Accelerating Fast Fourier Transform with half-precision floating point hardware on GPUs

Anumeena Sorna & Xiaohe Cheng

Mentors: Eduardo D'Azevedo, Kwai Wong, and Stanimire Tomov

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

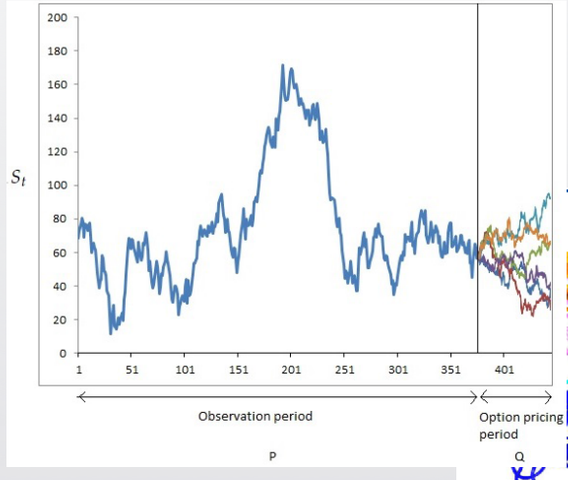
THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

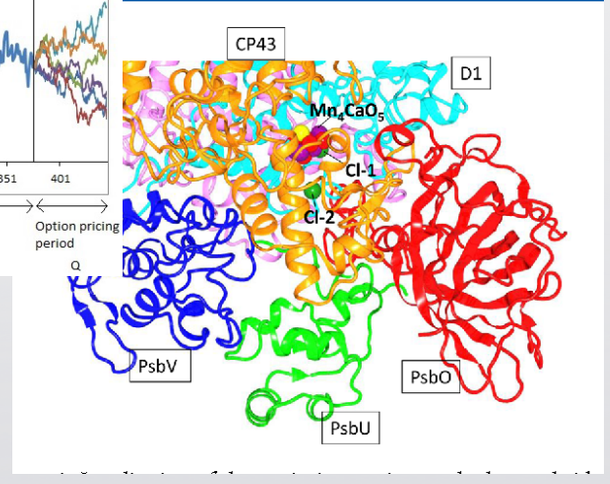
Outline

- Background
- Motivation
- Methodology
- Implementation
- Experimental Results
- Performance Analysis
- Conclusions
- Future Work

Background

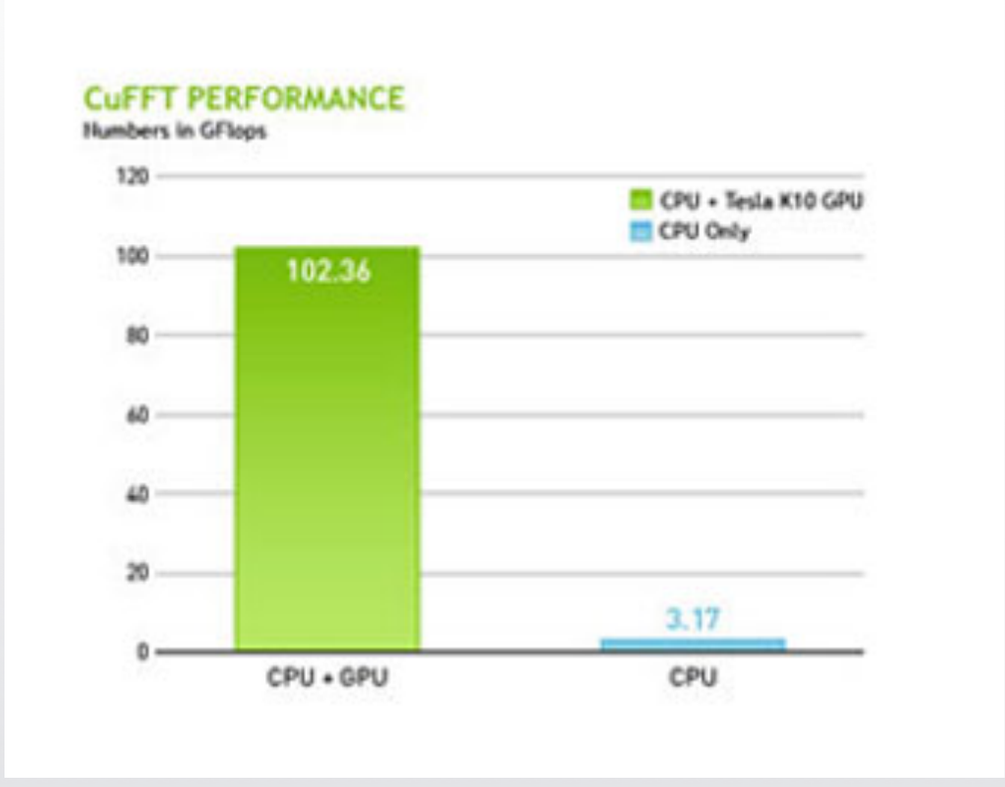
- Fast Fourier Transform is a useful tool in many high performance computing applications
  - Convolution
  - Image processing
  - Protein structure analysis
  - Option pricing
  - Computer Vision
  - Molecular dynamics





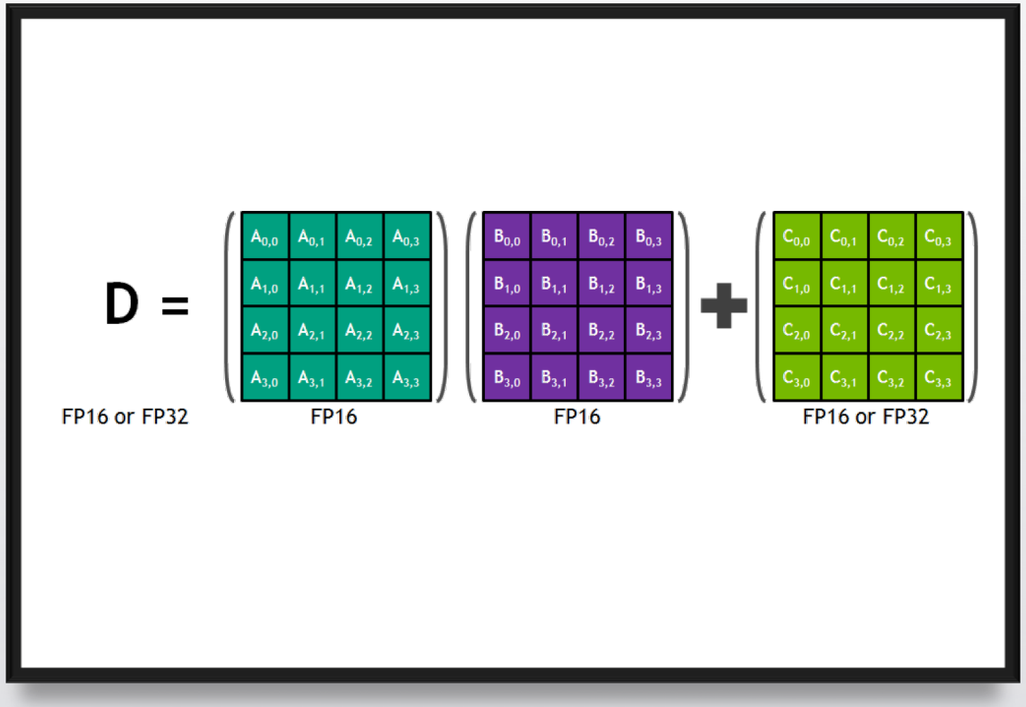
Background

- FFT computation has high parallelism
- Utilize parallel architecture: GPUs, NVIDIA CUDA
- State of the art: NVIDIA CUDA Fast Fourier Transform library (cuFFT)



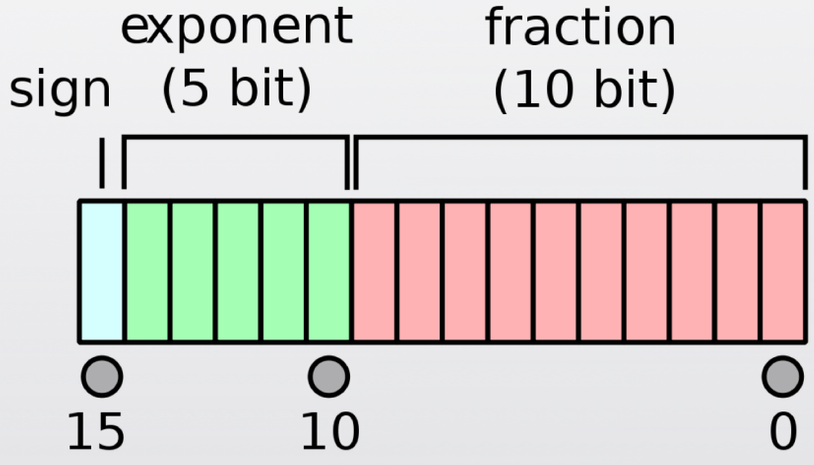
Motivation

- Tensor cores in new Volta GPU architecture
- Deliver up to 125 Tensor TFLOPS
- Speedup FFT calculation if fully utilized



Motivation

- cuFFT has yet to utilize tensor cores due to accuracy limitations
- Half precision number:
  - 65504 (max half precision)
  - 6.10352×10^{-5} (minimum positive normal)
  - 1.0009765625 (next smallest float after 1)
- The narrow dynamic range does not satisfy the requirements of scientific applications



Motivation

- Mixed-precision approach in computational physics
  - Single precision calculation is faster than double
  - Mix 32-bit and 64-bit arithmetic for acceleration
  - Design algorithm to maintain 64-bit accuracy
- Favorable properties of cuFFT
  - Linearity: Straightforward splitting and combination
  - Numerical stability: preserve norm, avoid error propagation

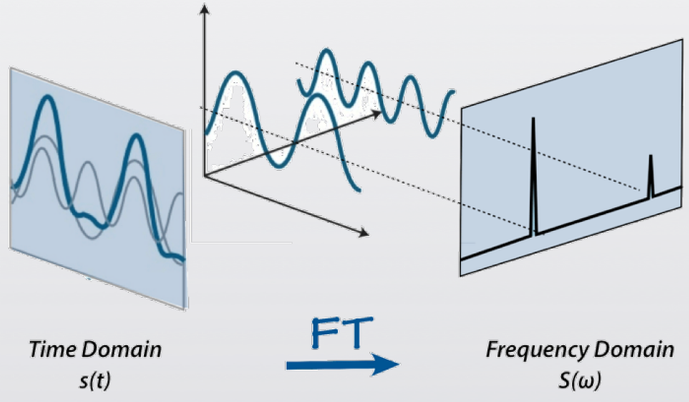
Methodology - FFT

- Every Signal can be broken down into signals of different frequencies
- The Discrete Fourier transform converts a time domain signal to a frequency domain signal:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\left(\frac{2\pi}{N}\right)nk}$$

- Inverse:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cdot e^{j\left(\frac{2\pi}{N}\right)nk}$$



Methodology - FFT

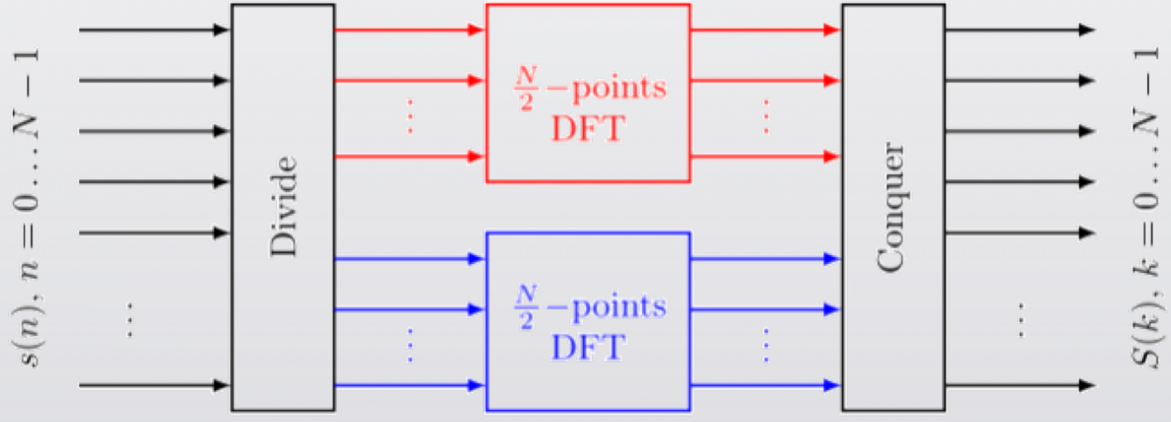
$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\left(\frac{2\pi}{N}\right)nk}$$

- It can be expressed as matrix multiplication:

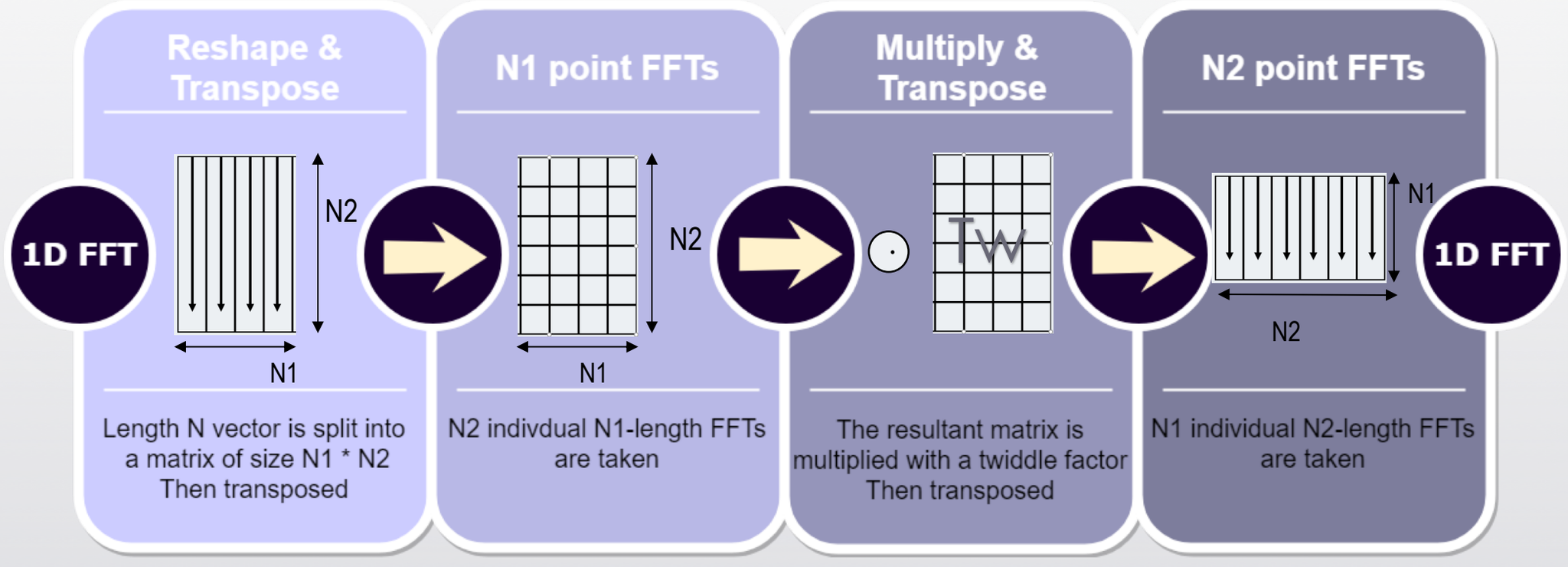
$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ \vdots \\ X[N-1] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^1 & W_N^2 & \cdots & W_N^{(N-1)} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{(N-1)} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)^2} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix}$$

Methodology – FFT ..

- Fast Fourier transform: Reduces number of computations
- Computations are now of order O(nlogn)
- Divide and conquer, recursion



Methodology – FFT ..



We apply the well-known FFT algorithm

Methodology – FFT ..

N1 is chosen to be 4, as 4*4 Fourier matrix is accurately representable in FP16

$$F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

$$F_{4real} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & -1 & 0 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix}$$

$$F_{4imag} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

Methodology - Splitting

- Matrix multiplication is needed in:
  - The base case of recursion
  - The N1-point FFT
- We split the FP32 input before multiplication

$$X(:) = \alpha X_{hi}(:) + \beta X_{lo}(:)$$

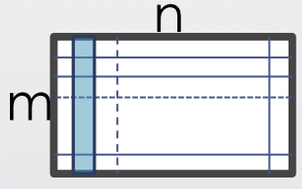
- And carry out matrix multiplication in FP16
- Then rescale the results and combine them together

Implementation

- We implemented the twiddle multiplication, transpose, splitting, and combine kernels using CUDA to utilize parallel hardware
- The splitting factors are dynamically determined

$$\alpha_j = \max_i |x_{ij}|$$

$$\beta_j = \max_i |\tilde{x}_{ij}|$$



- The multiplication is performed by calling cuBLAS API
- Batch execution is supported by handling each input independently and in parallel

Implementation ..

- The classical 4-step algorithm requires 3 matrix transpose at every level
- We employ the transpose property to avoid 2 of them

$$(F \cdot X^T)^T = X \cdot F^T$$

- Note that Fourier matrix is symmetric for N=4
- We adjust the transpose and combine kernel, and reverse the order of operands in matrix multiplication

Experimental Results

- We tested the program on a NVIDIA Tesla V100 GPU.
- It has 640 tensor cores and 5120 CUDA cores, with 16 GB GPU memory and 900G GB/sec memory bandwidth.

Double-Precision Performance	**7 TFLOPS**
Single-Precision Performance	**14 TFLOPS**
Tensor Performance	**112 TFLOPS**

Experimental Results ..

- The error statistics indicate that our implementation preserves high accuracy, even with growing input sizes

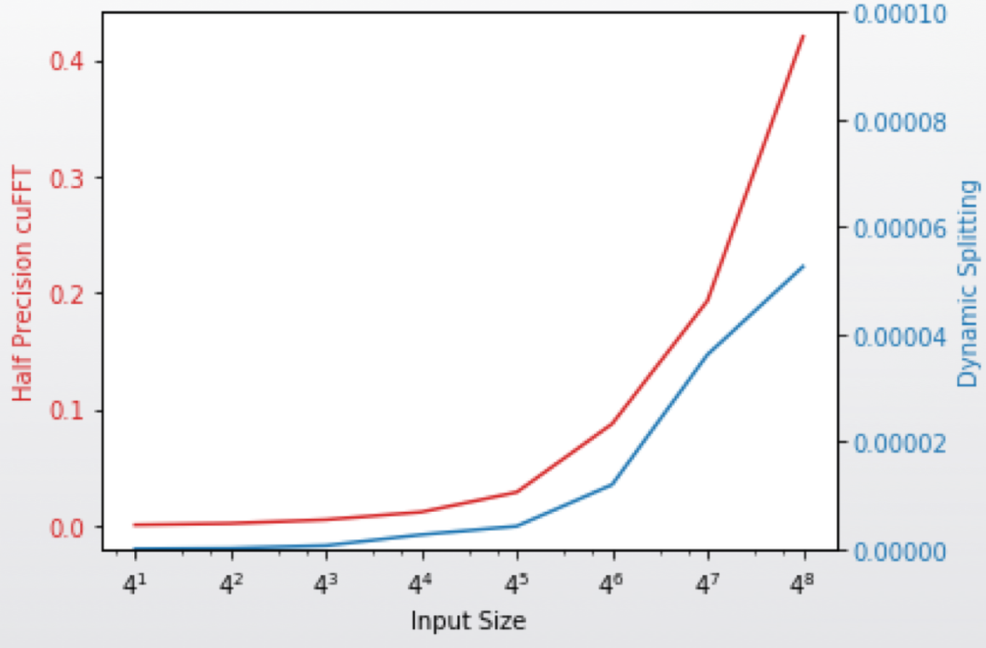


Figure1. Accuracy of half-precision cuFFT and our implementation. The average error of FP16 cuFFT is 7322x greater than our method.

Experimental Results ..

- The dynamic splitting method preserves high accuracy over a wide range of inputs.

Data range	FP16 cuFFT	Dynamic Splitting
$[-10^{-9}, 10^{-9}]$	14.6843805313	0.0000568428
$[-10^{-6}, 10^{-6}]$	0.5265535712	0.0000029240
$[-10^{-3}, 10^{-3}]$	0.0126493834	0.0000029515
[-1.0, 1.0]	0.0126134995	0.0000029261
$[-10^{2}, 10^{2}]$	0.0125578260	0.0000029014
$[-10^{4}, 10^{4}]$	N/A	0.0000028950
$[-10^{10}, 10^{10}]$	N/A	0.0000030171

Table1. Relative error of half-precision cuFFT and our implementation at different input data ranges.

Experimental Results ..

- Compared with matrix multiplication, the time spent on splitting and combine is not significant.

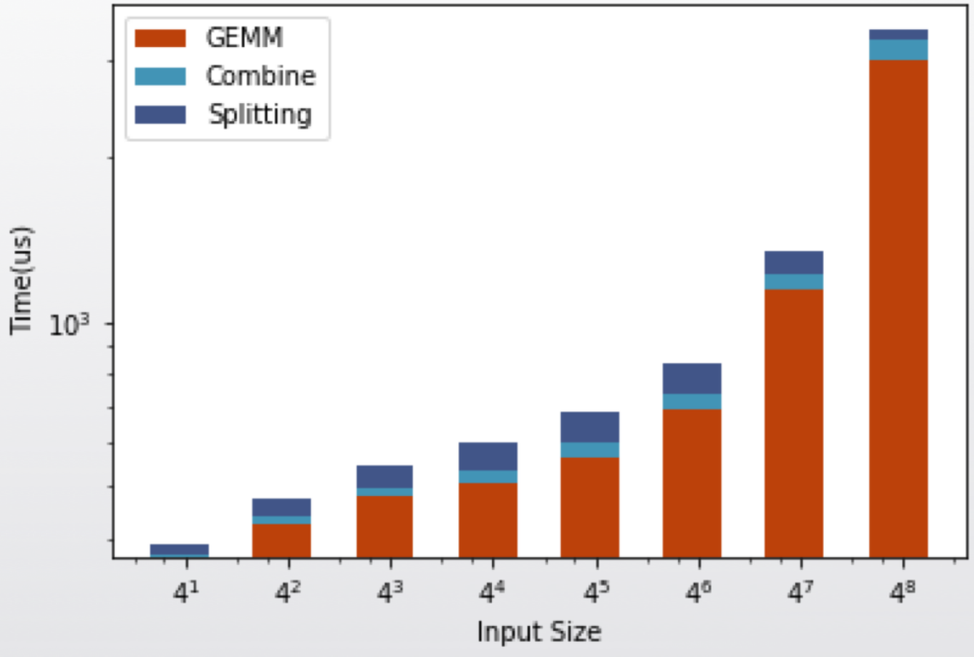


Figure1. Execution time breakdown at different input sizes. The matrix multiplication (by calling cublasGemmStridedBatchedEx) consumes around 90% of total time.

Performance Analysis

- The speed of our implementation is inferior to cuFFT library. It can be optimized by implementing customized kernels. We also expect it to gain advantage with large input sizes, as tensor cores can be fully utilized and setup cost can be amortized.

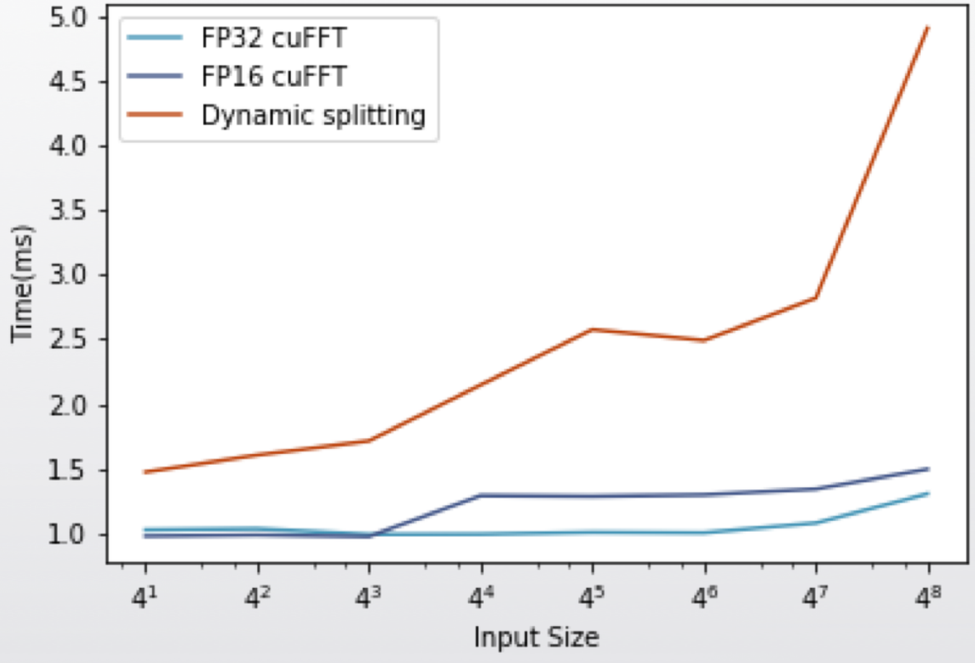


Figure1. Average execution time of three different FFTs with growing input sizes.

Performance Analysis ..

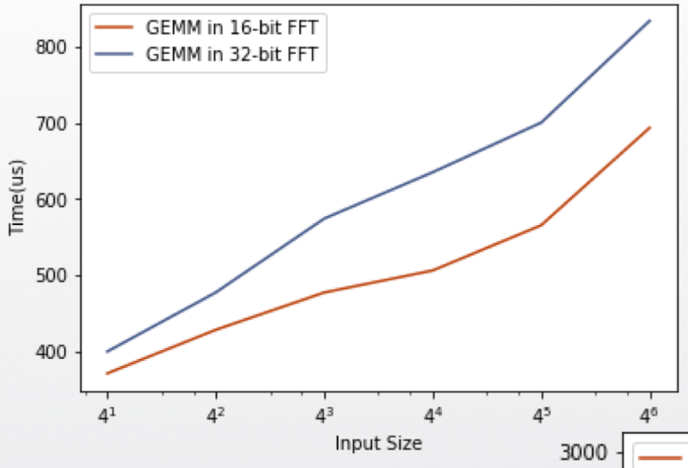
- Current implementation can handle a large variety of inputs. The relative error exceeds 0.1% or the program throws error when:
  - Input data range $\geq 3 * 10^{10}$; or
  - Input data range $\leq 5 * 10^{-11}$
- These numbers are closed to the edge of the dynamic range of single precision number.
- The range may be further enlarged by pre-scaling all input numbers

Conclusions

- Our dynamic splitting method performs matrix multiplication in half precision. It utilizes the tensor cores and efficiently computes fast Fourier transform.
- The implementation effectively emulates single precision calculation, and produces highly accurate results from a variety of inputs.

Future Work

- The time spent on 16-bit GEMM grows quickly when input size exceeds 16384.
- This may due to the inefficient cuBLAS implementation.
- We'll consider designing a customized GEMM kernel that is optimized for this input size.



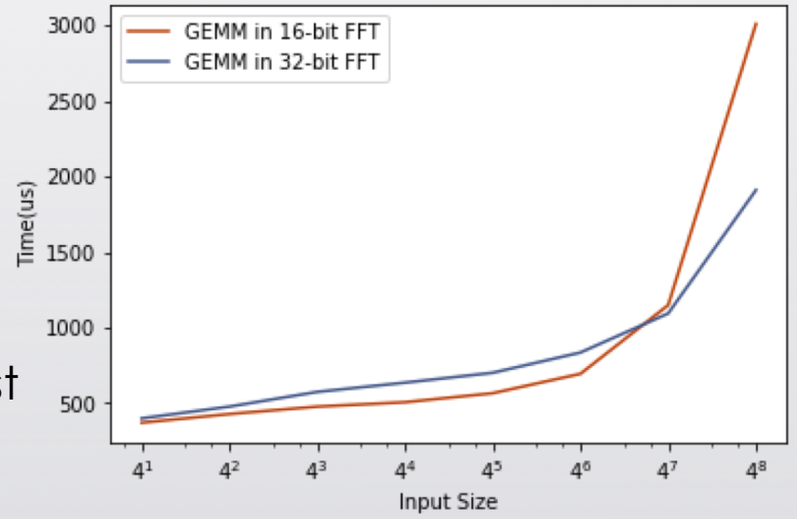
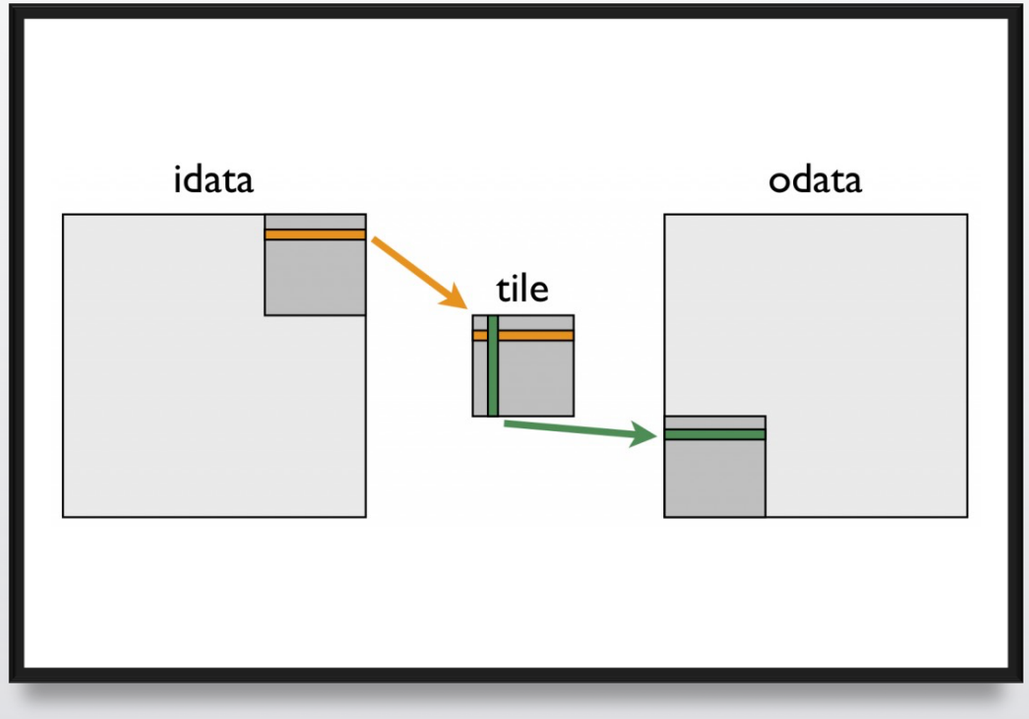


Figure1&2. Time cost by cuBLAS GEMM kernel in 16-bit and 32-bit execution.

Future Work ..

- Our implementation of matrix transpose kernel has yet to take advantage of the shared memory.
- It can be accelerated by applying the “tiled” design.



Future Work ..

- A more sophisticated splitting algorithm may be designed.
- Adding support for ill-conditioned inputs
- Optimizing overhead by performing static splitting when applicable
- Preserving more information through bit copying

References

- [1] J. Kong and S. Yu, "Fourier transform infrared spectroscopic analysis of protein secondary structures," *Acta biochimica et biophysica Sinica*, vol. 39, no. 8, pp. 549–559, 2007.
- [2] T. R. Hurd and Z. Zhou, "A fourier transform method for spread option pricing," *SIAM Journal on Financial Mathematics*, vol. 1, no. 1, pp. 142–157, 2010.
- [3] C.Guo,Q.Ma,andL.Zhang,"Spatio-temporalsaliencydetectionusing phase spectrum of quaternion fourier transform," 2008.
- [4] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, "Scalable molecular dynamics with namd," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.
- [5] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov, "Accelerating scientific computations with mixed precision algorithms," *Computer Physics Communications*, vol. 180, no. 12, pp. 2526–2533, 2009.

References ..

- [6] NVIDIA. (2018, Jul.) Cuda fast fourier transform library. [Online]. Available: https://docs.nvidia.com/cuda/cublas/index.html
- [7] J. Appleyard and S. Yokim. (2017, Oct.) Programming tensor cores in cuda 9. [Online]. Available: https://devblogs.nvidia.com/programming- tensor-cores-cuda-9/
- [8] V.Kumar,A.Grama,A.Gupta,andG.Karypis,*Introductiontoparallel computing: design and analysis of algorithms*. Benjamin/Cummings Redwood City, 1994, vol. 400.
- [9] W. M. Gentleman and G. Sande, "Fast fourier transforms: for fun and profit," in *Proceedings of the November 7-10, 1966, fall joint computer conference*. ACM, 1966, pp. 563–578.
- [10] NVIDIA. (2017, Jul.) Nvidia tesla v100 gpu accelerator. [Online]. Available: http://www.nvidia.com/content/PDF/Volta-Datasheet.pdf
- [11] M. Harris. (2018, Feb.) An efficient matrix transpose in cuda c/c++. [Online]. Available: https://devblogs.nvidia.com/efficient-matrix- transpose-cuda-cc/

Q & A

"Let's embrace the era of GPU-accelerated computing"