Pembangunan Aplikasi Manajemen Data Statistik Berbasis Komputasi Paralel

Muhammad Miftakhul Romadlon
BPS Kabupaten Bangka Tengah
Jl. Raya Bypass Koba, Komplek Perkantoran Pemkab Bangka Tengah, (0718) 7362085
e-mail: akhulromadlon@gmail.com

Abstrak

Penelitian ini dilatarbelakangi oleh belum terpenuhinya kebutuhan aplikasi manajemen data statistik yang efisien dan user friendly. Manajemen data dibutuhkan untuk menyiapkan dan mengolah data sebelum masuk pada tahap analisis. Aplikasi manajemen data statistik dibangun dengan memanfaatkan multicore processor. Sumber daya komputer ini mendukung dilakukannya komputasi paralel, yaitu membagi proses komputasi untuk dikerjakan oleh beberapa sumber daya komputer secara bersamaan. Aplikasi dibangun sebagai R Graphical User Interface (GUI) dengan pendekatan plugin. Hal ini bertujuan untuk memudahkan developer lain yang ingin mengintegrasikan modul statistik. Dari hasil uji coba, dapat disimpulkan bahwa proses manajemen data pada aplikasi yang dibangun telah menerapkan komputasi paralel sehingga performanya lebih optimal. Setiap penambahan jumlah core, akan meningkatkan performa proses manajemen data. Pada data set dengan dua juta observasi, kecepatan tertinggi dihasilkan dari proses duplicate count. Perbandingan kecepatan yang dihasilkan sebesar 1,659 kali lebih cepat dengan menggunakan dua core dan 2,107 kali lebih cepat dengan menggunakan empat core.

***Kata kunci**: komputasi paralel, multicore processor, manajemen data, aplikasi statistik, R GUI*

1. Pendahuluan

Informasi dari hasil analisis data merupakan salah satu bahan pertimbangan penting untuk pengambilan keputusan, baik bagi individu maupun organisasi, seperti pemerintah dan swasta. Metode pengolahan dan penganalisisan data merupakan salah satu faktor penentu kualitas informasi sehingga penggunaan aplikasi yang sesuai turut mempengaruhi. Aplikasi yang memiliki performa optimal, mudah digunakan, serta dapat melakukan beragam uji statistik adalah jenis aplikasi yang dijadikan prioritas oleh pengguna.

R adalah salah satu aplikasi statistik yang dewasa ini banyak dimanfaatkan dalam pengolahan dan penganalisisan data. R merupakan aplikasi statistik yang *powerful*, *open source* dan *multiplatform* [1]. Namun demikian, tampilan R yang menggunakan *command language interpreter* (CLI) atau biasa dikenal dengan *command line interface* terkadang menjadi kendala bagi beberapa pengguna. Perbedaan kemampuan dan persepsi dari setiap pengguna menjadi salah satu alasan adanya fenomena tersebut [2]. *Graphical user interface* (GUI) bisa digunakan sebagai alternatif antarmuka bagi pengguna yang kurang menyukai CLI. GUI mampu memfasilitasi pengguna untuk bisa melakukan manipulasi antarmuka dengan memanfaatkan *mouse* dan *keyboard*.

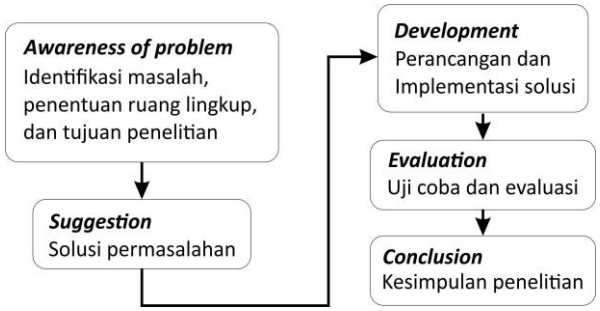
Manajemen data merupakan bagian penting dari aplikasi statistik. Sebelum masuk ke tahap analisis, data perlu disiapkan dan diolah terlebih dahulu yang dapat dikategorikan menjadi tiga, yaitu *preparation*, *handling*, dan *transformation*. Proses manajemen data pada beberapa aplikasi statistik, termasuk R GUI yang sudah ada masih dikerjakan secara *sequential* (berbasis komputasi serial), seperti R Commander, Deducer, dan RKWard. Padahal, jika dikerjakan secara paralel, proses tersebut bisa diselesaikan dengan lebih cepat. Salah satu alternatif yang bisa digunakan untuk mengerjakan proses secara paralel adalah dengan memanfaatkan *multicore processor*.

Menurut Xiaowen, *Multicore processor* telah menjadi arah pengembangan teknologi *processor* di masa depan [3]. Seiring perkembangan teknologi, saat ini telah banyak perangkat komputer yang dilengkapi dengan *multicore processor*. Sumber daya komputer tersebut mendukung dilakukannya komputasi paralel, yaitu membagi satu atau lebih permasalahan dan menyelesaikannya secara bersamaan.

Berdasarkan permasalahan di atas, penelitian ini bertujuan untuk membangun aplikasi manajemen data statistik yang efisien dan *user friendly*. *Multicore processor* dimanfaatkan supaya performanya lebih optimal. Selain bertujuan untuk meningkatkan performa, aplikasi dibangun dengan pendekatan *plugin*. Dengan cara ini, berbagai macam modul yang sesuai dapat diintegrasikan dengan mudah.

2. Metode Penelitian

Metode penelitian yang digunakan dalam penelitian ini adalah *design research*. Metode ini digunakan untuk menemukan suatu masalah dan menyelesaikannya menggunakan artefak sebagai solusinya. Dalam penelitian ini, artefak yang dimaksud adalah aplikasi manajemen data statistik yang akan dibangun. Tahapan metode penelitian dapat dilihat pada gambar 1.

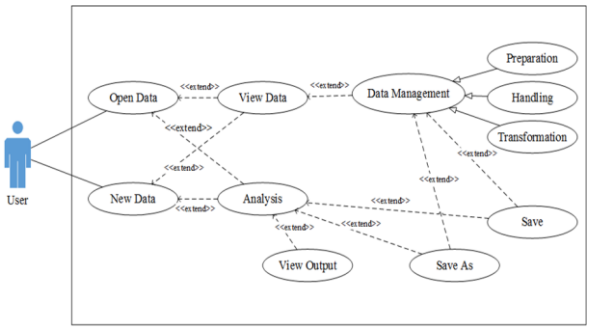


Gambar 1. Tahapan penelitian *design research*

2.1. Rancangan Sistem Usulan

Untuk mencapai tujuan penelitian, selain melakukan analisis dan menampilkan hasilnya, aplikasi sebaiknya memiliki *data editor*. *Data editor* digunakan untuk melihat dan memanipulasi data dengan lebih mudah sebelum memasuki tahap analisis. Selain itu, aplikasi juga harus bisa membuka dan menyimpan data dengan berbagai format.

Rancangan sistem usulan berdasarkan kebutuhan tersebut dimodelkan menggunakan *use case diagram*. Diagram ini digunakan untuk menunjukkan interaksi antara pengguna dan aplikasi. Aktor dalam *use case* ini hanya satu, yaitu pengguna yang diberi nama *user*. Tingkatan pengguna aplikasi disamakan sehingga tidak ada perbedaan hak akses antar aktor. *User* bisa membuka data yang telah dimiliki atau membuat data baru. Data yang telah dibuka pada aplikasi bisa ditampilkan menggunakan *data editor* dan *variable editor*. Setelah ditampilkan, data bisa diolah dan disiapkan dengan menu *data management*, yang terdiri dari *preparation*, *handling*, dan *transformation*. Proses analisis bisa dilakukan dengan atau tanpa proses manajemen data terlebih dahulu. Apabila data telah siap untuk dianalisis, proses analisis bisa dilakukan dan hasilnya akan ditampilkan di *output*. Kemudian pengguna bisa menyimpan data dan *output* hasil analisis yang telah dikerjakan.



Gambar 2. *Use case diagram* proses bisnis sistem usulan

2.2. Rancangan Struktur Data

Berbagai format data yang dibuka aplikasi akan diubah menjadi objek R, yaitu *data frame*. Isi *data frame* akan dimasukkan ke dalam *container* C++ terlebih dahulu karena objek R tidak bisa diproses secara paralel. Dari berbagai macam jenis *container* yang ada, *vector* digunakan karena objek R bisa dikonversi dengan mudah ke dalam *vector* [4]. Selain itu, *vector* dipilih karena sesuai dengan kebutuhan pembangunan program dan memiliki beberapa kelebihan dibandingkan dengan *container* yang lain. Josuttis menjelaskan beberapa kelebihan *vector*, antara lain:

1. *Vector* adalah *sequence container* yang mirip dengan *array*, tetapi ukurannya bisa diubah secara dinamis. Oleh karena itu, *vector* juga biasa disebut dengan *array dinamis*.
2. *Vector* memungkinkan akses acak ke elemen di dalamnya dengan waktu yang konstan.
3. *Vector* menghasilkan performa yang bagus apabila menambah atau menghapus elemen di indeks terakhir. [5]

Data yang akan diolah bisa berupa angka atau huruf. Oleh karena itu, tipe data yang ditampung di dalam *vector* adalah *struct*. *Struct* memungkinkan untuk menyimpan data dengan beberapa tipe. Dengan kelebihan ini, *vector* bisa digunakan untuk menampung data berupa angka ataupun huruf.

2.3. Rancangan Algoritma

Sesuai dengan pemaparan sebelumnya, proses manajemen data akan dilakukan secara paralel. Meski demikian, proses manajemen data secara serial juga tetap dilakukan untuk mendapatkan hasil perbandingan kedua proses tersebut. Pada tahap ini, algoritma paralel akan dibuat berdasarkan algoritma serial. Namun, tidak semua manajemen data dapat dijalankan secara paralel. Faktor-faktor yang

menyebabkan hal tersebut adalah proses komputasinya tidak terlalu kompleks, data yang digunakan hanya sedikit, dan adanya proses yang tidak bisa dijalankan secara paralel. Oleh karena itu, manajemen data yang akan dikerjakan secara paralel antara lain: *sort cases, find, identify duplicate cases, duplicate count, merging data by column, merging data by row, aggregate data, select cases, computing variables, random number generator, recoding same and different variable,* dan *rank cases*.

Algoritma dari manajemen data secara umum dapat dituliskan sebagai berikut:

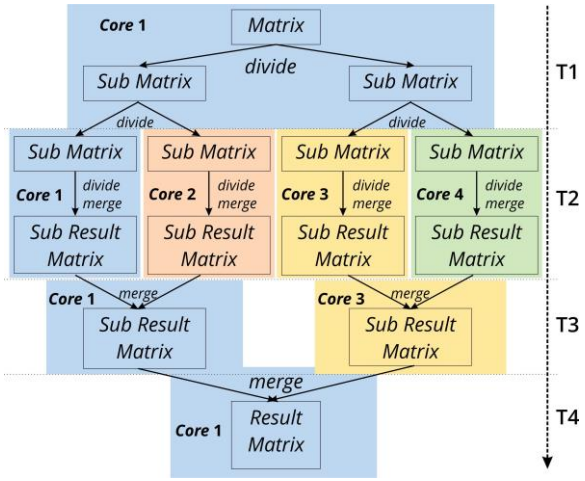
1. Pengguna memasukkan *input* yang dibutuhkan melalui *dialog*.
2. Mengeksekusi sebuah fungsi dari manajemen data.
3. Menampilkan hasil proses manajemen data pada *data editor*.

Perbedaan algoritma serial dengan algoritma paralel terdapat pada langkah kedua. Pada algoritma paralel, proses manajemen data dilakukan secara bersamaan menggunakan jumlah *core* yang telah ditentukan. Misalnya, data yang akan diproses memiliki satu juta observasi dan lima variabel. Apabila jumlah *core* yang digunakan sebanyak empat, setiap *core* bertanggung jawab menyelesaikan 250.000 observasi dan lima variabel.

Proses pengurutan (*sorting*) banyak digunakan di dalam proses manajemen data. Oleh karena itu, penjelasan rancangan algoritma lebih difokuskan ke algoritma *sort cases*. Setiap proses manajemen data yang melakukan pengurutan akan menghasilkan indeks baris yang telah diurutkan. Indeks ini disimpan dan akan digunakan apabila ada proses manajemen data lain yang melakukan pengurutan pada variabel yang pernah diurutkan sebelumnya. Dengan cara ini, kombinasi beberapa proses manajemen data tersebut bisa dipercepat.

Ada berbagai macam algorima yang bisa digunakan untuk sorting, seperti *quick, merge, shell, insertion, selection, bubble, counting, gnome,* dan *cocktail*. Algoritma yang terbaik bisa dilihat berdasarkan kecepatan, memori yang digunakan, kestabilan, dan kompleksitas. Berdasarkan hasil penelitian Rao dan Ramesh, *merge* dan *quick* merupakan algoritma yang memiliki kecepatan lebih tinggi dibandingkan dengan algoritma lainnya. Namun, *merge* lebih stabil dan pada kondisi terburuk lebih cepat dibandingkan dengan *quick* [6]. Oleh karena itu, algoritma yang digunakan adalah *merge sort*.

Rancangan algoritma *merge sort* secara paralel diilustrasikan pada gambar 3. Langkah pertama adalah membagi *matrix* berdasarkan baris sesuai dengan jumlah *core*. Garis panah di sisi kanan gambar menunjukkan perbedaan penggunaan *core* berdasarkan waktunya. Pembagian *matrix* pada saat T1 hanya menggunakan satu *core*. Proses paralel yang sesungguhnya dilakukan pada saat T2 untuk mengerjakan langkah selanjutnya, yaitu membagi *matrix* menjadi dua bagian secara berulang sampai setiap bagian hanya memiliki satu baris. Setelah itu, setiap dua *sub result matrix* digabungkan (*merge*) sampai tersisa dua *sub result matrix*. Dikarenakan hanya dua matrix yang tersisa, hanya dua *core* yang bisa digunakan pada saat T3. Kemudian penggabungan yang terakhir (T4) dikerjakan hanya menggunakan satu *core* saja untuk menghasilkan *matrix* yang utuh kembali dan telah diurutkan berdasarkan *input*. *Input* yang dibutuhkan adalah satu atau beberapa variabel dan tipe pengurutannya (*order*), yaitu *ascending* atau *descending*. Indeks variabel dan tipe pengurutannya dimasukan ke dalam *list.*



Gambar 3. Ilustrasi algoritma *merge sort* secara paralel

Kompleksitas algoritma dapat memengaruhi proses komputasi paralel. Perbandingan kompleksitas semua rancangan algoritma manajemen data dapat dilihat di tabel 1. Berikut ini adalah fungsi kompleksitas algoritma untuk proses *sort cases* menggunakan notasi Big-O:

O(paralel) = O(persiapan) + O(*merge sort* paralel) + O(*merge* sisanya)

$$= O(\frac{n}{c}(v+1)) + O(\frac{n}{c}\log\frac{n}{c}) + O(4n(1-\frac{1}{c}))$$

$$f(\text{paralel}) = O(\frac{n}{c}\,[\,v+1+\log\frac{n}{c}+4(c-1)])$$

Keterangan: c = jumlah *core*, v = jumlah variabel, n = jumlah observasi pada *data set*

Tabel 1. Perbandingan kompleksitas rancangan algoritma manajemen data

No.	Manajemen Data	Kompleksitas Algoritma
(1)	(2)	(3)
1.	*Sort cases*	$O(\frac{n}{c}[v+1+\log\frac{n}{c}+4(c-1)])$
2.	*Find*	$O(\frac{n}{c}[v+1+\log\frac{n}{c}+4(c-1)]+\log\frac{n}{c})$
3.	*Identify duplicate cases*	$O(\frac{n}{c}[v+2+\log\frac{n}{c}+4(c-1)])$
4.	*Duplicate count*	$O(\frac{n}{c}[v+2+\log\frac{n}{c}+4(c-1)])$
5.	*Merging data by column*	$O(\frac{n}{c}[v+1+\log\frac{n}{c}+4(c-1)]+\log\frac{n}{c})$
6.	*Merging data by row*	$O(n\frac{c+1}{c})$

No.	Manajemen Data	Kompleksitas Algoritma
(1)	(2)	(3)
7.	*Aggregate data*	$O(\frac{n}{c}[v+2+\log\frac{n}{c}+4(c-1)])$
8.	*Select cases*	$O(\frac{n}{c}[v+1+\log\frac{n}{c}+4(c-1)]+\log\frac{n}{c})$
9.	*Computing variables*	$O(n\frac{c+1}{c})$
10.	*Random number generator*	$O(n\frac{c+1}{c})$
11.	*Recoding variable*	$O(\frac{n}{c}[v+1+\log\frac{n}{c}+4(c-1)]+\log\frac{n}{c})$
12.	*Rank cases*	$O(n\frac{c+1}{c})$

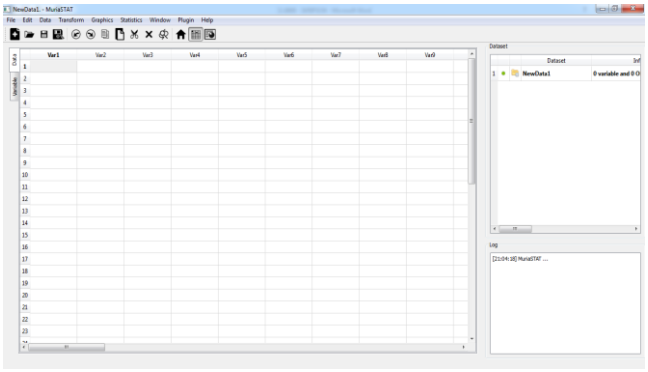
3. Hasil dan Pembahasan

3.1. Implementasi Sistem Usulan

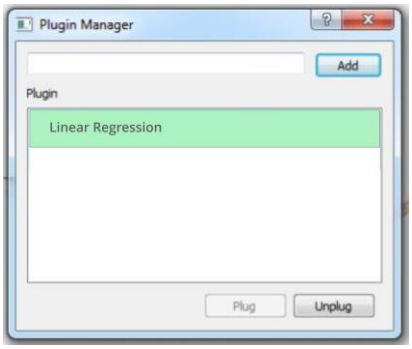
Hasil rancangan pada bab sebelumnya telah diimplementasikan menggunakan bahasa pemrograman C++ dan *framework* Qt. Keduanya digunakan untuk membagun antarmuka aplikasi dengan R sebagai *back end.* Ada dua *package* R yang digunakan untuk menghubungkan C++ dengan R, yaitu 'Rcpp' dan 'RInside'. RInside menyediakan C++ *wrapper class* yang memudahkan *programmer* untuk menggunakan R di dalam aplikasi C++ dengan cepat [7]. *Package* lain yang digunakan untuk membuka dan menyimpan data adalah 'Foreign'. Dengan *package* ini aplikasi bisa mengolah data dengan format *comma separated values* (.csv), *text file* (.txt), SPSS (.sav), SAS (.xpt), dan STATA (.dta).

Manajemen data yang telah diimplementasikan dibagi menjadi dua, yaitu manajemen data yang melakukan manipulasi secara langsung di *data editor* dan menggunakan *dialog*. Manajemen data kelompok pertama yang telah diimplementasikan adalah *copy*, *paste*, *cut*, *delete*, *undo*, dan *redo*. Sedangkan manajemen data dengan *dialog* antara lain *sort cases, find, identiy duplicate cases, duplicate count, merging data by column, merging data by row, aggregate data, select cases, computing variables, random number generator, recoding same and different variable,* dan *rank cases*. Untuk manajemen data kelompok ini, proses komputasinya dibangun berbasis komputasi paralel.

Hasil implementasi antarmuka aplikasi yang dibagun dapat dilihat pada gambar 4. Gambar tersebut menunjukkan tampilan *menu*, *toolbar*, *data set*, *log*, dan *data editor*. Selain *data editor*, aplikasi memiliki tampilan *variable editor* dan *output*. Aplikasi juga sudah dilengkapi dengan menu *plugin manager* sehingga modul lain dapat diintegrasikan dengan lebih mudah. Gambar 5 menampilkan contoh modul *linear regression* yang telah diintegrasikan ke dalam aplikasi manajemen data statistik.



Gambar 4. Implementasi antarmuka aplikasi manajemen data statistik



Gambar 5. Implementasi menu *plugin manager*

3.2. Implementasi Algoritma

Sesuai dengan rancangan algoritma pada bab sebelumnya, implementasi algoritma paralel dibangun dari algoritma serial. Perbedaan keduanya adalah pada algoritma paralel ada pembagian dan penggabungan kembali proses manajemen data. Hasil pembagian proses tersebut akan dikerjakan oleh beberapa *core* secara bersamaan. Gambar 6 menunjukkan kode program untuk membagi proses komputasi menggunakan OpenMP. OpenMP menyediakan fungsi *section* untuk mengalokasikan setiap kode di dalamnya ke *core* yang tersedia. Setiap *section* mengeksekusi *method* "mergeSortStruct()" dengan batas indeks yang telah dibagi sebelumnya. *Method* tersebut berfungsi untuk melakukan *sorting* dengan pendekatan *merge*.

```
double t = omp_get_wtime();
#pragma omp parallel sections
{
    #pragma omp section
    {
        mergeSortStruct(VM1,temp1,low1,high1,cols,order, types);
    }
    #pragma omp section
    {
        mergeSortStruct(VM1,temp1,low2,high2,cols,order, types);
    }
    #pragma omp section
    {
        mergeSortStruct(VM1,temp1,low3,high3,cols,order, types);
    }
    #pragma omp section
    {
        mergeSortStruct(VM1,temp1,low4,high4,cols,order, types);
    }
}
```

Gambar 6. Contoh kode program proses komputasi paralel menggunakan *section*

3.3. Hasil Pengujian

Salah satu hasil penelitian ini adalah waktu yang dibutuhkan untuk eksekusi proses manajemen data pada program berbasis serial dan paralel. Pada saat uji coba, diasumsikan tidak ada program lain yang berjalan selain *startup program* yang diperlukan sistem operasi. *Data set* yang digunakan dalam uji coba adalah data hasil bangkitan dari R. Hasil uji coba tersebut akan dibandingkan dan dianalisis.

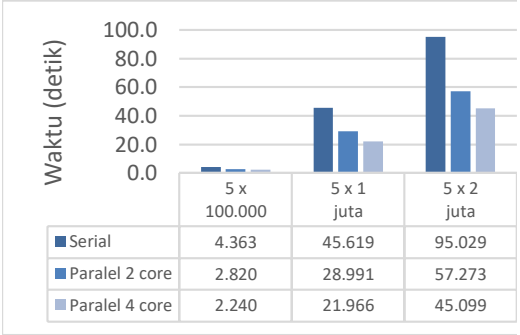
Tabel 2. Perbandingan performa *sort cases* berbasis serial, paralel dua *core*, dan paralel empat *core*

Data Set	**Jenis Komputasi**	***Merge Sort* (detik)**	***Reload* (detik)**	**Waktu (detik)**	**Kecepatan**	**Efisiensi**
(1)	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**
5 x 100.000	Serial	1,028	0,056	1,084	1	1
	Paralel 2 *core*	0,689	0,056	0,746	1,454	0,727
	Paralel 4 *core*	0,558	0,056	0,614	1,764	0,441
5 x satu juta	Serial	11,886	0,406	12,291	1	1
	Paralel 2 *core*	7,660	0,406	8,066	1,524	0,762
	Paralel 4 *core*	5,844	0,406	6,250	1,967	0,492
5 x dua juta	Serial	24,800	0,820	25,620	1	1
	Paralel 2 *core*	15,912	0,820	16,733	1,531	0,766
	Paralel 4 *core*	11,868	0,820	12,688	2,019	0,505

Pengukuran waktu pada percobaan ini menggunakan fungsi yang disediakan oleh OpenMP, yaitu "omp_get_wtime()". Fungsi dijalankan di awal dan akhir proses komputasi untuk menghitung waktu yang dibutuhkan pada algoritma serial dan paralel. Pada algoritma paralel, persiapan alokasi ke setiap *core* dan penggabungan hasil komputasi dari semua *core* juga dihitung.

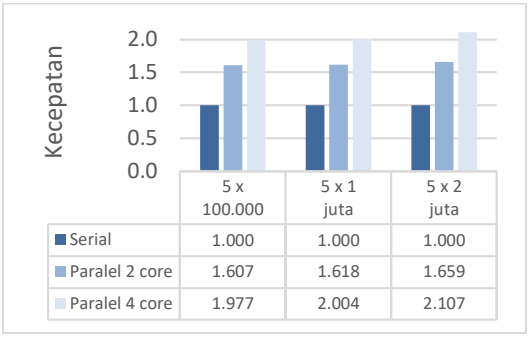
Performa proses komputasi berbasis serial, paralel dua *core*, dan paralel empat *core* dibandingkan berdasarkan waktu yang dibutuhkan untuk mengeksekusi proses manajemen data. Selain itu, kecepatan (*speed up*) dan efisiensi juga dibandingkan untuk mengetahui pengaruh penambahan jumlah *core*. Perbandingan performa proses komputasi *sort cases* dapat dilihat pada tabel 2.

Proses yang menghasilkan performa paling tinggi ditampilkan pada gambar 7 sampai 9. Pada *data set* dengan dua juta observasi, selisih waktu eksekusi terbesar jika dibandingkan dengan serial dihasilkan oleh proses *find*, yaitu 37,756 detik lebih cepat dengan menggunakan dua *core* dan 49,930 detik lebih cepat dengan menggunakan empat *core*. Sedangkan

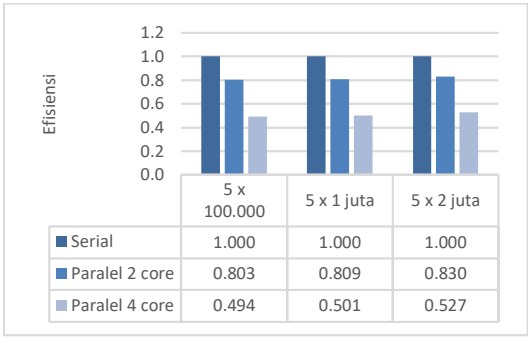


Gambar 7. Perbandingan waktu eksekusi proses *find*

kecepatan dan efisiensi tertinggi dihasilkan dari proses *duplicate count*. Dengan menggunakan dua *core*, komputasi paralel bisa menghasilkan 1,659 kali kecepatan komputasi serial dan 2,107 kali lipat dengan menggunakan empat *core*. Efisiensi yang dihasilkan menunjukkan penggunaan dua *core* memanfaatkan 83 persen dari kedua *core* tersebut dan penggunaan empat *core* memanfaatkan 52,7 persen dari setiap *core*.



Gambar 8. Perbandingan kecepatan proses *duplicate count*



Gambar 9. Perbandingan efisiensi proses *duplicate count*

Berdasarkan hasil uji coba di atas, perbandingan performa dapat disimpulkan sebagai berikut:

1. Penambahan jumlah *core* pada seluruh manajemen data menghasilkan kecepatan yang lebih tinggi. Kecepatan yang paling tinggi diperoleh menggunakan empat *core*.
2. Terjadi peningkatan kecepatan pada komputasi paralel seiring dengan penambahan jumlah *data set*. Peningkatan tersebut membuktikan bahwa jumlah *data set* berpengaruh pada kompleksitas dan performa proses komputasi paralel.
3. Berbeda dengan kecepatan, efisiensi yang dihasilkan menggunakan empat *core* lebih rendah dibandingkan dengan dua *core*. Penurunan efisiensi ini munjukan bahwa semua *core* yang digunakan pada proses komputasi paralel belum dimanfaatkan dengan optimal. Akan tetapi, hal tersebut lebih baik dibandingkan dengan adanya *core* yang tidak dimanfaatkan sama sekali.

4. Simpulan

Berdasarkan penelitian yang telah dilakukan, dapat disimpulkan bahwa aplikasi manajemen data statistik yang dibangun telah menerapkan komputasi paralel sehingga performanya lebih optimal. Penambahan jumlah *core* sebanding dengan peningkatan performanya. Pada data set dengan dua juta observasi, kecepatan tertinggi dihasilkan dari proses duplicate count. Perbandingan kecepatan yang dihasilkan sebesar 1,659 kali lebih cepat dengan menggunakan dua core dan 2,107 kali lebih cepat dengan menggunakan empat core. Aplikasi juga dibangun dengan pendekatan *plugin* sehingga modul lain bisa diintegrasikan dengan mudah.

Untuk pengembangan selanjutnya, aplikasi diharapkan bisa mengolah data yang lebih besar. Salah satu caranya adalah dengan menggunakan *database*. Selain itu, perbaikan proses komputasi paralel juga perlu dikaji lebih lanjut supaya hasilnya bisa mencapai *linear speedup*.

Daftar Pustaka

[1] Pramana S, dkk. Dasar-Dasar Statistika dengan Software R Konsep dan Aplikasi. Bogor: In Media. 2016. 1.
[2] Lawrence M, Verzani J. Programming Graphical User Interfaces in R. Boca Raton: CRC Press. 2012. xiii-xiv.
[3] Xiaowen L. Research on Multi-Core PC Parallel Computation Based on OpenMP. *International Journal of Multimedia and Ubiquitous Engineering*. 2014. 9(7): 131-140.
[4] Eddelbuettel D. Seamless R and C++ integration with Rcpp. New York: Springer. 2013. 45 dan 201-203.
[5] Josuttis N M. The C++ standard library: a tutorial and reference. Edisi kedua. Upper Saddle River, NJ, USA: Addison-Wesley. 2012. 167-168.
[6] Rao D, Ramesh B. Experimental Based Selection of Best Sorting Algorithm. *Journal of Modern Engineering Research*. 2012. 2(4): 2908-2912.
[7] Eddelbuettel D, François R. RInside: C++ Classes to Embed R in C++ Applications. R package version 0.2.13. 2015. 2.