Accelerating 2D FFT: Exploit GPU Tensor Cores through Mixed-Precision

Extended Abstract

Xiaohe Cheng
Hong Kong University of Science and Technology
xchengaj@connect.ust.hk

Anumeena Sorna
National Institute of Technology Tiruchirappalli, India
108115011@nitt.edu

Eduardo D'Azevedo (advisor)
Oak Ridge National Laboratory
dazevedoef@ornl.gov

Kwai Wong (advisor)
University of Tennessee
kwong@utk.edu

Stanimire Tomov (advisor)
University of Tennessee
tomov@icl.utk.edu

ABSTRACT

The two-dimensional Fourier Transform is a widely-used computational kernel in many HPC applications. The popular NVIDIA cuFFT library provides a simple interface to compute 2D FFT on GPUs, but it's yet to utilize the recent hardware advancement in half-precision floating-point arithmetic. In this poster, we propose a mixed-precision method to accelerate 2D FFT by exploiting the FP16 matrix-multiply-and-accumulate units on the newest GPU architecture, known as tensor cores. We achieve a balance between speed and accuracy by dynamically splitting the single-precision input data into two half-precision operands and performing FFT separately. We present a CUDA-based implementation that achieves 3-digit more accuracy than half-precision cuFFT. We also demonstrate the stability and scalability of our approach and conclude that it attains high accuracy with tolerable splitting overhead.

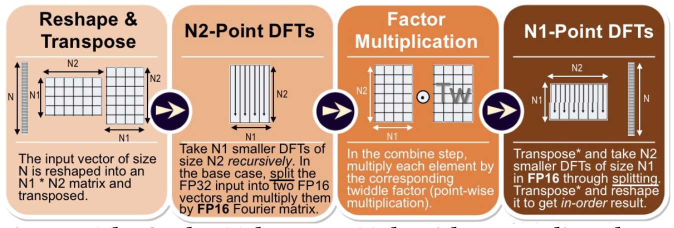
KEYWORDS

Fast Fourier Transform, GPU Tensor Core, CUDA, Mixed-Precision

1 INTRODUCTION

The two-dimensional Fourier transform has been extensively used in many HPC applications, including radar image formulation, big integer multiplication, and quantum cluster simulation [2, 6, 8]. The NVIDIA CUDA Fast Fourier Transform library (cuFFT) provides some simple APIs that perform 2D FFT on the graphics processing units (GPUs) and achieve 10x performance improvement over pure CPU implementations [7]. The tensor cores on recent Volta GPU architecture considerably increase half-precision floating-point compute throughput, but this has not been fully utilized by cuFFT library, because FP16 calculation does not fulfill the accuracy requirements of most scientific applications.

To exploit the fast half-precision arithmetic on tensor cores, we propose a mixed-precision 2D FFT that dynamically splits every FP32 input into two FP16 elements and performs matrix multiplication in half-precision. This extended abstract will introduce the distinctive characteristics of tensor cores and fast Fourier transform, and explain how these characteristics can be leveraged to accelerate 2D FFT. Then in section 4 we evaluate our CUDA-based implementation through experiments on NVIDIA®Tesla®V100 GPU. We will demonstrate that our implementation of accelerated FFT achieves desired accuracy with tolerable splitting overhead.



The input vector of size N is reshaped into an N1 * N2 matrix and transposed.

Take N1 smaller DFTs of size N2 *recursively*. In the base case, split the FP32 input into two FP16 vectors and multiply them by **FP16** Fourier matrix.

In the combine step, multiply each element by the corresponding twiddle factor (point-wise multiplication).

Transpose* and take N2 smaller DFTs of size N1 in **FP16** through splitting. Transpose* and reshape it to get *in-order* result.

Figure 1: The Cooley-Tukey 1D FFT algorithm. We adjust the N1-point DFT kernel to avoid taking transpose in last step.

2 MOTIVATION

A critical feature in the new Volta GPU architecture is tensor core, the matrix-multiply-and-accumulate unit that significantly accelerates half-precision arithmetic. The Tesla®V100 GPU contains 640 tensor cores and delivers up to 125 TFLOPS in FP16 matrix multiplication [1]. However, the narrow dynamic range of FP16 numbers limits its adoption by scientific computations such as 2D fast Fourier transform. This accuracy limitation incentivizes us to design a mixed-precision solution, which is traditionally known as emulating a double precision number with two single precision numbers [3].

We apply this method to 2D FFT due to its important role in HPC applications as well as its numerical properties. The implementation of 2D FFT involves multiplication of large matrices; the linearity of 2D FFT allows trivial combination of computational results, and the numerical stability provides an accuracy guarantee.

3 METHODOLOGY

Our goal is to compute two-dimensional discrete Fourier transform, defined as:

$$Y = F_m * X * F_n^{\mathsf{T}},$$

where X is the input matrix of size $m{*}n$, and F_m, F_n are the $m{*}m$ and $n * n$ Fourier matrix, respectively. We implement it by performing two 1D FFTs on each column, together with a matrix transpose in between.

We adopt the well-known Cooley-Tukey algorithm to compute 1D FFT, which is illustrated in Figure 1. It reshapes the input length-N vector to a N1*N2 matrix and performs per-column 1D FFT recursively [5]. Here N1 is chosen to be four as the 4*4 Fourier matrix can be accurately represented in FP16. The cuBLAS GEMM API is called in the base case of recursion and the N1-point FFT to

ALGORITHM 1: Dynamic Splitting pseudo-code

```
Data: $X = [\vec{v}_1, ..., \vec{v}_n]$ ;                    ▷ Data matrix, $\{x_{i,j}\}$
Output:
$X_{hi} = [\vec{v}_{h1}, ..., \vec{v}_{hn}]$ ;               ▷ Higher half vectors
$X_{lo} = [\vec{v}_{l1}, ..., \vec{v}_{ln}]$ ;               ▷ Lower half vectors
$\alpha = [a_1, ..., a_n]$ ;          ▷ Scaling factors of higher part
$\beta = [b_1, ..., b_n]$ ;           ▷ Scaling factors of lower part
for $j \leftarrow 1$ to $n$ do
    /* Parallel For implemented with CUDA        */
    $a_j = \max_{i=1}^{m} \|x_{i,j}\|$;
    $\vec{v}_{hj} = \vec{v}_j / a_j$ ;              ▷ Cast scaled data to FP16
    $\vec{r}_j = \vec{v}_j - \vec{v}_{hj} * a_j$ ;                 ▷ Residual, $\{\tilde{x}_{i,j}\}$
    $b_j = \max_{i=1}^{m} \|\tilde{x}_{i,j}\|$;
    $\vec{v}_{lj} = \vec{r}_j / b_j$;               ▷ Cast scaled data to FP16
end
```

multiply the Fourier matrix and data matrix. The essence of our method is to split every column vector into two FP16 vectors before matrix multiplication, so that the multiplication can be carried out in half precision. Algorithm 1 summarizes how the splitting is performed.

4 RESULTS

We evaluate our implementation by testing its performance on one NVIDIA®Tesla®V100 GPU. CUDA events and the nvprof profiler are used to measure the execution time and analyze the splitting overhead. We also calculate its deviation from the FP32 cuFFT results and compare the accuracy with FP16 cuFFT.

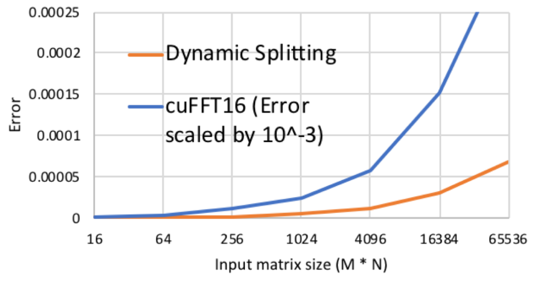


Figure 2: The relative error of 2D FFT at different input sizes.

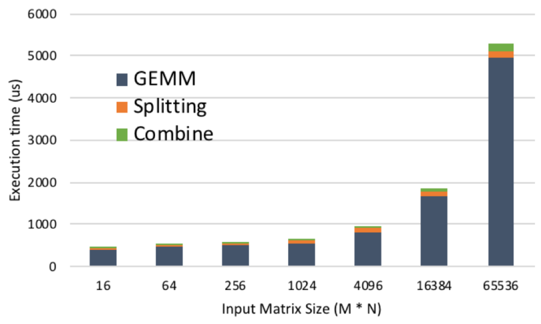


Figure 3: Execution time breakdown at different input sizes. About 90% of total time is spent on calling the cuBLAS batched matrix multiplication API.

Figure 2 compares the relative error of half-precision cuFFT and our implementation. It shows that the dynamic splitting method achieves 3-digit better accuracy, so the method may satisfy the requirements of many scientific applications. Figure 3 illustrates the proportion of time spent on different operations, and indicates that the overhead of data splitting is tolerable.

In addition, we measure the accuracy of the mixed-precision method under various settings. The results and implications are presented on the poster.

5 CONCLUSIONS AND FUTURE WORK

We have designed and implemented a FP32-FP16 mixed-precision 2D FFT that takes advantage of the recent tensor core hardware. The dynamic splitting method effectively emulates single-precision calculation and produces highly accurate results from a variety of inputs.

The speed of current cuBLAS-based implementation is inferior to cuFFT APIs, but we expect it to gain advantage as the input size grows, because the tensor core can be fully utilized and the setup cost can be amortized. In the future we will consider further optimizing the implementation by customizing the transpose and matrix multiplication kernels [4]. We may also design an auto-tuning splitting algorithm that supports ill-conditioned inputs and causes less overhead.

ACKNOWLEDGMENTS

This project was performed under the Research Experience for Undergraduates (REU) award sponsored by NSF under Contract No. 165902, with additional support from the JICS at University of Tennessee Knoxville. It used allocations from the Extreme Science and Engineering Discovery Environment (XSEDE). This research is also sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725, and on technical workstations donated by the BP High Performance Computing Team.

REFERENCES

[1] J. Appleyard and S. Yokim. 2017. Programming Tensor Cores in CUDA 9. Retrieved July 27, 2018 from https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/
[2] H. Bantikyan. 2014. Big integer multiplication with CUDA FFT (cuFFT) library. *world* 2, 11 (2014).
[3] D. Göddeke, R. Strzodka, and S. Turek. 2007. Performance and accuracy of hardware-oriented native-, emulated-and mixed-precision solvers in FEM simulations. *International Journal of Parallel, Emergent and Distributed Systems* 22, 4 (2007), 221–256.
[4] M. Harris. 2013. An Efficient Matrix Transpose in CUDA C/C++. Retrieved July 26, 2018 from https://devblogs.nvidia.com/efficient-matrix-transpose-cuda-cc/
[5] V. Kumar, A. Grama, A. Gupta, and G. Karypis. 1994. *Introduction to parallel computing: design and analysis of algorithms*. Vol. 400. Benjamin/Cummings, Redwood City. 377–406 pages.
[6] R.W. Linderman, J. Corner, and S. Tucker. 2006. Real-time wide swath synthetic aperture radar image formation using embedded HPC. In *HPCMP Users Group Conferences, 2006*. IEEE, 244–251.
[7] Nvidia. 2018. cuFFT library. Retrieved July 28, 2018 from https://docs.nvidia.com/cuda/cufft/
[8] P. Staar, T.A. Maier, R. Solca, G. Fourestey, M.S. Summers, and T.C. Schulthess. 2013. Taking a quantum leap in time to solution for simulations of high-Tc superconductors. In *2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 1–11.