Thomas Anung Basuki

Komputasi Paralel Sebagai Alternatif Solusi Peningkatan Kinerja Komputasi

Intisari

Makalah ini membahas komputasi paralel pada jaringan komputer menggunakan PVM. Untuk memperjelas, diimplementasikan dua contoh dan diukur waktu eksekusinya serta dibandingkan dengan waktu eksekusi ideal.

Abstract

This paper discusses parallel computation on computer network using PVM. For illustration, two programs are given and their execution time are compared to their ideal execution time.

I. Pendahuluan

Saat ini penggunaan komputer untuk menyelesaikan masalah sudah merasuk ke segala bidang. Hal ini karena komputasi dianggap lebih cepat dibandingkan dengan penyelesaian masalah secara manual. Seiring dengan hal tersebut, semakin dituntut proses komputasi yang semakin cepat. Untuk meningkatkan kecepatan proses komputasi, dapat ditempuh dua cara :

- peningkatan kecepatan perangkat keras,
- peningkatan kecepatan perangkat lunak.

Komponen utama perangkat keras komputer adalah processor. Saat ini, peningkatan kecepatan processor benar-benar luar biasa. Processor Pentium 4 yang dikeluarkan Intel kecepatannya sudah mencapai 1.8 GHz. Meskipun kecepatan processor dapat ditingkatkan terus, namun karena keterbatasan materi pembuatnya, tentu ada suatu batas kecepatan yang tak mungkin lagi dapat dilewati. Karena itu timbul ide pembuatan komputer multiprocessor. Dengan adanya banyak processor dalam satu komputer, pekerjaan bisa dibagi-bagi kepada masing-masing processor. Dengan demikian lebih banyak proses dapat dikerjakan dalam satu saat.

Perubahan arsitektur komputer menjadi multiprocessor memang bisa membuat lebih banyak proses bisa dikerjakan sekaligus, namun tetap tidak bisa meningkatkan kecepatan masing-masing proses. Peningkatan kecepatan setiap proses bisa dicapai melalui peningkatan kecepatan perangkat lunak. Kecepatan perangkat lunak sangat ditentukan oleh algoritmanya. Usaha untuk mencari algoritma yang lebih cepat tidaklah mudah, namun dengan adanya komputer multiprocessor, dapatlah dirancang

algoritma yang lebih cepat, yaitu dengan memparalelkan proses komputasinya.
Saat ini komputer multiprocessor masih jarang dan mahal harganya. Hal ini menyebabkan algoritma paralel yang ada sukar diimplementasikan. Untuk mengatasinya dirancanglah mesin paralel semu. Mesin paralel semu ini sebenarnya adalah jaringan komputer yang dikendalikan oleh sebuah perangkat lunak yang mampu mengatur pengalokasian proses-proses komputasi kepada processor-processor yang tersebar dalam jaringan tersebut.
Pada makalah ini dibuat dua aplikasi dari komputasi paralel. Tujuan dari pembuatan aplikasi ini hanyalah untuk menunjukkan peningkatan kecepatan yang diperoleh dari paralelisasi algoritma sekuensial, sehingga sengaja dipilih aplikasi yang sederhana. Karena ketiadaan perangkat keras komputer paralel, maka kedua aplikasi tersebut dibuat dengan menggunakan PVM (*Parallel Virtual Machine*), suatu sistem perangkat lunak yang bekerja pada jaringan komputer dan mampu mensimulasikan kerja komputer paralel. Program dibuat dengan bahasa pemroraman C pada sistem operasi Red Hat Linux.
Terdapat dua aplikasi sederhana yang dipilih untuk diimplementasikan, yaitu penjumlahan *n* bilangan secara paralel dan pengurutan *n* bilangan dengan menggunakan suatu sorting network yang disebut *Bitonic Sort*. Untuk masing-masing aplikasi akan dibandingkan dengan algoritma sekuensialnya.

II. *Parallel Virtual Machine (PVM)*

PVM adalah suatu perangkat lunak yang mampu mensimulasikan pemrosesan paralel pada jaringan komputer. Saat ini ada dua bahasa pemrograman yang didukung oleh PVM, yaitu FORTRAN dan C. Versi PVM yang paling umum digunakan berbasis UNIX, meskipun ada juga PVM berbasis Windows. Cara kerja PVM adalah dengan membuat (*spawning*) proses-proses anak yang akan dikirim ke processor-processor yang tersebar di jaringan komputer. Dengan PVM bisa ditentukan berapa jumlah processor yang akan dilibatkan dalam proses komputasi.

III. Penjumlahan Bilangan

III.1. Penjumlahan Secara Sekuensial

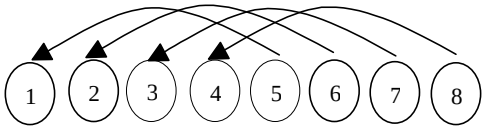
Algoritma penjumlahan secara sekuensial untuk *n* data, adalah sebagai berikut :

```
1   sum ← a[1]
2   for i ← 2 to n do
3       sum ← sum + a[i]
```

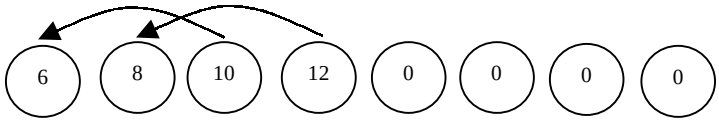
Waktu eksekusi algoritma di atas sebanding dengan banyaknya eksekusi loop for. Loop tersebut pasti akan dieksekusi sebanyak *n*-1 kali. Jadi waktu eksekusinya adalah *O*(*n*).

III.2. Penjumlahan Secara Paralel

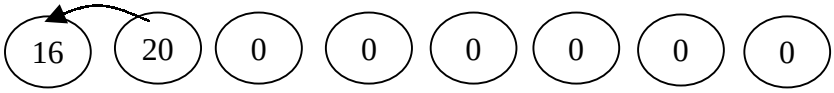
Untuk menjumlahkan *n* buah bilangan, terdapat satu syarat yaitu *n* adalah bilangan yang merupakan hasil perpangkatan dari 2 (*power of* 2). Cara kerja algoritma penjumlahan paralel dapat dilihat pada contoh berikut ini.
Pada contoh ini akan dijumlahkan 8 buah data (1+2+…+8). Untuk itu diperlukan 4 buah prosesor. Mula-mula :



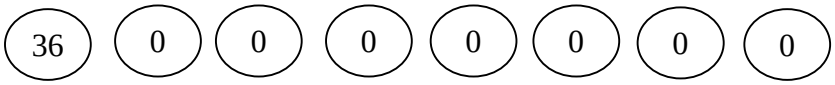
Tahap 1: prosesor ke-*i* akan menyimpan hasil penjumlahan data ke-*i* dan data ke-(*i*+4)



Tahap 2 : prosesor ke-*i* akan menyimpan hasil penjumlahan data ke-*i*, ke-(*i*+2), ke-(*i*+4) dan ke-(*i*+6)



Tahap 3: seluruh data sudah dijumlahkan dan disimpan di prosesor ke-1



Dari gambar di atas dapat dirancang algoritma penjumlahan paralel tersebut :

```
1    k ← n/2
2    while (k > 0) do
3     for i ← 1 to k in
parallel do
4      a[i] ← a[i] + a[i+k]
5     k ← k/2
```

Hasil akhir dari operasi di atas akan disimpan di a[1]. Jika diperhatikan, waktu eksekusi dari algoritma di atas sebanding dengan banyaknya eksekusi while loop. While loop akan berhenti dieksekusi bila $k = 0$, berarti bergantung dari k. Karena pada setiap akhir eksekusi while loop, nilai k akan dibagi 2, maka waktu eksekusinya adalah $O(\log n)$. Dengan demikian peningkatan yang diperoleh dengan memparalelkan penjumlahan adalah $O(n/\log n)$.

Untuk mengimplementasikan algoritma di atas idealnya dibutuhkan $n/2$ processor. Dengan teknik-teknik pemrograman paralel, algoritma di atas bisa diimplementasikan dengan jumlah processor kurang dari $n/2$.

Algoritma tersebut diimplementasikan menjadi dua program : `mainsum.c` yang merupakan terjemahan dari algoritma di atas, dan `adder.c` yang merupakan program untuk menjumlahkan dua bilangan (sama dengan baris ke-4 dari algoritma di atas). Pada setiap iterasi `mainsum.c` akan membuat k buah proses `adder.c` dan mengirim data $a[i]$ dan $a[i+k]$ ke proses ke-i. Setiap proses `adder.c` akan mengirim kembali hasil penjumlahan $a[i]$ dan $a[i+k]$ ke proses `mainsum.c`.

IV. Pengurutan *(Sorting)*

IV.1 Pengurutan Sekuensial

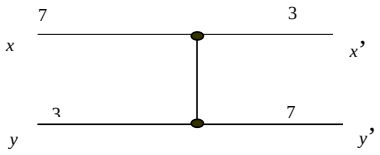
Terdapat dua jenis algoritma pengurutan yaitu pengurutan dengan pembandingan dan pengurutan tanpa pembandingan. Yang termasuk ke dalam algoritma pengurutan dengan pembandingan antara lain adalah : Bubble Sort, Insertion Sort, Selection Sort dengan waktu eksekusi $O(n^2)$, serta Heap Sort dan Quick Sort dengan waktu eksekusi $O(n \log n)$. Yang termasuk algoritma pengurutan tanpa pembandingan antara lain adalah Count Sort dan Radix Sort dengan waktu eksekusi $O(n)$.

IV.2 Bitonic Sort

Bitonic Sort adalah suatu algoritma pengurutan paralel menggunakan sorting network. Komponen terkecil dari sorting network adalah ***comparator***, yaitu suatu perangkat dengan dua masukan, *x* dan *y*, dan dua keluaran, *x*' dan *y*', di mana *x*' = min(*x*,*y*) dan *y*' = max(*x*,*y*).

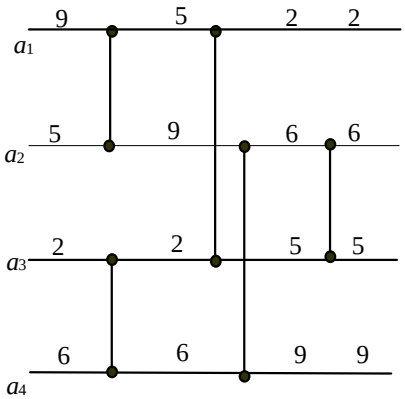
```
COMPARATOR(x,y)
      if x > y then
         temp ← x
         x ← y
         y ← temp
```

Algoritma 1 : Comparator



Gambar 1 : Comparator

Diasumsikan setiap comparator membutuhkan waktu *O*(1) untuk melakukan operasinya. Dengan menghubungkan sejumlah comparator dengan kabel, dapat dibentuk sebuah ***comparison network***.



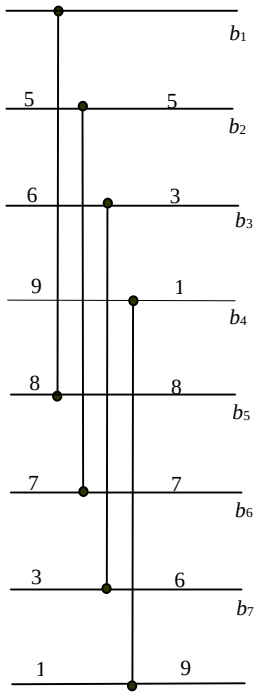
Sebuah ***sorting network*** adalah comparison network di mana untuk sembarang masukan akan menghasilkan keluaran yang terurut menaik ($b_1 \leq b_2 \leq \ldots \leq b_n$).

Suatu ***bitonic sequence*** adalah suatu sequence yang terurut menaik kemudian terurut menurun atau terurut menurun kemudian menaik. Sequence <1,4,6,8,3,2> dan <9,8,3,2,4,6> adalah contoh bitonic sequence.

Suatu ***half cleaner*** adalah comparison network dengan kedalaman 1 di mana masukan ke-*i* akan dibandingkan dengan masukan ke-(*i*+*n*/2) untuk *i* = 1,2,…,*n*/2. Jika masukan dari half cleaner adalah bitonic sequence berukuran *n*, maka keluarannya adalah dua bitonic sequence yang masing-masing berukuran *n*/2 dan data pada bitonic sequence ke-2 ≥ data bitonic sequence ke-1. Half cleaner dapat diterjemahkan menjadi modul algoritma berikut ini, yang waktu eksekusinya adalah *O*(1) :

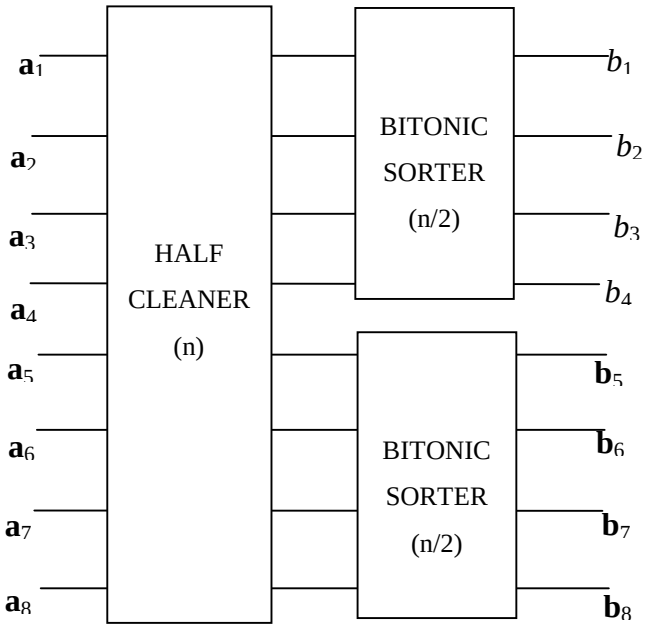
```
HALF-CLEANER(a, start, n)
   for i ← start to
   start+n/2-1 in parallel do
     COMPARATOR(a[i],a[i+n/2])
```

Algoritma 2 : Half-Cleaner



Gambar 2 : Half-Cleaner

Dengan menggunakan half cleaner dapat dibentuk ***Bitonic Sorter***. Bitonic Sorter(*n*) dapat dibentuk dengan menghubungkan kabel keluaran dari half cleaner(*n*) dengan kabel masukan dari dua Bitonic Sorter(*n*/2).



Gambar 3 : Bitonic-Sorter

```
BITONIC-SORTER(a, start, n) /*asumsi n genap*/
   HALF-CLEANER(a, start, n)
   if n>2 then
      for i ← 0 to 1 in parallel do
          BITONIC-SORTER(a, start+i*n/2, n/2)
```

Algoritma 3 : Bitonic-Sorter

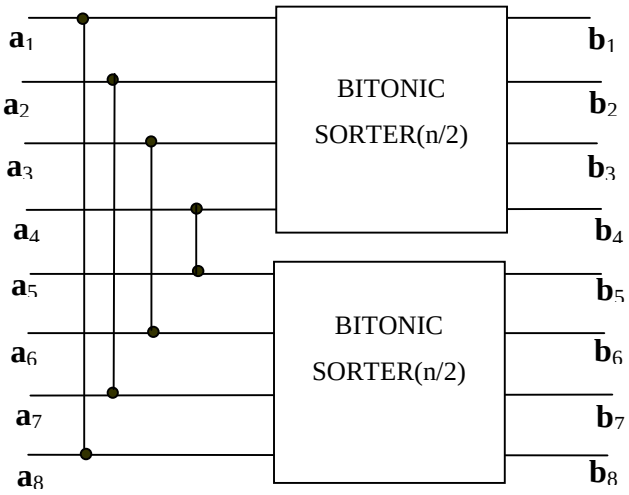
Waktu eksekusi Bitonic Sorter(n) dapat dihitung sebagai berikut :

$$T(n) = \begin{cases} 0 & ,n = 1 \\ 1 + T(n/2) & ,n > 1 \end{cases}$$

Solusinya adalah T(n) = O(log n). Jadi waktu eksekusi Bitonic Sorter(n) = O(log n).

Bitonic Sorter sudah mampu mengurutkan, namun masukannya berupa bitonic sequence. Diinginkan sebuah sorting network yang mampu mengurutkan sembarang data. Tahap berikutnya dari pembuatan sorting network tersebut adalah membentuk network yang mampu menggabungkan dua sequence berukuran n/2 yang sudah terurut menaik, menjadi sebuah sequence berukuran n yang terurut menaik. Ini disebut ***merging network***. Merging network berukuran n dapat dibentuk dengan memodifikasi half-

cleaner pertama dari bitonic sorter(n).
Waktu eksekusinya juga $O(\log n)$.



Gambar 4 : Merging Network

```
MERGER(a, start, n) /*asumsi n genap*/
  for i ← 1 to n/2 in parallel do
      COMPARATOR(a[start+i-1], a[start+n-i])
  if n>2 then
     for i ← 0 to 1 in parallel do
         BITONIC-SORTER(a, start+i*n/2, n/2)
```

Algoritma 4 : Merging Network

Akhirnya dengan menggabungkan merging network, didapat sebuah sorting network yang dapat mengurutkan sembarang sequence berukuran *n* (SORTER(*n*)).

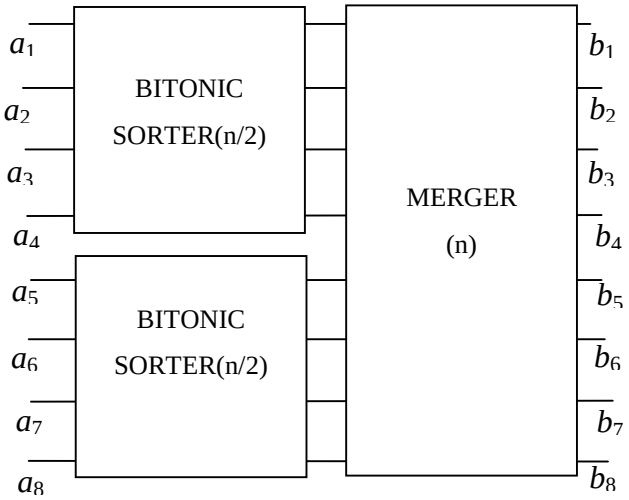
```
SORTER(a, start, n) /*asumsi n genap*/
   if n>2 then
       for i ← 0 to 1 in parallel do
            SORTER(a, start+i*n/2, n/2)
   MERGER(a, start, n)
```

Algoritma 5 : Bitonic-Sorter

Waktu eksekusi Bitonic-Sort adalah :

$$T(n) = \begin{cases} 0 & ,n = 1 \\ O(\log n) + T(n/2) & ,n > 1 \end{cases}$$

Solusinya adalah $O(\log^2 n)$.

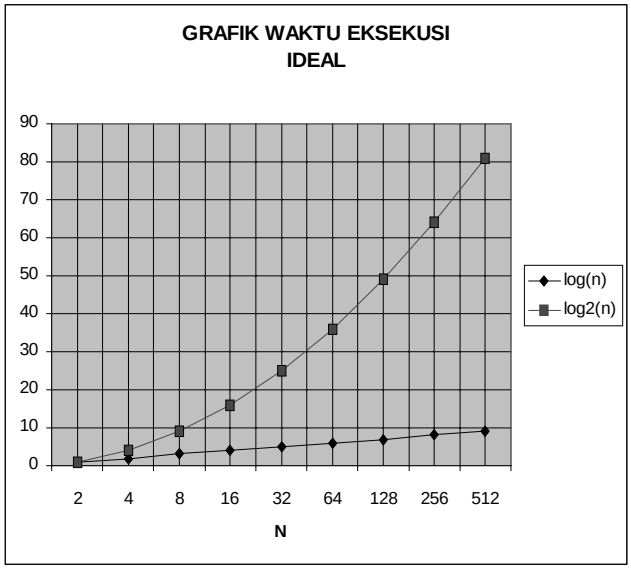


Gambar 5 : Bitonic-Sorter

V. Analisis Hasil

Dari pembahasan pada bab-bab sebelumnya diketahui bahwa untuk penjumlahan paralel waktu eksekusinya adalah $O(\log n)$ sedangkan bitonic sort waktu eksekusinya adalah $O(\log^2 n)$. Berikut ini akan diperiksa apakah waktu eksekusi yang sesungguhnya sesuai dengan waktu teoritisnya.

Grafik di bawah ini menunjukkan grafik $\log n$ dan $\log^2 n$. Terlihat bahwa jika n adalah kepangkatan dari 2, maka waktu eksekusi penjumlahan paralel ($\log n$) seharusnya menaik secara linier, sedangkan waktu eksekusi Bitonic Sort ($\log^2 n$) seharusnya menaik secara kuadratik.



Tabel 1. Waktu Eksekusi Penjumlahan Paralel (dalam detik)

n	1 prosesor	2 prosesor	3 prosesor	4 prosesor	Rata-rata
16	0.00	0.01	0.01	0.00	0.01
32	0.01	0.01	0.01	0.02	0.01
64	0.02	0.02	0.01	0.03	0.02
128	0.05	0.05	0.02	0.02	0.04
256	0.05	0.04	0.04	0.02	0.04
512	N/A	0.17	0.11	0.11	0.13

Tabel 2. Waktu Eksekusi Pengurutan Paralel (dalam detik)

n	1 prosesor	2 prosesor	3 prosesor	4 prosesor	Rata-rata
16	0.02	0.02	0.03	0.04	0.03
32	0.14	0.11	0.09	0.11	0.11
64	0.67	0.64	0.47	0.53	0.58
128	2.41	2.75	2.37	2.24	2.44
256	N/A	9.03	8.96	9.55	9.18
512	N/A	N/A	34.97	35.98	35.48

Dari tabel 1, tampak bahwa waktu eksekusi penjumlahan paralel menaik secara linier. Jadi **waktu eksekusi penjumlahan paralel** bisa dianggap **sesuai dengan waktu idealnya** ($O(\log n)$).

Dari tabel 2, tampak bahwa waktu eksekusi Bitonic Sort tidak menaik secara kuadratik. Jadi **waktu eksekusi pengurutan paralel** ternyata **lebih buruk daripada waktu idealnya**.

Pada tabel 1, untuk data sebanyak 512 dengan 1 prosesor, PVM tidak mampu melakukannya (pesan kesalahan : gagal mengirim data). Penulis tidak tahu pasti penyebab terjadinya hal tersebut, namun diduga hal itu terjadi karena keterbatasan PVM dalam mengolah data yang cukup besar. Pada tabel 2, untuk data yang cukup besar dengan jumlah prosesor sedikit (n = 256, 1 prosesor; n = 512, 1 atau 2 prosesor), waktu eksekusinya terlalu lama (> 1 jam) sehingga tidak penulis sertakan. Hal ini diduga juga terjadi karena **kelemahan PVM dalam mengolah data besar dengan prosesor sedikit**. Selain itu juga tampak bahwa **jumlah prosesor ternyata tidak terlalu berpengaruh pada PVM**.

VI. Referensi

[1] Cormen, T.H., Leiserson, C. dan Rivest, R.E. *Introduction to Algorithms*. MIT Press, 1990

[2] Kernighan, B.W. dan Ritchie, D.M. *The C Programmming Language*. Prentice-Hall, 1989.

[3] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V. *PVM : Parallel Virtual Machine, A Users' Guide And Tutorial For Networked Parallel Computing*. MIT Press, 1994.

Penulis

Thomas Anung Basuki adalah dosen jurusan Ilmu Komputer FMIPA Universitas Katolik Parahyangan.