

# Particle Mesh Ewald for Molecular Dynamics in OpenCL on an FPGA Cluster

Lawrence C. Stewart  
*Silicon Therapeutics*  
 451 D St, Suite 205  
 Boston, MA, USA  
 larry.stewart@silicontx.com

Carlo Pascoe  
*Silicon Therapeutics*  
 451 D St, Suite 205  
 Boston, MA, USA  
 carlo.pascoe@silicontx.com

Emery Davis  
*Silicon Therapeutics*  
 451 D St, Suite 205  
 Boston, MA, USA  
 emery.davis@silicontx.com

Brian W. Sherman  
*Silicon Therapeutics*  
 451 D St, Suite 205  
 Boston, MA, USA  
 woody@silicontx.com

Martin Herbordt  
*College of Engineering*  
*Boston University*  
 Boston MA, USA  
 herbordt@bu.edu

Vipin Sachdeva  
*Silicon Therapeutics*  
 451 D St, Suite 205  
 Boston MA, USA  
 vipin@silicontx.com

**Abstract**—Molecular Dynamics (MD) simulations play a central role in physics-driven drug discovery. MD applications often use the Particle Mesh Ewald (PME) algorithm to accelerate electrostatic force computations, but efficient parallelization has proven difficult due to the high communication requirements of distributed 3D FFTs. In this paper, we present the design and implementation of a scalable PME algorithm that runs on a cluster of Intel Stratix 10 FPGAs and can handle FFT sizes appropriate to address real-world drug discovery projects (grids up to  $128^3$ ). To our knowledge, this is the first work to fully integrate all aspects of the PME algorithm (charge spreading, 3D FFT/IFFT, and force interpolation) within a distributed FPGA framework. The design is fully implemented with OpenCL for flexibility and ease of development and uses 100 Gbps links for direct FPGA-to-FPGA communications without the need for host interaction. We present experimental data up to 4 FPGAs (e.g., 206 microseconds per timestep for a 65536 atom simulation and  $64^3$  3D FFT), outperforming GPUs for smaller FFT sizes. Additionally, we discuss design scalability on clusters with differing topologies up to 64 FPGAs (with expected performance far greater than all known GPU implementations) and integration with other hardware components to form a complete molecular dynamics application. We predict best-case performance of 6.6 microseconds per timestep on 64 FPGAs.

**Index Terms**—FPGA, Molecular Dynamics, HPC, Reconfigurable Computing, FFT

## I. INTRODUCTION

Molecular dynamics (MD) simulation engines such as AMBER [1] and OpenMM [2] provide high performance implementations for CPU and GPU, and provide a flexible framework in which new computational technologies can be assessed. FPGA implementations have also been explored for many years [3]–[6] including a recent study showing promising single-FPGA performance [7].

MD plays a critical role in computational chemistry in general and in drug discovery in particular. There, long timescales

and small problem sizes lead to tremendous challenges in strong scaling, especially in the electrostatics computation. This has led to the creation of ASIC-based solutions [8]–[10]; their limitations, however, due to cost and availability, make COTS alternatives essential. Of these, only FPGA clusters have shown potential past a small number of nodes [11], [12]. These preliminary studies, however, were based only on the parallelization of the 3D FFT and not full electrostatics, much less complete system integration.

The electrostatic force computation often uses Ewald summation to split the work into short-range and long-range terms. Methods for the long-range term include k-space summation [13],  $\mu$ -series [14], and use of Fourier Transforms to solve Poisson’s Equation [15]. The FFT methods reduce computation from  $O(N^2)$  to  $O(N \log N)$ , but require global communication and strong scaling of 3D FFTs in the size range from  $32^3$  to  $128^3$  as well as a complex mapping function [16].

In this paper we describe the architecture, implementation, and evaluation of a distributed 3D FFT-based Smooth Particle Mesh Ewald electrostatic force computation. Our implementation outperforms GPUs in the problem size ranges of interest for drug discovery, and is scalable to multiple pipelines on multiple boards. It is implemented entirely in OpenCL. There are a number of contributions.

- This is the first parallel complete FPGA electrostatics.
- This work provides a still-rare case study of a production HPC application successfully implemented in OpenCL and distributed across a parallel cluster of tightly coupled FPGAs.
- To our knowledge, this effort is the first to obtain strong scaling for MD problems by using off the shelf hardware. The system is integrated into a complete MD application and results validated against OpenMM over millions of

timesteps.

The potential impact is as follows. Many desirable drug targets in cancer, auto-immune, neurodegenerative, and infectious diseases are currently considered undruggable due lack of binding predictions (binding free energies, conformational changes) [17] that could be provided with long timescale MD [18]. The current work will enable these timescales to be achieved an order of magnitude faster.

The outline of this paper is as follows: Section II discusses background and related work on MD and FFT targeting contemporary architectures including CPUs, GPUs, and ASICs. Section III details the system architecture of our long-range pipeline and 3D FFT, and Section IV follows up with the implementation details. Section V summarizes the results of our work, along with performance comparison to other hardware. Finally, Section VI concludes the paper and describes our plans for future work.

## II. BACKGROUND AND RELATED WORK

Molecular Dynamics models the behavior of atoms and molecules by individually calculating the various forces that act on them. Forces that apply to bonded atoms include bond torsions and tensions. Forces that apply to non-bonded atoms include short-range forces that include both van der Waals and electrostatics, and long-range forces, which are mainly electrostatic. Short-range forces are managed by pairwise computations. For long-range forces, applications instead use multipole approximations [19] or Ewald summations. Our focus is on an Ewald variation known as Smooth Particle Mesh Ewald (SPME) [20]. SPME calculates a charge distribution on a grid, then uses Fourier Transforms and a Green's function to calculate a potential field. Potential gradients then are used to calculate forces.

Molecular dynamics simulations have proven to be a valuable tool in drug discovery for understanding protein motion. Open-source GPU accelerated molecular dynamics applications such as GROMACS [21], NAMD [22], OpenMM [2], and CP2K [23] allow many practitioners to use MD simulations as a regular tool. To our knowledge, the only study showing strong scaling on multiple GPUs for a 100,000 atom system is with the recently redeveloped GROMACS package [24], [25], which does not distribute the FFT.

Several efforts to develop, at great expense, custom ASICs for small molecule simulations have been undertaken. The earliest initiative is the MDGRAPE [8] series of supercomputers. Another well known ASIC initiative is the Anton series [9] developed by D. E. Shaw Research. Anton 1 was released in 2007, with performance for a 23,000 atom system close to 17 microseconds/day. Anton 2, released in 2014, increased this performance five-fold to 85 microseconds/day [10].

The most challenging part of scaling molecular dynamics simulations is the electrostatic forces computation, of which FFT is often a major component. Anton 1 could solve FFT problems of size  $32^3$  in 3.7 microseconds, and  $64^3$  in 13.3 microseconds on 512 nodes. Anton 2 did not use FFT in its

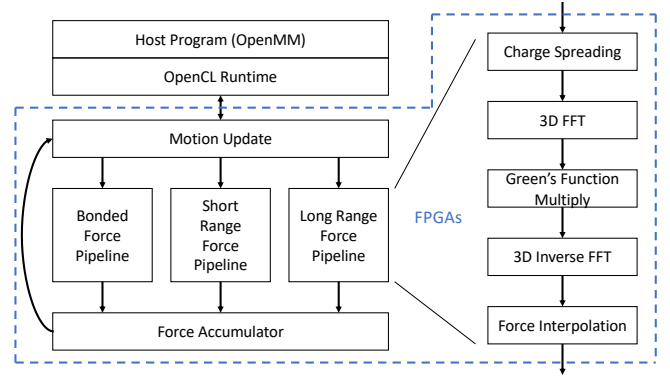


Fig. 1. Molecular dynamics application overview.

simulations, instead relying on a different decomposition called the  $\mu$ -series [10], [14].

Efforts to get parts of molecular dynamics simulations running on FPGAs have been explored over the past few years [26], [27]. More recently, the increase in FPGA resources such as logic elements, DSPs, BRAM, etc., have allowed full MD simulations to run on a single FPGA [7].

A great strength of FPGAs is the I/O transceivers, which are capable of providing a great deal of bandwidth with very low latency [28]. Some clusters with highly interconnected FPGAs are the Novo-G# built at the University of Florida in a 3D torus interconnect [29] and the first version of the Microsoft Catapult [30]. More recently, University of Paderborn has developed Noctua [31] and Tsukuba University has deployed Cygnus, a hybrid GPU-FPGA system [32]. FPGA communications can also now be programmed using OpenCL, providing both high-performance and a productive development environment for distributed applications. Prior work on 3D FFTs on single FPGAs includes [23], [33]–[36] while work on multiple FPGAs includes [11], [12]. Design of FFT for MD simulations is presented in [37]. The earliest 2D floating point FFT on multiple FPGAs of which we are aware is [38].

## III. SYSTEM ARCHITECTURE

Figure 1 shows the OpenCL portions of our modified OpenMM application and the role played by 3D FFT. Our goal is to run multiple timesteps of the full MD application on a network of FPGAs without any additional host communication beyond initialization and result collection. The focus of this paper is the long range portion of the system.

### A. Long-Range Pipeline

The architecture of the long-range force pipeline is shown in Figure 2. Each timestep, the LR pipeline accepts atom positions and charges as input, and delivers long-range electrostatic force updates per atom back to the force accumulator and motion update portions of the system.

1) *Charge Spreading*: Charge Spreading accepts atom positions and charges and constructs the FFT input volume. Any particular atom has a fractional position between grid points. The charge attached to the atom is spread to grid points

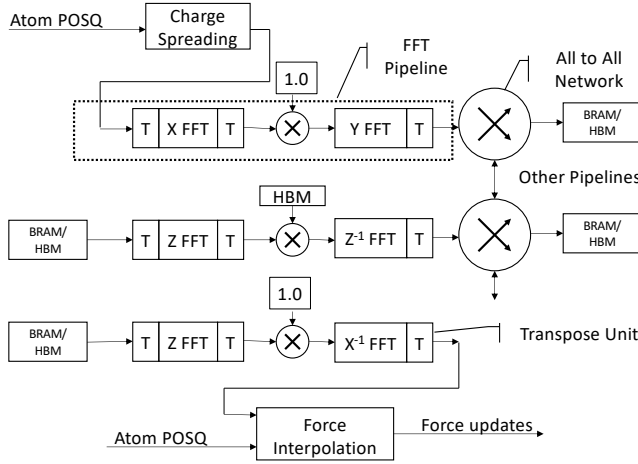


Fig. 2. Logical view of long-range force pipeline.

in the  $4 \times 4 \times 4$  cell surrounding the atom's position by using cardinal B-splines as interpolating functions. We use 64-way parallel hardware to achieve a single pipeline charge spreading throughput of one atom per clock cycle. The computation is further parallelized by multiple pipelines and multiple FPGAs. Because atom volumes of influence overlap, we use additional hardware to dynamically reorder atoms to avoid pipeline hazards. Atoms which affect multiple pipelines are processed by each affected pipe.

2) *FFT*: The 3D FFT is described in more detail in section IV. Our implementation divides the input volume into slabs in the Z dimension. Each slab runs independently to compute X and Y transforms, then an all-to-all network exchanges data – called corner-turning – so that the slabs subdivide the Y dimension. Parallel pipelines then compute Z direction transforms, multiply by Green's function data, and compute  $Z^{-1}$  transforms. A second corner turn feeds  $Y^{-1}$  and  $X^{-1}$  transforms.

3) *Force Interpolation*: The force interpolation unit shares the 64-way parallel arithmetic and BRAM design of the charge spreading hardware. It accepts data from the output of the inverse FFT, and atom position and charge data. 3D cardinal B-splines and their derivatives are used to calculate X, Y, and Z forces for each atom based on gradients of the electrostatic potential field calculated by the FFT. 64-way combining trees calculate force updates which are fed back to the force accumulation block of the main MD application. We plan to share hardware between charge spreading and force interpolation, but this is not yet done.

### B. Scaling 3D FFT

The three dimensional FFT of an XYZ volume can be computed as a sequence of one dimensional transforms [39], [40]. There are alternative parallel 3D FFT formulations such as 2D decomposition [41] and the generalized vector radix decomposition [42] but for up to 64 nodes and  $128^3$  our initial focus is on the simple decomposition.

3D FFT can be scaled by using pipelined parallel hardware, and then by using multiple parallel compute units, provided that the necessary operands can be routed to the compute units.

In 2013, Garrido et al. showed how to build pipelined parallel FFT hardware using a feedforward architecture that is well suited to FPGA implementations [43]. A single precision complex 8-wide FFT unit uses 4% of a Stratix 10 device depending on transform size and consumes and delivers about 19 GB/sec of data.

This hardware can complete 1D FFTs in the number of clock cycles it takes to read the data. Using a nominal 300 MHz design speed, Table I shows the time in microseconds to complete  $N^2$  1D FFTs for various size transforms, given different numbers of 8-wide vector compute units. To complete a full forward and inverse 3D FFT will take six times as much work but can be both pipelined and parallelized.

In order to distribute such a system over a network of FPGAs, it is necessary to balance communications and computation performance and it is also necessary to choose points in the solution spaces for FFT and for All to All in which the bandwidths match.

The cells in Table I with parenthesized references represent particular solution choices that match well with potential communications designs, which are discussed in section III-C.

### C. Scaling All to All

In this section we analyze potential network topologies to evaluate points in the solution space that are compatible with distributed FFT designs. The all to all network is responsible for interchange of data among multiple processing pipelines, both when colocated on a single board and when distributed across multiple FPGAs.

In any parallelization into N units,  $(N - 1)/N$  of the data must move. Since FFT and all to all are pipelined, the overall performance will be set by the slower function.

Table II relates numbers of FPGA modules, network topology, and the time to complete the All to All. The environment for this analysis consists of a number of FPGA nodes, each equipped with either four or six links running at 100 Gbps. Configurations marked “Switched” use Ethernet packet framing to route messages via 100 Gbps Ethernet switches to achieve single hop connections. Each board has physical interfaces for six links, with four enabled on our test platforms. PtoP configurations use point to point cables, other topologies require on-FPGA switches and higher hopcounts. As an example, a 4 dimensional hypercube for 16 nodes has an average hopcount of 2, because on average a destination node ID differs in only two bits from the source node ID.

The time to complete figures are given by

$$D * \frac{N - 1}{N} * \frac{H}{B * N * L}$$

where D is the FFT data volume in bits.  $(N - 1)/N$  is the fraction of data that must be sent to a different board. H is the average hop count, B is the link bandwidth in bits per second,

TABLE I  
TIME FOR FFT SIZES VS NUMBER OF UNITS, AT 300 MHz ( $\mu$ s)

XYZ	Data Points	Data Bits	1	2	4	8	16	32	64	128
32x32x32	32,768	2,097,152	13.7	6.8	3.4	1.7	0.9	0.4	0.2	0.1
64x64x64	262,144	16,777,216	109.2	54.6	27.3	13.7	6.8	3.4	1.7	0.9
64x64x128	524,288	33,554,432	218.5	109.2	54.6	27.3	13.7	6.8	3.4	1.7
96x96x96	884,736	56,623,104	368.6	184.3	92.2	46.1	23.0	11.5	5.8	2.9
128x128x128	2,097,152	134,217,728	873.8	436.9	218.5	109.2 (1)	54.6 (2)	27.3 (3)	13.7(4)	6.8 (5)

Data within () reference similar entries in table II

TABLE II  
NETWORK TIMING FOR ALL TO ALL ( $\mu$ s)

Nodes	Topology	Hopcount	Links	32x32x32	64x64x64	64x64x128	96x96x96	128x128x128
2	PTOP	1	4	1.7	13.4	26.9	45.4	107.5 (1)
4	PtoP	1	3	1.7	13.4	26.9	45.4	107.5 (1)
8	2D Torus	1.5	4	1.1	8.8	17.6	29.8	70.6
8	Hypercube	1.5	3	1.5	11.8	23.5	39.7	94.1
8	Hypercube++	1.25	4	0.9	7.4	14.7	24.8	58.8 (2)
8	3D Torus	1.5	3	1.5	11.8	23.5	39.7	47.1
8	Switched	1	4	0.7	5.9	11.8	19.8	47.1
16	2D Torus	2	4	0.8	6.3	12.6	21.3	50.4
16	3D Torus	2	6	0.5	4.2	8.4	14.2	33.6 (3)
16	Hypercube	2	4	0.8	6.3	12.6	21.3	50.4
16	Switched	1	4	0.4	3.2	6.3	10.6	25.2 (3)
32	2D Torus	3	4	0.6	4.9	9.8	16.5	39.1
32	3D Torus	2.5	6	0.3	2.7	5.4	9.2	21.7
32	Hypercube	2.5	5	0.4	3.3	6.5	11.0	26.0
32	Switched	1	4	0.2	1.6	3.3	5.5	13.0 (4)
64	2D Torus	4	4	0.4	3.3	6.6	11.2	26.5
64	3D Torus	3	6	0.2	1.7	3.3	5.6	13.2 (4)
64	Hypercube	3	6	0.2	1.7	3.3	5.6	13.2
64	Switched	1	4	0.1	0.8	1.7	2.8	6.6 (5)

Data within () reference similar entries in table I

L is the number of links per board, and N is the number of boards that partition the problem.

The scaling behavior of this equation is such that the time to completion goes as  $H/N$ , assuming equal link loading, uniform traffic, and perfect link scheduling. All to all certainly has uniform traffic and there is a symmetry argument for uniform link loading.

For a switched network, perfect scheduling of an all to all is straightforward. In round  $i$ , node  $n$  transmits to node  $(n + i) \bmod N$ . Point to point networks are also easy to schedule. As for multihop scheduling, the question is still open but we are experimenting with static scheduling.

For four boards, direct point to point cables suffice, but we must use an on-FPGA router for multihop cases. The prototype router uses a crossbar switch with buffering at each crosspoint, together with a statically compiled switch schedule. This permits the network to operate in a streaming mode without packet headers or frame boundaries. Links from application logic to the router use Intel's OpenCL Channel extension, as do the off-board links themselves.

Because the all to all communication pattern is symmetric, switch scheduling reduces to a bin packing problem of packing message fragments into open channel time slots, while managing the maximum buffer occupancy [44], [45]. There is no danger of livelock or deadlock and no need for traditional techniques such as virtual channels, because all messages are known at compile time. Flow control and error recovery are provided by the vendor Board Support Package (BSP). We expect to report results in future work.

#### D. Balancing Computation vs Communications

Table I and Table II identify consistent points in solution space for 8 through 128 pipelines where computation performance is a good match for communications performance. Our current configuration is (1), eight pipelines, split among four FPGAs using point to point links. Configurations (2), (3), (4), and (5) identify points in solution space for 8, 16, 32, and 64 FPGAs. These solution points permit balanced designs, in which no unit runs faster than necessary.

This analysis is done for 128x128x128 transforms, but similar choices exist for other size FFTs. Since the completion times are entirely bandwidth limited, they scale only with the total data volume and the same points in solution space apply to all sizes. However, for the smaller transforms, hardware unit latencies and communications latencies start to become important.

For  $A$  atoms and 3D FFT size  $N^3$ , the charge spreading step operates in  $O(A)$  time, the 3D FFT is  $O(N^3 \log N)$ , and the force interpolation is again  $O(A)$ . By choosing an FFT volume proportional to the number of atoms (reasonably uniform density), and by using hardware to remove the  $\log N$  term, the entire LR pipeline becomes  $O(A)$ , a dramatic improvement over  $O(A^2)$  pairwise methods of computing electrostatic forces.

## IV. SYSTEM IMPLEMENTATION

### A. FFT

Intel provides an OpenCL computational kernel example using the feedforward parallel FFT of Garrido et al, and we

took it as our starting point. [43], [46] (This is also used in [36]). This design accepts vectors bit-reversed by lane and in order by vector.

The internal structure of the parallel FFT is shown in Figure 3. This example is 8-wide, compiled for a 64 point FFT. The design compiles a variable number of stages, corresponding to the base 2 logarithm of the transform size. As a result, the  $\log N$  term in FFT's  $O(N \log N)$  is subsumed by hardware and the unit runs in  $O(N)$  time.

### B. Bit Dimension Permutations

In order to assemble 1D FFT units into a pipelined 3D FFT the sequencing of the data must be modified in order to deliver to and accept data from the FFT units in the correct order. This problem is referred to as bit dimension permutations [47].

This structure is able to perform nearly arbitrary bit dimension permutations of the input sequence. A standalone python program uses a control file to generate the OpenCL source code necessary to control the hardware. The limitations that exist can be removed by using 5-stage Benes networks [48] at the input and output but we have found the three stage networks suffice for the permutations encountered in the 3D FFT.

### C. All to All

The All to all network is responsible for interchange of data among multiple processing pipelines, both when colocated on a single board and when distributed across multiple FPGAs.

Figure 5 is the design for 8 pipelines distributed across four FPGAs. Each board has a direct connection to each other board. Some connections remain on board, but in the 4 board example they are modelled as a loopback cable that connects a board to itself. In order to use the same bit file on each of the four FPGAs, additional logic in the A2A unit routes these “virtual cables” to the correct external port or internal loopback. This internal crossbar switch is a prototype of the future hardware supporting routed networks.

### D. Using OpenCL for FPGA programming

We chose OpenCL as the programming language for our implementation. This is due to several reasons: OpenCL allows more productive software development, and also allows us to be vendor-agnostic. Using OpenCL on the host side has allowed us to reuse OpenMM’s framework for launching kernels on the FPGA as well. OpenCL compilation for FPGAs

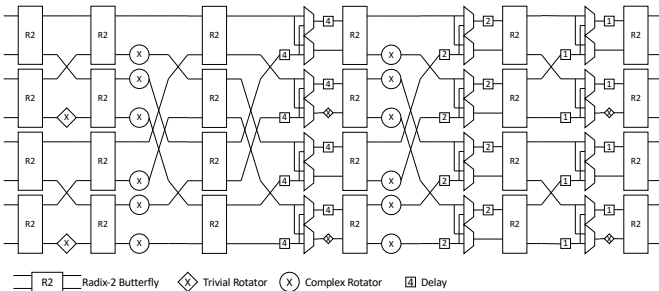


Fig. 3. 8-Wide, single-precision complex FFT compute unit (64 point).

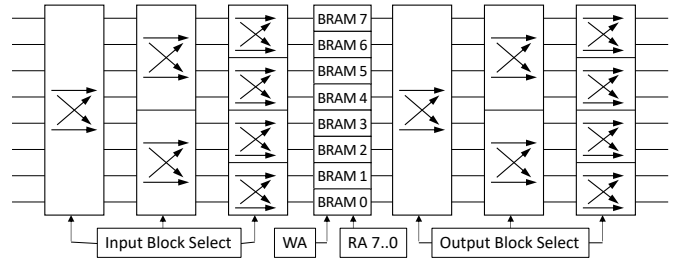


Fig. 4. Transpose Unit.

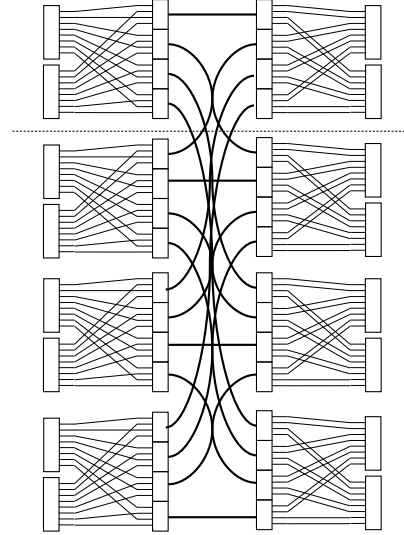


Fig. 5. All to all network. 8 pipelines on four FPGAs, with cables.

transforms high-level source code into a dataflow graph and instantiates the necessary hardware. We approach FPGA coding in OpenCL with a hardware engineer’s perspective. It is possible to visualize the dataflow hardware you want, as in Figure 3 or Figure 4 and then write fairly straightforward code to realize it. It can be complex to achieve the same level of control as with HDL; however, we have largely been able to overcome challenges.

In the future, we may move some parts of our implementation into VHDL or Verilog for optimal resource utilization. OpenCL does permit linking to HDL provided the HDL modules provide certain prescribed interfaces.

## V. EVALUATION

We report here on two implementations. Both are in OpenCL, with no Verilog or VHDL components. The first is a FFT only design, with up to 16 processing pipelines per FPGA. The second is an implementation of Smooth Particle Mesh Ewald which implements the full long-range force pipeline.

### A. Experimental Setup

Our hardware setup comprises 8 BittWare 520N-MX boards on a single hardware node. The hardware node has 2 8-core Intel Xeon Silver CPUs as well as 768 GB of memory (used mostly for OpenCL compilation jobs). The CPUs also serve as host processor for the OpenCL programs and OpenMM. Each

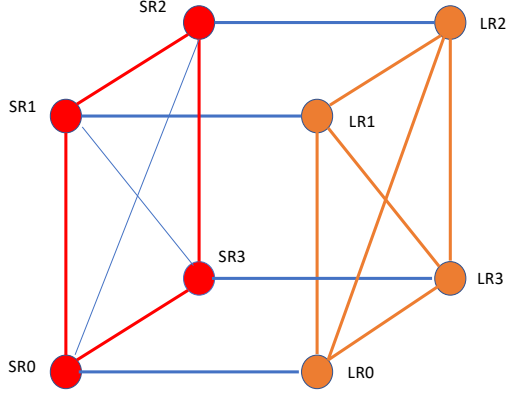


Fig. 6. Eight FPGA Testbed

TABLE III  
BRAM-BASED 3D FFT

FFT Size	No. of Pipes	fMax (MHz)	Time ( $\mu$ s)	Ideal ( $\mu$ s)	BRAM (% usage)	DSP (% usage)
32x32x32	1	290	59	42	2	3
32x32x32	8	243	8.5	6.3	21	18
32x32x32	16	266	3.87	3.27	28	52
64x64x64	16	275	24.5	22.3	49	58

520N-MX has a single Intel Stratix 10 MX2100 FPGA [28] and 4 QSFP28 channels, each capable of communicating at a peak bandwidth of 100 Gb/s. Each FPGA is configured with the p520\_max\_m210h BSP to allow OpenCL as the programming model for FPGA computation as well as communication between the different boards. We are operating with a preliminary BSP which runs the links at 78 Gbps. Our application source code is compiled using Quartus release 20.3. The server runs CentOS 7.6. We use *SLURM* to manage both compilation and hardware resources.

The interconnect topology is shown in Figure 6. The four boards implementing the long-range subsystem, marked LR0 to LR3 are fully connected and implement the full SPME algorithm. Each board has a link to the corresponding short-range board. The short-range boards are connected in a ring and are responsible for short range forces as well as motion update and force accumulation.

### B. FFT Only

We first present results of our BRAM-based FFT. In this version, the entire dataset fits into the BRAM of the FPGA. Results from this design are shown in table III. We present single FPGA results for this design, with up to 16 processing pipelines, for FFT sizes  $32^3$  and  $64^3$ . A BRAM-only  $128^3$  design will not fit on our current hardware available due to BRAM limitations. The  $32^3$  version occupies 28% of the BRAMS and 52% of the DSP blocks and runs in 3.87 microseconds at 266 MHz. The  $64^3$  version occupies 49% of the BRAMS and 58% of the DSP blocks and runs in 24.5 microseconds.

This version illustrates similar performance to Anton 1’s 3.7 microseconds for  $32^3$  transforms, albeit 10 years later. We wryly note that collapsing a system of 512 ASICs into a single FPGA is fully consistent with Moore’s law.

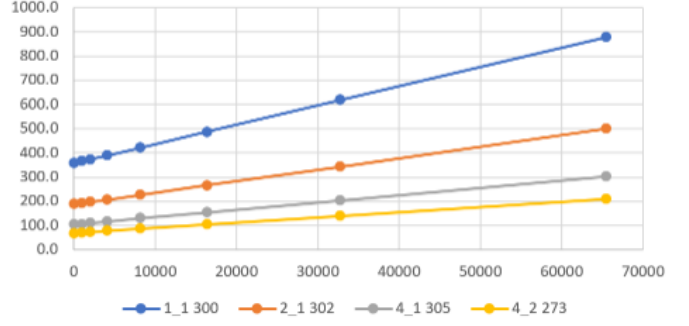


Fig. 7. Long-range pipeline performance ( $\mu$ s).

In Table III, the column labelled “Ideal” is the predicted runtime if the design were able to deliver results at exactly the compiled speed. There are two reasons for measured runtimes that are slower than ideal. First, loop dependencies may prevent the OpenCL compiler from generating a full dataflow design that can accept new operands every cycle. In OpenCL this is known as the initiation interval and the ideal value is 1. All of our designs achieve this goal. Second, unit pipeline latency and data dependency latency in the transpose unit impose delays that occur once per pass through the hardware. These effects are identifiable because they affect small transforms such as  $32^3$  much more than larger ones.

### C. Full LR Pipeline

The second design is the complete implementation of the long range portion of the Smooth Particle Mesh Ewald algorithm. It includes charge spreading, b-spline calculation, atom reordering for hazard suppression, forward and backwards 3D FFT, and force interpolation.

This design is nearly all BRAM based. It uses HBM memory only for storing Green’s function data. We found that any deviation from sequential memory access causes substantial degradation in memory bandwidth when using HBM. With four FPGAs,  $128^3$  problems will not fit. We can use HBM buffering in these cases, but at a performance cost.  $128^3$  will fit in BRAM for configurations of 8 FPGAs or above.

Figure 7 shows the performance of the full long-range pipeline, including charge spreading,  $64^3$  forward and inverse FFT, and force interpolation. The vertical axis is in microseconds and the horizontal axis reports the size of the problem in atoms. The four lines represent 1 board 1 pipeline, 2 boards 1 pipeline each, 4 boards 1 pipeline each, and 4 boards 2 pipelines each. The 4\_2 configuration does not scale perfectly with respect to 4\_1 because the clock speed is lower (273 MHz) and because charge-spreading performance is cable bandwidth limited when there are two processing pipelines per board. An additional source of imperfect scaling is that as more pipelines are added, a greater proportion of atoms overlaps the boundaries between pipelines. At 16 FPGAs, we plan to reverse this effect by doubling the number of CS and FI pipelines because at that scale each one will require fewer resources.

Figure 7 can be used to read out FFT performance by considering the 0-atom case, but we measure this directly.

TABLE IV  
3D FFT PORTION OF LR SUBSYSTEM

Size	B_P	fMax	$\mu s$	Ideal
32x32x32	1_1	313	67	39
32x32x32	2_1	297	36	21
32x32x32	4_1	312	24	10
32x32x32	4_2	289	16	5.3
64x64x64	1_1	311	348	321
64x64x64	2_1	289	193	170
64x64x64	4_1	311	99	79
64x64x64	4_2	276	65	45

We have included extra hardware in the design to obtain cycle accurate timestamps for events such as last-charge-spread-atom and first-force-interpolation-atom, that bracket the FFT computation. These results are shown in Table IV. The ideal column is the minimum possible at the given fMax, with no allowance for unit latency or communications. Current "extra" time is about 5000 clock cycles but this has not been optimized. About 40% of the excess is due to slower than necessary transpose units. When comparing Table III and Table IV bear in mind that the full LR results of Figure 7 and Table IV include two 3D FFTs and are distributed across multiple FPGAs.

#### D. Discussion

The results shown in Table III show close agreement between the ideal results and actual results, with the gap becoming smaller for larger problems. This is consistent with the effects of pipeline latency. As OpenCL compilers improve, we expect the pipeline delays will shrink. These figures are about 150-200 cycles for memory fetch, 134 for Transpose units, and 11 for the FFT. Such improvements would be helpful for small transforms like  $32^3$  but become much less important for  $128^3$  since there is 64 times as much data.

#### E. Performance comparison with other architectures

3D FFT, due to its wide applications in many areas has been benchmarked extensively on many architectures including CPUs, GPUs and ASICs. Many of the benchmarks focus on larger FFTs ( $256^3$  and above) but there is some public information on smaller FFTs applicable to MD. Table V compares performance of our FPGA FFT with CPUs, GPUs and Anton. For converting timing to flops we use  $15N^3 \lg(N)$  for complex FFT.

For GPU measurements, we have depended both on in-house experiments as well as performance benchmarks from [25], [49]. Our GPU code uses CUDA *cuFFT* library [50] for computing FFT. We test this code on a single V100 GPU [51] with CUDA 11.1 compilers and libraries. Our in-house experiments performed on V100 with NVLINK2 as well as [25] show that using multiple GPUs does not improve performance of sizes up to  $128^3$ .

Anton 1 [9] has details of timings for both  $32^3$  and  $64^3$  on 512 nodes, which we have also included in the table.

We also include CPU-based benchmarks in this table. [52] shows the timings using Intel MKL and FFTW on a 56 core Intel Xeon Platinum processor. For sizes  $32^3$ ,  $64^3$  and  $128^3$ , performance on the processor is approximately 200,

TABLE V  
FFT GFLOPS/S FOR MULTIPLE ARCHITECTURES

Size	Size	Size	Size	System	Citation
$32^3$	$64^3$	$64^2 \times 128$	$128^3$		
647	963	-	-	BRAM 16 pipe	Table III
-	-	969	810	HBM 8 pipe	Inhouse tests
664	1774	-	-	Anton-1 512 nodes	[9]
109	139	-	180	Novo-G 8 FPGA	[12]
218	1358	1561	1247	V100 cuFFT	Inhouse tests
180	400	500	610	56C Xeon 8280L	MKL [52]
-	-	-	9	BG/P 512 nodes	[49]

400 and 600 GFlops respectively. We have not found many public benchmarks on performance of smaller distributed 3D FFT. We have included timings on JUGENE, a BlueGene/P architecture [53]. [49] shows 12 milliseconds for a problem of size  $128^3$  on 512 BlueGene/P nodes.

We have also included Novo-G's timings of distributed 3D FFT on 8 Stratix V FPGAs [12] for comparison. Compared to other architectures in Table V, we outperform all architectures for  $32^3$  and  $64^3$  except 512 nodes of Anton 1, and V100 cuFFT for larger sizes such as  $128^3$ .

There is, as far as we know, no magic to achieving excellent 3D FFT performance. It is a game of balancing computation, memory bandwidth, and communication. It should not be a surprise that custom ASICs can do well, nor that modern GPUs like the V100 can achieve more than a teraflop once the problem size grows large enough to sustain efficient memory access (V100 has about 50% more flops than a Stratix 10 FPGA and almost double the memory bandwidth [28], [51]). The attractive features of FPGA designs are that they can run efficiently across a range of sizes and that one can connect compute pipelines directly to communications resources, which means that one can relatively easily distribute a parallel implementation across multiple FPGAs.

Regarding the full implementation of PME we have more limited comparisons. As reported above, our full LR pipeline currently completes a  $64^3$  64K Atom problem in 206 microseconds on four FPGAs. Running on a GeForce RTX 2080TI completes the equivalent phases in 523 microseconds. As discussed above, we note that the FPGA implementation scales readily with cluster size while the GPU implementation does not.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we demonstrate that FPGAs can implement Particle Mesh Ewald in a scalable way, even for the small 3D FFTs applicable in molecular dynamics. The results show that our architecture and implementation balances computation, memory bandwidth, and communications bandwidth to produce implementations that run efficiently across multiple FPGAs. Our implementation works for a variety of molecule sizes and FFT grid sizes, and is completely written in OpenCL for portability and flexibility. Our results show that we outperform or are competitive with a wide variety of architectures including CPUs, GPUs, and ASICs.

The goal of our work is to achieve strong scaling for FFT and the long range pipeline for molecular dynamics on multiple



FPGAs. We plan to grow our FPGA cluster to 16 FPGAs, and present results on scalability of both FFT and the long-range pipeline. A larger cluster will also allow us to use BRAM only on FPGAs for 128<sup>3</sup> transform as well. We also plan to explore avenues such as reducing precision for communications, exploring different layouts of the FFT dataset as well as linking VHDL/Verilog code with OpenCL.

## REFERENCES

- [1] R. Salomon-Ferrer, D. A. Case, and R. C. Walker, "An overview of the AMBER biomolecular simulation package," *WIREs Computational Molecular Science*, vol. 3, no. 2, pp. 198–210, 2013. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1121>
- [2] P. Eastman *et al.*, "OpenMM 7: Rapid development of high performance algorithms for molecular dynamics," *PLoS computational biology*, vol. 13, no. 7, p. e1005659, 2017.
- [3] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, "Reconfigurable molecular dynamics simulator," in *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, 2004, pp. 197–206.
- [4] V. Kindratenko and D. Pointer, "A case study in porting a production scientific supercomputing application to a reconfigurable computer," in *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, 2006, pp. 13–22.
- [5] R. Scrofano, M. Gokhale, F. Trouw, and V. Prasanna, "A hardware/software approach to molecular dynamics on reconfigurable computers," in *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, 2006, pp. 23–32.
- [6] S. Alam, P. Agarwal, M. Smith, J. Vetter, and D. Caliga, "Using FPGA devices to accelerate biomolecular simulations," *Computer*, vol. 40, no. 3, pp. 66–73, 2007.
- [7] C. Yang *et al.*, "Fully integrated FPGA molecular dynamics simulations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–31.
- [8] I. Ohmura, G. Morimoto, Y. Ohno, A. Hasegawa, and M. Taiji, "MDGRAPE-4: a special-purpose computer system for molecular dynamics simulations," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 372, no. 2021, p. 20130387, 2014.
- [9] D. E. Shaw, M. M. Deneroff *et al.*, "Anton, a special-purpose machine for molecular dynamics simulation," *Commun. ACM*, vol. 51, no. 7, p. 91–97, Jul. 2008. [Online]. Available: <https://doi.org/10.1145/1364782.1364802>
- [10] D. E. Shaw, J. Grossman *et al.*, "Anton 2: raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 41–53.
- [11] J. Sheng, B. Humphries, H. Zhang, and M. C. Herbordt, "Design of 3D FFTs with FPGA clusters," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, 2014, pp. 1–6.
- [12] A. Lawande, "A Reconfigurable Interconnect for Large-Scale FPGA Applications and Systems," Ph.D. dissertation, University of Florida, 2016.
- [13] R. Halver, J. H. Meinke, and G. Sutmann, "Kokkos implementation of an ewald coulomb solver and analysis of performance portability," *Journal of Parallel and Distributed Computing*, vol. 138, pp. 48–54, 2020.
- [14] C. Predescu, A. K. Lerer, R. A. Lippert, B. Towles, J. Grossman, R. M. Dirks, and D. E. Shaw, "The  $\mu$ -series: A separable decomposition for electrostatics computation with improved accuracy," *arXiv preprint arXiv:1911.01377*, 2019.
- [15] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen, "A smooth particle mesh ewald method," *The Journal of chemical physics*, vol. 103, no. 19, pp. 8577–8593, 1995.
- [16] A. Sanaullah, A. Khoshparvar, and M. Herbordt, "FPGA-Accelerated Particle-Grid Mapping," in *IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines*, 2016, pp. 192–195, doi: 10.1109/FCCM.2016.53.
- [17] B. Stockwell, *The quest for the cure: The science and stories behind the next generation of medicines*. Columbia University Press, 2011.
- [18] J. Klepeis, K. Lindorff-Larsen, R. Dror, and D. Shaw, "Long-timescale molecular dynamics simulations of protein structure and function," *Current Opinion in Structural Biology*, vol. 19, no. 2, pp. 120–127, 2009.
- [19] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 135, no. 2, pp. 280–292, 1997.
- [20] U. Essmann *et al.*, "A smooth particle mesh ewald method," *The Journal of Chemical Physics*, vol. 103, no. 19, pp. 8577–8593, 1995. [Online]. Available: <https://doi.org/10.1063/1.470117>
- [21] H. J. Berendsen, D. van der Spoel, and R. van Drunen, "GROMACS: a message-passing parallel molecular dynamics implementation," *Computer physics communications*, vol. 91, no. 1-3, pp. 43–56, 1995.
- [22] J. C. Phillips *et al.*, "Scalable molecular dynamics with NAMD," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.
- [23] T. D. Kühne *et al.*, "Cp2k: An electronic structure and molecular dynamics software package-quickstep: Efficient and accurate electronic structure calculations," *The Journal of Chemical Physics*, vol. 152, no. 19, p. 194103, 2020.
- [24] A. Gray, "Creating faster molecular dynamics simulations with gromacs 2020," *devblogs.nvidia.com*, 2020. [Online]. Available: <https://devblogs.nvidia.com/creating-faster-molecular-dynamics-simulations-with-gromacs-2020>
- [25] *Multi-GPU FFT Performance on Different Hardware Configurations*, GTC Silicon Valley 2019, 05 2019.
- [26] M. A. Khan, M. Chiu, and M. C. Herbordt, "FPGA-Accelerated Molecular Dynamics," *Springer*, 2013.
- [27] M. Chiu and M. C. Herbordt, "Molecular Dynamics Simulations on High-Performance Reconfigurable Computing Systems," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 4, Nov. 2010. [Online]. Available: <https://doi.org/10.1145/1862648.1862653>
- [28] *Intel® Stratix® Device Datasheet*, Intel Corporation, March 2020.
- [29] A. George *et al.*, "Novo-G#: A Community Resource for Exploring Large-Scale Reconfigurable Computing Through Direct and Programmable Interconnects," in *HPExC*, 2016.
- [30] A. Putnam, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *Proc. International Symposium on Computer Architecture*, 2014, pp. 13–24.
- [31] C. Plessl, "Bringing FPGAs to HPC production systems and codes," in *Fourth International Workshop on Heterogeneous High-performance Reconfigurable Computing, workshop at Supercomputing*, 2018.
- [32] R. Kobayashi *et al.*, "OpenCL-ready high speed FPGA network for reconfigurable high performance computing," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, 2018, pp. 192–201.
- [33] T. Sasaki, K. Betsuyaku, T. Higuchi, and U. Nagashima, "Reconfigurable 3D-FFT Processor for the Car-Parrinello Method," *Journal of Computer Chemistry, Japan*, vol. 4, no. 4, pp. 147–154, 2005.
- [34] C.-L. Yu, K. Irick, C. Charkrabarti, and V. Narayanan, "Multidimensional DFT IP Generator for FPGA Platforms," *IEEE Trans. Circuits and System I*, vol. 58, no. 4, 2011.
- [35] B. Humphries, H. Zhang, J. Sheng, R. Landaverde, and M. Herbordt, "3D FFT on a Single FPGA," in *Proc. Field Programmable Custom Computing Machines*, 2014.
- [36] A. Ramaswami, *FFT3D for FPGA (CP2K)*, 2019, <http://github.com/pc2/fft3d-fpga>.
- [37] J. Sheng, C. Yang, A. Caulfield, M. Papamichael, and M. Herbordt, "HPC on FPGA Clouds: 3D FFTs and Implications for Molecular Dynamics," in *Proc. Field Programmable Logic and Applications*, 2017.
- [38] J. Lee, L. Shannon, M. J. Yedlin, and G. F. Margrave, "A multi-fpga application-specific architecture for accelerating a floating point fourier integral operator," in *2008 International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, 2008, pp. 197–202.
- [39] R. T. M. An and C. Lu, *Algorithms for Discrete Fourier Transform and Convolution*. New York: Springer-Verlag, 1989.
- [40] M. Frigo and S. G. Johnson, "The design and implementation of fftw3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.
- [41] D. Pekurovsky, "P3dfft: A framework for parallel computations of fourier transforms in three dimensions," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. C192–C209, 2012.
- [42] D. Harris, J. McClellan, D. Chan, and H. Schuessler, "Vector radix fast fourier transform," in *ICASSP'77. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2. IEEE, 1977, pp. 548–551.



- [43] M. Garrido, J. Grajal, M. A. Sanchez, and O. Gustafsson, "Pipelined radix- $2^k$  feedforward fft architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 1, pp. 23–32, 2013.
- [44] F. Annexstein and M. Baumslag, "A unified approach to off-line permutation routing on parallel networks," in *Proceedings of the second annual ACM symposium on Parallel algorithms and architectures*, 1990, pp. 398–406.
- [45] H. Subramoni *et al.*, "Designing topology-aware communication schedules for alltoall operations in large infiniband clusters," *2014 43rd International Conference on Parallel Processing*, pp. 231–240, 2014.
- [46] *OpenCL 2D Fast Fourier Transform Design Example*, Intel Corporation, April 2019.
- [47] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit-dimension permutations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 5, pp. 1148–1160, 2019.
- [48] Lenfant, "Parallel Permutations of Data: A Benes Network Control Algorithm for Frequently Used Permutations," *IEEE Transactions on Computers*, vol. C-27, no. 7, pp. 637–647, 1978.
- [49] A. Sunderlanda *et al.*, "An analysis of FFT performance in PRACE application codes," in *PRACE Whitepaper*, 2012.
- [50] nVidia Corporation, *nVidia@cuFFT*, January 2021, <https://developer.nvidia.com/cufft>.
- [51] —, *nVidia@V100*, January 2021, <https://www.nvidia.com/en-us/data-center/v100/>.
- [52] *Intel Math Kernel Library Performance Benchmarks*, Intel Corporation, 2019.
- [53] I. journal of Research and D. staff, "Overview of the ibm blue gene/p project," *IBM J. Res. Dev.*, vol. 52, no. 1/2, p. 199–220, Jan. 2008.