# A framework for low communication approaches for large scale 3D convolution

Anuva Kulkarni
anuvak@alumni.cmu.edu
Carnegie Mellon University
USA

Jelena Kovačević
jelenak@nyu.edu
New York University
USA

Franz Franchetti
franzf@ece.cmu.edu
Carnegie Mellon University
USA

## ABSTRACT

Large-scale 3D convolutions computed using parallel Fast Fourier Transforms (FFTs) demand multiple all-to-all communication steps, which cause bottlenecks on computing clusters. Since data transfer speeds to/from memory have not increased proportionally to computational capacity (in terms of FLOPs), 3D FFTs become bounded by communication and are difficult to scale, especially on modern heterogeneous computing platforms consisting of accelerators like GPUs. Existing HPC frameworks focus on optimizing the isolated FFT algorithm or communication patterns, but still require multiple all-to-all communication steps during convolution. In this work, we present a strategy for scalable convolution such that it avoids multiple all-to-all exchanges, and also optimizes necessary communication. We provide proof-of-concept results under assumptions of a use case, the MASSIF Hooke's law simulation convolution kernel. Our method localizes computation by exploiting properties of the data, and approximates the convolution result by data compression, resulting in increased scalability of 3D convolution. Our preliminary results show scalability of 8 times more than traditional methods in the same compute resources without adversely affecting result accuracy. Our method can be adapted for first-principle scientific simulations and leverages cross-disciplinary knowledge of the application, the data and computing to perform large-scale convolution while avoiding communication bottlenecks. In order to make our approach widely usable and adaptable for emerging challenges, we discuss the use of FFTX, a novel framework which can be used for platform-agnostic specification and optimization for algorithmic approaches similar to ours.

## CCS CONCEPTS

• **Theory of computation → Parallel algorithms**; • **Data intensive parallel algorithms**; • **Computer systems**;

## KEYWORDS

Scalable Convolutions, Fast Fourier Transform, GPU, Green's functions, scientific simulations

**ACM Reference Format:**
Anuva Kulkarni, Jelena Kovačević, and Franz Franchetti. 2022. A framework for low communication approaches for large scale 3D convolution. In *51th International Conference on Parallel Processing Workshop (ICPP Workshops '22), August 29-September 1, 2022, Bordeaux, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3547276.3548626

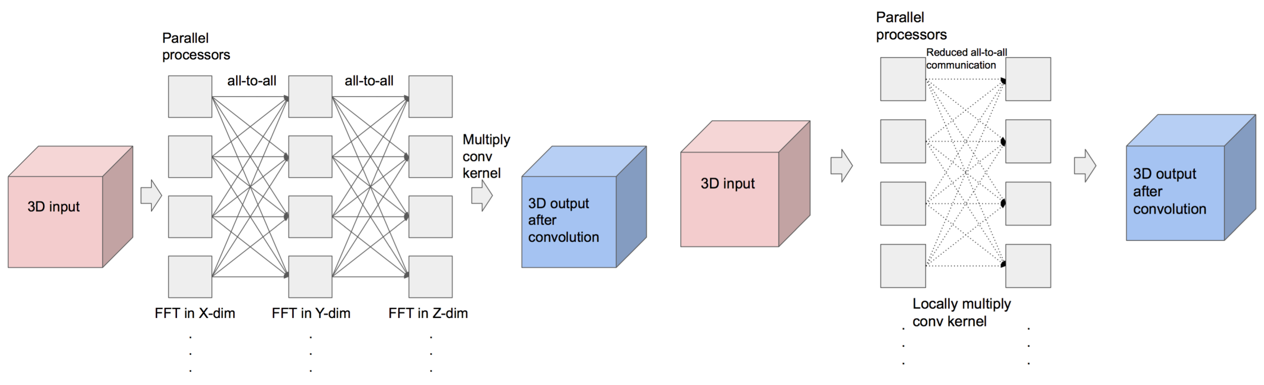
## 1 INTRODUCTION

Large-scale 3D convolutions are key components of many scientific simulations that are used in important applications in materials science, physics, earthquake studies, and many other fields [1]. The Fast Fourier Transform (FFT) is an efficient algorithm for performing convolution since it reduces the complexity of computation from $O(N^2)$ to $O(N\log N)$ [35]. Hence, the FFT is widely used in many important large-scale computing applications like MASSIF[17, 18, 21], LAMMPS [26, 28], HACC [14], XGC [6], WarpX [34], NWChemEx [30, 33].

However, applications using large-scale convolutions face scalability issues due to the FFT algorithm. 3D FFTs, which form the core of 3D convolution computations, while relying on Message Passing Interface (MPI) [29] require all parallel workers to exchange data two or three times, which becomes a scalability barrier because it causes a severe communication bottleneck when number of workers are in the thousands.

The prohibitive all-to-all communication cost for thousands of workers undermines the advantage of parallelism and high compute speeds. The skewed compute-to-communication ratio can cost thousands of dollars when expensive GPU nodes are not used efficiently. Many scientific applications used in US Department Of Energy labs using legacy code face this issue due to difficulty in scaling FFT applications [4].

In order to increase scalability and reduce communication cost, we propose leveraging properties of the kernel like sparsity, symmetry, and impulse-like behavior, which are known to be present in many scientific applications using large-scale convolutions. For example, an FFT-based Hooke's law stress-strain simulation for composite materials called MASSIF, which belongs to the class of partial differential equations (PDEs) solved using integral equations with a Green's function-based kernel, has symmetric and rapidly decaying convolution kernels. Green's functions are used in integral equation solvers that use Fourier methods, and generally are impulse-like functions with symmetries [19]. In this paper, we leverage these properties to demonstrate an approach for highly scalable 3D convolutions that avoids all-to-all communication bottlenecks. Our approach involves algorithmic changes to the convolution pipeline to localize computation while reducing the memory required to store the computation result. We exploit the properties of the input data and kernel to approximate the convolution result with minimal loss in accuracy. The approximate result is stored in a compressed form using smaller amount of memory, and therefore reduces communication between processes. A simplified illustration of traditional approaches versus our approach is shown in Fig.



(a) (b)

**Figure 1: Comparison of traditional FFT approach and proposed method. (a) Traditional distributed FFT-based convolution involving several all-to-all communication steps. (b) Our approach reduces the number of all-to-all communication steps, as well as optimizes the amount of data communicated in them. The FFT-based convolution computation is local to the workers till the last step of accumulation of results.**

1(a). Traditional parallel FFT-based convolution involves multiple all-to-all communication steps to compute the FFT, followed by multiplication with a convolution kernel. An additional inverse FFT also needs to be computed, but it is not shown here. As opposed to this, our approach, illustrated in Fig. 1(b), keeps computation local and avoids data communication until the all-to-all exchange in the last step. The gains of our approach can be realized particularly for resource-demanding scientific applications since the inputs, outputs or convolution kernels in many of them have advantageous and well-known properties, such as infinite domain boundary conditions, symmetry, zero regions, or regions-of-interest as opposed to requirement of the full solution.

This paper makes two main contributions. First, we validate our method with a proof-of-concept implementation with the MASSIF simulation as a use case. Our preliminary results show improved scalability of 8× on a proof-of-concept implementation on a GPU.

Secondly, we discuss ways to make such new algorithmic solutions portable so that they may be adapted to other large-scale FFT-based applications for improving scalability. Currently, algorithm design with complex data mappings is often not portable due to the shortcomings of available FFT libraries and frameworks. We discuss converting specifications with complex mappings into optimized code using FFTX [9], an upcoming framework for building exascale FFT applications, which can be widely used by the scientific community.

## 2 BACKGROUND

### 2.1 Scalability limited by all-to-all communication

For the past decade, it has been known that parallel FFT computation is communication-bounded [8]. Typically, the $N \times N \times N$ point 3D FFT is decomposed into $N^2$ 1D FFTs, each of length $N$, distributed evenly among $P$ nodes. In total, each node sends approximately $N^3/P$ data points. Then, for a fully connected network with link bandwidth $\beta_{link}$, the communication time for a node is

$$T_{Comm,FFT} = 2 \times \frac{N^3}{P * \beta_{link}} \qquad (1)$$

where multiplication with 2 indicates two all-to-all communication stages during 3D FFT computation [7]. There is also additional setup cost for every round of communication, which is accounted for in the $\alpha$-$\beta$ model. In this model, the time required to send a message between two workers is approximated by a linear function:

$$t = \alpha + \beta \times m \qquad (2)$$

where $\alpha$ is the latency of setting up a link for one communication round, $\beta$ is the communication bandwidth cost and $m$ is the message length. Lower bounds for the FFT algorithm's all-to-all communication cost for $P$ workers are derived in [5] and show that the lower bound for number of communication rounds, as well as for message length is a function of $P$ and thus communication cost for FFT worsens for large-scale problems that require thousands of processors $P$.

Most popular high-performance libraries like Intel MKL, FFTW [12], or cuFFT [23] compute multi-dimensional FFTs as a sequence of 1D FFT calls with all-to-all communication between the multiple stages. The all-to-all cost becomes prohibitively high when number of communication rounds is high among $P$ workers. When GPUs are used, data transfers into and out of the GPU are needed repeatedly because large-scale 3D FFTs do not fit within small on-device memory of a GPU. A study in [3] shows that when a $1024^3$ FFT was computed in parallel on 4 CPU nodes, 49.45% of the runtime is spent in communication and only 11.77% in computing the FFT. When accelerated using 4 GPU nodes, the communication time was 97% of the runtime, even though computation was 43× faster. Thus, for a simulation that runs for many days, 97% of the allocated resource budget, amounting to several thousand dollars, is wasted because of communication latency.

Recent years have seen new strategies that are more communication aware. heFFTe [2] uses an asynchronous approach and includes a routine to provide better management of multi-rail communication. But the routine has several all-to-all communication kernels. So, heFFTe can scale to a greater number of nodes than MPI FFT, but eventually also reaches a scalability limitation at a larger node count.

New frameworks such as Halide [27] and AccFFT [13] target optimization of FFTs on CPUs and GPUs. However, most of these frameworks still treat the FFT computation in isolation and/or resort to MKL, CUFFT or FFTW calls. P3DFFT [25] was developed to maximize parallelism but it also relies on MPI and FFTW. Thus, ultimately the frameworks become limited by the same problems faced by all-to-all communication routines in MKL, FFTW and CUFFT. Other high-performance FFT libraries such as FFTE[16] or NukadaFFT[22] are non-ideal due to difficulty of use or restrictions on problem sizes. Recently, there has been a trend towards approximate FFTs. Algorithms such as Peter Tang's low-communication 1D FFT [31], or the sparse FFT (sFFT) [15] may be used by some applications. But Tang's method is not useful for dense 3D convolutions. The sFFT is useful for stricter sparsity conditions on the FFT.

Hence, libraries or frameworks that optimize the FFT algorithm in isolation with strategies like better communication patterns still suffer from communication bottlenecks because they require multiple all-to-all stages and therefore, have limited scalability. It is crucial to reduce communication in order to increase scalability. We believe that the emphasis should be on new algorithmic strategies for parallelizing the convolution pipeline, with a way to avoid data movement. Data movement can be avoided if the memory footprint of the computation can be managed in a better way.

## 2.2 Use case: MASSIF Simulation

MASSIF is a fixed-point iteration method for solving Hooke's law PDEs. The MASSIF simulation is legacy code written originally in Fortran for single-CPU use and has been well-studied by domain experts. Recently, a multi-CPU version is also available [32]. In each iteration, MASSIF runs a stress-strain computation on a 3D grid which represents the discretized microstructure of a composite material. Larger grid sizes allow scientists to better study material properties at high resolution. Scaling and accelerating MASSIF has a wide range of applications for studying micromechanical properties of polycrystals.

We choose the convolution in MASSIF as our use case because of the following reasons. First, because the assumptions of the problem including boundary conditions are well-stated. Secondly, and more importantly, MASSIF is one example of a Green's function-based differential equation solver. Some other examples are simulations modeling heat flow, or elasticity. Other simulations belonging to the same family of linear inhomogeneous PDEs can benefit from adapting our approach. Thirdly, the closed form of the Green's function for MASSIF is known in frequency domain, so it can be computed on-the-fly during convolution, further reducing memory requirement. Finally, MASSIF is a good use case because it is a perfect example of how large-scale convolutions become a bottleneck with high parallelism - each iteration of MASSIF requires multiple 3D convolutions, so the number of all-to-all communication rounds for each iteration is very high, along with the memory required to store results. A parallel FFTW MPI implementation of MASSIF (CPU only) on $1024 \times 1024 \times 1024$ grids has a memory requirement of more than 2TB [32]. Scaling to larger 3D grids increases the number of parallel workers, and due to multiple FFTs computed in each iteration, the all-to-all communication results in exponentially high time cost. We choose the MASSIF convolution example as our use case to show that avoiding all-to-all reduces bottlenecks and increases scalability in fewer resources.

## 2.3 FFTX and the need for specification frameworks

Our proposed algorithm design involves complex sampling patterns and interpolation interwoven with FFT and convolution operations on the 3D grids. These operations are not easily expressible in any off-the-shelf FFT library like FFTW[11]. Other scientific applications also use approximation algorithms or complex data mappings. For example, Poisson's equation solvers using Hockney's method [20] exploit zero-structure and Maxwell's equation solvers [34] have properties which allow the 3D space to be processed as overlapping local sub-problems. But it is difficult to find a single platform-agnostic framework for converting algorithm design into clean, correct and optimized code. Ideally, the code should contain user interfaces to a library that handles optimization operations in the back-end. In addition, the application should be portable and optimized for various hardware platforms. An emerging framework that can fill this gap is FFTX[9].

FFTX is a new framework for building large-scale FFT-based applications. FFTX builds on the FFTW interface while extending it to enable high-performance on exascale machines. FFTX is designed to enable application developers to leverage expert-level, automatic optimizations while navigating a familiar interface. FFTX is backwards compatible to FFTW and extends the FFTW Interface into an embedded Domain Specific Language (DSL) expressed as a library interface. The novelty of the FFTX API is that it can express complex mappings of multidimensional data to well-optimized FFT-based kernels, while a SPIRAL-based code generation backend [10] handles optimization across various hardware platforms. The back-end enables build-time source-to-source translation and advanced performance optimizations, such as cross-library calls optimizations, targeting of accelerators through offloading, and inlining of user-provided kernels. Thus, the advantage of FFTX is that it effectively decouples algorithm specification and code optimization. Section 6 shows how our approach for MASSIF can be expressed using FFTX.

## 3 METHOD

## 3.1 General approach

We present a framework for scaling convolutions and lowering all-to-all communication cost by reducing number of all-to-all rounds, as well as amount of data exchanged. Given a 3D input and a convolution kernel with certain properties, our general approach, a combination of domain knowledge, signal processing and optimized FFT kernels, is outlined below. This approach is also applicable to convolution kernels with limited support or Dirac-like behavior.

**Domain decomposition.** The 3D input is split into chunks, or sub-domains. For now, we assume regular volumetric sub-domains but irregular partitions can also be made.

**Local FFT and convolution with compression.** One or more chunks are batch processed locally inside a worker node. Zero structure is implicit in the 1D calls, so padding is applied to the 1D data, and not to the full 3D array. 1D FFT is applied in each dimension, avoiding redundant computation on the implicit zero structure. After all dimensions are processed, the pointwise multiplication with the convolution kernel takes place. However, during the inverse FFT (iFFT), a compression algorithm is applied after each 1D iFFT stage. Our goal is to reduce communication, which can be achieved if the result to be communicated has a smaller memory footprint. Hence, we perform lossy compression of the convolution result. We use a multi-resolution adaptive sampling technique for compression. The compression algorithm's hyperparameters depend on properties of the input and convolution kernel. This step requires the assumptions of the application to be considered, and we will elaborate more about this in the context of MASSIF.

**Accumulation of results and interpolation.** Exchange of samples between the workers in the last step followed by interpolation gives us the approximate result of the full convolution.

The approach is illustrated in Fig. 2. Given the reduced memory requirement of our method, multiple chunks can be batch processed by a single worker. Unlike traditional methods, the FFT is not computed in parallel. Rather, the entire convolution pipeline is parallelized using domain decomposition and local computing. In order to elaborate on each of the steps above, we require a real-world example which can lend us assumptions that allow us to define the hyperparameters of our approach. The next subsection discusses the MASSIF use case and convolution kernel.

## 3.2 Applying the approach to MASSIF

As discussed before, the MASSIF simulation is a PDE solver for simulating stress and strain in composite materials. It consists of iterative updates to stress and strain fields, $\sigma$ and $\epsilon$ respectively, by convolving stress fields with the Green's function kernel $\Gamma$. In the algorithm below, describing the iterative update, the subscripts indicate tensor component indexing, $\mathbf{x}$ is a 3D spatial grid point, $\xi$ is a frequency domain grid point and $\hat{:}$ indicates the Fourier domain.

**Algorithm 1** MASSIF Inner loop, Iteration $(i+1)$

```
1: while not converged do
2:     FFT of stress tensor: $\hat{\sigma}^{(i)}_{mn}(\xi) \leftarrow \mathrm{FFT}(\sigma^{(i)}_{mn}(\mathbf{x}))$
3:     Convolution with Green's function: $\Delta\hat{\epsilon}^{(i+1)}_{k\ell}(\xi) \leftarrow \hat{\Gamma}_{k\ell mn}(\xi) : \hat{\sigma}^{(i)}_{mn}(\xi)$
4:     Update strain: $\hat{\epsilon}^{(i+1)}_{k\ell}(\xi) \leftarrow \hat{\epsilon}^{(i)}_{k\ell}(\xi) - \Delta\hat{\epsilon}^{(i+1)}_{k\ell}(\xi)$
5:     Inverse FFT of strain tensor: $\epsilon^{(i+1)}_{k\ell}(\mathbf{x}) \leftarrow \mathrm{iFFT}(\hat{\epsilon}^{(i+1)}_{k\ell}(\xi))$
6:     Update stress: $\sigma^{(i+1)}_{mn}(\mathbf{x}) \leftarrow C_{mnk\ell}(\mathbf{x}) : \epsilon^{(i+1)}_{k\ell}(\mathbf{x})$
7:     Check convergence
8: end while
```

The stress-strain computation consists of convolution of rank-2 3D tensor fields with rank-4 Green's function tensors. Hence, 9 convolutions are performed for updating each stress component. There are a total of 9 stress components at each 3D grid point. This amounts to multiple large 3D convolutions per iteration, as seen in steps 2-5 above. The closed form of the Green's function is known in Fourier domain as stated in [21] and as shown in Eqn. 3.

$$\hat{\Gamma}_{ijkl}(\xi) = \frac{1}{4\mu_0|\xi|^2}(\delta_{ki}\xi_l\xi_j + \delta_{li}\xi_k\xi_j + \delta_{kj}\xi_l\xi_i + \delta_{lj}\xi_k\xi_i) - \frac{\lambda_0+\mu_0}{\mu_0(\lambda_0+2\mu_0)}\frac{\xi_i\xi_j\xi_k\xi_l}{|\xi|^4} \tag{3}$$

Here, $\mu_0, \lambda_0$ are Lamé coefficients $\delta$ is the Kronecker delta function and $\xi_i$, where $i \in \{1,2,3\}$ are components of the frequency vector $\xi$. When computationally converted to spatial domain, we observe that the Green's function tensor has the property of rapid decay which is useful in compressing the convolution result.

**Properties of Green's functions.** Convolution with Green's functions is a commonly used technique to solve differential equations, mainly certain types of linear inhomogeneous PDEs. For a given second order linear inhomogeneous differential equation, the Green's function is a solution that yields the effect of a point source, which mathematically is a Dirac delta function. Consider a linear, inhomogeneous equation of the form given below and its solution in the form of an integral equation (inverse of a differential equation), shown on the right:

$$\mathcal{L}u(\mathbf{x}) = f(\mathbf{x}) \rightarrow u(\mathbf{x}) = \int_\Omega G(\mathbf{x},\mathbf{x_0})d\mathbf{x_0} \tag{4}$$

where $u$, $f$ are functions whose domain is $\Omega$ and $\mathcal{L}$ is a linear differential operator. The integral operator $G(\mathbf{x},\mathbf{x_0})$ is the Green's function. The integral can be understood as sum over influences created by sources at each value of $\mathbf{x_0}$. For this reason, $G$ is sometimes called the influence function, typically has decaying behavior, and is used in differential equation solvers to obtain the influence of a particular input. Thus, the Green's function is the solution to $\mathcal{L}G(\mathbf{x},\mathbf{x_0}) = \delta(\mathbf{x}-\mathbf{x_0})$. Typically, the appropriate Green's function is derived by using knowledge of the initial state, boundary conditions, etc. of the differential equation. In the case of Poisson's inhomogeneous equation, the Green's function is

$$G(\mathbf{x},\mathbf{x_0}) = \frac{1}{4\pi|\mathbf{x}-\mathbf{x_0}|} \tag{5}$$

which also has properties in common with MASSIF i.e. decay $\propto 1/x$. Similar Green's functions can also be used to solve complicated equations relating to heat flow, light and particle scattering. Hence, our approach for the MASSIF use case can benefit similar differential equation solvers.

**Approach for scaling convolutions in MASSIF.** Algorithm (2), our method for scaling the 3D convolutions in MASSIF, computes an approximate convolution with compression performed by a multi-resolution adaptive sampling incorporated in the convolution pipeline. Let us consider the specifics of our approach from the perspective of our use-case.

**Step 1: Domain decomposition.** The $N \times N \times N$ 3D input grid is divided into smaller chunks or $k \times k \times k$ 3D 'sub-domains' where $k < N$. The sub-domains can be thought of as composing a 'microstructure' in the case of MASSIF.

**Step 2: Local FFT-based convolution.** In this step, the goal is to compute FFT and convolution locally on each sub-domain within a worker node. However, this step can present a problem to the traditional FFT. Performing convolution on each small sub-domain (which is embedded in a larger volume of zero values) would yield

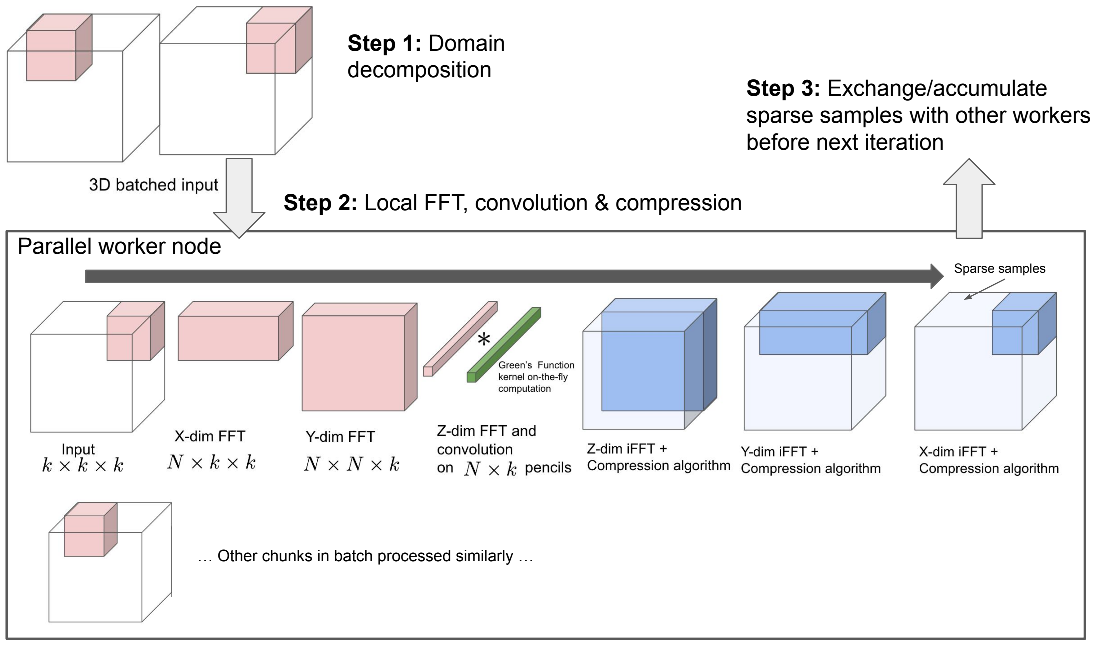

Step 1: Domain decomposition
Step 3: Exchange/accumulate sparse samples with other workers before next iteration
3D batched input
Step 2: Local FFT, convolution & compression
Parallel worker node
Sparse samples
*
Green's Function kernel on-the-fly computation
Input $k \times k \times k$
X-dim FFT $N \times k \times k$
Y-dim FFT $N \times N \times k$
Z-dim FFT and convolution on $N \times k$ pencils
Z-dim iFFT + Compression algorithm
Y-dim iFFT + Compression algorithm
X-dim iFFT + Compression algorithm
… Other chunks in batch processed similarly …

**Figure 2: The outline of the proposed method on parallel workers. Operations taking place inside one worker node are shown. The larger grid resides in main memory and is divided into sub-domains, each sub-domain processed separately. Each sub-domain is locally convolved using FFT with the Green's function and the result is compressed. The $.*$ indicates element-wise multiplication. A small amount of data is exchanged between all workers in the accumulation step *after* completion of each sub-domain convolution.**

a full grid-sized non-zero result because the FFT needs to be zero-padded for correct results. As the grid size scales, local computations for a sub-domain cannot be contained on individual workers. Our solution to avoid this problem scenario is to apply a compression strategy derived from properties of the data and Green's function convolution kernel. The compression technique used is adaptive multi-resolution sampling, where the convolution result is sampled at different rates throughout the 3D grid. The high rate of decay of the Green's function particularly plays a role since it results in a decaying convolution result over a sub-domain embedded in zeros. Such a convolution result lends itself favorably to compressed representation by sampling. Adaptive sampling lowers memory usage and our method performs FFT-based convolution on each sub-domain locally as seen in Fig. 2 and Fig. 4.

**Step 3: Adaptive octree-based multi-resolution sampling as the compression algorithm.** Compression using sampling reduces memory footprint of convolution on the local worker node and allows large memory savings for local convolution. An octree data structure is used to determine sampling patterns in various regions of the 3D grid.

**Step 4: Accumulation.** Accumulating sub-domain results by interpolation and minimal data communication avoids all-to-all between FFT stages. Only sparse samples are exchanged at the end of the computation.

In the pseudocode for our method expressed in Algorithm 2, line 3 computes local FFT of stress field for domain $d$. Line 4 computes Fourier space convolution and tensor contraction. Line 5 depicts

**Algorithm 2** Proposed MASSIF inner-loop, Iteration (i+1)

```
1: Initialize:
       Sub-domain $d$, $\Omega_d = \{\mathbf{x} | \mathbf{x} \in \text{Sub-domain } d\}$. $\sigma_{d,mn}$
       denotes the tensor component $(m, n)$ local to $d$.
2: While not converged, do
3: $\hat{\sigma}^{(i)}_{d,mn}(\xi) \leftarrow \text{Local FFT}(\sigma^{(i)}_{d,mn}(\Omega_d))$
4: $\Delta\hat{\epsilon}^{(i+1)}_{d,k\ell}(\xi) \leftarrow \hat{\Gamma}_{k\ell mn}(\xi) : \hat{\sigma}^{(i)}_{d,mn}(\xi)$ // Convolution
5: $\Delta\epsilon^{(i+1)}_{d,k\ell}(\Omega_d)$ + exterior sub-domain samples $\leftarrow \text{iFFT}\big(\Delta\hat{\epsilon}^{(i+1)}_{d,k\ell}(\xi)\big)$
   // sampling and iFFT
6: Accumulate over all sub-domains, get $\Delta\epsilon^{(i+1)}_{\text{accum},k\ell}(\Omega_d)$.
7: $\epsilon^{(i+1)}_{d,k\ell}(\Omega_d) \leftarrow \epsilon^{(i)}_{d,k\ell}(\Omega_d) - \Delta\epsilon^{(i+1)}_{\text{accum},k\ell}(\Omega_d)$ // local strain update
8: $\sigma^{(i+1)}_{d,mn}(\Omega_d) \leftarrow C_{mnk\ell}(\Omega_d) : \epsilon^{(i+1)}_{d,k\ell}(\Omega_d)$ // local stress update
```

the inverse transform with adaptive sampling to get updated strain $\Delta\epsilon_{d,k\ell}$. This yields a dense field over the sub-domain $\Omega_d$ and some sparse samples which are used to accumulate results on the remaining sub-domains.

**Estimated reduction in memory footprint.** For local convolution proposed in our method, memory requirement on a single worker for double-precision convolution is $8 \cdot N \times N \times k$ bytes. For various values of $N$ and $k$, Table 1 shows memory required to store the full $N \times N \times N$ convolution result and also the estimated

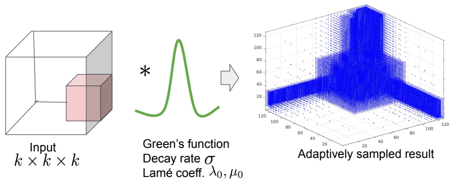


**Figure 3: Octree-based sampling result for sub-domain of size $32 \times 32 \times 32$ in a larger $128 \times 128 \times 128$ grid convolved with Green's function. The adaptive sampling used here downsamples the grid by factors of 2 around the sub-domain in a region of width $k/2$. Further out, the downsampling rate increases and samples are more sparse. The edges of the grid, subject to specific boundary conditions, are densely sampled again.**

memory for processing a sub-domain using our method. The reduced memory footprint of our method means we can scale the 3D convolution possible on a single worker node. Localizing the computation on a worker node means that all-to-all exchange is not needed during the computation except in the last step.

Thus, we expect to achieve higher scalability of convolutions in fewer resources than previously possible. In the next section, we provide the proof-of-concept implementation and experimental results to validate these estimates.

## 4 PROOF-OF-CONCEPT IMPLEMENTATION

Our focus is on improving the compute-to-communication ratio by making optimal use of compute resources on worker nodes and avoiding communication. Instead of adding more parallel resources to attack larger problems, our work shows the increased capability of a single worker in handling computation of very large-scale 3D convolutions using faithful approximations. The proof-of-concept (POC) implementation aims to compute the largest possible 3D convolution using the proposed algorithm on a sub-domain using a single worker node. Then, based on the results, we can justify deploying the algorithm on multi-node platforms in the future.

**Choice of convolution kernel.** The convolution kernel used in each iteration of MASSIF is the Green's function, which has two properties that should be noted: (1) it is rapidly-decaying and (2) it has a real-valued FFT. The exact values of the Green's function depend on the stiffness tensor for the material in question, but generally, for different materials, it has the same decaying behavior. For the POC implementation, we simplify this by using a decaying function with the same properties but without making it specific to a particular material. A sharp Gaussian function fits the requirement. The center of the Gaussian should be at $(N/2+1, N/2+1, N/2+1)$ when using an $N \times N \times N$ grid. This makes sure that the Fourier transform of the Gaussian is real-valued.

**Octrees for adaptive sampling.** The adaptive sampling criteria partitions the 3D grid into regions with varying sample density. Fig. 3 shows the adaptive sampling pattern derived for convolving a sub-domain in the 3D grid with the Green's function. Since the result magnitude reduces with distance from the domain, partitions of the grid with increased distance from the sub-domain are more aggressively down-sampled, thus reducing the memory required to store the result.

The adaptive sampling operation must translate into an efficient implementation for realizing the gains of the proposed algorithm. An efficient data structure is needed to store the sampled result in a compressed format. Storing zeros in the full 3D grid is not memory efficient. If storing in Compressed Sparse Row (CSR) format, it is not easy to localize the result over the different domains or to quickly identify which samples to exchange. Hence we use octrees as a data structure to overcome both these problems. An octree-based adaptive sampling strategy is used to select and store relevant samples. Essentially, the octree captures an estimate of where the hotspots (densely sampled region) will occur once the convolution with the sub-domain is performed. The user parameterizes the sampling strategy around the sub-domain with the spread, decay rate of the Green's function and the size of the sub-domain as parameters to determine sample density in different regions. The sub-domain itself is always sampled at full resolution. The octree metadata is stored in an array, with five consecutive integers capturing the details of one octree cell. The five numbers represent the co-ordinates of the corner point $(x, y, z)$, the downsampling rate of that cell and a count of the total number of samples in the cells that come before the current cell. The last entry helps to decode the octree. The memory footprint is quite small and can be compressed further using lower precision (since we store only integers). The structure of the octree also makes it easier to accumulate results on a distributed system.

**Hardware setup.** A CPU node is used to verify correctness by comparison with FFTW. Different GPUs are used to compare the scalability limits on devices with varying amounts of memory capacity. We use the following compute nodes available on the Bridges Supercomputer at the Pittsburgh Supercomputing Center[24]: (1) HPE Apollo 2000 node: CPU with 2 Intel Broadwell E5-2683 v4 CPUs; 16 cores/CPU, 128 GB RAM (DDR4-2400) with NVIDIA Tesla P100 Pascal architecture GPUs. (2) HPE Apollo 6500 node: 2 Intel Xeon Gold 6148, 20 cores/CPU (40 cores total), 192 GB RAM (DDR4-2666) and NVIDIA Volta V100 GPUs with 16 GB/ GPU. (3) AI node: DGX-2 with 16 V100 GPUs, each with 32 GB/GPU. The CPU is Intel Xeon Platinum 8168 with 24 cores/CPU.

**Software setup.** The POC implementation is done on a GPU in order to show successful scaling of computation within memory constraints. Our POC implementation uses cuFFT in order to conduct a first-order study of performance. Following the idea in Fig. 2, the CUDA program in the POC performs the FFT in two stages: first, the small domain undergoes a 2D transform to a slab. The slab is then transformed in a batch fashion by taking 1D transforms of $B$ pencils at a time in the z-dimension. We use callback functions in cuFFT calls (Fig. 4) to perform padding in the desired regions and point-wise multiplication. In the reverse stage, callback functions implement compression and the full 3D result is not materialized on the GPU.

**Performance metrics and benchmarks.** As this work is in a preliminary stage, the key validation of the algorithm design is the ability to yield a correct result for a larger-than-previously-possible 3D convolution performed domain-by-domain on a single worker. Other performance metrics like compute time and memory usage on the worker are also monitored.

**Table 1: Back-of-envelope calculation for memory required for traditional FFT (stores result in full resolution) and our domain-local FFT (stores result in compressed form) for various grid sizes and sub-domain choices. Our method is estimated to have a lower memory footprint.**

| Problem size $N \times N \times N$ | Domain size $k \times k \times k$ | Memory for traditional FFT [GB] | Memory for local FFT (ours) [GB] |
|---|---|---|---|
| $1024 \times 1024 \times 1024$ | $128 \times 128 \times 128$ | 8 | 1 |
| $1024 \times 1024 \times 1024$ | $512 \times 512 \times 512$ | 8 | 4 |
| $2048 \times 2048 \times 2048$ | $128 \times 128 \times 128$ | 64 | 4 |
| $2048 \times 2048 \times 2048$ | $512 \times 512 \times 512$ | 64 | 16 |
| $4096 \times 4096 \times 4096$ | $128 \times 128 \times 128$ | 512 | 16 |
| $4096 \times 4096 \times 4096$ | $512 \times 512 \times 512$ | 512 | 64 |
| $8192 \times 8192 \times 8192$ | $64 \times 64 \times 64$ | 4096 | 32 |
| $8192 \times 8192 \times 8192$ | $128 \times 128 \times 128$ | 4096 | 64 |

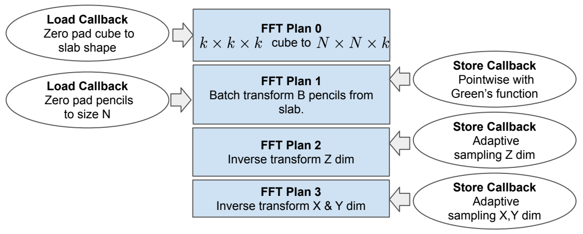

Load Callback
Zero pad cube to slab shape
FFT Plan 0
$k \times k \times k$ cube to $N \times N \times k$
Store Callback
Pointwise with Green's function
Load Callback
Zero pad pencils to size N
FFT Plan 1
Batch transform B pencils from slab.
FFT Plan 2
Inverse transform Z dim
Store Callback
Adaptive sampling Z dim
FFT Plan 3
Inverse transform X & Y dim
Store Callback
Adaptive sampling X,Y dim

**Figure 4: Illustrating CUDA program flow diagram where FFT plans are created and callback functions are attached to them to enable extra functionality.**

## 5 RESULTS

For our proof-of-concept implementation, our aim is to establish that a single worker node can successfully (1) perform local convolution on a sub-domain for larger 3D grids than previously possible, and (2) yield an approximate result within error tolerance. The results here deal with maximizing resource usage of a single worker, which has direct benefits to reducing costs of large-scale simulations. Further work can be focused on extending this algorithm to multiple worker nodes. In our results, we consider GPU worker nodes. Scalability of convolutions locally on GPUs considerably impacts acceleration of large-scale convolutions.

### 5.1 Scaled convolution & low communication

GPUs are a particularly challenging platform for large-scale FFTs because GPUs have limited on-device memory, requiring many more data transfers than a CPU during computation. The value of our method is in processing convolutions on sub-domains for large $N$ within a GPU without storing the full-resolution $N \times N \times N$ result. Our POC implementation demonstrates that our compression algorithm allows successful scaling of MASSIF convolutions to larger 3D grids on a GPU. Moreover, for smaller 3D grids, the method retains its advantage by batch processing multiple 3D convolutions on a GPU, optimizing cluster usage with fewer resources.

As seen in Table 2, our method enables double-precision convolutions of size up to $2048 \times 2048 \times 2048$ on a single GPU. This is 8× points more than traditional cuFFT, which processes up to $1024 \times 1024 \times 1024$ grids without compression. Our method works for combinations of $N$ and $k$ up to a certain $k$ for which GPU memory usage is optimized and sufficient for computation. We use a NVIDIA Volta V100 GPUs with 16 GB/ GPU for grid sizes up to $N = 512$, and one V100 GPU from the DGX-2 with 32 GB/GPU for $N > 512$. For the sake of preliminary results, the GPU sequentially processes the sub-domains. Due to the decaying properties of the Green's function, the convolution result of each sub-domain can be localized on and around the sub-domain, with sparse sampling in the remaining volume without much loss of accuracy. Moreover, the global communication in the FFT stages has been eliminated since each sub-domain is convolved locally on a GPU. As seen before, a traditional parallel 3D FFT's communication time is estimated by equation 1. By reducing the number of all-to-all rounds and by reducing the data exchange in the last round, we can estimate communication time for our method as

$$T_{ours} = \frac{k^3 + \text{sparse samples}}{P * \beta_{link}} \tag{6}$$

where if a $k \times k \times k$ sub-domain is embedded in a $N \times N \times N$ grid and average downsampling rate is considered to be $r$ in each dimension, number of sparse sampled points is $(N^3 - k^3)/r^3$. Thus $T_{ours} < T_{Comm,FFT}$. The downsampling rate $r$ can be increased to reduce the memory requirement further if needed, but at the cost of accuracy.

Recall that the state-of-the-art MASSIF code currently scales only up to $1024 \times 1024 \times 1024$ on parallel CPUs and scaling on GPUs is inefficient due to communication bottlenecks.

Our work enables a GPU implementation for $N = 2048$ that yields an approximate result for MASSIF, which helps MASSIF acceleration efforts. Currently, we do not scale to $N > 2048$ on a single GPU due to high actual memory usage on account of using cuFFT, which creates intermediate temporary variables (see Table 4).

### 5.2 Speedup & Scalability on GPU

Next, we compare speedup and scalability between a traditional CPU implementation and our POC GPU version. A comparison of the runtimes using an Intel Xeon Gold 6148 CPU FFTW implementation vs. our proof-of-concept GPU implementation (NVIDIA V100, 32GB) is seen in table 3 for a convolution on a single sub-domain. We fix the sub-domain size $k = 32$ and vary the grid size

**Table 2: Use of our multi-resolution method allows us to process convolution on grid sizes upto than $2048 \times 2048 \times 2048$ in double precision within the memory of a single GPU. $N$ is the grid size and $k$ is the sub-domain size.**

| $N$ | Allowable $k$ | NVIDIA GPU used |
|---:|---:|---:|
| 128 | $\leq 64$ | V100, 16GB |
| 256 | $\leq 128$ | V100, 16GB |
| 512 | $\leq 256$ | V100, 16GB |
| 1024 | $\leq 256$ | V100, 32GB |
| 2048 | $\leq 64$ | V100, 32GB |

$N$ and downsampling rate $r$. GPU use provides high speedup: for $N = 512$, the GPU speeds up the execution by a factor of 21. For $N = 1024$, GPU speed up is 24×, while using high compression and also preserving result accuracy. For $N > 1024$, the memory limit on the CPU running the FFTW code is reached and comparison is not possible. This is a significant result because we achieve higher scalablity for 3D grids on a GPU than on CPU, even though GPUs have a much smaller on-device memory. In addition to $\approx 20X$ speedup with a GPU, our GPU implementation can compute local FFT for grids with $N > 1024$ due to the compression algorithm, whereas the CPU FFTW's traditional implementation only scales up to $N = 1024$.

### 5.3 Approximation error

We compute approximation error as the L2 relative error norm between the actual and the approximate convolution result. Our POC GPU implementation is compared against a CPU-only FFTW implementation that does not use any approximations. In our approach for MASSIF, we choose adaptive sampling rates for various problem sizes seen in Table 3 to ensure approximation error is low (therefore, accuracy is high). For MASSIF, a fixed-point simulation, convolution error up to 3% did not largely impact convergence or number of iterations, hence we design a heuristic sampling strategy accordingly.

In reality, the accuracy can be tuned to the needs of the application in terms of trade-offs between compute time, downsampling, accuracy and scalability. The error stems from sampling and interpolation. Hence, error bounds for popularly used interpolation methods derived with Taylor's theorem are applicable. Future work will rigorously derive error bounds as a function of our design choices $N$, $k$ and $r$.

### 5.4 Selecting hyperparameters

We are limited to using cuFFT for our POC implementation, requiring hand-tuning of hyperparameters.

**Downsampling rate** $r$ depends on the size of the grid, application requirements and octree granularity. A sweep search for the right downsampling rate, domain size and desired accuracy can be performed under known application requirements. In our experiments, we use $r = 2$ for distance $k/2$ from sub-domain, increase it to $r = 8$ for distance $> k/2$ and $< 4k$, and set it to high values like $r = 16$ or 32 beyond.

**Batch parameter.** During local computation of convolution within the GPU, the $N \times N \times k$ slab is transformed in batches of $B$ 1D FFTs in order to keep memory usage low. Changing the number of 1D 'pencils' processed in a batch using cuFFT has performance gains with respect to GPU compute time. For N = 256 , changing $B$ from 512 to 1024 results in a speedup of 19.9%. These gains are smaller for larger sizes, as other operations might dominate runtime. For example, for N = 1024, changing $B$ from 1024 to 2048 gives a modest 7.35% speedup in time. For the 2048 cube with k = 64, the speedup is modest and in the range of 5-7% for $B = 4096, 8192$ or 32768 due to eventually saturating the parallel concurrency capacity of the batched transforms. None of these are as large as for the N = 256 case, which shows that for smaller sizes, the choice of $B$ matters more.

**Sub-domain size** $k$ should be such that the $N \times N \times k$ slab fits in GPU memory. The GPU memory dictates maximum $k = 32$ in our proof-of-concept implementation as seen in Table 2.

## 6 FFTX FOR ALGORITHM SPECIFICATION

It is highly difficult to hand-optimize the FFT-based approximate convolution with octree-based sampling across various heterogeneous platforms. Using a popular framework such as cuFFT requires the user to write complicated callback functions to get the correct answer stored in a compressed array. Additionally, the pruned or sampled points need to be mapped back into their location in the dense output cube eventually. To solve these problems, the FFTX platform provides two key components: a library interface and a code generation backend. In this section, we show how our algorithm can be written using FFTX API calls, thus decoupling algorithm specification and code optimization. Instead of users writing their own callback functions, FFTX API calls can be used in the code, just like calling a library. The convolution pipeline for MASSIF (forward FFT, point-wise multiplication, inverse FFT with sampling) can be expressed as shown in Fig. 5.

**Structure of the program.** The calls to the `fftx_init` and `fftx_shutdown` functions set up the environment with appropriate options, such as declaring that FFTX should operate in high-performance mode (i.e., enabling symbolic analysis, code generation, and autotuning in the backend). Next, the computation is defined. Similarly to FFTW, this is achieved by first building a plan, i.e., a sequence of computational and data movement steps that, when executed, applies the computation to the application input.

**Plan composition.** The overall FFTX plan is composed of a sequence of sub-plans. Each sub-plan handles a separate task, such as a forward transform, an inverse transform, input padding or output pruning. Fig. 5 shows how FFTX callback functions are used while composing sub-plans. The optimization and code-generation are applied to the overall plan, and hence, across all the sub-plans. The plan can be executed more than once and is an FFTX specification which is transformed into highly optimized code. The FFTX approach allows for fine control over resource expenditure during the optimization. Users can control compile-time, initialization-time, invocation time and optimization resources if they need to [9].

**Code generation.** The core code generation, symbolic analysis, and autotuning software for FFTX is based on SPIRAL [10]. SPIRAL automatically maps computational kernels across a wide range of computing platforms to highly efficient code, and proves the correctness of the synthesized code. This addresses two fundamental problems that software developers are faced with: (1) performance

**Table 3: Speedup in execution on GPU vs CPU FFTW. Here, $N$ is the grid size $N \times N \times N$, $k$ is sub-domain size, $r$ is downsampling factor. The convolutions are local to a single CPU and single GPU. For a GPU with a much smaller on-device memory than a CPU, our method enables scalability and high speedup.**

| $N$ | $k$ | $r$ | Our method runtime (ms) | FFTW runtime (ms) | Speedup | Our Approx. Error |
|---|---|---|---|---|---|---|
| 128 | 32 | 4 | 25.12 | 104.67 | 4.17 | $\leq 3\%$ |
| 256 | 32 | 4 | 88.15 | 1050.25 | 11.91 | |
| 512 | 32 | 4 | 468.01 | 9002.29 | 19.24 | |
| 512 | 32 | 8 | 419.82 | 9009.95 | 21.46 | |
| 1024 | 32 | 32 | 2947.96 | 72016.2 | **24.43** | |

**Table 4: Estimated memory usage and actual memory usage of a GPU while performing approximate convolution on a $k \times k \times k$ sub-domain inside a $N \times N \times N$ grid. $r$ is the downsampling rate. The difference between the values is due to the use of CUFFT, which creates temporaries in the midst of calculations.**

| $N$ | $k$ | $r$ | Estimated Memory(GB) | Actual Memory(GB) |
|---|---|---|---|---|
| 512 | 32 | 16 | 0.62 | 1.29 |
| 1024 | 32 | 32 | 2.49 | 4.33 |
| 2048 | 8 | 128 | 3.52 | 5.67 |
| 2048 | 16 | 128 | 5.02 | 8.16 |
| 2048 | 32 | 128 | 8.00 | 13.16 |
| 2048 | 32 | 64 | 9.97 | 16.20 |
| 2048 | 64 | 64 | 15.92 | 26.20 |

portability across the ever-changing parallel platforms, and (2) verifiable correctness of sophisticated floating-point code. Thus, the application developer does not have to worry about optimization across hardware platforms, which will be handled by SPIRAL. Thus, writing GPU code involving approximations for FFTs in scientific applications will become easier for application developers.

## 7 CONCLUSION

This paper demonstrated a framework for designing approximate FFT-based convolution algorithms for large-scale 3D convolutions applicable for convolution kernels with Green's function-like properties widely used in scientific simulations. The highlights of our method are (1) avoidance of all-to-all during the FFT operation (2) adaptive sampling as an approximation and compression strategy (3) increase in scalability, savings in parallel computing resources in exchange for tolerable approximation error. Using a single-GPU proof-of-concept implementation, our experimental results show reduction of memory footprint due to the adaptive sampling technique, coupled with high accuracy of the approximate result, while also taking advantage of the GPU to accelerate computing speed. Our result sampling strategy avoids all-to-all communication during FFT stages, reducing memory footprint of the parallel convolution and computing using fewer memory resources, making GPUs good target platforms for accelerating our implementation.

Our preliminary results are demonstrated on the MASSIF use case but future work will extend design to other spectral applications. Field calculations for particle-in-cell simulations require large 3D FFTs of $10^9$ -$10^{12}$ points. Other simulations may require relatively small sizes (around $256^3$ data points) but many instances of 3D FFTs per iteration. These first-principle simulations may be handed down over the years as legacy code, which is difficult to scale with traditional parallel FFTs. Future work will involve adapting our multi-resolution approach to these methods to help reduce memory footprint and increase scalability. This paper also discussed how existing frameworks fall short in efficient and portable implementation of approximate algorithms like ours, which require complex data mappings. As an alternative, we highlighted use of FFTX and presented a sketch of how FFTX API calls can be used to easily implement MASSIF and other similar spectral algorithms.

## ACKNOWLEDGMENTS

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). The authors also sincerely thank Dr. Anthony Rollett for valuable feedback and discussions.

## REFERENCES

[1] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. 2006. *The landscape of parallel computing research: A view from berkeley.* Technical Report. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

[2] Alan Ayala, Stanimire Tomov, Azzam Haidar, and Jack Dongarra. 2020. heFFTe: Highly Efficient FFT for Exascale. In *Computational Science – ICCS 2020*, Valeria V. Krzhizhanovskaya, Gábor Závodszky, Michael H. Lees, Jack J. Dongarra, Peter M. A. Sloot, Sérgio Brissos, and João Teixeira (Eds.). Springer International Publishing, Cham, 262–275.

[3] Alan Ayala, Stanimire Tomov, Xi Luo, Hejer Shaiek, Azzam Haidar, George Bosilca, and Jack Dongarra. 2019. Impacts of Multi-GPU MPI Collective Communications on Large FFT Computation. In *2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI)*. 12–18.

[4] Alan Ayala, Stanimire Tomov, Miroslav Stoyanov, and Jack Dongarra. 2021. Scalability Issues in FFT Computation. In *Parallel Computing Technologies*, Victor Malyshkin (Ed.). Springer International Publishing, Cham, 279–287.

```
#include <stdio.h>
#include "fftx.h"

// persistent label for top level plan
#define MY_PLAN_LABEL 0x1234
//#define USE_PERSISTENT_PLAN // Define in make file

#ifdef USE_PERSISTENT_PLAN
#define MY_FFTX_MODE  FFTX_HIGH_PERFORMANCE
#else
// flags for FFTX
#define MY_FFTX_MODE  FFTX_MODE_OBSERVE
#define MY_FFTX_MODE_TOP (FFTX_ESTIMATE | FFTX_MODE_OBSERVE)
#define MY_FFTX_MODE_SUB (MY_FFTX_MODE_TOP | FFTX_FLAG_SUBPLAN)


fftx_plan massif_convolution_plan(fftx_real *small_cube, fftx_real *out,
             fftx_complex *greens_function, int n,
             int n_in, int n_out, int n_freq){

//Initialize
int rank = 3, // 3D = rank 3
numsubplans = 4;// need 4 FFTX subplans for pruned convolution
...
// FFTX iodim definitions for 3D + pruning
//slab_dims, padded_dims, batch_dims etc defined here
...


// RDFT converts small cube into slab (FFT in X and Y dims)
slab = fftx_create_temp_complex(rank, slab_dims);
plans[0] = fftx_plan_guru_dft_r2c(rank, padded_dims,
             batch_rank, &batch_dims,
             small_cube, slab,
             MY_FFTX_MODE_SUB);


// pointwise operation
tmp3 = fftx_create_temp_complex(rank, batch_pencils_dims);
plans[1] = fftx_plan_guru_pointwise_c2c(rank, freq_dimx,
              batch_rank, &batch_dimx,
              slab, tmp3, greens_function,
              (fftx_callback)complex_scaling,
              MY_FFTX_MODE_SUB | FFTX_PW_POINTWISE);

// iRDFT on the scaled data
tmp4 = fftx_create_temp_real(rank, batch_pencils_dims);
plans[2] = fftx_plan_guru_dft_c2r(rank, padded_dims, batch_rank,
             &batch_dims, tmp3, tmp4,
             (fftx_callback)adaptive_sampling,
             MY_FFTX_MODE_SUB);

// copy out the rank-dimensional data cube in the right place in the output.
//the callback function copy_offset() is responsible for placing the
//samples in the right place in the output array.
plans[3] = fftx_plan_guru_copy_real(rank, out_dimx, tmp4,
               out, (fftx_callback)copy_offset,
               MY_FFTX_MODE_SUB);

// create the top level plan. this copies the sub-plan pointers.
p = fftx_plan_compose(numsubplans, plans, MY_FFTX_MODE_TOP);

// plan to be used with fftx_execute()
return p;

}
```

**Figure 5: FFTX plan written using a collection of subplans. In the context of our use case, FFTX can make the code much more portable than the current hand-tuned cuFFT implementation that requires the user to be a CUDA expert.**

[5] Jehoshua Bruck, Ching-Tien Ho, Shlomo Kipnis, Eli Upfal, and Derrick Weathersby. 1997. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on parallel and distributed systems* 8, 11 (1997), 1143–1156.

[6] Choong-Seock Chang, Seunghoe Ku, and H Weitzner. 2004. Numerical study of neoclassical plasma pedestal in a Tokamak geometry. *Physics of Plasmas* 11, 5 (2004), 2649–2667.

[7] Kenneth Czechowski, Casey Battaglino, Chris McClanahan, Kartik Iyer, P.-K. Yeung, and Richard 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

[8] Alan Edelman, Peter McCorquodale, and Sivan Toledo. 1999. The Future Fast Fourier Transform? *SIAM J. Sci. Comput.* 20, 3 (Jan. 1999), 1094–1114. https://doi.org/10.1137/S1064827597316266

[9] F. Franchetti et al. 2018. FFTX and SpectralPack: A First Look. In *IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC)*.

[10] F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, 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

[11] 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".

[12] Matteo Frigo and Steven G Johnson. 2012. FFTW: Fastest Fourier transform in the west. *Astrophysics Source Code Library* (2012).

[13] Amir Gholami, Judith Hill, Dhairya Malhotra, and George Biros. 2015. AccFFT: A library for distributed-memory FFT on CPU and GPU architectures. *arXiv preprint arXiv:1506.07933* (2015).

[14] Salman Habib, Adrian Pope, Hal Finkel, Nicholas Frontiere, Katrin Heitmann, David Daniel, Patricia Fasel, Vitali Morozov, George Zagaris, Tom Peterka, et al. 2016. HACC: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy* 42 (2016), 49–65.

[15] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. 2012. Simple and Practical Algorithm for Sparse Fourier Transform. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms* (Kyoto, Japan) *(SODA '12)*. Society for Industrial and Applied Mathematics, USA, 1183–1194.

[16] Fastest Fourier Transform in The East. 2018. FFTE. http://www.ffte.jp/.

[17] Anuva Kulkarni, Jelena Kovačević, and Franz Franchetti. 2020. Massive Scaling of MASSIF: Algorithm Development and Analysis for Simulation on GPUs. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (Geneva, Switzerland) *(PASC '20)*. Association for Computing Machinery, New York, NY, USA, Article 13, 10 pages. https://doi.org/10.1145/3394277.3401857

[18] R. A. Lebensohn. 2001. N-site modeling of a 3D viscoplastic polycrystal using fast Fourier transform. *Acta Materialia* 49, 14 (2001), 2723–2737.

[19] Peter Li and Luen-Fai Tam. 1987. Symmetric Green's Functions on Complete Manifolds. *American Journal of Mathematics* 109, 6 (1987), 1129–1154. http://www.jstor.org/stable/2374588

[20] P. McCorquodale, P. Colella, G. Balls, and S. Baden. 2006. A Local Corrections Algorithm for Solving Poisson's Equation inThree Dimensions. *Communications in Applied Mathematics and ComputationalScience* 2 (10 2006). https://doi.org/10.2140/camcos.2007.2.57

[21] 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.

[22] Akira Nukada, Yutaka Maruyama, and Satoshi Matsuoka. 2012. High Performance 3-D FFT Using Multiple CUDA GPUs. In *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units* (London, United Kingdom) *(GPGPU-5)*. Association for Computing Machinery, New York, NY, USA, 57–63. https://doi.org/10.1145/2159430.2159437

[23] Nvidia. 2018. NVidia Cuda Based FFT Library. https://developer.nvidia.com/cufft.

[24] 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

[25] Dmitry Pekurovsky. 2012. P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions. *SIAM Journal on Scientific Computing* 34, 4 (2012), C192–C209. https://doi.org/10.1137/11082748X

[26] Steve Plimpton, Roy Pollock, and Mark Stevens. 1997. Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. In *In Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*.

[27] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *ACM SIGPLAN Notices* 48, 6 (2013), 519–530.

[28] Timothy W Sirk, Stan Moore, and Eugene F Brown. 2013. Characteristics of thermal conductivity in classical water models. *The Journal of chemical physics* 138, 6 (2013), 064505.

[29] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. 1996. *MPI: The Complete Reference.* The MIT Press.

[30] TP Straatsma, EJ Bylaska, HJJ van Dam, N Govind, WA de Jong, K Kowalski, and M Valiev. 2011. Advances in scalable computational chemistry: NWChem. In *Annual Reports in Computational Chemistry*. Vol. 7. Elsevier, 151–177.

[31] 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

[32] 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 (8 2018).

https://doi.org/10.1016/j.actamat.2018.05.036

[33] Marat Valiev, Eric J Bylaska, Niranjan Govind, Karol Kowalski, Tjerk P Straatsma, Hubertus JJ Van Dam, Dunyou Wang, Jarek Nieplocha, Edoardo Apra, Theresa L Windus, et al. 2010. NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications* 181, 9 (2010), 1477–1489.

[34] J-L Vay, A Almgren, J Bell, L Ge, DP Grote, M Hogan, O Kononenko, R Lehe, A Myers, C Ng, et al. 2018. Warp-X: A new exascale computing platform for beam–plasma simulations. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* (2018).

[35] Martin Vetterli, Jelena Kovacevic, and Vivek K Goyal. 2014. *Foundations of Signal Processing*. Cambridge University Press. https://doi.org/10.1017/CBO9781139839099