Perkalian Matrix dengan Divide and Conquer dan Algoritma Strassen

Otniel and 13508108
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*author@itb.ac.id*

Penghitungan matrix sangat umum di bidang matematika. Namun, pemodelan matrix jika dihitung secara bruteforce akan menjadi sangat tidak optimal. Jika pemodelan penghitungan ke bentuk matrix sudah mencapai ordo yang sangat besar, akan sangat lama jika dihitung menggunakan metode bruteforce. Untuk mendapatkan hasil dari kalkulasi perkalian matrix dengan cepat digunakan algoritma divide and conquer. Divide and conquer memiliki kelebihan dari segi kompleksitas algoritma yang berdampak pada waktu penghitungan. Namun, jika menggunakan metode penghitungan biasa, kompleksitasnya akan teteap sama. Oleh karena itu dibutuhkan bantuan algoritma strassen untuk mengoptimalkan proses kalkulasi. Dalam makalah in akan dibahas penerapan divide and conquer untuk masalah penghitungan perkalian matrix ini.

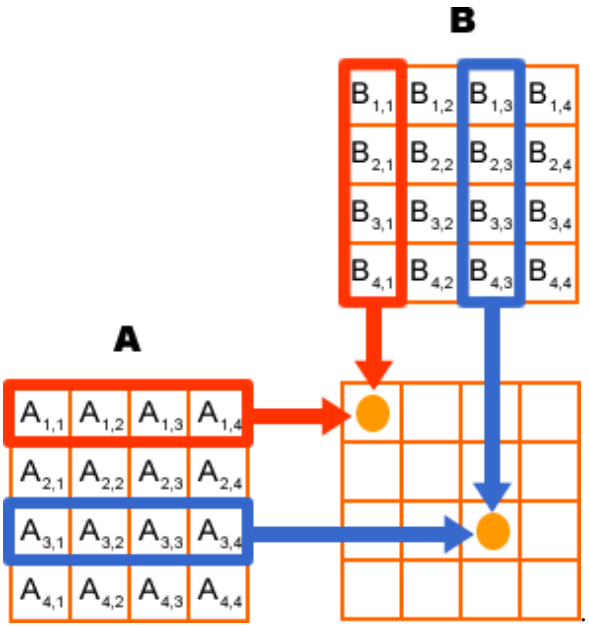
Kata kunci : **matrix, divide and conquer, kompleksitas, algoritma.**

I. PENDAHULUAN

Pemodelan data dengan matriks sudah sangat umum dikenal di dalam dunia science terutama yang berhubungan dengan matematika. Penggunaannya pun sangat bermacam-macam. Salah satunya adalah untuk merepresentasikan posisi suatu tempat dalam kordinat kartesius (bidang x-y).

Matrix dapat dioperasikan, seperti dikurang, ditambah, dan dikalikan (cross dan dot product). Masalah yang ditemui dalam penghitungan matriks adalah pengoptimalan kalkulasi perkalian cross pada dua matriks. Misal kita memiliki dua buah matriks A dan B, dengan A dan B berordo sama, $n \times n$. Penggunaan yang sering digunakan untuk penghitungan ini adalah dengan cara brute force. Namun penggunaan ini tidak cocok digunakan untuk kebutuhan para ilmuan yang membutuhkan pengoperasian matriks dengan cepat.

Pengoperasian perkalian matriks melibatkan penghitungan dari baris dan kolomnya Tiap cell pada kolom di matriks A akan dikalian dengan dengan tiap cell pada matriks B kemudian dijumlahkan



Gambar 1 Perkalian matriks

Hasil dari penjumlahan tersebut akan diletakkan pada cell dengan posisi sesuai dengan baris dan kolom yang dihitung, misal dari matriks A baris 1, dan dari matriks B kolom 1, maka hasil penghitungan akan diletakkan pada cell 1,1.

II. PERBANDINGAN ALGORITMA

Untuk mengetahui mana yang lebih efisien, pertama-tama akan dipaparkan dulu penghitungan dari segi algoritma bruteforce dan dari segi divide and conquer.

A. Penyelesaian dengan Algoritma Brute Force

Misal, diketahui dua matriks, A dan B dengan ordo sama, yaitu $n \times n$, akan dilakukan penghitungan hasil perkalian matriks, maka tiap cell pada baris di matriks pertama akan dikalikan pada tiap cell pada kolom di matriks kedua. Penghitungan diformulasikan seperti berikut ini :

$$c_{i,j} = a_{i,1}b_{1,j} + \cdots + a_{i,n}b_{n,j} = \sum_{k=1}^{n} a_{i,k}b_{k,j}$$

Begitu seterusnya sampai semua cell dikalikan. Berikut adalah algoritma pengalian matriks dengan algoritma bruteforce

```
  procedure PerkalianMatriks(input A, B : Matriks, input n : integer, output C : Matriks)
  {  Mengalikan matriks A dan B yang berukuran n × n, menghasilkan matriks C yang juga berukuran n × n
    Masukan: matriks integer A dan B, ukuran matriks n
    Keluaran: matriks C
  }

  Deklarasi
    i, j, k : integer

  Algoritma
     for i¬1 to n do
       for j¬1 to n do
          C[i,j]¬0    { inisialisasi penjumlah }
          for k ¬ 1 to n do
             C[i,j]¬C[i,j] + A[i,k]*B[k,j]
          endfor
       endfor
   endfor
```

Dengan 3 loop for bersarang, program yang menggunakan algoritma semacam ini akan memiliki kompleksitas T(n) = O(n^3).

Contoh pengoperasian matriksnya adalah seperti berikut :

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

Maka, untuk menghitungnya dilakukan langkah seperti berikut :

$$\begin{aligned} C_{1,1} &= A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} \\ C_{1,2} &= A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2} \\ C_{2,1} &= A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1} \\ C_{2,2} &= A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2}. \end{aligned}$$

Pada perhitungan diatas terlihat terdapat 8 operasi perkalian dan 4 operasi penjumlahan.

B. Penyelesaian dengan Divide and Conquer

Berbeda dengan algoritma brute force yang dikerjakan secara iterative, algoritma divide and conquer dikerjakan secara rekursif.

$$T(n) = \begin{cases} O(1) & n = 1, \\ 8T(n/2) + O(n^2) & n > 1. \end{cases}$$

Seperti pada contoh di atas, penghitungan matriks melibatkan 8 operasi perkalian (divide) dan 4 operasi penjumlahan (conquer) dengan kompleksitas penjumlahan matriks adalah O(n^2). Jadi total kompleksitas penghitungan matriks secara rekurens adalah. Namun pada pengerjaan seperti di atas, masih akan menghasilkan kompleksitas sebesar O(n^3). Karena ada 8 operasi rekursif perkalian $\frac{n}{2} x \frac{n}{2}$, maka persamaan kompleksitasnya akan seperti berikut ini :

$$\boldsymbol{T(n) = 8(T)(n/2) + O}(\mathrm{n}^2)$$

Akhirnya akan didapatkan persamaan

$$\boldsymbol{T(n) = n^{log_2 8} = O(n^3)}$$

Dengan menggunakan metode penghitungan perkalian strassen, akan mengurangi kompleksitas algoritma penghitungan perkalian matriks. Berikut adalah cara penghitungan matriks dengan metode strassen :

Misalkan, diketahui dua buah matriks A dan B yang dikalikan seperti berikut

C = AB dengan **A,B,C** adalah ∈ $\boldsymbol{R^{2n\,x\,2n}}$

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

dengan

$$\boldsymbol{A_{i,j} x B_{i,j} x C_{i,j} \in R^{(2n-1)x(2n-1)}}$$

maka akan didapatkan

$$\begin{aligned} C_{1,1} &= A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} \\ C_{1,2} &= A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2} \\ C_{2,1} &= A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1} \\ C_{2,2} &= A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2}. \end{aligned}$$

Kemudian akan dibentuk matriks-matriks baru seperti di bawah ini :

$$\begin{aligned} \boldsymbol{M_1} &= \boldsymbol{(A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})} \\ \boldsymbol{M_2} &= \boldsymbol{(A_{2,1} + A_{2,2})(B_{1,1})} \\ \boldsymbol{M_3} &= \boldsymbol{(A_{1,1})(B_{1,2} - B_{2,2})} \\ \boldsymbol{M_4} &= \boldsymbol{(A_{2,2})(B_{2,1} - B_{1,1})} \\ \boldsymbol{M_5} &= \boldsymbol{(A_{1,1} + A_{1,2})(B_{2,2})} \\ \boldsymbol{M_6} &= \boldsymbol{(A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})} \\ \boldsymbol{M_7} &= \boldsymbol{(A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})} \end{aligned}$$

Setelah semua matriks didapatkan, akan dihitung $C_{1,1}$,$C_{1,2}$,$C_{1,3}$, dan $C_{1,4}$

$$\begin{aligned} \boldsymbol{C_{1,1}} &= \boldsymbol{M_1 + M_4 - M_5 + M_7} \\ \boldsymbol{C_{1,2}} &= \boldsymbol{M_3 + M_5} \\ \boldsymbol{C_{2,1}} &= \boldsymbol{M_2 + M_4} \\ \boldsymbol{C_{2,2}} &= \boldsymbol{M_1 - M_2 + M_3 + M_6} \end{aligned}$$

Dalam notasi algoritma, akan melibatkan algoritma

strassen. Berikut adalah contoh algoritma divide and conquer untuk meghitung perkalian matriks

Contoh sederhana dari algoritma divide and conquer dengan bantuan algoritma strassen.

```
Algoritma
Strassen(A,B)
if n =1Output A × B then
  else
  Compute            A11,B11,...,A22,B22
//bycomputing m = n/2
  P1 ← Strassen(A11,B12 – B22)
  P2 ← Strassen(A11 + A12,B22)
  P3 ← Strassen(A21 + A22,B11)
  P4 ← Strassen(A22,B21 – B11)
  P5 ← Strassen(A11 + A22,B11 + B22)
  P6 ← Strassen(A12 – A22,B21 + B22)
  P7 ← Strassen(A11 – A21,B11 + B12)
  C11 ← P5 + P4 – P2 + P6
  C12 ← P1 + P2
  C21 ← P3 + P4
  C22 ← P1 + P5 – P3 – P7
  Output C
endIf
```

Dengan modifikasi mengikuti algoritma strassen, didapatkan kompleksitas baru yang lebih mangkus dari yang lama. Karena pengoperasian perkalian berkuran dari 8 menjadi 7, maka persamaan kompleksitasnya yang baru adalah

$$\boldsymbol{T(n) = 7(T)(n/2) + O}(\mathrm{n}^2)$$
$$\boldsymbol{T(n) = n^{log_2 7} = O(n^{2,8})}$$

Perubahan jumlah rekursif yang tadinya 8 menjadi 7 menyebabkan nilai kompleksitas berubah menjadi $O(n^{2,8})$.

Penurunan nilai dari kompleksitas algoritma tersebut mengacu ke bentuk paling sederhana perkalian matriks, yaitu matriks 2x2. Untuk n x n matriks dengan n bilangan genap, dapat dihitung perkalian matriks dengan metode strassen dengan membagi n x n matriks tersebut menjadi beberapa sub matriks. Misal seperti matriks di bawah ini :

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

$$\boldsymbol{A_{11} = \begin{bmatrix} a_{11} & a_{12} \\ b_{21} & b_{22} \end{bmatrix}, A_{12} = \begin{bmatrix} a_{13} & a_{14} \\ a_{23} & a_{24} \end{bmatrix},}$$

$$\boldsymbol{A_{21} = \begin{bmatrix} a_{31} & a_{32} \\ a_{41} & a_{42} \end{bmatrix}, A_{22} = \begin{bmatrix} a_{33} & a_{34} \\ a_{43} & a_{42} \end{bmatrix}}$$

Matriks seperti di atas dapat dipartisi menjadi 4 submatriks dengan metode divide and conquer. Ketika sudah ditemukan matriks 2 x 2, akan dihitung hasil dari matriks - matriks 2 x 2 tersebut. Jika terdapat operasi perkalian matriks AXB, matriks B juga dibagi menjadi 4 buah sub matriks seperti pada matriks A. Tiap submatriks yang sudah didapatkan kemudian diperlakukan seperti halnya matriks biasa. Berarti aka nada 7 perkalian matriks dan 49 operasi perkalian elemen matriks yang mengikuti variasi strassen.

Sekarang jika terdapat matriks dengan ordo $2^k x 2^k$, maka akan ada 7^kperkalian yang dikerjakan.

III. KELEMAHAN DAN OPTIMALISASI

A. Kelemahan

Dari persamaan – persamaan yang sudah diturunkan seperti di atas, terlihat metode penghitungan perkalian matriks dengan metode strassen dan divide and conquer lebih mangkus dibandingkan cara konvensional (brute force). Namun, meskipun dapat berjalan lebih cepat dibandingkan dengan metode bruteforce, penggunaan rekursifitas dalam penghitungan perkalian matriks akan membutuhkan memory yang sangat besar karena dilakukan dengan rekursi.

B. Optimalisasi

Optimalisasi terhadap pemakaian memory dari algoritma strassen memang sangat penting. Satu metode yang dikenal adalah Strassen–Winograd. Dari hasil pengujian, pemakaian metode penghitungan perkalian matriks dengan algoritma Strassen-Winograd menurunkan penggunaan memory. Kemudian, selain dari masalah memory, penghitungan matriks dengan metode Strassen yang biasa hanya dapat dilakukan pada matriks berordo genap. Jadi tidak dapat dilakukan apabila ordo matriks ganjil.

Inti dari pengurangan kebutuhan memory dan pengoptimalisasian penghitungan dari algoritma Strassen-Winograd adalah dari hal-hal berikut ini :

1. Titik pemotongan Rekursi (Recursion Truncation Point)
2. Penanganan ordo matriks yang acak (metode yang

V. KESIMPULAN

Meskipun penghitungan perkalian matriks dengan menggunakan algoritma divide and conquer dan Strassen manghasilkan kompleksitas waktu yang lebih kecil dibandingkan dengan algoritma brute-force, namun sayangnya algoritma ini hanya dapat diterapkan pada matriks berordo kelipatan 2 (power of 2). Selain itu juga,

karena menggunakan metode rekursi, pengoperasian algoritma ini mahal akan kebutuhan memory. Algoritma ini membutuhkan pemakaian memory yang lebih banyak dibandingkan dengan algoritma brute-force ($O(n^3)$).

Namun sudah ditemukan juga teknik pengoptimalisasian penggunaan algoritma Strassen yang dimodifikasi menjadi Strassen-Winograd. Dengan metode tersebut kedua masalah yang dihadapi tadi bisa ditangani.

Tidak semua algoritma yang secara kompleksitas baik, namun dari segi pemakaian resource itu baik.

REFERENCES

Rinaldi Munir, Slide kuliah Strategi Algoritma.
http://en.wikipedia.org/wiki/Strassen_algorithmH.
http://en.wikipedia.org/wiki/Matrix_multiplication.
http://src.acm.org/loos/loos.html
http://www-math.mit.edu/~djk/18.310/18.310F04/Matrix_%20Multiplication.html
http://www.cs.duke.edu/~alvy/papers/sc98/index.htm#ljjtt:96

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, 29 April 2010

ttd

Otniel - 13508108