An OpenCL 3D FFT for Molecular Dynamics Simulations on Multiple FPGAs

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

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

Martin Herboldt  
College of Engineering  
Boston University  
451 D St, Suite 205  
Boston MA, USA  
herboldt@bu.edu

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

Abstract—3D FFTs are used to accelerate MD electrostatic forces computations but are difficult to parallelize due to communications requirements. We present a distributed OpenCL 3D FFT implementation on Intel Stratix 10 FPGAs for grids up to $128^3$. We use FPGA hardware features such as HBM2 memory and multiple 100 Gbps links to provide scalable memory accesses and communications. Our implementation outperforms GPUs for smaller FFTs, even without distribution. For $32^3$ we achieve 4.4 microseconds on a single FPGA, similar to Anton 1 on 512 nodes. For 8 parallel pipelines (hardware limited), we reach the same performance both locally and distributed, showing that communications are not limiting the performance. Our FFT implementation is designed to be part of the electrostatic force pipeline of a scalable MD engine.

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

I. INTRODUCTION

Molecular dynamics (MD) simulations play an important role in drug discovery. 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.

One of the key challenges in molecular dynamics simulations is the long-range (LR) electrostatic force computation, which 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 [3], μ-series [4], and use of Fourier Transforms to solve Poisson’s Equation [5], [6]. 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 $128^3$.

In this paper we describe an FPGA-based architecture, implementation, and evaluation of a distributed 3D FFT applicable to molecular dynamics simulations. Our implementation outperforms GPUs on small transforms, and is scalable to pipelines on a single or multiple boards. It is implemented entirely in OpenCL and works for a variety of sizes applicable to drug discovery. Furthermore, we provide an overall design for a scalable long-range MD pipeline.

The outline of this paper is as follows: Section [III] details the system architecture of our FFT as well as long-range pipeline, and Section [IV] follows up with the implementation details in OpenCL. Section [V] describes how the design from Section [III] fits into a complete MD engine running on multiple FPGAs with minimal host communication. Section [VI] summarizes the results of our work, along with performance comparison to other hardware. Finally, section [VII] concludes the paper along with our plans for future work.

II. BACKGROUND AND RELATED WORK

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 [7], NAMD [8], OpenMM [2], and CP2K [9] allow many practitioners to use MD simulations as a regular tool. AMBER [1], while not open-source, is another MD engine that is widely used in industry. This has led to clusters of GPUs running replica or throughput MD jobs in both academic and commercial settings, but there is a need for improved runtime of fixed size problems (strong scaling) to enable faster development cycles. Strong scaling of molecular dynamics on cluster of GPUs has proved to be successful in systems with millions of atoms. Summit, a hybrid POWER9 Tesla V100 GPU system, has proven to be scalable on thousands of nodes for systems with 21 million and 224 million atoms [10]. Each node on the system is comprised of 2 Power9 nodes with 6 Tesla V100 GPUs. The GPUs are directly connected using the NVLINK2 interconnect, while the nodes themselves are connected using Infiniband. While single GPUs or clusters of GPUs have proved to be enormously successful in throughput jobs and scaling systems with millions of atoms, strong scaling of a single protein system has proved to be far more challenging. To our knowledge, the only study showing strong scaling on GPUs for a 100,000 atom system is with the recently redeveloped GROMACS package [11], [12], which does not distribute the FFT across multiple GPUs.

Several efforts to develop custom ASICs for small molecule simulations have been undertaken. ASICs require long development times and lack reconfigurability, but sheer performance makes them attractive. The earliest initiative is the MDGRAPE [13] series of supercomputers, developed by the...
RIKEN center in Japan. MDGRAPE-4a, the fourth generation of the MDGRAPE series of supercomputers is capable of 1.1 microseconds for a 100,000 atom system. Another very well known initiative for strong scaling molecular dynamics ASICs is the Anton series of supercomputers [14] developed by D. E. Shaw Research and capable of tens of microseconds in a single day. 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 [15].

The most challenging part of scaling molecular dynamics simulations is the electrostatic forces computation, of which FFT is often a major component. Anton required many custom architectural features such as over 300 gigabits per second of bandwidth per node, message latency in the hundreds of nanoseconds, and support for word-level writes and single-ended communication [16]. Using these architectural features, 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 simulations, instead relying on a different decomposition called the $\mu$-series [4], [15]. By using a separable kernel, the long-range Ewald term can be directly calculated by X, Y, and Z 1D convolutions without the use of FFT. This improved performance by 15% but increased communications requirements.

Efforts to get parts of molecular dynamics simulations running on FPGAs have been explored over the past few years [17], [18]. 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 [19].

One of the greatest strength of FPGAs is the I/O transceivers, which are capable of providing hundreds of gigabits of bandwidth per FPGA with very low latency [20]. Some clusters with highly interconnected FPGAs are the Novo-G# built at the University of Florida in a 3D torus interconnect [21] and the first version of the Microsoft Catapult [22]. More recently, University of Paderborn has developed Noctua [23], a Cray system with Stratix 10 FPGAs connected via an optical circuit switch. In addition, Tsukuba University has deployed Cygnus, a hybrid GPU-FPGA system [24]. FPGA communications can also now be programmed using OpenCL [25], providing a high-performance productive environment for distributed applications. Prior work on 3D FFTs on single FPGAs includes [9], [26]–[29], while work on multiple FPGAs includes [30], [31]. Design of FFT for MD simulations is presented in [32]. The earliest 2D floating point FFT on multiple FPGAs of which we are aware is [33]. In this paper, we present the design and implementation of 3D FFT running on multiple directly connected FPGAs and discuss its inclusion in a multi-FPGA MD engine. The entire code is written in OpenCL for ease of programming and portability. Our implementation shows superior performance to GPUs for smaller sizes, and demonstrates strong scaling for multiple pipelines.

III. SYSTEM ARCHITECTURE

In this section, we briefly show how FFT fits into a full Molecular Dynamics application. We then detail our FFT pipeline and communications architecture, and also describe our planned full long-range pipeline that incorporates the charge spreading, FFT, and force interpolation steps, which compute long-range electrostatic forces. We describe the overall architectural and scaling issues that constrain the system design. The overall architectural problem is to build an efficient long-range pipeline which can stream data. A pipeline runs at the speed of its slowest unit, so we apply parallel hardware to different portions of the design in order to balance the whole.

A. Molecular Dynamics

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, but this becomes infeasible for the large numbers of atoms at increasing range. For long-range forces, applications instead use multipole approximations [34] or Ewald summations. Our focus is on an Ewald variation known as Smooth Particle Mesh Ewald (SPME) [35]. 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. 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.

B. 3D FFT

The three dimensional FFT of an XYZ volume can be computed as a sequence of one dimensional transforms [36]. [37]. First, each vector of samples in the X direction (fixed Y and Z) is transformed. There are $N^2$ such 1D transforms,
known as pencils. Then 1D transforms in the Y direction are made, followed by 1D transforms in the Z direction as shown in Figure 2(a). Each 1D transform takes \( N \log N \) operations, leading to a total computation of \( 3N^3 \log N \) operations. For complex data, this takes approximately \( 15N^3 \log N \) flops. Because all X transforms are independent, they can be done in parallel, provided the appropriate input data is local to the FFT hardware. Similarly, all the Y transforms are independent, as are the Z. Figure 2(b) shows a parallelization in which the input volume is broken into slabs that contain subsets of the Z dimension. Within each slab, all data is locally available for the X and Y transforms, but only half the data for each Z dimension transform. After the YFFT phase, the pipelines must exchange data in an all to all pattern so that each pipeline has the data it needs to complete subsets of the Z direction FFTs.

There are alternative parallel 3DFFT formulations such as 2D decomposition as used in [38] and the generalized vector radix decomposition [39] but for up to 64 nodes and 128\(^3\) our initial focus is on the slab decomposition.

In 2012, Galvez, Huang, and Chen showed how to build pipelined parallel FFT hardware using a feedforward architecture that is well suited to FPGA implementations [40]. As an example, an 8-wide parallel FFT unit will compute a 32-point 1D FFT in 4 cycles plus a 3 cycle pipeline latency. Multiple transforms can be pipelined with a 4 cycle pitch. A 64-point FFT takes 8 cycles with a 6 cycle latency. A 128-point FFT takes 16 cycles with a 7 cycle latency. The OpenCL implementation can be compiled for any power of two size. 8-wide single precision complex requires a 512-bit wide data bus, which is a good match for Stratix 10 Fmax in the 250-350 MHz range. This sort of computational unit uses 5-9\% of a Stratix 10 device depending on transform size and consumes and delivers about 19 GB/sec of data.

The challenge of any FFT is that every output depends on every input. The FFT is a great advance over the DFT in that it reduces an \( O(N^2) \) problem to an \( O(N \log N) \) problem, but it doesn’t help with memory requirements or communications. It is straightforward to do the FFT calculations, but not so straightforward to get the correct operands to the functional units in the correct order.

A fully pipelined FFT can complete \( N^2 \) 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 3D FFT will take three times as much computation but the work can be both pipelined and parallelized by interconnecting multiple FFT cores.

1) FFT Scaling: When multiple FFT units are combined on a single FPGA, connecting them is not usually a problem because FPGAs have a great deal of internal connectivity and bandwidth. An 8-wide single precision complex FFT has 512 bit input and output busses, which operate at the OpenCL compiler speeds of around 300 MHz. Each pipeline accepts and delivers data at 154 Gbps or about 19 GBps. These speeds are well within the performance of two striped HBM2 banks, and a Stratix 10 FPGA has 32 such banks. We have packaged and interconnected eight FFT pipelines on a single device with HBM2 I/O and achieved up to 969 GF/s computation performance and 370 GB/s of HBM bandwidth. This is encouraging, but it uses 74\% of the arithmetic resources of the chip, leaving little for the rest of the molecular dynamics problem. In order to achieve still higher performance and permit a fully integrated MD application, it is necessary to parallelize the FFT across multiple FPGAs.

In order to distribute such a system over a network of FPGAs, it is necessary to balance communications and computation performance 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. 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. The role of the all to all network is shown, for two pipelines, in Figure 2. It must reorganize the data of the FFT volume from one distributed along the Z dimension to one distributed along the X or Y direction. A convenient intuition is to consider storage of the FFT volume as being described by a global address, in which the high bits identify the FFT pipeline. The purpose of the All to all is to swap the lowest address bits with the highest bits.

For two pipelines, the overall volume is split between two pipes. Initially, the Z dimension is distributed across the pipelines. After the exchange, the Y dimension is distributed. Each unit must send half its data to the other to accomplish this exchange. For eight processing pipelines, each pipe manages one eighth of the data, and must transmit 1/8 of their share (1/64 of the total) to each of the other pipelines. 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.

1) Network Topologies: In order to implement an efficient all to all, we require a network with very high bandwidth and very high bisection bandwidth, as constrained by available hardware.

Our current testbed uses four BittWare 520N-MX modules [41], each with a single Intel Stratix 10 MX2100 FPGA [20] and 4 QSFP28 communications ports operating at 100 Gbps. The vendor board support package makes the communications links available as 256 bit wide FIFO channels which can operate at up to 390 MHz (100 Gbps). We are operating with a preliminary version which runs up to 305 MHz (78 Gbps).

For two-board designs, we can operate with 4 parallel full duplex links, offering 312 Gbps in each direction. We can operate four boards with point to point links connecting each pair of boards. For larger numbers of boards, the available four ports permit hypercubes up to 16 nodes, or arbitrarily large two dimensional torus networks. We are evaluating use of short copper cables to create two additional links for in-chassis connections. Six links would permit hypercube networks up to 64 boards or 3D torus networks of any size. We use optical external cables so we are not constrained by cable length for off-chassis connections. With four FPGA boards per server, we can build single rack systems up to 64 nodes.

Direct networks such as Hypercube, 2D, and 3D torus networks require multiple hops for full connectivity using an on-chip router (see Section IV-C1). Another possibility would be to use on-board hard Ethernet MACs and 100 Gbps Ethernet switches to build single hop networks.

2) All to all scaling: We can now analyze potential network topologies to evaluate points in the solution space that are compatible with distributed FFT designs.

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. The switched configurations use all available ports, each connected to a different switch to form multiple independent single-hop networks. This is feasible for up to 64 boards due to the availability of 64 port 100G switches. PtoP configurations use point to point cables, which is feasible for up to 5 modules. 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

\[ D \times \frac{N - 1}{N} + \frac{H}{B + N + L} \]

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

| Nodes | Topology | Hopcount | Links | Data Points | Data Bits | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|-------|----------|----------|-------|-------------|-----------|---|---|---|---|----|----|----|----|
| 2     | PtoP     | 1        | 4     | 1.7         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 4     | PtoP     | 1        | 4     | 1.7         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 8     | 2D Torus | 1.5      | 4     | 1.7         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 8     | Hypercube| 1.5      | 3     | 1.5         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 8     | Hypercube++| 1.5 | 4     | 1.7         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 8     | 3D Torus | 1.5      | 3     | 1.5         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 8     | Switched | 1        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 16    | 2D Torus | 1        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 16    | 3D Torus | 1        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 16    | Hypercube| 1        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 32    | Switched | 2        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 32    | 2D Torus | 2        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 32    | 3D Torus | 2        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 32    | Hypercube| 2        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 64    | Switched | 4        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 64    | 2D Torus | 4        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 64    | 3D Torus | 4        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 64    | Hypercube| 4        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |
| 64    | Switched | 1        | 4     | 0.8         | 13.4      | 26.9| 45.4| 107.5 | 1 | 1.5 | 3.3 | 6.6 |

Data within () reference similar entries in table II
average hop count, B is the link bandwidth in bits per second, L is the number of links per board, and N is the number of boards.

The scaling behavior of this equation is such that for a fixed topology, 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 all to all is straightforward. In round $i$, node $n$ transmits to node $(n+i) \mod N$. 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 as described in Section IV-C1.

D. 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.

1) Eight pipelines—Eight pipelines, split among either two or four FPGAs using point to point links. The $128^3$ FFT time for 8 pipelines is 109.2 usec, and the corresponding communications time for 2 or 4 nodes is 107.5. The two board solution requires four parallel point to point links delivering 400 Gbps connectivity.

2) 16 pipelines—16 pipelines split among eight nodes using a hypercube++ topology that has additional links across the major diagonals. The $128^3$ FFT time for 16 pipelines is 54.6 usec, and the corresponding communications time is 58.8 usec.

3) 32 pipelines—32 pipelines split among 16 FPGAs, using a switched topology or a 3D Torus. The $128^3$ FFT time for 32 pipelines is 27.3 usec, and the corresponding communications time for the switched topology is 25.2 usec. The 3D Torus is slightly slower at 33.6 usec. The tradeoff between the two is the relative cost of development work for additional links versus Ethernet encapsulation, switches, and twice as many cables.

4) 64 pipelines—64 pipelines split among 32 FPGAs with a switched topology or among 64 FPGAs with a 3D torus. The $128^3$ time is 13.7 usec and the communications time 13.1 or 13.2.

5) 128 pipelines—128 FFT pipelines split across 64 boards with a switched topology. The $128^3$ FFT time for 32 pipelines is 6.8 usec, and the corresponding communications time for the switched topology is 6.6 usec.

This analysis is done for $128 \times 128 \times 128$ 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.

These solution points permit balanced designs, in which no unit runs faster than necessary, thus minimizing hardware.

E. Long-Range Pipeline

In the previous subsection, we detailed the FFT pipeline and discussed scaling FFT to multiple pipelines on one or more boards. Now we describe the entire long-range force pipeline for molecular dynamics. The overall architecture of the long-range force pipeline is shown in Figure 3. The LR pipeline accepts atom positions and charges as input, and delivers long-range electrostatic force updates per atom back to the overall MD application. The LR pipeline is composed of three separate pieces of hardware, each of which is capable of doing several tasks in time interleaved fashion.

- Charge spreading—transform atom position and charge data into FFT input dataset. The same hardware is used for force interpolation.
- Dual FFT—Two 1D FFT pipelines, with transpose units for data reordering. The hardware is used three times, for X and Y forward transforms, Z and Z-inverse transforms, and Y-inverse and X-inverse transforms. In the Z, Z-inverse case, an additional multiplication is made by the Green’s function data for the long-range kernel in the frequency domain.
- All to All network—This hardware is responsible for exchanging data between FFT pipeline instances and among multiple FPGA boards. It is used twice, to transpose data from the output of the forward Y transform to the input of the forward Z transform, and again to transpose data from the output of the Z-inverse transform to the input of the Y-inverse transform. Data is buffered in high bandwidth memory because the Z transform cannot start until the Y transform is finished.

The overall idea of this LR pipeline is to be able to subdivide the long-range problem into 1 to 128 parallel instances of the LR pipeline that can each process 8 input samples per cycle. With one processing pipeline, a 128-cube problem would take 786432 cycles ($XY$, $ZZ^{-1}$, and $Y^{-1}X^{-1}$ at 262,144 each), while with 128 pipelines, with perfect efficiency process would finish in 6,144 cycles. At 350 MHz, that could be 18
forces for each atom based on gradients of the wide vector FFT, that takes 262,144 cycles, divided by the vectors and four output, offset by a 3 cycle latency, as shown.

The performance of the inverse FFT, and atom position and charge data. 3D cardinal B-splines and their derivatives are used to calculate the electrostatic potential field calculated by the FFT. These updates require completing the charge spreading calculations for one atom in, on average, two cycles in order to not limit performance. To that end, we have built a 64-way parallel pipelined charge spreading unit that can process one atom per cycle, with additional cycles necessary for atoms whose range of influence crosses boundaries of the chunk size of the FFT pipeline input.

2) FFT Unit: Described in section IV

3) Force Interpolation: The force interpolation unit shares the 64-way parallel arithmetic and BRAM units of the charge spreading hardware. It accepts streaming 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. These updates are fed back to the main MD application. The output data from the FFT is temporarily buffered in 8x8x128 chunks in BRAM.

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, 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. 40, 41 (This is also used in 29). This design accepts vectors bit-reversed by lane and in order by vector. For a 32 point transform, there are four input vectors and four output, offset by a 3 cycle latency, as shown in Figure 4.

The internal structure of the parallel FFT is shown in Figure 5. 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 43.

It is convenient to consider a generalized sample address such as X0 X1 X2 X3 X4 Y0 Y1 Y2 Y3 Y4 Z0 Z1 Z2 Z3 Z4 which represents a notional \(32^3\) transform stored in memory with the X indices varying fastest. The low three bits, X0 X1 X2, determine the 8-wide vector lane. The high two bits, Z3 Z4, determine the pipeline unit of a four-pipe design. The middle bits, X3...Z2, represent sequential storage locations in memory or sequential communications through a channel.

The FFT processing pipeline is shown in Figure 6. For this processing pipeline, the generalized sample address must progress through a series of permutations mandated by ordering requirements.

For memory operations, the sequencing is constrained by the data layout required by other parts of the system. To some extent, the memory address generator can change the order but cannot change address bits associated with a parallel datapath and should not change the low few bits of the memory address in order to preserve maximally sequential addressing.

The Input transpose unit changes the ordering of the low 5 bits as required by the FFT hardware input ordering. The FFT output is in bit reversed order, which must also be modified to set up conditions for the following unit.

To accomplish these transformations, we have adopted a flexible transpose unit, shown in Figure 6. The 512 bit vector input proceeds through three ranks of 2-1 multiplexors forming a butterfly network. The first exchanges the high and low 256

---

1 Measured on V100 with cuFFT 9.1
bits. The second exchanges the 128 bit half of each 256 bits. The third rank exchanges 64 bit values in each 128 bit group. At the output of the third exchange, individual 64 bit complex values are written into 8 independent block rams. The input side of the BRAMS have a common write address. The output side read addresses are independent. The 512 bit aggregate output then passes through another three ranks of multiplexors controlled by an output block select signal. 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 “nearly arbitrary” limitation is that this structure cannot transform, say X0 X1 X2 ... into X2 X0 X1 in which the same bits are present in the parallel part of the datapath but in different locations. The limitation can be removed by using 5-stage Benes networks [44] at the input and output but we have found the three stage networks suffice for the permutations encountered in 3DFFT. The main constraint is that each BRAM bank has one read port and one write port, so the scheduler must assure that data read during the same cycle must have been written into different banks.

1) BRAM Only FFT: Molecular modelling, at the scale of interest to us, does not use very much memory, but demands extraordinarily high memory bandwidth. This suggests that it might be advantageous to keep the entire dataset for FFT in FPGA block rams, rather than in HBM memory. We have designed an FFT processing pipeline in this style as well, as shown in Figure 7. This design uses a single FFT processing core, and a single transpose unit, with BRAM expanded to hold a complete slice of the FFT volume. Multiple pipelines are interconnected via the same All to All unit as used elsewhere. The all BRAM design requires three passes to complete a 3D FFT, but because it uses much less hardware than the HBM design, we can fit as many as 16 copies of the pipeline on a single FPGA for FFT sizes at or below $64^3$. As discussed in Section III, the communications requirements mean this is too much computation relative to communications for smaller systems. For designs with 8 FPGAs and up, it becomes possible to fit an entire $128^3$ FFT into BRAM, and an all BRAM FFT will permit substantial hardware savings. We report some performance results for all BRAM designs in Section VI.

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 8 shows such a design for four processing pipelines on two FPGAs. In this case, two full-duplex interboard cables are used for point to point service, with each cable carrying serialized 256 bit words in each direction. The left side of the figure represents 8-sample wide parallel busses from 4 processing pipelines connecting to the All to all network. The right side of the figure represents the output of the all to all connecting back to the 4 processing pipelines. Inter board cables are shown in bold.

Figure 9 is a design for 8 pipelines distributed across four FPGAs. Each board has a direct connection to each other board. Similar to Figure 8, 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.

1) Router: There is a large literature on interconnect networks and routing strategies, but we are taking a somewhat different tack. For the distributed molecular modelling application, we coordinate the activities of multiple FPGAs both through host communications and direct communications. For direct communications, we need nearest neighbor communications for the range-limited and bonded force pipelines, and for
An individual long-range processing pipeline is shown in Figure 10. This unit is replicated in order to achieve the desired performance. The FFT portion of the unit consists of two FFT cores and three transpose units. The FFT can accept input from the charge spreading unit, or from HBM memory. The output of the second FFT can be streamed to the force interpolation unit or sent to the All to All hardware and then buffered in HBM memory.

Within each processing pipeline, as soon as a single XY plane is complete, the second FFT core can begin processing Y direction transforms. As soon as a Y direction transform completes, the output data can stream to a pipelined all to all function, which redistributes data in preparation for Z direction transforms. The output of the all to all cannot be directly streamed to a third FFT core, because the Z direction transforms cannot begin until all the Y direction transforms are complete. Instead, the all to all output is buffered in HBM memory.

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. 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. Long-Range Pipeline

Fig. 9. All to all network. 8 pipelines on four FPGAs, with cables.

An individual long-range processing pipeline is shown in Figure 10. This unit is replicated in order to achieve the desired performance. The FFT portion of the unit consists of two FFT cores and three transpose units. The FFT can accept input from the charge spreading unit, or from HBM memory. The output of the second FFT can be streamed to the force interpolation unit or sent to the All to All hardware and then buffered in HBM memory.

Within each processing pipeline, as soon as a single XY plane is complete, the second FFT core can begin processing Y direction transforms. As soon as a Y direction transform completes, the output data can stream to a pipelined all to all function, which redistributes data in preparation for Z direction transforms. The output of the all to all cannot be directly streamed to a third FFT core, because the Z direction transforms cannot begin until all the Y direction transforms are complete. Instead, the all to all output is buffered in HBM memory. A second pass reuses one of the FFT cores to compute Z direction transforms. The output of the Z direction transform is streamed to the Green’s function multiplier and then to the $Z^{-1}$ transform and then to a second pass through the All to All. A third pass implements the $Y^{-1}$ and $X^{-1}$ transforms and feeds the force interpolation unit. The full forward and inverse 3D FFT can be accomplished in three times the time shown in Table I. At the time of submission of this paper, the full long-range pipeline gives correct results on test data from OpenMM but is not yet running at full speed on actual hardware. Our long-range pipeline execution currently requires hardware and FPGA connection debugging, which can only be performed in person. This has been a bottleneck due to COVID-19 related restrictions on movement.

E. 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 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 5, and then write fairly straightforward code to realize it. The tricky parts are cajoling the compiler into doing what you want. Paradoxically, the language is too high-level for this to be straightforward, yet the development cycle is still much faster with OpenCL than with traditional hardware description languages e.g., Verilog or VHDL. Further, the software and FPGA parts of the OpenCL Runtime provide good tools for moving data between host and devices. Our wish list for OpenCL includes the ability to declare BRAM storage with explicit geometry and to use it from multiple OpenCL kernels, support for pipelines and channels with both endpoints in the same kernel, support for multiple clock domains, better support for bitfields, and low level access to memory controllers and to serial communications hardware. OpenCL provides only high level abstractions, which to some extent hides the underlying power of the hardware. In short we’d like better control over low level implementation, while retaining ease of host program integration.
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\(^2\) provided the HDL modules provide certain prescribed interfaces.

V. Molecular Dynamics on Multiple FPGAs

Our ultimate goal is to perform multiple timesteps of the full MD application on a network of FPGAs without any additional host communication beyond initialization and result collection. This architecture is shown in Figure 1. Required MD functionality must include: short-range, long-range and bonded force calculations, motion integration, on-chip particle data storage and management, etc. [19]. In this paper, we focus on 3D FFT and its use in long-range force calculation while other aspects are the focus of current and future work. We chose FFT instead of the \(\mu\)-series that Anton 2 implemented [15]. At the time of implementation, it was not clear to us if \(\mu\)-series will give us benefits on the FPGA. The \(\mu\)-series seems to increase communication requirements, and therefore this algorithm might be better suited for the Anton-2 architecture and in any case a fast FFT is useful for a variety of problems in addition to MD.

As the required on-chip resources for problem sizes of interest (e.g., 64k-128k atom simulations) exceed that which is available for a single target FPGA, we must employ some degree of resource sharing and/or circuit partitioning across multiple FPGAs to achieve the required performance. We use a straight-forward spatial decomposition scheme to partition work across multiple FPGAs. The short-range and bonded force pipelines operate on nearby particles, with particle migration handled by a motion update module (see Figure 1). Given a fixed problem size, as we increase the number of FPGAs, the required resources per FPGA will decrease. For a high number of FPGAs, there are sufficient resources per FPGA to employ a homogeneous design (i.e., all MD functionality for a given sub-region of simulated space is performed on the same FPGA). For a low number of FPGAs (e.g., 4-8), this is not the case and we must employ a heterogeneous design. This design will assign long-range pipelines to a subset of FPGAs, with all other functionality assigned to the remaining FPGAs. This approach is similar to, and inspired by, the parallelization strategy used in GROMACS [17]. A diagram is shown in Figure 11.

Data communication between long-range and other pipelines is performed through QSFP channels, which directly communicate between FPGAs without coordinating with the host. The communication interface to/from the LR pipeline is a stream of atom positions and charges to and a stream of atom force updates from the pipelines. Although, we ultimately plan to construct a 64-FPGA cluster configured as a \(4^3\) 3D-torus, our initial target of a 4-FPGA prototype is configured as 2 FPGAs with 2 LR pipelines each and the other 2 FPGAs are reserved for the rest of the application. With this configuration, we can support four atoms per cycle and four force updates per cycle, which matches the capability of the LR pipeline design to accept data and deliver results without processing delays. We balance the performance of the various pipelines to minimize the maximum runtime and thus minimize the timestep execution time.

VI. Evaluation

We have developed three versions of 3D FFT. All of these versions are in OpenCL, with no Verilog or VHDL components. The first is a BRAM only design, with up to 16 processing pipelines per FPGA. The second is an HBM-based design with up to 8 processing pipelines per FPGA, but each pipeline includes two FFT units chained together. The third design is a full implementation of FFT and inverse FFT for use in the long-range force pipeline. Because the forward-backward FFT requires six 1D FFTs, it can share hardware and runs in about 2/3 the time of two single transforms.

A. Experimental Setup

Our hardware setup comprises 4 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 FFT programs. Each 520N-MX has 4 QSFP28 channels, each capable of communicating at a peak bandwidth of 100 Gb/s. Each board is connected to each other board using a QSFP28 channel. The fourth channels are connected in pairs to provide double bandwidth for two-board configurations. Each FPGA is configured with the \texttt{p520\_max\_m210h\_BSP} to allow OpenCL as the programming model for FPGA computation as well as communication between the different boards. Our application source code is compiled using Quartus release 19.3. The server runs CentOS

\(^2\)Hardware Description Language: typically VHDL or Verilog
7.6. We use SLURM to manage both compilation and hardware resources.

B. BRAM-based FFT

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 4.4 microseconds at 260 MHz. The $64^3$ version occupies 49% of the BRAMS and 58% of the DSP blocks and runs in 24.5 microseconds.

Because interboard communications are limited to 400 Gbps, one cannot use more than 8 processing pipelines in a two-board or four-board design. For larger systems, BRAM-based $128^3$ transforms become feasible because the amount of BRAM storage scales with the system size. The $128^3$ would require 134 megabits of BRAM, spread out over the entire system. As shown in Table III, the $64^3$ design uses 49% of the BRAMS on one FPGA, so the $128^3$, which has eight times the storage requirement, would fit on 8 FPGAs or up.

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 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. HBM-based FFT

The second design uses BRAM only for transpositions and stages data in HBM memory. We have run versions up to 8 pipelines on a single FPGA or split between two FPGAs or split among four FPGAs. Due to communications limits and available hardware we cannot run more than 8 parallel pipelines. The eight pipeline version for $64^3$ runs in 34 microseconds and the $128^3$ implementation in 272 microseconds. These performance figures are somewhat worse than the BRAM version due to HBM memory latencies and some non-sequential accesses.

D. Full LR Pipeline Design

The forward and inverse pipeline for the full long-range force calculation uses the same structure, but adds a multiplier between the two FFT units and runs the hardware in three passes as shown in Figure [3]. It is thus possible to run the complete FFT and inverse FFT for the long-range force pipeline in 50% longer than the time for a single direction 3D FFT. Results for 1 and 2 board configurations up to 4 pipelines total are shown in Table VI.

E. 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. Hopefully as OpenCL compilers improve, the pipeline delays will shrink to the minimums required by the datapath. 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. The relative benefit of reduced pipeline delays then builds again as the algorithm is distributed over more boards.

There is a much larger performance gap between ideal and measured in our HBM results shown in Tables IV – VI. We include a measurement in Table VI taken with HBM stores disabled, illustrating that HBM stores are responsible for the bulk of the gap between measured and ideal timing. This gap would be eliminated for an all-BRAM solution.

The data in Table VI for 1 board 4 pipeline (981 usec) versus the data for 2 boards 2 pipelines (972 usec) illustrate that...
distribution across multiple boards, given adequate bandwidth, is scalable.

F. 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 (2563 and above) but there is some public information on smaller FFTs applicable to MD. Table VI compares performance of our FPGA FFT with CPUs, GPUs and Anton. For converting timing to flops we use 15*N^3/lg(N) for complex FFT.

For GPU measurements, we have depended both on in-house experiments as well as performance benchmarks from [12], [48]. Our GPU code uses CUDA cuFFT library [49] for computing FFT. The code uses cufftPlan3d to build a FFT plan for best performance and then computes the plan. In our timings, we only include the FFT execution step. We test this code on a single V100 GPU [50] with CUDA 9.1 compilers and libraries. Our in-house experiments performed on V100 with NVLINK2 as well as [12] show that using multiple GPUs does not improve performance of sizes up to 1283.

Anton 1 [14] has details of timings for both 323 and 643 on 512 nodes, which we have also included in the table.

We also include CPU-based benchmarks in this table. [51] shows the timings using Intel MKL and FFTW on a 56 core Intel Xeon Platinum processor. For sizes 323, 643 and 1283, performance on the processor is approximately 200, 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 [52], [48] shows 12 milliseconds for a problem of size 1283 on 512 BlueGene/P nodes.

We have also included Novo-G’s timings of distributed 3D FFT on 8 Stratis V FPGAs [31] for comparison. Compared to other architectures in Table VII, we outperform all architectures for 323 except 512 nodes of Anton 1, and V100 cuFFT for larger sizes such as 1283. Energy readings for 520N-MX FPGAs are not operational in the experimental board support package (BSP) for OpenCL. We are talking to Bittware to get this support in the next release of the BSP.

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 surprising 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 [20], [50]). The attractive features of FPGA designs are that they can run efficiently across a range of problem sizes and that one can connect compute pipelines directly to communications resources, which means that (subject to bandwidth limits) one can relatively easily distribute a parallel implementation across multiple FPGAs.

VII. CONCLUSION AND FUTURE WORK

In this paper, we demonstrated that FPGAs can compute 3D FFTs in a scalable way, even for small transforms applicable in molecular dynamics that are difficult to distribute. The results show that our architecture and implementation balances computation, memory bandwidth, and communications bandwidth to produce 3D FFT implementations that run efficiently across multiple FPGAs. Our implementation works for a variety of sizes, and is completely written in OpenCL for portability. Our results show that we outperform or are competitive with a wide variety of architectures including CPUs, GPUs, and ASICs. We have also outlined our design of long-range pipeline such that the computation of long-range electrostatic forces will not limit the scalability of our future work in molecular dynamics applications.

The goal of our work is to achieve strong scaling for FFT and eventually the full long range pipeline for small molecules on multiple FPGAs. Our future work will include benchmarking the long-range pipeline comprising the FFT implementation on multiple FPGAs. We plan to grow our FPGA cluster to 4 and subsequently 8 FPGAs, and present results on scalability of both FFT and long-range pipeline up to 8 FPGAs. A 4 or 8 FPGA cluster will also allow us to use BRAM only on FPGAs for 1283 transform as well. We also plan to further improve performance and scalability of our current implementation through several avenues such as reducing precision for communications, exploring different layouts of the FFT dataset as well as linking VHDL/Verilog code with OpenCL.

VIII. ACKNOWLEDGMENTS

We would like to thank several individuals at Intel, especially Nick Finamore Jr. and Nadav Zinger for their support of...
this project, including providing loaner hardware as well as discussions and debugging support that was provided. We would also like to thank Molex (Kenneth Robertson and Gildas Genest) for providing hardware, instructions and debugging support of 520N-MX boards.

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, J. Swails, J. D. Chodera, K. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrison, C. D. Stern et al., “OpenMM 7: Rapid development of high performance algorithms for molecular dynamics,” *PLoS computational biology*, vol. 13, no. 7, p. e1005659, 2017.

[3] 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.

[4] C. Plessl, A. K. Lerer, R. A. Grossman, R. M. Dirks, and D. E. Shaw, “The μ-series: A separable decomposition for electrostatics computation with improved accuracy,” *arXiv preprint arXiv:1911.01377*, 2019.

[5] T. Darden, Y.ork, and L. Pedersen, “Particle mesh ewald: An n log (n) method for ewald sums in large systems,” *The Journal of chemical physics*, vol. 98, no. 12, pp. 10 889–10 902, 1993.

[6] U. Eismann, L. Perera, M. I. 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.

[7] 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.

[8] J. C. Phillips, R. Bauern, W. J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, “Scalable molecular dynamics with NAMD,” *Journal of chemical computation*, vol. 26, no. 16, pp. 1781–1802, 2005.

[9] T. D. Kühne, M. Iannuzzi, M. Del Ben, V. V. Rybkin, P. Seewald, F. Stein, T. Laino, R. Z. Khalilullin, O. Schütz, F. Schiﬀmann et al., “Cp2k: An electron 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.

[10] B. Acun, D. Hardy, L. V. Kale, K. Li, J. Phillips, and J. Stone, “Scalable molecular dynamics with NAMD on the Summit system,” *IBM Journal of Research and Development*, vol. 62, no. 6, pp. 4–18, 2018.

[11] 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/

[12] Multi-GPU FFT Performance on Different Hardware Configurations, GTC Silicon Valley 2019, 05 2019, https://developer.nvidia.com/discretionary-[[[content/discretionary-][[[dam/discretionary-][[[www/discretionary-][[[programmable/discretionary-][[[us/en/discretionary-][[[pdf/discretionary-][[[literature/hb/discretionary-][[[stratus-10/discretionary-][[[s10.datash/]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]}
[35] 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. [Online]. Available: https://doi.org/10.1063/1.470117

[36] R. T. M. An and C. Lu, *Algorithms for Discrete Fourier Transform and Convolution*. New York: Springer-Verlag, 1989.

[37] M. Frigo and S. G. Johnson, “The design and implementation of fftw3,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–233, 2005.

[38] 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.

[39] 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.

[40] M. Garrido, J. Grajal, M. A. Sanchez, and O. Gustafsson, “Pipelined radix-2k feedforward fft architectures,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 1, pp. 23–32, 2013.

[41] *Bittware 520N-MX Datasheet*, BittWare Corporation, February 2020, https://www.bittware.com/support-resources/technical-briefs/datasheets/v100/volta-v100-datasheet-update-us-1165301-r5.pdf.

[42] *OpenCL 2D Fast Fourier Transform Design Example*, Intel Corporation, April 2019, https://www.intel.com/support/programmable/support-resources/design-examples/design-software/opencl/fft-2d.html.

[43] 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.

[44] 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.

[45] 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.

[46] H. Subramoni, K. Kandalla, J. Jose, K. Tomko, K. H. Schulz, D. Pekurovsky, and D. K. Panda, “Designing topology-aware communication schedules for alltoall operations in large infiniband clusters,” *2014 43rd International Conference on Parallel Processing*, pp. 231–240, 2014.

[47] *Gromacs 2020 Documentation*, 2020, http://manual.gromacs.org/documentation/2020/index.html.

[48] A. Sunderlanda, S. Picklesa, Milo, Nikoli, A. Jovi, J. Jaki, V. Slavni, I. Girottoc, P. Nashc, and M. Lysaghtc, “An analysis of fft performance in prace application codes,” in *PRACE Whitepaper I. Girotto, P. Nash, and M. Lysaght, “An analysis of fft performance in prace application codes,” in PRACE Whitepaper*, 2012.

[49] NVIDIA® CUDA® FFT, November 2019, https://docs.nvidia.com/cuda/fft/index.html.

[50] NVIDIA® V100, NVIDIA Corporation, January 2020, https://images.nvidia.com/content/technical-briefs/technologies/volta-v100-datasheet-update-us-1165301-r5.pdf.

[51] *Intel Math Kernel Library Performance Benchmarks*, Intel Corporation, 2019, https://software.intel.com/content/technical-briefs/features/benchmarks.

[52] 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. 199220, Jan. 2008.