ANALISA KINERJA CLUSTER LINUX DENGAN PUSTAKA MPICH TERHADAP PERKALIAN MATRIKS

Fani Fatullah, A.Benny Mutiara MQN,Chandra Yulianto

Universitas Gunadarma
Jl. Margonda Raya 100, Depok, 16424
e-mail : amutiara@staff.gunadarma.ac.id,chandra@staff.gunadarma.ac.id

Abstrak

Cluster yaitu penyatuan sekelompok data yang mempunyai korelasi atau karakteristik sejenis, berarti kita menyatukan PC-PC atau server (atau apapun device yang ber-OS) ke dalam satu kesatuan komputasi. Cluster pada dasarnya juga menggunakan jaringan, tapi yang menjadi ciri khususnya adalah bagaimana konfigurasi ini digunakan untuk menyelesaikan problem. Problem disini adalah bagaimana menyelesaikan sebuah perkalian matrix sehingga akan terlihat speedUp PC cluster dengan pustaka MPICH berupa garis linear.

Kata kunci : matriks, cluster, MPICH

1. Pendahuluan

Awalnya, semua komputer datang dalam model yang sangat besar dan sangat lambat bila dibandingkan dengan komputer saat ini. Saat itu merupakan jaman prehistoris dalam dunia komputer, dimana komputer harus diimplementasikan dengan biaya apapun. Dan walaupun tidak bisa dibilang cepat, komputer-komputer saat itu dengan mudah dapat melampaui kemampuan hitung manusia.

Komputasi paralel merupakan salah satu teknologi paling menarik yang penting sejak ditemukannya komputer elektronik pada tahun 1940-an. Terobosan dalam pemrosesan paralel selalu berkembang dan mendapatkan tempat disamping teknologi-teknologi lainnya sejak Era Kebangkitan (1950-an), Era Mainframe (1960-an), Era Minis (1970-an), Era PC (1980-an), dan Era Komputer Paralel (1990-an). Dengan berbagai pengaruh atas perkembangan teknologi lainnya, dan bagaimana teknologi ini mengubah persepsi terhadap komputer, dapat dimengerti betapa pentingnya komputasi paralel itu. Inti dari *komputasi paralel* yaitu hardware, software, dan aplikasinya. Paralel Prosesing merupakan suatu pemrosesan informasi yang lebih mendekatkan pada manipulasi rata-rata dari elemen data terhadap satu atau lebih penyelesaian proses dari sebuah masalah. Oleh karena itu komputasi paralel dapat diartikan bahwa komputer dengan banyak prosesor yang mampu untuk melakukan paralel prosesing.

Paralelisme telah diakui sebagai solusi terhadap masalah untuk membuat komputer semakin cepat dan lebih cepat lagi. Dan kelihatannya belum ditemukan batas terhadap kemampuan peralatan elektronik dalam peningkatan kecepatan. Tapi faktanya, terdapat batas fisik untuk menaikkan kecepatan dari masing-masing komponen. Cepat atau lambat, batas ini akan tercapai, dan paralelisme akan menjadi satu-satunya penyelamat.

Suatu program paralel memerlukan koordinasi ketika sebuah tugas bergantung pada tugas lainnya. Ada dua macam bentuk koordinasi pada komputer paralel: asynchronous dan synchronous.
Bentuk synchronous merupakan koordinasi pada hardware yang memaksa semua tugas agar dilaksanakan pada waktu yang bersamaan dengan mengesampingakan adanya ketergantungan tugas yang satu dengan lainnya. Sementara bentuk asynchronous mengandalkan mekanisme pengunci untuk mengkoordinasikan prosesor tanpa harus berjalan bersamaan.

Pada shared memory multiprocessor, sejumlah prosesor terkoneksi dengan sejumlah modul memori. Setiap prosesor terkoneksi dengan sejumlah modul memori. Setiap prosesor bisa mengakses modul memori manapun sehingga disebut konfigurasi shared memory.Contoh sederhana dari shared memory multiprocesor adalah dual pentium dan quad pentium.

2. Distributed Shared Memory

Multiprocessor diminta dengan penggunaan shared memory, yaitu memori yang dapat dipakai bersama-sama oleh seluruh pemroses. Distributed Shared Memory (DSM) menyediakan lingkungan shared memory secara logic pada jaringan komputer (network of workstation) yang secara fisik memorinya terdistribusi.

Karena pertimbangan biaya, banyak sistem pemrosesan parallel saat ini dibangun dengan menghubungkan komputer PC dalam jaringan lokal yang murah. Untuk mengembangkan program paralel di lingkungan ini lebih mudah diajukan DSM sehingga memungkinkan penggunaan lingkungan pemrograman shared-memory pada sistem memori yang secara fisik terdistribusi.

Keuntungan dari DSM software terletak pada kemudahan menggabungkan teknik optimasi sehingga banyak overhead dari messege passing bisa dihilangkan. Pengiriman messege tidak dilakukan secara eksplisit (tersembunyi dari program), dan bisa juga menangani data yang besar dan kompleks, tanpa replikasi atau mengirim data ke proses. Pemrogram juga bisa mengetahui perilaku DSM melali antarmuka untuk meningkatkan kinerja dari aplikasi.

Sejumlah Penelitian mengenai DSM mengkaji permasalahan mengenai :

1. Model konsistensi.
2. Algoritma dan protokol koherensi memori.
3. Implementasi hardware dan software.
4. Dukungan bahasa.
5. Analisis dan evaluasi kinerja.
6. Sinkronisasi, fault tolerance, heterogenitas, dan persistensi.

alam hal ini terdapat sejumlah model konsistensi, antara lain :

- Strict Consistency : proses melihat hasil paling up-to-date (suatu read akan melihat write terakhir). Kerugiannya, jumlah message dan latency besar, juga tidak selalu diperlukan update seketika.
- Sequential Consistency : hasil eksekusi eksekusi seperti rpogram sequensial.
- Relaxed Consistency :terdapat delay untuk mengurangi jumlah message.
- Weak Consistency : pemrogram harus menggunakan operasi sinkronisasi untuk memaksakan hasil sequentially consistency bila diperlukan.
- Release Consistency : pemrogram harus menentukan operasi sinkronisasi, acquire (sebelum shared data dibaca), dan release, misalnya lock dan unlock. Bisa juga dianggap perluasan dari weak consistency.
- Lazy Release Consistency: update hanya dilakukan saat acquire, jumlah message lebih sedikit.
- Banyak sistem DSM telah dikembangkan dengan mengikuti strategi rancangan yang berbeda dan penekanan berbeda pada aspek kinerja dan fungsionalitas. Sayangnya, shared memory tersebut biasanya memiliki kecenderungan bottleneck, dan sistemnya memiliki tingkat skalabilitas yang rendah. Salah satu pemikiran dalam implementasi DSM adalah pada lapisan mana DSM akan dibuat.

Klasifikasi Distributed Shared Memory

Berdasarkan cara pembagian shared memory-nya, DSM dapat dibagi menjadi dua macam, yakni :

DSM yang berdasarkan blok/page (blok/page based DSM)

Pada DSM berdasarkan blok, ruang pengalamatan shared memory dibagi menjadi halaman logika berupa blok yang berukuran tetap kemudian didistribusikan ke host dalam sistem. Melalui sebuah memory manager, host bisa mengakses ke setiap halaman dalam ruang pengalamatan shared. DSM berdasarkan blok biasanya merupakan perluasan dari sistem memori semu tradisional, dan biasanya diimplementasikan pada lapisan lapisan perangkat keras atau sistem operasi. Keuntungan implementasi ini adalah sifat transparansinya. Sifat transparansi ini ditunjukkan dengan kenyataan bahwa sistem memori terdistribusi pada komputer yang berbeda sepenuhnya tersembunyi dari pemakai. Pemakai bisa melihat sistem DSM seperti shared memory pada multiprocessor, namun dengan kecepatan komunikasi yang lebih lambat.

Bagaimanapun juga, DSM berdasar blok memiliki masalah pemilihan ukuran blok. Ukuran blok yang besar meningkatkan delay dalam melakukan pengiriman blok dan kemungkinan false sharing, sedangkan ukuran blok yang kecil menghasilkan intensitas yang tinggi pada transmisi pada blok sehingga meningkatkan overhead untuk transfer blok. Keperluan untuk memelihara status blok juga meningkat. Ukuran blok tidak hanya bergantung pada ciri dari sistem tapi juga pada aplikasinya. Sejauh ini belum ada cara yang sistematis untuk memilih parameter ukuran blok. Dengan demikian, hal ini merupakan pekerjaan yang sulit, tidak hanya bergantung pada karakteristik sistem, tapi juga aplikasi.

Masalah lain dengan DSM berdasarkan blok adalah kerumitan perancangan. Implementasi pada lapisan perangkat keras dan sistem operasi memerlukan modifikasi atas sistem yang sudah ada, yang memerlukan usaha yang besar. Biaya untuk mengadakan dan memelihara sistem tersebut akan menjadi tinggi.
Komputer yang berbeda memiliki ukuran halaman yang berbeda dan metode manajemen memori yang berbeda. DSM yang berdasarkan blok biasanya bergantung pada sistem sehingga sulit untuk diimplementasikan pada sistem yang heterogen.

DSM yng berdasarkan object (Object Based DSM)

DSM berdasarkan object melakukan pembagian shared memory ke dalam unit-unit kecil. Setiap unit memuat data yang berhubungan secara logika. Implementasinya biasa dilakukan pada lapisan bahasa/kompilator dan librari. Karena DSM berdasar objek diimplementasikan pada lapisan yang lebih tinggi, kinerjanya tidak sebaik DSM berdasarkan blok tetapi lingkungan yang dihasilkan lebih independen dan fleksibel. DSM bisa diimplementasikan diatas subsistem komunikasi yang sudah ada seperti PVM, TCP/IP, atau DCE. Akibatnya, usaha pengembangan sistem bisa berkurang dengan cepat, dan pengubahan sistem pada arsitektur yang lebih mudah. Lingkungan tersebut akan mampu mendukung sistem yang heterogen. Pada DSM berdasarkan objek, pemrogram memiliki kendali yang besar atas bagaimana data akan didistribusikan sesuai dengan ciri dari aplikasi. Pemrogram juga dibiarkan untuk menentukan parameter yang penting seperti ukuran blok, metode akses, mekanisme komunikasi, dan model konsistensi memori.

Pengolahan Paralel adalah pengolahan informasi yang menekankan pada manipulasi data-data elemen secara simultan dengan maksud untuk mempercepat komputasi dari sistem komputer dan menambah jumlah keluaran yang dapat dihasilkan dalam jangka waktu tertentu.
Komputer Paralel memiliki kemampuan untuk melakukan pengolahan paralel. Sedangkan throughput yang dihasilkan adalah keluaran yang dihasilkan per unit waktu. Peningkatan throughput dapat dilakukan dengan meningkatkan kecepatan operasi serta meningkatkan jumlah operasi yang dapat dilakukan dalam satu waktu tertentu (concurrency).

Dalam menyelesaikan suatu permasalahan persamaan matematika terdapat dua alternative yakni menggunakan algoritma sequensial dan algoritma parallel.
Sebagai contoh bila terdapat persamaan sebagai berikut :

$$\begin{aligned} 1X_1 + 1X_2 - 1X_3 + 4X_4 &= 8 \\ -2X_2 - 1X_3 + 1X_4 &= 5 \\ 2X_3 - 3X_4 &= 0 \\ 2X_4 &= 4 \end{aligned}$$

Bila persamaan tadi diselesaikan dengan Algoritma Sequensial maka didapat sebagai berikut :

Vektor $\mathbf{x} = [\, X_1 \;\; X_2 \;\; X_3 \;\; X_4 \,]^T$ diperoleh melalui prosedur berikut :

1. menghitung : $X_4 = 4/2 = 2$
2. nilai X_4 disubtitusikan ke semua persamaan di atasnya, koreksi ruas kanannya persamaan; hasilnya adalah :

$$1X_1 + 1X_2 - 1X_3 = 0$$
$$-2X_2 - 1X_3 = 3$$
$$2X_3 = 6$$

3. langkah (1) dan (2) diulangi berturut-turut untuk X_3 dan X_2
4. langkah (1) diulangi untuk X_1

Dengan cara ini maka diperoleh :

$\mathbf{x} = [X_1 \; X_2 \; X_3 \; X_4]^T = [9 \; -6 \; 3 \; 2]^T$

Sistem di atas dapat dituliskan sebagai **A x** = **b**, dimana :

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & -1 & 4 \\ & -2 & -3 & 1 \\ & & 2 & -3 \\ & & & 2 \end{bmatrix} = \text{matriks segitiga atas}, \quad \mathbf{b} = \begin{bmatrix} 8 \\ 5 \\ 0 \\ 4 \end{bmatrix}$$

Untuk sistem segitiga atas berukuran *n* prosedur di atas dapat digeneralisasi menjadi algoritma sekuensial berikut :

Algoritma BACK.SUBSTITUTION (SISD) :

```
Global  n                   {ukuran sistem}
        a[1.. n][ 1.. n]    {elemen-elemen matriks A}
        b[1.. n]            {elemen-elemen vektor b}
        x[1.. n]            {elemen-elemen vektor x}
        i                   {indeks kolom}
        j                   {indeks baris}
```

```
1. begin
2.   for i ← n downto 1 do
3.     x[i] ← b[i]/a[i][i]
4.     for j ← 1 to i –1 do
5.       b[j] ← b[j] – x[i] × a[j][i]
6.           a[j][i] ← 0 {statement ini bersifat opsional}
7.     endfor
8.   endfor
9. end
```

Algoritma melakukan operasi pembagian, perkalian, dan pengurangan. Banyaknya operasi pembagian adalah *n* (baris 3) sedangkan banyaknya operasi pengurangan dan perkalian (baris 5) adalah :

$$(n-1) + (n-2) + ... + 2 + 1 = \tfrac{1}{2} n(n-1) \qquad (1)$$

sehingga kompleksitas algoritma 9.1. adalah $\theta(n^2)$.

Instruksi pada baris 6 bersifat opsional, di antaranya untuk menayangkan matriks pada setiap iterasi.

Kemudian bila persamaan di atas menggunakan Algoritma Paralel maka Berdasarkan potensi paralelisasi maka algoritma paralel dapat disusun sebagai berikut :

Algoritma BACK.SUBSTITUTION (UMA MULTIPROSESOR) :

```
Global  n                       {ukuran sistem}
        p                       {banyaknya proses}
        a[1.. n][ 1.. n]        {elemen-elemen matriks A}
        b[1.. n]                {elemen-elemen vektor b}
        x[1.. n]        {elemen-elemen vektor x}
        i                       {indeks kolom}
Lokal   j                       {indeks indetifier proses}
        k                       {indeks baris}
```

```
1.  begin
2.    for i ← n downto 1 do
3.      x[i]← b[i]/ a[i][i]
4.      forall P[j] where 1 ≤ j ≤ p do
5.        for k ← j to i –1 step p do
6.              b[k]← b[k] – x[i] × a[k][i]
7.              a[k][i] ← 0 {statement ini bersifat opsional}
8.        endfor
9.      endforall
10.   endfor
11. end
```

Algoritma paralel melakukan *n* operasi pembagian. Banyak-nya operasi pengurangan dan perkalian adalah :

$$\lceil (n-1)/p \rceil + \lceil (n-2)/p \rceil + ... + \lceil 2/p \rceil + \lceil 1/p \rceil = \tfrac{1}{2} \lceil n(n-1)/p \rceil \quad (2)$$

sehingga kompleksitas algoritma adalah $\theta(n^2 / p)$.

Berikut ini adalah *tracing* penerapan algoritma paralel terhadap sistem pada persamaan (2). untuk *p = 2* :

i = 4 : x[4] = b[4]/a[4][4] = 4/2 = 2 **∴ x[4] = 2**

∴ x[4] = 2

P[1] : k = 1 : b[1] = b[1] - x[4] × a[1][4] = 8 - 2 × 4 = 0
a[1][4] = 0
k = 3 : b[3] = b[3] - x[4] × a[3][4] = 0 - 2 × (-3) = 6
a[3][4] = 0

P[2] : k = 2 : b[2] = b[2] - x[4] × a[2][4] = 5 - 2 × 1 = 3
a[2][4] = 0

i = 3 : x[3] = b[3]/a[3][3] = 6/2 = 3 **∴ x[3] = 3**

∴ x[3] = 3

P[1] : k = 1 : b[1] = b[1] - x[3] × a[1][3] = 0 - 3 × (-1) = 3
a[1][3] = 0

P[2] : k = 2 : b[2] = b[2] - x[3] × a[2][3] = 3 - 3 × (-3) = 12
a[2][3] = 0

∴ x[2] = -6

i = 2 : x[2] = b[2]/a[2][2] = 12/(-2) = -6 **∴ x[2] = -6**

P[1] : k = 1 : b[1] = b[1] - x[2] × a[1][2] = 3 - (-6) × 1 = 9
a[1][2] = 0

∴ x[1] = 9

i = 1 : x[1] = b[1]/a[1][1] = 9/(1) = 9 **∴ x[1] = 9**

3. Speedup

Pengukuran speedup sebuah algoritma paralel adalah salah satu cara untuk mengevaluasi kinerja algoritma tersebut.

Speedup adalah perbandingan antara waktu yang diperlukan algoritma sekuensial yang paling efisien untuk melakukan komputasi dengan waktu yang dibutuhkan untuk melakukan komputasi yang sama pada sebuah mesin pipeline atau paralel.

$$\text{Speedup} = \frac{\text{Worst case running time dari algoritma sekuensial terefisien}}{\text{Worst case running time dari algoritma paralel}}$$

4. Arsitektur Jejaring Cluster Linux dengan Pustaka MPICH

Spesifikasi komputer Jaringan Cluster :

Server :

- Prorocessor AMD Duron 1300 MHz On Board Motherboard ECS K7SOM+
- Memori DDR 256 MB
- Hard Disk 40 GB 5400 RPM
- LAN Card 10/100 MBPS Edimax
- Switch 100 MBPS
- VGA On board 32 MB share memory
- - Sistem Operasi Linux Slackware 9.1

Client :

- Prorocessor AMD Duron 1300 MHz On Board Motherboard ECS K7SOM+
- Memori DDR 128 MB
- Hard Disk 40 GB 5400 RPM
- LAN Card 10/100 MBPS Edimax
- VGA Onboard 32 MB Share memory
- Sistem Operasi Slackware 9.1

5. Hasil Pengujian Cluster Linux dengan Pustaka MPICH

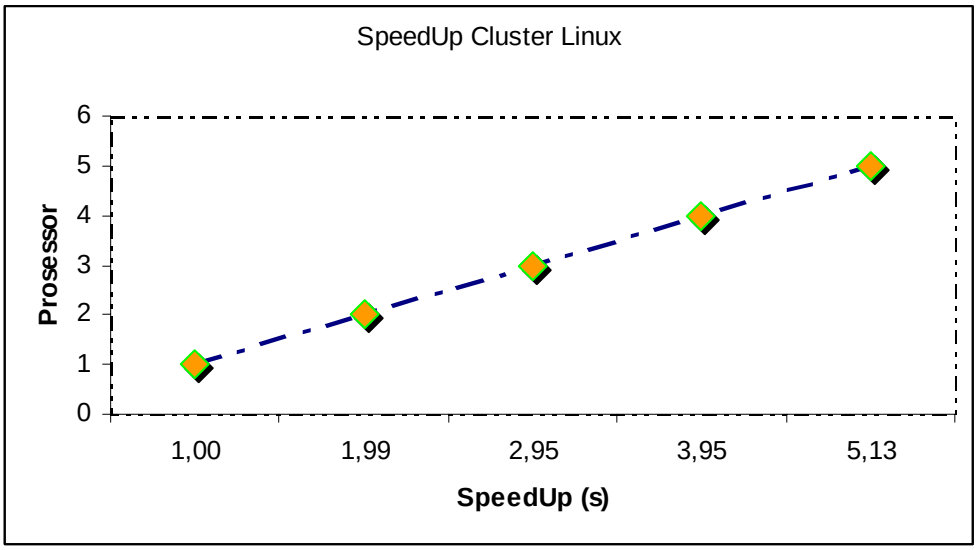
Pengujian cluster linux dengan pustaka MPICH terhadap perkalian matrix dengan menggunakan algorima paralel.

Dari tabel dibawah, bisa diprediksi waktu yang diperlukan oleh sebuah cluster linux untuk menyelesaikan perkalian yang dinamakan kompleksitas paralel. Kompleksitas Paralel akan sama dengan waktu dari 3 data yang diambil, pengambilan ini dilakukan untuk melihat waktu tercepat cluster linux menyelesaikan perkalian matrix.

Tabel 1. Pengujian Cluster MPI dengan dimensi Matrix 2048

p	n	Dimensi Matrix (n^2)	Waktu (s)	Rata-Rata (s)	Kompleksitas (n^2/p)	SpeedUp
1	12	2048	2.072.435.246	2063,949194	2048,00	1.00
1	12	2048	2.067.135.424			
1	12	2048	2.063.949.194			
2	12	2048	1.047.131.981	1038,698729	1024,00	1.99
2	12	2048	1.038.698.729			
2	12	2048	1.050.809.621			

3	12	2048	707.322.012	700,822230	682,00	2.95
3	12	2048	706.542.089			
3	12	2048	700.822.230			
4	12	2048	526.096.258	522,545004	512,00	3.95
4	12	2048	522.545.004			
4	12	2048	531.531.853			
5	12	2048	371.265.447	402,623147	409,60	5.13
5	12	2048	420.623.147			
5	12	2048	423.268.466			



Gambar 1. SpeedUp Cluster MPI

Untuk jumlah prosesor tertentu nilai SpeedUp akan meningkat menurut kenaikan ukuran matriks *n*. SpeedUp paralelisasi algoritma substitusi mundur untuk berbagai ukuran matriks tridiagonal. Dengan bertambahnya node cluster akan terlihat SpeedUp nya semakin besar, tetapi juga perlu diperhatikan kondisi jaringan pada saat melakukan pengujian.

6. Kesimpulan.

Dari hasil yang diperoleh selama pengujian cluster openMosix dan cluster MPI maka penulis dapat mengambil kesimpulan bahwa :

1. Apabila ada suatu proses, maka proses itu akan langsung disebar kesemua node tergantung option ynag dipakai apakah akan 2 node, 3 node, 4 node, atau 5 node proses yang akan tersebar.
2. Pada algoritma paralel perhitungan paralelnya n^2 / p akan mendekati waktu eksekusi tiap proses, dimana n itu jumlah dimensi dan p itu waktu proses.

3. SpeedUp dapat dihasilkan dari waktu sequensial dibagi dengan waktu paralel. Waktu squensial disini adalah waktu terkecil dari 3 kali percobaan.

7. Daftar Pustaka

[1] Michael J. Quinn, ***Parallel Computing : Theory and Practice Second Edition***, 1994.
[2] Ted G. Lewis & Hesham El-rewini, *Introduction to Parallel Computing*, Prentice-Hall Internasional Editions, 1992.
[3] The cluster how-to URL: http://www.sourceforge.net/
[4] Instalasi lam-mpi URL : http://www.lam-mpi.org
[6] Instalasi dan Konfigurasi MPICH URL: ftp://ftp.mcs.anl.gov/mpi/tcltk
[7] Firrar Utdirartatmo, *Pemrograman Paralel dengan PVM di LINUX dan WINDOWS*, ANDI Yogyakarta, 2002.
[8] _____, *Pemrograman Paralel di LINUX Berbasis DSM (Distributed Shared Memory)*, ANDI Yogyakarta, 2003.
[9] _____ , Installation and User's Guide to a Portable Implementation of MPIv1.2.5 The device for Workstation Networks