Massive Scaling of MASSIF: Algorithm Development and Analysis for Simulation on GPUs

Anuva Kulkarni
Carnegie Mellon University
Pittsburgh, USA
anuvak@andrew.cmu.edu

Jelena Kovačević
New York University
New York City, USA
jelenak@nyu.edu

Franz Franchetti
Carnegie Mellon University
Pittsburgh, USA
franzf@andrew.cmu.edu

ABSTRACT

Micromechanical Analysis of Stress-Strain Inhomogeneities with Fourier transforms (MASSIF) is a large-scale Fortran-based differential equation solver used to study local stresses and strains in materials. Due to its prohibitive memory requirements, it is extremely difficult to port the code to GPUs with small on-device memory. In this work, we present an algorithm design that uses domain decomposition with approximate convolution, which reduces memory footprint to make the MASSIF simulation feasible on distributed GPU systems. A first-order performance model of our method estimates that compression and multi-resolution sampling strategies can enable domain computation within GPU memory constraints for 3D grids larger than those simulated by the current state-of-the-art Fortran MPI implementation. The model analysis also provides an insight into design requirements for further scalability. Lastly, we discuss the extension of our method to irregular domain decomposition and challenges to be tackled in the future.

CCS CONCEPTS

• Theory of computation → Parallel algorithms; • Data intensive parallel algorithms;

KEYWORDS

GPU, Fast Fourier Transforms, Domain Decomposition, Algorithms

1 INTRODUCTION

Supercomputers with thousands of cores are used to run scientific simulations to study various phenomena in science and engineering. Today, the fastest supercomputers in the world are powered by thousands of Graphical Processing Units (GPUs), which have tremendous computing power. However, high-memory Fortran simulation workloads that are designed to run on CPU clusters face data movement bottlenecks on modern heterogeneous computing environments. Simulations that involve parallel Fast Fourier Transforms (FFTs) require all-to-all communication [3] and extreme memory requirements well beyond the small on-device memory of a GPU. Thus, algorithm redesign is necessary to bridge the increasing gap between the rate of increase of compute speeds and

that of memory speeds in scaling scientific codes to heterogeneous systems.

In the current computing scenario, GPUs are a relatively inexpensive way of boosting compute power. However, even on a top-ranking supercomputer like Summit (ORNL), communication speeds are limited by the network to 12.5 GB/s. To overcome this bottleneck, we investigate an approach that avoids data movement by redesigning the application algorithm to incorporate data approximations that reduce storage requirements of the FFT computation. The reduced-memory computation can be performed on GPUs. Our use case is a Hooke's law simulation called MASSIF [12, 15, 16], an FFT-based stress-strain computation for composite materials, wherein a microstructure (arrangement of grains as seen in Fig. 1) is discretized on a regular grid. A partial differential equation (PDE) is iteratively solved by convolving 3D tensor fields with rank-4 Green's function tensors. The largest size currently simulated with MPI is $1024 \times 1024 \times 1024$ with a memory requirement of more than 2TB [24]. Scaling and accelerating the MASSIF simulation has a wide range of applications where micromechanical properties of polycrystals are studied.

Figure 1: Illustration a composite microstructure, composed of individual grains [24].

Previous work on scaling MASSIF outlines a domain decomposition algorithm with data modeling and compression to reduce data movement between CPU and GPUs [11]. The algorithm developed in this work focuses on multi-resolution strategies for approximate convolution on distributed domains. We present a method suitable for a distributed-GPU scenario and provide theoretical performance analysis of scalability. More specifically, our main contributions are (1) a multi-resolution octree-based adaptive sampling incorporated in the convolution pipeline, (2) a theoretical study of the gains of our algorithm characterized by data compression and octree-based sampling using performance models.

Finally, we have to recognize that while algorithm innovations such as adaptive sampling reduce communication and memory requirements, it is highly difficult to optimize these operations across various heterogeneous platforms during implementation. The widely-used FFTW interface [7] is not able to express some

This work is licensed under a Creative Commons Attribution International 4.0 License.

PASC '20, June 29-July 1, 2020, Geneva, Switzerland

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7993-9/20/06.

<https://doi.org/10.1145/3394277.3401857>

Figure 2: A component of the stress field tensor in a composite microstructure simulated using MASSIF. The grain boundaries are regions of interest due to the rapid change in the behavior of stress as shown in the pop-out image.

of these special pruning and sampling patterns and hence the user cannot leverage the highly optimized FFTW library for parallel computations. Emerging APIs like FFTX [6] are being built with the goal of user-friendly algorithm specification and optimization with code generation back-ends, but these frameworks are not fully available yet. Hence, in this paper, we focus on algorithm design and leave the implementation details to be discussed at a later date.

The paper is organized as follows. Section 2 describes problem background and related work, Section 3 describes the proposed method for porting MASSIF to GPUs. Section 4 contains description and analysis of the performance models of the proposed algorithm. Section 5 discusses the steps involved in extending our method to irregular domains, and efforts towards high performance implementations.

2 BACKGROUND

This section considers related work and traditional approaches to scaling FFT-based simulations.

MASSIF simulation. MASSIF simulates local stresses and strains in a microstructure by solving a PDE with periodic boundary conditions, which is formulated using the stress-strain constitutive relation and equilibrium conditions. The PDE is solved iteratively using the method of Green’s functions. Each iteration involves FFT-based convolution of rank-2 stress tensor with a rank-4 Green’s function tensor at each 3D grid point. Thus, each iteration requires extensive storage for computation of 3D FFTs of tensor fields.

The pseudocode for the MASSIF kernel is given by Algorithm 1. Strain and stress fields are rank-2 tensors of size 3×3 at 3D grid point $\mathbf{x}$, and are denoted by $\epsilon(\mathbf{x})$ and $\sigma(\mathbf{x})$ respectively. The subscripts denote tensor components. $C_{mnk\ell}(\mathbf{x})$ is the rank-4 stiffness tensor at $\mathbf{x}$. E is initial average strain. The Green’s operator in Fourier space at frequency point ξ is $\hat{\Gamma}_{mnk\ell}(\xi)$. The convergence error is e_s and tolerance error is e_{tol} . $\Delta\epsilon_{k\ell}$ is the computed perturbation in component (k, ℓ) of the strain tensor. Superscripts indicate iteration number. For a given time step, the iterative scheme continues till convergence is reached. In Algorithm 1, line 3 computes the FFT of the stress field. Line 5 computes a Fourier domain convolution and tensor contraction with the Green’s function operator for each stress tensor component. Fourier domain strain update on line 6 is followed by an inverse FFT on the updated strain field. Finally, line 8 updates the stress in the entire volume as per the stress-strain constitutive relation. The estimated memory requirement for storing and computing on multiple 3D tensor quantities for large

Table 1: Extreme memory requirement of MASSIF (serial version) arising from storage for 3D tensor variables.

Input data size	32^3	64^3	128^3	256^3	512^3	1024^3
Memory required (GB)	0.07	0.55	4.44	35.5	284	2272

sized grids is prohibitive, as seen in Table 1. Higher resolutions (grid sizes larger than $1024 \times 1024 \times 1024$) are very difficult to simulate using the current algorithm as-is. Currently, there is no GPU-enabled MASSIF implementation.

Algorithm 1 MASSIF Inner loop

- 1: **Initialize:**
 $\epsilon^0 \leftarrow E, \quad \sigma_{mn}^0(\mathbf{x}) \leftarrow C_{mnk\ell}(\mathbf{x}) : \epsilon_{k\ell}^0(\mathbf{x})$
- 2: **while** $e_s > e_{\text{tol}}$ **do**
- 3: $\hat{\sigma}_{mn}^{(i)}(\xi) \leftarrow \text{FFT}(\sigma_{mn}^{(i)}(\mathbf{x}))$
- 4: Check convergence
- 5: $\Delta\epsilon_{k\ell}^{(i+1)}(\xi) \leftarrow \hat{\Gamma}_{k\ell mn}(\xi) : \hat{\sigma}_{mn}^{(i)}(\xi)$
- 6: Update strain: $\epsilon_{k\ell}^{(i+1)}(\xi) \leftarrow \epsilon_{k\ell}^{(i)}(\xi) - \Delta\epsilon_{k\ell}^{(i+1)}(\xi)$
- 7: $\epsilon_{k\ell}^{(i+1)}(\mathbf{x}) \leftarrow \text{iFFT}(\epsilon_{k\ell}^{(i+1)}(\xi))$
- 8: Update stress: $\sigma_{mn}^{(i+1)}(\mathbf{x}) \leftarrow C_{mnk\ell}(\mathbf{x}) : \epsilon_{k\ell}^{(i+1)}(\mathbf{x})$

Application algorithm redesign. The traditional approach to scaling scientific simulations that use FFTs has been to hand down the application algorithm as-is, while using more distributed CPU nodes and distributed FFTs implemented with Message Passing Interface (MPI). Optimizations are generally made in FFT implementations using libraries or by improving communication overhead [2, 17]. Adapting the application as-is to GPUs is difficult due to the GPU memory limitation. Non-uniform FFTs [4, 10] and pruned FFTs [1, 21, 22] can be used to reduce storage of large FFTs, but have certain drawbacks, which make them unattractive to use in the context of MASSIF due to various reasons such as data quantities not being exactly sparse, non-zero values being scattered in the grid, etc. Hence, these traditional approaches fail to make the best use of modern hardware. Other examples of related work are approximate FFTs such as [23] that operate on local parts of the signal in parallel and then accumulate the results. In contrast, in this work, we modify the application algorithm itself by using approximations in the Fourier space convolution and inverse transform pipeline. An adaptive octree-based sampling is applied to the convolution result, which is why it is referred to as *approximate convolution*. Sampling introduces approximation since some information from the data is lost, but the sampling criteria is chosen such that samples can be used to reconstruct the convolution result faithfully. A related work, the Method of Local Corrections (MLC) [14] also computes local convolutions. However, MLC is used when we are interested in only a subset of the output, whereas our method computes, samples, and reconstructs the full convolution result. As we will discuss later, our approach exploits the nature of the convolution filter to drastically reduce the storage requirement of the results.

High performance implementations with optimized FFT libraries and APIs. Over the last decade or so, the FFTW API has been the de-facto standard FFT interface [7]. High performance FFT libraries by Intel (Math Kernel Library, MKL [9], and Cluster MKL), IBM (ESSL [8] and PESSL), and Nvidia (cuFFT [18]) implement (at least a subset) of the FFTW interface. However, due to increasing complexity of node architectures and memory hierarchies that are, to varying extents, user-controlled, users now need complex mappings of multidimensional FFTs to core optimized 1D FFTs. An example of this is our multi-resolution sampling for MASSIF. Some of these complex mappings are simply not expressible in the current FFTW interface. Hence, new alternatives are emerging, and will be necessary to build high performance implementations of redesigned scientific algorithms (such as the one proposed for MASSIF). For example, the FTX interface [5] is a new framework for building FFT-based applications that builds on the FFTW interface, while extending it to enable performance on exascale machines. The FTX API can express complex mappings of multidimensional data to well-optimized FFT-based kernels, while a SPIRAL-based code generation back-end [6] handles optimization across various hardware platforms. Thus, the advantage of FTX is that it effectively decouples algorithm specification and code optimization. FTX can be used to express algorithm specifications for spectral methods like MASSIF. However, it is still in very early stages of development. Hence, in this paper, we focus our attention only on algorithm development and analysis.

3 ADAPTING MASSIF FOR GPUS

This section presents an overview of the proposed algorithm (Fig. 3). The adaptive sampling strategy for memory reduction on domain-local GPU computations is based on octrees and is also described in detail.

Figure 3: Flow diagram of the proposed domain decomposition method to adapt MASSIF to GPUs. The CPU offloads individual domains to the GPU for iterative update.

3.1 Overview

Algorithm. For simplicity, we assume a problem size of $N \times N \times N$ and all grains (domains) of size $k \times k \times k$. Domain d is processed

locally on the GPU separate from other domains, and Ω_d is the support of domain d , $\Omega_d = \{\mathbf{x} | \mathbf{x} \in \text{Domain } d\}$. We use an additional subscript to denote the domain corresponding to a quantity. Eg., $\sigma_{d,mn}$ denotes the tensor component (m, n) and that the local computation pertains to domain d . The field on the support of the domain alone is denoted with $\sigma_{d,mn}(\Omega_d)$. Superscripts indicate the iteration number. The pseudocode for this method is given by Algorithms 2 and 3.

Algorithm 2 Proposed CPU Algorithm

- 1: **Initialize:**
 $\epsilon^0 \leftarrow E, \quad \sigma_{mn}^0(\mathbf{x}) \leftarrow C_{mnkl}(\mathbf{x}) : \epsilon_{kl}^0(\mathbf{x})$
 - 2: Domain decomposition of the 3D volume of size $N \times N \times N$ into $k \times k \times k$ domains
 - 3: Obtain data models by compressing stress field on the domains
 - 4: Offload domain data models to GPU
-

Algorithm 3 Proposed GPU Algorithm (domain-local processing)

- 1: **Initialize:**
 Reconstruct domain d of size $k \times k \times k$
 - While** $e_s > e_{\text{tol}}$
 - 2: $\hat{\sigma}_{d,mn}^{(i)}(\xi) \leftarrow \text{Domain-local FFT}(\sigma_{d,mn}^{(i)}(\Omega_d))$
 - 3: $\Delta \hat{\epsilon}_{d,kl}^{(i+1)}(\xi) \leftarrow \hat{\Gamma}_{klmn}(\xi) : \hat{\sigma}_{d,mn}^{(i)}(\xi)$
 - 4: $\Delta \epsilon_{d,kl}^{(i+1)}(\Omega_d) + \text{Domain exterior samples} \leftarrow \text{iFFT}(\Delta \hat{\epsilon}_{d,kl}^{(i+1)}(\xi))$
 - 5: Accumulate convolution results from all other domains in $\Delta \epsilon_{\text{accum},kl}^{(i+1)}(\Omega_d)$. Samples exterior to the domain are communicated to other domains for accumulation on their respective supports.
 - 6: $\epsilon_{d,kl}^{(i+1)}(\Omega_d) \leftarrow \epsilon_{d,kl}^{(i)}(\Omega_d) - \Delta \epsilon_{\text{accum},kl}^{(i+1)}(\Omega_d)$
 - 7: $\sigma_{d,mn}^{(i+1)}(\Omega_d) \leftarrow C_{mnkl}(\Omega_d) : \epsilon_{d,kl}^{(i+1)}(\Omega_d)$
 - 8: Check convergence.
-

In Algorithm 3, for an iteration i , line 2 computes domain-local FFT of stress field for domain d . Line 3 computes Fourier space convolution and tensor contraction. Line 4 depicts the inverse transform with adaptive sampling to get updated strain $\Delta \epsilon_{d,kl}$. This yields a dense field over Ω_d and some sampled points in the remaining volume. The adaptive sampling formulation is based on the rapidly decaying property of the Green's function. Line 6 and line 7 corresponds to strain update and stress update on the local domain respectively.

An in-depth description of the domain decomposition strategy is as shown in Fig. 4, using regular 2D square grains to demonstrate offloading, domain-local processing and accumulation of results in a single iteration of the GPU-adapted MASSIF algorithm. Individual grains are used as domains to take advantage of the fact that stress/strain fields in the grain interior are smooth and compressible. These fields can be fitted to a model before offloading to the GPU, thus reducing communication latency if the CPU-GPU link is slow. Following the domain compression and offloading, the GPU performs an FFT-based convolution with the Green's function on the local domain. This operation yields a dense result. However, the Green's function is rapidly decaying, and therefore, so is the

Figure 4: Detailed depiction of domain-local computation in the proposed GPU-adapted MASSIF algorithm on a single domain (blue). A small amount of data is exchanged between all domains in the accumulation step. The sampling criteria is designed using the rate of decay of Green’s function and the geometry of the domain.

convolution result. An adaptive octree-based sampling is used to sparsely sample sub-regions of the dense result such that it fits within GPU memory. Accumulation of results from other domains is performed before proceeding to the next iteration. Thus, compression, domain-local FFT and adaptive sampling enable us to perform iterative updates without materializing the full 3D data volume in memory.

Domain-local FFT and Convolution Domain decomposition methods divide the large 3D grid into smaller domains to be processed in parallel on a number of workers. However, performing convolution on each small domain (which is embedded in a larger volume of zero values) yields a full grid-sized non-zero result because the FFT needs to be zero-padded for correct results. As the problem scales, it becomes impossible to contain domain-local computations on individual workers (GPUs). This is where we apply a multi-resolution sampling strategy derived from properties of the data and Green’s function convolution kernel.

Given that GPU memory is a limiting factor, FFT of a domain residing on the GPU is computed pencil-wise, one dimension at a time, as shown in Fig. 5. Pencils which are all zeros can be ignored. Transformation in the X-dimension gives a beam of $N \times k \times k$ non-zeros. By transforming in Y dimension, we get a slab of $N \times N \times k$ non-zeros. In the Z-dimension, we transform one pencil at a time. Then, element-wise multiplication is performed with a Green’s function pencil, which is computed at required frequency points using the closed form in [16]. The pencil is immediately inverse transformed so that an $N \times N \times N$ grid does not need to be stored. As discussed before, the decaying convolution result is adaptively sampled to further reduce memory required to store the result. This crucial step allows large memory savings for domain-local convolution. We discuss the adaptive sampling method next.

3.2 Octree-based Adaptive Sampling for Approximate Convolution

The Green’s function convolution result of a non-zero domain embedded in a larger, zero padded volume is a dense, decaying 3D field since the result is non-zero outside the domain. A simple illustration of a sampling pattern that can be used for this volume is seen in Fig. 6(a). Octrees are the natural candidate to perform multi-resolution sampling of this type over the 3D volume.

Advantages of octree-based sampling. Fig. 6(b) shows another possible sampling pattern for the domain-local convolution result, using shells of various resolutions around the cubic domain. However, this technique is at a disadvantage during accumulation, which is a key step before every iteration, and involves extracting points from convolution results on other domains. Since the shell is described with respect to the local domain, accumulating implies that only some parts of shells from other domains need to be extracted. This presents difficulties because of the additional processing and communication steps involved in locating the relevant parts of the shells in distributed memory. In distributed nodes or distributed GPU machines, accumulating results is thus a major issue. Since an octree essentially samples the volume around the domain in a shell pattern but describes the shell as a union of multiple octree cells, it is easier to extract the addresses associated with a given cell and communicate them while accumulation of the result. Thus, the structure of the octree makes it easier to accumulate results on a distributed system.

Construction of octree. The octree is a hierarchical data structure based on recursive decomposition, mainly used to represent images or 3D point clouds. Octrees are used to partition a $\mathbb{R}^n$ space by recursively subdividing it into 2^n equal-sized hypercubes, called cells. Thus, each internal node in the tree has 2^n children ($n = 3$ for 3D space) and the tree can have arbitrary depth. The criterion for subdivision depends on satisfaction of certain conditions e.g., a maximum number of points per cell. In our case, construction of

Figure 5: Illustration of convolution on GPU of a domain using domain-local FFT. The Green's function is computed on-the-fly, one pencil at a time, to avoid storage. The $.*$ indicates element-wise multiplication.

Figure 6: (a) Octree decomposition sampling pattern, with varying sampling rates. h is the full resolution sampling interval on the domain. The cells around it have decreased sampling rate, i.e., larger sampling interval. (b) A possible adaptive sampling of the decaying convolution result, using shells of various resolutions around the cubic domain.

the octree is based on sample density of a cell instead of occupancy. We use spread and decay rate of the Green's function and the size of the domain as parameters to determine sample density around the domain. Fig. 7 shows the octree-derived sampling pattern after convolving a cubical domain in the 3D data cube with the Green's function. The octree downsamples cells in the volume aggressively with increased distance from the domain, thus reducing the memory required to store the result on the GPU.

Compact encoding. From a programming perspective, the octree is encoded by an object S which specifies the (x, y, z) coordinates of the left corner point of each cell, size and the sampling rate used inside the cell. Thus, S maps the array of data values stored on the GPU to locations of the sampled points using only 5 integers per octree cell. Thus, the compact object S is used during accumulation and interpolation at the end of each iteration.

Accumulating results. The convolution result in the exterior of some domain d is captured by the corresponding octree sampling descriptor S_d , along with an array of data values at the sampled points. These results from all domains must be accumulated before proceeding to the next iteration. In this section, we develop an algorithm for accumulating the multi-resolution samples locally on each domain.

Figure 7: Octree-based sampling result for domain of size $32 \times 32 \times 32$ embedded in a cube of size $128 \times 128 \times 128$ with a smooth field in the interior, convolved with Green's function.

First, let us define some notation used to describe the algorithm. Let D be total number of domains that the 3D volume is divided into. Let $\Delta\epsilon_d^{(i)}(\Omega_d)$ be the convolution result over domain d at the end of iteration i . $\Delta\epsilon_{d'}^{(i)}(\Omega_d)$ is the convolution result over the support of domain d obtained from domain d' , where $d' \in \{1, \dots, D\}$, but $d' \neq d$. We do not denote the tensor components of $\Delta\epsilon$ here for simplicity of notation, but accumulation is done for each component. Now, $\Delta\epsilon_d^{(i)}(\Omega_d)$ is sampled at full resolution on Ω_d . However, $\Delta\epsilon_{d'}^{(i)}(\Omega_d)$, which comes from domain d' , can have sub-regions which are sampled at different resolutions. The 2D example with four domains in Fig. 8 demonstrates this. We denote these sub-regions by $\omega_{d',j}$, where the first subscript refers to the local domain d' , and the second subscript j is an identifier of the sub-region $j \in \mathbb{Z}$. We know that $S_{d'}$ captures the sampling pattern for the domain d' result in the entire 3D data cube, and can be viewed as a union of sub-regions or octree cells $\omega_{d',j}^r$ with different sampling rates given by r . Extracting cells on the support of domain d from a descriptor $S_{d'}$ is denoted by $S_{d'}(\Omega_d)$. This amounts to extracting sub-regions from $S_{d'}$ such that their union is the support of domain d . Hence, if $\Omega_d = \omega_{d',1}^{2h} \cup \omega_{d',2}^{2h} \cup \omega_{d',3}^{4h}$, then

$$S_{d'}(\Omega_d) = S_{d'}(\omega_{d',1}^{2h} \cup \omega_{d',2}^{2h} \cup \omega_{d',3}^{4h}) \quad (1)$$

and the result extracted would be $\Delta\epsilon_{d'}(\Omega_d) = \Delta\epsilon_{d'}(\omega_{d',1}^{2h} \cup \omega_{d',2}^{2h} \cup \omega_{d',3}^{4h})$.

Figure 8: Accumulating results for domain 3 from sampled results on other domains. It can be seen that for domains 1, 2 and 4, the support of domain 3 is made up of sub-regions of varying sampling rates. The sub-regions can be located using $S_1(\Omega_3)$, $S_2(\Omega_3)$, $S_3(\Omega_3)$. Algorithm 4 interpolates the coarser sub-regions and accumulates the results from the three domains onto the support of domain 3.

The main goal here is to linearly interpolate octree cells from each domain d' to full resolution over Ω_d , then add the domain d result to get the accumulated result over Ω_d . This is done by first interpolating all octree cells in Ω_d to an intermediate sampling rate r_f , and then, using the linearity property, performing an addition followed by single interpolation to full resolution. Using this method avoids creating multiple copies of domain d by every other domain. Algorithm 4 outlines the steps in our interpolation method at the end of each iteration.

Algorithm 4 Proposed Accumulation Algorithm for domain d

- 1: For each domain $d' \in \{1, \dots, D\}$, $d' \neq d$, extract all octree cells $\omega_{d',j}$ ($j \in \mathbb{Z}$) over Ω_d using $S_{d'}(\Omega_d)$. Each $\omega_{d',j}$ has a different sampling rate r , $r \in \{h, 2h, 4h, \dots\}$, denoted by the superscript.

$$\omega_{d',j}^r \leftarrow S_{d'}(\Omega_d)$$

- 2: Identify finest sampling rate. Denote this by r_f .
- 3: Interpolate $\Delta\epsilon_{d'}$ over all $\omega_{d',j}^r$ from the respective coarser sampling rate r to rate r_f using the interpolation operator $I_r^{r_f}$.

$$\Delta\epsilon_{d'}(\omega_{d',j}^{r_f}) \leftarrow I_r^{r_f} \Delta\epsilon_{d'}(\omega_{d',j}^r)$$

- 4: Combine all results over all d' and j since they have the same sampling rate r_f .

$$\Delta\epsilon_{\text{combined}}^{r_f} \leftarrow \sum_{d'} \sum_j \Delta\epsilon_{d'}^{r_f}(\omega_{d',j}^{r_f})$$

- 5: Now, the combined result from all other domains over Ω_d has a sampling rate of r_f . Interpolate this to full resolution h .

$$\Delta\epsilon_{\text{combined}}^h \leftarrow I_{r_f}^h \Delta\epsilon_{\text{combined}}^{r_f}$$

- 6: Add to full-resolution domain result. This is the accumulated result for domain d .

$$\Delta\epsilon_d^h(\text{accum}) \leftarrow \Delta\epsilon_d^h + \Delta\epsilon_{\text{combined}}^h$$

Figure 9: Comparison of original domain convolution result to that of sampled and interpolated result. The agreement between them shows that the sampling technique can recover the result faithfully.

3.3 Correctness

We performed some proof-of-concept experiments to verify correctness of the reconstructed results. A 2D slice of a 512×512 domain embedded in a 1024×1024 matrix of zeros is shown on the left in Fig. 9 after convolution with the Green's function operator. The right side shows the reconstructed result after octree sampling for 2D and linear interpolation. It can be observed that this result is in agreement with the original (left). The compression ratio due to sampling is roughly 8 and the 2-norm error is 0.02%. For smaller domains, the compression ratio is larger (compression was 31 times for a 128×128 domain in 1024×1024 problem), with similar error. Additionally, we compared error in convolution result across resolutions. A 128×128 grid simulated with original MASSIF (no approximations) and a refined 1024×1024 grid with approximate convolution showed agreement in the results. Additionally, the refined grid had fewer oscillations. This may be due to the smoothing effect of interpolation.

4 PERFORMANCE MODELS

In this section, we present a theoretical study of the gains from a first-order GPU-adapted MASSIF algorithm model, characterized by data compression and adaptive sampling. This exercise provides analysis and insights into expected scalability from our design, and can be used to gauge programmer efforts in different directions during implementation.

Building the Model. Key operations in the algorithm can be studied from two perspectives, namely memory requirement and computational complexity. We use this analysis to build an algorithm model to estimate GPU memory and compute time in order to predict gains for MASSIF.

Let us assume equal cubical domains of size $k \times k \times k$ and problem size of $N \times N \times N$. First let us consider the **memory footprint** of various operations as enumerated below.

- (1) *Domain offloading:* The domain can be offloaded in either compressed or uncompressed form. For an uncompressed domain, the CPU offloads six double-precision stress tensor components on the domain. The size of the data to be communicated is therefore, $8 \cdot 6 \cdot (k \times k \times k)$ bytes. For the compressed domain fitted with cubic spline models, we communicate model parameters and boundary points. We use 10 polynomial pieces with 5 coefficients each on every 2D plane of the domain. Hence, number of parameters is $10 \cdot 5 \cdot k$ for each of the six tensor component over the domain. Additionally,

since the grain boundaries are not compressed, we communicate the double-precision boundary points (equal to number of points in a p -voxel thick domain boundary, where p is user-determined, typically 4 or 8) along with model parameters. The storage for boundary points is $(k^3 - (k - p)^3) \cdot 8 \cdot 6$ bytes.

- (2) *Domain-local FFT*: Adaptive sampling is used to avoid storage of convolution result outside the domain, but the GPU memory should be able to hold a contiguous slab of data to perform the FFT computation locally. The size of the slab is $N \times N \times k$, hence memory requirement on the GPU is $8 \cdot N \times N \times k$ bytes.
- (3) *Accumulation*: At the end of each iteration, data exchanged between GPUs consists of (1) the full-resolution domain, (2) the octree descriptor capturing locations of samples in the 3D volume, (3) a data array of values for each sample. If double-precision is used, the size of the data exchanged by each GPU is $8 \cdot ((k \times k \times k) + \text{sizeof}(S)) + 4 \cdot (5 \cdot \text{sizeof}(S))$ where S is the octree descriptor, each entry is 5 integers and $\text{sizeof}(S)$ is the number of entries.

Now let us consider **compute time**. To estimate compute time, we use computational complexity of various algorithm operations to obtain a rough operations count for the dominant computations.

- (1) *Compression and decompression of spline models*: The spline fitting problem is solved using block QR factorization. Here, we use the worst-case QR block size estimate to compute number of operations. The parameters are the degree of cubic spline $d_s = 3$, multiplicity $d_s + 1 = 4$, number of partitions/polynomials pieces $M = 10$. Then, worst-case number of blocks in the QR factorization is $M + 1$. Complexity of QR on $p \times q$ matrix is $2pq^2 - \frac{2}{3}q^3$. Decompression is simply evaluating the polynomial at given grid points. A total of $4(d_s + 1)^2$ multiplications are required to compute the reconstructed value for each grid point in the domain [13].
- (2) *Domain-local FFT*: The number of operations to compute FFT of one 1D pencil is $5N \log(N)$. Using our domain local method, we compute k^2 N -length FFTs in the X -dimension, $N \cdot k$ N -length FFTs in Y -dimension and N^2 N -length FFTs in Z -dimension. Hence, total number of operations is $5(k^2 + N \cdot k + N^2)N \log(N)$.
- (3) *Green's function*: The computation of the Green's function involves LU factorization of a 3×3 matrix, along with operations to compute a rank-4 $3 \times 3 \times 3 \times 3$ tensor for each grid point. We attribute about 18 floating point operations for an efficient LU computation, and the number of multiplications to compute the rank-4 tensor is $2 \cdot 3^4 = 162$. Thus, the Green's function computation at N^3 grid points is a substantial portion of the GPU compute time.
- (4) *Tensor contraction*: This is a simple pointwise multiplication and addition across two indices, at each grid point. Number of operations is proportional to number of domain points and number of octree sampled points. We use N^3 for worst estimate.

First-order performance model analysis. In order to estimate performance, we use the memory footprint vs. compute time

comparison in three main aspects of the algorithm: (1) Offloading of decomposed domains from CPU to GPU, (2) Accumulation of results by data transfer between GPUs (3) Computing on GPUs in the presence of memory constraints. This analysis is expanded below, and gives us an overview of various latencies and resource usage estimates for a range of grid sizes and domain sizes. It can be used as a roadmap for making decisions about the resources needed for implementing simulations as per the grid and domain size.

Offloading domains. Consider a single node system that uses either PCI Express (PCIe) 3.0 or NVLink 2.0 between the CPU and GPU. If two to four GPUs are present, the PCIe bandwidth is roughly 16 GB/s (8 GB/s if eight GPUs) while NVLink provides 50 GB/s. These links are the limiting factor in data movement since fast communication channels are assumed to be available between CPU and main memory. We consider two scenarios (1) without domain compression (2) with compression using regression splines. Table 2 gives estimates of offload time in these two scenarios over different types of links for a problem size of $1024 \times 1024 \times 1024$ (since a single node cannot simulate very large sizes). The communication time with uncompressed domains is t_{comm}^u and that with compressed domains is t_{comm}^c , while the respective compute times are t_{comp}^u and t_{comp}^c . It can be seen that for both PCIe and NVLink, compressing the domain can reduce communication by a factor of 1.5 to 5.5 during offloading. This is useful when the cluster available is small and number of domains is larger than number of GPUs. Then, data needs to be moved in and out of the GPU multiple times in each iteration so that all domains are processed. The decompression task does not substantially affect compute time. Hence, in cases where the CPU-GPU link is slow, domain compression provide faster offloading to GPUs. If NVLink 2.0 is present between CPU-GPU, compression may not be necessary since it is 3 times faster than PCIe 3.0 if two or four GPUs are used and 6 times if eight GPUs are used.

Accumulation of results. Accumulation occurs at the end of each iteration in both, single-node, multi-GPU systems and multi-node, multi-GPU systems. The projected number of octree samples for various domain sizes on a 1024 cube volume dataset is used to compute data communication time from GPU to GPU. Note that the octree greatly reduces the communication overhead, however, it is necessary to identify where bottlenecks could occur in the implementation. In table 3, we estimate communication times for the multi-node network connects (Intel Omni-Path and Mellanox EDR InfiniBand) as well as for single node (PCIe or NVLink). In a multi-node system, the speed of the network (Omni-Path or InfiniBand) is the limiting factor. However, the number of GPUs available in a multi-node cluster may be equal to number of domains, therefore data exchange only needs to be done once per iteration. It takes about 20 to 30 iterations for the simulation to converge, and so this latency may be acceptable. For nodes equipped with NVLink between GPUs, data exchange within a node is very fast. Without NVLink, the data must go through the CPU and once again, the PCIe can become a bottleneck. To reduce latency in the accumulation step, octree sampling rate can be adjusted to reduce the data points being exchanged, at the cost of accuracy.

Table 2: Estimated offload times from CPU to GPU on PCIe 3.0 link with 16 GB/s (2 or 4 GPUs per node) and 8 GB/s (8 GPUs) bandwidth and NVLink 2.0 with a 50 GB/s bandwidth. Assumed bandwidth usage efficiency is 90%. A superscript of ‘u’ indicates uncompressed domains are used, while ‘c’ indicates domains are compressed.

Domain size	PCIe 3.0@90%				NVLink 2.0@ 90%		Compute time	
	2 or 4 GPUs		8 GPUs		t_{comm}^u [ms]	t_{comm}^c [ms]	t_{comp}^u [ms]	t_{comp}^c [ms]
	t_{comm}^u [ms]	t_{comm}^c [ms]	t_{comm}^u [ms]	t_{comm}^c [ms]				
$16 \times 16 \times 16$	1.27E-02	1.36E-02	2.543E-02	2.721E-02	4.069E-03	4.355E-03	284.441	284.442
$32 \times 32 \times 32$	0.102	6.377E-02	0.203	0.127	3.255E-02	2.041E-02	285.445	285.447
$64 \times 64 \times 64$	0.813	0.480	1.627	0.960	0.260	0.154	287.542	287.555
$128 \times 128 \times 128$	6.510	2.168	13.021	4.337	2.083	0.694	292.096	292.223
$256 \times 256 \times 256$	52.083	17.231	104.167	34.462	16.667	5.514	302.642	303.653
$512 \times 512 \times 512$	416.667	73.423	833.334	146.846	133.334	23.495	329.485	338.410

Table 3: Estimated time for accumulation of results at the end of each iteration for various types of communication links. Problem size is $1024 \times 1024 \times 1024$ and upto 4 GPUs assumed present per node. Sparse samples obtained by the octree method capture the domain exterior volume using only a small amount of memory.

Domain size	Data to be exchanged [GB]	Omni-Path/EDR InfiniBand	PCIe	NVLink
		100 Gbit/s @80% time [ms]	16 GB/s @90% time [ms]	50 GB/s @90% time [ms]
$16 \times 16 \times 16$	0.084	33.764	5.861	1.875
$32 \times 32 \times 32$	0.1157	46.294	8.037	2.571
$64 \times 64 \times 64$	0.1651	66.068	11.470	3.670
$128 \times 128 \times 128$	0.2311	92.447	16.049	5.135
$256 \times 256 \times 256$	0.4976	199.079	34.562	11.059
$512 \times 512 \times 512$	1.614	645.899	112.135	35.883

GPU compute time vs. memory required (feasibility of very large sizes). On multi-node multi-GPU systems, we explore feasible problem grid sizes based on the tradeoff between compute time and memory required. The number of operations for the three most dominant GPU computations, namely Green’s function FFT, and tensor contraction, helps to estimate GPU compute times for the double precision case. We consider some sample domain sizes for 3D grids with 1024, 2048, 4096 and 8192 points in each dimension. The problem is observed to be compute-dominated as grid size increases, as seen in table 4. On the GPU, we compute FFT in pencils along each dimension. Green’s function tensor is also computed on-the-fly due to large memory requirements. These two operations are compute-heavy and are a function of problem size rather than domain size. It follows that a large number of small-sized domains have a much higher latency than a few large domains. Larger domains require extensive memory on the GPU, and computation quickly becomes infeasible. A possible solution can be to reduce memory requirements further using more aggressive sampling and mixed precision.

Insights and predictions on various machine setups. Consider the following heterogeneous machine setups along with our first-order model:

- (1) **Single node:** Dual-socket with upto 8 GPUs linked to the CPU with NVLink or PCIe 3.0. Eg. An HPE Apollo 6500 server node with 8 Tesla V100 with 16 GB HBM2 per GPU, part of Bridges-AI (Pittsburgh Supercomputing Center) [19].

- (2) **Workstation:** DGX2 with 16 Tesla V100 GPUs with 32 GB HBM2 memory each, and system memory up to 1.5 TB DDR4.
- (3) **Multi-node, multi-GPU:** Nodes interconnected with Omni-Path or InfiniBand. CPU-GPU connected with PCIe 3.0 or NVLink. Eg., The Summit supercomputer (Oak Ridge National Lab) [20] has 4608 nodes, with 6 Nvidia Volta V100 GPUs (each with 16 GB HBM2) per node and NVLink between the CPU and GPUs.

In table 5, we judge if a given problem size and corresponding domain size can be simulated on one of these platforms. This is an estimate based on available memory and computing resources. Additional calculations in MASSIF will need to be factored in, however, we try to capture the dominant operations in this first-order model.

The model estimates that a single node with two or four GPUs can run problems upto $2048 \times 2048 \times 2048$ with domain sizes upto $512 \times 512 \times 512$. This will improve greatly over state-of-the-art MPI code, which requires the Bridges supercomputer for a $1024 \times 1024 \times 1024$ grid. In order to run larger grids such as $4096 \times 4096 \times 4096$, with $128 \times 128 \times 128$ domains, a workstation like DGX2 with 16 GPUs (32 GB HBM2 each) can be used. However, the problem is estimated to take many hours to finish a single iteration. For larger domain sizes, GPU memory beyond current capacities is needed, hence we mark the problem as infeasible. A single $64 \times 64 \times 64$ domain in

Table 4: Estimated single-GPU double-precision compute time for all domains (considering Nvidia Volta V100, 7.5 TFlops at 70% efficiency). The estimate is based on number of operations used for on-the-fly computation of Green’s function, FFT and tensor contraction. Domain decompression time is ignored since it is a small fraction.

Problem size	Domain size	Compute time/domain [s]	No. of domains	Total compute time (single GPU) [Hrs]	Memory for Domain-local FFT [GB]
1024 × 1024 × 1024	128 × 128 × 128	0.292	512	0.041	1
1024 × 1024 × 1024	512 × 512 × 512	0.329	8	0.0007	4
2048 × 2048 × 2048	128 × 128 × 128	2.352	4096	2.676	4
2048 × 2048 × 2048	512 × 512 × 512	2.485	64	0.044	16
4096 × 4096 × 4096	128 × 128 × 128	19.079	32,768	173.662	16
4096 × 4096 × 4096	512 × 512 × 512	19.589	512	2.786	64
4096 × 4096 × 4096	1024 × 1024 × 1024	20.399	64	0.362	128
8192 × 8192 × 8192	64 × 64 × 64	154.882	2,097,152	90224.994	32
8192 × 8192 × 8192	128 × 128 × 128	155.208	2,62,144	11301.902	64
8192 × 8192 × 8192	512 × 512 × 512	157.272	4096	178.940	256
8192 × 8192 × 8192	1024 × 1024 × 1024	160.303	512	22.798	512

a problem of size $8192 \times 8192 \times 8192$ requires about 154 seconds in double-precision computation, which means that if all 27,648 available GPUs on Summit are utilized, it will take about 3.25 hours to process the full volume. However, the memory required for each domain is at the maximum GPU limit. With $128 \times 128 \times 128$ domains, the problem will run faster (0.4 hours) if memory is available (by reducing precision, increasing downsampling, etc).

If all computation can be performed in single precision, the compute time and storage needs will be very different. For example, a $4096 \times 4096 \times 4096$ grid with $256 \times 256 \times 256$ domains could run extremely fast (0.0004 hours/iteration) while an $8192 \times 8192 \times 8192$ grid with $64 \times 64 \times 64$ domains could take 1.5 hours/iteration on Summit, with the domain-local FFT needing 16 GB GPU memory. Error analysis and tradeoffs due to single-precision computation remains to be theoretically determined.

Thus, the performance model provides estimates and insights into the expected gains of our algorithm design. The gains look promising, justifying moving towards implementation of this method.

5 FUTURE WORK

Irregular domains. So far, we considered only regular (cubic) grain topologies. Material science datasets have irregular grains with compressible properties that can be exploited to reduce data movement. Hence, irregular domain decomposition is a difficult yet much needed solution for such problems. The main distinction for irregular domains is that non-zero domains require a description that maps the irregular topology to a regular 3D grid, which is required for continuing with the spectral approach. Hence, different algorithmic strategies may be required to determine optimal sampling patterns. Given a certain pre-determined sample density for regions of interest, samples have to be placed such that the convolution result around irregular features of convex domains (eg. sharp corners) can be faithfully recovered from the sampled result. Sampling can be described using a combination of Cartesian and radial coordinates, or closed-form functions, for compactness. Future work will also explore strategies for very large grids by

using additional approximations, pruning and downsampling to study tradeoffs between performance and accuracy.

Comparison of irregular sampling strategies. Sampling the field exterior to regular polytopic and irregular domains is a difficult problem because of the trade-off between memory requirement and reconstruction accuracy. However, future work will outline a couple of approaches to do so. The most suitable approach will be determined after comparing the trade-offs for each. The two types of approaches are:

- (1) *Octree on irregular polytope.* An octree algorithm will be used to sample around the domain with the required sampling density. Error analysis will be done for the sampling and interpolation operation. Performance models will be built using projected number of sample points required to accurately reconstruct the data.
- (2) *Combination of coordinate systems.* This approach involves using a combination of Cartesian and radial coordinates. It remains to be studied whether this approach is better than using octrees with respect to reconstruction accuracy and memory requirements.

High performance implementations. Emerging tools like FFTX are needed to allow the user to express scientific algorithm specifications with complex data mappings that are then translated to high performance implementations on most hardware platforms. The FFTX API encodes algorithm specifications as an executable plan composed of various sub-plans. Future work includes contributions to FFTX in the form of extensions that can benefit similar spectral applications. For example, a new descriptor functionality (such as the one described in Section 3.2) for octree and quadtree data structures is needed, which can be used instead of offsets and strides in data mapping functions. This representation is better suited for a distributed computing environment, since sub-regions of varying size and resolutions can be located in memory using addresses inferred from the descriptor. Also, a functionality that can describe the domain-local operations is needed to formulate

Table 5: Estimated machine requirements for simulation of various problem sizes when decomposed into equal sized domains as indicated. $N = 1024$ indicates that problem size is $1024 \times 1024 \times 1024$ and $k = 128$ means domain size is $128 \times 128 \times 128$. (.) indicates that the problem can be tractable with additional approximations and conversion to single precision.

Machine	N = 1024		N = 2048		N = 4096			N = 8192			
	k=128	k=512	k=128	k=512	k=128	k=512	k=1024	k=64	k=128	k=512	k=1024
HPE Apollo 6500 single node	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
DGX2	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗
Summit (ORNL)	✓	✓	✓	✓	✓	✓	✓	✓	(✓)	✗	✗

sub-plans that specify FFT, convolution, uniform/non-uniform sampling or multi-resolution sampling, which will be compiled into a single executable plan for the pipeline of domain operations.

6 CONCLUSION

In this paper, we presented an algorithm design and first-order performance analysis for adapting large-scale MASSIF simulations to distributed GPU systems. The algorithm is characterized by domain decomposition and approximating convolutions with octree-based adaptive sampling. These techniques introduce some loss in accuracy, but are necessary to make the large-memory problem tractable on available heterogeneous systems. Our performance model estimates that our GPU-adapted MASSIF algorithm could simulate large grids such as $2048 \times 2048 \times 2048$ or $4096 \times 4096 \times 4096$ on a small workstation like DGX2, which will already be a huge improvement of the state-of-the-art MPI implementation that requires a large multi-node cluster to simulate a $1024 \times 1024 \times 1024$ grid due to memory constraints. Additionally, our model estimates that domain compression will enable upto 5.5 times faster data offloading from CPU to GPU. Additional approximation techniques, use of mixed-precision and similar strategies will be needed in order to scale the method to very large grids, since GPU memory once again proves to be a limiting factor. Thus, the model provides important insights about expected gains before implementation effort begins. We also discuss extending the algorithm design to domains of irregular, complex topologies. Our future work includes contributions to software frameworks like FTX that can be used by other spectral simulations in the scientific community.

Acknowledgements. This work used the Extreme Science and Engineering Discovery Environment (XSEDE) supported by National Science Foundation grant number ACI-1548562 and the Bridges system, supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC). This research was also supported at the Lawrence Berkeley National Laboratory by the Office of Advanced Scientific Computing Research of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

- [1] J. D. Markel. 1971. FFT pruning. In *IEEE Transactions Audio Electroacoust.*, Vol. 19. 305–311.
- [2] A. Nukada, Y. Ogata, T. Endo, and S. Matsuoka. 2008. Bandwidth intensive 3-D FFT kernel for GPUs using CUDA. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. 1–11. <https://doi.org/10.1109/SC.2008.5213210>
- [3] K. Czechowski, C. McClanahan, C. Battaglini, K. Iyer, P.-K. Yeung, and R. Vuduc. 2012. On the communication complexity of 3D FFTs and its implications for exascale. In *Proc. ACM Int'l. Conf. Supercomputing (ICS)*. San Servolo Island, Venice, Italy. <https://doi.org/10.1145/2304576.2304604>
- [4] A. Dutt and V. Rokhlin. 1993. Fast Fourier Transforms for Nonequispaced Data. *SIAM J. Sci. Comput.* 14, 6 (Nov 1993), 1368–1393. <https://doi.org/10.1137/0914081>
- [5] F. Franchetti et al. 2018. FTX and SpectralPack: A First Look. In *IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC)*.
- [6] F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Puschel, J. C. Hoe, and J. M. F. Moura. 2018. SPIRAL: Extreme Performance Portability. *Proc. IEEE* 106, 11 (Nov 2018), 1935–1968. <https://doi.org/10.1109/JPROC.2018.2873289>
- [7] M. Frigo and S. G. Johnson. 2005. The Design and Implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".
- [8] IBM. [n.d.]. Engineering and scientific subroutine library. ([n.d.]). www.ibm.com/support/knowledgecenter/en/SSHYH8/esslwelcome.html
- [9] Intel. [n.d.]. Math kernel library. ([n.d.]). software.intel.com/mkl
- [10] S. Kestur, S. Park, K. M. Irick, and V. Narayanan. 2010. Accelerating the Nonuniform Fast Fourier Transform Using FPGAs. In *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*. 19–26. <https://doi.org/10.1109/FCCM.2010.13>
- [11] A. Kulkarni, F. Franchetti, and J. Kovačević. 2018. Large-Scale Algorithm Design for Parallel FFT-based Simulations on GPUs. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. 301–305. <https://doi.org/10.1109/GlobalSIP.2018.8646675>
- [12] R. A. Lebensohn. 2001. N-site modeling of a 3D viscoplastic polycrystal using fast Fourier transform. *Acta Materialia* 49, 14 (2001), 2723–2737.
- [13] T. Lyche and K. Morken. [n.d.]. Spline Methods. ([n.d.]). <https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF-MAT5340/v10/undervisningsmateriale/book.pdf>
- [14] P. McCorquodale, P. Colella, G. Balls, and S. Baden. 2006. A Local Corrections Algorithm for Solving Poisson's Equation in Three Dimensions. *Communications in Applied Mathematics and Computational Science* 2 (10 2006). <https://doi.org/10.2140/camcos.2007.2.57>
- [15] J.C. Michel, H. Moulinec, and P. Suquet. 2000. A computational method based on augmented Lagrangians and fast Fourier Transforms for composites with high contrast. *CMES - Computer Modeling in Engineering and Sciences* 1 (01 2000), 79–88.
- [16] H. Moulinec and P. Suquet. 1998. A numerical method for computing the overall response of nonlinear composites with complex microstructure. *Computer methods in applied mechanics and engineering* 157, 1-2 (1998), 69–94.
- [17] A. Nukada, K. Sato, and S. Matsuoka. 2012. Scalable multi-GPU 3-D FFT for Tsubame 2.0 Supercomputer. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (Salt Lake City, Utah) (SC '12)*. IEEE Computer Society Press, Los Alamitos, CA, USA, Article 44, 10 pages. <http://dl.acm.org/citation.cfm?id=2388996.2389056>
- [18] NVIDIA. [n.d.]. cuFFT. ([n.d.]). developer.nvidia.com/cufft
- [19] N.A. Nystrom, M. J. Levine, R. Z. Roskies, and J. R. Scott. 2015. Bridges: A Uniquely Flexible HPC Resource for New Communities and Data Analytics. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure* (St. Louis, Missouri) (XSEDE '15). ACM, New York, NY, USA, Article 30, 8 pages. <https://doi.org/10.1145/2792745.2792775>
- [20] S. S. Vazhkudai et al. 2018. The Design, Deployment, and Evaluation of the CORAL Pre-Exascale Systems. 661–672. <https://doi.org/10.1109/SC.2018.00055>
- [21] H. V. Sorensen and C. S. Burrus. 1993. Efficient computation of the DFT with only a subset of input or output points. *IEEE Transactions on Signal Processing* 41, 3 (March 1993), 1184–1200. <https://doi.org/10.1109/78.205723>
- [22] T. Sreenivas and P. Rao. 1979. FFT algorithm for both input and output pruning. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 27, 3 (June 1979), 291–292. <https://doi.org/10.1109/TASSP.1979.1163246>
- [23] P. T. P. Tang, J. Park, D. Kim, and V. Petrov. 2013. A Framework for Low-communication 1-D FFT. *Sci. Program.* 21, 3-4 (July 2013), 181–195. <https://doi.org/10.1155/2013/672424>
- [24] V. Tari, R. A. Lebensohn, R. Pokharel, T. J. Turner, P. A. Shade, J. V. Bernier, and A. D. Rollett. 2018. Validation of micro-mechanical FFT-based simulations using High Energy Diffraction Microscopy on Ti-7Al. *Acta Materialia* 154, C (8 2018). <https://doi.org/10.1016/j.actamat.2018.05.036>