159.735 Studies in Parallel and Distributed System

Parallel Fast Fourier Transform

Name: Bo LIU

ID: 03278999

Introduction

Fourier Transform plays an important role in signal processing, image processing and voice recognition and so on. It has been using for wide range of areas. It may be used for people's life, and it may be used for scientific research as well. The Fourier Transform has many applications in science and engineering. For example, it is often used in digital signal processing applications such as signal processing, voice recognition and image processing. The Discrete Fourier Transform is a specific kind of Fourier Transform. It maps a sequence over time to another sequence over frequency. However, if the Discrete Fourier Transform is implemented straightforward, the time complexity is $O(n^2)$. It is not a better way to be used in practice. Alternatively, the Fast Fourier Transform is just O(n log n) algorithm to perform the Discrete Fourier Transform which can be easily parallelized as well.

Fourier analysis

Fourier analysis is the representation of continuous function by a potentially infinite series of sin and cosine functions. It is grown out of the study of Fourier series. The Fourier series is a function which can be expressed as the sum of a series of sins and cosines.

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos(nx) + \sum_{n=1}^{\infty} b_n \sin(nx)$$

Where n = 1, 2, 3 …

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)dx \qquad a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)dx$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)dx$$

The numbers a_n and b_n are called Fourier coefficients of *f.* so, infinite sum *f(x)* is called the Fourier series of *f.* Fourier series can be generalized to complex numbers, and further generalized to derive the Fourier Transform.

Fourier Transform

The Fourier Transform is defined by the expression:

Forward Fourier Transform:

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i k x} dk$$

Inverse Fourier Transform:

$$f(x) = \int_{-\infty}^{\infty} F(k) e^{2\pi i k x} dk$$

Note: $e^{xi} = \cos(x) + i\sin(x)$

The equation defines *F(k)*, the Fourier Transform of *f(x)*. *f(x)* is termed a function of the variable time and *F(k)* is termed a function of the variable frequency.

Fourier Transform actually maps a time domain (series) into the frequency domain (series). So, the Fourier Transform is often called the frequency domain. Inverse Fourier Transform maps the domain of frequencies back into the corresponding time domain. The two functions are inverses of each other.

Frequency domain ideas are important in many application areas, including audio, signal processing and image processing. For example, spectrum analysis is widely used to analyzed speech, compress images, and search for periodicities in a wide variety of data in biology, physics and so on. Particularly, JPEG compression algorithm which are widely used and very effective, use a version of the Fourier Cosine Transform to compress the image data.

However, the Fourier transform is not suitable for machine computation because infinity of samples have to be considered. There is an algorithm called Discrete Fourier Transform, which is modified based on the Fourier Transform, can be used for machine computation.

Discrete Fourier Transform

The Discrete Fourier Transform is a specific kind of Fourier Transform. It requires an input function that is discrete and whose non-zero values have a finite duration. The input function can be a finite sequence of real or complex numbers, thus the DFT is ideal for processing information stored in computers. But, if the data is continuous, the data has to be sampled when we use the DFT. There is a possibility that if the sampling interval is too wide, it may cause the aliasing effect, however, if it's too narrow, the size of the digitalized data might be increased. The definition is expressed in the following.

Given a sequence of f_n for $k = 0,1,2, \ldots, N-1$, is transformed into the sequence of X_k by the DFT according to the formula:

$$X_k = \sum_{n=0}^{N-1} f_n e^{\frac{-2\pi i}{N} kn}$$

The inverse DFT is given by:

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn}$$

Where $w_n = e^{\frac{2\pi i}{N}} = \cos(2\pi / N) + i \sin(2\pi / N)$ is a primitive Nth root or unity.

DFT Computation

Given n elements vector x, the DFT Matrix vector product $F_n x$, where $f_{i,j} = w_n^{ij}$ for 0 <= I, j < n. The following examples are done based on formula above and DFT Matrix.

Examples of DFT computation:

DFT of vector (2, 3), the primitive square root of unity for w_2 is -1.

$$\begin{pmatrix} w_2^{0\times 0} & w_2^{0\times 1} \\ w_2^{1\times 0} & w_2^{1\times 1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 5 \\ -1 \end{pmatrix}$$

The inverse of DFT:

$$\begin{pmatrix} w_2^{0\times 0} & w_2^{0\times 1} \\ w_2^{1\times 0} & w_2^{1\times 1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 5 \\ -1 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

DFT of vector (1, 2, 4, 3), the primitive 4th root of unity for w_4 is i.

$$\begin{pmatrix} w_4^0 & w_4^0 & w_4^0 & w_4^0 \\ w_4^0 & w_4^1 & w_4^2 & w_4^3 \\ w_4^0 & w_4^2 & w_4^4 & w_4^6 \\ w_4^0 & w_4^3 & w_4^6 & w_4^9 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 10 \\ -3-i \\ 0 \\ -3+i \end{pmatrix}$$

The inverse of DFT:

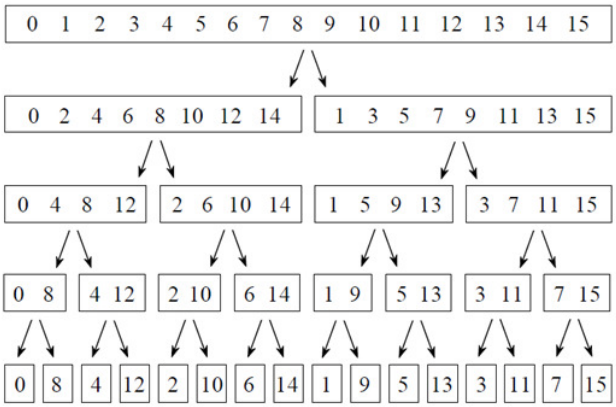
$$\frac{1}{4} \begin{pmatrix} w_4^0 & w_4^0 & w_4^0 & w_4^0 \\ w_4^0 & w_4^{-1} & w_4^{-2} & w_4^{-3} \\ w_4^0 & w_4^{-2} & w_4^{-4} & w_4^{-6} \\ w_4^0 & w_4^{-3} & w_4^{-6} & w_4^{-9} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \begin{pmatrix} 10 \\ -3-i \\ 0 \\ -3+i \end{pmatrix}$$

$$= \frac{1}{4} \begin{pmatrix} 4 \\ 8 \\ 16 \\ 12 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 4 \\ 3 \end{pmatrix}$$

Fast Fourier Transform

As the time complexity of DFT for n samples is O (n^2) if the DFT is implemented straightforward. So, using DFT is not a best way in practice. There is an improved algorithm called Fast Fourier Transform (FFT) which produces exactly the same result as the DFT. It uses divide – and – conquer strategy. So, it only takes O(n log n) time to compute n samples. The only difference between DFT and FFT is that FFT is much faster than DFT. It can be thought as a fast version of DFT.

The idea is that keep dividing a DFT sequence of *N* samples into two sub sequence. It splits the even index and odd index each step. If *N* is a power of 2, it keeps splitting the sequence until each subsequence only has one element. The rearranged index is just the bit-reversed order as the original index.



Decimal	*Binary*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Decimal	*Binary*
0	0000
8	1000
4	0100
12	1100
2	0010
10	1010
6	0100
14	1110
1	0001
9	1001
5	0101
13	1101
3	0011
11	1011
7	0111
15	1111

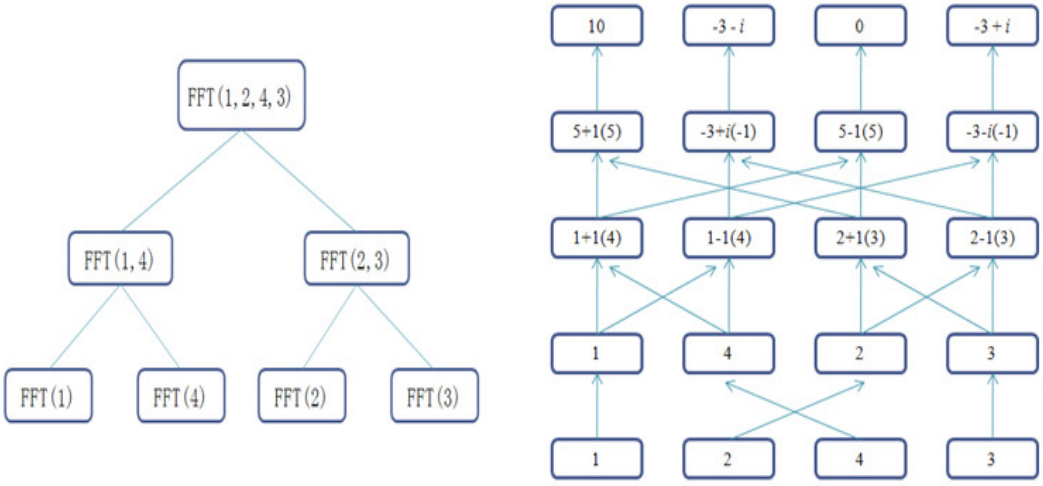
We can rewrite DFT function as follows:

$$X_k = \sum_{n=0}^{N-1} f_n w_N^{-kn} = \sum_{n=0}^{N/2-1} f_{2n} w_N^{-2kn} + \sum_{n=0}^{N/2-1} f_{2n+1} w_N^{-(2k+1)n}$$

$$= \sum_{n=0}^{N/2-1} f_n^{even} w_{N/2}^{-kn} + w_N^k \sum_{n=0}^{N/2-1} f_n^{odd} w_{N/2}^{-kn}$$

So, for example, 16 points, we have $\log_2$ N steps which is 4 steps and each has N operations. Finally the time complexity is *O(N log N)*.

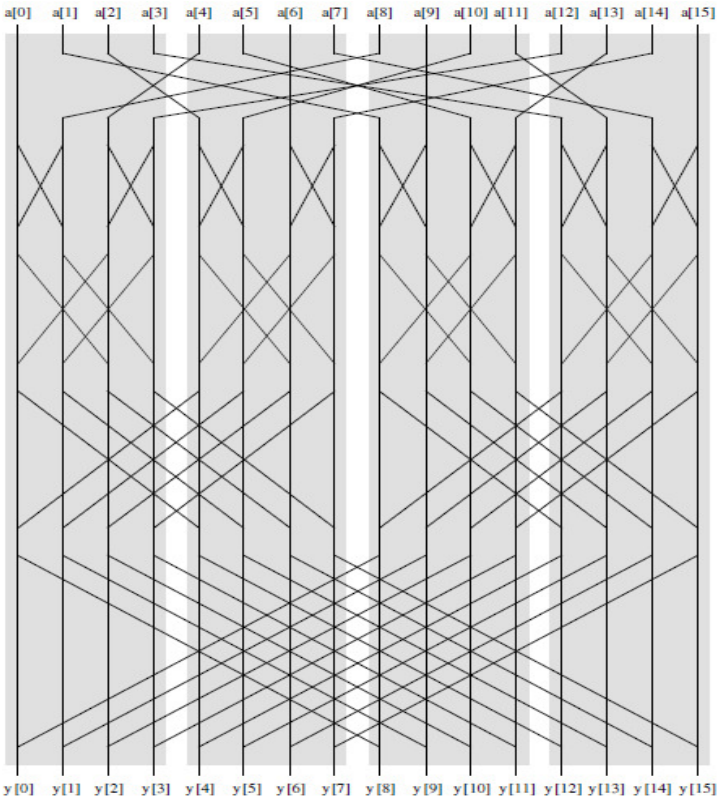
Use example in the Discrete Fourier Transform section to re-do it with FFT. The diagram is shown below.



Parallel Fast Fourier Transform

When parallelize the FFT algorithm, we have to consider that which algorithm is suitable for implementing the FFT. The recursive way for the FFT algorithm is easy to implement. However, there are two reasons for using an iterative way for FFT algorithm. First, iterative version of the FFT algorithm can perform fewer index computations. Second, it is easier to derive a parallel FFT algorithm when the sequential algorithm is in iterative form. As we may already know that the output index is the bit-reversed as the input index. So, use this idea to rearrange the index.

The following graph shows the process for parallel Fast Fourier Transform:



Top sequence is input and bottom sequence is output. Each process is represented by a gray rectangle.

There are three phrases for the parallel algorithm. Assume *n* is number of elements, and *p* is number of processes. First, the processes permute the input sequence a, rearrange the indices. In the second phrase, the processes perform the first log n – log p iterations of the FFT by performing the required multiplications, additions and subtraction on complex numbers. In the third phase the processes perform the final log p iterations of the FFT, and swapping values with partner across hypercube dimension.

So, each process controls n/p elements of input sequence a. There are log p iterations in which each process swaps about n/p values with a partner process. The overall communication time complexity is O ((n/p) log p), and the computational complexity of the parallel algorithm is O (n log n/p).

Reference

Quinn, M.J., (2004). *Parallel programming in C with MPI and OpenMP.* New York: McGraw-Hill Higher Education.

Gray, R.M., & Goodman, J.W., (1995). *Fourier transforms: an introduction for engineers.* Boston: Kluwer Academic Publishers.

Brigham, E.O., (1988). *The fast Fourier transform and its applications*. Englewood Cliffs, N,J.: Prentice Hall.

Chu, E., & George, A., (1999). *Inside the FFT black box: serial and parallel fast Fourier transform algorithms*. Boca Raton, Fla.: CRC Press.