Penerapan Algoritma Divide and Conquer untuk Analisis Frekuensi Gelombang Sinusoidal menggunakan Cooley Tukey Fast Fourier Transform Algorithm

Fakih Anugerah Pratama - 13521091
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521091@std.stei.itb.ac.id

***Abstract*— Being known for its vast use in solving problems, Fast Fourier Transform algorithm is currently one of the most popular algorithms. Many would also agree that Fast Fourier Transform algorithm is also one of the best algorithms ever invented. Fast Fourier Transform algorithm is implemented using the concept of Divide and Conquer algorithm as to why it has a time complexity of N log N. In this paper, we will analyze sinusoidal waves using Fourier Transform implemented with Radix-2 Cooley Tukey Fast Fourier Transform algorithm.**

Keywords—Fourier Transform; Divide and Conquer; Sinusoidal Wave; Dynamic Programming;

I. PENDAHULUAN

Transformasi Fourier adalah sebuah transformasi yang dapat mengubah sebuah fungsi menjadi bentuk berbeda yang merepresentasikan frekuensi-frekuensi yang dimiliki oleh fungsi asal. Sebuah fungsi gelombang yang dipetakan dalam *time domain* akan memberikan informasi mengenai simpangan yang dihasilkan oleh fungsi untuk waktu masukan tertentu. Menggunakan transformasi Fourier pada fungsi asal untuk interval terbatas tertentu akan menghasilkan sebuah kumpulan **N** titik-titik yang dapat digambarkan sebagai sebuah gelombang dalam domain frekuensi (*frequency domain*). Kumpulan titik ini dapat ditransformasikan kembali menggunakan inverse transformasi Fourier menghasilkan sampel **N** titik awal yang menjadi masukan fungsi transformasi.

Transformasi Fourier banyak digunakan dalam proses yang membutuhkan interpolasi fungsi dan perataan sinyal. Beberapa contohnya adalah *processing* terhadap sinyal suara dalam *voice noise-filtering*. Selain itu, menggunakan transformasi Fourier dua dimensi, piksel dengan frekuensi spasial tinggi dapat dengan mudah dihilangkan dari sebuah gambar yang diproses.

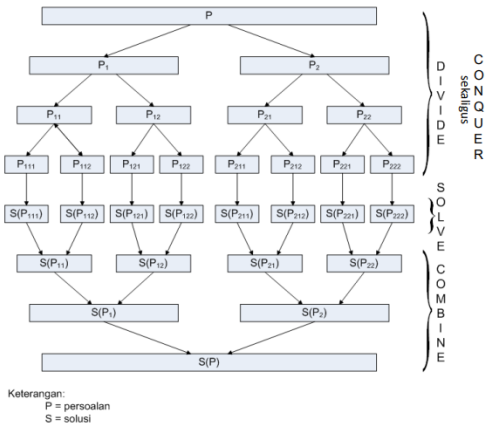
Salah satu bentuk transformasi Fourier adalah transformasi Fourier Diskrit yang dapat mengubah **N** titik hasil sampel dari sebuah fungsi sinusoid menjadi **N** buah titik sampel dari fungsi transformasi Fourier waktu-diskrit. Hasil dari transformasi tersebut adalah kombinasi linear dari titik-titik sampel yang menjadi masukan.

Untuk menghitung transformasi Fourier dengan cepat, dimanfaatkan lah algoritma *Divide and Conquer*. Menggunakan DnC, algoritma dapat melakukan operasi penghitungan dengan lebih efisien. Algoritma Fast Fourier Transform yang dihasilkan akan memiliki kompleksitas waktu N log N, berbeda dengan kompleksitas N^2 yang dihasilkan oleh kalkulasi transformasi Fourier Diskrit (DFT).

II. TEORI DASAR

A. *Divide and Conquer*

Algoritma *Divide and Conquer* adalah sebuah algoritma yang bekerja dengan membagi sebuah kasus menjadi kasus-kasus yang lebih kecil secara rekursif dengan kemiripan persoalan yang mirip (*Divide*). Upakasus hasil pembagian tadi akan diselesaikan menggunakan algoritma tertentu untuk mendapatkan hasil yang diinginkan. Bersamaan dengan selesainya kasus yang lebih kecil, kasus-kasus tersebut akan dikombinasikan kembali menjadi sebuah kasus yang memiliki ukuran yang lebih besar dan akan diselesaikan Kembali secara rekursif sehingga didapatkan sebuah hasil perhitungan yang mewakili perhitungan persoalan semula.



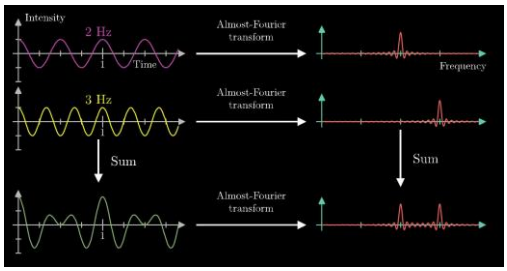
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf

Kompleksitas algoritma Divide and Conquer adalah

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \ldots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

T(n) : kompeksitas persoalan yang memiliki ukuran n
g(n) : kompleksitas upa-persoalan terkecil
$T(n_1) + T(n_2) + \ldots + T(n_r)$: kompleksitas waktu pemrosesan setiap upa-persoalan
f(n) : kompleksitas waktu penggabungan (*combine*) masing-masing upa-persoalan

B. *Fourier Transform*



Sumber: https://www.3blue1brown.com/lessons/fourier-transforms

Transformasi Fourier adalah sebuah transformasi yang mengubah sebuah fungsi dengan domain waktu menjadi bentuk yang mewakili titik-titik yang berada dalam domain frekuensi.

Fourier transform

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\, e^{-i2\pi\xi x}\, dx. \quad \textbf{(Eq.1)}$$

Sumber: https://en.wikipedia.org/wiki/Fourier_transform

Bilangan kompleks $\hat{f}(\xi)$ mewakili transformasi fungsi f(x) di frekuensi ξ.

Sebuah fungsi sinusoid f(x) juga dapat dibentuk Kembali dari kumpulan bilangan kompleks $\hat{f}(\xi)$. Sehingga, sifat ini disebut sebagai invers dari transformasi Fourier.

Fourier inversion integral

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)\, e^{i2\pi\xi x}\, d\xi, \quad \forall\, x \in \mathbb{R},$$

Sumber: https://en.wikipedia.org/wiki/Fourier_transform

C. *Discrete Fourier Transform*

Transformasi Fourier Diskrit dapat mengubah **N** Diskrit buah titik sampel dari sebuah fungsi sinusoid menjadi **N** buah titik sampel transformasi Fourier Diskrit.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn}$$

Sumber: https://en.wikipedia.org/wiki/Discrete_Fourier_transform

III. Implementasi

A. *Algoritma Discrete Fourier Transform (DFT)*

Discrete Fourier Transform diimplementasikan menggunakan bruteforce terhadap semua titik untuk sampel kumpulan titik masukan. Transformasi ini akan menghasilkan sebuah kumpulan titik Diskrit yang mewakili nilai sebuah bilangan imajiner hasil penghitungan Fourier.

```
const math = require('mathjs')

const dft = (X) => {
    for (let i = 0; i < X.length; i++) {
        if (!math.isComplex(X[i])) {
            X[i] = math.complex(X[i], 0)
        }
    }

    let result = []

    for (let k = 0; k < X.length; k++) {
        result[k] = 0

        for(let n = 0; n < X.length; n++) {
            result[k] = math.add(result[k],
math.multiply(X[n],
math.exp(math.multiply(-2 * math.pi * k /
X.length, math.i))));
        }
    }

    return result
}

const idft = (X) => {
    for (let i = 0; i < X.length; i++) {
        if (!math.isComplex(X[i])) {
            X[i] = math.complex(X[i], 0)
        }
    }
```

```
    let result = []


    for (let k = 0; k < X.length; k++) {
        result[k] = 0

        for(let n = 0; n < X.length; n++) {
            result[k] = math.add(result[k],
math.multiply(X[n],
math.exp(math.multiply(2 * math.pi * k /
X.length, math.i))));
        }
    }

    return result
}
```

B. Algoritma Fast Fourier Transform Cooley Tukey

Fast Fourier Transform Cooley Tukey atau Radix-2 diimpelementasikan menggunakan divide and conquer dengan membagi kumpulan titik menjadi dua bagian yaitu bagian dengan indeks yang ganjil dan genap. Setiap sub-bagian tersebut akan dievaluasi secara rekursif dengan fungsi yang sama (divide) sehingga akan mengembalikan dua buah nilai **xe** dan **xo** yang mewakili hasil rekursi transformasi cepat Fourier untuk upapersoalan dengan tingkat satu di bawahnya. Kedua nilai dihitung menjadi sebuah nilai akhir sebagai lineariasi dari kumpulan nilai yang dihasilkan oleh rekursi anakan. Hasil akhir dari fungsi ini adalah sebuah kumpulan titik yang mewakili sebuah bilangan imajiner hasil perhitungan transformasi Fourier.

```
const math = require('mathjs')

const fftdit = (X) => {
    if (X.length == 1) return [X[0]]

    let xe = [], xo = []
    for (let i = 0; i < X.length; i++) {
        if (i % 2 == 0) // odd
            xo.push(X[i])
        else
            xe.push(X[i])
    }

    xe = fftdit(xe)
    xo = fftdit(xo)

    let result = []

    for (let k = 0; k < X.length/2; k++) {
        let p = xo[k];
        let q = math.multiply(xe[k],
math.exp(math.multiply(-2 * math.pi * k /
X.length, math.i)));

        result[k] = math.add(p, q)
        result[k + X.length/2] =
math.subtract(p, q)
    }

    return result
}

const ifftdit = (X) => {
    if (X.length == 1) return [X[0]]

    let xe = [], xo = []
    for (let i = 0; i < X.length; i++) {
        if (i % 2 == 0) // odd
            xo.push(X[i])
        else
            xe.push(X[i])
    }

    xe = ifftdit(xe)
    xo = ifftdit(xo)

    let result = []

    for (let k = 0; k < X.length/2; k++) {
        let p = xo[k];
        let q = math.multiply(xe[k],
math.exp(math.multiply(2 * math.pi * k /
X.length, math.i)));

        result[k] = math.add(p, q)
        result[k + X.length/2] =
math.subtract(p, q)
    }

    return result
}
```

```
const fftRadix2 = (X) => {
    if (X.length != 0 &&
!math.isComplex(X[0]))
        return fftdit(X.map(e =>
math.complex(e, 0))).map(e => e.toVector())

    return fftdit(X).map(e => e.toVector())
}

const ifftRadix2 = (X) => {
    return ifftdit(X).map(e =>
math.divide(e, X.length))
}
```

C. Display Chart

```
import { Line } from 'react-chartjs-2'
import { fft, ifft } from 'fft-js'
import {fftRadix2, ifftRadix2} from
'../fft/fft-r2'

import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
} from 'chart.js';


ChartJS.register(
    CategoryScale,
    LinearScale,
    PointElement,
    LineElement,
    Title,
    Tooltip,
    Legend
  );

  const options = {
    responsive: true,
    plugins: {
      legend: {
        position: 'top',
      },
      title: {
        display: true,
        text: 'Wave Frequency Analysis',
      },
    },
  };

  var labels = []
  var sinusoidWave = []
  for (let i = 0; i < 512; i++) {
    labels.push(i)
    sinusoidWave.push(Math.sin(3 * i / 180
* Math.PI ) + Math.sin(i / 180 * Math.PI ))
  }
  let dataR2 = fftRadix2(sinusoidWave)
  let dataIR2 = ifftRadix2(dataR2)


  let dataF = fft(sinusoidWave)


const data = {
          labels,
        datasets: [
          {
            label: 'Fourier Frequency',
            data: dataR2.map(e =>
Math.sqrt(e[0] * e[0] + e[1] * e[1])),
            borderColor: 'rgb(255, 99,
132)',
            backgroundColor: 'rgba(255, 99,
132, 0.5)',
          },
          {
            label: 'Sinusoid Wave',
            data: sinusoidWave.map(e => e *
128),
            borderColor: 'rgb(110, 99,
252)',
            backgroundColor: 'rgba(255, 99,
132, 0.5)',
          },
        ],
      }
```

IV. Hasil dan Analisis

A. *Efisiensi Algoritma*

Perbandingan efisiensi algoritma transformasi Fourier dilakukan dengan membandingkan waktu eksekusi yang dibutuhkan oleh Discrete Fourier Transform (bruteforce), Radix-2 Fast Fourier Transform, dan Fast Fourier Transform Algorithm optimal milik *library* FFT-JS.

Berikut adalah hasil perhitungan untuk 128 dan 512 titik yang diambil secara random.

```
PS D:\Obsidian\fakihap\Kuliah\IF2211 - Stima\Makalah\fft-radix2\Frequency-Analysis-with-Fast-Fourier-Transform> node .\fft\fft-test.js
time elapsed for DFT:   71.27300000190735 ms
time elapsed for FFT:   3.353100001811981 ms
time elapsed for FFTJS: 1.471500039100647 ms
time elapsed for IDFT:  55.620899975299835 ms
time elapsed for IFFT:  17.974600001707077 ms
time elapsed for IFFTJS: 1.0257999897003174 ms
```

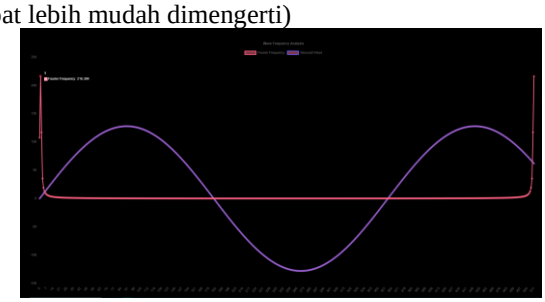
```
PS D:\Obsidian\fakihap\Kuliah\IF2211 - Stima\Makalah\fft-radix2\Frequency-Analysis-with-Fast-Fourier-Transform> node .\fft\fft-test.js
time elapsed for DFT:   660.3794000148773 ms
time elapsed for FFT:   12.109000027179718 ms
time elapsed for FFTJS: 4.541999995708466 ms
time elapsed for IDFT:  669.2440999746323 ms
time elapsed for IFFT:  54.23059999942795 ms
time elapsed for IFFTJS: 3.07150000333786 ms
```

Dapat diamati bahwa perbedaan waktu eksekusi DFT dan FFT sangatlah signifikan.

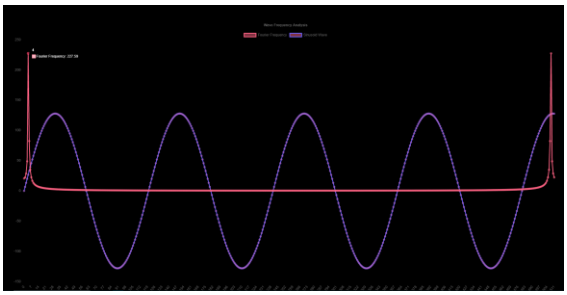
Untuk besar sampel 128 titik, bruteforce DFT memerlukan waktu kurang lebih 71ms. Berbeda jauh dengan FFT yang memanfaatkan divide and conquer sehingga hanya memerlukan waktu eksekusi selama kurang lebih 3ms.

B. *Analisis Gelombang Hasil Transformasi Fourier*

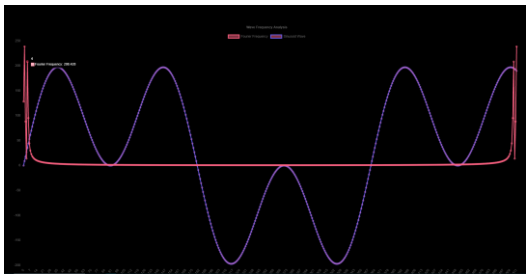
Fungsi f(x) = sin(x) di-plot menjadi sebuah gelombang sinusoidal dengan frekuensi 1Hz (Lihat grafik dengan warna ungu sebagai grafik simpangan vs waktu). (Catatan : gelombang sinusoidal pada gambar di bawah dibuat memiliki amplitude yang sangat besar agar ilustrasi dapat lebih mudah dimengerti)



Hasil transformasi Fourier gelombang tersebut juga di-plot menjadi sebuah gelombang berwarna merah yang menyatakan intensitas pada domain frekuensi sinyal. Terlihat gelombang memiliki maxima di sekitar “frekuensi” 1.



Grafik di atas menggambarkan grafik fungsi f(x) = sin(3x) dan hasil transformasi Fourier-nya. Perhatikan Kembali bahwa transformasi memiliki puncak di sekitar “frekuensi” 3Hz



Grafik di atas menggambarkan fungsi f(x) = sin(x) + sin(3x). Dari gelombang hasil transformasi Fourier-nya dapat dilihat bahwa muncul 2 puncak yaitu di sekitar frekuensi 1Hz dan 3Hz, sesuai dengan fungsi sinusoid asalnya.

Analisis di atas menunjukkan bahwa transformasi Fourier dapat menunjukkan frekuensi yang dimiliki oleh sebuah gelombang sinusoidal. *Properties* di atas lah yang membuat transformasi banyak dimanfaatkan dalam kehidupan sehari-hari untuk menyelesaikan permasalahan yang berkaitan dengan gelombang, seperti *audio processing*.

Analisis lebih lanjut juga dapat dilakukan dengan melakukan manipulasi terhadap hasil transformasi Fourier. Hasil transformasi fungsi f(x) = sin(x) + sin(3x) dapat dimanipulasi sehingga salah satu puncaknya menjadi rata (mendekati 0). Setelah maipulasi dilakukan, gunakan Inverse Fast Fourier Transform terhadap transformasi tersebut untuk menyusun ulang sebuah gelombang sinusoid baru yang memiliki fungsi berbeda. Contohnya f(x) = sin(x), karena puncak di 3-4Hz dimanipulasi sehingga mendekati 0.

V. Penutup

A. *Kesimpulan*

Transformasi Fourier memiliki sangat banyak kegunaan. Salah satu kegunaan yang ditampilkan dalam makalah ini adalah untuk menganalisa sebuah gelombang. Penghitungan nilai hasil Transformasi Fourier sebuah gelombang sinusoidal dapat dilakukan dengan beberapa cara. Menggunakan brute-force kepada sekumpulan titik diskrit (DFT) menghasilkan sebuah algoritma dengan kompleksitas $O(N^2)$. Sedangkan, penggunaan Divide and

Conquer untuk mempersingkat waktu kalkulasi berhasil membuat algoritma penghitungan transformasi Fourier lebih cepat dengan kompleksitas waktu O(N log N).

B. Saran

Pemanfaatan algoritma divide and conquer dalam penghitungan nilai hasil transformasi Fourier yang ditunjukkan dalam makalah ini masih bisa untuk didalami lebih lanjut dan ditingkatkan. Selain itu, masih banyak optimisasi algoritma yang dapat dilakukan untuk mendapatkan algoritma Fast Fourier Transform yang jauh lebih cepat dari yang berhasil dihasilkan melalui makalah ini.

LINK VIDEO YOUTUBE

https://youtu.be/EAwcISqDIgU

LINK GITHUB REPOSITORY

fakihap/Frequency-Analysis-with-Fast-Fourier-Transform (github.com)

VI. REFERENSI

[1] Algoritma Divide and Conquer (itb.ac.id) oleh Rinaldi Munir

[2] Fourier transform - Wikiversity

[3] Radix-2 Fast Fourier Transform, MATH3511 - Numerical Analysis II, Spring semester 2019 (uconn.edu)

[4] Fourier Transform - Definition, Formula, Properties, Applications and Examples (byjus.com)

[5] Playing with Discrete Fourier Transform Algorithm in JavaScript | by Oleksii Trekhleb | Medium

[6] Basic implementation of Cooley-Tukey FFT algorithm in Python (github.com)

[7] Image Transforms - Fourier Transform (ed.ac.uk)

[8] https://www.nayuki.io/res/free-small-fft-in-multiple-languages/fft.js

[9] Microsoft PowerPoint - 23-Divide-and-Conquer-the-FFT (washington.edu)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

Fakih Anugerah Pratama
13521091