ALGORITMA *DIVIDE AND CONQUER* UNTUK MEMBUAT KERJA *CACHE* PADA KOMPUTER PARALEL LEBIH EFISIEN

Ega Dioni Putri (13506095)

Program Studi Teknik Informatika,
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung 40132
e-mail: if16095@students.if.itb.ac.id

ABSTRAK

Algoritma *divide and conquer* mempunyai kelebihan dapat mengurangi kompleksitas pencarian solusi suatu masalah karena prinsip kerjanya yang membagi-bagi masalah menjadi upamasalah-upamasalah yang lebih kecil. Kelebihan tersebut banyak menguntungkan dari segi waktu, tenaga, dan sumber daya. Salah satu penerapan dari algoritma ini adalah pada mekanisme komputer (atau mesin) paralel. Algoritma *divide and conquer* terbukti menampilkan hasil yang paling baik dan paling sesuai untuk komputer dengan hirarki memori tinggi serta memiliki *cache*. *Cache* merupakan memori sementara komputer yang berfungsi untuk mempersingkat pengaksesan data. Masalah yang dihadapi komputer paralel antara lain adalah bagaimana membuat *cache* bekerja lebih efisien pada tiap-tiap komputer yang terhubung secara paralel. Tanpa strategi tertentu, kinerja *cache* pada komputer paralel akan memakan waktu yang lama karena ketika menemui suatu masalah, pemecahan harus dilakukan secara bergantian, satu per satu. Sedangkan komputer paralel menuntut semua perangkatnya mampu bekerja secara paralel. Paralelitas tersebut dapat diatasi dengan penerapan algoritma *divide and conquer* yang memungkinkan pemecahan masalah secara independen, namun tetap konkuren.

Kata kunci: *Divide and Conquer, Cache.*

1. PENDAHULUAN

Teknik *divide and conquer* menjadi dasar algoritma yang efisien untuk banyak jenis persoalan misalnya pengurutan, persoalan-persoalan program dinamis, perkalian matriks, *discrete Fourier transform*, dan lain-lain. Selain itu, teknik ini dapat diimplementasikan oleh algoritma non-rekursif yang menyimpan upamasalah secara parsial pada struktur data eksplisit seperti *stack* (tumpukan), *queue* (antrian), atau *priority queue* (antrian berprioritas).

Penggunaan teknik *divide and conquer* begitu luas meliputi berbagai bidang karena kelebihan-kelebihannya, antara lain dapat bekerja secara rekursif sehingga mempersingkat penjabaran penyelesaian, mempunyai prinsip membagi masalah menjadi upamasalah yang lebih kecil sehingga menyederhanakan pencarian solusi, dan unsur kebebasan yang diberikan dalam pemecahan masalah sehingga pengguna bebas untuk memilih masalah mana yang akan diselesaikan terlebih dahulu dan yang selanjutnya akan diselesaikan.

Untuk mengatasi permasalahan kurangnya efisiensi, komputer atau mesin paralel menggunakan algoritma *divide and conquer* dalam sistem kerjanya. Secara alami, algoritma ini cenderung membuat *cache* lebih efisien. Hal ini disebabkan karena dengan *divide and conquer*, jika sekali saja upamasalah dapat diselesaikan melalui *cache*, tanpa mengakses memori utama yang lebih lambat, maka upamasalah lain yang lebih kecil juga pasti dapat diselesaikan [1]. Prinsip upamasalah dalam *divide and conquer* yang dapat diselesaikan secara independen membuat pemecahan masalah dapat dilakukan secara paralel. Pemecahan masalah di satu *cache* tidak mengganggu proses pemecahan masalah di *cache* lain namun ketika solusi-solusi tersebut digabungkan, masalah utama dalam sistem paralel akan dapat diatasi. Selain itu, algoritma ini membuat performansi hirarki memori lebih baik karena secara rekursif mengurangi ukuran data atau dengan kata lain mengurangi ukuran ruang kerja.

2. ALGORITMA *DIVIDE AND CONQUER*

Agar pembaca lebih memahami penggunaan algoritma *divide and conquer* untuk membuat *cache* bekerja lebih efisien, pada upabab ini akan dibahas mengenai asal-usul, definisi, dan skema umum algoritma *divide and conquer*.

2.1 Sejarah Algoritma *Divide and Conquer*

Algoritma *divide and conquer* ditemukan oleh seorang ilmuwan Rusia bernama Anatolii Alexeevich Karatsuba pada tahun 1960. Pada mulanya, Anatolii

menemukan algoritma yang lebih cepat untuk mengalikan dua buah bilangan bulat yang besar dengan kompleksitas $O(n^{log\ 3})$. Cara yang digunakan Anatolii adalah sebagai berikut :

Misalkan

$$r = (a+b)(c+d) = ac + (ad+bc) + bd$$

maka,

$$(ad+bc) = r - ac - bd = (a+b)(c+d) - ac - bd$$

sehingga perkalian *X.Y* dapat dimanipulasi menjadi

$$X.Y = ac.10^{2s} + (ad+bc).10^{s} + bd$$
$$= ac.10^{2s} + \{(a+b)(c+d) - ac - bd\}.10^{s} + bd$$

bentuk perkalian ini hanya membutuhkan tiga operasi kali saja, yaitu

$$p = ac, r = (a+b)(c+d), q = bd$$

Dibandingkan dengan perkalian serupa yang ditemukan oleh Andrey Kolmogorov pada tahun 1956 yang memiliki kompleksitas $O(n^2)$ [4], cara Karatsuba di atas jelas lebih mangkus.

Jika kita perhatikan, Anatolii menggunakan langkah membagi dan memecah masalah menjadi masalah yang lebih kecil. Bilangan *X* yang memiliki *n* digit dibagi menjadi *a* (hasil dari $X/10^s$) dan *b* (hasil dari $X \bmod 10^s$). Begitu pula bilangan *Y*, dibagi menjadi *c* (hasil dari $Y/10^s$) dan *d* (hasil dari $Y \bmod 10^s$). Selanjutnya, keempat elemen tersebut dijumlahkan dan dikalikan sampai membentuk solusi yang tepat.

2.2 Definisi Algoritma *Divide and Conquer*

Dilihat dari sisi harfiah, istilah *divide and conquer* sudah menunjukkan cara kerja dari algoritma ini sendiri. *Divide* berarti membagi, sedangkan *conquer* berarti memecahkan atau mengalahkan (dalam pertempuran). Definisi masing-masing kata tersebut merupakan tahapan pemecahan masalah yang dilakukan dalam *divide and conquer*.

Divide and conquer adalah metode pemecahan masalah dengan membagi-bagi masalah menjadi upamasalah-upamasalah yang lebih kecil kemudian menyelesaikan tiap upamasalah tersebut secara independen untuk selanjutnya disusun menjadi penyelesaian masalah semula [2]. Metode ini bekerja secara rekursif karena pembagian masalah dan pemecahan dari tiap upamasalah dapat dilakukan berulang kali, tergantung seberapa banyak kita membagi masalah ke dalam masalah yang lebih kecil. Upamasalah memiliki karakteristik yang sama dengan masalah asal.

Pada dasarnya, algoritma ini memiliki tiga tahap pencarian solusi sebagai berikut:

1. Proses *divide* (membagi) : membagi masalah menjadi beberapa upamasalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama antara upamasalah satu dengan lainnya)
2. Proses c*onquer* (memecah) : memecahkan (menyelesaikan) masing-masing upamasalah (secara rekursif)
3. Proses *combine* (menggabung) *:* menggabungkan solusi masing-masing upamasalah sehingga membentuk solusi masalah semula

2.3 Skema Umum Algoritma *Divide and Conquer*

Skema umum algoritma *divide and conquer* dijelaskan dalam algoritma *pseudo-code* berikut ini:

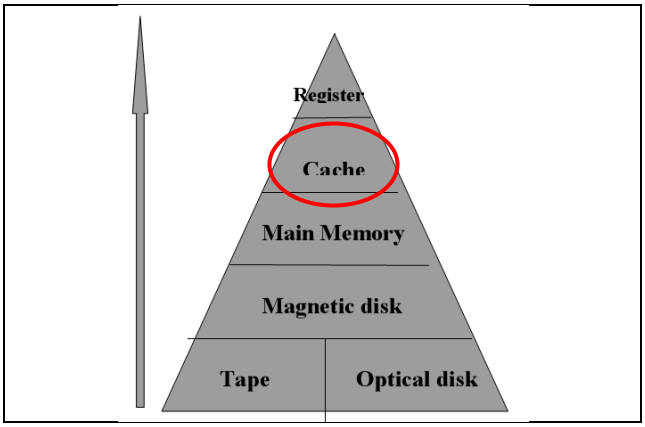
```
procedure DIVIDE_and_CONQUER(input n : integer)

{ Menyelesaikan masalah dengan algoritma divide and conquer
  Masukan: masukan yang berukuran n
  Keluaran: solusi dari masalah semula
}

Deklarasi
     r, k : integer

Algoritma
  if n ≤ n0 then
  {ukuran masalah sudah cukup kecil }
     SOLVE upamasalah yang berukuran n ini
  else
     bagi menjadi r upamasalah, masing-masing berukuran n/k
     for masing-masing dari r upamasalah
        do
        DIVIDE_and_CONQUER(n/k)
        {bagian rekursif}
     endfor
     {gabung solusi dari r upamasalah menjadi solusi masalah semula}
  endif
```

3. *CACHE* DAN CARA KERJANYA

3.1 Pengertian *Cache*

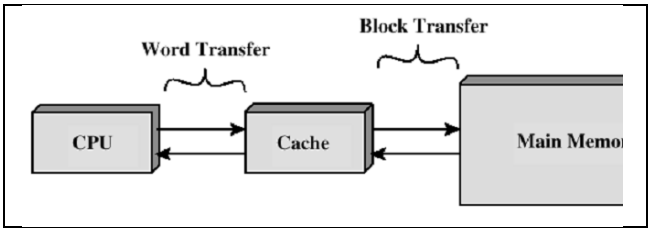
Cache (baca: kaesh) merupakan sebuah tempat penyimpanan data sementara yang dapat menyimpan pengaksesan data secara berkala sehingga mempercepat pengaksesan ketika suatu saat data tersebut dibutuhkan lagi. Dalam arsitektur komputer, *cache* termasuk memori yang berada pada hirarki memori bagian atas tepat di

bawah *register*. *Cache* berkapasitas kecil namun kekuatannya mendekati prosesor [3].



Gambar 1. Posisi cache pada hirarki memori komputer

Cache menjadi perantara *CPU (Central Unit Process)* dan memori utama pada komputer seperti yang ditunjukkan pada gambar berikut:



Gambar 2. Posisi cache di antara CPU dan memori uatama

Cache memiliki berbagai macam pilihan jenis dan ukuran. Jenis *cache* dibedakan berdasarkan *level*-nya seperti L1 *(level 1),* L2 *(level 2),* L3 *(level 3),* dan sebagainya. Sedangkan ukuran *cache* dinyatakan dalam MB *(megabyte)* misalnya 6 MB, 24 MB, 32 MB, dan seterusnya.

3.2 Sistem Kerja *Cache*

Prinsip kerja *cache* adalah menyalin data dari nilai-nilai aslinya kemudian menyimpannya di tempat lain. Data tersebut nantinya akan dibaca dari *cache* jika ada pengaksesan lagi, tidak perlu dicari dari data asli pada memori utama. Dengan begitu, ongkos pengaksesan akan lebih "murah" dan waktu akses dapat dipersingkat.

4. PERAN ALGORITMA *DIVIDE AND CONQUER* UNTUK MEMBUAT KERJA *CACHE* LEBIH EFISIEN

Algoritma *divide and conquer* seakan menantang problematika dalam mesin atau komputer paralel yang cenderung memiliki sejumlah besar paralelisme turun-temurun dan bekerja dengan *cache* dan hirarki memori yang tinggi. Hal ini karena dalam implementasi program, algoritma *divide and conquer* dikode sebagai prosedur rekursif (mengandung dirinya sendiri) dan menggunakan *pointer* untuk aritmatika dan pengalokasian memori secara dinamis. Fitur itulah yang tidak kompatibel jika diterapkan pada algoritma selain *divide and conquer* [1].

4.1 Memecahkan masalah secara paralel dan konkuren

Dengan *divide and conquer*, paralelisme turun-temurun tidak menjadi masalah karena ketika pembagian masalah sudah dilakukan, upamasalah-upamasalah akan bersifat independen sehingga dapat dipecahkan secara paralel. Bahkan, tahap pembagian dan pemecahan masalah dapat dijalankan secara konkuren sehingga mempertahankan mesin tetap sibuk. Misalnya, ketika sebuah upamasalah sedang diselesaikan oleh komputer A, dapat saja pada komputer B yang terhubung secara paralel dengan A sedang dilakukan proses pembagian masalah baru. Begitu seterusnya sehingga tidak ada proses penungguan antara proses penyelesaian upamasalah satu dengan upamasalah lainnya.

4.2 Bekerja rekursif untuk setiap upamasalah

Algoritma *divide and conquer* dapat menghasilkan performansi yang bagus dalam komputer paralel. Setiap kali suatu upamasalah telah terselesaikan dalam suatu *cache*, maka sebuah solusi rekursif standar akan disimpan dan digunakan lagi pada data *cache* yang lain sampai semua upamasalah diselesaikan secara lengkap. Jika kita menilik algoritma lain, maka akan terlihat proses rekursif membutuhkan waktu eksekusi yang besar karena masalah tidak dibagi-bagi terlebih dahulu menjadi masalah yang lebih kecil sehingga kita harus memanggil rekurens 'lebih jauh' dari basisnya. Bisa dibayangkan misalnya saja struktur data yang digunakan adalah pohon, maka pohon rekursif dalam algoritma selain *divide and conquer* akan memiliki aras *(level)* yang dalam daripada algoritma *divide and conquer*.

4.3 Beradaptasi dengan mudah pada *cache, level* hirarki memori, dan mesin apa saja

Algoritma *divide and conquer* secara alami dapat bekerja dengan baik pada *cache* dengan ukuran berapa pun. Selain itu, algoritma ini juga dapat beradaptasi pada semua level hirarki memori. Sama seperti kasus pemecahan masalah untuk setiap *cache*, pada hirarki memori, ketika suatu upamasalah satu level tertentu berhasil diselesaikan, maka upamasalah pada level

lainnya, baik level di atas maupun di bawahnya, akan diselesaikan dengan solusi standar yang sama sampai semua masalah terpecahkan. Algoritma *divide and conquer* beradaptasi dan berjalan dengan baik pada semua hirarki *cache* atau memori tanpa modifikasi untuk mesin apa pun yang digunakan.

5. KESIMPULAN

Divide and conquer adalah algoritma terbaik untuk menyelesaikan berbagai permasalahan pada sistem komputer paralel yang membutuhkan kinerja secara paralel. *Divide and conquer* dapat bekerja secara paralel karena membagi-bagi masalah menjadi upamasalah yang dapat diselesaikan sendiri-sendiri (independen) tanpa menganggu satu sama lainnya dan berjalan bersamaan (konkuren). Hal ini menyebabkan kerja *cache* lebih efisien karena mengurangi waktu eksekusi data dan pemecahan masalah yang seringkali terjadi pada *cache*.

REFERENSI

[1] Ragina,Radu dan Martin Rinard, "Automatic Parallelization of Divide and Conquer Algorithms", Laboratory of Computer Science Massachusetts Institute of Technology, 1999.

[2] Rinaldi Munir, "Diktat Kuliah IF2251 Strategi Algoritmik", Program Studi Informatika ITB, 2006.

[3] Partasubita, Santika Wachyudin. "Organisasi dan Arsitektur Komputer 1", Departemen Teknik Informatika, Institut Teknologi Bandung, 2007.

[4] http://wikipedia.org, akses: 19 Mei 2008, 10.45

[5] http://mathworld.wolfram.com, akses: 20 Mei 2008, 7.48