# Multithreading untuk Algoritma Divide and Conquer

Novan Parmonangan Simanjuntak(13509034)
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
impulse.impulse0@gmail.com; mathprince@ymail.com

***Abstract*—Biasanya algoritma divide and conquer dilakukan dieksekusi dengan satu buah prosesor saja. Setiap subrutin dieksekusi satu per satu, sehingga kompleksitas waktu dihitung berdasarkan jumlah waktu setiap subrutin. Pada makalah ini akan dibahas penggunaan multithread untuk algoritma divide and conquer. Dengan penggunaan multithread, maka setiap subrutin dieksekusi secara konkurensi(bersamaan) sehingga kompleksitas waktu totalnya bisa jauh berkurang. Pada makalah ini akan dibahas cara pemodelan untuk multithread untuk algoritma divide and conquer. Selain itu juga akan dibahas kelemahan dalam penggunaan multithread ini serta parameter yang harus dipertimbangkan dalam menentukan kompleksitas baik ruang maupun waktu dari algoritma divide and conquer. Pembaca makalah diharapkan pernah memahami hal-hal mengenai konkurensi dan dasar-dasar dari sistem operasi dalam alokasi dan eksekusi program. Pada makalah ini juga akan dijelaskan sedikit mengenai alokasi resource dan multithread itu sendiri.**

***Kata kunci – Divide and Concquer, multithread, sistem operasi, kompleksitas, performansi***

## I. PENDAHULUAN

Banyak algoritma yang diterapkan merupakan algoritma serial. Salah satunya adalah algoritma Divide and Concuqer. Algoritma serial artinya algoritma tersebut dijalankan secara serial pada komputer uniprocesor(atau oleh satu prosesor), di mana pada algoritma serial hanya satu instruksi dieksekusi pada suatu waktu. Di zaman yang modern ini, komputer kebanyakan sudah multiprosesor(mulai dari dualcore, core to duo, core i3, core i5 dan corei7). Dengan adanya lebih dari satu prosesor, kita bisa memaksimumkan semua prosesor ini untuk memperbaiki performansi dari algoritma yang ada. Oleh karena itulah penulis tertarik untuk membahas mengenai penggunaan multithread terutama untuk algoritma divide and conquer.

Algoritma yang akan dipilih di sini adalah algoritma divide and conquer. Karena kebanyakan masalah divide and conquer secara natural paling mudah diselesaikan dan dimodelkan dengan multithread programming. Hal ini karena algoritma divide and conquer sendiri membagi masalah menjadi subrutin yang bisa dikerjakan secara konkuren atau bersamaan. Meskipun demikian akan diberikan satu contoh yang bukan merupakan algoritma divide and conquer untuk memudahkan pemodelan multithread untuk algoritma divide and conquer.

Untuk contoh pemecahan masalah akan dibahas masalah fibonacci sebagai dasar dari pemodelan multithread serta algoritma quicksort untuk permasalahan divide and conquer.Akan diperlihatkan tes untuk mengetahui kelebihan dan kekurangan dalam menggunakan multithread untuk algoritma divide and conquer.

## II. ALGORITMA MULTITHREAD

### A. Definisi

Multithread artinya kemampuan CPU untuk mngeksekusi satu atau lebih instruksi(penjumlahan, perkalian dan lainnya) pada waktu yang bersamaan. Thread itu sendiri merupakan bagian terkecil yang bisa dieksekusi oleh sistem operasi dan ini bisa dijalankan secara konkurensi atau bersamaan dengan thread lain yang sedang berjalan. Hal ini dimungkinkan karena adanya proses multiprogramming(pergantian eksekusi) oleh sistem operasi. Sistem operasi sekarang(berbasis linux dan NT atau windows sudah bisa melakukan multiprogramming). Dengan adanya jumlah prosesor lebih dari satu, multithread bisa dilakukan dengan lebih efektif lagi(bisa saja satu thread memakai satu core untuk melakukan eksekusinya). Seperti yang sudah dijelaskan pada pendahuluan, komputer multiprosesor sudah banyak beredar di kalangan informatikawan, pengembang perangkat lunak bahkan di kalangan masyarakat sendiri(mulai dari dua core sampai tujuh core). Meskipun demikian, sampai sekarang tidak ada satupun model yang diterima secara luas. Alasan utamanya adalah karena *vendor* tidak menyetujui pada sebuah model untuk komputer pararel. Berikut dijelaskan beberapa paradigman pemodelan multithread :

1. Shared memory, artinya masing-masing prosesor dapat secara langsung mengakses lokasi memory thread manapun.
2, Distributed memory, di mana di pemodelan ini setiap prosesor memiliki memori sendiri yang bersifat privat(tidak bisa diakses oleh prosesor lain). Agar antarprosesor bisa saling mengakses memori, harus ada pesan yang secara eksplisit disampaikan antarprosesor.

Dengan adanya teknologi multicore di setiap laptop ataupun komputer. Pemodelan shared memory lebih banyak digunakan. Perlu diperhatikan bahwa digunakan static thread untuk pemodelan ini, ini dikarenakan cost yang digunakan setiap kali menciptakan dan menghancurkan thread mahal, sehingga thread tersebut harus hidup selama program berjalan, sehingga disebut

static thread. Meskipun dengan adanya shared memory yang bisa mengimplementasikan static thread, melakukan implementasi static thread sangat sulit dan kemungkinan bug program besar. Oleh karena itulah diciptakan dan digunakan platform khusus untuk pemrograman multithread, ini semacam interface dalam menggunakan thread sehingga tidak perlu menggunakan komunikasi protokol yang kompleks serta memikirkan hal-hal lain seperti pembagian kerja antarthread. Hal ini secara otomatis dilakukan oleh platform. Platform yang konkurensi yang akan digunakan untuk pemodelan multithread ini adalah multithreading dinamis.

Multithreading dinamis adalah model yang akan digunakan disini. Multithreading dinamis merupakan model di mana programmer tidak harus khawatir mengenai masalah komunikasi protokol, pmebagian kerja thread dan hal-hal lainnya. Terdapat dua fitur dari multithreading dinamis, antara lain :

1. *Nested parallelism*, di mana setiap subrutin dieksekusi dengan thread baru, jadi fungsi yang memanggil bisa mengerjakan hal lain selama subrutin ini sedang dikerjakan.
2. *Parallel loop*, di mana pada setiap loop iterasi dari loop bisa dilakukan secara konkuren atau bersamaan.

Pemodelan multuthreading dinamis mempunyai dua *keyword* atau istilah yang digunakan untuk mendeskripsikan multithreading dinamis, yaitu **spawn,** dan **sync. Spawn** digunakan untuk menyatakan dipakainya thread baru untuk mengeksekusi instuksi, sedangkan **sync** berfungsi untuk sinkronisasi antarthread(contohnya seperti thread saling menuggu agar thread lain selesai karena data dari thread tersebut dibutuhkan untuk proses selanjutnya). Jika *keyword* ini dihapus, maka algoritma akan menjadi serial dan berjalan seperti biasa.

B. Contoh

Contoh 1 : Algoritma rekursif perhitungan fibonacci.

Untuk contoh dasar akan digunakan problem fibonacci, adapun algoritma yang digunakan adalah rekursif, contoh ini ditujukan agar pembaca lebih memahami hal mengenai pemodelan dinamis multithreading.

Algoritma rekursif serial(tanpa multithreading) dijelaskan dengan pseudo-code berikut ini :

```
Program fibonacciserial(n)
if (n≤1)
   return n
else
   x = fibonacciserial(n-1)
   y = fibonacciserial(n-2)
   return x+y
```

Perhatikan pada pseudo-code di atas, hanya satu prosesor saja digunakan, nilai y dihitung jika nilai x sudah dihitung terlebih dahulu, ini dikarenakan algoritma tersebut hanya menggunakana satu core prosesor sehingga eksekusi haruslah sekuensial.

Pemodelan dinamis multithreading untuk permasalah fibonacci adalah sebagai berikut :

```
Program fibonaccithreading(n)
if (n≤1)
   return n
else
   x = spawn fibonaccithreading(n-1)
   y = fibonaccithreading(n-2)
   sync
   return x+y
```

Perhatikan bahwa pseudo-code diatas berbeda dengan program serial, yaitu :

1. *Keyword* **spawn** menandakan bahwa saat pencarian x digunakan sebuah thread static, sehingga nilai y langsung ikut dihitung, perhatikan bahwa terdapat dua thread, satu thread untuk menghitung nilai x dan thread lain yang memanggil(bisa disebut thread utama yang merupakan thread awal saat program dijalankan). Perhitungan nilai x dan y dilakukan secara bersamaan, hal ini dimungkinkan karena perhitungan nilai x dan y tidak berhubungan atau saling lepas, jadi thread untuk menghitung nilai x tidak berkaitan dengan thread untuk menghitung nilai y.
2. *Keyword* **sync** menandakan bahwa kedua thread harus saling sinkronisasi atau telah selesai keduanya. Hal ini dikarenakan nilai x dan y dibutuhkan untuk menghitung nilai x+y. Oleh karena itu, nilai x dan y harus diketahui keduanya.

C. Penggunaan

Dinamis multithread sangat banyak diaplikasikan terutama untuk sistem yang menginginkan performansi yang bagus dengan mengerahkan seluruh resource yang ada, contoh penggunaannya antara lain adalah di bidang :

1. Basis data, di mana tranksaksi menyatakan thread dan data menyatakan resource. Agar performansi bagus, maka harus ada mekanisme pararel. Di sini salah satunya digunakan pemodelan multithread.
2. Program-program kompleks seperti Word editor menggunakan mekanisme multithread. Thread baru tersebut bisa berupa *background* thread atau thread yang berjalan dibelakang menjalankan tugas semestinya. Contohnya pada Word editor adalah adanya thread khusus untuk selalu mengupdate kata, sedangkan ada thread lain untuk menerima input dari user.
3. Pada web-server, terdapat mekanisme thread disebut *threadworker*.*Threadworker* bekerja sebagai thread yang melayani satu permintaan, sedangkan terdapat thread utama yang mengatur permintaan mana dijalankan oleh thread mana.

## III. ALGORITMA MULTITHREAD DIVIDE AND CONQUER

### *A. Definisi*

Skema umum dari algoritma divide and conquer bisa ditulis sebagai berikut :

1. Misalkan ukuran masalah sebesar n, maka jika jika ukuran masalah n ≤ n0 , dimana n0 menyatakan batasan masalah yang cukup kecil yang bisa diselesaikan, maka masalah tersebut langsung diselesaikan.
2. Jika n > n0 , maka bagi n menjadi r subrutin, masing-masing berukuran n/k. Untuk setiap subrutin ini panggil lagi prosedur divide and conquer(ulangi dari langkah satu).
3. Dari r subrutin yang sudah diselesaikan, solusi dari masing-masing subrutin dikombinasikan menjadi solusi dari masalah semula.

B. *Aplikasi dinamis multithread untuk algoritma divide and conquer*

Perhatikan pada *psudo-code* algoritma divide and conquer di atas, dengan mengapliklasikan dinamis multithread skemanya bisa diubah menjadi :

1. Misalkan ukuran masalah sebesar n, maka jika jika ukuran masalah n ≤ n0 , dimana n0 menyatakan batasan masalah yang cukup kecil yang bisa diselesaikan, maka masalah tersebut langsung diselesaikan.
2. Jika n > n0 , maka bagi n menjadi r subrutin, masing-masing berukuran n/k. Untuk setiap subrutin panggil **spawn** (jika masih ada core prosesor idle),untuk menggunakan thread static untuk memanggil kembali prosedur divide and conquer. Dengan demikian semua subrutin bisa berjalan bersamaan. Untuk minimasi core,subrutin terakhir bisa menggunakan thread pemanggil.
3. Lakukan **sync** atau sinkronisasi dari r subrutin yang sudah diselesaikan, ini karena solusi dari masing-masing subrutin akan dikombinasikan menjadi solusi dari masalah semula.

## IV. METODE LOGIKA

Di sini akan dijelaskan metode perkiraan dari kompleksitas algoritma divide and conquer dengan dan tanpa dinamis multithread.

Tanpa dinamis multithread kompleksitas waktu algoritama divide and conquer bisa dinyatakan sebagai berikut :

1. $T(n) = O(1),\ n \le n0.$
2. $T(n) = r * T(n/k) + C(n)$

di mana C(n) merupakan komplesitas waktu untuk partisi dan kombinasi solusi.

Sedangkan untuk komplesitas ruang dari algoritma divide and conquer dengan dinamis multithread(asumsikan jumlah core prosesor mencukupi semua thread static yang dibutuhkan) :

1. $T(n) = O(1),\ n \le n0.$
2. $T(n) = \max( T_i(n/k) ) + C(n)$

dengan $1 \le i \le r$ dan C(n) menyatakan kompleksitas waktu untuk partisi dan kombinasi solusi.

Perhatikan bahwa $T_i(n/k)$ menyatakan kompleksitas dari subrutin ke-i, bisa dilihat pada point 2 bahwa ini tergantung dari subrutin dengan kompleksitas waktu terbesar. Agar minimum, maka setiap subrutin harus dibagi secara rata, sehingga tidak ada subrutin yang cepat selesai dibandingkan subrutin lain. Hal ini bisa dilakukan dengan membagi rata ukuran masalah untuk setiap subrutin. Untuk mempermudah kita asumsikan waktu untuk setiap subrutin sama, sehingga bisa ditulis :

$T(n) = T(n/k) + C(n).$

Perhatikan bahwa komplesitas waktu algoritma divide and conquer berkurang jauh. Untuk lebih memperjelas, akan dijelaskan dengan contoh yaitu algoritma quicksort.

Contoh 2 : Algoritma quicksort untuk masalah pengurutan bilangan.

Akan dihitung kompleksitas waktu dari algoritma quicksort(asumsi bahwa pivot yang terpilih selalu membagi total elemen menjadi dua bagian sama besar).

Kompleksitas waktu algoritma quicksort tanpa multithread :

1. $T(n) = O(1),\ n \le 1.$
2. $T(n) = 2T(n/2) + O(n).$

Dengan master method didapat
$T(n) = O(n \lg n).$

Sedangkan kompleksitas waktu algoritma quicksort dengan multithread adalah :

1. $T(n) = O(1),\ n \le 1.$
2. $T(n) = T(n/2) + O(n).$

Dengan master method didapat
$T(n) = O(n)$

Perhatikan bahwa untuk sorting kompleksitas yang dibutuhkan menjadi liner(lebih baik dari sebelumnya).

## V. IMPLEMENTASI

Sebelumnya telah dijelaskan mengenai contoh dinamis multithread yaitu algoritma rekursif fibonacci dan algoritma divide and conquer. Meskipun kompleksitas ruang dari dinamis multihtread lebih kecil, tetapi hal tersebut belum menjamin waktu nyata eksekusi program

lebih cepat ini dikarenakan beberapa hal, salah satunya adalah mahalnya cost untuk alokasi thread itu sendiri dan saat komunikasi serta protokol lainnya. Berikut akan diberikan contoh implementasi untuk analisis dari penerapa dinamis multithread :

Pseudo-code untuk algoritma quicksort dengan dinamis multithread :

```
Program quicksortthread(A[p..r])
if (p<r)
   q =  Partition(A,p,r)
   spawn quicksortthread(A[p..q-1])
   quicksortthread(A[q+1..r])
   sync
```

Untuk algoritma Partition sama seperti algoritma Partition pada quicksort tanpa dinamis multithread. Berikut diberikan algoritma untuk partisi :

```
Function Partition(A[p..r])
x = A[r]
I = p-1
for j=p to r-1
   do if (A[j] ≤ x)
      then i = i + 1
           swap( A[i], A[j] )
swap(A[i+1 , A[r])
return i+1
```

Untuk perbandingan dan analisis performansi dinamis multithread diberikan dua buah pseudo-code untuk fibonacci(psudo-code pertama ditulis dalam bahasa C, sedangkan pseudo-code kedua ditulis dalam bahasa Java) :

*Pseudo-code* pengujian dinamis thread algoritma rekursif fibonacci dengan menggunakan dua thread dengan memanggil fungsi fibonacci serial pada Bagian II . *Pseudo-code* di bawah ini ditulis dalam bahasa C :

```
Program testing2threadfib(int n)
search  = n

//Kasus untuk menghitung waktu tanpa thread
start = time();
x = spawn fibonacciserial(n-1)
y = fibonacciserial(n-2)
end = time();
output(end-start)

//Kasus untuk menghitung waktu dengan thread
z = fibonacciserial(n)
output(end-start)
```

Untuk *Pseudo-code* pengujian dinamis thread algoritma rekursif fibonacci dengan menggunakan lima thread dengan memanggil fungsi fibonacci serial pada Bagian II. *Pseudo-code* di bawah ini ditulis dalam bahasa Java :

```
Program testing5threadfib(int n)
//Kasus untuk menghitung waktu tanpa thread
start = time();

for i = 1 to 5 do
  val = fibonacciserial(45);

end = time();
output(end-start)


//Kasus untuk menghitung waktu dengan thread
start = time();
for i = 1 to 5 do
     val = spawn fibonacciserial(45);

for all thread running except mainthread
  sync
end = time();
output(end-start)
```

## V. PENGUJIAN DAN ANALISIS

Pengujian dilakukan di komputer penulis, di mana mesin komputernya mempunyai tujuh core prosesor. Implementasi code ditulis dalam bahasa Java dan C. Berikut lingkungan pengujian dari implementasi dinamis multithread pada algoritma divide and conquer

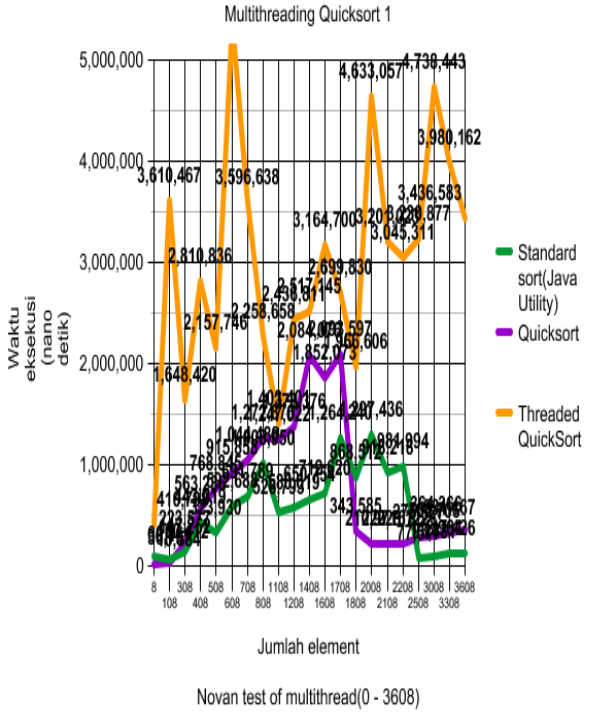
| Parameter | Nilai |
|---|---|
| Sistem operasi | Linux 32bit 11.10 OneiricOcelot |
| JDK Version | 1.6 |
| CPU | Core i7 1.79GHz |
| Memori | 4GB, 7Mb 3 level cache |

Tabel 1. Lingkungan pengujian dinamis multithread

Berikut dilakukan dua jenis percobaan :

1. Untuk percobaan quicksort dengan dinamis thread, dilakukan perbandingan dengan algoritma sort dari java library. Berikut hasil percobaan yang dilakukan

Percobaan 1 : quicksortthread dengan n kecil




Gambar 1. Pengujian quicksort dengan ukuran n kecil

Percobaan 2 : quicksort dengan n besar

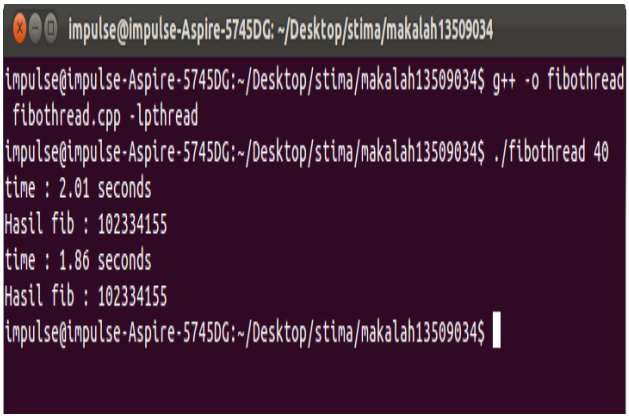
Gambar 2. Pengujian quicksort dengan ukuran n besar

Analisis percobaan 1 dan 2 :

Bisa diperhatikan dari kedua pengujian di atas didapat bahwa dinamis multithread lebih efektif saat nilai n besar.Ini dimungkinkan karena waktu cost yang dihabiskan oleh thread masih lebih besar dibandingkan dengan waktu lainnya, sehingga algoritma tanpa dinamis multithread lebih cepat.

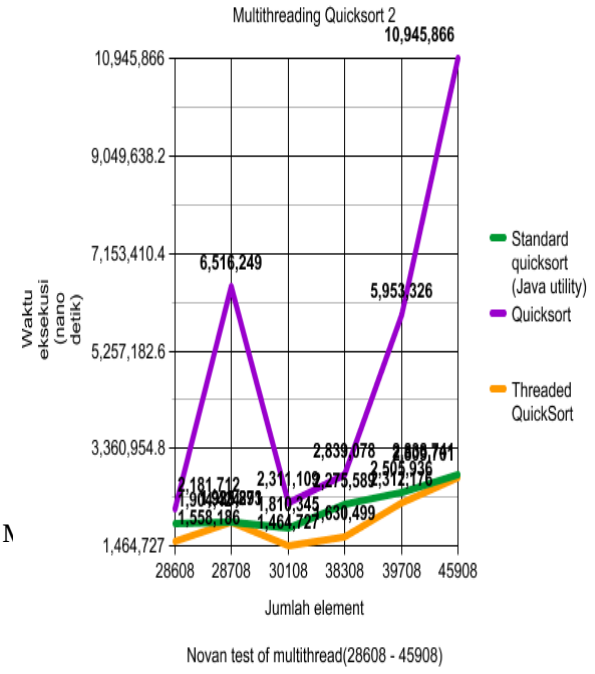
2. Untuk percobaan dinamis multithread dengan menggunakan algoritma rekursif fibonacci dilakukan perbandingan dengan menggunakan dua thread dan lima thread(sesuai dengan implementasi)

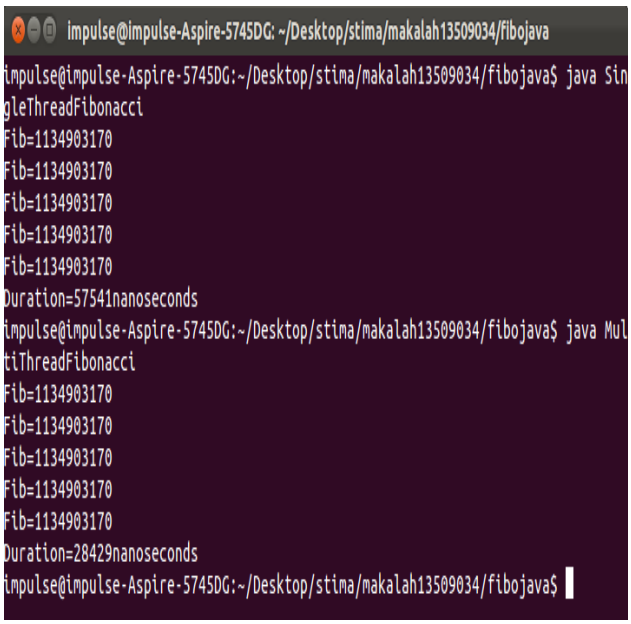
Percobaan 3 : fibonacci rekursif dengan dua thread




Gambar 3. Percobaan dinamis multithread untuk algoritma rekursif fibonacci dengan dua thread

Percobaan 4 : fibonacci rekursif dengan dua thread






Gambar 4. Percobaan dinamis multithread untuk algoritma rekursif fibonacci dengan lima thread

Analisis percobaan 3 dan 4 :

Bisa diperhatikan dari percobaa 3 dan 4, dimana jumlah thread yang banyak akan semakin bagus dalam menyelesaikan suatu permasalahan. Hal ini wajar, karena dengan rekursif saat fibonacci memerlukan waktu yang lama, sehingga semakin banyak thread maka performansi dari algoritma bisa semakin baik(asumsikan core prosesor mencukupi).

## VII. KESIMPULAN

Dari pembahasan bab-bab sebelumnya penulis membuat beberapa kesimpulan sebagai berikut :

1. Jumlah core prosesor mempengaruhi kecepatan program. Untuk uniprosesor(hanya satu core prosesor saja), implementasi multithread pada malah akan memperburuk performansi algoritma.
2. Penciptaan thread dan pemanggilannya mahal, sehingga perlu dipertimbangkan mengenai ukuran data dan banyak thread yang diperlukan sehingga bisa mencapai performansi yang optimal
3. Penggunaan dinamis multithread memang bagus, tetapi setiap thread mempunyai *resource*-nya sendiri, sehingga akan memakan ruang lebih banyak daripada tanpa dinamis multithread(kompleksitas ruang bertambah).
4. Implementasi thread bisa lebih bagus untuk ukuran n yang lebih besar.
5. Algoritma divide and conquer secara natural(dengan mudah) bisa dibuat model dinamis multithreadnya.
6. Algoritma divide and conquer secara tidak langsung telah mengurangi space function(stack untuk function dan lainnya), karena untuk setiap subrutin masalah, fungsi yang dipanggil sama.

## REFERENCES

[1] Cormen. H. THOMAS, “Introduction to Algorithm” , 3ed, Ed. New York: McGraw-Hill, 1964, pp. 772–780.
[2] http://sahits.ch/blog/?p=601#third
[3] http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=6908&lngWId=2
[4] http://lycog.com/concurency/performance-single-multi-thread-java/

## 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, 8 Desember 2011

Novan Parmonangan Simanjuntak
13509034