Accelerating fast Fourier Transform with half-precision floating-point hardware on GPU

Fourier Transform

The Fourier transform is widely used in many applications including signal communication, image processing, and probability. The discrete Fourier transform of size N is to evaluate N-th root of unity:

$$S(k) = \sum_{n=0}^{N-1} s(n) \cdot \exp\left(-j \cdot \frac{2\pi}{N} \cdot n \cdot k\right), \; k = 0 \ldots N-1.$$

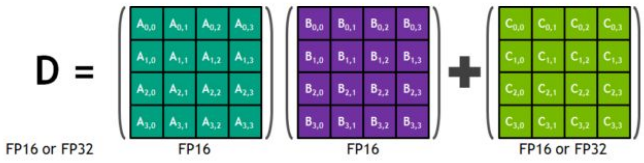
This can easily be represented in matrix form.

Fast Fourier Transform

The fast fourier transform (FFT) is a “divide and conquer” algorithm that allows computation of the fourier transform with less amount of computations. By dividing a problem of size N to two (or x) problems of size N/2 (or N/x), it attains time complexity O(nlogn). With a considerably large problem size, a significant performance improvement can be observed with the FFT. Besides the original algorithm, there are many “tricks” that lead to faster computation. For example, the 3M algorithm can cut the computational time by 25% with high stability. We'd like to exploit these tricks to further improve the implementation of FFT.

Tensor Core Hardware

Nvidia's new Volta architecture includes tensor cores which is a unit that multiplies two 4×4 FP16 matrices, and then adds a third FP16 or FP32 matrix to the result. It computes half precision operations with 12 times more speed than previous gpu architectures.



We plan on exploiting these tensor cores for mixed precision FFT. Looking at that radix-2, and radix-4 fourier transform matrices we see that the real and imaginary matrices only contain the values 1,0,-1. These numbers can be represented as half precision numbers without any error. Radix-8 fourier transform matrix would include sqrt(2)/2, which can be represented as 985/1393 with an error of 1.8e-7.

We can also represent single precision numbers as half precision and double precision numbers as single precision.

$$x\_fp32(:) = s1\_fp32 * x1\_fp16(:) + s2\_fp32 * x2\_fp16(:)$$

$$F * x\_fp32(:) = s1\_fp32 * (F * x1\_fp16(:)) + s2\_fp32 * (F * x2\_fp16(:))$$

This allows us to use the tensor core hardware, speeding up fourier transform, without much loss in accuracy.