Architecture/Algorithm Codesign of Molecular Dynamics Processors

Martin C. Herbordt
Computer Architecture and Automated Design Laboratory
Department of Electrical and Computer Engineering
Boston University; Boston, MA 02215

***Abstract*—Molecular Dynamics is notable in High Performance Computing in that it is both sufficiently critical, and not well-enough served by off-the-shelf processors, that it has been continually targeted with domain-specific architectures. These range from the common, i.e., use of accelerators, to FPGA-centric servers describe here, to full-scale ASIC-based systems (e.g., the Anton processor). The strict constraints on achieving strong scaling leads to novel designs, which in turn suggest restructurings of the underlying algorithms. These in turn suggest some changes to the hardware. We discuss the technological issues involved, the implications of various approaches on architecture/algorithm codesign, and how these are likely to affect future high-end processors, both domain-specific and off-the-shelf.**

I. Introduction

"*Codesign* refers to a computer system design process where scientific problem requirements influence architecture design and technology and constraints inform formulation and design of algorithms and software.[1]" Both halves of this statement are obvious: architects optimize systems based on software performance and application developers (re)formulate software accounting for the target system. What makes codesign topical is that it combines these two domains. Implied in the approach is that it be economically viable: Completely off-the-shelf is perhaps not acceptable, but neither is unrealistic variation in either hardware or software.

Codesign is most interesting when there are many viable ways to solve the target application, preferably with variations occurring at multiple levels, e.g., choice of algorithm, code transformation, arithmetic mode, and communication mechanism. For codesign to be useful, these different variations must have different optimal target architectures. Molecular Dynamics (MD) fits these criteria: variations at all these levels have been described extensively and the optimality of the variations often differs depending on hardware support. Moreover, variations in different parts of the application affect each other in their own "codesign" process.

This work is based on our codesign experiences with MD and its mapping onto FPGA-centric clusters. We cast codesign as an iterative process; while hardware and software can be optimized jointly, we believe that for complex applications such as MD altering multiple variables simultaneously may not be the best the best method to traverse the solution space. Specifically, we describe three phases of codesign. In phase 0, we hypothesize FPGA-centric clusters as the initial target architecture, although we also use GPU and ASIC mappings for reference. Phase 1 is the mapping of MD to the target hardware and involves modifying the application. In phase 2, we suggest changes in hardware based on these new application characteristics.

A potential problem with this approach is local minima: the initial target architecture may specify a point in the solution space from which no minimal path to the optimal may exist. The practical answer is simply to measure the results against existing MD systems: if the codesigned system cannot improve on what exists then the hypothesis fails.

The rest of this paper is organized as follows. In the next section we briefly describe MD. In the three following sections, we examine three classes of optimizations of MD with respect to the target architecture: (i) polynomial evaluation; (ii) neighbor lists; and (iii) communication. At the end of each of these sections we briefly discuss possible changes to the hardware. In the conclusion we summarize current status.

II. MD Preliminaries

MD is central to computational biochemistry and comprises a large fraction of all High Performance Computing cycles. This section provides background based on material in [2].

MD is an iterative application of Newtonian mechanics to ensembles of atoms and molecules (see, e.g., [3] for details). Each iteration consists of two phases, force computation and motion integration. The forces may include non-bonded (Lennard-Jones or LJ and Coulomb) and bonded terms:

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

Because the bonded terms affect only neighboring atoms, computing their effect is $O(N)$ in the number of particles N being simulated. Motion integration is also $O(N)$. The LJ force for particle i can be expressed as:

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

where the ϵ_{ab} and σ_{ab} are parameters related to the types of particles. The Coulombic force can be expressed as:

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

This work was supported in part by the NIH through award #R01-RR023168-01A1 and the NSF through award #CNS-1205593. Web: www.bu.edu/caadlab. Email: herbordt@bu.edu

A standard way of computing the non-bonded forces is by applying a cut-off. Then 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. The problem with cut-off is that it introduces an error.

A number of methods have been developed to compute the force outside the cut-off. Two of the most popular are based on Ewald Sums [4], which generally involves an FFT, and multigrid [5]. We use the standard convention of calling the component within the cut-off *range-limited* and that outside *long-range*. Depending on the methods used to compute these components, different correction terms are applied to simplify the computation. For single threaded execution, the range-limited force dominates accounting for roughly 90% of the time. When parallelized, however, the long-range force can dominate, especially for large computer systems operating on small to medium sized problems.

III. Polynomial Evaluation

The range-limited force on each particle is the sum of its interations with all particles within radius r_c. These are computed using the expressions given in the previous section plus correction terms to smooth the cut-off and simplify the long-range force. These expressions are sufficiently complex that a large number of choices emerge in their evaluation (see, e.g., [2], [6], [7], [8] and references therein). For FPGA-based systems the object is to minimize pipeline size to enable more computation per area and thus more parallelism and faster execution. Sufficient simulation quality must be guaranteed, however, especially to ensure stability. Similar reasoning goes into ASIC designs; for CPUs and GPUs there are fewer choices, but lower precision and use of integer rather than floating point yield higher performance with lower quality.

In our studies we have considered direct computation versus table look up with interpolation; with table look up, the number of bins, order of interpolation, and method of generating coefficients; precision; and arithmetic mode including fixed point, floating point, and a hybrid representation. Considerations are use of FPGA resources versus simulation quality. Additionally, some particular hardware features or application characteristics shift these tradeoffs. An interesting observation is that our system has gone through three generations of solutions. The changes are due to changing application requirements and characteristics of the base hardware.

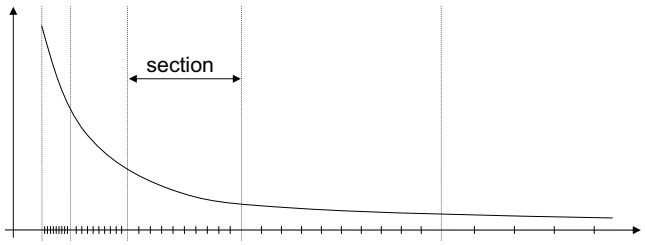

Fig. 1. Table look-up varies in precision across r^{-k}.

1. MD software system = ProtoMol [9], FPGA = Xilinx Virtex 5 [8], [10], [7]. The significance of the ProtoMol integration is that it supports a Multigrid implementation of the long-range force and has the following Coulombic component of the range-limited force:

$$\frac{\mathbf{F}_{ji}^{C_{rl}}(r_{ji}(a,b))}{\mathbf{r_{ji}}} = QQ_{ab}(r^{-3} + G_0 + G_1 r^2 + G_2 r^4), \quad (4)$$

where QQ_{ab} is a precomputed parameter and the G_i are the smoothing terms. The significance of the Virtex 5 is that it has limited floating point support. We made the following phase 1 changes based on the choices of ProtoMol and Virtex 5.

- Sweet spots for precision are at 35 and 53 bits as derived from measured stability and resource utilization. The unusual word sizes are due to size of the hardwired multiplier blocks.
- The lack of floating point support indicated using table look up. The functions being evaluated lend themselves to two optimizations: using r^2 rather than r as an index, and having the bin size double with increasing interval (see Figure 1).
- We found two preferred designs based on acceptable error and resource utilization: 3rd order with 128 bins per interval and 2nd order with 512 bins per interval.
- An observation about exponent usage led us to the development of a more efficient floating point representation, Semi-FP, which uses half the resources.

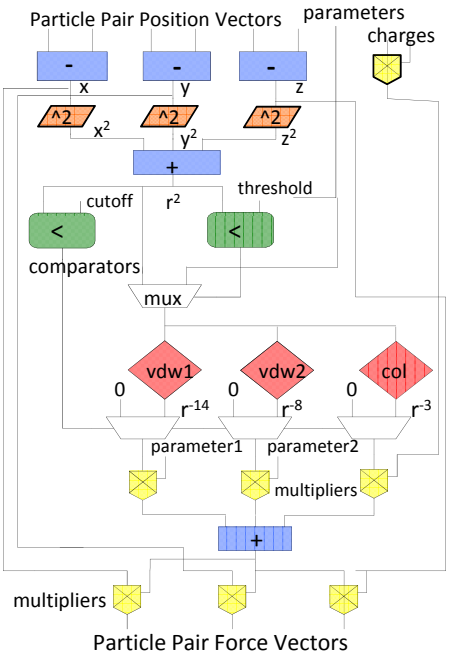

Fig. 2. Force pipeline template.

2. MD software system = ProtoMol, FPGA = Altera Stratix III [11], [2]. The significance of the Stratix III is that it has improved floating point support. At the same time, Altera released its Floating Point Compiler that optimizes floating point pipelines. Together, these led to the following changes.

- Direct computation rather than table look up (see Figure 2).
- Mix of fixed and single precision floating point.
- Higher precision for accumulation.

3. MD software system = NAMD [12], FPGA = Altera Stratix IV [6]. The major change with moving to NAMD is that the range-limited force calculation is substantially more complex. Here is the electrostatic calculation.

$$E = \frac{1}{8\pi\epsilon_0} \sum_n \sum_{i=1}^{N} \sum_{i=0}^{n} \frac{q_i q_j}{|r+nL|} erfc(\frac{|r+nL|}{\sqrt{2}\sigma}) \quad (5)$$

A more general trend, inspired by the emergence of GPUs as MD accelerators, is a better of understanding the effect of precision on simulation quality. Changes are as follows.

- The evaluation is done with table look up.
- First order interpolation is used with 256 bins per interval and 10 intervals. We find that multipliers are the critical resource, rather than on-chip memory, so select the lower order solution.

We now comment on analogous mappings of polynomial evaluation to two other processors. NVIDIA GPUs have a texture processor that gives direct support for first order interpolation making this the method of choice [13]. Also with GPUs, floating point support is superior than integer so floating point is used exclusively. And in older GPUs, single precision was proportionally much faster than double precision leading to the use of single precision throughout except for accumulation [13]. The Anton processor from D.E. Shaw uses custom ASICs and was developed in parallel to the work described here [14]. Use of ASICs leads to a different solution: Anton uses 2nd and 3rd order interpolation with variable segment size and a small number of bins (256 total).

The phase 1 implementation points to possible hardware improvements (phase 2). For FPGAs, more multipliers and better floating point support are obvious recommendations. For GPUs, there is already direct support for first order interpolation. Naturally second order would improve the quality. And since a substantial fraction of the computation can be done in fixed point, better support there would also help.

IV. Neighbor Lists

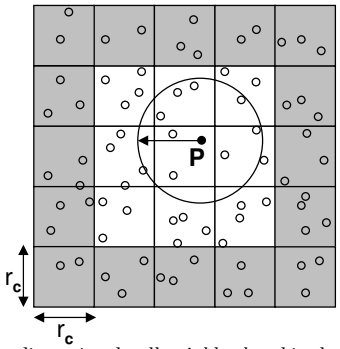

Fig. 3. P's two dimensional *cell neighborhood* is shown in white; cells have edge size equal to the cut-off radius. Particles within the P's cut-off circle are in P's neighbor list.

Two methods are used to prevent zero calculations of the range-limited force: cell lists and neighbor lists (see Figure 3). With cell lists, the simulation space is partitioned (typically) 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*). Cell lists can be created very quickly but have only 15.5% efficiency. With neighbor lists, P has associated with it a list of exactly those partner particles within r_c. Creating neighbor lists is much more time consuming and so is usually done only every 15-20 iterations or as needed. An optimization that increases the recompute interval is to increase the neighbor list radius r_{nl}; the closer r_{nl} to r_c the higher the efficiency but the more frequently the neighbor lists need to be adjusted.

The problems with using neighbor lists on accelerators include the size of the lists, the complexity of building the lists, and the need to follow pointers. Since accelerators generally trade off latency for throughput, the last of these is especially problematic. The FPGA (and ASIC) solution begins with the trivial observation that the force pipeline already computes r^2 (see Figure 2). We use this observation by noting that only 15.5% of the computations need to advance from this point through the rest of the pipeline. It should thus be possible to replicate the r^2 calculation (the filter) 6 times for each copy of the rest of the force pipeline. This is our first "candidate" filter design: it has several filters per force pipeline; runs at nearly 100% utilization, rather than 15.5%; requires 2.5 times the area per pipeline; and has $2.6\times$ the throughput.

We note that r_{filter} need not be as accurate as r_{force} used for the full force computation. This requires rounding up r_{filter} so that it is $> r_c$, which reduces the utilization slightly (to 99.5%). But it also reduces the size of the entire combined filter/force pipeline by a factor of 2 over the first candidate design. Summarizing the second candidate filter design: it has the same number of filters per force pipeline; runs at 99.5% utilization, which results in 3% extra work; requires 25% more area per pipeline than the original; and has $5.2\times$ the throughput of the original design.

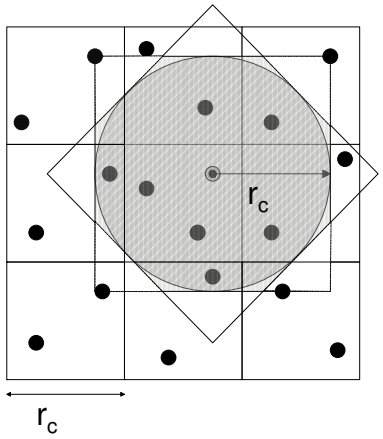

Fig. 4. Filtering with planes rather than a sphere – 2D analogue.

For the third canditate design we note that a linear rather than a quadratic filter avoids multiplies, which are relatively more expensive resource than adds on an FPGA. In this method we avoid multiplication by thresholding with planes rather than a sphere (see Figure 4 for the 2D analog). 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% added work with respect to perfect filtering. The advantage is that no multipliers are needed by the filter, reducing the total by nearly 25%. Where that is the critical resource, as in current FPGAs, the total throughput is increased by almost 18%.

We now comment on analogous mappings of filter with ASICs; we are not aware of a GPU mapping that implements filtering. Anton uses a reduced precision filter similar to that described here [15], but we are not aware that it uses a linear filter. This may be because all arithmetic units are hardwired, rather than just multipliers.

Possible hardware improvements (phase 2) are as follows. For FPGAs, the filtering scheme described is an excellent fit with balanced use of resources. GPUs could implement filtering with inter-thread communication.

V. Strong Scaling

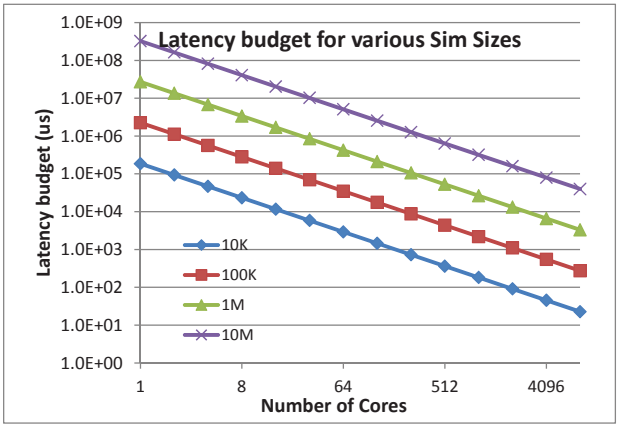

Fig. 5. Time per timestep for various simulation sizes and core counts assuming perfect scaling. This shows computation only and gives the time budget for communication.

While the long-range force only takes about 10% of the compute cycles, this fraction assumes that communication latency can be hidden. This becomes increasingly difficult as cluster size increases and problem size remains fixed (see Figure 5 and [16] for details). Since many critical applications, e.g., protein folding, involve fewer than 100K particles and long time scales (10^{12} timesteps or more), strong scaling is critical. Note that for a cluster with 8K cores running a 100K particle simulation, the communication budget is $200\mu s$ per timestep. Since this communication involves multiple all-to-all communications among (in this case) hundreds of nodes, this is likely to require extremely careful network design.

Communication time is the sum of time-of-flight latency and transmission time (inversely proportional to bandwidth). Raw bandwidth and latency are basic in this calculation, but there are other factors. On the negative side, congestion both increases latency and decreases apparent bandwidth. On the positive side, judicious in-channel filtering can reduce the amount of data that needs to be transmitted and so increase the apparent bandwidth. These two factors add a third criterion to our communication network design (beyond latency and bandwidth): application aware processing.

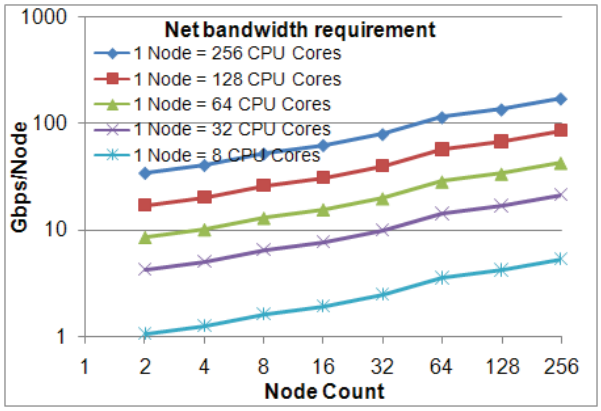

Fig. 6. Bandwidth requirement for various systems for a 100K particle problem size. Systems are ideal with all-to-all interconnect and no in-channel particle filtering.

We now explore these three criteria, beginning with bandwidth. For this we summarize the discussion in [16]. MD communication has four components: range-limited, which is mostly nearest neighbor; grid interpolation, which is also near neighbor; grid distribution, which is a scatter and its inverse which is a gather; and FFT which involves a transpose. We model this communication for NAMD and validate the model with a medium-sized cluster. Using this model we derive the overall bandwidth required per node; Figure 6 shows these results for nodes of various sizes and counts.

Next we determine the actual bandwidth requirement, assuming a 3D bi-directional torus network. Our goal here is to determine the bandwidth needed for each of the 12 channels on a node. Data communication for the range-limited non-bonded force computation is contained within neighboring nodes at 1-3 hops. This will on average cause about a $2\times$ increase in data communication. At the same time, however, the FPGA easily supports in-channel filtering to remove particles not needed by a particular neighbor. For typical cell/patch and cut-off sizes, this results in a reduction of data to be transferred (weighted by number of hops) to 73% of the original. For long-range communication, all-to-all communication is required which roughly doubles the data amount for a $4 \times 4 \times 4$ node system and further doubles it on an $8 \times 8 \times 8$ node system.

The final bandwidth requirement for a 100K simulation

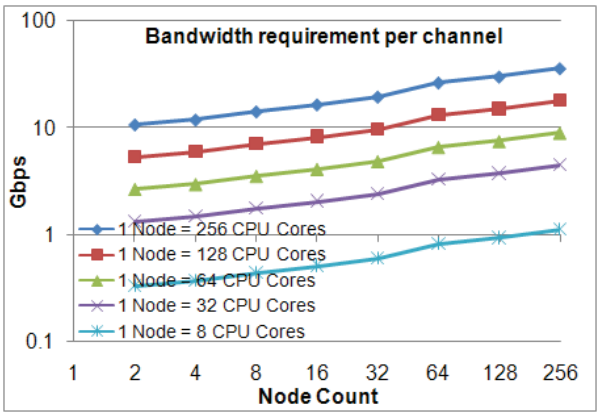

Fig. 7. Bandwidth per channel requirement for various systems for a 100K problem size. Some likely system information is integrated such as number of hops per packet and in-channel particle filtering.

is shown in Figure 7. The series `node = 256 cores` represents projected performance of a 4 FPGA node. For a system with 64 such nodes, configured in a 3D bidirectional torus, each channel must support 27Gb/s bandwidth; this is possible with 2 14Gb/s serial links. The aggregate of 24 links is a small fraction of the roughly 200 available among the four FPGAs on such a node.

To minimize latency we stipulate direct accelerator to accelerator connections as available with FPGAs (and ASICs) and likely to be available soon with many-core processors. This bypasses the bus and NIC. FPGA systems like this have been built for numerous applications, including simulation of neural networks [17]. Point-to-point time-of-flight at the application layer is typically less than 100ns with 50ns reported.

The third aspect of the design involves support for application-aware processing. In MD there are at least two opportunities, in-channel filtering and application-aware packet routing. The first of these has already been mentioned above: when transferring data in the range-limited force calculation a simple position filter removes over half of the data from the stream. The second is possible with regular deterministic patterns. In the case of a transpose, there is a pattern-specific schedule based on de Bruijn sequences that guarantees congestion-free routing [18]. Schedules can be implemented with in-channel reordering.

Possible hardware improvements (phase 2) are as follows. For FPGAs, we find again that they are an excellent match for the proposed scheme. In this case this is not surprising since the primary market for high-end FPGAs is as network routers. If follows that for many-core processors to be able to follow suit they would need similar communication support.

VI. Discussion

We have described the codesign process for Molecular Dynamics using FPGA-centric clusters as a starting point (phase 0). We describe three aspects of MD implementation: polynomial evaluation, creation of neighbor lists, and communication. We find that for neighbor lists and communication, the FPGA's unusual capabilities—including flexibility and communication support—suggest restructuring the application to improve performance (phase 1). Finally, these restructurings sometimes suggest improvements to the hardware (phase 2). Overall we find that the concern of remaining in a local minimum is unfounded: per chip performance is competitive [6] while the intrinsic communication support is unparalleled for COTS parts.

References

[1] *Scientific Discovery through Advanced Computing (SciDAC): Co-Design*, Department of Energy: Office of Science, http://science.energy.gov/ ascr/research/scidac/co-design/, Accessed 11/2013.

[2] M. Chiu and M. Herbordt, "Molecular dynamics simulations on high performance reconfigurable computing systems," *ACM Trans. on Reconfigurable Technology and Systems*, vol. 3, no. 4, pp. 1–37, 2010.

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

[4] T. Darden, D. York, and L. Pedersen, "Particle Mesh Ewald: an $N\log(N)$ method for Ewald sums in large systems," *J. of Chemical Physics*, vol. 98, pp. 10 089–10 092, 1993.

[5] R. Skeel, I. Tezcan, and D. Hardy, "Multiple grid methods for classical molecular dynamics," *J. of Computational Chemistry*, vol. 23, pp. 673–684, 2002.

[6] M. Chiu, M. Khan, and M. Herbordt, "Efficient calculation of pairwise nonbonded forces," in *Proc. IEEE Symp. on Field Programmable Custom Computing Machines*, 2011.

[7] Y. Gu, T. VanCourt, and M. Herbordt, "Accelerating molecular dynamics simulations with configurable circuits," *IEE Proc. on Computers and Digital Technology*, vol. 153, no. 3, pp. 189–195, 2006.

[8] ——, "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.

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

[10] Y. Gu and M. Herbordt, "FPGA-based multigrid computations for molecular dynamics simulations," in *Proc. IEEE Symp. on Field Programmable Custom Computing Machines*, 2007, pp. 117–126.

[11] M. Chiu and M. Herbordt, "Efficient filtering for molecular dynamics simulations," in *Proc. IEEE Conf. on Field Programmable Logic and Applications*, 2009.

[12] J. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. Skeel, L. Kale, and K. Schulten, "Scalable molecular dynamics with NAMD," *J. of Computational Chemistry*, vol. 26, pp. 1781–1802, 2005.

[13] J. Phillips, J. Stone, and K. Schulten, "Adapting a message-driven parallel application to GPU-accelerated clusters," in *Proc. ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis – Supercomputing*, 2008.

[14] Shaw, D.E., et al., "Anton, a special-purpose machine for molecular dynamics simulation," in *Proc. Int. Symp. on Computer Architecture*, 2007, pp. 1–12.

[15] 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.

[16] M. Khan and M. Herbordt, "Communication requirements for FPGA-centric molecular dynamics," in *Symposium on Application Accelerators for High Performance Computing*, 2012.

[17] S. Moore, P. Fox, A. Markettos, and A. Majumdar, "Bluehive–A Field Programmable Custom Computing Machine for Extreme-Scale Real-Time Neural Network Simulation," in *Proc. IEEE Symp. on Field Programmable Custom Computing Machines*, 2012.

[18] P. Swarztrauber, "Transposing arrays on multicomputers using de Bruijn sequences," *J. of Parallel and Distributed Computing*, vol. 53, pp. 63–77, 1998.