Final project: FFT on GPU

Olga Gorbunova

Skolkovo Institute of Science and Technology

27th May 2022

What is DFT

- Discrete Fourier Transform (DFT) can be understood as a numerical approximation to the Fourier transform (FT).
- This is used in the case where both the time and the frequency variables are discrete (which they are if digital computers are being used to perform the analysis).
- To convert the integral FT into the DFT, we can do following steps:
  - Assume the sampling window is T. The number of sampling points is N. Define the sample interval $\Delta T = \frac{T}{N}$.
  - Define the sample points $t_k = k\Delta T$ for $k = 0, \ldots, N-1$.
  - Define the signal values at each sampling points as $f(t_k)$.
  - Define the frequency sampling points $\omega_n = \frac{2\pi n}{T}$, where $\frac{2\pi n}{T}$ is termed as the fundamental frequency.
  - Consider the problem of approximating the FT of f at the points ω_n.

What is DFT

The answer is:

$$F(\omega_n) = \int_{-\infty}^{+\infty} e^{-i\omega_n t} f(t)dt,\ \ n = 0, ..., N-1 \qquad (1)$$

Approximate this integral by Riemann sum approximation using the points t_k since $f \sim 0$ for $t > T$.

$$F(\omega_n) = \sum_{k=0}^{N-1} e^{-i\omega_n t_k} f(t_k),\ \ n = 0, ..., N-1 \qquad (2)$$

The inverse Discrete Fourier Transform is defined as

$$f(t_k) = \frac{1}{N} \sum_{n=0}^{N-1} e^{i\omega_n t_k} F(\omega_n),\ \ k = 0, ..., N-1 \qquad (3)$$

What is FFT

- Fast Fourier Transform (FFT) is a effective algorithm of DFT developed by Cooley and Tukey at 1965.
- This algorithm reduces the computation time of DFT for N points from N^2 to $N \log_2 N$. (This algorithm is called Butterfly algorithm.)
- The only requirement of this algorithm is that number of point in the series have to be a power of 2 (2^n points).
- Zero padding at the end of the data set if the sampling number is not equal to the exact the power of 2.

Algorithm

- Let $A(x) = a_0 + a_1 x + \cdots + a_n x^n$ be polynomial with given coefficients.
- Write $A(x) = A_0(x^2) + x A_1(x^2)$, where

$$A_0(x^2) = a_0 + a_2 x^2 + a_4 x^4 + \cdots \quad A_1(x^2) = a_1 + a_3 x^2 + a_5 x^4 + \cdots$$

- Compute the Fourier transforms of A_0 and A_1

Algorithm

- The Fourier transform of A is given by

$$\tilde{a}_k = \tilde{a}_k^0 + \varepsilon_n^k \tilde{a}_k^1,\ \tilde{a}_{k+\frac{n}{2}} = \tilde{a}_k^0 - \varepsilon_n^k \tilde{a}_k^1, 0 \leq k < \frac{n}{2}$$

where $\varepsilon_n = e^{\frac{2\pi i}{n}}$ is the nth complex root of unity.
- This gives rise to divide and conquer approach: we can calculate the Fourier transform for length n from two Fourier transforms for length $\frac{n}{2}$ in linear time. By master theorem, the algorithm has $O(n \log n)$ complexity in time.

Inplace Algorithm

- Reorder the elements within the array of coefficients according to bits starting from the least significant bit, zeros - to the left, ones - to the right, e.g.

$$(a_{00}, a_{01}, a_{10}, a_{11} \to (a_{00}, a_{10}, a_{01}, a_{11})$$

  where indices are given in binary.
- Calculate inplace the Fourier coefficients for sub arrays

$$(\tilde{a}_0^0, \ldots, \tilde{a}_{n/2}^0, \tilde{a}_0^1, \ldots \tilde{a}_{n/2}^1)$$

Inplace Algorithm

- Iteratively calculate the Fourier coefficients

$$(\tilde{a}_0^0 + \tilde{a}_0^1, \tilde{a}_1^0 \ldots, \tilde{a}_{n/2}^0, \tilde{a}_0^0 - \tilde{a}_0^1, \ldots \tilde{a}_{n/2}^1)$$

$$\vdots$$

$$(\tilde{a}_0^0+\tilde{a}_0^1, \tilde{a}_1^0+\varepsilon_n\tilde{a}_1^1 \ldots, \tilde{a}_{n/2}^0+\varepsilon_n^{n/2}\tilde{a}_{n/2}^1, \tilde{a}_0^0-\tilde{a}_0^1, \ldots \tilde{a}_{n/2}^0-\varepsilon_n^{n/2}\tilde{a}_{n/2}^1)$$

Inverse Fourier transform

- The inverse Fourier transform is given by

$$a_k = \frac{1}{n} \sum_m \tilde{a}_m \varepsilon_n^{-mk}$$

- Essentially it is the same as the direct Fourier transform except we replace ε_n by $\overline{\varepsilon}_n$ and divide the result by n

Applications of Fourier Transform

- Convolution can be computed in $O(n \log n)$ instead of n^2 if calculated directly.

$$a * b = \mathcal{F}^{-1}(\mathcal{F}(a)\mathcal{F}(b))$$

  Below are the some particular cases of convolution.

- Polynomial multiplication
- Schönhage–Strassen algorithm for integer multiplication

Applications of Fourier Transform

- Convolution can be computed in $O(n \log n)$ instead of n^2 if calculated directly.

$$a * b = \mathcal{F}^{-1}(\mathcal{F}(a)\mathcal{F}(b))$$

  Below are the some particular cases of convolution.
- Polynomial multiplication
- Schönhage–Strassen algorithm for integer multiplication

Applications of Fourier Transform

- Image compression, e.g. JPEG
- Video compression, e.g. MPEG
- Roughly speaking, these lossy compression algorithms calculate the two dimensional Fourier transform of the image matrix and truncate components corresponding to higher frequencies.

Acceleration of fast Fourier transform on GPU

- Recall that the first step of the inplace FFT is to calculate a permutation of the original array
- Reordering of the elements in the array can be parallelized, each thread calculating a block of the new array.

Acceleration of fast Fourier transform on GPU

- The next step is a loop
- On k-th iteration, the Fourier transform of blocks of length 2^k is calculated inplace, using the calculated subblocks of length 2^{k-1}.
- The calculations for each block are independent and can be done in separate threads

Acceleration of fast Fourier transform on GPU

- In later iterations when the blocks become bigger (but also there are fewer blocks), we can parallelize the calculation within the block
- Recall the formula for calculating the Fourier transform of a block

$$(\tilde{a}_0^0 + \tilde{a}_0^1, \tilde{a}_1^0 \ldots, \tilde{a}_{n/2}^0, \tilde{a}_0^0 - \tilde{a}_0^1, \ldots \tilde{a}_{n/2}^1)$$

$$\vdots$$

$$(\tilde{a}_0^0 + \tilde{a}_0^1, \tilde{a}_1^0 + \varepsilon_n \tilde{a}_1^1 \ldots, \tilde{a}_{n/2}^0 + \varepsilon_n^{n/2} \tilde{a}_{n/2}^1, \tilde{a}_0^0 - \tilde{a}_0^1, \ldots \tilde{a}_{n/2}^0 - \varepsilon_n^{n/2} \tilde{a}_{n/2}^1)$$

Acceleration of fast Fourier transform on GPU

- Note that the k-th and $k + n/2$-th Fourier coefficients are only dependent on $\tilde{a}_k^0$ and $\tilde{a}_k^1$
- Each thread can calculate k-th and $k + n/2$-th coefficients of the block inplace in parallel