Parallel Fast Fourier Transform

Yu Pei
Weikang Wang
CS 581 Algorithm
April 17th 2018

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Test Questions

1. What is the meaning of radix in FFT?

2. When people mention FFT, which specific algorithm are they generally referring to?

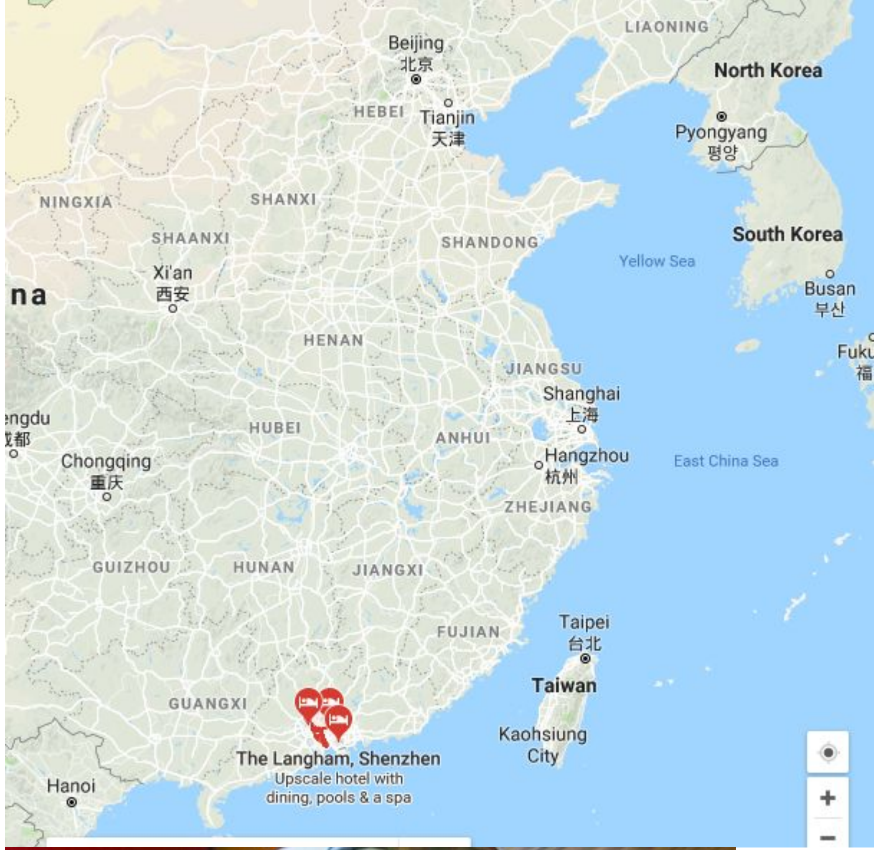
3. Why is the deepest full binary tree based parallel FFT slower than sequential FFT?

About Me – Yu Pei

From Guangzhou, China
A lot of factories, “Made in China”

A PhD student working on distributed runtime system and numerical linear algebra

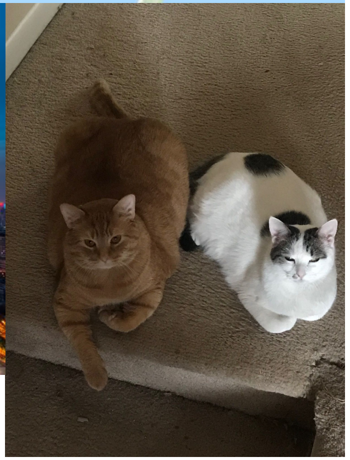
Like to play basketball and enjoy good food



About me – Weikang Wang

Born in Chengdu, Sichuan, China
A PhD student in Electrical Engineering
Cooking, Cats (Potato & Bubbles)
Interned in OSIsoft in Johnson City, TN, 2017
Area of interest:

- Wide Area Measurement System
- Distributed System
- Software Engineering



Outline

1. Overview
2. History
3. Algorithms
4. Implementation & Performance
5. Applications
6. Available packages
7. Open Issues
8. References

Overview

What is Fast Fourier Transformation?

- A family of algorithms that collectively go by the name of "The Fast Fourier Transform"

What does Fourier Transformation do?

- The most ubiquitous algorithm used today in the analysis and manipulation of digital or discrete data

What are the applications of the Fourier Transformation

- Signal processing, image processing, pattern recognition, radar and communications, etc.

FFT History

Originated from Gauss's unpublished work in **1805** to interpolate the orbit of asteroids *Pallas* and *Juno*

He dealt with trigonometric interpolation to periodic functions, which is expressed by a Fourier series of the form

$$f(x) = \sum_{k=0}^{m} a_k \cos 2\pi kx + \sum_{k=1}^{m} b_k \sin 2\pi kx,$$

Gauss's trick:

$$N = N_2 * N_1$$

N_2 sets of equally spaced samples with each of N_1 size. (DP)

FFT History

"Interaction Algorithm" Yates, 1932

Not in a general form

Used to compute the Hadamard & Walsh transforms

"doubling trick" Danielson & Lanczos, 1942

Reduce a DFT on 2N points to 2 DFS, each on N points using linear time transformation

FFT History

Cooley and Tukey, 1965 formally proposed the algorithm, and it coincide with the development of analog to digital converters (ADC), so the algorithm provided a fast way to analyze digital data!

$$T(N) = NLog(N)$$

The Cooley-Tukey FFT algorithm is **the FFT algorithm** that are generally mentioned

Basic Computation

Recall the definition of discrete Fourier transform:

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{jk} , \qquad \text{where } \omega_n = e^{-2\pi\sqrt{-1}/n}$$

For an input array of length n, the direct method will take $O(n^2)$ time.

Cooley – Tukey Main idea

Trick: If $n = pq$, write $j = pj_1 + j_2$ and $k = k_1 + qk_2$.

$$
\begin{aligned}
y_{k_1+qk_2} &= \sum_{j_2=0}^{p-1} \sum_{j_1=0}^{q-1} x_{pj_1+j_2} \omega_n^{pj_1k_1} \cdot \omega_n^{j_2k_1} \cdot \omega_n^{pqj_1k_2} \cdot \omega_n^{qj_2k_2} \\
&= \underbrace{\sum_{j_2=0}^{p-1}}_{\text{size-}p\text{ DFTs}} \left[\left(\underbrace{\sum_{j_1=0}^{q-1} x_{pj_1+j_2} \omega_q^{j_1k_1}}_{\text{size-}q\text{ DFTs}} \right) \underbrace{\omega_n^{j_2k_1}}_{\text{twiddles}} \right] \omega_p^{j_2k_2}.
\end{aligned}
$$

Namely we break down the computation into a 2D DFT of size p * q, usually p or q is a small “radix” r, and 2 is a very common choice

Cooley – Tukey, radix = 2

$$X_{k_2} = \sum_{n_2=0}^{N/2-1} x_{2n_2} W_{N/2}^{n_2 k_2}$$

$$+ W_N^{k_2} \sum_{n_2=0}^{N/2-1} x_{2n_2+1} W_{N/2}^{n_2 k_2},$$

$$X_{N/2+k_2} = \sum_{n_2=0}^{N/2-1} x_{2n_2} W_{N/2}^{n_2 k_2}$$

$$- W_N^{k_2} \sum_{n_2=0}^{N/2-1} x_{2n_2+1} W_{N/2}^{n_2 k_2}.$$

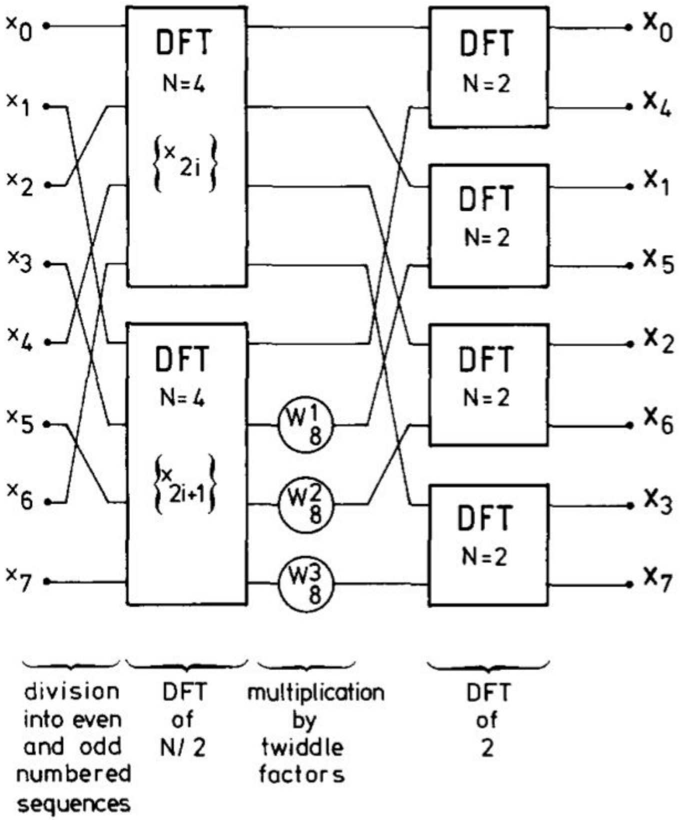
Decimation in Time (DIT), i.e. p = 2

$$X_{2k_1} = \sum_{n_1=0}^{N/2-1} W_{N/2}^{n_1 k_1} (x_{n_1} + x_{N/2+n_1}),$$

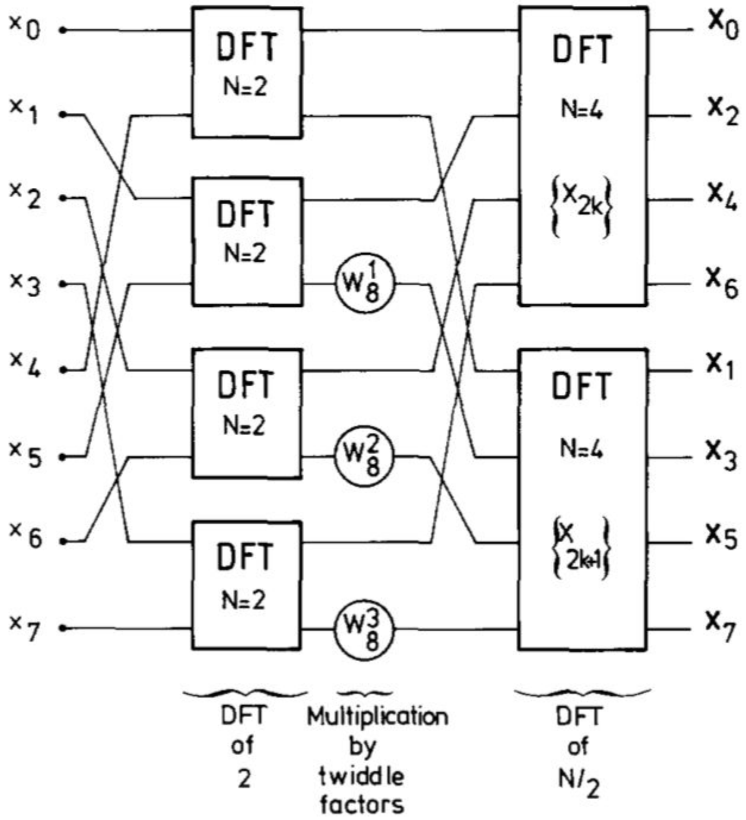
$$X_{2k_1+1} = \sum_{n_1=0}^{N/2-1} W_{N/2}^{n_1 k_1} W_N^{n_1} (x_{n_1} - x_{N/2+n_1}).$$

Decimation in Frequency (DIF), i.e. q = 2

Cooley – Tukey, radix = 2



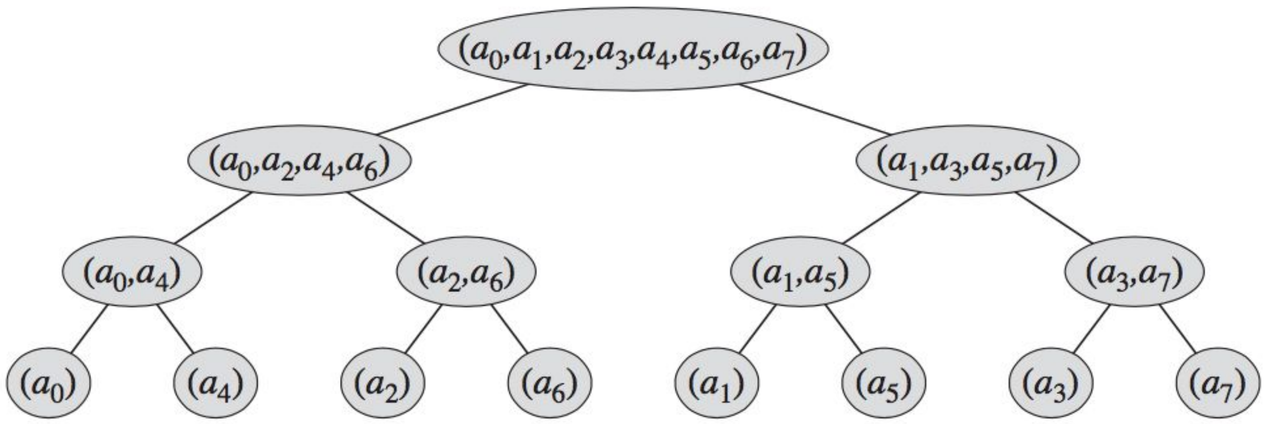
Decimation in Time (DIT), i.e. p = 2



Decimation in Frequency (DIF), i.e. q = 2

Implementation

Classic implementation use a procedure called bit-reversal, which moves the element to its final position in the output sequence beforehand, or after computation



Original : 000; 001; 010; 011; 100; 101; 110; 111

Reversed: 000; 100; 010; 110; 001; 101; 011; 111

Implementation

Benefits of in-place computation is we don't need extra memory, the drawback is strided memory access!!!

Speed = #computation + #memory/cache access

The name of the game: do as much work as possible before going out of cache.

Implementation

Use Stockham Algorithm instead. Which use another array of size N as working set:

[recursion stage-0]

fft0(8,1,0,x,y)

access to x[p]

[recursion stage-1]

fft0(4,2,0,x,y) fft0(4,2,1,x,y)

access to x[2p+0] access to x[2p+1]

[recursion stage-2]

fft0(2,4,0,x,y) fft0(2,4,2,x,y) fft0(2,4,1,x,y) fft0(2,4,3,x,y)

access to x[4p+0] access to x[4p+2] access to x[4p+1] access to x[4p+3]

fft0(N, S, q, x, y)

N: length of the array

S: Stride of the access

q: even or odd access

x: input array

y: working set

We can use iteration instead of recursive calls for the recursion stage-2, just loop q from 0 to 3

Implementation

Use Stockham Algorithm instead. Which use another array of size N as working set:

```
for (int q = 0; q < s; q++) { // Iteration for recursive-call
        for (int p = 0; p < m; p++) {
            const complex_t wp = complex_t(cos(p*theta0), -sin(p*theta0));
            const complex_t a = x[q + s*(p + 0)];
            const complex_t b = x[q + s*(p + m)];
            y[q + s*(2*p + 0)] =  a + b;
            y[q + s*(2*p + 1)] = (a - b) * wp;
        }
}
fft0(n/2, 2*s, y, x);
```

fft0(N, S, x, y)

N: length of the array

S: Stride of the access

~~q: even or odd access~~

x: input array

y: working set

$$X_{2k_1} = \sum_{n_1=0}^{N/2-1} W_{N/2}^{n_1 k_1}(x_{n_1} + x_{N/2+n_1}),$$

Implementation

Use Stockham Algorithm instead. Which use another array of size N as working set:

```
for (int q = 0; q < s; q++) { // Iteration for recursive-call
        for (int p = 0; p < m; p++) {
            const complex_t wp = complex_t(cos(p*theta0), -sin(p*theta0));
            const complex_t a = x[q + s*(p + 0)];
            const complex_t b = x[q + s*(p + m)];
            y[q + s*(2*p + 0)] =  a + b;
            y[q + s*(2*p + 1)] = (a - b) * wp;
        }
}
fft0(n/2, 2*s, y, x);
```

Non sequential array access! Switch the order of the two for loops!

Then we can see that we are accessing the array consecutively, which is good!

Parallel Implementation - SIMD!

Intel® Advanced Vector Extensions (Intel® AVX) is a set of instructions for doing Single Instruction Multiple Data (SIMD) operations on Intel® architecture CPUs.

```
for (int p = 0; p < m; p++) {
        const double cs = cos(p*theta0);
        const double sn = sin(p*theta0);
        const __m256d wp = _mm256_setr_pd(cs, −sn, cs, −sn);
        for (int q = 0; q < s; q += 2) {
            double* xd = &(x + q)−>Re;
            double* yd = &(y + q)−>Re;
            const __m256d a = _mm256_load_pd(xd + 2*s*(p + 0));
            const __m256d b = _mm256_load_pd(xd + 2*s*(p + m));
            _mm256_store_pd(yd + 2*s*(2*p + 0),           _mm256_add_pd(a, b));
            _mm256_store_pd(yd + 2*s*(2*p + 1), mulpz2(wp, _mm256_sub_pd(a, b)));
        }
}
```

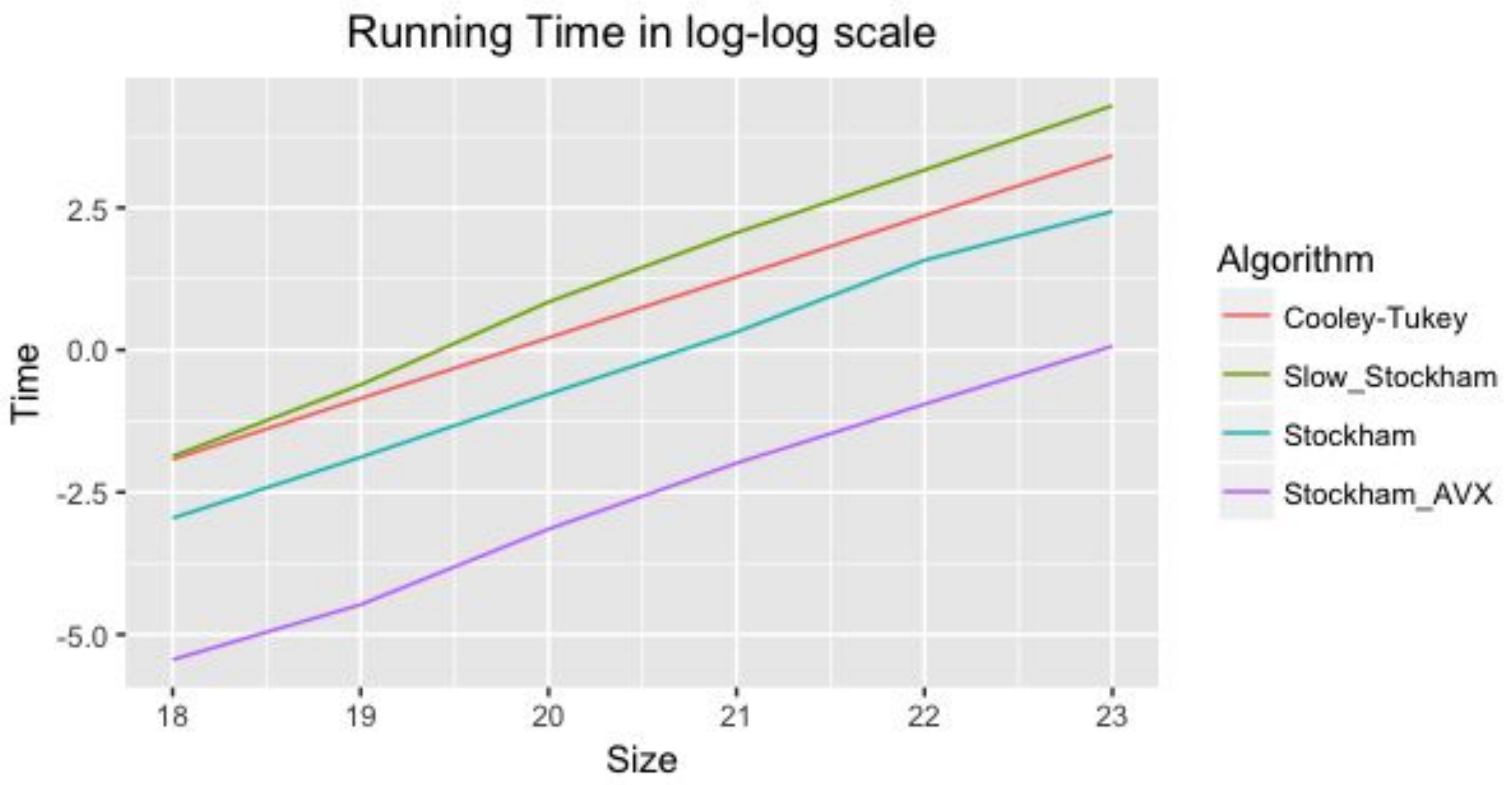
_mm256_load_pd: a compiler intrinsic that load 256 bits (4 doubles) in one instruction (Intel® AVX instruction is `VMOVAPD`)

similar for other instructions...

Interesting note: this idea is suggested in 1987 for the vector computer, AVX is enabling the comeback of vector computing!

Performance Measurements

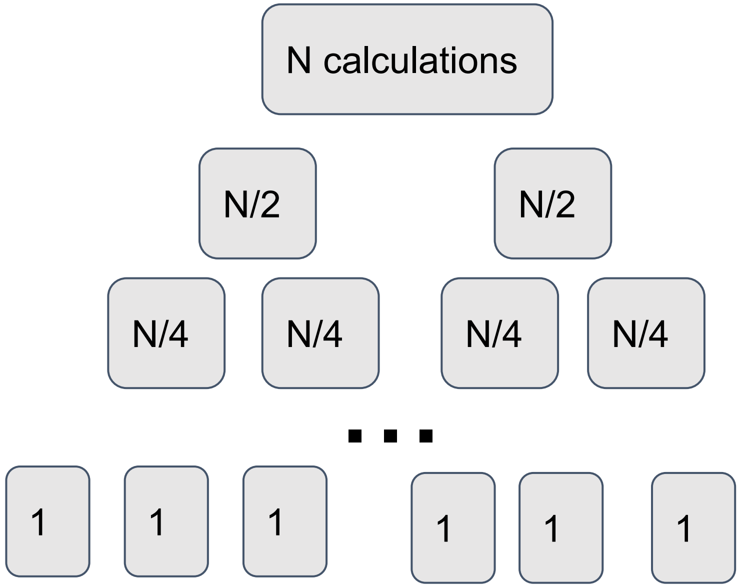
Experimented on a compute node with [Haswell E5-2650 v3](#) @ 2.30GHz



Implementation (Classic)

Assumption: Cooley-Tukey Algorithm is used

Goal: Calculate the array using parallel FFT



$2^{(D-d)}+d$, $0<d<D$, $D = \log(N)$

Let $x = D-d$

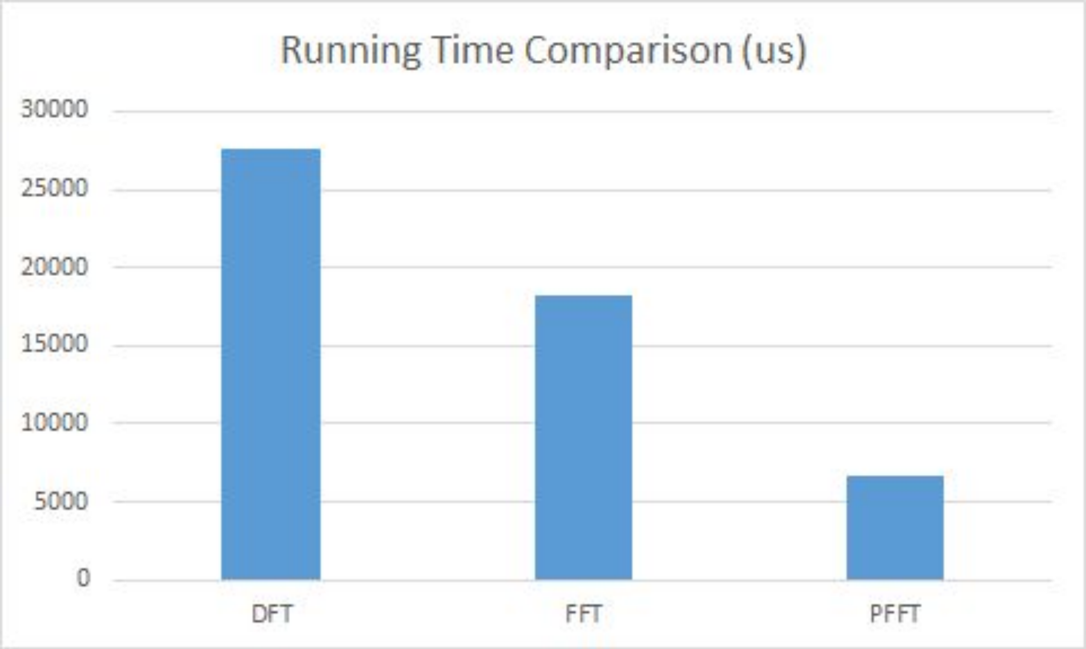
Let $T(x) = 2^{(x)}+D-x$

Take the derivative of T(d)

$T'(x) = 2^x \ln 2 - 1$

Optimal solution at $x = 1$

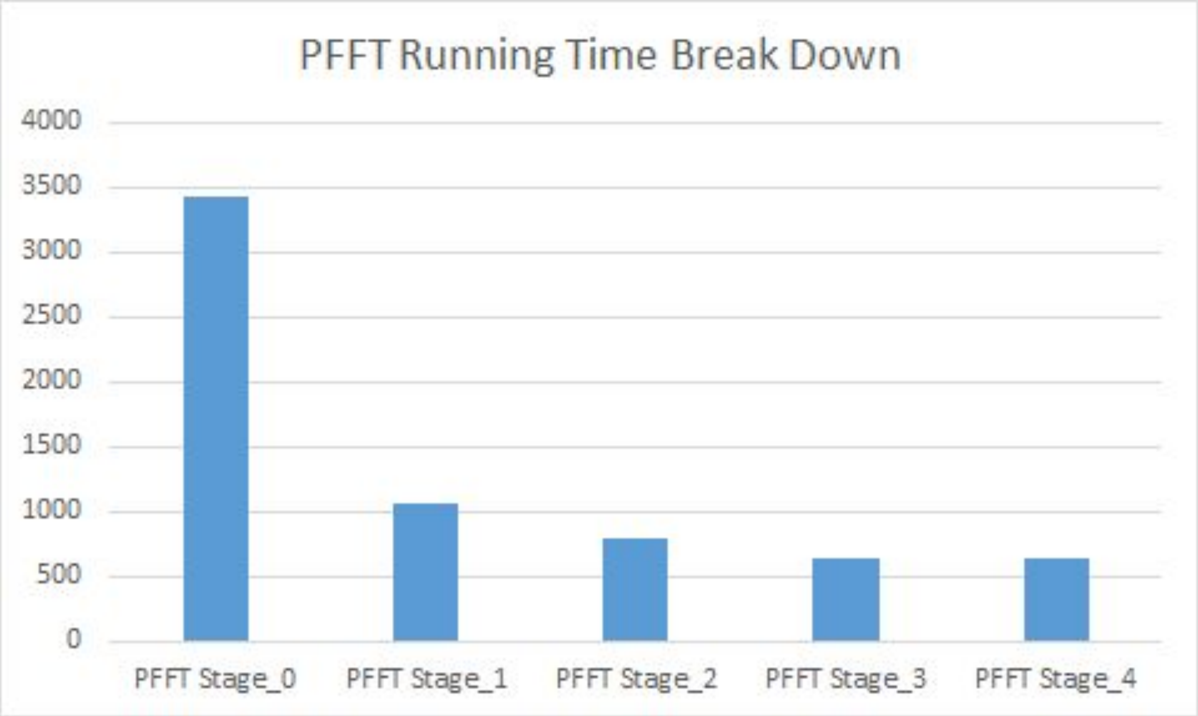
Implementation (Classic)



DFT	27589
FFT	18286
PFFT (Calculation Time)	6585

N = 24

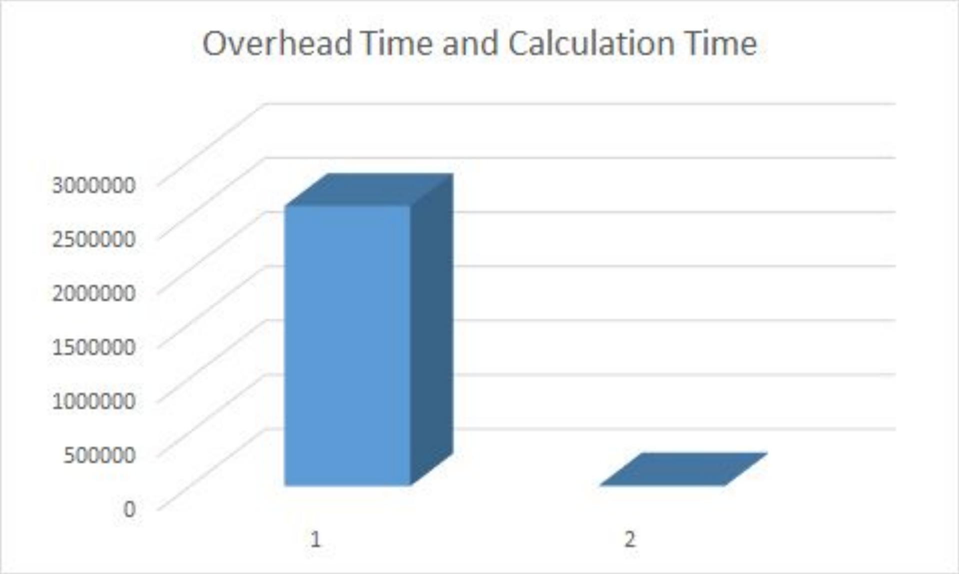
Implementation (Classic)



PFFT Stage_0	3440
PFFT Stage_1	1,069
PFFT Stage_2	792
PFFT Stage_3	642
PFFT Stage_4	642

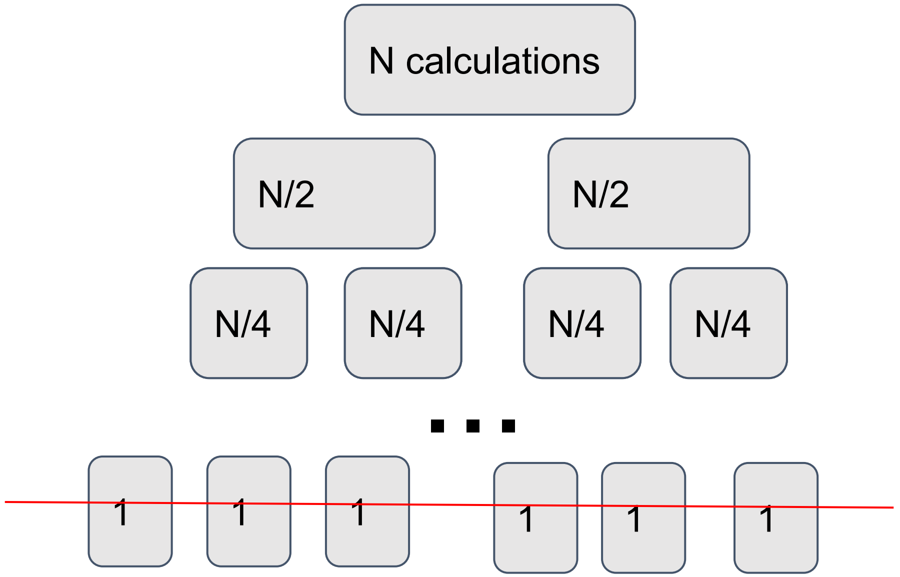
Implementation (Classic)

Too Idealized!
The **overhead time** (thread invoking, function invoking, etc) is 300 times more than the **actual calculation time** in the deepest **full binary tree** structure

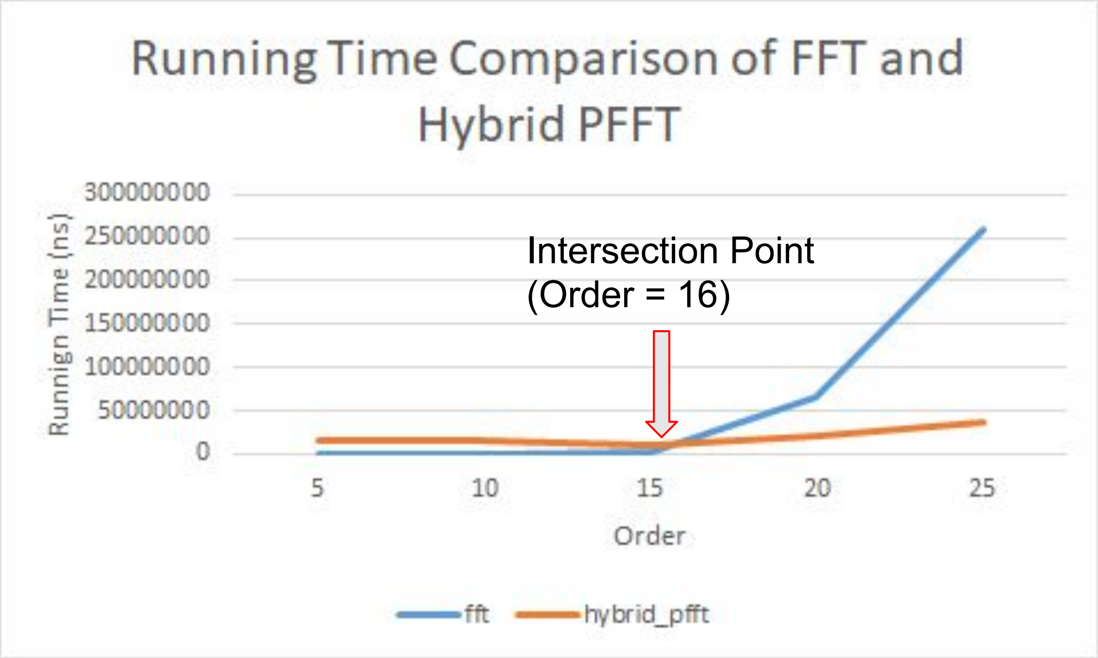


Implementation (Revised)

Revision: Calculate the array using parallel FFT, but limit the number of layers



Implementation (Revised)



	fft	hybrid_pfft
5	27268	14180074
10	59348	14499913
15	2319717	10200851
20	66233349	19368399
25	258501644	36931006

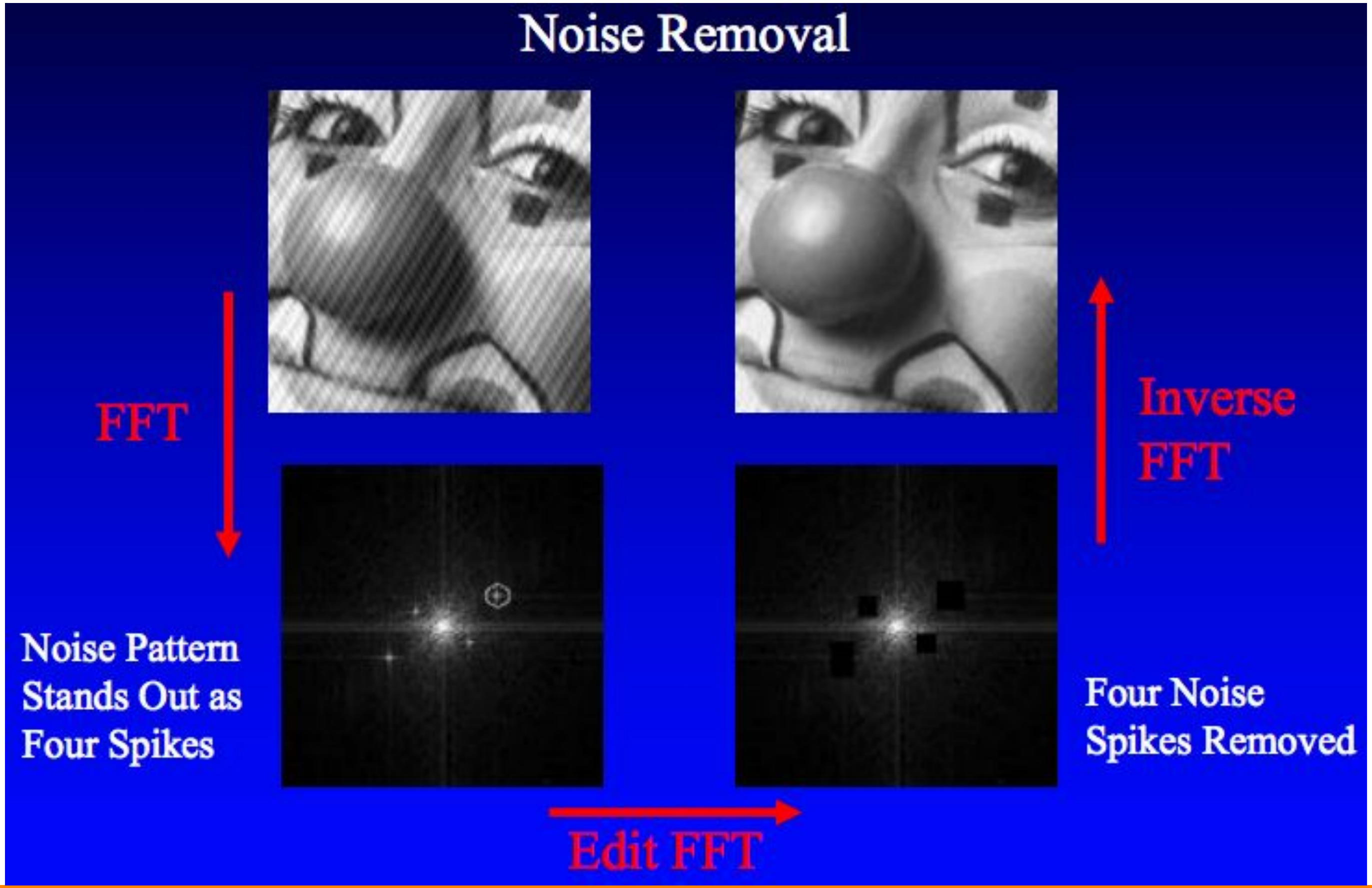
Implementation (Revised)

Be careful

- Decide the optimal division of the tasks accordingly
- Establish the procedure mapping before all the calculation

Applications

There are many applications one example is image processing.



Available Packages

Don’t try to write your own FFT packages!!!

FFT is very complicated, we only show you one scenario when N is a power of 2. What if the input has prime length etc? Many algorithms to handle those cases.

FFTW (Fastest Fourier Transform in the west) have you covered. It can auto generate the kernel that is suitable for your input/machine etc.

Open Issues

1. Optimization algorithm(s) to find the best (layers, N) combination
2. Hardware support for the parallel FFT algorithm

References

[1] D.N. Rockmore. “The FFT - an algorithm the whole family can use.”, Dartmouth College. Hanover, NH. Oct 11 1999.

[2] Cooley, James W.; Tukey, John W. (1965). "An algorithm for the machine calculation of complex Fourier series". *Math. Comput.* **19**: 297–301.

[3] P. Duhamel M. Vetterli Fast Fourier Transforms: A tutorial review and a state of the art, Signal Processing, 1990

[4] Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Cliffor Introduction to Algorithms, Third Edition 2009

Questions

1. What is the meaning of radix in FFT?
2. When people mentioned FFT, which specific algorithm are they generally referring to?
3. Why is the deepest full binary tree based parallel FFT slower than sequential FFT?