 34 (2006), pp. 29–36
33. L. Nilsson, Efficient table lookup without inverse square roots for calculation of pair wise atomic interactions in classical simulations. J. Comput. Chem. **30**(9), 1490–1498 (2009)
34. J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kalé, K. Schulten, Scalable molecular dynamics with NAMD. J. Comput. Chem. **26**(16), 1781–1802 (2005)
35. J.C. Phillips, J.E. Stone, K. Schulten, Adapting a message-driven parallel application to GPU-accelerated clusters, in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC)* (2008), IEEE Press Piscataway, NJ, USA, pp. 8:1–8:9
36. L. Phillips, R.S. Sinkovits, E.S. Oran, J.P. Boris, The interaction of shocks and defects in Lennard-Jones crystals. J. Phys.: Condens. Matter **5**(35), 6357–6376 (1993)
37. S. Plimpton, Fast parallel algorithms for short-range molecular dynamics. J. Comput. Phys. **117**(1), 1–19 (1995)
38. J.W. Ponder, D.A. Case, Force fields for protein simulations. Adv. Protein Chem. **66**, 27–85 (2003)
39. D.C. Rapaport, *The Art of Molecular Dynamics Simulation*, 2nd edn. (Cambridge University Press, London, 2004)

40. P. Schofield, Computer simulation studies of the liquid state. Comp. Phys. Comm. **5**(1), 17–23 (1973)
41. R. Scrofano, M. Gokhale, F. Trouw, V.K. Prasanna, A hardware/software approach to molecular dynamics on reconfigurable computers, in *The 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2006), IEEE Computer Society Washington, DC, USA, pp. 23–34
42. Y. Shan, J. Klepeis, M. Eastwood, R. Dror, D. Shaw, Gaussian split Ewald: a fast Ewald mesh method for molecular simulation. J. Chem. Phys. **122**(5), 54101:1–54101:13 (2005)
43. D.E. Shaw, M.M. Deneroff, R.O. Dror, J.S. Kuskin, R.H. Larson, J.K. Salmon, C. Young, B. Batson, K.J. Bowers, J.C. Chao, M.P. Eastwood, J. Gagliardo, J.P. Grossman, C.R. Ho, D.J. Ierardi, I. Kolossváry, J.L. Klepeis, T. Layman, C. McLeavey, M.A. Moraes, R. Mueller, E.C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, S.C. Wang, Anton, a special-purpose machine for molecular dynamics simulation, in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)* (2007), ACM New York, NY, USA, pp. 1–12
44. D.E. Shaw, M.M. Deneroff, R.O. Dror, J.S. Kuskin, R.H. Larson, J.K. Salmon, C. Young, B. Batson, K.J. Bowers, J.C. Chao, M.P. Eastwood, J. Gagliardo, J.P. Grossman, C.R. Ho, D.J. Ierardi, I. Kolossváry, J.L. Klepeis, T. Layman, C. McLeavey, M.A. Moraes, R. Mueller, E.C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, S.C. Wang, Anton, a special-purpose machine for molecular dynamics simulation. Comm. ACM **51**(7), 91–97 (2008)
45. D.E. Shaw, R.O. Dror, J.K. Salmon, J.P. Grossman, K.M. Mackenzie, J.A. Bank, C. Young, M.M. Deneroff, B. Batson, K.J. Bowers, E. Chow, M.P. Eastwood, D.J. Ierardi, J.L. Klepeis, J.S. Kuskin, R.H. Larson, K. Lindorff-Larsen, P. Maragakis, M.A. Moraes, S. Piana, Y. Shan, B. Towles, Millisecond-scale molecular dynamics simulations on Anton, in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC)* (2009), ACM New York, NY, USA, pp. 39:1–39:11
46. R.D. Skeel, I. Tezcan, D.J. Hardy, Multiple grid methods for classical molecular dynamics. J. Comput. Chem. **23**(6), 673–684 (2002)
47. M. Snir, A note on N-body computations with cutoffs. Theor. Comput. Syst. **37**(2), 295–318 (2004)
48. J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, K. Schulten, Accelerating molecular modeling applications with graphics processors. J. Comput. Chem. **28**(16), 2618–2640 (2007)
49. L. Verlet, Computer "Experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. Phys. Rev. **159**(1), 98–103 (1967)
50. C. Young, J.A. Bank, R.O. Dror, J.P. Grossman, J.K. Salmon, D.E. Shaw, A 32x32x32, spatially distributed 3D FFT in four microseconds on Anton, in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC)* (2009), ACM New York, NY, USA, pp. 23:1–23:11