POSTER: FFT Blitz: The Tensor Cores Strike Back

Sultan Durrani
University of Illinois at Urbana-Champaign
sultand2@illinois.edu

Muhammad Saad Chughtai
Georgia Institute of Technology
chughtai@gatech.edu

Abdul Dakkak
University of Illinois at Urbana-Champaign
dakkak@illinois.edu

Wen-mei Hwu
University of Illinois at Urbana-Champaign
w-hwu@illinois.edu

Lawrence Rauchwerger
University of Illinois at Urbana-Champaign
rwerger@illinois.edu

Abstract

The fast Fourier Transform (FFT), a reduced-complexity formulation of the Discrete Fourier Transform (DFT), is an important tool in many areas of science and engineering. FFTW is a well-known package that follows this approach and is currently one of the fastest available implementations of the FFT. NVIDIA introduced its version of FFTW called cuFFT that achieves high performance on the GPUs. In this work we present a novel way to map the FFT algorithm on the newly introduced Tensor Cores by adapting the the Cooley-Tukey recursive FFT algorithm. We present four major types of optimizations that enhance the performance of our approach for varying FFT sizes and show that the approach consistently outperforms cuFFT with a speedup of about 15% to 250% on average.

CCS Concepts: • **Computing methodologies** → **Massively parallel algorithms**; **Vector / streaming algorithms**.

Keywords: DFT, FFT, GPU, Tensor Cores

1 Introduction

The FFT has been applied extensively to a wide range of applications ranging from image filtering to differential equation solvers. In this work, we focus on optimizing FFT for efficient execution on the Tensor Cores of recent Graphics Processing Units (GPUs) [1]. The main strength of the Tensor Cores is their very high compute throughput and operand supply bandwidth for modest-sized matrices. This work makes the following four primary, novel contributions in optimizing FFT algorithms for efficient execution on Tensor Cores in GPUs:

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PPoPP '21, February 27-March 3, 2021, Virtual Event, Republic of Korea
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8294-6/21/02.
https://doi.org/10.1145/3437801.3441623

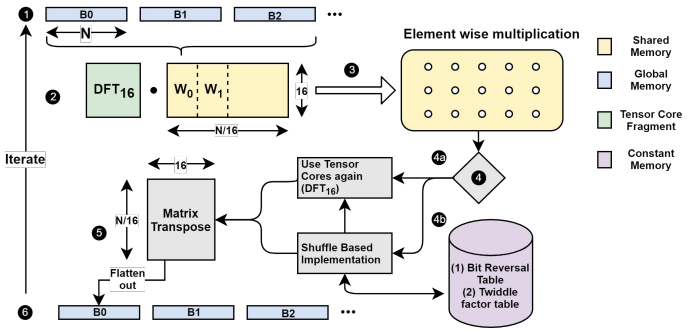

Figure 1. Concept Diagram for our optimized algorithmic approach

- An iterative version of the Cooley-Tukey FFT [2] algorithm which decomposes large DFT computations into sequences of parallel, small baseline DFT computations that can fit well into Tensor Cores.
- Warp-shuffle-XOR-based (WSX) FFT butterfly computation that allows the shuffle hardware to complement the Tensor Cores for some of the DFT computation generated by Cooley-Tukey that are too small to make good use of the Tensor Cores.
- Using constant memory to store bit reversal and twiddle factor lookup tables to drastically reduce the use of registers and improve occupancy of the SMs.
- Efficiently utilizing shared memory (on-chip scratchpad) to completely remove global memory (DRAM) accesses within the inner core of the algorithm and alleviate memory bandwidth bottleneck.

These key optimizations along with the more routine optimizations such as using CUDA streams and asynchronous data copies to overlap communication with computation allows our approach to consistently outperform cuFFT [3].

2 Design and Methodology

In this section we will dive into our overall algorithmic approach as depicted by the concept diagram in Figure 1. As the Tensor Core matrix cannot be a vectorized type we define two Fp16 matrices, one for the real part and the other for the

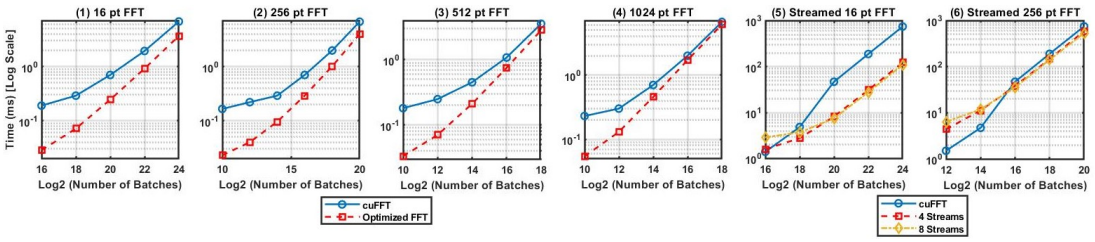

Figure 2

imaginary. The following formulation shows the complex matrix multiplication operation performed.

$$Y_{real} \leftarrow N_{img} \cdot X_{img} + (F_{real} \cdot X_{real})$$
$$Y_{img} \leftarrow F_{real} \cdot X_{img} + (F_{img} \cdot X_{real})$$

where F is the Fourier matrix, N is the negative Fourier matrix, X is the input while Y is the output. Next we will talk about our novel *WSX* based FFT butterfly approach for smaller FFT sizes. This approach has two parts i.e the bit reversal step and the butterfly exchange step. In the first step a bit reversal table is used which controls what input value a particular thread would load. Listing 1 shows the CUDA implementation for the second step where *cmul* is complex multiply and *smul* is scalar complex multiply.

Our optimized algorithm is divided into the following 6 stages.

(1) First, data movement and pre-computations take place. Input Data is moved from CPU host memory to GPU global memory, the constant memory is populated with the read-only bit reversal and the twiddle factor tables and the base Fourier matrices (Tensor Core tile sized) are generated.

(2) The main CUDA kernel is launched which computes the column wise DFT operation using the Tensor Cores by effectively utilizing shared memory for faster accesses. For smaller sizes the *WSX* algorithm is directly used.

(3) Now a per thread element wise complex number multiplication is done with the resultant output.

(4) We now move towards a row based FFT computation. A decision is made whether to use the Tensor Cores or the *WSX* algorithm. An appropriate element wise twiddle factor computation is done where required.

(5) We now do a matrix transpose to dump the final output back to global memory. This can be achieved via the Tensor Cores or a per thread approach.

(6) Lastly, we move towards the next iteration and continue doing so until all batches are exhausted.

3 Experimental Results

In all of our FFT experiments we have used NVIDIA's Volta V100 Titan V GPU. We use Fp16 supported cufftXt library for comparisons which does not support more than 4 billion elements and is restricted to powers of two. Figure 2 shows the line graphs for our results. For the first four graphs the kernel execution time was included while for the stream graphs the host to device copying time was also included. It is clear that our implementations consistently outperform cuFFT while not compromising on precision. Moreover, there is a clear speedup when using streams for higher batch sizes although the stream creation and multiple kernel launch overhead negatively affects performance for lower batches.

Listing 1 Shuffle XOR based FFT butterfly exchange step

```
__inline__ __device__ half2 btfy(half2 a,int num){
    unsigned lane_id,j,x,y,ind;
    asm volatile ("mov.u32 %0, %laneid;" : "=r"(lane_id));
    a = fft_2_pt(a,2);
    if(num==2) return a;
    for(int i=4;i<=num;i*=2){
        j = (i/2); x = (2*!((lane_id%i)/j))-1;
        y = ((lane_id%i)/j); ind = (lane_id%j)+(j-1);
        half2 w1 = __shfl_xor_sync(0xffffffff,a,j);
        half2 b1 = __hadd2(w1,cmul(TF_table[ind],smul(a,x)));
        half2 b2 = __hadd2(a,cmul(TF_table[ind],w1));
        a = __hadd2(smul(b1,y),smul(b2,!y));
    }
    return a;}
```

4 Conclusion and Future Work

GPUs will continue to evolve in the near future and we might see the rise in popularity of additional domain specific architecture features. It is therefore very important that we think of ways of using them without being limited to their original, narrowly intended usage like we successfully utilized Deep Learning intended Tensor Cores for FFT computations.

Nevertheless, in the future we would like to test the Fp32/64 Tensor Cores and sparse matrix support in the newer Ampere GPUs [4] and see how our implementation scales with increasing bit precision. Also currently we only take into account FFT sizes of powers of 2. There have been specialized algorithms like Rader's [5] which compute FFT for prime sizes, but it is currently unclear how these algorithms should be mapped on the Tensor Cores.

References

[1] Tesla NVIDIA. V100 gpu architecture, 2017.
[2] Tuckey Cooley. An algorithm for machine calculation of complex fourier series, 1965.
[3] CUDA NVIDIA. Programming guide, cufft library user guides.
[4] NVIDIA. A100 gpu architecture, 2020.
[5] Charles M Rader. Discrete fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968.