ISSN 2355-3286

Analisis Kinerja Algoritma Quick Double Merge Sort Paralel Menggunakan openMP

I Nyoman Aditya Yudiswara[1], Abba Suganda[2]
Computer Science Department, BINUS Graduate Program Master of Computer Science
Bina Nusantara University, Jakarta, Indonesia 11480
i.yudiswara@binus.edu[1], agirsang@binus.edu[2]

Diterima 11 November 2019
Disetujui 20 Desember 2019

Abstract — Processor technology currently tends to increase the number of cores more than increasing the clock speed. This development is very useful and becomes an opportunity to improve the performance of sequential algorithms that are only done by one core. This paper discusses the sorting algorithm that is executed in parallel by several logical CPUs or cores using the openMP library. This algorithm is named QDM Sort which is a combination of sequential quick sort algorithm and double merge algorithm. This study uses a data parallelism approach to design parallel algorithms from sequential algorithms. The data used in this study are the data that have not been sorted and also the data that has been sorted is integer type which is stored in advance in a file. The parameter measured to determine the performance of the QDM Sort algorithm is speedup. In a condition where a large amount of data is above 4096 and the number of threads in QDM Sort is the same as the number of logical CPUs, the QDM Sort algorithm has a better speedup compared to the other parallel sorting algorithms discussed in this study. For small amounts of data it is still better to use sequential sorting algorithm.

Keywords : core, double merge, logical CPU, QDM sort, quick sort, speedup, thread

I. PENDAHULUAN

Mulai tahun 2004 trend dari teknologi processor lebih menambah jumlah core dari pada meningkatkan clock speed. Dalam sebuah processor terdiri dari beberapa core, dan di dalam core bisa menjalankan lebih dari satu thread. Perkembangan ini sangat bermanfaat dan menjadi sebuah kesempatan untuk meningkatkan kinerja pada sebuah algoritma sekuensial yang hanya dikerjakan oleh satu CPU atau satu core dengan cara memodifikasi algoritma tersebut agar bisa dikerjakan secara paralel oleh beberapa core. Salah satu algoritma sekuensial yang sering dipakai pada sebuah program adalah algoritma sorting. Sudah banyak algoritma sorting yang dikembangkan baik yang sekuensial maupun yang paralel, namun ruang untuk lebih meningkatkan lagi kinerja algoritma sorting khususnya algoritma sorting paralel masih terbuka lebar sejalan dengan perkembangan arsitektur dan organisasi komputer.

Paper ini membahas mengenai gabungan algoritma sekuensial sorting yang sudah ada yaitu quick sort dan merge sort yang akan dimodifikasi menjadi algoritma paralel, yang selanjutnya penulis sebut dengan nama Quick Double Merge Sort (QDM Sort). Untuk menjadikan algoritma sekuensial menjadi algoritma paralel dapat menggunakan dua pendekatan yaitu data parallelism dan control parallelism. Data parallelism adalah metode dengan membagi data menjadi beberapa blok data yang kemudian semua blok data dikerjakan secara paralel. Control parallelism adalah membagi instruksi menjadi beberapa sub-instruksi yang kemudian dijalankan secara paralel. QDM Sort menggunakan kedua pendekatan diatas agar dapat mengoptimalkan pemakian semua core yang ada pada prosesor.

Kinerja algoritma QDM Sort diukur dengan menghitung nilai speedup dan efisiensi, dengan merubah faktor faktor yang mempengaruhinya antara lain jumlah data yang diurut, jumlah thread pada program QDM Sort dan jumlah core pada prosesor komputer. Paper ini bertujuan untuk mengetahui seberapa besar peningkatan kecepatan (speedup) dan efisiensi dari algoritma QDM Sort dibandingkan dengan algoritma sorting sekuensial dan parelel yang lainnya.

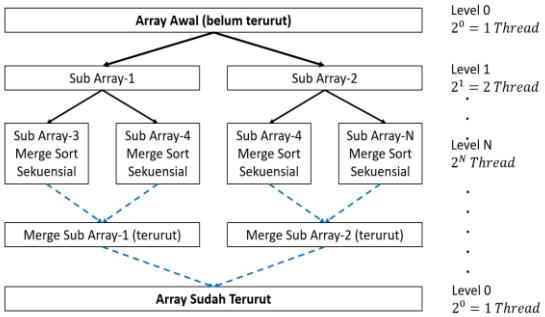
II. TINJAUAN PUSTAKA

A. Merge Sort Paralel

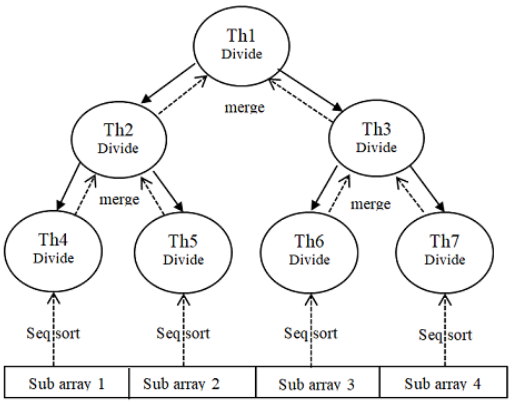
Algoritma merge sort paralel dijelaskan oleh Atanas Radenski [1] yang secara garis besar cara kerja algoritmanya seperti gambar 1. Array akan dibagi menjadi sub-sub array sampai level N dimana 2^N = jumlah thread. Jika sudah sampai pada level N setiap sub-array akan dilakukan pengurutan oleh masing-masing thread dengan menggunakan algoritma merge sort sekuensial. Setelah setiap thread selesai melakukan pengurutan kemudian dilakukan merge secara paralel oleh sejumlah thread mulai dari $2^{(N-1)}$ thread sampai terakhir hanya oleh 1 thread. Hasil penelitiannya menunjukkan nilai speedup dijalankan pada komputer dengan 2, 4 dan 8 core adalah 1.7, 2.6

dan 3.2 kali.

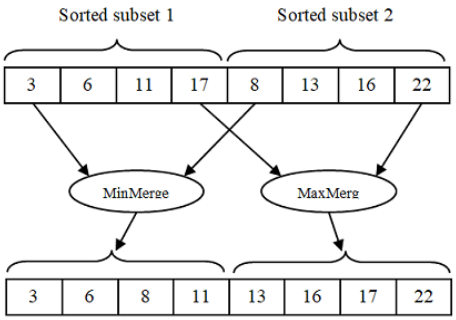
Pada penilitan Ahmet Uyar [2] menjelaskan tentang pengembangan algoritma paralel merge sort, dimana pada penelitiannya menggunakan dua thread untuk melakukan operasi merge secara bersamaan (double merge). Thread pertama akan melakukan proses merge min dan thread ke dua akan melakukan merge max seperti gambar 2 dan gambar 3. Hasilnya setelah diimplementasikan dengan menggunakan bahasa pemrograman Java peningkatan kecepatan sebanyak 20% - 30% jika dibandingkan dengan merge sort yang dimplementasikan di Java Library.



Gambar 1. Merge Sort Paralel



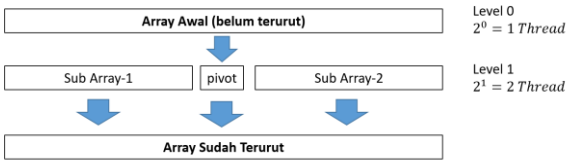
Gambar 2



Gambar 3

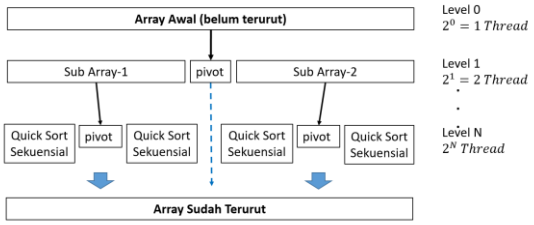
B. *Quick Sort Paralel*

Pada penilitan Yong Liu [3] dan Sinan Sameer M.[4] menjelaskan tentang pengembangan algoritma paralel quick sort. Sebenarnya kedua penelitian diatas hampir sama menggunakan maksimal dua thread seperti pada gambar 4 dibawah. Pencarian pivot pertama kali dilakukan oleh satu thread dan setelah itu masing-masing sub-array dilakukan quick sort dengan algoritma sekuensial oleh dua thread secara parallel.



Gambar 4. Paralel Quick Sort

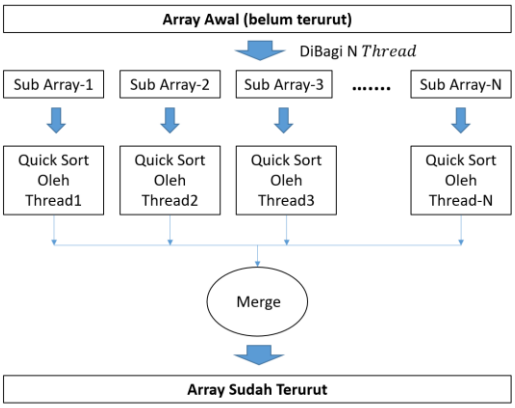
Pada penelitian Daniel Langr [5] menjelaskan algoritma quick sort paralel dengan menggunakan sejumlah thread sampai level N sehingga terdapat sebanyak 2^N sub-array yang akan di sorting secara parallel oleh masing-masing thread dengan menggunakan algoritma quick sort sekuensial, seperti gambar 5 dibawah.



Gambar 5. Quick Sort Paralel

C. *Hybrid Quick Sort*

Algoritma paralel hybrid quick sort dijelaskan oleh Kil Jae Kim [6] seperti pada gambar 6 dibawah. Mula-mula array yang belum terurut dibagi dengan jumlah thread, kemudian masing-masing sub-array ini dilakukan sorting oleh setiap thread secara bersamaan dengan algoritma quick sort sekeunsial. Sub-array yang sudah terurut ini kemudian dilakukan merge atau penggabungan maka secara keseluruhan akan diperoleh array yang sudah terurut.



Gambar 6. Quick Merge Sort

III. METODE PENELITIAN

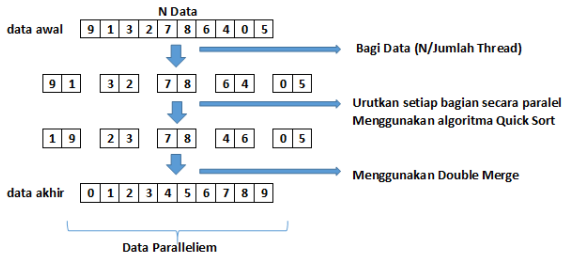
Penelitian ini dibagi menjadi beberapa tahap sebagai berikut :

- ✓ Identifikasi masalah
- ✓ Pendekatan penyelesaian masalah
- ✓ Metode yang diusulkan
- ✓ Implementasi (Coding)
- ✓ Pengukuran kinerja
- ✓ Evaluasi hasil & kesimpulan

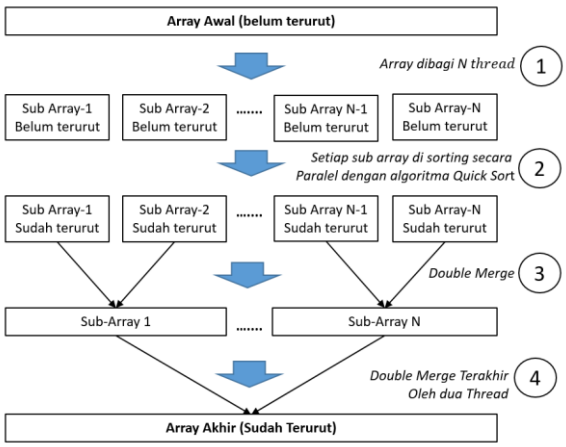
Untuk merancang algoritma paralel umumnya menggunakan pendekatan data parallelism, task parallelism atau kombinasi antara data parallelism dan task parallelism. Perancangan algoritma QDM Sort paralel menggunakan pendekatan *Data Parallelism* seperti gambar 7. Pendekatan ini pada dasarnya mencoba untuk mengoptimalkan pemakaian semua core yang ada pada prosesor komputer pada saat menjalankan program QDM Sort. Parameter yang diukur untuk mengetahui kinerja algoritma QDM Sort adalah speedup yang dihitung dengan menggunakan rumus sbb:

$$Speedup = \frac{T_S}{T_P} \quad \ldots\ldots\ldots\ldots \quad (1)$$

T_s adalah waktu eksekusi pada program dengan algoritma sekuensial dan T_p adalah waktu eksekusi pada program dengan algoritma paralel untuk menyelesaikan masalah yang sama.

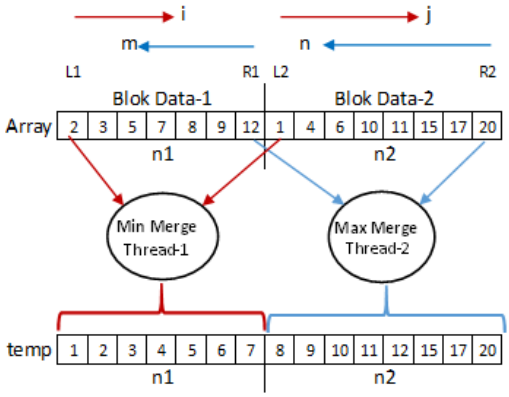


Gambar 7. Data Parallelism



Gambar 8. QDM Sort

Metode yang diusulkan untuk diimplementaskan pada QDM Sort adalah seperti gambar 8. Jika jumlah data yang akan diurut pada array adalah N, maka pada tahap awal data dibagai menjadi N/T sub-array, dimana T jumlah thread pada program QDM Sort. Kemudian setiap sub-array akan dilakukan sorting secara bersamaan oleh masing-masing thread. Setelah semua sub-array terurut kemudian dilakukan merge dengan menggunakan double merge, yaitu min-merge dan max-merge seperti gambar 9. Min Merge melakukan merge mulai dari data terkecil ke data lebih besar sebanyak n1 dan Max Merge melakukan merge mulai data terbesar ke data yang lebih kecil sebanyak n2. Min Merge dan Max Merge dilakukan secara bersamaan oleh 2 thread, sehingga pada tahap merge yang paling terakhir akan dilakukan oleh 2 thread. Jadi dengan double merge akan dilakukan tahapan merge sebanyak $\log_2(N)$, dengan jumlah thread pada tiap tahapan adalah N, N/2 sampai 2 thread. Jadi operasi merge yang terakhir akan dilakukan oleh 2 thread.



Gambar 9. Proses Double Merge

IV. IMPLEMENTASI DAN HASIL

A. *Ujicoba*

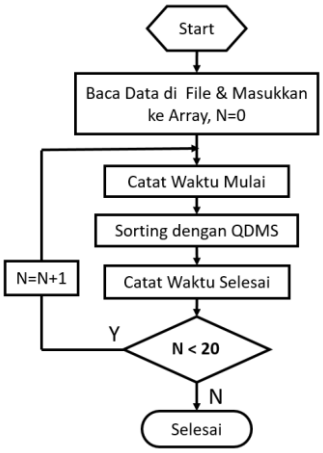
Komputer yang digunakan untuk percobaan pada penelitian ini seperti table-1 dibawah.

Tabel 1. Spesifikasi Komputer Untuk Percobaan

No	Spesifikasi Komputer	
1	Komputer	Desktop Acer Veriton M4610
	CPU	Intel i7 2600, 3.4 GHz, 4 Core 8 Thread (8 Logical CPU)
	RAM	DDR3 16 GByte PC3-12800
	Sistem Operasi	Linux Ubuntu 64-bit, 18.04
2	Komputer	All In One MSI
	CPU	Intel i3-2120, 3.3 GHz, 2 Core, 4 threads (4 Logical
	RAM	8 GByte
	Sistem Operasi	Linux Ubuntu 64-bit 18.04
3	Komputer	Laptop Dell Inspiron 1470
	CPU	Core2Duo U9400 1.4GHz, 2 Core (2 logical CPU)
	RAM	4 GByte
	Sistem Operasi	Linux Mint 19.1 Tessa

Data yang digunakan untuk mengetahui kinerja algoritma QDM Sort adalah data integer yang nilai nya random dengan jumlah data nya bervariasi mulai dari 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 49152, 65536, 131072, 262144, 393216, 524288, 786432 dan 1048576, dan data ini masing-masing disimpan pada sebuah file.

Membaca data di file dan memasukkan data ke array tidak dihitung dalam pencatatan waktu eksekusi yang diperlukan oleh program seperti gambar 10. Pencatatan run time program dilakukan beberapa kali (lebih dari 20 kali) dan dipilih waktu run time yang paling cepat / kecil. Pada saat pencatatan waktu run time tidak ada program lain yang dijalankan dan komunikasi internet/wifi juga dimatikan.



Gambar 10

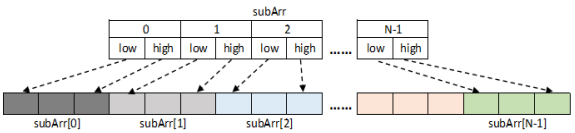
B. *Implementasi*

Program QDM Sort diimplementasikan menggunakan bahasa pemrograman C dan library openMP versi 3.1, menggunakan Integrated Development Environment (IDE) CodeBlocks versi 17.12, mengunakan sistem operasi Linux dan compiler GCC. Algoritmanya seperti potongan program dibawah ini.

- ✓ Baris 1 adalah structure untuk menyimpan index yang terendah (low) dan index tertinggi (high) dari setiap sub-array
- ✓ Baris 5 adalah nama fungsinya (QDMS) dengan parameter A, Size, T dan N. A adalah Array yang akan di sorting, Size adalah jumlah data dari Array, T adalah Array yang besarnya sama dengan A yang digunakan untuk menampung data sementara dan N adalah jumlah thread.
- ✓ Pada baris 10 dihitung jumlah elemen dari setiap sub-array, kecuali sub-array yang terakhir ukurannya bisa berbeda.
- ✓ Baris 14 - 28 adalah area paralel dengan jumlah thread ditentukan oleh variable N
- ✓ Baris 15 setiap thread akan membaca ID nya masing-masing yang penomorannya dimulai dari 0 s/d N-1
- ✓ Baris 17-20 pada sub-array terakhir dilakukan sorting dengan algoritma Quick Sort sekuensil, dan nilai index terkecil dan terbesar dari sub-array disimpan pada array struct subArray.
- ✓ Pada baris 23–27 semua thread akan melakukan sorting pada masing-masing sub-array dan sekaligus menyimpan nilai index terkecil dan indek terbesar dari sub-array pada array struct subArray.
- ✓ Pada baris 32-36 semua sub-array yang sudah diurut dengan algoritma quick sort sekuensial dilakukan merge dengan metode double merge secara parallel.

```
1  typedef struct subArray{
2    int low;  int high;
3  }SA;
4
5  void QDMS(int A[],int Size,int T[],int N)
6  {
7    int i, j, sizeSA;
8    SA *subArr;
9    subArr=(SA*)malloc(sizeof(SA)* N);
10   sizeSA=round((float)Size/(float)N);
11
12   #pragma omp parallel num_threads(N)
13   {
14     int low, high;
15     int tid = omp_get_thread_num();
16     if(tid == (N-1)){
17       low = tid*sizeSA; high=Size-1;
18       QuickSortSerial(A, low, high);
19       subArr[tid].low  = low;
20       subArr[tid].high = high;
21     }
22     else{
23       low = tid*sizeSA;
24       high = tid*sizeSA + sizeSA-1;
25       QuickSortSerial(A, low, high);
26       subArr[tid].low  = low;
27       subArr[tid].high = high;
28     }
29   }
30
31   while(N >= 2){
32    DoubleMerge(A,Size,T,subArr,N);
33    N=N/2;
34    for(j=0; j < N; j++){
35     subArr[j].low = subArr[j*2].low;
36     subArr[j].high= subArr[j*2+1].high;
37    }
38   }
39 }
40
```

Gambar 11. Algoritma QDMS dengan openMP



Gambar 12. Array of structure digunakan untuk menyimpan index low dan high pada setiap sub-array

C. *Hasil Uji Coba*

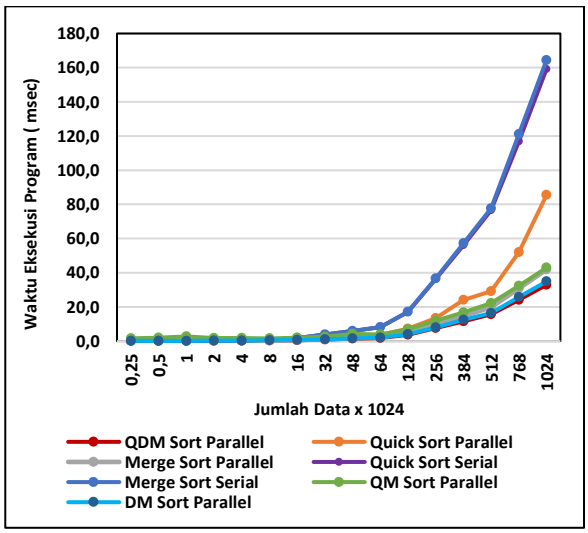
Sebagai bahan evaluasi ada 7 program sorting yang di buat dan dijalankan untuk diambil datanya yaitu program QDM sort, Quick sort paralel, Merge sort paralel, Quick sort serial, Merge sort serial, Quick

Merge sort paralel, Double merge sort paralel dan seperti yang dijelaskan pada tinjauan pustaka diatas dan yang diusulkan pada penelitian ini yaitu Quick Double Merge Sort (QDM Sort).

Table 1. Waktu eksekusi dari algoritma sorting yang digunakan pada penelitian

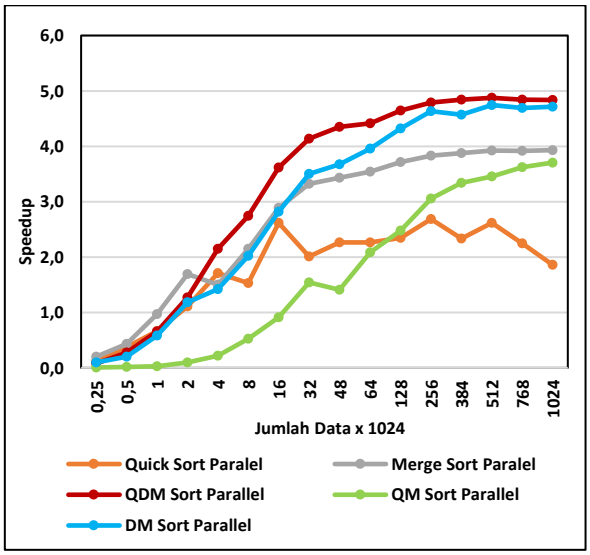
No	Jumlah Data Integer	Execution Time (mikro second)						
		QDM Sort Paralel	Quick Sort Paralel	Merge Sort Paralel	Quick Sort Serial	Merge Sort Serial	Quick Merge Sort Paralel	Double Merge Sort Parallel
1	256	137	93	79	11	16	1568.0	160.0
2	512	112	83	86	31	37	2016.0	182.0
3	1024	122	120	97	80	94	2645.0	161.0
4	2048	136	156	142	173	240	1812.0	203.0
5	4096	177	222	242	380	364	1748.0	256.0
6	8192	300	537	372	823	799	1564.0	395.0
7	16384	495	684	604	1791	1744	1960.0	618.0
8	32768	917	1889	1133	3795	3762	2458.0	1074.0
9	49152	1363	2620	1721	5934	5906	4209.0	1606.0
10	65536	1836	3583	2280	8109	8075	3885.0	2041.0
11	131072	3658	7255	4633	17000	17221	6860.0	3982.0
12	262144	7570	13495	9554	36260	36628	11857.0	7904.0
13	393216	11600	24033	14752	56150	57177	16801.0	12504.0
14	524288	15685	29186	19811	76473	77693	22147.0	16372.0
15	786432	24118	51959	30911	116827	121152	32267.0	25807.0
16	1048576	32897	85454	41829	159197	164433	42956.0	34879.0

Semua algoritma sorting pada Tabel-2 dijalankan pada komputer dengan spesifikasi seperti Tabel-1 no.1 yaitu komputer dengan 8 logical CPU. Hasil ujicoba untuk waktu eksekusi ke tujuh program tersebut ditampilkan pada Tabel-2 dan juga ditampilkan dalam bentuk grafik seperti gambar 13 dan gambar 14.



Gambar 13. Waktu eksekusi vs jumlah data

Pada gambar 13 menunjukkan waktu eksekusi versus jumlah data untuk beberapa algoritma sorting, dimana pada program sorting serial hanya menggunakan 1 logical CPU sedangkan algoritma sorting paralel menggunakan 8 logical CPU. Berdasarkan hasil pada Tabel-2 dan gambar 13 program Quick Double Merge Sort (QDM Sort) untuk jumlah data diatas 16 x 1024 memerlukan waktu yang lebih cepat dibandingkan dengan program sorting lainnya.



Gambar 14. Speedup vs jumlah data

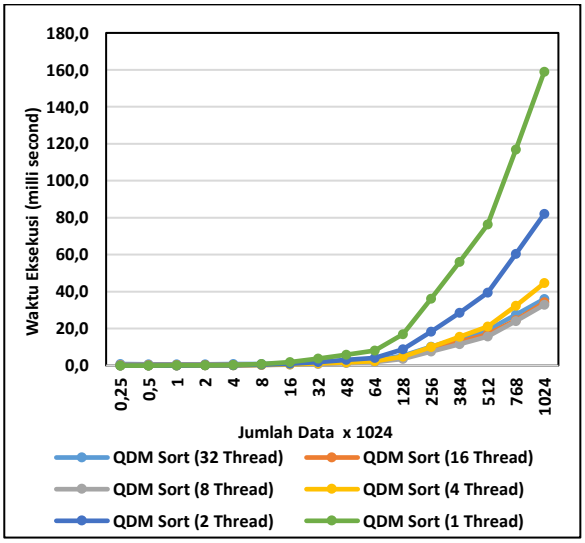
Pada gambar 14 menunjukkan besarnya nilai pertambahan kecepatan (speedup) versus jumlah data untuk semua program sorting paralel yang digunakan pada penelitian ini. Speedup dihitung dengan menggunakan rumus (1) diatas. Untuk menghitung speedup QDM sort paralel, Quick sort paralel dan Quick merge sort paralel nilai T_s diambil dari waktu eksekusi quick sort serial. Sedangkan untuk menghitung speedup double merge sort paralel dan merge sort paralel nilai T_s diambil dari nilai waktu eksekusi merge sort serial. Berdasarkan Tabel-2 dan gambar 14 untuk jumlah data diatas 4 x 1024, QDM

Sort memiliki nilai speedup yang lebih besar dibandingkan dengan program lainnya.

Tabel-3 berisi waktu eksekusi program QDM sort dengan jumlah thread yang berbeda-beda yang dijalankan pada komputer dengan spesifikasi seperti Tabel-1 no.1.

Table 3. Waktu eksekusi program QDM Sort untuk jumlah thread yang berbeda

No.	Jumlah Data x 1024	Waktu eksekusi (milli second)					
		QDM Sort (32 Thread)	QDM Sort (16 Thread)	QDM Sort (8 Thread)	QDM Sort (4 Thread)	QDM Sort (2 Thread)	QDM Sort (1 Thread)
1	0,25	0,8	0,364	0,137	0,024	0,007	0,011
2	0,5	0,678	0,32	0,112	0,028	0,014	0,031
3	1	0,719	0,322	0,122	0,047	0,033	0,079
4	2	0,685	0,333	0,136	0,083	0,085	0,173
5	4	0,755	0,401	0,177	0,158	0,193	0,38
6	8	0,901	0,528	0,3	0,303	0,428	0,823
7	16	1,2	0,766	0,495	0,617	0,925	1,791
8	32	1,698	1,295	0,917	1,134	1,956	3,795
9	48	2,256	1,846	1,363	1,708	3,028	5,934
10	64	2,91	2,448	1,836	2,325	4,131	8,109
11	128	5,4	4,326	3,658	4,818	8,775	16,997
12	256	10,145	9,173	7,57	10,012	18,381	36,221
13	384	14,676	12,863	11,6	15,531	28,553	56,09
14	512	19,134	16,976	15,685	21,077	39,542	76,42
15	768	27,439	24,605	24,118	32,331	60,471	116,827
16	1024	36,028	34,177	32,897	44,59	82,066	159,015

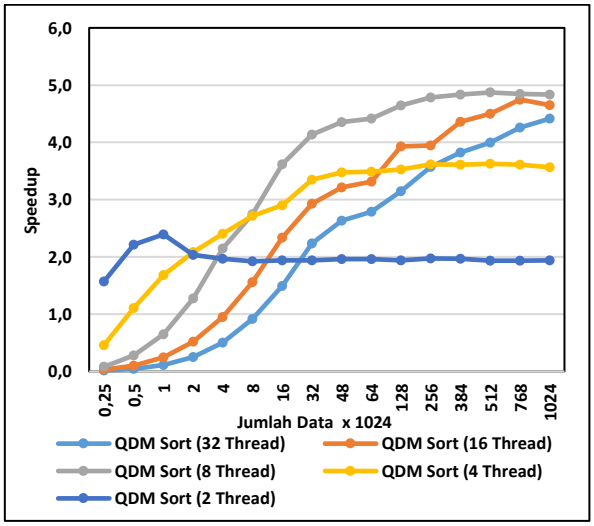


Gambar 15. Waktu Eksekusi vs Jumlah Data vs Jumlah Thread

Representasi dalam bentuk grafik untuk tabel-3 dapat dilihat pada gambar 15 dan 16. Gambar 15 menunjukan pengaruh jumlah data dan jumlah thread pada program QDM sort terhadap waktu eksekusi. Dari Table-3 dan gambar 15 dapat dilihat untuk jumlah data yang besarnya diatas 8 x 1024 dan untuk program QDM sort yang memiliki jumlah thread sama dengan jumlah logical CPU nya memiliki waktu eksekusi lebih cepat. Karena program QDM sort ini dijalankan pada komputer dengan 8 logical CPU maka program QDM sort dengan 8 thread memiliki waktu eksekusi paling cepat dibandingkan dengan jumlah thread yang lainnya.

Gambar 16 menunjukkan pengaruh jumlah data dan jumlah thread pada program QDM sort terhadap speedup. Speedup dihitung dengan menggunakan rumus (1) diatas. Untuk menghitung speedup QDM sort dengan jumlah thread 2, 4, 8, 16 dan 32 nilai T_s diambil dari waktu eksekusi QDM sort untuk 1 thread.

Pada gambar 16 terlihat untuk data yang besarnya diatas 8 x 1024 dan untuk program QDM Sort yang memiliki jumlah thread sama dengan jumlah logical CPU akan memiliki speedup yang yang paling besar. Program QDM sort dengan 8 thread untuk jumlah data lebih besar dari 8x1024 memiliki speedup paling tinggi.



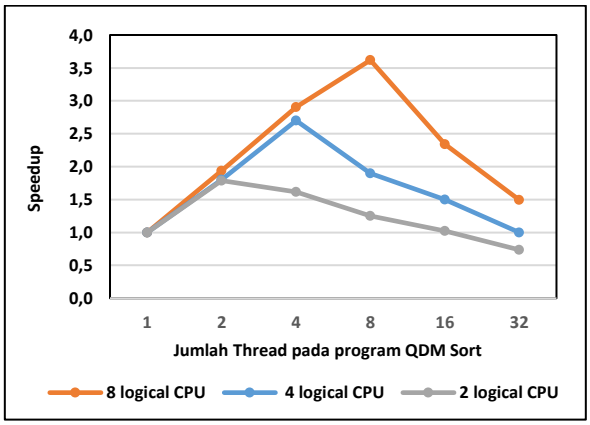
Gambar 16. Speedup vs Jumlah Data vs Jumlah Thread

Tabel-4 berisi data hasil perhitungan speedup dan efisiensi program QDM sort dengan jumlah thread yang berbeda-beda dan dengan jumlah data yang di urut

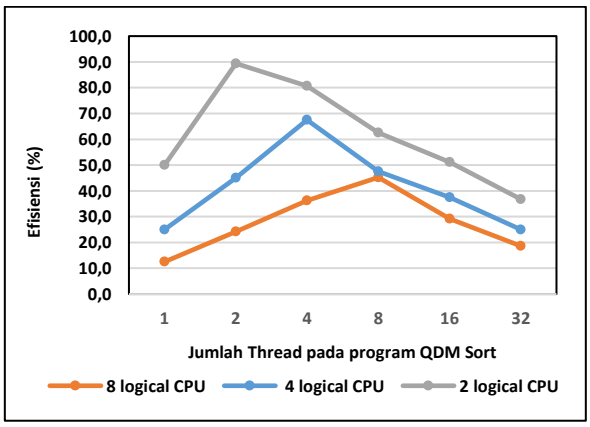
sebesar 16x1024, dan dijalankan pada tiga buah komputer yang berbeda dengan spesifikasi seperti pada tabel-1. Representasi dalam bentuk grafik untuk Tabel-4 dapat dilihat pada gambar 17 dan 18. Kedua gambar ini menunjukkan speedup dan efisiensi yang optimal didapat jika jumlah data yang diurut cukup besar diatas 16384 dan jumlah thread pada program QDM sort sama dengan jumlah logical CPU pada komputer.

Tabel 4 – Speedup dan Efisiensi program QDM sort pada 3 komputer yang beda

No	Jumlah Thread	Komputer dengan 8 logical CPU		Komputer dengan 4 logical CPU		Komputer dengan 2 logical CPU	
		Speedup	Efisiensi	Speedup	Efisiensi	Speedup	Efisiensi
1	1	1.000	12.500	1.000	25.000	1.000	50.000
2	2	1.936	24.203	1.800	45.000	1.788	89.394
3	4	2.903	36.284	2.700	67.500	1.614	80.724
4	8	3.618	45.227	1.900	47.500	1.251	62.547
5	16	2.338	29.227	1.500	37.500	1.023	51.164
6	32	1.493	18.656	1.000	25.000	0.735	36.770



Gambar 17 – Speedup vs Jumlah Thread pada program vs Jumlah logical CPU



Gambar 18 – Efisiensi vs Jumlah Thread pada program vs Jumlah logical CPU

V. KESIMPULAN

A. *Kesimpulan*

Dari hasil ujicoba diatas dapat disimpulkan sbb :

1. QDM Sort memiliki waktu eksekusi yang lebih cepat dibandingkan dengan algoritma Sorting paralel lainnya yang digunakan sebagai referensi pada penelitian ini, untuk jumlah data yang besar diatas diatas 4096. Sedangkan untuk jumlah data yang kecil lebih baik menggunakan algoritma sorting sekuensial.
2. QDM Sort memiliki speedup yang optimal jika jumlah data yang besar diatas 4096 dan jumlah thread-nya sama dengan jumlah logical CPU yang dimiliki oleh prosesor komputer.
3. Untuk mendapatkan speedup yang optimal, program QDM Sort secara otomatis akan menyesuaikan jumlah threadnya sama dengan jumlah logical CPU, dan jika datanya kecil (dibawah 4096) program QDM sort secara otomatis akan menggunakan algoritma quick sort sekuensial.

B. *Saran*

Program QDM sort ini masih bisa ditingkatkan speedupnya jika pada tahapan proses merge nya di setiap tahapnya dapat dilakukan oleh semua logical CPU yang dimiliki oleh prosesor. QDM sort baru bisa menggunakan logical CPU pada tahapan merge mulai dari N, N/2, N/4 dan pada tahap terakhir dilakukan oleh 2 logical CPU (N=Jumlah logical CPU). Disarankan agar penelitian ini bisa dilanjutkan dengan memanfaatkan semua resources (logical CPU) di setiap tahap dari tahapan proses merge.

REFERENSI

[1] Atanas Radenski, Shared Memory, Message Passing, and Hybrid Merge Sorts for Standalone and Clustered SMPs. Proc. PDPTA'11, the 2011 International Conference on Parallel and Distributed Processing Techniques and Applications, CSREA Press (H. Arabnia, Ed.), 2011, pp. 367 - 373.

[2] Uyar, A., Parallel Merge Sort with Double Merging. IEEE 8th International Conference on Application of Information and Communication Technologies (AICT), 2014

[3] Yong Liu, Y. Y. (2013). Quick-MergeSort Algorithm Based on Multi-coreLinux. International Conference on Mechatronic

[4] Sinan Sameer Mahmood Al-Dabbagh, N. H., Parallel QuickSort Algorithm using OpenMP. International Journal of Computer Science & Mobile Computing (*IJCSMCC)*, Vol.5 Issue.6, June- 2016, pg. 372-381.

[5] Daniel Langr, Pavel Tvrdik and Ivan Simecek, AQsort : Scalable Multi-Array In-Place Sorting With OpenMP. Scalable Computing: Practice And Experience, Volume 17, Number 4, 2016, pp. 369–391. http://www.scpe.org

[6] Kil Jae Kim, Seong Jin Cho and Jae-Wook Jeon, Parallel Quick Sort Algorithms Analysis using OpenMP 3.0 in Embedded

System. 11th International Conference on Control, Automation and Systems Oct. 26-29, 2011 in KINTEX, Gyeonggi-do, Korea

[7] Sciences, Electric Engineering and Computer (MEC), 2013, 1578 - 1583.

[8] Task Parallelism vs Data Parallelism [online] Tersedia dalam : https://www.allprogrammingtutorials.com/tutorials/task-parallelism-vs-data-parallelism.php [diakses 1-Nop-2019]