Accelerated Particle-Grid Mapping

Ahmed Sanaullah **Martin C. Herbordt (Advisor)**

Electrical and Computer Engineering Department, Boston University, MA, USA

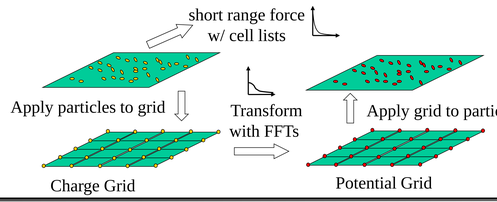
ABSTRACT

Charge Mapping is critical to electrostatic computations for Molecular Dynamics. It reduces the complexity of evaluating long-range coulombic forces by diffusing discrete particle charges to a regular grid. Efficient charge mapping on accelerators (GPUs, FPGAs) is non-trivial, with the compute and memory intensive nature of the algorithm limiting performance benefits in naive implementations. Typical GPU codes, such as [1]–[4], avoid atomic operations and thread divergence through complex algorithms at the expense of an increase in both memory and compute transactions. On FPGAs, resource constraints have only allowed low order (bicubic) interpolations [5]. In our work, we explore methods for improving the performance on both platforms. These include application specific data structures and low complexity kernels for GPUs and deep pipelines with interleaved memory access for FPGAs. Our best case implementation shows > 62x speed-up over existing CPU codes and >30x speed-up over existing GPU codes. We also find that, when using the Altera Arria 10, high resource availability enables the building of a balanced accelerator for the entire long-range electrostatics computation on a single FPGA.

MOTIVATION

Why Particle-Grid Mapping?

In Particle Mesh Ewald methods, grid-particle and particle-grid mappings are crucial as they typically are substantially more compute intensive than FFTs. For example, the 92224 particle ApoA1 protein simulation typically requires >40M FLOPs to map while a 3D FFT on the resulting 32^3 charge grid takes ≈ 2.5M FLOPs.

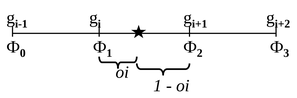


INTERPOLATION OVERVIEW

- Particle-grid distances used to evaluate the third order interpolation basis function (ϕ)
- $$\phi(\xi) = \begin{cases} (1-|\xi|)(1+|\xi|-\frac{3}{2}\xi^2) & |\xi| \le 1 \\ -\frac{1}{2}(|\xi|-1)(2-|\xi|)^2 & 1 \le |\xi| \le 2 \\ 0 & 2 \le |\xi| \end{cases}$$
- Grid charge density = $\rho_g = \sum_P Q_P \phi(|x_g - x_P|)\phi(|y_g - y_P|)\phi(|z_g - z_P|)$
- Force = $\boldsymbol{F}_{p,x} = \sum_P \varphi_g \partial\phi(|x_P - x_g|)\phi(|y_P - y_g|)\phi(|z_P - z_g|)\,\mathbf{i}$
- Each grid point is affected by the closest *k+1* points, *k* being the basis function order
- The basis function is C^1 continuous to provide a gradient for these operation

- The basis function is modified to be a set of polynomials of ***oi*** for the particle's neighboring supporting grid points
  - Substitute ξ with $oi + 1, oi, 1 - oi$ and $2 - oi$

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



PLATFORMS

CPU

Intel Core i5-4570 • 2.9 GHz

GPU

TESLA k40m • Kepler Architecture • 2880 cores • CUDA NVCC 7.5

TESLA M2070 • Fermi Architecture • 440 cores • CUDA NVCC 7.5

FPGA

Altera Arria 10 • 427200 ALMs • 55562240 Memory Bits • 1518 DSP Blocks

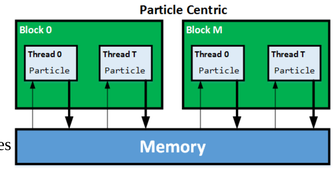
Altera Stratix 5 • 262400 ALMs • 52572160 Memory Bits • 1963 DSP Blocks

SYSTEM DESIGN

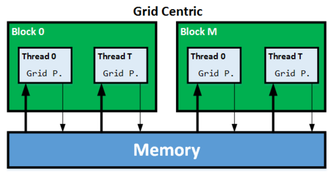
GPU

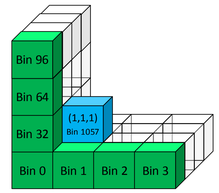
- Particles are sorted into bins corresponding to the grid cell in which they are located
- Grid cells are assigned bin IDs based on their nearest grid point and the system size
  - Figure on the right shows part of a 32x32x32 grid with allocated bin IDs
  - Grid cell corresponding to the coordinates (1,1,1) is assigned a bin ID = 1057
- Binning reduces memory transaction timeframes by improving locality
  - Up to 75% overlap of required data between adjacent bins
  - GPU threads are assigned contiguous bins to perform charge mapping
- Dummy particles are not used for avoiding thread divergence
  - Threads in a warp remain idle after processing the contents of their assigned bin
  - Warps are active for the local maxima of bin size (as opposed to global in [4])
  - Number of memory transactions is significantly reduced

- **Implementation 1:** Particle Centric
  - Threads are assigned a single bin where bin ID = thread ID
    - Compute charge contributions of all particles in the bin
    - Accumulate in global memory using atomic add
  - Advantages
    - Low complexity kernel
    - Low memory read transactions
  - Disadvantage
    - Atomic operations needed due to resource contention during global memory writes



- **Implementation 2:** Grid Centric
  - Threads are assigned a grid point where grid point = thread ID
    - Compute charge contributions from 64 neighboring bins
    - Accumulate in local memory and write back final charge value to global memory
  - Advantage
    - Single write to global memory per thread
    - No atomic operations (and resulting potential stalls)
  - Disadvantage
    - Large number of reads
    - Unbalanced traffic





Optimized Particle Centric Algorithm

```
1:  ID ← getThreadID()
2:  Bin = Mem[ID]
3:  Compute nearest four grid points in each dimension
4:  gz[4] = boundary ? wrap around : gz[4]
5:  gy[4] = boundary ? wrap around : gy[4]
6:  gx[4] = boundary ? wrap around : gx[4]
7:  for irr = 1 to Bin.counter do
8:      Particle = Bin[irr]
9:      phiz[4] = Basis Function(PX.z - gz[2])
10:     phiy[4] = Basis Function(PX.y - gy[2])
11:     phix[4] = Basis Function(PX.x - gx[2])
12:     for z = 1 to 4 do
13:         for y = 1 to 4 do
14:             for x = 1 to 4 do
15:                 AtomicAdd (ChargeGrid(gz[z],gy[y],gx[x]) ,
16:                 phiz[z]*phiy[y]*phix[x]*Q )
17:             end
18:         end
19:     end
20: end
```

Optimized Grid Centric Algorithm

```
1:  Grid Point ← getThreadID()
2:  Compute nearest four grid points in each dimension
3:  gz[4] = boundary ? wrap around : gz[4]
4:  gy[4] = boundary ? wrap around : gy[4]
5:  gx[4] = boundary ? wrap around : gx[4]
6:  for z = 1 to 4 do
7:      for y = 1 to 4 do
8:          for x = 1 to 4 do
9:              RemoteBinID = f(gz[z],gy[y],gx[x])
10:             for irr = 1 to counter[RemoteBinID] do
11:                 BinSlot = h(irr,RemoteBinID)
12:                 Particle = Mem[BinSlot]
13:                 phiz[4]=BasisFunction(PX.z - floor(PX.z))
14:                 phiy[4]=BasisFunction(PX.y - floor(PX.y))
15:                 phix[4]=BasisFunction(PX.x - floor(PX.x))
16:                 LocalVariable+=phiz[z]*phiy[y]*phix[x]*Q
17:             end
18:         end
19:     end
20: end
21: ChargeGrid(getThreadID) = Local Variable
```

FPGA

Our preliminary version of the FPGA implementation [5] addresses the following key design aspects

Coordinates and Indexing

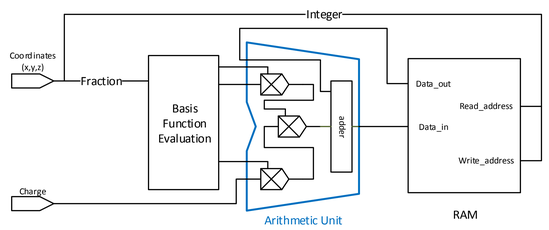
- Inputs are particle coordinates and charge values
  - Fixed point numbers with user selectable precision for coordinates
  - Single precision floating point values for all other values
- For coordinates, integer is used to index cells and fraction to determine the position of the particle within the cell

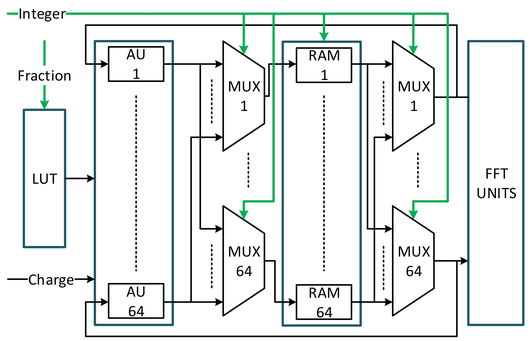
Memory Interleaving

- BRAM access though memory interleaving
  - Fits easily on contemporary FPGAs
  - Resource intensive so only possible on newer FPGA technologies
- Multiplexers used to align the 64 data ports
- Storage location within RAM depend on all integer bits

Charge Density Computation

- Particle contribution is computed, accumulated with current charge density value and written back
- All read, write and data routes (per particle) are fully pipelined and operate concurrently
- Fully pipelined arithmetic using Arria-10 hard FP cores with a throughput of one particle per clock cycle



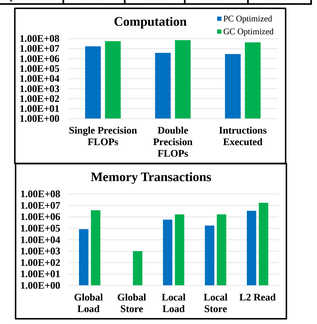


RESULTS

- Performance is evaluated as the throughput achieved i.e. mapped Particles Per µS (PPUS)
- Baseline (without binning) and optimized versions of both GPU implementations are tested on different device and benchmark sizes
- Benchmarks used are the 92K particle ApoA1 and the 68K particle DMPC

		Throughput / PPUS			
		NVIDIA M2070		NVIDIA K40m	
	GPU Version	ApoA1	DMPC	ApoA1	DMPC
Particle Centric	Baseline	5.6	5.1	129.7	124.0
	Optimized	50.9	45.9	198.4	194.0
Grid Centric	Baseline	2.4	2.0	3.5	3.3
	Optimized	15.2	14.1	26.6	28.2

- The table (right) presents the performance results of our implementations.
- Optimized Particle Centric gives the highest throughput in all cases despite having ≈400K atomic transactions
  - Hardware support for floating point atomic operations
  - Memory access patterns, as shown in [4], result in no intra-warp resource contention for writes
  - Latency of atomic write transactions masked by the computations for subsequent particles in a bin
- Both implementations scale with increasing compute resources and better atomic hardware
  - ≈ 2x for Grid Centric and ≈ 4x for Particle Centric
- Both implementations show consistent performance with varying benchmark sizes
- Binning and avoiding dummy particles leads to a significantly better performance in all cases

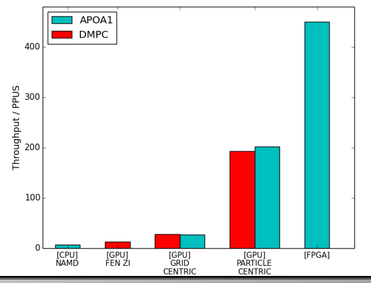


- The graphs (right) illustrate factors leading to higher performance for Particle Centric
  - Orders of magnitude fewer local and global memory transactions
  - Small kernel size (fewer FLOPs and instructions executed)

- FPGA implementation statistics, post place-and-route, for charge assignment using LUTs and FP arithmetic (Direct)
- Stratix-5 (Direct) architecture implemented in hardware with 450 PPUS throughput using the ApoA1 benchmark

Architecture	Logic Utilization	Registers	Memory Block Bytes	DSP Blocks	Maximum Frequency
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 (2%)	64 (4%)	513.35MHz

- Best versions of each implementation are compared with existing charge mapping codes NAMD [1] and FENZI [2] (run on TESLA k40m)
  - All optimizations achieved significantly better performance
- FPGA gives largest throughput due to deep pipelines
- Grid centric implementations perform worse than all other optimizations
  - But still better than existing codes



FUTURE WORK

- Several tasks in progress
  - Finishing mapping onto an Arria 10
  - Integrating the mapping functions with the FFT to support full electrostatics
  - Instrumenting existing CPU and GPU MD codes and porting the ApoA1 and DMPC benchmarks for these programs

REFERENCES

[1] Phillips et al, "Scalable molecular dynamics with namd," Journal of computational chemistry, vol. 26, no. 16, pp. 1781–1802, 2005.
[2] Ganesan et al, "Fenzi: Gpu-enabled molecular dynamics simulations of large membrane regions based on the charmm force field and pme," in Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on. IEEE, 2011, pp. 472–480.
[3] G. Stantchev, W. Dorland, and N. Gumerov, "Fast parallel particle-to-grid interpolation for plasma pic simulations on the gpu," Journal of Parallel and Distributed Computing, vol. 68, no. 10, pp. 1339–1349, 2008.
[4] F. B¨uy¨ukkec¸eci, O. Awile, and I. F. Sbalzarini, "A portable opencl implementation of generic particle–mesh and mesh–particle interpolation in 2d and 3d," Parallel Computing, vol. 39, no. 2, pp. 94–111, 2013.
[5] Sanaullah et al, "FPGA-Accelerated Particle-Grid Mapping," in IEEE Symposium on Field Programmable Custom Computing Machines (FCCM), 2016