FFTX for Micromechanical Stress-Strain Analysis

Anuva Kulkarni
Electrical and Computer Engineering
Carnegie Mellon University
Email: anuvak@andrew.cmu.edu

Daniele G. Spampinato
Electrical and Computer Engineering
Carnegie Mellon University
Email: spampinato@cmu.edu

Franz Franchetti
Electrical and Computer Engineering
Carnegie Mellon University
Email: franzf@cmu.edu

***Abstract*—Porting scientific simulations to heterogeneous platforms requires complex algorithmic and optimization strategies to overcome memory and communication bottlenecks. Such operations are inexpressible using traditional libraries (e.g., FFTW for spectral methods) and difficult to optimize by hand for various hardware platforms. In this work, we use our GPU-adapted stress-strain analysis method to show how FFTX, a new API that extends FFTW, can be used to express our algorithm without worrying about code optimization, which is handled by a backend code generator.**

I. Introduction

Large-scale simulations involving parallel Fast Fourier Transforms (FFTs) on distributed heterogeneous systems suffer from bottlenecks due to all-to-all communication and extreme memory requirements. Algorithm innovations such as compression or pruning are therefore needed to exploit data sparsity or symmetries to reduce data movement and efficiently perform large parallel FFTs. However, while doing so, it is difficult to optimize these operations across various heterogenous platforms, as the best strategy depends on the computing platform and tradeoffs with needs of the overall simulation. The FFTW interface [1] is not able to express some of these special pruning and sampling patterns and hence the user cannot leverage the highly optimized FFTW library for parallel computations. The FFTX API [2] is designed to provide a familiar interface for expressing c
of multidimensional data to well-optimized F
while a SPIRAL-based code generation bac
optimizations across various hardware platfo
as a domain specific language (DSL) for th
effectively decoupling algorithm specificatic
mization.

In this work, we consider the Microm
sis of Stress-Strain Inhomogeneities with F
(MASSIF), an FFT-based stress-strain simul
composites [4], [5], [6], [7]. The method
crostructure on a regular grid and uses FFTs
differential equations (PDE) using Green'
FFTs are used to perform convolutions with a Green's function tensor. Convolution in each iteration requires computation of three dimensional (3D) FFTs of tensor fields, thus requiring extensive storage. The maximum size simulated currently with MPI and FFTW [7] is $1024 \times 1024 \times 1024$ with a memory requirement of more than 2TB (computed using knowledge of 3D double precision variables computed and stored by MASSIF) [7], and for larger sizes, the memory required is prohibitively large. Scaling and accelerating the MASSIF simulation is part of the DoD HPC PETTT project and has a wide range of applications where micromechanical properties of polycrystals are studied.

We describe our new algorithmic solution to implement MASSIF using Graphical Processing Units (GPUs) and show how FFTX can be used to express it. GPUs provide tremendous compute power, but simply do not have the on-chip memory to store large 3D FFTs. Hence, our proposed solution [8] reduces memory requirements using domain decomposition and adaptive sampling such that GPUs can be used for domain-local computation. The 3D volume is decomposed into domains and a domain-local convolution followed by adaptive sampling is performed under the constraints of GPU memory while preserving accuracy of the overall convolution result (across the full volume). The full 3D volume is not materialized on the GPU during FFT computation. Our approach exploits the sparse structure of the data and symmetries of the Green's function and thus involves expressing complicated 3D sampling patterns. FFTX functionalities allows us to express these patterns for cubical grains.

In the background section, we briefly describe our proposed method for porting MASSIF to GPUs. Using this example, the following section shows how FFTX is used to express

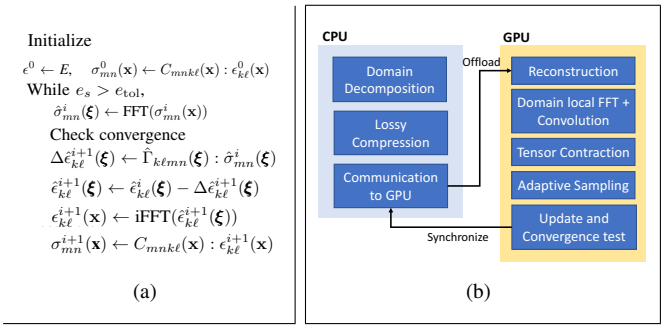

Fig. 1. (a) Original MASSIF method computes FFT on full 3D volume. (b) Proposed domain decomposition algorithm for porting MASSIF to GPUs operates on local domains to reduce memory requirement [8].

II. Background

The original FORTRAN scheme [4] is a fixed-point iterative numerical method on a 3D grid and is outlined in Fig. 1(a). Consider a simple example of a $n \times n \times n$ microstructure grid. At each grid point, the stress field is a 3×3 tensor, i.e., stress

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

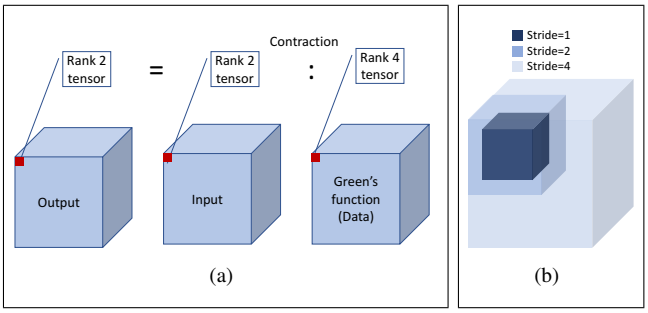

Fig. 2. (a) Tensor contraction. The cubes represent the 3D data grid. A rank-2 or rank-4 tensor is present at each point (x,y,z). (b) An example of an adaptive sampling pattern in a 3D grid.

at each (x,y,z) has nine components. The flow of our proposed domain decomposition-based method is as shown in Fig. 1(b). Restricting ourselves to regular domains for simplicity, we assume $k \times k \times k$ cubical grains. Our method decomposes the 3D volume into $k \times k \times k$ domains to be operated on separately. In a single iteration, we compute domain-local FFTs of each of the nine tensor components as discussed in [8] followed by tensor contraction with the rank-4 Green's function operator (see Fig. 2(a)).

Since the Green's function is rapidly decaying in space domain, convolution results in a low-magnitude and low-varying field in the volume outside the domain. This important observation allows us to develop an adaptive sampling scheme to compress the convolution result and further reduce memory required by using different resolutions in different parts of the volume. An example of a required sampling pattern is seen in Fig. 2(b). We a multi-resolution octree-based sampling strategy to reduce the memory footprint of the domain-local convolution result. Along with adaptive sampling, the symmetries in the Green's function operator can be exploited to further reduce the memory footprint of the overall algorithm.

III. FFTX PROGRAM

In this section, we outline the FFTX program for our proposed algorithm. FFTX allows us to express the specifications of the algorithm as a plan composed of various sub-plans. The FFTX code generation back-end optimizes across the sub-plans, which was not possible with FFTW. [2] contains more information on FFTX objects and callback functions.

Fig. 3 outlines the creation of sub-plans. The sub-plans are provided a number of parameters required to describe the data involved in a given computation. These descriptors include information about the dimensionality, strides and offsets of the input and output data. The `fftx_sample` sub-plan takes in various strides and copies to the same output multiple times. This represents the multi-resolution sampling operation around the cubic domain as seen in Fig. 2(b).

IV. CONCLUSION

The high-performance execution of FFT-based simulations on emerging platforms requires algorithm designs that address memory issues and communication bottlenecks. In this work, we use FFTX to express one such algorithm design to port

```
#define NUMSUBPLANS 7
fftx_plan subplans[NUMSUBPLANS];
fftx_plan p; // top-level plan
//... Initialize ....
// Create zero-initialized temporary arrays
//tmp1, tmp2, tmp3 and tmp4
//They are n x n x n arrays with 3 x 3 tensor at each point

// copy k x k x k input domain into n x n x n tmp1
subplans[0] = fftx_copy_plan(domain, tmp1);// (from,to)

// DFT on the input
subplans[1] = fftx_dft_plan(tmp1);

//Tensor contraction
//In this case we know that output size is the same as tmp2
subplans[2] = fftx_tensor_contraction_plan(tmp2, data,
     tmp3, dimensions_to_contract);//(in,data,out,info)

// iDFT on the contracted output
subplans[3] = fftx_inverse_dft_plan(tmp3, tmp4);

//The next plans apply adaptive sampling
subplans[4] = fftx_plan_sample(tmp4, final_output, offset0,
     s0); // (from,to,offset,stride)
subplans[5] = fftx_plan_sample(tmp4, final_output, offset1,
     s1);
subplans[6] = fftx_plan_sample(tmp4, final_output, offset2,
     s2);

// create the top level plan. this copies the sub-plan
     pointers
p = fftx_plan_compose(NUMSUBPLANS, subplans);

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

Fig. 3. FFTX plan for porting MASSIF to GPUs.

the MASSIF code to GPUs. FFTX's SPIRAL-based code-generation back-end helps address fundamental problems such as (1) performance portability across the ever-changing landscape of parallel platforms, and (2) verifiable correctness of sophisticated floating-point code. SPIRAL's ability to automatically map computational kernels across a wide range of computing platforms to highly efficient code is a big advantage of FFTX.

REFERENCES

[1] M. Frigo *et al.*, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

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

[3] F. Franchetti *et al.*, "SPIRAL: Extreme Performance Portability," *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1935–1968, Nov 2018.

[4] H. Moulinec *et al.*, "A numerical method for computing the overall response of nonlinear composites with complex microstructure," *Computer methods in applied mechanics and engineering*, vol. 157, no. 1-2, pp. 69–94, 1998.

[5] J. Michel *et al.*, "A computational method based on augmented Lagrangians and fast Fourier Transforms for composites with high contrast," *CMES - Computer Modeling in Engineering and Sciences*, vol. 1, pp. 79–88, 01 2000.

[6] R. A. Lebensohn, "N-site modeling of a 3D viscoplastic polycrystal using fast Fourier transform," *Acta Materialia*, vol. 49, no. 14, pp. 2723–2737, 2001.

[7] V. Tari *et al.*, "Validation of micro-mechanical fft-based simulations using high energy diffraction microscopy on ti-7al," *Acta Materialia*, vol. 154, 8 2018.

[8] A. Kulkarni *et al.*, "Large-scale algorithm design for parallel FFT-based simulations on GPUs," in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Nov 2018, pp. 301–305.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE