Parallel Fast Fourier Transform

Jessica Grogan

University at Buffalo

Spring 2023

Motivation

There’s many meaningful impacts of the Fourier transform. For example,

- modern medicine
- noise cancellation

History

Joseph Fourier, born in 1768, was orphaned by age 10.

Rough childhood, kept busy by studying, became a mathematician.

He claimed, in the early 19^{th} century, that all functions are composed of sine and cosine waves.

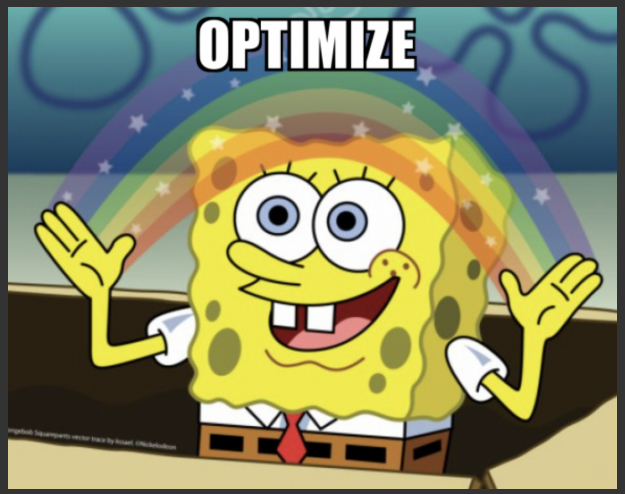
He was “approximately" correct.

Definition

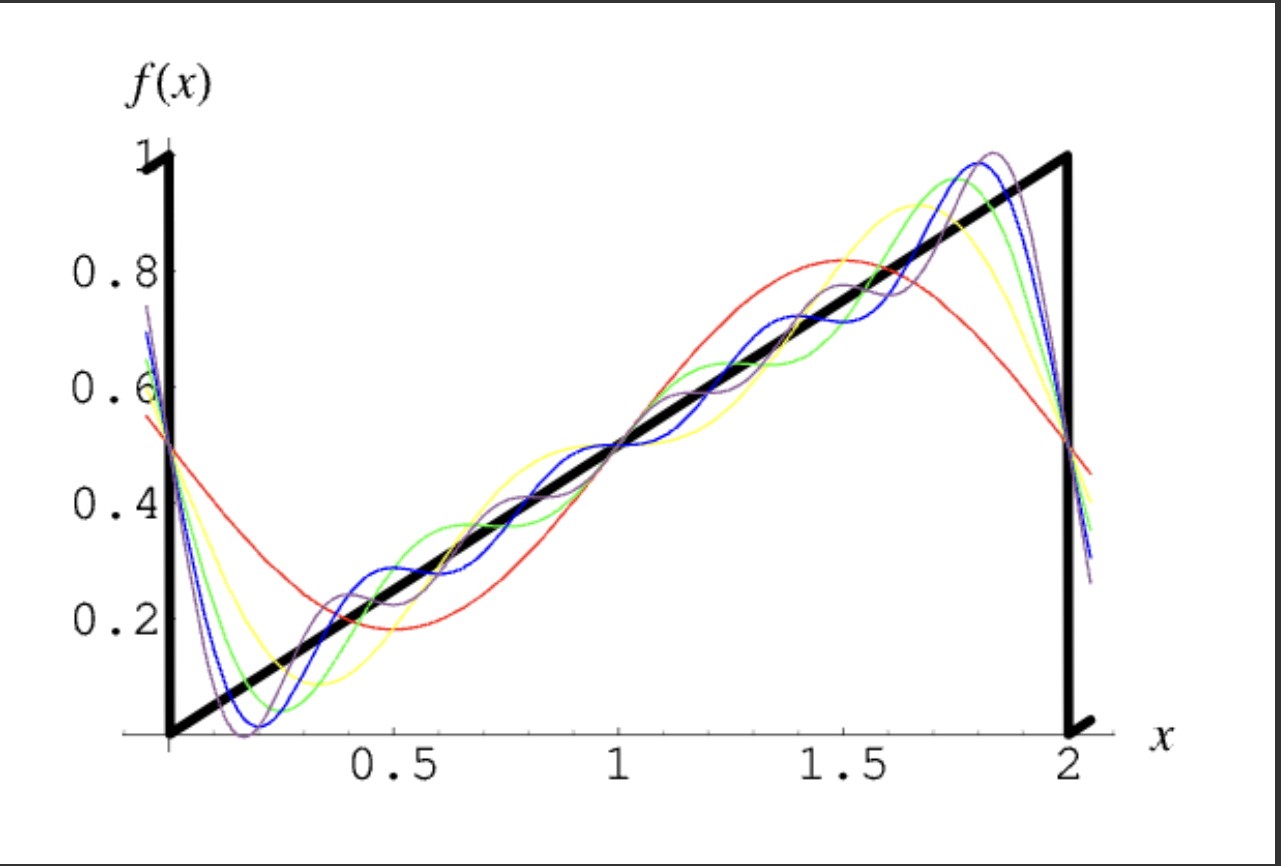
The Fourier transform takes an input from the one domain (time) and transfers it to another domain (frequency).

The discrete Fourier transform (DFT) is used for non-continuous functions. [3] [1]

The Fast Fourier Transform (FFT) is an efficient algorithm for the DFT. [2]



Sawtooth approximation



We can approximately represent other functions by many cosine and sine functions

Fourier matrix

The Fourier matrix is composed of entries that make up the values of the Fourier transform.

Example: $n = 4$, $\mathbf{F}_4$ can be defined as,

$$\mathbf{F}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix}$$

where $\omega = e^{\frac{-2\pi i}{4}}$. From Euler's formula we know $e^{ix} = \cos(x) + i\sin(x)$.

Algorithm

The FFT can be viewed as a matrix vector multiplication, where $\bar{\mathbf{X}} \in \mathbb{R}^n$ is our input vector,

$$\mathbf{y} = \mathbf{F}_n \cdot \bar{\mathbf{x}}$$

Thanks to Cooley and Tukey we can take advantage of related entries. So we split the vectors into even and odd components, repeatedly.

Sequential algorithm

We can decompose the matrix $\mathbf{F}_n$ into a product of matrix multiplications,

$$\mathbf{F}_n = \mathbf{A}_{r-1} \ldots \mathbf{A}_1 \mathbf{A}_0$$

where each matrix $\mathbf{A}_i$ is a $n \times n$ matrix.

Therefore we can represent, $\mathbf{F}_n \cdot \bar{\mathbf{x}}$ as,

$$\mathbf{A}_{r-1} \ldots \mathbf{A}_1 \mathbf{A}_0 \cdot \bar{\mathbf{x}}$$

Cont.

$$\mathbf{F}_n \cdot \bar{\mathbf{x}} = \begin{bmatrix} \mathbf{I}_{\frac{n}{2}} & \Omega_{\frac{n}{2}} \\ \mathbf{I}_{\frac{n}{2}} & -\Omega_{\frac{n}{2}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{F}_{\frac{n}{2}} & 0 \\ 0 & \mathbf{F}_{\frac{n}{2}} \end{bmatrix} \cdot \begin{bmatrix} \bar{\mathbf{x}}(0:2:n-1) \\ \bar{\mathbf{x}}(1:2:n-1) \end{bmatrix}.$$

With

$$\Omega_{\frac{n}{2}} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & \omega_n & 0 & \dots & 0 \\ 0 & 0 & \omega_n^2 & \dots & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \dots & \omega_n^{\frac{n}{2}-1} \end{bmatrix}$$

Sorting matrix

$\mathbf{S}_n \cdot \bar{\mathbf{x}}$ gives you the even odd split Where

$$\mathbf{S}_n = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ & & \vdots & & & & \vdots & \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ & & \vdots & & & & \vdots & \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix}$$

Spliting $\bar{\mathbf{x}}$

We get

$$\mathbf{S}_n \cdot \bar{\mathbf{x}} = \begin{bmatrix} \bar{\mathbf{x}}(0:2:n-1) \\ \bar{\mathbf{x}}(1:2:n-1) \end{bmatrix}$$

Kronecker product

The kronecker product is used for things like

$$\mathbf{I}_2 \otimes \mathbf{F}_{\frac{n}{2}} = \begin{bmatrix} \mathbf{F}_{\frac{n}{2}} & 0 \\ 0 & \mathbf{F}_{\frac{n}{2}} \end{bmatrix}$$

Butterfly matrix

$$\mathbf{B}_n = \begin{bmatrix} \mathbf{I}_{\frac{n}{2}} & \Omega_{\frac{n}{2}} \\ \mathbf{I}_{\frac{n}{2}} & -\Omega_{\frac{n}{2}} \end{bmatrix}$$

Conclusion of the algorithm

Cooley-Tukey decomposition of $\mathbf{F}_n$ is

$$\mathbf{F}_n = (\mathbf{I}_1 \otimes \mathbf{B}_n)(\mathbf{I}_2 \otimes \mathbf{B}_{\frac{n}{2}})(\mathbf{I}_4 \otimes \mathbf{B}_{\frac{n}{4}}) \ldots (\mathbf{I}_{\frac{n}{2}} \otimes \mathbf{B}_2)\mathbf{R}_n.$$

Where $\mathbf{R}_n$ is the bit reversal permutation matrix:

$$\mathbf{R}_n = (\mathbf{I}_{\frac{n}{2}} \otimes \mathbf{S}_2) \ldots (\mathbf{I}_4 \otimes \mathbf{S}_{\frac{n}{4}})(\mathbf{I}_2 \otimes \mathbf{S}_{\frac{n}{2}})(\mathbf{I}_1 \otimes \mathbf{S}_n).$$

Parallelize

Each processor deals with a matrix multiplication such as:

$$\begin{bmatrix} \mathbf{S}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_2 \end{bmatrix} \cdot \mathbf{S}_4$$

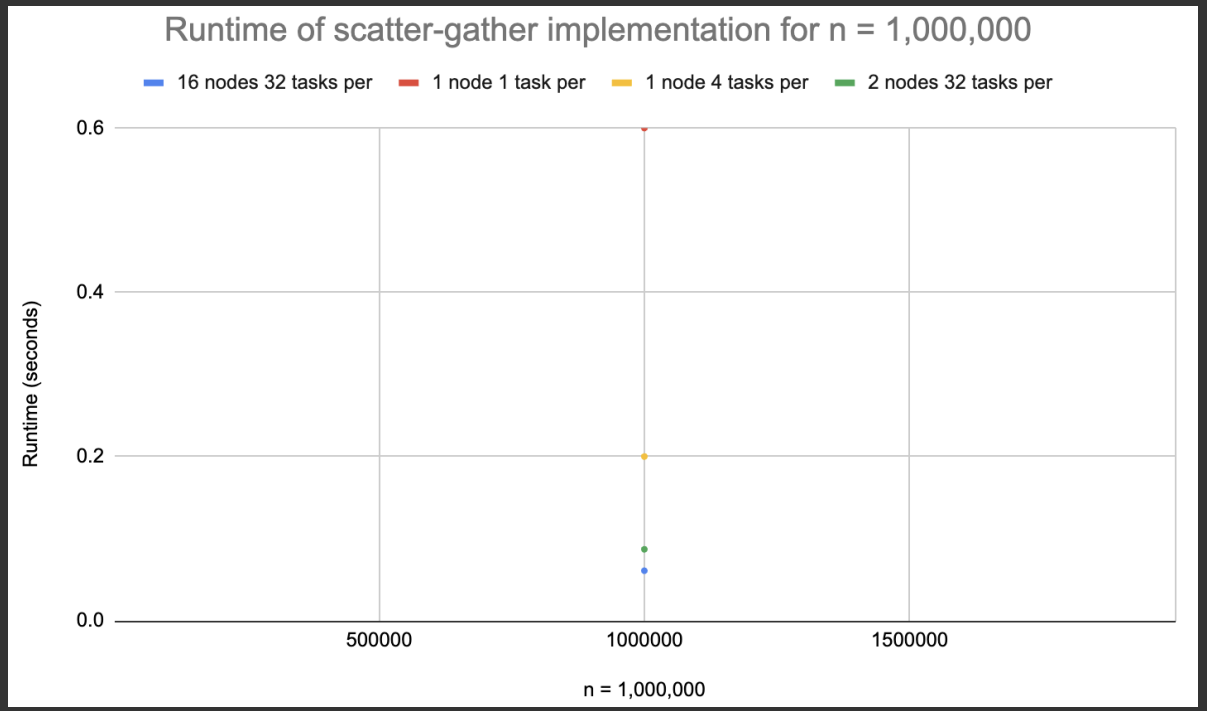
$$\mathbf{B}_4 \cdot \begin{bmatrix} \mathbf{B}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_2 \end{bmatrix}$$

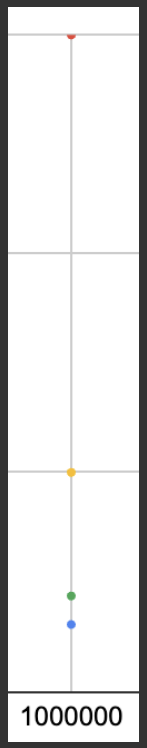
And gets sent to a ready processor in order of how the matrices need to be multiplied.

I do acknowledge there's many other ways to parallelize the Fourier Transform that are probably more efficient. [4]

Scatter Gather

I was able to get a scatter gather version working and run it multiple times on node variations. This is the runtime with n= 1,000,000. The fraph shows the runtime decreasing with the number of processors used. 1 node with 1 task, runtime was .6 seconds. 1 node with 4 tasks, runtime goes to .2 seconds. Then 2 nodes with 32 tasks per node gets .087 seconds. Finally, we got .074 seconds with 16 nodes and 32 tasks per node. We can see the runtime start to level off.





Complete table

This is the complete table of numbers I was able to get for scatter gather method.

	A	B	C	D	E	F	G	H
1	size of n	Runtime sequential	1 node 32 tasks per	16 nodes 4 tasks per	16 nodes 32 tasks per	1 node 1 task per	1 node 4 tasks per	2 nodes 32 tasks per
2	128	0.002288		0.0005				
3	256	0.005433						
4	512	0.030513	0.002					
5	1024	0.096223						
6	2048	0.245302						
7	4096	0.737164						
8	8192	2.480509						
9	16384	10.161637						
10	32768	41.74056	0.007					
11	1,000,000			0.074	0.061	0.6	0.2	0.087

Difficulties / observations

- Data distribution for my algorithm, ordering the mat-muls.

- Data delay for busy processors.

Future work

- Parallelize the inner workings of the matrix multiplications / implement matrix multiplication using BLAS (Basic linear algebra subprograms).

- Run this algorithm on GPUs.

References

Discrete-time Fourier Transform.

Discrete-time fourier transform — Wikipedia, the free encyclopedia, 2005. https://en.wikipedia.org/wiki/Discrete-time_Fourier_transform.

Fast Fourier Transform.

Fast foruier transform — Wikipedia, the free encyclopedia, 2008. https://en.wikipedia.org/wiki/Fast_Fourier_transform.

Professor Rob H. Bisseling Utrecht University.

Nonrecursive fast fourier transform, 2022. https://www.youtube.com/watch?v=WcCl4q2p8cM&t=824s.

Professor Rob H. Bisseling Utrecht University.

Parallel fast fourier transform, 2022. https://www.youtube.com/watch?v=K-HwNf1BxwA&t=244s.