PENYELESAIAN PROBLEM GAUSSIAN ELIMINATION MENGGUNAKAN POSIX THREAD, OPENMP DAN INTEL TBB

Ari Wibowo
Jurusan Teknik Informatika, Politeknik Negeri Batam
Jl. Parkway, Batam Center, 29461
wibowo@polibatam.ac.id

Abstrak

Peningkatan kinerja komputer itu sendiri tidak hanya memanfaatkan prosessor saja namun bisa menggunakan GPU yang saat ini kecepatannya bisa melebihi prosessor sekalipun. Peningkatan ini sebenarnya juga dimaksudkan agar dalam satu kali intruksi, komputer bisa langsung menangani berbagai proses sekaligus tanpa terjadi *deadlock.* Cara yang paling lazim digunakan adalah dengan menggunakan *thread.* Penggunaan *thread* mampu meningkatkan performansi sampai ribuan kali dibandingkan perhitungan dengan serial.

Kata Kunci : *thread, possix, open mp, intel tbb*

Abstract

Improved performance of the computer itself is not only utilize the processor alone, but could use the current GPU speed can exceed the processor though. This increase was also intended that in one instruction, the computer can directly handle the various processes at once without deadlock. The most commonly used way is to use a thread. The use of threads can improve performance up to thousands of times compared to the serial computation.

Keyword: *thread, possix, open mp, intel tbb*

1. PENDAHULUAN

1.1 Latar Belakang

Zaman modern saat ini banyak dikembangkan metode untuk meningkatkan kinerja proses dari suatu komputer. Apalagi dengan adanya *multiprocessor* yang sudah sampai delapan *core* mengakibatkan kinerjanya yang cepat sampai hitungan nanosecon. Peningkatan kinerja komputer itu sendiri tidak hanya memanfaatkan prosessor saja namun bisa menggunakan GPU yang saat ini kecepatannya bisa melebihi prosessor sekalipun. Peningkatan ini sebenarnya juga dimaksudkan agar dalam satu kali intruksi, komputer bisa langsung menangani berbagai proses sekaligus tanpa terjadi *deadlock* ataupun *hazard* lain yang menghambat kecepatan komputer.

Cara yang paling lazim digunakan adalah dengan menggunakan *Thread.* Untuk pemanfaatan *thread* ini sudah diaplikasikan sekarang dengan komputer yang memiliki lebih dari satu prosessor. Ada juga komputer yang prossesornya satu namun mendukun *hyperthreading.* Adanya thread memang terbukti komputer mampu melakukan lebih dari satu proses sekaligus. *Thread* juga bisa dikerjakan secara konkuren lebih satu (*multiple threads*) untuk satu

proses yang menggunakan *resources* yang sama (menggunakan memory). Namun untuk *resources* yang berbeda, hal tersebut tidak bisa dilakukan.

Perusahaan Intel yang terkenal dengan produk Core Duo dan sekarang dengan produk barunya i5/i7, juga telah mengembangkan teknologi penanganan tersebut dengan adanya Intel Threading Building Block (Intel TBB). Produk ini tidak lagi menggunakan *thread* namun menggunakan *task*. Implementasi Intel TBB ini menggunakan bahasa C++ dengan memanfaatkan *library* bawaan dari Intel. Selain itu, NVIDIA sebagai pengembang GPU juga telah menghasilkan CUDA yang digunakan untuk menangani hal tersebut juga. Namun CUDA ini baru bisa diimplementasikan untuk produk GPU baru didukung oleh GeForce 8100 ke atas saja sehingga untuk mengukur peningkatan performansinya masih butuh biaya yang relatif mahal. Hebatnya untuk CUDA ini prosessor yang dimiliki bisa mencapai 30. Penelitian membuktikan untuk pengukuran suatu permasalahan (misalnya perhitungan matriks), adanya penggunaan CUDA mampu meningkatkan performansi sampai 3000 kali dibandingkan perhitungan dengan serial.

1.2 Perumusan Masalah

Bagaimana membuktikan pengukuran suatu permasalahan (misalnya perhitungan matriks), adanya penggunaan CUDA mampu meningkatkan performansi sampai 3000 kali dibandingkan perhitungan dengan serial.

1.3 Tujuan Penelitian

Adapun tujuan dilakukan penelitian ini adalah:
1.Mengetahui dan memahami pemanfaatn pemrograman pararel.
2.Membandingkan kinerja dari masing-masing teknik Possix, Open MP, dan Intel TBB

2. TINJAUAN PUSTAKA

2.1 POSIX Thread

Pthreads mendefinisikan himpunan bahasa pemrograman C baik dari tipe, fungsi, dan konstanta. Semuanya diimplementasikan dengan *file header* pthread.h dan *library thread*. Programmer juga bisa menggunakan *Pthread* untuk membuat, memanipulasi, dan mengatur thread, sama seperti sinkronisasi antar thread menggunakan *mutex*, *condition variable*, dan *semaphore*.

2.2 OpenMP

OpenMP adalah implementasi dari *multithreading*, sebuah metode untuk paralelisasi dimana ada master "thread" (rangkaian intruksi yang dijalankan bersamaan) "forks" sebuah jumlah dari *slave* "threads" dan *task* yang dibagi diantara mereka. Kemudian thread bisa jalan secara konkuren, dengan *runtime environment* mengalokasikan *thread* di prosesor yang berbeda. Tiap thread memiliki *id* yang bisa didapat dengan memanggil fungsi (dinamakan *omp_get_thread_num()* di C/C++ dan *OMP_GET_THREAD_NUM()* di Fotran). Id tersebut berbentuk integer dan master thread mempunyai id "0". Setelah eksekusi dari kode yang paralel, thread barusan di-"join" kembali oleh master thread, dan dilanjutkan kembali sampai akhir program.

Secara *default*, tiap thread mengeksekusi bagian yang paralel tersebut secara independen. "Work-sharing construct " dapat digunakan untuk membagi sebuah task diantara thread sehingga tiap thread mengeksekusi bagian yang teralokasi terkode. Baik kedua task parallelism dan data parallelism dapat didapatkan menggunakan OpenMP. Runtime environment mengalokasikan thread untuk prosesor-

prosesor tergantung dari penggunannya, mesin *load* dan factor-faktor lain. Jumlah *thread*-nya bisa di-assign oleh runtime environment berdasarkan *environment variable* atau kode menggunakan fungsi. Fungsi openMP dimasukkan ke *file header* diberi label "omp.h" di C/C++.

Contoh : menampilkan String menggunakan lebih dari satu thread

```
int main(int argc, char **argv) {
   const int N = 100000;
   int i, a[N];

   #pragma omp parallel for
   for (i = 0; i < N; i++)
      a[i] = 2 * i;

   return 0;
}
```

2.3 Intel TBB

TBB mengimplementasikan "task stealing" untuk menyeimbangkan kerja paralel sepanjang proses yang tersedia supaya meningkatkan *core utilization* dan kemudian *scaling. Task stealing* TBB model sama seperti kerja stealing yang diaplikasikan di *Cilk* (sebuah bahas pemrograman umum yang didesain untuk komputasi paralel menggunakan *multithreading*). Kinerjanya dibagi secara rata ke prosesor yang tersedia. Jika ada satu prosesor telah menyelesaikan pekerjaan di antriannya, makan TBB akan meng-*assign* pekerjaan dari prosesor yang sibuk ke yang telah selesai tersebut. Kemampuan dinamik ini mempermudah programmer, mengijinkan aplikasi tertulis menggunakan *library* untuk mengukur, dan mengatur prosesor yang tersedia dengan tidak merubah kode atau *file executable*.

2.4 Gaussian Elimination

Pada aljabar linear, *Gaussian elimination* adalah algoritma untuk menyelesainkan persamaa linear, mencari rangking matriks, dan mengkalkulasikan inverse dari sebuah matrik NxN yang mampu di-*inverse*. Dinamakan *gaussian elimination* karena penemunya adalah seorang matematikawan asal Jerman yang bernama Carl Friedrich Gauss. Prosesnya sendiri dibagi menjadi dua bagian. Pertama, *Forward Elimination* untuk mengurangi sistem yang bisa berbentuk *triangular* atau *echelon form*, atau menghasilkan hasil yang tidak ada solusi, menandakan sistem tidak memiliki solusi. Ini bisa didapatkan melalui penggunaan operasi elementer baris. Langkah kedua adalah menggunakan *back substitution* untuk mencari solusi dari sistem di atas. Berhubungan dengan matriks, langkah pertama mengurangi matriks ke bentuk baris eselon menggunakan operasi baris elementer sedangkan yang kedua menguranginya menjadi baris eselon terkurang atau bentuk kanonik baris.

Pada sudut pandang lain, dimana bisa jadi sangat berguna untuk menganalisis algoritmanya, adalah *Gaussian elimination* mengkomputasi matriks dekomposisi. Operasi elementer tiga baris digunakan di *Gaussian elimination* (perkalian baris, pergantian baris, dan menabah beberapa baris ke baris lain) sejumlah kelipatannya dari matriks original dengan matriks inversenya dari kiri. Bagian pertama dari algortima mengkomputasi LU dekomposisi, sedangkan bagian keduanya menulis matriks original sebagai matriks determinan inverse yang unik dan matriks baris eselon pengurangan determinan yang unik juga.

Gaussian elimination untuk menyelesainkan *n* persamaan dan *n* variabel membutuhkan pembagian $n(n+1) / 2$, perkalian $(2n^3 + 3n^2 – 5n)/6$, dan subtraksi

$(2n^3 + 3n^2 – 5n)/6$, untuk secara total mencapai $2n^3 / 3$ operasi sehingga menjadi $O(n^3)$. Algoritma ini dapat digunakan oleh komputer untuk sistem yang mencapai ribuan persamaan dan variabel. Bagaimanapun biaya menjadi penghalang untuk sistem dengan jutaan persamaan. Sistem yang besar ini diselesaikan menggunakan metode *iterative*. Ada metode khusus untuk sistem dengan koefisien yang memiliki pola tertentu. Penggunaan *Gaussian* elimination sendiri stabil secara numerik untuk matriks diagonal dominan atau *positif-definite.* Untuk matriks yang general, *Gaussian elimination* biasanya bisa digolongkan stabil jika Anda melihat *partial pivoting* seperti algoritma yang dideskripsikan di bawah, walaupun ada contoh yang juga bisa menyebabkannya tidak stabil.

Algoritma (berbentuk *pseudocode*) yang digunakan adalah seperti ini (menggunakan pivot)

```
i := 1
j := 1
while (i ≤ m and j ≤ n) do
  Find pivot in column j, starting in row i:
  maxi := i
  for k := i+1 to m do
    if abs(A[k,j]) > abs(A[maxi,j]) then
      maxi := k
    end if
  end for
  if A[maxi,j] ≠ 0 then
    swap rows i and maxi, but do not change the value of i
    Now A[i,j] will contain the old value of A[maxi,j].
    divide each entry in row i by A[i,j]
    Now A[i,j] will have the value 1.
    for u := i+1 to m do
      subtract A[u,j] * row i from row u
      Now A[u,j] will be 0, since A[u,j] - A[i,j] * A[u,j] = A[u,j] - 1 * A[u,j] = 0.
    end for
    i := i + 1
  end if
  j := j + 1
end while
```

3. HASIL PENELITIAN DAN PEMBAHASAN

3.1 Skenario pengukuran

Eksperimen dilakukan pada notebook Compaq seri CQ-40. Pada saat melakukan eksperimen tidak da aplikasi yang runing selain Command Prompt sebagai eksekutor dan MS-Exel untuk menyimpan dan mengolah data hasil eksperimen.

Untuk pengukuran pada ketiga teknik di atas dilakukan berbeda. Fungsi waktu yang digunakan adalah *clock()* dan *gettimeofday()*. Pemilihan waktu dengan dua fungsi ini dilakukan karena saya tidak tahu mengapa untuk pemakaian *clock_gettime()*, tipe data *timespec* tidak bisa digunakan.

Pada POSIX Thread, jumlah thread yang digunakan adalah sebanyak 10, OpenMP sebanyak 30 dan Intel TBB adalah sebanyak 10. Eksekusi dilakukan sebanyak lima kali dan diambil rata-rata waktu untuk masing-masing N. Teknik penyelesainnya digunakan *barrier* dimana prosedur yang digunakan untuk menunggu *thread* lain selesai melakukan proses.

Pada eksperimen ini dilakukan pengukuran untuk nilai N yang bervariasi yaitu 16, 32, 64, 100, 200, 400, 800 dan 1600.

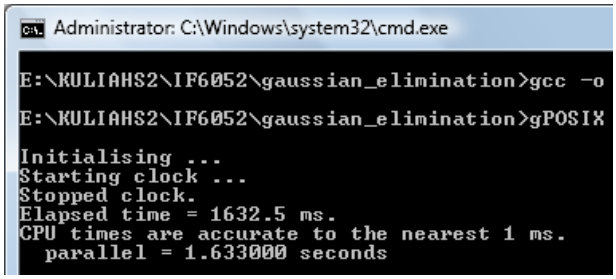
Pengukuran dilakukan lima (5) kali pengulangan untuk masing-masing nilai N. Hal ini dilakukan untuk menghasilkan beberapa nilai pembanding karena setiap kali pengukuran dapat menghasilkan nilai berbeda.

Dalam ekperimen ini nilai rata-rata dari pengukuran yang dijadikan nilai pengukuran yang akan dianalisis. Selanjutnya hasil pengukuran dimasukkan kedalam tabel dan dan ditampilkan dalam bentuk grafik.

3.2 Skenario pengukuran dengan POSIX Thread

Kompilasi menggunakan minGW "gcc -o gPOSIX gaussthread.c -l pthreadGC2"

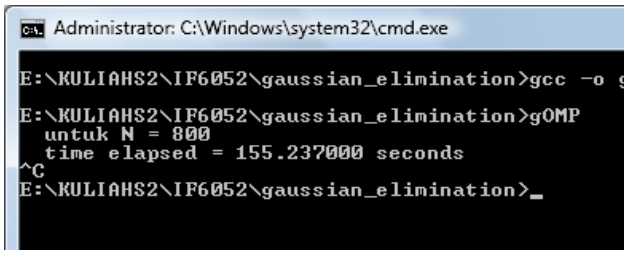
Hasil eksekusi untuk nilai N=800



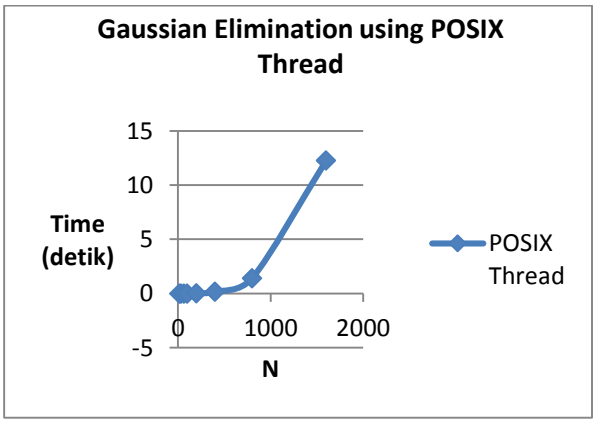
3.3 Skenario pengukuran dengan OpenMP

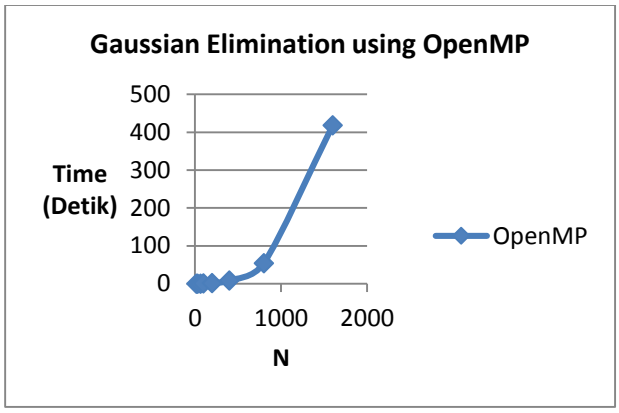
Kompilasi menggunakan minGW "gcc -o gOMP gaussopenmp.c -l pthreadGC2"

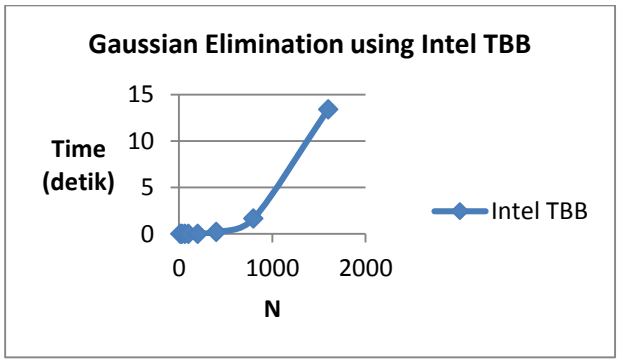
Hasil eksekusi untuk nilai N=800



3.4 Hasil eksperimen







3.5 Analisis hasil ekperimen

Pada pengukuran dengan menggunakan POSIX Thread hasil pengukuran baru terlihat pada N = 200 dan seterusnya, sedangkan untuk N=16, 32, 64 dan 100 t bernilai 0. Waktu tertinggi pada pengukuran dengan POSIX thread yaitu 12,2696 detik untuk N=1600.

Pada pengukuran dengan menggunakan OpenMP jauh lebih lambat dibandingkan dengan POSIX thread, dimana waktu tertinggi adalah 417,977 detik untuk N=1600

Hal ini mungkin diakibatkan perbedaan penggunakan fungsi CPU timer.

Namun berbeda dengan intel TBB, hasil [engukuran mendekati hasil pengukuran pada POSIX Thread.

4. KESIMPULAN DAN SARAN

4.1 Kesimpulan

Dari tiga metode pengujian yang telah dilakukan, dapat disimpulkan :

1. Masalah-masalah komputasi berat bisa diselesaikan dengan lebih cepat.
2. Operasional sistem bisa menjadi lebih hemat biaya maupun energi
3. Makin banyak peneliti bisa memakai komputasi kinerja tinggi.

4.2 Saran

Perlunya menyediakan fasilitas komputasi yang memadai untuk berbagai beban komputasi modern, khususnya mendukung proses pengolahan data dan simulasi dalam perancangan material nano dan optimasi proses industri

DAFTAR PUSTAKA

1. Koen, Billy Vaughn, "Definition of The Engineering Method", American Society for Engineering Education, Washington D.C., 1985,
2. Easterbrook, Steve, et.all, "Selecting Empirical Methods for Software Engineering Research",
3. Hong, Leung Yee, "RESEARCH METHODS IN ENGINEERING AND SCIENCE", Curtin University