Evaluasi Kinerja *Sparse Matrix-Vector Multiplication* menggunakan Format Penyimpanan *CSR* dan *BCSR* pada MPI

Performance Evaluation of Sparse Matrix-Vector Multiplication using CSR and BCSR storage format on MPI

Akyas Khaqqi Maulana[1], Fitriyani[2]

[1]Telkom University, akyaskhaqqi@outlook.com. [2]Telkom University, fitriyani@gmail.com

Abstrak

Banyak permasalahan di dunia ini yang dimodelkan dengan matematika dalam proses penyelesaiannya, salah satunya memodelkan dalam bentuk matriks, Banyak penelitian menggunakan matriks jarang yang dihitung menjadi *Sparse Matrix-Vector Multiplication* (SpMV) dalam *benchmarking* perangkat keras mereka, dengan tujuan mendapatkan waktu optimal dalam mengeksekusi matriks tersebut, sehingga semakin cepat waktu eksekusinya, akan semakin baik kinerja dari perangkat keras mereka. Untuk mempermudah pemetaan matriks dibutuhkan format penyimpanan yang baik pula, format penyimpanan ini akan berfungsi penuh saat pemetaan matriks dan mempermudah pembacaan suatu matriks yang sudah di konversi dari matriks koordinat,penelitian ini akan menggunakan dua format penyimpanan yaitu CSR dan BCSR. , kinerja kedua format ini di evaluasi dan di jalankan pada *personal computer* dengan dua mekanisme yaitu secara serial dan parallel, pada mekanisme parallel akan dijalankan dengan protokol komunikasi MPI. Hasil pengujian menggunakan beberapa matriks dengan tipe data yang berbeda-beda dan dengan ukuran baris kolom yang beragam, masing-masing matriks di eksekusi dengan lima iterasi agar mendapatkan hasil yang optimal, masing-masing format penyimpanan menghasilkan hasil yang beragam, disebabkan oleh ukuran matriks dan sebaran data pada matriks koordinat, pada format CSR peningkatan kecepatan sebanyak 229 kali terdapat pada *thread* 3 menuju *thread* 4, pada dan pada format BCSR, peningkatan kecepatan 300 kali lebih cepat pada *thread* 4, namun penurunan kecepatan pada *thread* 3 untuk setiap matriks, dalam hal ini dapat disimpulkan setiap metode memiliki keunggulan dan kelemahan tersendiri dalam mengeksekusi SpMV. Yaitu pengaruh pada ukuran matriks dan sebaran data pada matriks tersebut.

Kata kunci: *SpMV, CSR, BCSR, MPI, thread*

BAB 1

Pendahuluan

1.1 Latar Belakang Masalah

Kemajuan teknologi yang semakin pesat menuntut kinerja dari sebuah permasalahan dengan cepat dan mudah, teknlogi berperan sangat penting dalam hal ini, permasalahan yang diselesaikan bukan hanya permasalahan sederhana lagi, permasalahan dalam dunia nyata semakin kompleks dan tidak lepas dari bantuan teknologi komputer. Salah satu permasalahan yang dihadapi adalah matriks jarang dalam analisis numerik, matriks jarang adalah matriks yang sebagian besar elemen adalah nol. Sebaliknya, jika sebagian besar elemen tidak nol, maka matriks dianggap padat. Fraksi dari nol elemen atas jumlah elemen dalam matriks disebut *sparsity* (*density*). Perkalian Matriks-Vektor Jarang (*SpM*[illegible]*A*.[illegible] Matriks SpMV masih memberikan tantangan ketika membahas tentang kinerjannya, beberapa penelitian lain telah membahas beberapa metode penyimpanan SpMV dengan berbagai arsitektur, Arin Wahyuningsih pada yang membandingkan *CSR* dan *CSV* pada *GPU* [1], Ryan Eberhardt et. al dengan optimasi SpMV pada *Intel Xeon*-Phi dan *CUDA* [2], Liu et. al dengan Kinerja *Multithreaded SpMV* pada OpenMP [3], Greathouse dan Daga yang mengembangkan kinerja pada *CSR* dengan peningkatan kemampuan pada *GPU* [4].

Penelitian ini akan lebih fokus pada pembagian blok pada format penyimpanan *Compressed Sparse Row (CSR)* & *Block Compressed Sparse Row (BCSR)* dengan menggunakan protokol komunikasi *Message Passing Interface* (MPI), pada penelitian sebelumnya dengan arsitektur yang hampir mirip yaitu OpenMP masih didapatkan keterbatasan pada kecepatan saat menjalankan matriks ukuran cukup besar dengan total *Non-zero elements* yang cukup banyak [3],

dimana pada MPI dapat dengan mudah membagi alokasi memori tiap blok yang akan dijalankan pada sebuah perangkat keras.

BAB 2

Landasan Teori

2.1 Matriks Jarang

Dalam analisis numerik, matriks jarang adalah matriks yang sebagian besar elemen adalah nol. Sebaliknya, jika sebagian besar elemen tidak nol, maka matriks dianggap padat. Fraksi dari nol elemen atas jumlah elemen dalam matriks disebut sparsity (density).[9]. Sebuah matriks jarang dijabarkan dalam bentuk format yang baru, format tersebut akan menghilangkan angka 0 yang tidak dibutuhkan dalam proses komputasi, oleh karena itu, dalam mengakses sebuah matriks jarang tidak dapat dilakukan secara langsung, optimasi tingkat lanjut yang spesifik pada sebuah arsitektur adalah hal yang sangat dibutuhkan dan sangat penting.

$$A=\begin{pmatrix} 10 & 20 & 0 & 0 & 0 \\ 0 & 30 & 0 & 40 & 0 \\ 0 & 0 & 50 & 60 & 0 \\ 0 & 0 & 0 & 0 & 80 \end{pmatrix}$$

2.2 Compressed Sparse Row

Salah satu yang paling banyak digunakan adalah metode CSR (Compressed Sparse Row). Dimana metode ini hanya menyimpan elemen non-zero pada kolom indeks matriks itu sendiri. Matriks A adalah matriks m x n , dan nilai dari elemen tidak nol adalah nz,

csrNz[*nz*]	Menyimpan nilai dari masing-masing elemen tidak nol
csrCols[*nz*]	Menyimpan indeks kolom dari masing-masing elemen pada *array csrNz*
csrRowStart[m+1]	Menyimpan indeks dari elemen tidak nol yang pertama dari masing-masing baris pada *array csrNz* dan *csrCols*, dan *csrRowStart*[m]=*nz*

Dapat dilihat pada matriks A di bawah ini,

Sebagai contoh, jika matriks $A_1 =$  dimana :

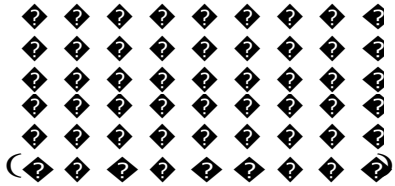
m=4,*n*=4,*nz*=6, A_1 akan disimpan sebagai berikut:

csrNz = {1,2,3,4,5,6};
csrCols ={0,3,2,2,0,3};
csrRowStart ={0,2,3,4,6};

2.3 Block Compressed Sparse Row

Sebagai pengembangan dari format CSR, BCSR membagi matriks menjadi beberapa blok, berdasarkan BCSR, blok yang sudah di register dalam algoritma ke depannya akan meningkatkan efisiensi dalam mengakses data tersebut, dengan cara memakai kembali data pada register. Blok algoritma yang sudah di register membagi matriks jarang *A*[*m*][*n*] menjadi banyak blok *r***c* yang kecil. Setelah *A* dibagi menjadi baris *m/r* dan kolom *n/c*, elemen dari masing-masing blok dihitung satu demi satu. Dalam situasi ini, akan ada elemen *c* dari vector *x* pada register dan digunakan kembali sebanyak *r* waktu untuk masing-masing blok.

bcsrNz[*nzb*r*c*]	Menyimpan nilai dari elemen pada masing-masing blok *non-zero*.
bcsrCols[*nzb*]	Menyimpan indeks kolom dari elemen pertama dari masing-masing blok *non-zero*.
bcsrRowStart[*m/r*+1]	Menyimpan indeks dari blok *non-zero* pertama pada *bcsrCols* untuk masing-masing baris blok, dan *bcsrRowStart*[*m/r*]=*nzb*.

Diberikan sebuah matriks A_2 =  sebagai:

Contoh, dimana *m*=6, *n*=9, *nz*=20, sesuai dengan format dari BCSR, maka matriks A_2 dapat ditunjukkan sebagai berikut:

bcsrNz = {1,1,1,1,0,0,2,2,2,0,2,0,3,3,0,0,3,3,4,0,4,4,4,0,5,5,0,5,5,0};
bcsrCols = {0,6,3,3,6};
bcsrRowStart = {0,2,3,5}.

2.4 Message Passing Interface (MPI)

MPI merupakan sebuah protokol komunikasi yang *sifatnya language-independent, portable* dalam mendukung berbagai platform, dan memiliki spesifikasi *semantic* yang mengatur bagaimana perilaku setiap impelementasinya. MPI mendukukung komunikasi baik dengan tipe point-to-point maupun yang bersifat kolektif.

1. MPI akan menjadi sebuah *library* untuk membangun program aplikasi dan bukan *distributed operating system*.
2. MPI akan mendukung *thread-safe* yang penting dalam *symmetric multiprocessor* pada lingkungan jaringan komputer yang heterogen.
3. MPI akan mampu untuk men-deliver *high-performance computing.*
4. MPI akan bersifat *extensible*, sehingga dapat terus dikembangkan dan memenuhi kebutuhan komputasi masa akan datang.
5. MPI akan mendukung heterogeneos komputasi.
6. MPI akan memiliki *semantic behavior* yang telah terspesifikasi dengan jelas, sehingga dapat menghindari beberapa permasalahan kritis seperti *race-conditions, dead-lock* dan sebagainya.

BAB 3

PERANCANGAN SISTEM

3.1 Deskripsi Sistem

Pada penelitian Tugas Akhir ini akan berfokus pada kinerja matriks SpMV dengan menggunakan dua metode penyimpanan yaitu CSR dan BCSR. Sistem yang dibuat merupakan algoritma yang dapat diterapkan secara parallel dan dijalankan pada arsitektur MPI melalui OpenMP, dengan Sistem Operasi Ubuntu. Akan dibandingkan hasil kinerja kedua metode penyimpanan dan akan di evaluasi lebih spesifik dalam setiap *thread* yang menjadi pembeda antara CSR dan BCSR. Pada tahap pertama adalah menjalankan SpMV dengan matriks yang sudah di dapatkan dari *The University of Florida Sparse Matrix Collection.[5]* secara serial pada CSR dan BCSR yang kemudian akan didapatkan waktu *running* dari masing-masing matriks. Tahap kedua adalah menjalankan CSR dan BCSR dengan matriks yang sama dengan program CSR dan BCSR serial secara parallel pada MPI dengan menggunakan OpenMPI, untuk CSR pembagian *thread* pada prosessor dibagi dengan tiap baris pada matriks, sedangkan BCSR akan dibagi dengan kolom-kolom kecil berisi baris dan kolom. Ukuran matriks akan disesuaikan untuk BCSR.

3.2 Data

Nama Matriks	Judul Matriks	Baris	Kolom	*Non-zeros*	Tipe	Struktur
Bccstk27,mtx	STIFFKNESS MATRIX BUCKLING PROBLEM (ANDI MERA)	1.224	1.224	56.126	real	symmetric
Pores_2.mtx	UNSYMMETRIC MATRIX FROM PORES	1.224	1.224	9.613	real	unsymmetric
Sherman4.mtx	U BLACK OIL, IMPES SIMULATION, 16 BY 23 BY 3 GRID, ONE UK	1.104	1.104	3.786	real	unsymmetric
Mcca.mtx	2D/3D problem	180	180	2.659	real	general
De080285.mtx	Problematic linear programming problem, Meszaros test set	936	1.908	5.082	real	rectangular
Filter2d.mtx	Tunable optical filter	1.668	1.668	10.750	real	symmetric
P0291.mtx	Linear programming problem, C. Meszaros test set	252	543	2.283	real	rectangular

3.3 Skenario Pengujian Sistem

Sistem akan diuji menggunakan komputer PC yang dilengkapi dengan prosessor 4 ***thread***, dengan tiap ***thread*** akan mengeksekusi masing-masing blok dari BCSR dan baris yang dibagi dari CSR. Beberapa matriks diambil secara acak dan dengan tersedianya ukuran yang beragam, masing-masing matriks akan dijalankan sebanyak lima kali pada setiap ***thread*** nya, dan akan dicari nilai rata-rata pada setiap ***thread*** yang akan disajikan dalam bentuk tabel. Waktu yang optimum akan dianalisis untuk masing-masing metode penyimpanan.

BAB 4

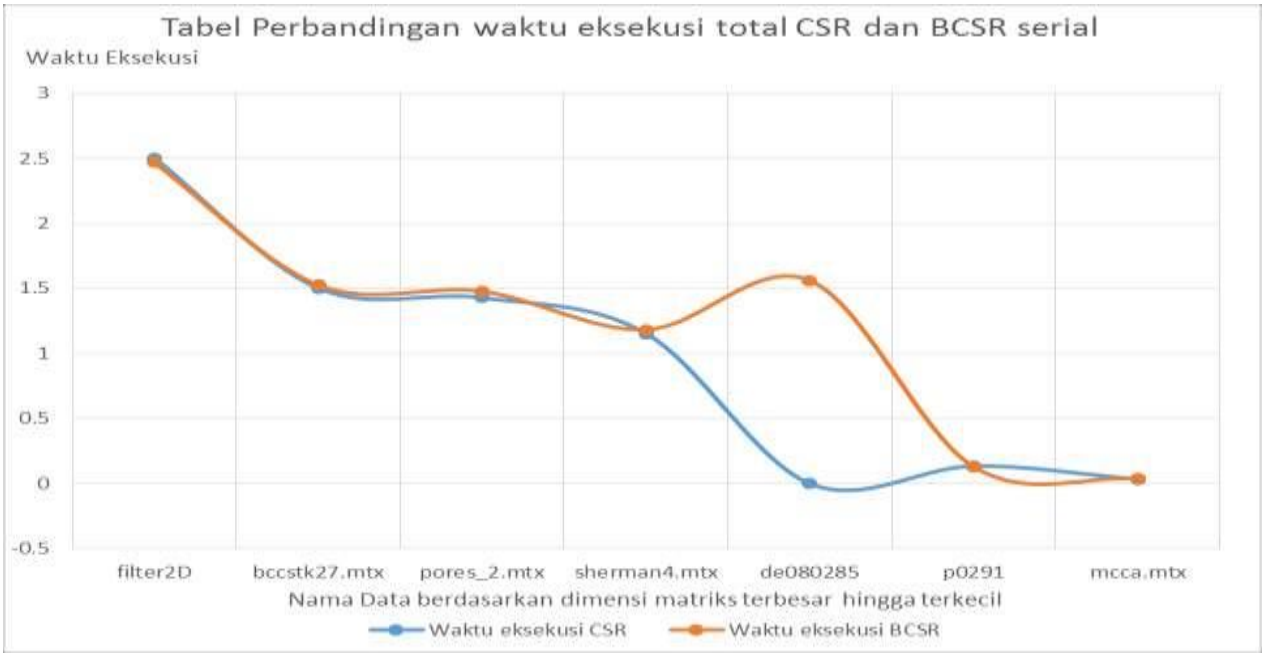
Pengujian dan Analisis

4.1 Spesifikasi Media Pengujian

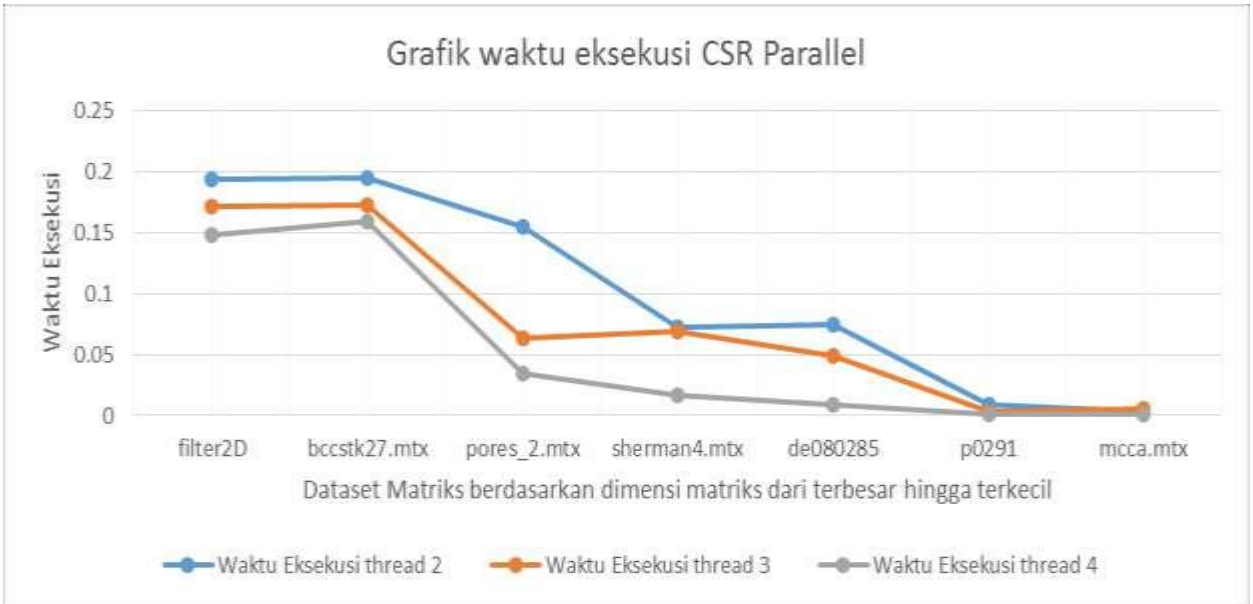
Hardware		
Prosessor	: Intel® Core™ i5 3230M CPU, Ivy Bridge @ 2.60 GHz	
RAM	: 6 GBytes DDR 3, Dual Channel	
Cores	:2	
Thread	:4	
Software		
Sistem Operasi	: Ubuntu 14.04 LTS	
Compiler	: GCC GNU 4.9	MPI 1.8

4.2 Skenario Pengujian

Pada eksekusi serial, hanya menggunakan *single core* sehingga menghasilkan waktu yang cukup lama dalam menjalankan satu matriks, berikut hasil eksekusi kedua metode yang dijalankan secara serial:



Pada grafik diatas, terlihat bahwa perbedaan eksekusi antara CSR dan BCSR secara serial tidak begitu signifikan, yang menjadi pembeda hanya matriks de080285 yang mengalami *segmental fault* atau *core dump*, hal ini terjadi karena pembagian matriks membutuhkan *thread* yang stabil dalam menjalankan program secara sequensial atau serial, dan yang digunakan adalah sebuah PC dimana memiliki *background process* yang tidak diketahui sehingga terjadi *core dump*, juga perbandingan kolom yang cukup jauh membuat komputer bekerja cukup keras, hal ini dapat diatasi dengan menjalankan algoritma pada *cluster supercomputer* yang memiliki kinerja stabil dengan konsep pembagian *job* yang terstruktur. Pada tahap eksekusi secara parallel, akan dijalankan menggunakan MPI untuk alokasi memori, proses eksekusi dilakukan lima kali untuk setiap matriks, karena waktu komputer PC tidak akan terlihat optimal dibandingkan dengan *cluster* yang mempunyai antrian yang spesifik dengan *slave* yang hanya menerima *job* dari master, selain itu tidak menjalankan apa-apa. Sedangkan komputer menjalankan *background process* yang tidak sedikit, sehingga waktu eksekusi bisa berubah-ubah setiap waktunya.

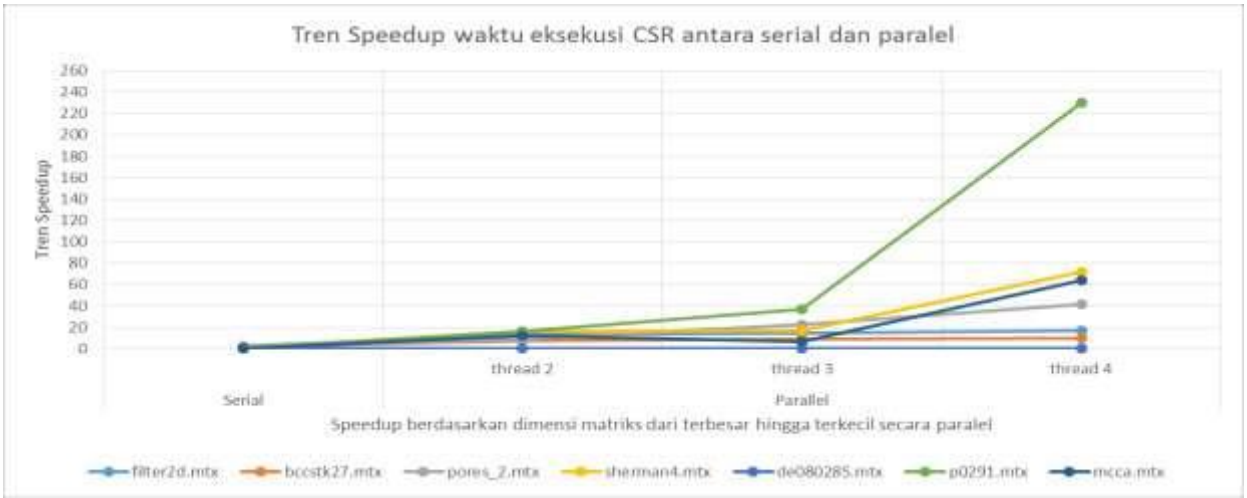


Dari kedua grafik diatas maka dapat dibandingkan kedua metode penyimpanan tersebut pada masing-masing *thread* sebagai berikut:

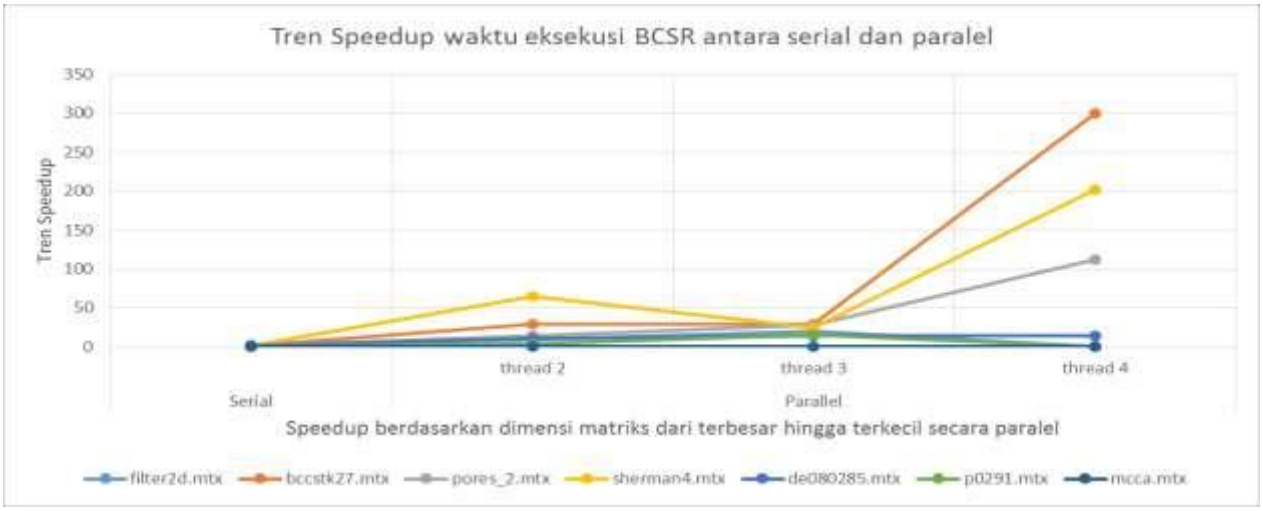
No	Nama Matriks	Thread 2		Thread 3		Thread 4	
		CSR	BCSR	CSR	BCSR	CSR	BCSR
1	filter2D	0.193815	0.262063	0.170707	0.123716	0.148357	0
2	bccstk27.mtx	0.19422	0.053164	0.172035	0.053274	0.159552	0.005084
3	pores_2.mtx	0.06382	0.104832	0.154038	0.05361	0.034172	0.013182
4	sherman4.mtx	0.07214	0.0182	0.068482	0.048656	0.016138	0.005872
5	de080285	0.074327	0.151955	0.049191	0.113481	0.009236	0.112534
6	p0291	0.008427	0.037948	0.003706	0.00871	0.000589	0
7	mcca.mtx	0.002799	0.051847	0.00502	0	0.00054	0

Dengan melihat perbandingan pada tabel tersebut, maka secara tidak langsung dapat terlihat metode CSR optimal dalam menjalankan matriks dibanding BCSR, namun secara waktu eksekusi lebih optimal menggunakan BCSR terlihat perbandingan kecepatan waktu dengan CSR. Berikut tren perbandingan waktu eksekusi antara kedua metode penyimpanan tersebut:

Untuk membandingkan hasil eksekusi masing-masing metode penyimpanan maka akan dihitung *speedup* dari kedua metode, berikut *speedup* dari metode CSR yang dibandingkan antara serial dan paralel:



Dari grafik 4-3-6, dapat diambil analisis yaitu, metode CSR dengan menggunakan arsitektur paralel MPI menghasilkan waktu eksekusi yang cukup baik dengan menggunakan matriks yang beragam, terlihat peningkatan paling signifikan pada matriks p0291, dengan peningkatan kecepatan dari *thread 3* menuju *thread* 4 sebesar 229 kali lebih cepat dibandingkan menggunakan serial, begitupun pada matriks yang lain yang mengalami peningkatan kecepatan dengan menggunakan pembagian *thread*.



Kenaikan kecepatan data juga terjadi pada format penyimpanan BCSR, namun beberapa matriks terkendala pada *thread* 3 dikarenakan tidak begitu efektif penyimpanan menggunakan 3 *thread*, namun pada *thread* 4 terjadi kenaikan yang sangat signifikan, terlihat pada matriks bccstk27, dengan kenaikan kecepatan 300 kali lebih cepat dibanding menggunakan serial.

BAB 5

Kesimpulan dan Saran

Setelah melakukan penelitian tersebut, maka dapat disimpulkan bahwa algoritma BCSR dengan membagi matriks menjadi blok-blok kecil lebih efektif dan efisien dibandingkan dengan CSR yang hanya membagi baris, algoritma BCSR sangat efektif untuk digunakan untuk menjalankan matriks jarang dengan ukuran yang sangat besar.

1. Hasil pengujian implementasi secara parallel pada MPI menunjukkan selisih perbedaan yang sangat signifikan antara algoritma BCSR dan CSR. Pada CSR di dapatkan speedup 229 kali lebih cepat daripada serial, dan pada BCSR didapatkan speedup sebanyak 300 kali lebih cepat daripada

serial, namun pada beberapa thread didapatkan hasil yang tidak optimal dikarenakan background process pada komputer dan sebaran data pada matriks koordinat yang beragam.

2. Berdasarkan pengujian, metode BCSR lebih cepat dibandingkan CSR .Namun dalam beberapa matriks yang berukuran tidak sesuai akan mempengaruhi kinerja beberapa *thread*, karena ukuran matriks tidak sesuai dengan jumlah *thread* yang tersedia pada prosesor. Sehingga ada suatu saat CSR dapat lebih cepat dibandingkan BCSR.
3. Secara umum, kedua metode dapat dijalankan pada tipe matriks apapun dan tidak mengalami kendala apapun dalam menjalankan menggunakan beberapa tipe data matriks yang berbeda-beda. Dan tipe data matriks tidak mempengaruhi hasil kinerja, *speedup* antara CSR dan BCSR.

Saran

1. Algoritma penyimpanan CSR dan BCSR tidak akan efektif apabila tidak di eksekusi pada komputer *multithread* dengan *thread* yang banyak.
2. Hasil eksekusi tidak akan optimal apabila hanya di eksekusi pada komputer biasa atau PC.
3. Tidak bisa memakai matriks bebas, apabila memakai matriks ukuran baris kolom berbeda maka harus dilakukan *preprocessing* sebelumnya.
4. Agar dikembangkan untuk dilakukan eksekusi pada arsitektur yang lain.

Daftar Pustaka

[1] Wahyuningsih, A., Fitriyani, & Nurkahfi, G. N. (2016). Implementasi Matriks Jarang Besar dengan format penyimpanan CSR dan CSV menggunakan Data Set University of Flourida. *Parallel Computing*, 1.

[2] Eberhardt, R., & Hoemmen, M. (2016). Optimization of Block Sparse Matrix-Vector Multiplication on Shared-Memory Parallel Architecture. *IEEE International Parallel and Distributed Processing Symposium Workshops*, 1-10.

[3] Liu, S., Zhang, Y., Sun, X., & Qiu, R. (2009). Performance Evaluation of Multithreaded Sparse Matrix-Vector Multiplication using OpenMP. *11th IEEE International Conference on High Performance Computing and Communications*, 1.

[4] L, Greathouse. J., & M, Daga. (2014). Efficient sparse matrixvector multiplication on GPUs using the CSR storage format. *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for, IEEE*, 769-780.

[5] Davis, T. A. (n.d.). The University Of Florida Sparse Matrix Collection. *Florida Sparse*, 1-15

[6] Barney, B., & Laboratory, L. L. (n.d.). *Message Passing Interface (MPI)*. Retrieved from LLNL Computing: https://computing.llnl.gov/tutorials/mpi/

[7] Utama, Z., Ahmad, Kurniawan, & Devi. (2009). LAPORAN PERCOBAAN IMPLEMENTASI PERHITUNGAN BILANGAN PRIMA MENGGUNAKAN MESSAGE PASSING INTERFACE (MPI). *Universitas Islam Negeri "Syarif Hidayatullah"*

[8] Akbar, A. R. (2014). *Pemrosesan Parallel*. Retrieved from Departemen Ilmu Komputer IPB: http://cs.ipb.ac.id/~auriza/parallel/pp.html#openmpi

[9] Agung Santoso, Matriks, Vektor, Skalar, Elemen Matriks dan Dimensi Matriks, link : https://sites.google.com/site/statistikuntukpsikologi/aljabar-matriks/pendahuluan

[10] T, A. (2015). *Getting + installing gcc/g++ 4.9 on Ubuntu?* Retrieved from Askubuntu: http://askubuntu.com/questions/428198/getting-installing-gcc-g-4-9-on-ubuntu

[11] Gazette, S. (2014, may 24). *Install Open MPI 1.8 in Ubuntu 14.04, 13.10*. Retrieved from Sysads Gazette: http://sysads.co.uk/2014/05/24/install-open-mpi-1-8-ubuntu-14-04-13-10/