Fast Fourier Transform

Perbandingan Algoritma Fast Fourier Transform Dengan Algoritma Dasar Dalam Mengalikan Polinomial

Andika Kusuma
Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13515033@std.stei.itb.ac.id

***Abstract*—Polinomial sering sekali digunakan perhitungan-perhitungan matematika. Polinomial seharusnya mempunyai sebnayak n akar sesuai *dengan degree* polinomialnya. Namun, beberapa dari akarnya ada yang merupakan bilangan kompleks. *Proses* perkalian antara 2 buah polinomial sendiri membutuhkan waktu yang cukup besar dengan cara *brute force.* Namun, waktu ini dapat dipercepat dengan menggunakan algoritma *Fast Fourier Transform.* Algoritma ini pun digunakan untuk memproses sinyal digital.**

Keywords*—polinomial, *Fast Fourier Transform*, algoritma, *Digital Signal Processing

I. Pendahuluan

DFT (*Discrete Fourier Transform)* adalah salah satu cara yang dapat digunakan untuk melakukan proses analisis sinyal input untuk mengetahui karakterisitik sinyal. Secara singkat, DFT mengubah sinyal analog menjadi sinyal diskrit dalam domain waktu, lalu mentranformasikan menjadi dikrit dalam domain frekuensi dengan cara mengalihkan sinyal dengan fungsi *kernel*.

Fast Fourier Transform digunakan untuk mempercepat proses analisi sinyal. FFT lebih cepat jika dibandingkan dengan DFT karena cara kerja FFT yang membagi sinyal yang ada menjadi beberapa bagian, lalu melakukan analisis di masing-masing bagian lalu menggabungkan hasil-hasil tersebut, seperti algoritma Divide and Conquer.

Digital Signal Processing mengacu pada bermacam-macam cara untuk memproses sinyal dengan meningkatkan akurasi untuk komunikasi *digital*. Sebagai contoh, pada saat memproses sinyal analog dari siaran televisi, sinyal analog akan diubah terlebih dahulu menjadi sinyal digital dengan asumsi voltase dan arus yang sudah pasti. Namun, karena adanya gangguan, nilai voltase dan arus ini menjadi tidak pasti sehingga dibutuhkan DSP untuk mengembalikan nilai tersebut menjadi nilai yang sudah diperkirakan.

II. Landasan Teori

A. Polinomial

[1] Polinomial adalah bentuk suku-suku yang terhingga dengan masing-masingnya di dalam bentuk $a_i x^i$ dengan a_i adalah konstanta riil dan i adalah bilangan bulat tidak negatif. Secara umum, polinomial berbentuk seperti:

$$f(x) = a_0 x^0 + a_1 x^1 + a_2 x^2 + \cdots + a_n x^n$$

- Suku-suku dari polinomial tersebut adalah

$$a_0 x^0, a_1 x^1, a_2 x^2, \ldots, a_n x^n$$

- Peubahnya adalah x
- Koefisien dari x^k adalah a_k
- Konstantanya adalah a_o (disebut juga koefisien x^0)
- Derajat dari polinomial tersebut adalah n (dengan syarat a_n tidak nol)

Contoh perkalian polinomial dengan derajat 4:

$(x^4 + x^3 + x^2 + x^1 + 1)(x^4 + x^3 + x^2 + x^1 + 1)$ didapat dengan cara mengalikan masing-masing suku di masing-masing polinomial sehingga akan dilakukan (n+1) x (n+1) operasi dengan n adalah derajat polinomial menghasilkan polinomial dengan derajat n + n.

- $x^4 x^4 = x^8$
- $x^4 x^3 = x^7$
- ...
- 1 1 = 1

Sehingga diperoleh hasilnya adalah $x^8 + 2x^7 + 3x^6 + 4x^5 + 5x^4 + 4x^3 + 3x^2 + 2x + 1$.

B. Bilangan Kompleks dan *Roots of Unity*

[2] Bilangan Kompleks berbentuk a + bi dengan $i^2 = -1$. *Roots of unity* sendiri adalah akar-akar dari persamaan $x^n = 1$. *Roots of unity*-nya adalah $\omega_n, \omega_n^2, \ldots, \omega_n^n$ dengan $(\omega_n^k) = e^{\frac{2\pi i}{n}k}$. *Roots of unity* berbentuk perpangkatan dari 1 sampai n karena *roots of unity* mempunyai sifat:

Jika z adalah *roots of unity*, maka z^k juga merupakan *roots of unity*.

Bukti: $(z^k)^n = (z^n)^k = 1^k = 1$

C. *Halving Lemma*

[3] Halving Lemma berbunyi sebagai berikut:

Jika kita mengkuadratkan setiap dari *roots of unity* dengan derajat n, kita akan memperoleh setiap dari *roots of unity* dengan derajat n/2, masing-masing 2.

Bukti: $(\omega_n^k)^2 = e^{\frac{2\pi i}{n} \cdot 2k} = e^{\frac{2\pi i}{n/2} \cdot k} = (\omega_{n/2})^k$

D. *DFT Invers*

[4] DFT Invers digunakan untuk mencari koefisien dari polinomial A(x) jika diketahui nilai-nilai A(x) di titik-titik *roots of unity* dari derajat n. Caranya adalah menghitung $A = V_n^{-1} \hat{A}$ dengan A adalah vector berisi koefisien A(x), $\hat{A}$ adalah vector berisi nilai-niali A(x) di *roots of unity*, dan V_n^{-1} adalah matrix Vandermonde invers.

$$V_n^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \cdots & \omega_n^{-(n-1)^2} \end{pmatrix}$$

E. *Fast Fourier Transform*

[4] [5] *Fast Fourier Transform* menggunakan Halving Lemma untuk mengkomputasi perkalian 2 polinomial. Evaluasi dengan FFT adalah sebagai berikut.

1. Menghitung nilai p(x), q(x) di ω_{2n}, ω_{2n}^2, … , ω_{2n}^{2n-1} dengan menggunakan Halving Lemma dimana:

$$A(x) = a_0x^0 + a_1x^1 + a_2x^2 + \cdots + a_nx^n$$
$$A(x) = (a_0x^0 + a_2x^2 + \cdots + a_{n-1}x^{n-1}) + x(a_1x^0 + a_3x^2 + \cdots + a_nx^{n-1})$$
$$A(x) = A_{genap}(x^2) + x\,A_{ganjil}(x^2)$$

   Sebagai contoh, menghitung nilai $P(\omega_{2n})$ dapat diperoleh dari menghitung nilai P_{genap} dari ω_n dan seterusnya. Langkah ini memakan waktu sebesar 2 * 2n * log (n), dimana log (n) adalah waktu perhitungan satu nilai p(x) atau q(x) di satu titik *roots of unity* dari 2n. Kompleksitas waktunya O(n log n).
2. Menghitung nilai dari (p.q)(x) yaitu p(x) . q(x) di 2n titik dari *roots of unity* dari 2n, yaitu ω_{2n}, ω_{2n}^2, … , ω_{2n}^{2n-1}. Langkah ini memakan waktu 2n (O(n)).
3. Interpolasi polinomial p.q dengan menggunakan DFT invers untuk memperoleh koefisien c_0, c_1, c_2, … , c_{2n-2}. Langkah ini memakan waktu O(n log n).

Sehingga FFT membutuhkan waktu O(n log n) secara keseluruhan yang lebih cepat dari perkalian *brute force*.

III. Perbandingan Algoritma Brute Force dan Fast Fourier Transform

Akan dibahas pada bab ini perbandingan antara algoritma Brute Force dan Fast Fourier Transform dalam menghitung perkalian 2 buah polinomial.

A. *Algoritma Brute Force*

Berikut ini adalah kode perkalian polinomial dalam bahasa pesudocode menggunakan *brute force*. Kode dalam bahasa C++ dapat dilihat di http://ideone.com/pHuUaJ.

```
// author : Andika Kusuma

// fungsi untuk mengembalikan selisih waktu
antara 2 buah timespec
timespec diff(timespec start, timespec end){
    timespec temp;
    if ((end.tv_nsec - start.tv_nsec) < 0)
        temp.tv_sec = end.tv_sec - start.tv_sec -
1;
        temp.tv_nsec = 1000000000 + end.tv_nsec -
start.tv_nsec;
    else
        temp.tv_sec = end.tv_sec - start.tv_sec;
        temp.tv_nsec = end.tv_nsec -
start.tv_nsec;
    endif
    -> temp;
}

// program utama
        int np, nq; // degree dari polinomial p
dan q
        input(np);
        nq <- np; // ukuran p dan q sama
        input(polinomp);
        input(polinomq);
        // perhitungan perkalian
        timespec start, stop, duration;
    clock_gettime(CLOCK_REALTIME, &start);
        // buat array menyimpan hasil perkalian
        bool first <- true;
        for(int i = 0; i < np + nq; i++)
                a[i] <- 0;
                for(int j = 0; j < np; j++)
                        if(i - j >= 0 && i - j <
nq)
                                a[i] <- a[i] +
ap[j] * aq[i - j];
                        endif
                }
                endfor
                output(a[i]);
        endfor
    clock_gettime(CLOCK_REALTIME, &stop);
    duration <- diff(start, stop);
    output(duration);
```

Algoritma *brute force* menghitung setiap komponen koefisien dari polinomial hasil perkalian dari 0 hingga n+m-1. Sedangkan, masing-masing koefisien dihitung dengan iterasi setiap koefisien dari masing-masing polinomial, sehingga diperoleh kompleksitas waktunya adalah O(m*n).

B. *Algoritma Fast Fourier Transform*

Berikut ini adalah pseudocode algoritma *FFT*. [6] Kode ini merupakan extensi dari kode yang terdapat di rosettacode.org dengan mengimprovisasi di bagian invers FFT menggunakan FFT biasa. Kode dalam bahasa C++ dapat dilihat di http://ideone.com/Pqe9nv.

```
// Deklarasi dan definisi struktur data
const double PI = 3.14159265358979238460;

typedef std::complex<long double> Complex;
typedef std::valarray<Complex> CArray;

// tranformasi FFT menjadi nilai nilai di roots
of unity-nya
void fft(CArray& x, Complex w)
{
        if banyakelemen = 1 then
                // tidak ada yang dilakukan
        else
                // divide polinomialnya
                even = bagianeven(x);
                odd = bagianodd(x);
                // conquer keuda polinomial
                fft(even, w*w);
                fft(odd, w*w);
                // karena primitive rootnya
adalah w*w
                // combine
                root <- 1; // menyimpan root of
unity pangkat k
                for k = 0; k < N/2; k++
                do
                        x[k] = even[k] + root *
odd[k];
                        x[k+N/2] = even[k] - root
* odd[k];
                        root *= w;
                endfor
        endif
}

// fungsi untuk mengembalikan selisih waktu
antara 2 buah timespec
timespec diff(timespec start, timespec end){
    timespec temp;
    if ((end.tv_nsec - start.tv_nsec) < 0)
        temp.tv_sec = end.tv_sec - start.tv_sec
- 1;
        temp.tv_nsec = 1000000000 + end.tv_nsec
- start.tv_nsec;
    else
        temp.tv_sec = end.tv_sec -
start.tv_sec;
        temp.tv_nsec = end.tv_nsec -
start.tv_nsec;
    endif
    -> temp;
}
```

```
// fungsi untuk mendapatkan primitive
root of unity
Complex getPrimitiveRootOfUnity(int gen)
{
    -> Complex(cos(2*PI/gen),
sin(2*PI/gen));
}

// program utama
    int m;
    input(m);
    input(polinomialp);
    input(polinomialq);
    timespec start, stop, duration;
    clock_gettime(CLOCK_REALTIME,
&start);
    // mulai perkalian
    Complex s <-
getPrimitiveRootOfUnity(2*m);
    // melakukan prosedur fft
    fft(p,s);
    fft(q,s);
    // buat polinomial hasil kali fft p
dan fft q
    for(int i = 0; i < 2*m; i++){
        test[i] <- p[i] * q[i];
    }
    s_ <- primitiveroot ^ (-1);
    // invers fft
    fft(data, s_);
    // selesai
    clock_gettime(CLOCK_REALTIME,
&stop);
    duration = diff(start, stop);
    output(duration);
    for(int i = 0; i < 2*m - 1; i++){
        output(koefisien/(2*m));
       // scaling terhadap ukuran
    }
```

C. *Perbandingan Algoritma Fast Fourier Transform dan Brute Force*

Dari hasil eksekusi kedua program pada beberapa input, terlihat bahwa algoritma *Fast Fourier Transform* mengeluarkan hasil perkalian polinomial dengan lebih cepat.

- Hasil eksekusi *brute force*

```
1
2
3
4
5
6
7
8
7
6
5
4
3
2
1
0
BF :0.000122259 s
```

Berikut di atas adalah contoh dengan data input m = 8 dan setiap koefisien polinomialnya = 1.

- Hasil eksekusi *FFT*

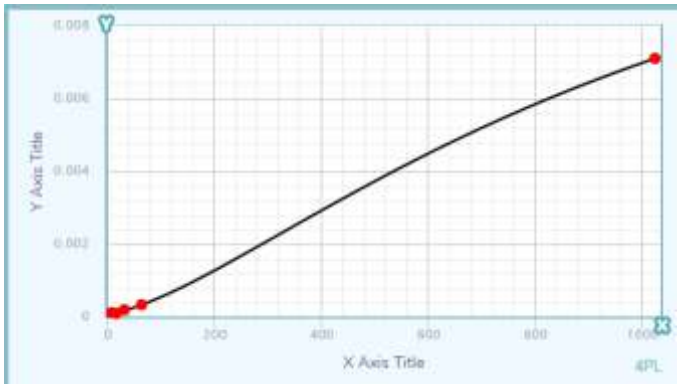
```
FFT :0.000021430 s
(1,-7.70027e-16)
(2,-1.04035e-15)
(3,-2.09549e-16)
(4,3.26169e-15)
(5,1.05368e-15)
(6,5.60224e-15)
(7,6.67663e-15)
(8,1.23161e-14)
(7,7.70027e-16)
(6,1.04035e-15)
(5,2.09549e-16)
(4,-3.26169e-15)
(3,-1.05368e-15)
(2,-5.60224e-15)
(1,-6.67663e-15)
```

Catatan :Untuk algoritma *FFT*, koefisien polinomial yang diambil adalah bagian riil dari bilangan kompleksnya.

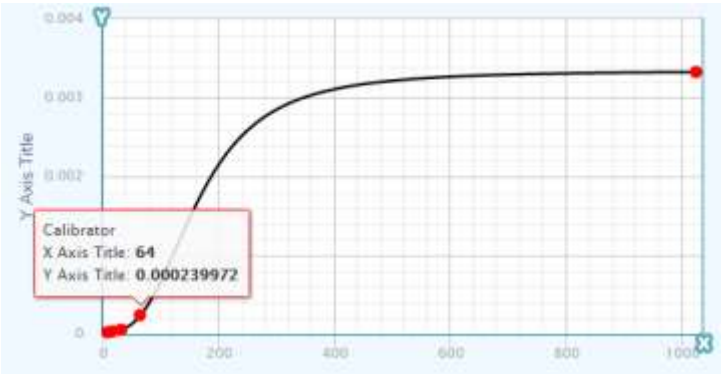
Berikut di bawah ini table perbandingan untuk beberapa data besar suku banyak.

Besar input (m)	Waktu eksekusi *Brute Force*	Waktu eksekusi *FFT*
8	0.000122259 s	0.000021430 s
16	0.000098867 s	0.000035163 s
32	0.000199498 s	0.000055090 s
64	0.000337260 s	0.000239972 s
1024	0.007108950 s	0.003329481 s

Tabel Perbandingan kecepatan eksekusi algoritma *Brute Force* dan *FFT*



Gambar 1 Kurva waktu terhadap besar input *Brute Force*



Gambar 2 Kurva waktu terhadap besar input *FFT*

Dari kedua kurva, terlihat bahwa semakin besar input akan sangat berpengaruh untuk algoritma *Brute Force*, dimana pengingkatan besar input tidak terlalu signifikan untuk algoritma *FFT*. Dari sini terlihat bahwa algoritma *FFT* lebih mangkus.

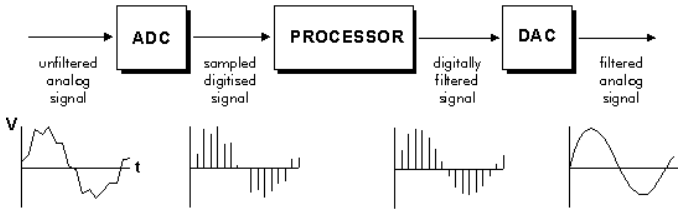
D. Algoritma Lain

[7] Dalam proses perkalian polinomial sendiri, terdapat algoritma lain yang memiliki kompleksitas waktu $O(n^{\log 3})$ yaitu dengan cara Divide dan Conquer bagian menjadi suku dengan pangkat lebih tinggi dari n/2 dan sisanya (sangat mirip dengan Fast Fourier Transform). Algoritma ini membutuhkan sedikit trik seperti Strassen's Matrix Multiplication. Algoritma ini tidak akan dibahas lebih lanjut pada makalah ini.

IV. Penerapan Algoritma Fast Fourier Transform Untuk Digital Signal Processing

A. *Kegunaan FFT di Digital Signal Processing*

Secara umum, *FFT* digunakan pada proses penyaringan *noise*. Proses penyaringan atau *filter* ini bertujuan untuk memisahkan sinyal yang benar dan gangguan-gangguan yang tidak penting. Algoritma *FFT* dipakai dalam mengubah sinyal menjadi kumpulan data diskrit yang kemudian diolah. Setelah itu, baru data-data diskrit tersebut diubah kembali menjadi sinyal lagi dengan menggunakan *inverse FFT*.

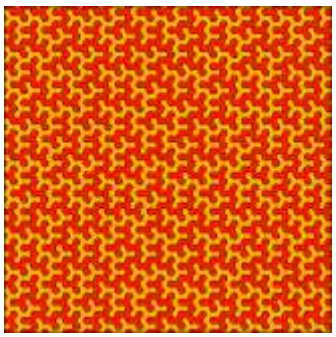


Gambar 3 Proses filter sinyal digital

Secara singkat, sinyal analog yang belum terfilter masuk dan diubah menjadi sinyal digital dengan menggunakan ADC (Analog to Digital Converter). Setelah itu, sinyal digital akan difilter dengan prosesor menjadi sinyal digital yang sudah terfilter. *FFT* digunakan dalam pemrosesan sinyal digital dengan membagi-bagi sinyal datang lalu memroses masing-masing bagian kecil dan menyatukannya kembali. Kemudian, sinyal digital ini akan diubah kembali menjadi sinyal analog dengan DAC (Digital to Analog Converter).

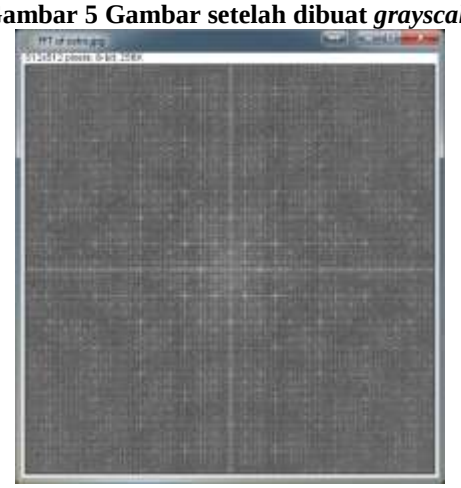
B. *Kegunaan FFT di Image Processing*

[8] *Image processing* adalah fungsi spasial yang bisa dikonversikan ke dalam domain frekuensi. *FFT* dapat digunakan untuk teknik *filtering* dalam *image processing*. Gambar di bawah ini merupakan contoh penggunaan *filtering* yang memanfaatkan *FFT*. Langkah-langkah secara umum adalah transformasikan gambar dengan algoritma FFT, lalu lakukan filter sesuai dengan yang diinginkan, lalu transformasikan kembali gambar dengan *inverse FFT*.



Gambar 4 Gambar sebelum dilakukan filter

Gambar 5 Gambar setelah dibuat *grayscale*



Gambar 6 Gambar setelah ditransformasikan dengan *FFT*

Gambar 7 Gambar setelah difilter dan ditransformasikan kembali dengan *FFT Inverse*

V. Simpulan

Algoritma *Brute Froce* cenderung memiliki kompleksitas waktu yang lebih lambat jika dibandingkan dengan algoritma seperti *Divide and Conquer*. Namun, jika dilihat algoritma *Brute Force* dapat terpikir dengan sangat mudah dan *straightforward*, dimana algoritma *Fast Fourier Transform* sangat sulit untuk dipikirkan dan dicerna.

Algoritma *Fast Fourier Transform* sangat mangkus untuk mengalikan 2 buah polinomial dengan derajat yang sama. Hal ini dikarenakan banyak proses penghitungan yang dilakukan dieefisienkan dengan cara membagi polinomial menjadi penjumlahan polinom-polinom dengan derajat setengahnya. Selain itu, algoritma *Fast Fourier Transform* sangat sering digunakan untuk melakukan *filtering* seperti pada *Digital Signal Processing* maupun *image processing*. Selain itu, algoritma ini juga dapat digunakam untuk menentukan karakteristik fisik dari suatu data sinyal dengan mengubahnya menjadi data diskrit. Hal ini sangat penting untuk mengetahui bagian-bagian yang merupakan gangguan dari sinyal dan bahkan memisahkannya.

VI. Ucapan Terima Kasih

Pertama-tama, penulis berterimakasih dan bersyukur kepada Tuhan Yang Maha Esa atas berkat-Nya sehingga penulis dapat menyelesaikan makalah ini dengan baik. Penulis juga mengucapkan terima kasih kepada orang tua penulis yang selalu membantu secara moral, material, dan doa. Penulis turut mengucapkan terima kasih kepada Bapak Rinaldi Munir, selaku dosen dari mata kuliah Strategi Algoritma yang telah mengajarkan mengenai algoritma-algoritma seperti *Brute Force* dan *Divide and Conquer*. Selain itu, penulis juga berterima kasih kepada teman-teman penulis yang telah membantu secara moral dan doa.

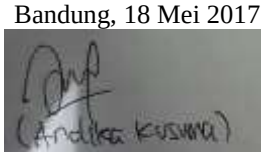
Daftar Pustaka

[1] https://alewoh.com/pengertian-polinom-suku-banyak.php
[2] Lang, Serge (2002). "Roots of unity". Algebra. Springer. pp. 276–277
[3] http://www.cs.fsu.edu/~burmeste/slideshow/chapter32/32-2.html]
[4] http://web.cs.iastate.edu/~cs577/handouts/polymultiply.pdf
[5] https://people.cs.umass.edu/~barring/cs611/lecture/3.pdf
[6] https://rosettacode.org/wiki/Fast_Fourier_transform#C.2B.2B
[7] http://www.geeksforgeeks.org/multiply-two-polynomials-2/
[8] https://martendarmawan.wordpress.com/tag/fast-fourier-transform/

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, 18 Mei 2017



Andika Kusuma