FPGA-Accelerated Particle-Grid Mapping

Ahmed Sanaullah Arash Khoshparvar Martin C. Herbordt
Department of Electrical and Computer Engineering; Boston University, Boston, MA

***Abstract*—Computing the forces derived from long-range electrostatics is a critical application and also a central part of Molecular Dynamics. Part of that computation, the transformation of a charge grid to a potential grid via a 3D FFT, has received some attention recently and has been found to work extremely well on FPGAs. Here we report on the rest of the computation, which consists of two mappings: charges onto a grid and a potential grid onto the particles. These mappings are interesting in their own right as they are far more compute intensive than the FFTs; each is typically done using tricubic interpolation. We believe that these mappings have been studied only once previously for FPGAs and then found to be exorbitantly expensive; i.e., only bicubic would fit on the chip. In the current work we find that, when using the Altera Arria 10, not only do both mappings fit, but also an appropriately sized 3D FFT. This enables the building of a balanced accelerator for the entire long-range electrostatics computation on a single FPGA. This design scales directly to FPGA clusters. Other contributions include a new mapping scheme based on table lookup and a measure of the utility of the floating point support of the Arria-10.**

***Keywords*-FPGA Acceleration; Tricubic Interpolation; Electrostatics; Molecular Dynamics**

I. Introduction

Computing forces derived from long-range electrostatics is a critical application in and of itself and also a central part of Molecular Dynamics (MD) simulations. Figure 1 gives an overview of the family of methods now most commonly used; they are based on Particle-Mesh Ewald summation [1]. After the initial partition into short and long-range components, it consists of three parts: (i) mapping, or discretizing, the charges onto a charge grid (also called charge assignment), (ii) computing the potential grid by solving Poisson's equation, e.g., using 3D FFTs or Multigrid, and (iii) computing the forces by applying the potential grid to each particle. The second part of that computation, the transformation of a charge grid to a potential grid via 3D FFTs, has received some attention recently and has been found to work extremely well on FPGAs [2]–[7]; speed-ups comparable to ASIC cluster implementations have been achieved. However, the grid-particle and particle-grid mappings are crucial in their own right as they are (typically) substantially more compute-intensive than the FFTs. Each is done using tricubic interpolation and requires on the order of 500 floating point operations per particle as opposed to around 100 for each FFT. We believe that these mappings have been studied only once previously for FPGAs and then found to be exorbitantly expensive; in 2007 only a small fraction (bicubic) would fit on an FPGA [8].

We revisit this problem using new FPGA technologies and a new algorithm. The contributions are as follows.

This work was supported in part by the National Science Foundation through Award #CNS-1405695. Email: (arashkh|sanaulah|herbordt)@bu.edu

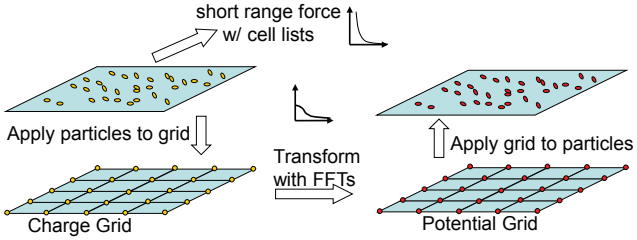

Fig. 1. Overview of the long-range electrostatics computation.

- We demonstrate that, given current FPGA technology, both mappings now easily fit in a high-end FPGA. Moreover, there is sufficient space left over for the 3D FFTs to be implemented on the same FPGA as well.
- We show that FPGAs are particularly well-suited to grid mapping computations. Not only is it floating point intensive, but it also has high fan-in/fan-out and, most significantly, a complex communication structure. The last two of these are challenging for CPUs and GPUs, but can be ideal for FPGAs. For the 92K ApoA1 benchmark, the two mappings together can be computed in less than 1ms on an Altera Arria-10; this likely to be more than 20× faster than a fully parallelized implementation on a high-end CPU.
- We find that the floating point support on the Arria-10 is of significant benefit in resource utilization, particularly with respect to the ALMs.
- We implement two different algorithms: direct, which had been done previously, and a new scheme based on table lookup. These are each fully pipelined and have similar operating frequency; they differ, however, in resource utilization with the first using more DSP blocks and the second more BRAMs. Together they give the designer flexibility with the preference depending on the resource requirements of the other functions being implemented.

The broader significance is that these results are a critical step in demonstrating the plausibility of building FPGA clusters for Molecular Dynamics that can achieve performance comparable to that of ASIC-based clusters.

II. Background

The Coulomb force is computed by generating the potential distribution by solving the Poisson Equation for the given charge distribution. The electrostatic potential can be expressed as

$$V_i^{CL} = \sum_{j \neq i} \frac{q_j}{|r_{ji}|} \tag{1}$$

After partitioning (not shown), grid-based algorithms map the smoothing function defined in the continuous coordinate space

to the one defined in the discrete grid coordinate space. This can be described by the following energy equation:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} q_i q_j g_a(|\vec{r}_j - \vec{r}_i|) = \sum_{k}\sum_{m} q_{h,m} q_{h,n} g_a(|\vec{r}_{h,k} - \vec{r}_{h,m}|) \tag{2}$$

where q_i and q_j are particle charges and $q_{h,m}$ and $q_{h,n}$ are charges at points m and n on a grid with spacing h.

Particle-grid interpolations are used to interpolate both charge densities at gridpoints using particle charge values and electric field vectors at particle positions using gridpoint potential values. The method used here applies a basis function for charge density calculation and the gradient of the same basis function for electric field vector calculation. We use Eq. 3 to calculate charge densities at gridpoints ρ_g from charge positions and Eq. 4 to calculate force vectors at particle positions from potentials φ_g at gridpoints. Indices *p* and *g* denote particle and gridpoint, respectively.

$$\rho_g = \sum_{p} Q_p \phi(|x_g - x_p|)\phi(|y_g - y_p|)\phi(|z_g - z_p|) \tag{3}$$

$$\vec{F}_{p,x} = \sum_{g} \varphi_g \partial\phi(|x_p - x_g|)\phi(|y_p - y_g|)\phi(|z_p - z_g|)\hat{i} \tag{4}$$

The basis function must be C^1 continuous to provide a gradient for these operations. Two such basis functions, a third order and a fifth order, have been proposed in [9]. Eq. 5 shows the third order basis function for spline interpolation where ξ is the distance between the particle and any gridpoint. As seen the basis function and its derivative may have non zero values just for the four closest gridpoints, mimicking third order polynomial interpolation. Thus, each particle is going to affect and be affected by the closest *k+1* gridpoints, *k* being the basis function order.

$$\phi(\xi) = \begin{cases} (1-|\xi|)(1+|\xi|-\frac{3}{2}\xi^2) & |\xi| \le 1 \\ -\frac{1}{2}(|\xi|-1)(2-|\xi|)^2 & 1 \le |\xi| \le 2 \\ 0 & 2 \le |\xi| \end{cases} \tag{5}$$

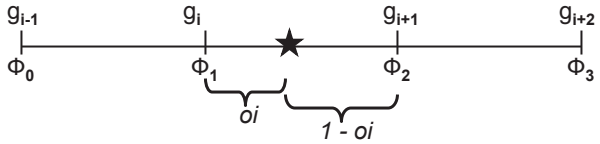

Fig. 2. Extract grid index and offset.

Now following [8] we modify the basis function to be a set of polynomials of oi for the particle's neighboring supporting grid points. By substituting ξ with $oi+1$, oi, $1-oi$, and $2-oi$, we have four polynomials corresponding to the four neighboring grid points (as shown in Figure 2).

$$\begin{cases} \phi_0(oi) = -\frac{1}{2}oi^3 + oi^2 - \frac{1}{2}oi \\ \phi_1(oi) = \frac{3}{2}oi^3 + \frac{5}{2}oi^2 + 1 \\ \phi_2(oi) = -\frac{3}{2}oi^3 + 2oi^2 + \frac{1}{2}oi \\ \phi_3(oi) = \frac{1}{2}oi^3 - oi^2 \end{cases} \tag{6}$$

III. Design

A. Coordinates and indexing

The inputs are scaled particle coordinates and charge values. Overall, except for coordinates, 32-bit single precision floating point is used. For coordinates we use fixed point with user-selectable precision, e.g., 16-bit with 5 bit integer and 11 bit fraction. The precision can be selected match the coordinates and resolution to ensure little or no loss of information. In particular, we use the integer to index the cell within the simulation space and the fraction to determine the position of the particle within the cell. For example, a typical MD simulation might have a 10^3 array of cells each with dimension $12\mathring{A}^3$ and resolution to the thousanth $\mathring{A}$. We would then need 4 bits of integer and 14 bits of fraction. Another design choice is shown in Figure 3; a mesh size of 32 on each dimension determines the nearest gridpoint, G_i, in each dimension.

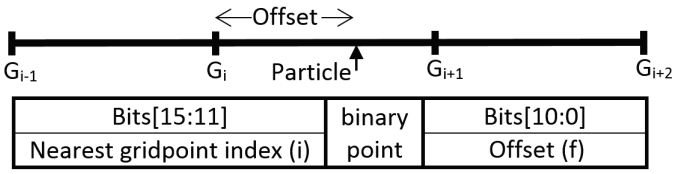

Fig. 3. Zero Cost Shifts to Obtain Nearest Gridpoint and Offset; $\xi \in \{1+f, f, 1-f, 2-f\}$ in the basis functions and their derivatives

The advantage is that only the precision needed, i.e., to specify the position within the cell, is actually used. This has multiple benefits. First, less storage and smaller arithmetic units are needed for the initial calculations (before conversion to floating point). Second, this will enable us to implement a pipeline all in fixed point (work in progress). And third, the modest precision (up to 14 bits) allows us to precompute some values and use the mantissa as an index into a lookup table. In particular, a k^{th} order basis function has non-zero values just on the *k+1* nearest gridpoints, so by precomputing and storing the *k+1* possibly non-zero basis function values in a look-up table, the fraction can be used as the offset to that look-up table for obtaining basis function values. This fact is also shown for the third order basis function in Figure 3.

B. Charge density computation

Each particle contributes charge to its four closest gridpoints in each dimension, i.e., to a cube of 64 gridpoints. Hence, the contribution of each particle to each of these 64 gridpoints needs to be computed, accumulated with the current charge density values, and written back to their respective locations. Since FPGAs have both a customizable interconnect and distributed memory, we implement an interleaved memory access module [10]. This enables, per particle, all reads, writes, and data routes to be fully pipelined and operate concurrently. As the rest of the arithmetic is also fully pipelined, using hard FP cores in the case of the Arria-10, the whole process, from reading particle data through updating gridpoint charge density, has a throughput of one particle per clock cycle.

The classic accumulation problem results from multicycle pipelined arithmetic using an accumulator as one of the inputs and the other input arriving at a rate faster than the latency of the arithmetic unit. This can be solved in many ways; here we schedule the particles and grid points so that the same accumulator (grid point or particle) is only used at a frequency lower than pipeline latency. We have also implemented a stall mechanism. This inhibits particle fetch if the particle about to enter a gridpoint clashes with any of the particles currently in the pipeline. Figure 4 shows how this has been implemented. This solution works with high efficiency (few stalls) as long as the FPGA is responsible for more than a few thousand particles. If fewer, then other solutions, e.g., pipelined reduction, are preferred and can easily be integrated.

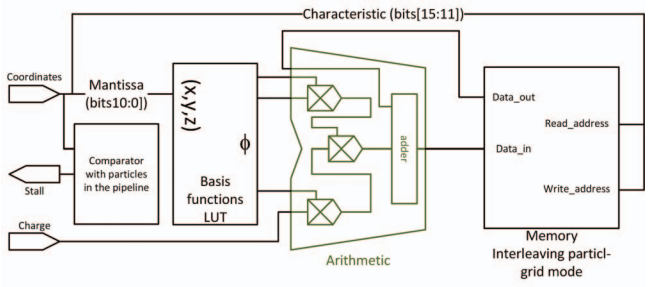

Fig. 4. Particle-Grid Interpolation Design; Hard FP Cores Shown in Green

The grid-to-particle module is basically the same as the particle-to-grid module, except that, in order to calculate the force at any particle's position in any direction, the derivative of the basis function in that direction is used. There must therefore be three copies of these modules to facilitate a throughput of one particle per cycle.

C. Memory Interleaving

BRAM access through memory interleaving is the core of the design; the fact that this easily fits on contemporary FPGAs makes them a suitable option for high-performance electrostatics computation. Such a design was not possible until recently due to the resource-intensive nature of the soft floating point cores.

The memory interleaving module consists of 64 true dual port RAMs. Each cube depicted in Figure 5 represents one such RAM. For a 32×32×32 grid, each RAM needs to have 512 32-bit words to store single precision floating point results. There are two modes of operation for the memory interleaving module, particle-grid access mode for charge density and force computation, as well as FFT access mode (not described here).

In particle-grid mode, multiplexers are placed in the design to organize the 64 data ports with respect to the LSBs of the nearest gridpoint address, e.g., the red cell in Figure 5. An address translation algorithm calculates the correct location, e.g., {-1,same,+1}, on each of the 64 RAMs with respect to the LSBs of the nearest gridpoint address. The number of LSBs used depends on the order of the interpolation basis function. Fig. 5 is drawn for a third order basis function, which accesses four gridpoints resulting in a 4×4×4 cube. A detailed description of this schema can be found in [10].

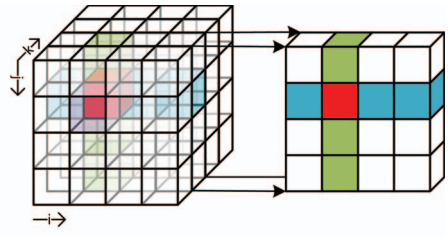

Fig. 5. Memory Interleaving Particle-Grid Access Schema

IV. Implementation and Results

A. FPGA resources and timing

We have implemented two designs, the one described above that uses a lookup table (LUT) and one that does not and computes all results directly (Direct). We test two different FPGAs: a high-end Altera Arria-10 (10AX115R4F40I3SG) and a high-end Altera Stratix-V (5SGSMD8). For the Stratix-V we use a Gidel ProcE-V card; the Arria-10 results are post place-and-route. Resource and timing results to date are shown in Table I. For the LUT algorithm we present results for n=14, which matches the precision of the reference code, and n=11, which uses substantially less memory.

Interpolation has been implemented using two stand-alone multipliers and one integrated adder-multiplier resulting in three DSP units. For a tricubic interpolation four points in each of the three dimensions results in a total 192 DSP units. There is one such module for particle-grid and three for grid-to-particle resulting in a total of 768 DSP units. There are approximately 6MB of memory on the device. Each Particle-Grid module takes 640KB (for n=14) which makes a total of 2.56MB for all four.

This leaves plenty of resources left over to store the particle positions and implement a 3D FFT and complete the electrostatics computation. For example, 32 32-bit FFT modules (sufficient for a 100K particle simulation), each occupying 26 DSP units, result in 832 DSP units. Memory usage by the FFTs is negligible, so a total of 5MB can be used for particle data storage, enough for more than 200K particles on chip. This level of FFT support can execute a 32^3 FFT in roughly 20μs.

B. Results quality

Determining whether MD accelerator designs produce results of sufficient quality is a difficult and much debated issue. It likely sufficient, however, to use the same precision and data types as are used in the reference design. But since FPGAs (and ASICs) lend themselves to flexible designs that trade off precision for compute resources, it is also useful to see the effects of reducing that precision. In the case of the LUT design, the impact on resource utilization is substantial: each bit removed from the fraction halves the size of the lookup table. In Table II we show the error on the gridpoint calculation resulting from the use of n-bit fixed point versus single precision floating point. We note that, as expected, the error is similar to the round off in the last bit. When $n = 14$ the error is on the order of 1 in 10,000, which matches the precision of the distance measurements in the simulation. In general, the final precision determination must

TABLE I
IMPLEMENTATION STATISTICS, POST-PLACE-AND-ROUTE, FOR CHARGE ASSIGNMENT.

Architecture	Logic Utilization	Registers	Memory Block Bytes	DSP Blocks	Maximum Frequency
Arria-10 (LUT, n=11)	13,246 (3%)	202	192KB (3%)	192 (13%)	269.03MHz
Stratix-5 (LUT, n=11)	50,704 (19%)	44,180	192KB (3%)	192 (10%)	252.59MHz
Arria-10 (LUT, n=14)	13,246 (3%)	202	640KB (10%)	192 (13%)	269.03MHz
Stratix-5 (LUT, n=14)	50,704 (19%)	44,180	640KB (10%)	192 (10%)	252.59MHz
Arria-10 (Direct)	3,752 (1%)	11,299	128KB (2%)	319 (20%)	570.45MHz
Stratix-5 (Direct)	62,617 (24%)	160,703	128KB (2%)	223 (11%)	450.00MHz
Memory Interleaving	1,451 (1%)	3788	128KB (3%)	64 (4%)	513.35MHz

follow full integration into an MD code and simulations over large numbers of timesteps to verify stability of conservative physical measures. For direct comparison we have instrumented the ProtoMol MD code [11]. We find for 16-bit Fixed Point an RMS Deviation of 7.45e-5; for single precision FP the RMS Deviation is 5.84e-5.

TABLE II
ERROR IMPOSED ON GRIDPOINTS DUE TO THE USE OF N-BIT FIXED POINT VERSUS FLOATING POINT COORDINATES.

	Max Fractional Error	RMS Deviation RMS Deviation	Min Fractional Error
11-bit	7.294e-4	2.041e-4	3.076e-7
12-bit	3.745e-4	1.018e-4	1.051e-7
13-bit	1.824e-4	5.101e-5	1.128e-7
14-bit	9.264e-5	2.546e-5	1.299e-8

C. Performance

For CPU performance comparison we use two production MD codes, ProtoMol [11] and NAMD [12]. For the latter two programs we obtained timing by instrumenting the relevant sections. In both cases we ran on a single core of an Intel Xeon CPU E3-1226 (Q2 2014, 4 cores) v3 @3.30GHz (see Table III). The CPU throughput results should be multiplied by 10-15 in order to make a chip-to-chip comparison between high-end FPGA with high-end CPU.

TABLE III
THROUGHPUT COMPARISON. PROTOMOL AND NAMD ARE FOR A SINGLE CORE.

MD Code	Throughput (particles/μs)
ProtoMol	0.295
NAMD	2
FPGA (Stratix V)	450

Charge assignment has also been implemented on GPUs (see, e.g., [13]), but, again, we are not aware of any published results of these functions alone. We have reconstructed the algorithm described in [13]. Using an NVidia Quadro K4200 (Kepler class) with 1344 cores, grid-to-particle takes 13ms and grid-to-particle takes 20ms. This GPU is somewhat smaller than the largest currently available. And again, our implementation is likely less than optimal. We are currently exploring existing GPU production MD codes and will present these results in future work.

V. DISCUSSION AND FUTURE WORK

We have implemented FPGA-accelerated charge assignment and force mapping, which are the remaining pieces to completing FPGA-accelerated electrostatics. We have shown that, in current technology, huge structures such as floating point tricubic interpolation pipelines (and communication support) easily fit on a single FPGA. Moreover, sufficient space remains to implement the rest of the electrostatics computation on the same FPGA.

Also, based on an observation of spatial resolution and partitioning in typical MD simulation, we have created a new lookup-table-based algorithm and found this be advantageous in logic contrained domains.

There are several tasks in progress: finishing the mapping onto an Arria-10, instrumenting production MD reference codes for GPU comparisons, and integrating the mapping functions with the FFT to support full electrostatics.

REFERENCES

[1] 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.

[2] B. Humphries, H. Zhang, J. Sheng, R. Landaverde, and M. Herbordt, "3D FFT on a Single FPGA," in *Proc. IEEE Symp. on Field Programmable Custom Computing Machines*, 2014.

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

[4] A. Lawande, A. George, and H. Lam, "Simulative analysis of a multidimensional torus-based reconfigurable cluster for molecular dynamics," in *Proc. Work. Het. and Unconventional Cluster Architectures and Apps.*, 2014.

[5] S. Lee, "An FPGA Implementation of the Smooth Particle Mesh Ewald Reciprocal Sum Compute Engine," Master's thesis, U. Toronto, 2005.

[6] J. Sheng, B. Humphries, H. Zhang, and M. Herbordt, "Design of 3D FFTs with FPGA Clusters," in *IEEE High Perf. Extreme Computing Conf.*, 2014.

[7] J. Sheng, C. Yang, and M. Herbordt, "Towards Low-Latency Communication on FPGA Clusters with 3D FFT Case Study," in *Proc. Highly Efficient and Reconfigurable Technologies*, 2015.

[8] 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.

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

[10] T. VanCourt and M. Herbordt, "Application-dependent memory interleaving enables high performance in FPGA-based grid computations," in *Proc. IEEE Conf. on Field Programmable Logic and Applications*, 2006, pp. 395–401.

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

[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. Comp. Chemistry*, vol. 26, pp. 1781–1802, 2005.

[13] N. Ganesan, B. Bauer, T. Lucas, S. Patel, and M. Taufer, "Structural, Dynamic, and Electrostatic Properties of Fully Hydrated DMPC Bilayers From Molecular Dynamics Simulations Accelerated with Graphical Processing Units (GPUs)," *J. Comp. Chemistry*, pp. 2958–2973, 2011.