IMPLEMENTASI *PARALLEL COMPUTING* UNTUK MENGOPTIMALKAN KOMPUTASI PADA APLIKASI TRANSLITERASI HURUF LATIN KE AKSARA JAWA

SKRIPSI

Oleh:

ANIS NURUL HIDAYAH
NIM. 11650093

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2018

IMPLEMENTASI *PARALLEL COMPUTING* UNTUK MENGOPTIMALKAN KOMPUTASI PADA APLIKASI TRANSLITERASI HURUF LATIN KE AKSARA JAWA

SKRIPSI

Diajukan kepada:
Universitas Islam Negeri Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh:

ANIS NURUL HIDAYAH
NIM. 11650093

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2018

HALAMAN PERSETUJUAN

IMPLEMENTASI *PARALLEL COMPUTING* UNTUK MENGOPTIMALKAN KOMPUTASI PADA APLIKASI TRANSLITERASI HURUF LATIN KE AKSARA JAWA

SKRIPSI

Oleh:
ANIS NURUL HIDAYAH
NIM. 11650093

Telah Diperiksa dan Disetujui untuk Diuji
Tanggal: 2017

Dosen Pembimbing I	Dosen Pembimbing II
A'la Syauqi, M.Kom	Fatchurrohchman, M. Kom
NIP. 19771201 200801 1 007	NIP. 19700731 200501 1 002

Mengetahui,
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

HALAMAN PENGESAHAN

IMPLEMENTASI PARALLEL COMPUTING UNTUK MENGOPTIMALKAN KOMPUTASI PADA APLIKASI TRANSLITERASI HURUF LATIN KE AKSARA JAWA

SKRIPSI

Oleh :

ANIS NURUL HIDAYAH
NIM. 11650093

Telah Dipertahankan di Depan Dewan Penguji Skripsi dan Dinyatakan Diterima sebagai Salah Satu Persyaratan Untuk Memperoleh Gelar Sarjana Komputer Strata Satu (S.Kom)

Tanggal, Januari 2018

Susunan Dewan Penguji :		Tanda Tangan
Penguji Utama	: Ainatul Mardhiyah, M.CS NIDT. 19860330 20160801 2 075	( )
Ketua Penguji	: Linda Salma Angreani NIP. 19770803 200912 2 005	( )
Sekretaris Penguji	: A'la Syauqi, M.Kom NIP. 19771201 200801 1 007	( )
Anggota Penguji	: Fatchurrochman, M.Kom NIP. 19700731 200501 1 002	( )

Mengetahui dan Mengesahkan
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

HALAMAN PERNYATAAN ORISINALITAS

Saya yang bertanda tangan dibawah ini:

Nama : Anis Nurul Hidayah

NIM : 11650093

Jurusan : Teknik Informatika

Fakultas : Sains dan Teknologi

Judul : Implementasi Parallel Computing untuk Mengoptimalkan Komputasi pada Aplikasi Transliterasi Huruf Latin ke Aksara Jawa

Menyatakan dengan sebenar – benarnya bahwa skripsi yang saya tulis ini tidak terdapat unsur – unsur penjiplakan karya penelitian atau karya ilmiah yang pernah dilakukan atau dibuat oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka. Apabila ternyata hasil penelitian ini terbukti terdapat unsur – unsur penjiplakan, maka saya bersedia untuk mempertanggungjawabkan, serta diproses sesuai peraturan yang berlaku.

Malang, 19 Desember 2017

Yang membuat pernyataan

Anis Nurul Hidayah

NIM. 11650093

HALAMAN PERSEMBAHAN

Karya ini saya persembahkan kepada
Kedua orang tua saya Bapak Sutarto dan Ibuk Mistiyarani yang telah membesarkan saya dengan penuh kasih sayang. Terima kasih atas kesabaran dan kasih saying yang terus mengalir tiada henti. Anis sayang Bapak dan Ibuk.

Adik-adik yang saya sayangi, Ipul, Anip, dan Faqih. Terima kasih telah menjadi pelipur lara bagi kakakmu ini, maaf karena Mbak Anis belum bisa menjadi panutan yang baik bagi kalian.

Pak Cahyo, selaku dosen wali. Terima kasih karena telah membimbing dan selalu memberi motivasi selama saya menjadi mahasiswa.

Pak Syauqi dan Pak Fatchur selaku dosen pembimbing skripsi. Terima kasih atas semua waktu yang diberikan disela-sela mengajar dan kesibukan bapak lainnya. Terima kasih atas kesabaran bapak dalam membimbing saya untuk menyelesaikan skripsi ini.

Guru-guru saya yang telah memberikan ilmunya kepada saya, semoga Bapak dan Ibu guru selalu diberi perlindungan oleh Allah SWT. Terima kasih atas semua ilmu yang Bapak Ibu Guru berikan kepada saya.

Pelatih dan teman-teman UKM Pagar Nusa UIN Malang, Mas Amin, Mbak Novi, Mbak Mala, Mas Zain, Mbak Ana, Mas Yasin, Mas Nuris, Mas Rochim, Mas Zaid, Mas Rhendra, Mas Taufiq, Mas Khori, Joko, Irham, Najib, Wafa, Ery, Mirna, Ami dan semua teman-teman yang tidak bisa saya sebutkan satu-persatu, bersama kalian segalanya selalu menjadi hal yang menyenangkan.

Sahabat-sahabat saya, Ulfa, Fauzan, Hafid, Ulfa Rosy (STN), Hizbun, Tum, Mbak Rima, Mbak Lachah, dan semua teman-teman yang tidak bisa saya sebutkan satu persatu., terima kasih atas semua dukungan dan perhatian yang kalian berikan, semoga hal-hal baik selalu menyertai kalian.

Para pembaca yang budiman.

MOTTO

إِنَّ مَعَ الْعُسْرِ يُسْرًا

Sesungguhnya bersama kesulitan ada kemudahan.(Al Insyirah/94:6)

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakatuh

Alhamdulillah segala puji dan syukur kehadirat Allah SWT atas berkah, rahmat serta hidayah-Nya, sholawat serta salam penulis haturkan kepada baginda Rosulullah SAW sebagai pembawa rahmat bagi seluruh alam ini yang senantiasa dilimpahkan kepada penulis, sehingga dapat menyelesaikan skripsi ini dengan judul "Implementasi Parallel Computing untuk Mengoptimalkan Komputasi pada Aplikasi Transliterasi Huruf Latin ke Aksara Jawa"

Dalam menyelesaikan skripsi ini banyak hambatan dan rintangan yang penulis hadapi namun pada akhirnya dapat terlampaui dengan adanya bimbingan, dukungan dan bantuan dari berbagai pihak baik secara moral maupun spiritual. Untuk itu pada kesempatan ini penulis menyampaikan terimakasih kepada:

1. Prof. Dr. Abdul Haris, M.Ag selaku Rektor Universitas Islam Negeri Maulana Malik Ibrahim Malang
2. Dr. Sri Harini, M.Si selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Malang
3. Dr. Cahyo Crysdian selaku Ketua Jurusan Teknik Informatika dan Dosen Wali yang senantiasa memberikan dorongan semangat, arahan dan motivasi.
4. A'la Syauqi, M.Kom selaku pembimbing I, yang telah meluangkan begitu banyak waktu, dan dengan penuh kesabarannya membimbing dan mengarahkan penulis dalam penyusunan skripsi ini.

5. Fatchurrohman, M. Kom selaku Dosen pembimbing II yang juga telah begitu banyak meluangkan waktu dan pemikirannya untuk membimbing penulis dalam menyelesaikan skripsi ini.
6. Segenap Dosen Teknik Informatika yang telah memberikan bimbingan keilmuan dengan begitu sabar selama masa studi penulis.
7. Bapak, Ibu, adik-adik dan segenap keluarga besar penulis di Sragen dan Solo yang telah banyak memberikan do'a, motivasi serta dukungan dalam penyelesaian skripsi ini.
8. Teman – teman seperjuangan teknik informatika yang telah membantu dan memberi dukungan dalam pengerjaan skripsi ini.
9. Teman-teman penulis di UKM Pagar Nusa yang sudah seperti keluarga bagi penulis.
10. Semua pihak, yang tidak dapat penulis sebutkan satu persatu, yang telah membantu dan memberikan inspirasi dalam pengerjaan skripsi ini.

Penulis menyadari bahwa skripsi ini masih jauh dari sempurna. Dengan segenap kerendahan hati, penulis mengharapkan kritik dan saran yang membangun penulisan skripsi ini agar dapat memberikan manfaat khususnya bagi penulis sendiri dan umumnya kepada seluruh pembaca skripsi ini. Semoga penulisan skripsi ini bermanfaat bagi pembaca sekalian

Wassalamu'alaikum Warahmatullahi Wabarakatuh.

Malang, 19 Desember 2017

Anis Nurul Hidayah

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Hidayah, Anis Nurul. 2018. Implementasi *Parallel Computing* Untuk Mengoptimalkan Komputasi Pada Aplikasi Transliterasi Huruf Latin ke Aksara Jawa. Skripsi. Jurusan Teknik Informatika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang.
Pembimbing: (I) A'la Syauqi, M.Kom dan (II) Fatchurrochman, M.Kom

Kata Kunci: Aksara Jawa, Transliterasi, *Parallel Computing*, Optimalisasi, Komputasi, *Multithread*.

Transliterasi, menurut Kamus Besar Bahasa Indonesia, adalah penyalinan dengan penggantian huruf dari abjad yang satu ke abjad yang lain. Aksara Jawa merupakan salah satu warisan leluhur bangsa Indonesia yang harus dilestarikan. Dari penelitian yang dikerjakan oleh Ayumitha (2014) diperoleh hasil bahwa dengan menggunakan metode *Decision Tree* tingkat akurasi kebenaran diatas 90%. Namun semakin banyak teks yang diinputkan akan semakin banyak pula waktu yang dibutuhkan untuk menyelesaikan transliterasi. Pada penelitian ini diajukan solusi untuk mengurangi waktu komputasi menggunakan *Parallel Computing*. *Parallel computing* merupakan model pemrograman yang menggunakan beberapa sumber komputer untuk mengerjakan suatu permasalah komputasi secara simultan. Salah satu model premrograman paralel adalah *Thread*. Model pemrograman ini adalah salah satu tipe dari *shared memory programming*. Dalam model pemrograman paralel *thread*, suatu proses yang dikategorikan berat dapat dibagi menjadi beberapa proses ringan, yang dieksekusi bersamaan. Penelitian ini menggunakan *Multithread* untuk melakukan proses transliterasi. Hasil dari penerapan *multithread* pada penelitian ini berhasil mengurangi waktu komputasi rata-rata hingga 97.41%.

ABSTRACT

Hidayah, Anis Nurul. 2018. Implementation of Parallel Computing To Optimize Computing On Transliteration Applications Latin Letters to Javanese Letters. Essay. Department of Informatics, Faculty of Science and Technology, State Islamic University Maulana Malik Ibrahim Malang.
Supervisor: (I) A'la Syauqi, M.Kom and (II) Fatchurrochman, M. Kom

Keywords: Javanese script, Transliteration, Parallel Computing, Optimization, Computing, Multithread.

Transliteration, according to Big Indonesian Dictionary, is a copy by substitution of letters from one alphabet to another. Javanese script is one of the ancestral heritage of Indonesia that must be preserved. From research done by Ayumitha (2014) obtained result that by using method of Decision Tree accuracy level of truth is above 90%. But the more text input will be the more time it takes to complete the transliteration. This study proposed solutions to reduce computing time using Parallel Computing. Parallel computing is a programming model that uses several computer sources to work on a computing problem simultaneously. One of the parallel programming models is Thread. This programming model is one type of shared memory programming. In the thread parallel programming model, a process that is categorized as heavy can have several light processes, which are executed concurrently. This research uses Multithread to perform the process of transliteration simultaneously. The result of multithread implementation in this study succeeded in reducing the computation time on average up to 97.41%.

ملخص

هداية، أنيس نورول. 2018. تنفيذ الحوسبة المتوازية لتحسين الحوسبة في تطبيقات التحويل الحروف اللاتينية إلى الحروف الجاوية. أطروحة. قسم المعلوماتية، كلية العلوم والتكنولوجيا، جامعة الدولة الإسلامية مولانا مالك إبراهيم مالانج

مستشار: (I) أعلى شوقي الماجستير، و (II) فتح الرحمان الماجستير.

كلمات البحث: النصي الجاوية، التحويل، الحوسبة المتوازية، التحسين، الحوسبة، مولتيثريد (Multithread).

التحويل ، وفقا لقاموس كبير إندونيسيان، هو نسخة عن طريق استبدال الحروف من الأبجدية إلى أخرى. النص الجاوي هو واحد من التراث السلفي من اندونيسيا التي يجب الحفاظ عليها. من البحوث التي أجرتها أيوميتا (٢٠١٤) حصلت على نتيجة أن باستخدام طريقة دقة القرار شجرة مستوى من الصحة هو فوق ٩٠٪. ولكن المزيد من إدخال النص سيكون المزيد من الوقت الذي يحتج لإكمال التحويل. في هذا البحوث اقترح حلول للحد من الوقت الحوسبة باستخدام الحوسبة المتوازية .الحوسبة المتوازية هي نموذج البرمجة الذي يستخدم العديد من مصادر الكمبيوتر للعمل على مشكلة الحوسبة في وقت واحد. أحد نماذج البرمجة المتوازية هو Thread.هذا النموذج البرمجة هو نوع واحد من Shared Memory Programming. في نموذج البرمجة المتوازي thread ، يمكن أن العملية التي تصنف على أنها ثقيلة عدة عمليات خفيفة، التي يتم تنفيذها في وقت واحد. يستخدم هذا البحث مولتيثريد (multithread) لتنفيذ عملية التحويل. نتيجة التنفيذ في هذا البحوث يحصل انقص وقت الحسوبة في المتوسط إلى 97.41٪.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Transliterasi, menurut Kamus Besar Bahasa Indonesia, adalah penyalinan dengan penggantian huruf dari abjad yang satu ke abjad yang lain. Transliterasi bisa dilakukan dari pengalihan aksara hijaiyah ke aksara latin (Murti, 2011), dari aksara Latin ke aksara Jawa (Utami, 2013) dan/ atau sebaliknya (Atina, 2012).

Aksara Jawa merupakan salah satu warisan leluhur bangsa Indonesia yang harus dilestarikan. Aksara Jawa kini semakin ditinggalkan, karena telah diperkenalkannya tulisan Latin sebagai standar Internasional (Atina, 2012). Banyak dari masyarakat suku Jawa sendiri yang tidak bisa menulis menggunakan Aksara Jawa, bahkan tidak jarang yang tidak bisa berbahasa Jawa. Oleh karena itu diajukanlah penelitian tentang transliterasi aksara Latin ke aksara Jawa sebagai salah satu bentuk pelestarian budaya yang dimiliki oleh bangsa Indonesia.

Penelitian tentang pembangunan aplikasi transliterasi aksara Jawa telah banyak dikerjakan dengan berbagai macam metode, antara lain *Finite State Automata* (Atina, 2012), *Rule Based* (Utami, 2013), dan *Decision Tree* (Ayumitha, 2014). Dari penelitian yang dikerjakan oleh Ayumitha (2014) diperoleh hasil bahwa dengan menggunakan metode *Decision Tree* tingkat akurasi kebenaran yang dihasilkan adalah yang paling tinggi dari pada aplikasi lainnya. Namun demikian, waktu komputasi yang dibutuhkan oleh aplikasi ini juga paling tinggi dibandingkan aplikasi lainnya. Hal ini berbanding lurus dengan banyaknya teks yang ditransliterasikan, sehingga semakin banyak teks

yang diinputkan akan semakin banyak pula waktu yang dibutuhkan untuk menyelesaikan transliterasi.

Secara tradisional *software* ditulis secara serial. Program dipecah menjadi beberapa instruksi, kemudian instruksi-instruksi tersebut dieksekusi secara sekuensial (berurutan), pekerjaan satu harus diselesaikan kemudian baru pekerjaan lainnya dieksekusi. Jenis komputasi ini disebut dengan komputasi serial, atau *serial computing.* Pada komputasi serial, program dieksekusi pada satu prosesor (*single processor*) dan hanya akan mengeksekusi satu instruksi pada satu waktu. Berbeda halnya dengan komputasi paralel yang merupakan kebalikan dari komputasi serial.

Secara sederhana, komputasi paralel atau *parallel computing* dapat diartikan dengan penggunaan beberapa sumber computer secara simultan atau bersamaan untuk menyelesaikan permasalahan komputasi. Langkah-langkah dari *parallel computing* adalah (Baeney, 2014):

1. Masalah dipecah menjadi bagian-bagian diskrit yang dapat diselesaikan secara bersamaan.
2. Tiap-tiap bagian selanjutnya dipecah menjadi serangkaian instruksi.
3. Instruksi dari tiap-tiap bagian mengeksekusi secara bersamaan pada prosesor yang berbeda.
4. Digunakan sebuah mekanisme kontrol / koordinasi secara keseluruhan.

Di dalam Al-Qur’an disebutkan betapa waktu adalah sesuatu yang sangat penting, seperti disebutkan dalam Al-Qur’an Surat Al-Asr/103 ayat 1-3.

وَالْعَصْرِ (١) إِنَّ الْإِنْسَانَ لَفِي خُسْرٍ (٢) إِلَّا الَّذِينَ آمَنُوا وَعَمِلُوا الصَّالِحَاتِ وَتَوَاصَوْا بِالْحَقِّ وَتَوَاصَوْا بِالصَّبْرِ (٣)

Artinya: Demi masa. Sesungguhnya manusia itu benar-benar berada dalam kerugian, kecuali orag-orang yang beriman dan beramal saleh dan saling menasehati agar menaati kebaikan dan kesabaran.(Q.S. Al-Asr/103:1-3)

Menurut Ibnu Katsir, *Al-Asr* berarti zaman atau masa yang dengannya manusia atau Bani Adam bergerak melakukan perbuatan baik dan buruk. Dalam surat ini Allah SWT. bersumpah dengan menyebutkan kata '*Al-Asr*' yang berarti masa, zaman atau waktu. Hal ini menunjukkan bahwa betapa pentingnya waktu dalam kehidupan manusia, karena Allah tidak bersumpah akan sesuatu di dalam Al-Qur'an melainkan untuk menunjukkan kelebihan yang dimiliki oleh sesuatu tersebut.

Oleh karena itu, perlu dibuat solusi untuk mengoptimalkan waktu pemrosesan pada aplikasi tersebut. Pada penelitian ini diajukan sebuah solusi untuk mengatasi masalah tersebut, yaitu dengan menerapkan *Parallel Computing* pada aplikasi yang telah dibangun. Pada penelitian ini *parallel computing* diterapkan dalam pemrosesan transliterasi. Teks akan dibagi menjadi beberapa bagian, lalu teks yang telah dibagi akan diproses secara bersama-sama / simultan sehingga waktu komputasi yang dibutuhkan akan lebih sedikit meskipun teks yang ditransliterasikan banyak. Model *parallel computing* yang digunakan adalah *thread* dengan bahasa pemrograman *Java*. Sehingga dengan menerapkan *parallel computing* diharapkan waktu yang dibutuhkan untuk melakukan proses komputasi akan lebih sedikit dibanding dengan tidak menggunakan multithreading.

1.2 Identifikasi Masalah

Dari latar belakang masalah yang telah diuraikan, maka dapat diidentifikasikan masalah yaitu:

1. Seberapa efisienkah penerapan *parallel computing* untuk mempercepat proses pada aplikasi trasliterasi huruf Latin ke aksara Jawa?

1.3 Tujuan Penelitian

Dari masalah yang telah diidentifikasikan, maka tujuan dari penelitian ini antara lain:

1. Pengukuran efisiensi waktu penerapan *parallel computing* pada proses aplikasi transliterasi huruf Latin ke aksara Jawa.

1.4 Manfaat Penelitian

Hasil dari penelitian ini diharapkan dapat mempercepat waktu komputasi aplikasi transliterasi aksara Jawa.

1.5 Batasan Masalah

Agar penelitian ini tidak keluar dari pokok permasalahan yang dirumuskan, maka ruang lingkup permasalahan dibatasi pada:

1. Format dokumen yang digunakan untuk dasar dan pengujian adalah dokumen *Word* ukuran kertas A4 dengan jenis *font*: *Times New Roman*, ukuran: 12, spasi antar baris: 1.5, tanpa tambahan spasi paragraf, dan *margin* atas dan bawah 1 inci; *margin* kanan dan kiri 1,25 inci. Diasumsikan bahwa rata-rata jumlah kata dalam satu halaman dokumen adalah 380 kata.

1.6 Sistematika Penelitian

Penelitian ini tersusun dalam lima bab dengan sistematika penulisan sebagai berikut:

BAB I Pendahuluan

Bab ini berisikan latar belakang penelitian, indentifikasi masalah, tujuan penelitian, manfaat penelitian, batasan masalah dan sistematika penyusunan penelitian.

BAB II Tinjauan Pustaka

Bab ini berisikan teori-teori dan penelitian-penelitian sebelumnya sebagai tinjauan dan dasar untuk melakukan penelitian ini. Dalam penelitian ini, tinjauan pustaka akan berisi teori-teori dasar penelitian-penelitian yang pernah dilakukan tentang transliterasi aksara Jawa dan Parallel Computing.

BAB III Metodologi Penelitian

Bab ini berisikan desain dan prosedur penelitian. Dalam penelitian ini, bab ini akan membahas tentang desain penelitian, yang meliputi (1) objek penelitian dan (2) sumber data, dan prosedur penelitian, yang meliputi (1) pemahaman sistem dan studi literatur dan (2) desain sistem.

BAB IV Uji Coba dan Pembahasan

Bab ini berisikan penjelasan mengenai pengujian penelitian dan pembahasan. Dalam penelitian ini bab ini akan berisi tentang uji coba dan pembahasan *parallel computing* yang diterapkan untuk mempercepat komputasi proses alih aksara atau transliterasi pada aplikasi Transliterasi Huruf Latin ke Aksara Jawa.

BAB V Penutup

Bab ini berisikan kesimpulan dan saran dari penelitian yang dilakukan.

BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terkait

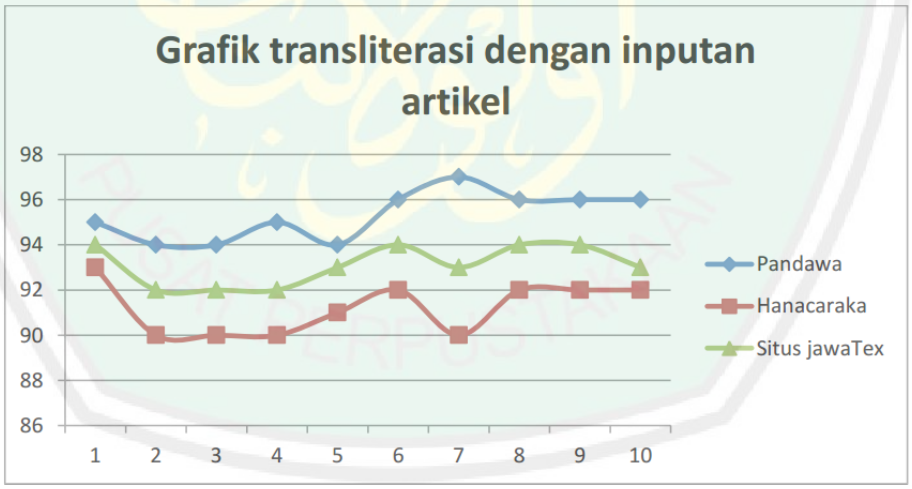
2.1.1 Transliterasi Aksara Jawa

Program transliterasi antara aksara Latin dan aksara Jawa dibangun menggunakan metode *Finite State Automata* jenis *Deterministic Finite Automata (DFA)* (Atina, 2012). Pada program tersebut, teks akan dipenggal menurut pola suku katanya berdasarkan aturan penulisan aksara Jawa. Program tersebut dapat mengalihkan aksara Latin ke aksara Jawa dan sebaliknya. Program tersebut juga dapat mengatasi setiap kata-kata yang rumit. Pada penelitian tersebut dilakukan pengujian efisiensi untuk mengukur kecepatan eksekusi pada program. Hasil uji efisiensi diperoleh grafik model linier yang menunjukkan bahwa banyaknya karakter dan tingkat kerumitan berbanding lurus dengan banyaknya waktu eksekusi yang diperlukan.

Aplikasi transliterasi aksara Latin ke aksara Jawa berbasis Web 'JawaTex' menggunakan metode *Rule-Based* dibangun oleh Utami (2013). Aplikasi tersebut dapat mengalih-aksarakan dokumen yang berisi teks Latin ke aksara Jawa. Pada penelitian tersebut metode yang digunakan untuk pengolahan dokumen teks Latin menjadi daftar pola pemenggalan string Latin adalah metode *rule based*. Dokumen berisi teks latin yang akan ditransliterasikan diparsing terlebih dahulu menggunakan algoritma *The Context-Free Recursive-Descent Parser* sehingga dihasilkan pemenggalan *string* Latin dengan benar, kemudian teks akan dipenggal menjadi per suku kata sesuai dengan aturan penulisan aksara Jawa. Setelah mendapatkan pola pemenggalan suku kata,

tahapan selanjutnya adalah pengubahan pola pemenggalan string latin ke aksara jawa dan pemetaan skema penulisannya menggunakan algoritma *Pattern Matching*.

Penelitian yang dilakukan Utami (2013) tersebut dapat mengalihkan tulisan Latin ke aksara Jawa sesuai dengan yang dikehendaki oleh pengguna, pengguna dapat memilih untuk mentransliterasikan seluruh isi dokumen atau hanya sebagian dokumen saja. JawaTex juga dapat digunakan tanpa harus menggunakan program pengecekan dan pemenggalan string Latin, namun pengguna dituntut untuk mampu melakukan pemenggalan string Latin serta pengkodean dalam format LaTex. Penelusuran pola pemenggalan string Latin dan pola pemetaan dilakukan sendiri oleh pengguna, sehingga pengguna memiliki pilihan untuk memilih bagian mana yang akan ditransliterasikan.

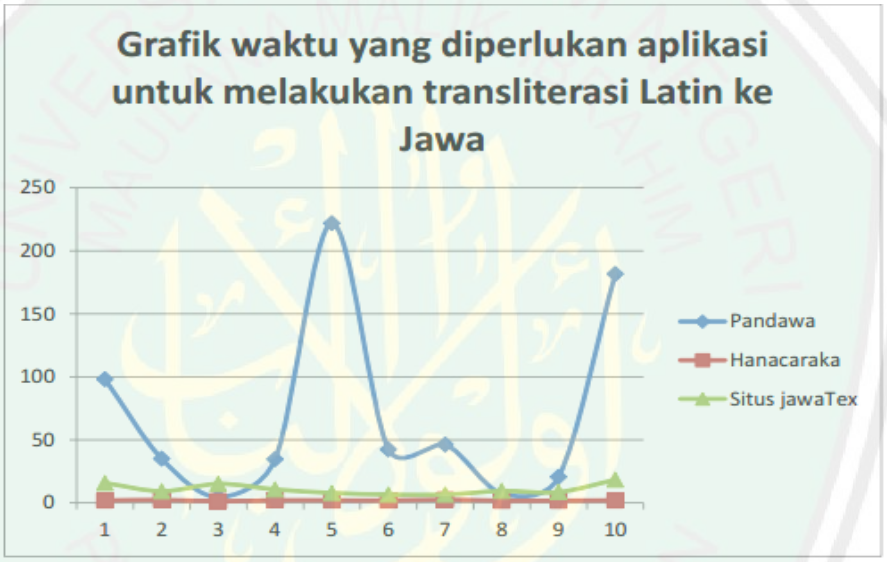


Gambar 2.1. Grafik akurasi transliterasi dengan inputan artikel (Ayumitha, 2014)

Aplikasi transliterasi aksara Latin ke aksara Jawa juga dibangun oleh Ayumitha (2013) menggunakan algoritma *Decision Tree*. Aplikasi bernama 'Pandawa' tersebut dapat mentransliterasikan kata, kalimat maupun alinea ke dalam aksara Jawa. Aplikasi ini memiliki tingkat akurasi yang tinggi, yaitu

diatas 90% dan yang paling tinggi nilai keakuratannya dari dua aplikasi yang dibandingkan dalam penelitian tersebut.

Dari grafik pada Gambar 2.1, pengujian dilakukan dengan mentransliterasikan artikel. Dari pengujian tersebut dapat dilihat bahwa aplikasi Pandawa memiliki tingkat akurasi yang paling tinggi diantara aplikasi lain yang dibandingkan. Dalam laporan penelitiannya dikatakan bahwa tingkat akurasi atau ketepatan aplikasi tersebut adalah di atas 90%.



Gambar 2.2. Grafik perbandingan waktu yang diperlukan untuk transliterasi inputan artikel (Ayumitha, 2014)

Pada penelitian yang dilakukan oleh Ayumitha tersebut, waktu yang dibutuhkan untuk mentransliterasikan berbanding lurus dengan banyaknya teks yang ditransliterasikan, semakin banyak teks maka akan semakin lama pula waktu pemrosesan yang dibutuhkan. Waktu yang diperlukan oleh aplikasi yang dibangun Ayumitha untuk mentransliterasikan artikel merupakan yang paling tinggi dibandingkan kedua aplikasi yang dibandingkan. Dari grafik hasil pengujian yang dilakukan dengan mentransliterasikan artikel menggunakan beberapa aplikasi untuk membandingkan waktu komputasi yang dibutuhkan,

didapatkan kesimpulan bahwa aplikasi Pandawa ini memiliki waktu yang paling tinggi dibandingkan dengan aplikasi lainnya.

2.1.2 Optimalisasi Komputasi

Optimasi Komputasi dapat dilakukan pada metode *support vector machines (SVM)* dengan menambahkan metode *Cross Entropy* (CE) untuk mendapatkan komputasi yang lebih cepat dan sederhana (Santosa dan Widyarini, 2009). Support vector machines merupakan metode yang handal dalam menyelesaikan masalah klasifikasi data. *Cross Entropy* (CE) merupakan suatu metode optimasi yang baru dikembangkan dengan dua prosedur utama, yaitu melakukan generate sample data dengan distribusi tertentu dan melakukan update parameter distribusi berdasarkan sampel terbaik untuk menghasilkan sampel yang lebih baik pada iterasi berikutnya. Pada penelitian yang dilakukan oleh Santosa dan Widyarini(2009) ini metode CE diterapkan untuk menyelesaikan masalah optimasi lagrange SVM agar mendapatkan komputasi yang lebih cepat dan lebih sederhana. Metode yang dikembangkan tersebut diujikan pada enam dataset. Data yang digunakan untuk pengujian terdiri atas 6 kumpulan dataset dari kasus nyata yaitu data Hepatitis, Iris, *Breast Cancer, Haberman's Survival, Credit Approval*, dan *Splice*. Pengujian aplikasi metode CE untuk SVM (SVM-CE) akan dibandingkan dengan pengujian metode KA (*Kernel Adatron*) dan SVM standar. Pada penelitian yang dilakukan Santosa dan Widyarini (2009) ini pengujian dilakukan pada enam dataset yang memiliki karakteristik berbeda, dimulai dari data berukuran kecil hingga besar, serta jumlah atribut sedikit hingga banyak. Banyaknya data berbanding lurus dengan

waktu komputasi. Semakin banyak data yang digunakan maka semakin besar waktu yang dibutuhkan untuk menyelesaikan masalah SVM baik dengan SVM standar maupun SVM CE. Sedangkan banyaknya atribut masih belum terlihat berpengaruh.

Hasil dari penelitian yang dilakukan ini adalah semakin besar jumlah data, semakin baik kinerja dari metode SVM-CE dari segi kecepatan waktu komputasi, dibandingkan dengan metode KA dan SVM Standar (Santosa dan Widyarini, 2009). Dari hasil pengujian pada keenam dataset, metode SVM-CE yang dikembangkan pada penelitian Santosa dan Widyarini (2009) juga terbukti mampu mengklasifikasi data dengan akurasi yang sebanding dengan SVM standar dan KA.

Penelitian untuk mengoptimasi sebuah simulator pelabuhan bernama SPIN (Simulator Pelabuhan Indonesia) (Supeno, 2015). Dari sekian banyak algoritma, Algoritma Genetika dipilih menjadi algoritma yang mengoptimasi kasus penjadwalan yang diangkat dikarenakan kemampuannya untuk mengeksplorasi ruang solusi yang lebih besar dibandingkan dengan exact algorithm. Proses optimasi kemudian dimasukan sebagai input bagi proses TAS (Terminal Appointment System) di gate terminal dalam sebuah simulator pelabuhan bernama SPIN (Simulator Pelabuhan Indonesia) sebagai alat pengujian simulasi yang dilakukan. Namun permasalahan berikutnya muncul, yaitu dengan digunakannya algoritma genetika untuk sebuah simulator maka waktu komputasi menjadi isu yang penting, oleh karena itu algoritma genetika dibangun menggunakan beberapa teknik dengan tujuan meminimalkan waktu komputasi yaitu antara lain permutation encoding dengan two part chromosome

dilanjutkan dengan greedy crossover, tournament selection, dan simple swap mutation dengan tujuan untuk mendapatkan hasil optimasi dalam waktu komputasi yang relatif cepat. Hasil pengujian dari penelitian oleh Supeno ini menunjukkan bahwa Algoritma Genetika yang telah dioptimasi dapat mengurangi waktu bongkar pada simulator dengan cara menugaskan truk kepada pekerjaan selanjutnya yang memiliki jarak terdekat. Teknik kombinasi yang dilakukan pada penelitian ini juga dapat mempercepat proses komputasi berjalannya algoritma genetika pada simulator (Supeno, 2015).

Penelitian dengan menggunakan Metode Moment Invariant untuk mengurangi waktu komputasi pada aplikasi pengenalan pola tanda tangan. Dalam melakukan pengenalan pola, matrik yang mempunyai dimensi besar berpengaruh pada waktu komputasi dan akurasi pengenalan. Berdasarkan hal tersebut maka digunakan metode Moment Invariant untuk mereduksi dimensi matrik. Prinsip kerja moment invariant adalah mengelompokkan matrik ke dalam vektor yang dihasilkan oleh fungsi posisi dan arah piksel citra yang *invariant* terhadap rotasi, translasi dan skala dan didefenisikan dalam momen geometri citra digital (Salambue, 2013). Untuk mengukur kemiripan dari masing-masing citra yang diuji maka dihitung selisih antara piksel matrik yang akan diuji dengan data pelatihan. Hasil penelitian ini menunjukkan bahwa Metode Invariant ini dapat digunakan untuk memperoleh matrik berdimensi rendah sehingga mempercepat waktu komputasi. Metode invariant ini juga tidak berpengaruh terhadap rotasi, translasi dan skala citra (Salambue, 2013).

Penelitian komputasi paralel pada algoritma *Probabilistic Neural Network* (PNN) yang diterapkan pada proses sortasi tomat untuk menguji

performa komputasi paralel (Seminar, Buono, dan Alim, 2006). *Probabilistic Neural Network* (PNN) merupakan salah satu jaringan saraf tiruan yang dapat diterapkan untuk membangun mesin klasifikasi sortasi tomat berbasis komputer. Dalam penelitian yang dilakukan Seminar, Buono, dan Alim (2006) dilakukan sembilan mode pelatihan, yang menurut penelitian sebelumnya mode pelatihan merupakan salah satu factor yang mempengaruhi akurasi klasifikasi PNN. Hasil dari penelitian Seminar, Buono, dan Alim (2006) menunjukkan bahwa nilai akurasi dari model komputasi sekuensial dan komputasi parallel adalah sekitar 89.2%. Namun jika dilakukan pemngamatan lebih seksama rata-rata akurasi komputasi sekuensial sedikit lebih tinggi daripada komputasi parallel, akan tetapi median akurasi komputasi sekuensial lebih rendah dari pada komputasi paralel. Sebaran data akurasi kedua model komputasi relative sama. Sedangkan waktu eksekusi total dari komputasi paralel lebih rendah daripada komputasi sekuensial, dimana rata-rata waktu eksekusi komputasi sekuensial adalah 548.66 detik dan rata-rata waktu eksekusi komputasi paralel adalah 439.57 detik. Penerapan dari komputasi paralel pada penelitian Seminar, Buono, dan Alim (2006) menghasilkan peningkatan kecepatan yang cukup signifikan namun tidak menyebabkan perubahan yang berarti terhadap akurasi klasifikasi. (Seminar, Buono, dan Alim, 2006).

2.1.3 Parallel Computing

Pengimplementasian komputasi paralel pada GPU CUDA dapat dilakukan untuk pengembangan *image inpainting* dengan metode *Perona-Malik* (Panca, 2015). Image inpainting adalah proses memperbaiki atau mengisi bagian

yang hilang pada citra digital berdasarkan informasi yang dikumpulkan dari daerah sekitarnya. Dalam penelitian yang dilakukan ini diteliti bagaimana melakukan *retouching* citra dengan mengimplemantasikan algoritma *Perona-Malik* yang dikombinasikan dengan algoritma *Heat* untuk melakukan *Inpainting* pada citra digital menggunakan komputasi paralel dengan memanfaatkan GPU melalui sebuah *platform* yang dinamakan CUDA. CUDA merupakan API *(application programming interface)* dari NVIDIA yang dapat menjalankan GPU.

Sejak tahun 2003 terdapat dua arsitektur dalam pengembangan mikroprosesor, yaitu multicore dan many-core. Arsitektur multicore bermula dari prosesor berinti dua dan terus berambah hingga saat ini sudah mencapai delapan inti pada satu mikroprosesor. Arsitektur multicore banyak digunakan pada *sentral processing unit* (CPU). Arsitektur ini berfokus untuk memberikan performa yang baik pada program yang didesain untuk berjalan pada banyak inti, dan juga tetap menjaga performa kecepatan eksekusi program yang didesain secara sekuensial. Arsitektur many-core berfokus untuk memaksimalkan kinerja program yang benar-benar didesain untuk bekerja secara paralel. Arsitektur many-core memiliki banyak unit pemroses daripada arsitektur multicore dan arsitektur ini biasanya digunakan pada *graphic processing unit* (GPU) (Panca, 2015).

Penelitian yang dilakukan ini adalah membandingkan metode Heat, Perona-Malik 1 dan Perona-Malik 2 pada CPU dan GPU (Panca, 2015). Hasil dari pengujian dari penelitian ini menunjukkan bahwa metode Heat adalah yang paling cepat dieksekusi pada GPU. Sedangkan yang paling lama eksekusinya

adalah metode Perona-Malik 1 pada CPU, ini dikarenakan rumus yang dieksekusi sangat kompleks sehingga CPU memerlukan waktu lama untuk mengeksekusinya secara serial. Dari hasil pengujian, perbandingan implementasi pada CPU dan GPU secara keseluruhan, GPU dapat memberikan hasil yang baik dari sisi kecepatan proses komputasi secara paralel, dan CPU dapat memberikan hasil proses yang baik dari sisi kualitas walaupun kualitas yang dihasilkan tidak terlalu jauh berbeda dari hasil GPU.

Penelitian mengenai *parallel computing* juga telah dilakukan dengan menganalisis unjuk kerja komputasi *Distributed Shared Memory* (DSM) pada sistem cluster komputer personal (Gozali dan Lagusto, 2005). Teknik DSM merupakan salah satu teknik manajemen pada komputer cluster. Dalam penelitian tersebut teknik ini memungkinkan proses yang berada pada mesin yang berbeda di computer cluster untuk saling berbagi penggunaan memori walaupun secara fisik mesin-mesin anggota cluster ini tidak memiliki memori bersama pada sistem memori terdistribusi, atau dapat juga dikatakan DSM akan menyediakan sebuah memori maya yang global atau *global virtual memory* (Gozali dan Lagusto, 2005). DSM dirancang untuk memberikan fasilitas shared memory pada sistem computer dengan memori yang secara fisik terdistribusi. Pada penelitian yang dilakukan ini sistem DSM diimplementasikan menggunakan Adsmith yang merupakan library pemrograman pada bahasa C++. Adsmith diimplementasikan untuk berjalan di atas sistem *parallel virtual machine*. Adsmith digunakan untuk mengetahui kinerja sistem DS pada computer cluster. Pada penelitian ini pengujian dilakukan pada dua sistem yaitu sistem Adsmith dan sistem PVM. Pengujian sistem PVM dilakukan untuk

mengetahui kinerja cluster yang tidak mengimplementasikan Adsmith. Hasil dari penelitian ini menunjukkan bahwa kinerja computer cluster yang menggunakan sistem PVM cenderung lebih baik dibandingkan dengan sistem Adsmih pada beban kerja yang relatif rendah dan kinerja computer cluster yang menggunakan sistem Adsmith cenderung lebih baik dibandingkan dengan sistem PVM pada beban kerja yang relatif lebih tinggi. Computer cluster yang menggunakan sistem adsmith memiliki keunggulan pada beban kerja yang cenderung lebih berat. Keunggulan kinerja sistem adsmith dibandingkan dengan sistem PVM disebabkan oleh kinerja sistem PVM yang menurun akibat beban komunikasi internal pada sistem PVM itu sendiri. Secara srsitektur sistem *distributed shared memory* akan memiliki keunggulan dalam hal pemrograman *shared memory* yang relative lebih sederhana dibandingkan dengan metode pemrograman dengan *message passing* yang digunakan oleh sistem PVM (Gozali dan Lagusto, 2005).

Multithreading diimplementasikan untuk meningkatkan kinerja pengolahan kinerja pengolahan citra digital (Mulya dan Abdiansyah, 2013). Dalam penelitian tersebut dilakukan pengujian menggunakan tiga sampel citra. Hasil pengujian dari penelitian yang dilakukan Mulya dan Abdiansyah (2013) tersebut menunjukkan bahwa teknik pemrograman *multithreading* menghasilkan pemrosesan yang lebih cepat. Selisih waktu pemrosesan dengan *single-thread* dan *multithread* menunjukkan bahwa adanya perbedaan waktu yang signifikan semakin besarnya sebuah citra yang diproses. Hal ini terjadi karena pemrograman *multithreading* dapat mengoptimalkan kinerja prosesor terutama untuk computer dengan prosesor ganda.

Pada komputer dengan prosesor tunggal teknik *multithreading* tidak menghasilkan proses yang benar-benar paralel karena beberapa *thread* tersebut dikerjakan pada satu prosesor atau dikerjakan dengan cara time-slicing. Sedangkan untuk komputer dengan prosesor ganda *thread-thread* tersebut akan benar-benar didistribusikan pada prosesor-prosesor dalam sebuah computer sehingga tercipta pemrosesan secara parallel yang sesungguhnya (Mulya dan Abdiansyah, 2013).

Multithreading juga diimplementasikan untuk meningkatkan kinerja Information Retrieval menggunakan metode GVSM (Pardede, 2014). Information Retrieval atau IR digunakan untuk menemukan suatu dokumen dari dokumen-dokumen tidak terstruktur yang memberikan informasi yang dibutuhkan dari koleksi dokumen yang sangat besar yang tersimpan dalam computer (Pardede, 2014). Metode GVSM *(Generalized Vector Space Model)* merupakan perluasan dari *Vector Space Model* (VSM) yaitu dengan menambahkan jenis informasi tambahan, disamping term, dalam merepresentasikan dokumen. IR dengan GVSM merepresentasikan dokumen dengan similaritas vektor terhadap semua dokumen yang ada (Pardede, 2014).

Penelitian ini mengimplementasikan *Thread* untuk melakukan beberapa tahapan yaitu,

1. Membuang kata depan dan kata penghubung
2. Menggunakan *stemmer* pada kumpulan dokumen dan *query* untuk menghilangkan imbuhan (awalan, akhiran)
3. Menentukan minterm untuk menentukan kemungkinan pola frekuensi kata

4. Menghitung banyaknya frekuensi kemunculan kata dalam kumpulan dokumen yang sesuai dengan *query*. Banyaknya *Thread* yang berjalan secara bersamaan adalah sesuai dengan banyaknya koleksi dokumen.

Pada penelitian yang dilakukan Pardede ini, pengujian dilakukan dengan melakukan pencarian sebuah dokumen dari sejumlah dokumen. Didapatkan kesimpulan dari hasil pengujian penelitian tersebut bahwa pengimplementasian pemrograman *multithreading* dapat mempercepat kinerja dari IR, yang dapat menghemat waktu hingga lebih dari 50%.

2.2 Implementasi *Parallel Computing* untuk Mengoptimalkan Waktu Komputasi Pada Aplikasi Transliterasi Aksara Jawa

Dari beberapa penelitian terkait transliterasi aksara Jawa, optimasi komputasi dan *parallel computing* dikerjakanlah penelitian yang menggabungkan antara keduanya, yaitu dengan menerapkan *parallel computing* yaitu *multithreading* ke dalam aplikasi transliterasi aksara Jawa untuk mempercepat waktu komputasi pada aplikasi tersebut.

Parallel computing merupakan model pemrograman yang menggunakan beberapa sumber komputer untuk mengerjakan suatu permasalah komputasi secara simultan. Masalah komputasi untuk dapat dilakukannya komputasi paralel adalah :

1. Dapat dipecah menjadi potongan-potongan pekerjaan yang bisa diselesaikan secara bersamaan
2. Dapat mengeksekusi secara bersamaan beberapa instruksi program setiap saat.

3. Dapat diselesaikan dalam waktu yang lebih singkat dengan komputasi menggunakan lebih dari satu sumber daya dibandingkan dengan komputasi dengan menggunakan satu sumber daya.
4. Sumber daya yang dibutuhkan dalam komputasi paralel adalah satu komputer dengan prosesor atau inti ganda/lebih dari satu.
5. Sejumlah komputer yang dihubungkan oleh sebuah jaringan.

Dalam konsep *parallel computing* Michael Flynn mengklasifikasikan komputer berdasarkan dari jumlah instruksi dijalankan dan aliran data yang diprosesnya. Flynn mengklasifikasikan komputer menjadi empat macam.

1. *Single Instruction, Single Data Stream* (SISD)

   SISD adalah komputer yang intruksi-intruksinya dijalankan satu-persatu dan sebuah instruksi tunggal berhubungan dengan tidak lebih dari satu operasi data. Karakter dari SISD adalah instruksi dijalankan secara sekuensial dan data diproses secara serial.

2. *Multiple Instruction, Single Data Stream* (MISD)

   MISD adalah komputer yang dapat melakukan banyak instruksi terhadap satu aliran data. Namun struktur computer ini tidak praktis karena memerlukan banyak sumber untuk mengoperasikan data yang sama, sehingga struktur komputer jenis ini tidak pernah diaplikasikan.

3. *Single Instruction, Multiple Data Stream* (SIMD)

   SIMD adalah kebalikan dari MISD, yaitu komputer yang mampu memproses banyak aliran data dengan hanya satu instruksi, sehingga operasi yang dilakukan adalah paralel. Contoh dari computer jenis ini adalah Processor Arryas dan GPU.

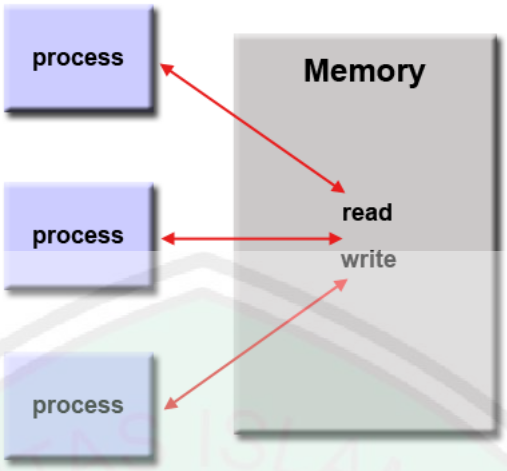
4. *Multiple Instruction, Multiple Data Stream* (MIMD)

MIMD merupakan sebuah tipe dari komputer paralel. Pada computer ini setiap prosesor dapat mengeksekusi instruksi yang berbeda dan mengoperasikan data yang berbeda. Akhir-akhir ini komputer jenis ini adalah jenis komputer paralel yang paling umum.

Dalam *parallel computing* ada beberapa model pemrograman (Bayney, 2014), yaitu:

1. *Shared Memory* (tanpa *thread*)

Dalam model pemrograman ini, proses / tugas berbagi ruang alamat umum, tampat proses dan tugas dapat menulis atau membaca perintah secara asinknron. Model pemrograman Shared Memory ini menggunakan berbagai mekanisme seperti kunci / semaphore untuk mengendalikan akses ke memori bersama (*shared memory*), menyelesaikan perselisihan dan untuk mencegah kondisi balapan dan kebuntuan (*deadlocks*). Model pemrograman shared memory ini merupakan model pemrograman paralel yang paling sederhana. Keuntungan dari model ini dari sudut pandang pemrogram adalah bahwa kurangnya gagasan tentang "kepemilikan" data, sehingga tidak perlu secara eksplisit menentukan komunikasi data antar tugas. Semua proses dapat melihat dan memiliki akses yang sama ke memori bersama. Kerugian penting dari model pemrograman ini dalam hal kinerja adalah semakin sulitnya untuk memahami dan mengelola lokasi data, yang mungkin diluar kendali pengguna umum/rata-rata.

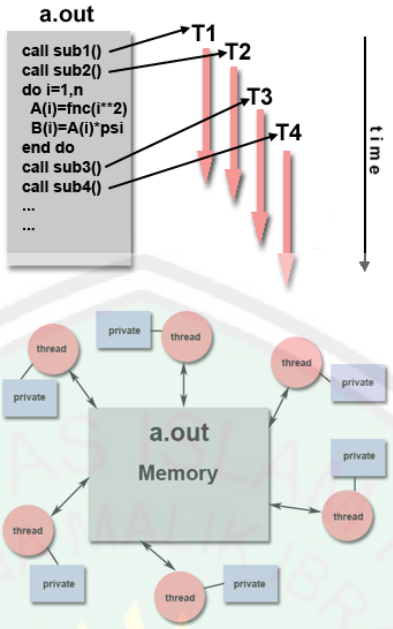


Gambar 2.3 Model Pemrograman *Shared Memory*

2. *Threads*

Salah satu model premrograman paralel adalah *Thread*. Model pemrograman ini adalah salah satu tipe dari *shared memory programming*. Dalam model pemrograman paralel thread, suatu proses yang dikategorikan "berat" dapat memiliki beberapa proses "ringan", yang dieksekusi bersamaan.

Implementasi thread bukanlah hal yang baru dalam komputasi. Secara historis, vendor perangkat keras telah mengimplementasikan versi thread milik mereka sendiri. Implementasi ini berbeda secara substansial satu sama lain sehingga menyulitkan programmer untuk mengembangkan aplikasi portabel ber-*thread*.



Gambar 2.4 Model Pemrograman *Thread*

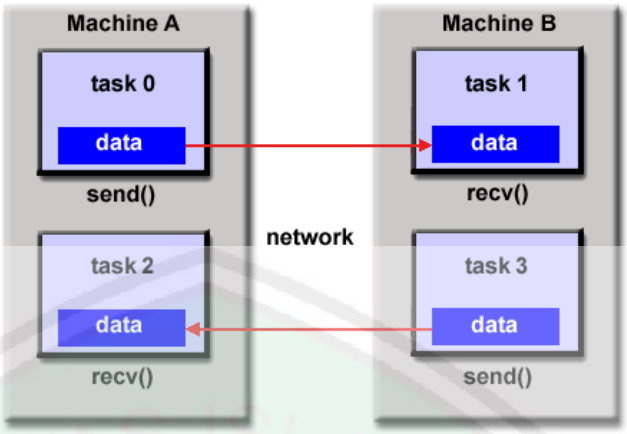
3. *Distributed Memory / Message Parsing*

   Model pemrograman ini memiliki karakteristik sebagai berikut:

   a. Selama komputasi berlangsung pengerjaan proses/tugas menggunakan memori local mereka sendiri.

   b. Pertukaran data pada saat komputasi dilakukan melalui komunikasi dengan mengirim dan menerima pesan.

   c. Transfer data biasanya membutuhkan operasi yang kooperatif untuk dilakukan setiap prosesnya. Misalnya, operasi kirim harus memiliki operasi penerimaan yang sesuai.

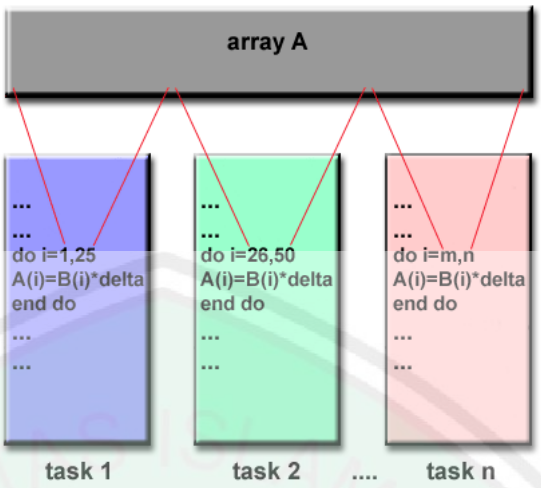


Gambar 2.5 Model Pemrograman Distributed memory/ Message Parsing Model

4. *Data Parallel*

Dapat juga disebut sebagai model *Partitioned Global Address Space* (PGAS) atau Ruang Alamat Global Terpartisi. Model pemrograman *data parallel* ini memiliki karakteristik berikut:

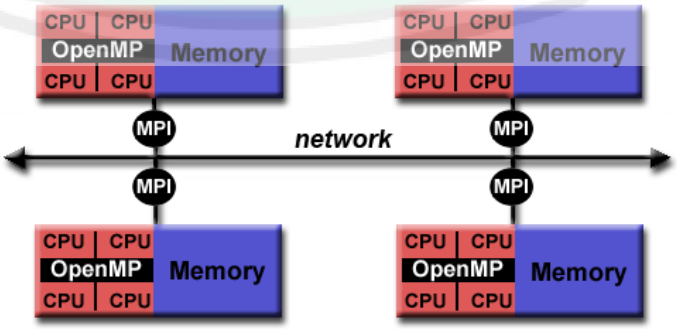
a. Ruang alamat diperlakukan secara global.
b. Sebagian besar pekerjaan paralel berfokus pada melakukan operasi pada kumpulan data. Kumpulan data biasanya disusun dalam struktur umum, seperti array atau kubus.
c. Satu set tugas bekerja secara kolektif pada struktur data yang sama, namun, masing-masing tugas bekerja pada partisi berbeda dari struktur data yang sama.
d. Tugas melakukan operasi yang sama pada partisi kerja mereka, misalnya, "tambahkan 4 ke setiap elemen array".
e. Pada arsitektur memori bersama, semua tugas mungkin memiliki akses ke struktur data melalui memori global.
f. Pada arsitektur memori terdistribusi, struktur data global dapat dibagi secara logis dan / atau secara fisik antar tugas.



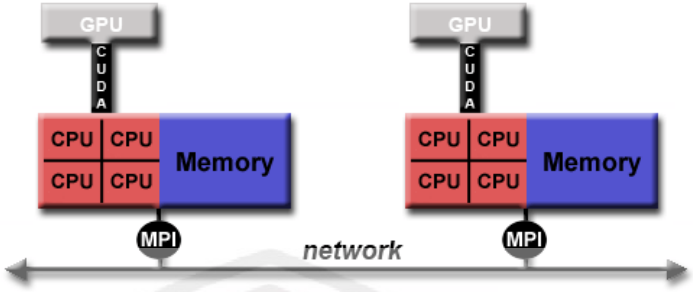
Gambar 2.6 Model Pemrograman *Data Parallel*

5. *Hybrid*

Model *Hybrid* menggabungkan lebih dari satu model pemrograman yang telah dijelaskan sebelumnya. Saat ini, contoh umum dari model *hybrid* adalah kombinasi dari *message passing model* (MPI) dengan model *thread* (OpenMP). Model *hybrid* ini cocok dengan arsitektur perangkat keras berkelas multi-core yang paling populer (saat ini). Contoh lain yang serupa dan semakin populer dari model hibrida adalah menggunakan MPI dengan pemrograman CPU-GPU (Graphics Processing Unit). Pertukaran data antara memori node-local dan GPU menggunakan CUDA (atau sesuatu yang setara).



Gambar 2.7 Model pemrograman Hybrid kombinasi *message passing* (MPI) dan *thread* (OpenMP)



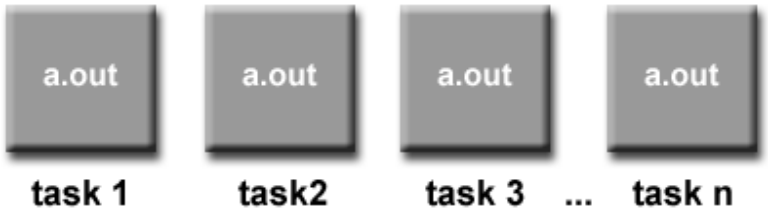
Gambar 2.8 Model pemrograman Hybrid kombinasi *message passing* (MPI) dan *CPU-GPU(CUDA)*

6. *Single Program Multiple Data (SPMD)*

SPMD sebenarnya adalah model pemrograman "tingkat tinggi" yang dapat dibangun di atas kombinasi model pemrograman paralel yang telah disebutkan sebelumnya. Karakter dari model pemrograman SPMD ini adalah sebagai berikut:

a. Program Single: Semua tugas menjalankan program yang sama secara bersamaan. Program ini bisa berupa *thread*, *message passing*, *data parallel atau hybrid.*
b. Data Multiple: Semua tugas dapat menggunakan data yang berbeda

Program SPMD biasanya memiliki logika yang diperlukan yang diprogramkan untuk memungkinkan tugas yang berbeda ke cabang atau kondisional hanya menjalankan bagian program yang dirancang untuk dijalankan. Artinya, tugas tidak harus harus melaksanakan keseluruhan program - mungkin hanya sebagian saja. Model SPMD, dengan menggunakan pemrograman *message parsing* maupun *hybrid*, merupakan model pemrograman paralel yang paling umum digunakan untuk *cluster multi-node*.



Gambar 2.9 Model Pemrograman *Single Program Multiple Data* (SPMD)

7. *Multiple Program Multiple Data (MPMD)*

Seperti SPMD, MPMD sebenarnya adalah model pemrograman "tingkat tinggi" yang dapat dibangun berdasarkan kombinasi model pemrograman paralel yang telah disebutkan sebelumnya. Karakter dari model pemrograman MPMD ini adalah sebagai berikut:

a. Program Multiple: Tugas dapat menjalankan program yang berbeda secara bersamaan. Programnya bisa berupa *thread, message passing, data parallel* atau *hybrid*.
b. Data Multiple: Semua tugas dapat menggunakan data yang berbeda

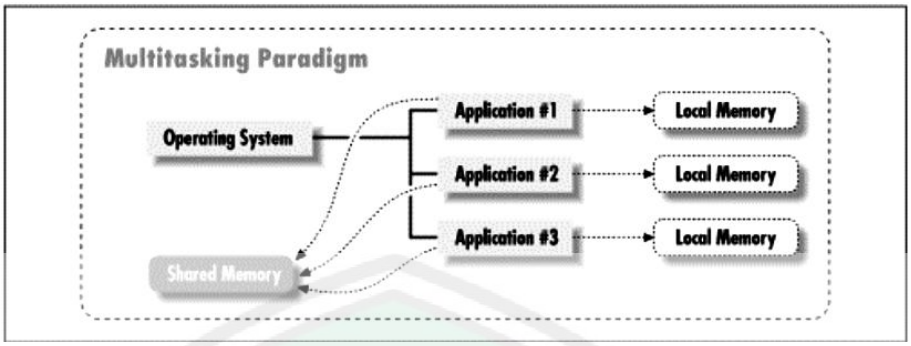
Aplikasi MPMD tidak biasa seperti aplikasi SPMD, namun mungkin lebih sesuai untuk jenis masalah tertentu, terutama yang memberikan penguraian fungsional lebih baik daripada dekomposisi domain (dibahas nanti di bawah Partioning).

Gambar 2.10 Model Pemrograman *Multiple Program Multiple Data* (MPMD)

Salah satu model premrograman paralel adalah *Thread*. Model pemrograman ini adalah salah satu tipe dari *shared memory programming*. Dalam model pemrograman paralel thread, suatu proses yang dikategorikan

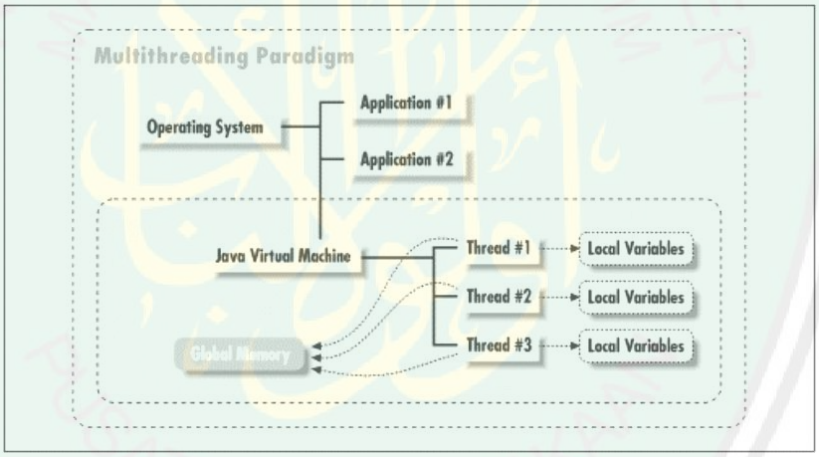
"berat" dapat memiliki beberapa proses "ringan", yang dieksekusi bersamaan (Baeney, 2014). Java menyediakan dukungan *built-in* untuk pemrograman *multithreaded*. Sebuah program *multithreaded* mengandung dua atau lebih *thread* yang dapat dijalankan secara bersamaan. Setiap *thread* masing-masing mendefinisikan eksekusi pada jalur yang terpisah (Schildt, 2014). *Multithreading* merupakan salah satu teknik dalam pemrograman komputer yang bertujuan untuk membuat proses berjalan secara paralel / bersamaan. Dengan teknik ini, dapat dimungkinkan untuk membuat program komputer melakukan pekerjaan yang banyak dalam satu waktu (Mulya dan Abdiansyah, 2013).

Sekarang ini sangat umum dengan digunakannya *multitasking operating system* yang dapat menjalankan beberapa program secara bersamaan atau simultan. Setiap program ini minimal memiliki satu *thread* di dalamnya. Misalkan saja kita menjalankan dua program dalam satu komputer, yaitu *text editor* dan *file manager*. Maka ada dua proses yang berjalan di dalam komputer tersebut, tiap proses tersebut memiliki satu *thread*. Masing-masing proses tersebut tidak perlu tahu tentang proses lainnya yang sedang berjalan, namun tergantung dengan sistem operasi yang berjalan di computer tersebut, karena ada beberapa cara dimana beberapa proses dapat saling berkomunikasi atau mengirim pesan. Salah satu yang paling umum adalah kita dapat menarik *file icon* dari *file manager* ke dalam *text editor* untuk meng-*edit file* tersebut. Masing-masing proses tersebut berjalan secara independen (Oaks dan Wong, 1999).



Gambar 2.11 Proses multitasking pada sistem operasi (Oaks dan Wong, 1999)

Penjelasan diatas mengarahkan kita pada sebuah analogi: kita misalkan thread adalah proses; dan sebuah program dengan banyak *thread* yang berjalan dengan satu instansi pada *Java virtual machine* adalah sebagai beberapa proses yang berjalan di sebuah sistem operasi.



Gambar 2.12 Perbandingan *Multitasking* dengan *Thread* (Oaks & Wong, 1999)

Dalam sebuah program *Java*, *multiple thread* memiliki beberapa property sebagai berikut:

1. Setiap *thread* memulai eksekusi pada lokasi yang telah diketahui. Pada salah satu *thread* di program lokasinya berada pada *main*() *method*; untuk *thread* lainnya berada di tempat yang ditentukan oleh *programmer* pada saat menulis *code*.
2. Setiap *thread* mengeksekusi *code* secara sekuensial atau berurutan, selalu mengeksekusi perintah berikutnya sesuai urutan ditulisnya *code*.

3. Setiap *thread* mengeksekusi *code* secara independen dari *thread* lain pada program.
4. *Thread* memiliki tingkat eksekusi simultan tertentu, yang bergantung pada beberapa factor.
5. *Thread* memiliki akses pada bermacam tipe data.

Setiap *thread* adalah terpisah, jadi variabel-variabel lokal di dalam *method* yang dieksekusi oleh *thread* adalah terpisah untuk thread lain. Variabel-variabel lokal ini bersifat *private*; sehingga tidak memungkinkan sebuah *thread* mengakses variabel lokal dari *thread* lainnya. Jika terdapat dua *thread* atau lebih yang mengeksekusi *method* yang sama, masing-masing *thread* mendapatkan salinan *(copy)* variabel lokal tersendiri.

Objek-objek dan variabel instance nya, dapat digunakan secara bersama-sama antar thread dalam program *Java*, dan sharing objek-objek antar thread di program *Java* ini lebih mudah dari pada *sharing* objek data antar proses di kebanyakan sistem operasi. Itulah mengapa, kemampuan *sharing* data objek antar *thread* dengan mudah adalah alasan lain pemrograman dengan *thread* sangat bermanfaat. Akan tetapi *thread Java* tidak dapat dengan bebas mengakses data objek masing-masing antar *thread*, *thread-thread* tersebut membutuhkan ijin (*permission*) untuk mengakses objek, dan sebuah *thread* perlu melalui *object reference* ke *thread* lainnya (Oaks & Wong, 1999).

Terdapat dua cara untuk membuat *thread* eksekusi baru. Cara pertama yaitu dengan mendeklarasikan kelas sebagai *subclass* dari kelas *Thread*. *Subclass* ini harus meng-*override method run* dari kelas *Thread*. Contohnya,

sebuah *thread* yang menghitung bilangan prima lebih besar dapat ditulis seperti pada Gambar 2.13.

```
Class ThreadPrima extends Thread {
      long minPrima;
      ThreadPrima(long minPrima) {
      this.minPrima = minPrima;
      }
      public void run(){
      //menghitung bilangan prima yang lebih besar dari
      minPrima
      ….
      }
}
```

Gambar 2.13 *Source code* untuk menghitung bilangan prima dengan subclass *thread*

Dari code di atas, lalu dibuatlah sebuah thread dan dipanggil *method start* untuk memulai *thread* yang akan mengerjakan kode diatas seperti pada Gambar 2.14.

```
ThreadPrima tp = new ThreadPrima(143);
tp.start();
```

Gambar 2.14 *Source code* untuk membuat dan memulai *thread*

Cara yang kedua yaitu dengan mendeklarasikan kelas yang menimplementasikan *interface Runnable*. Kelas ini akan megimplementasikan *method run*. Contoh yang sama dengan menggunakan cara ini adalah seperti pada Gambar 2.15.

```
Class RunPrima implements Runnable {
      long minPrima;
      RunPrima(long minPrima) {
            this.minPrima = minPrima;
      }
      public void run(){
      //menghitung bilangan prima yang lebih besar dari
minPrima
      ….
      }
}
```

Gambar 2.15 *Source code* untuk menghitung bilangan prima dengan *interface Runnable*

Dari kode pada Gambar 2.15, lalu dibuatlah sebuah *thread* dan dipanggil *method* start untuk memulai *thread* yang akan mengerjakan kode tersebut yang dapat dilihat pada Gambar 2.16.

```
RunPrima rp = new RunPrima(143);
New Thread (rp).start();
```

Gambar 2.16 *Source code* untuk membuat dan memulai *thread* dengan *interface Runnable*

Setiap thread memiliki nama sebagai identifikasi. Lebih dari satu thread dapat memiliki nama yang sama. Jika nama *thread* tidak dispesifikkan ketika thread dibuat, nama baru akan otomatis dibuat untuk thread tersebut (Oracle, 2014).

Penelitian ini menggunakan *parallel computing* model pemrograman *multithread* serta komputasi dari klasifikasi komputasi milik Flynn yaitu *Single Instruction, Multiple Data Stream* (SIMD) dimana komputasi yang dijalankan memiliki satu intruksi dan memproses banyak data. *Multithread* dalam penelitian ini adalah bekerja untuk memproses teks yang begitu banyak agar dibagi menjadi beberapa bagian dan mengerjakan bagian-bagian tersebut secara simultan. Dalam penelitian ini alur instruksi yang dijalankan adalah satu yaitu program transliterasi aksara Jawa sedangkan data yang diproses berbeda-beda karena setiap *thread* akan memproses satu halaman yang berbeda dengan *thread* lainnya.

BAB III

METODOLOGI PENELITIAN

3.1 Desain Penelitian

3.1.1 Bahan Penelitian

Pada penelitian ini terdapat beberapa bahan materi sebagai data uji yang dibutuhkan untuk mengukur waktu komputasi pada aplikasi transliterasi huruf Latin ke aksara Jawa. Materi yang digunakan adalah teks berbahasa Jawa yang dengan huruf Latin. Terdapat sepuluh data uji, data pertama berisikan teks berbahasa Jawa yang ditulis dengan huruf Latin sebanyak 10 halaman, data kedua berisi sebanyak 20 halaman, data ketiga berisi sebanyak 30 halaman dan seterusnya hingga 100 halaman.

3.1.2 Alat Penelitian

1. Kebutuhan Perangkat Keras

   Perangkat keras yang digunakan adalah sebuah laptop dengan spesifikasi sebagai berikut:

   a. Prosesor Intel Core i5 2410M CPU @ 2.30 GHz
   b. RAM 4.00 GB
   c. HardDisk 640 GB
   d. VGA 2311 MB
   e. Monitor 14”
   f. Keyboard

2. Kebutuhan Perangkat Lunak

   Perangkat lunak yang digunakan adalah:

a. Sistem Operasi Microsoft Windows 7 Home Premium
b. NetBeans IDE 8.0.2
c. Java 8u121

3.1.3 Objek Penelitian

Penelitian ini dilakukan dengan menerapkan *parallel computing* untuk mengoptimalkan waktu komputasi pada aplikasi transliterasi huruf latin ke aksara Jawa. *Parallel computing* yang diterapkan adalah model pemrograman *thread*. Sehingga fokus penelitian ini adalah penerapan *parallel computing* pada aplikasi transliterasi yang telah dibangun oleh Ayumitha (2014). Penelitian ini diharapkan dapat mengoptimalkan aplikasi transliterasi huruf latin ke aksara Jawa disamping keakuratannya yang sudah sangat baik.

3.1.4 Sumber Data

Sumber data merupakan segala sesuatu yang dapat memberikan informasi mengenai data. Berdasarkan sumbernya, data dibedakan menjadi dua, yaitu data primer dan data sekunder.

1. Data primer yaitu digunakan peneliti untuk menyelesaikan masalah yang dihadapi. Dalam penelitian ini yang menjadi sumber data primer yaitu aplikasi transliterasi huruf latin ke aksara Jawa yang pernah diteliti sebelumnya.
2. Data sekunder yaitu data yang dikumpulkan dengan maksud selain menyelesaikan masalah yang dihadapi, dalam penelitian ini yaitu berupa

literatur, jurnal, artikel serta situs internet yang berkenaan dengan penelitian ini.

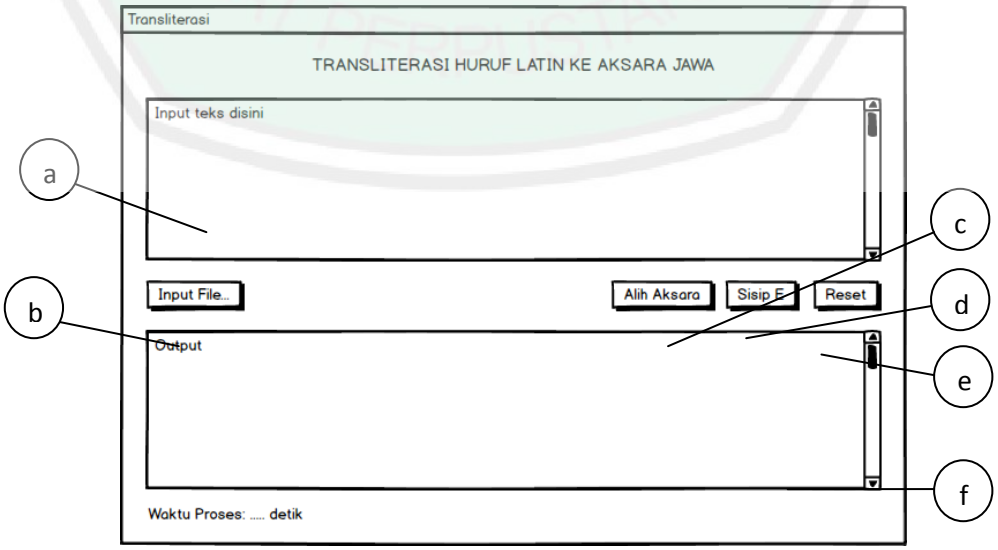
3.2 Prosedur Penelitian

3.2.1 Pemahaman Sistem dan Studi Literatur

Pada tahap ini dilakukan studi literatur mengenai transliterasi aksara Jawa dan mengenai *parallel computing* model *thread* yaitu dengan mengumpulkan informasi – informasi yang dibutuhkan berupa buku yang memuat teori-teori terkait dan jurnal – jurnal yang terkait dengan penelitian ini. Informasi yang dikumpulkan antara lain berupa metode – metode yang pernah digunakan untuk transliterasi aksara Jawa, teori – teori dasar *thread*, serta penelitian mengenai pengimplementasian *thread* yang pernah dilakukan.

3.2.2 Desain Interface

Rencana dari desain interface yang akan dibangun adalah seperti pada Gambar 3.1.



Gambar 3.1 Desain tampilan utama

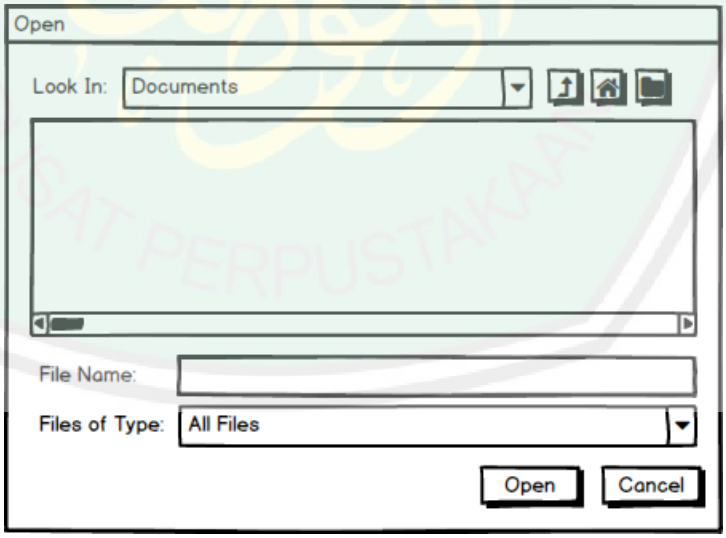
Berikut penjelasan dari Gambar 3.1:

a. *TextArea* Input

TextArea ini berfungsi sebagai tempat untuk memasukkan teks huruf Latin yang akan ditransliterasikan ke dalam aksara Jawa.

b. Tombol Input *File*

Tombol Input *File* ini berfungsi untuk megambil inputan dari *file* dengan format *txt*. Tombol ini menampilkan kotak dialog *open file* yang dapat digunakan pengguna untuk memilih *file* teks yang akan ditransliterasikan. Kotak dialog yang ditampilkan adalah seperti pada Gambar 3.2. Pengguna dapat memilih *file* yang dikehendaki melalui kotak dialog tersebut dan dapat menekan tombol *Open* setelah memilih *file* yang akan ditransliterasikan untuk menjadikan *file* tersebut sebagai teks masukan yang akan ditransliterasikan ke dalam aksara Jawa.



Gambar 3.2 Kotak dialog *Open File*

c. Tombol Alih Aksara

Tombol Alih Aksara ini berfungsi untuk mentransliterasikan teks dengan huruf Latin ke aksara Jawa yang mana komputasinya menggunakan model pemrograman *multithread*.

d. Tombol Sisip E

Tombol Sisip E ini berfungsi untuk menyisipkan huruf é dan/atau huruf è. Tombol ini mengarahkan pengguna ke jendela seperti pada Gambar 3.11. Terdapat dua tombol untuk menyisipkan huruf e taling, yaitu:

- è untuk bunyi e seperti pada kata rènè dan kènè
- é untuk bunyi e seperti pada kata péyék dan téh.

Gambar 3.3 Rencana tampilan jendela untuk menyisipkan huruf E.

e. Tombol Reset

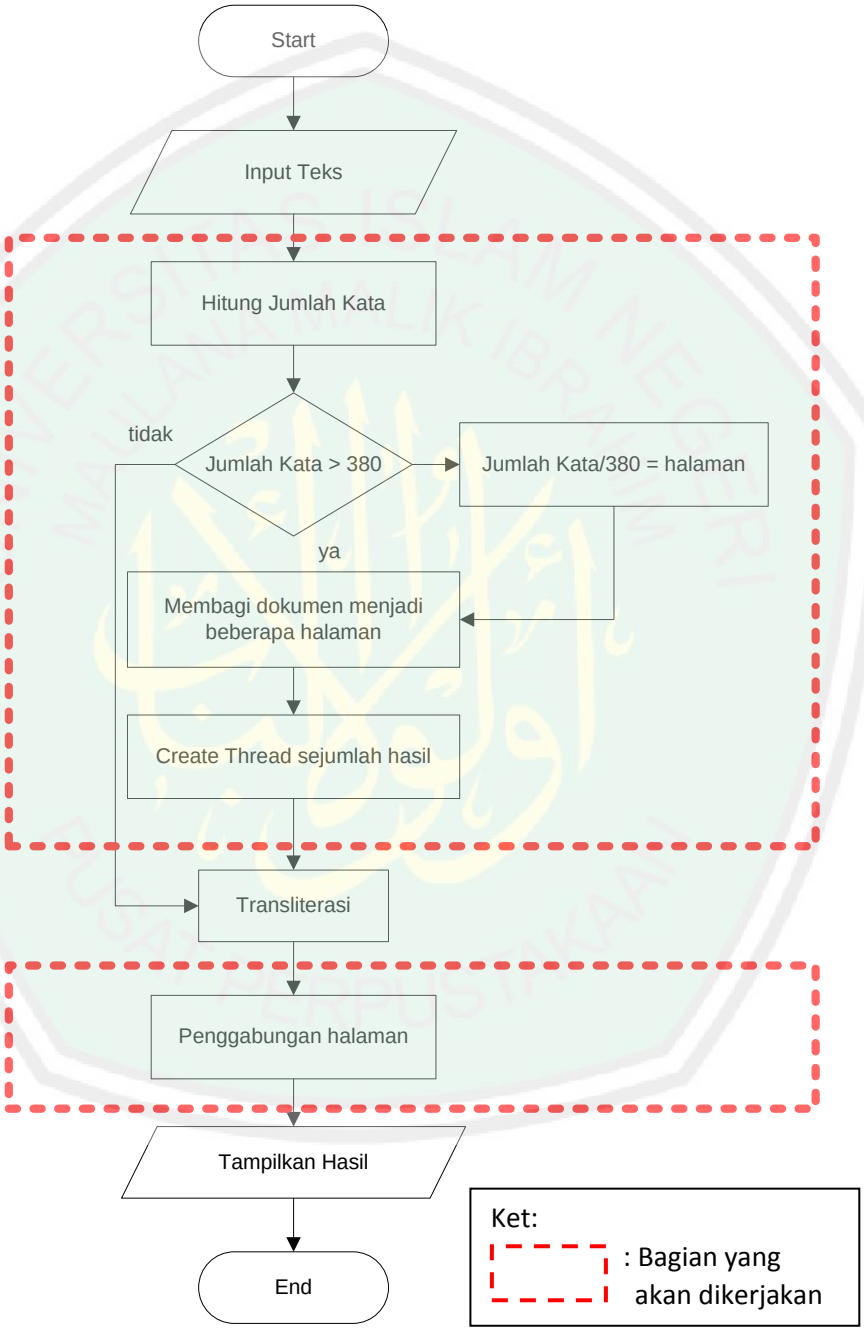
Tombol Reset ini berfungsi untuk mengosongkan *textarea* seperti keadaan semula.

f. *TextArea* Output

Textarea Output ini adalah tempat ditampilkannya keluaran atau hasil dari transliterasi berupa teks beraksara Jawa.

3.2.3 Desain Sistem

Desain sistem dari penelitian ini digambarkan pada diagram alur pada Gambar 3.4.



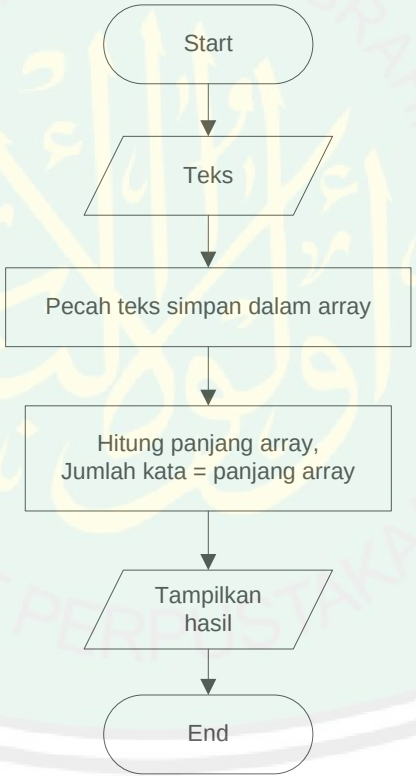
Gambar 3.4. *Flowchart* Sistem.

Penelitian ini akan difokuskan pada *preprocessing*, yang terdiri dari proses perhitungan kata, pembagian teks inputan menjadi beberapa bagian

(halaman), dan membuat *thread* sebanyak jumlah bagian teks. *Preprocessing* ini bertujuan untuk membagi teks menjadi beberapa bagian agar dapat dikerjakan secara paralel. Proses transliterasi akan dikerjakan secara paralel dengan *thread-thread* yang telah dibuat. Hasil transliterasi yang berupa aksara Jawa ditampilkan setelah digabungkan.

1. Penghitungan Jumlah Kata

Proses penghitungan kata dikerjakan setelah teks diinputkan. Alur dari proses ini dapat dilihat pada Gambar 3.5.



Gambar 3.5. *Flowchart* penghitungan jumlah kata

Berikut adalah langkah-langkah dari peoses penghitungan jumlah kata.

a. Input Teks

Teks dimasukkan ke dalam teks area, lalu program akan melakukan pemecahan teks yang diinputkan setelah pengguna menekan tombol transliterasi.

b. Pemecahan Teks

Pemecahan teks dilakukan berdasarkan spasi menggunakan fungsi *split* dan disimpan ke dalam sebuah *array*.

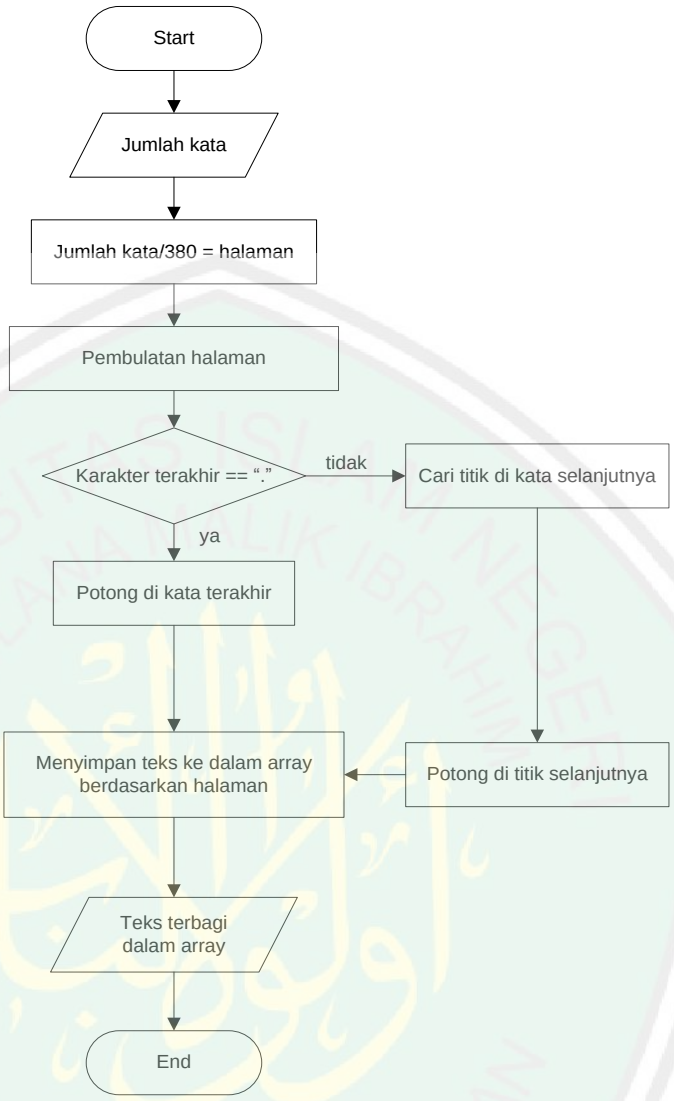
c. Penghitungan Jumlah Kata

Jumlah kata didapatkan dari panjang array dengan menggunakan fungsi array.length.

Setelah jumlah kata didapatkan, selanjutnya ada percabangan *if* sebelum memasuki proses kedua. Jika jumlah kata kurang dari / sama dengan 380 kata (jumlah kata rata-rata dalam satu halaman), percabangan akan langsung menuju ke proses transliterasi, namun jika jumlah kata melebihi 380 kata maka percabangan akan menuju ke proses kedua yaitu proses pembagian teks menjadi beberapa halaman.

2. Penghitungan dan Pembagian Halaman

Proses penghitungan jumlah halaman atau pembagi dilakukan untuk menentukan jumlah *thread* yang akan dibuat sehingga banyaknya halaman sama dengan banyaknya *thread* yang dibuat. Diagram alur dari proses penghitungan dan pembagian halaman ini dapat dilihat pada Gambar 3.6.



Gambar 3.6. *Flowchart* penghitungan dan pembagian halaman

Betikut adalah penjelasan dari diagram alur pada gambar 3.6:

a. Pembagian teks dengan 380

   Jumlah kata yang telah didapatkan dari proses pemecahan kata (*splitting*) akan dibagi dengan 380, yang diasumsikan sebagai rata-rata jumlah kata dalam satu halaman dokumen seperti pada batasan masalah, dan akan menghasilkan variabel jumlah halaman.

b. Pembulatan

Jika pembagian tidak habis dibagi 380 maka akan dibulatkan keatas. Pembulatan ini menggunakan fungsi *divide()* yang telah disediakan oleh *Java* dengan mode pembulatan *(rounding mode) CEILING* atau keatas.

c. Pembagian teks berdasarkan titik

Pembagian teks haruslah dibagi setelah tanda baca titik, karena dalam aturan penulisan aksara Jawa terdapat aturan pasangan yang mana pasangan memiliki fungsi untuk menghubungkan suku kata yang berakhiran konsonan dengan suku kata berikutnya. Oleh karena itu tidak bisa sembarangan memisahkan kata-perkatanya karena akan menyebabkan terjadinya kekeliruan pada hasil yang diperoleh dan akan menyebabkan penurunan pada tingkat akurasinya. Jika karakter terakhir pada kata terakhir di sebuah halaman bukan titik, maka akan dicari kata selanjutnya yang memiliki karakter terakhir titik.

Proses pencarian karakter titik pada kata selanjutnya ini akan menghasilkan batas baru yang menjadikan satu halaman tidak persis berjumlah 380 kata. Dan proses ini juga menyebabkan kemungkinan bahwa jumlah halaman tidak sama dengan jumlah *array* halaman yang telah dibuat sebelumnya.

d. Penyimpanan teks ke dalam *array* dan penghapusan *array* kosong

Setelah proses pembagian halaman selesai dan setiap halaman telah berakhiran titik, *array* yang menyimpan teks berdasarkan halaman ini akan diproses untuk menghapus *array* indeks yang kosong. Setelah didapatkan *array* penuh (tidak ada indeks yang kosong) maka masuklah

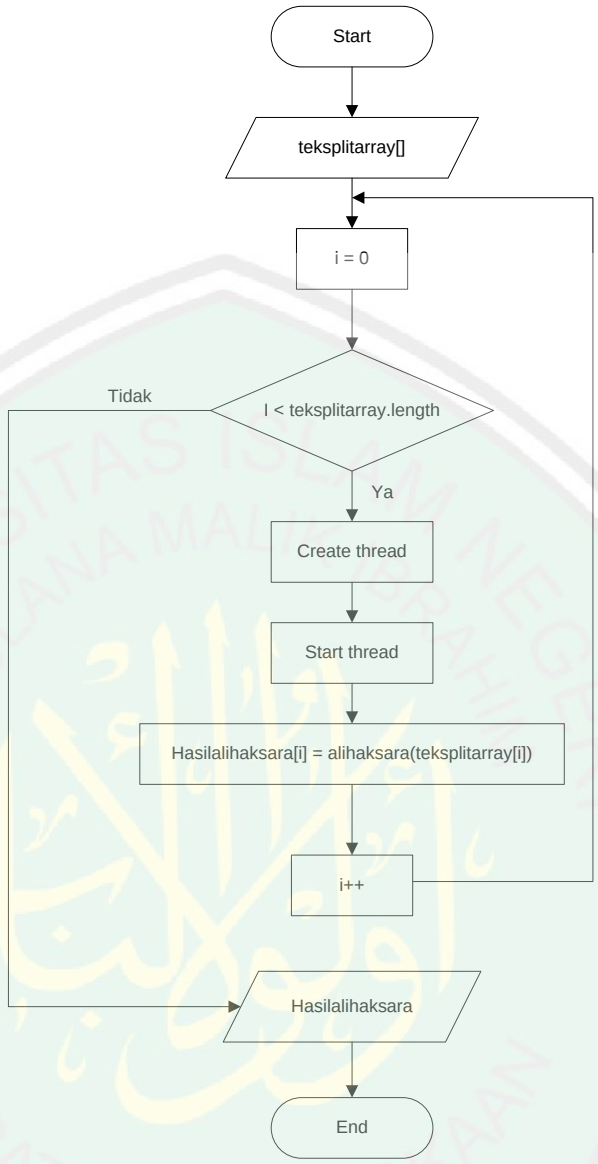
pada proses selanjutnya, yaitu proses transliterasi secara paralel menggunakan *thread* sesuai dengan indeks *array* tersebut.

3. Proses Transliterasi Menggunakan *Multithread*

Proses selanjutnya adalah proses transliterasi menggunakan metode *Desicion Tree* secara *parallel*. Pemrosesan transliterasi akan dikerjakan secara paralel menggunakan *thread-thread* yang jumlahnya dibuat sebanyak halaman dokumen. Diagram alur dari proses transliterasi ini dapat dilihat pada Gambar 3.7.

Gambar 3.7. *Flowchart* proses transliterasi

Proses transliterasi dilakukan dengan menggunakan aplikasi transliterasi dengan metode *Decision Tree* yang telah dibangun pada penelitian Ayumitha (2014).



Gambar 3.8. *Flowchart* proses transliterasi menggunakan *multithread*

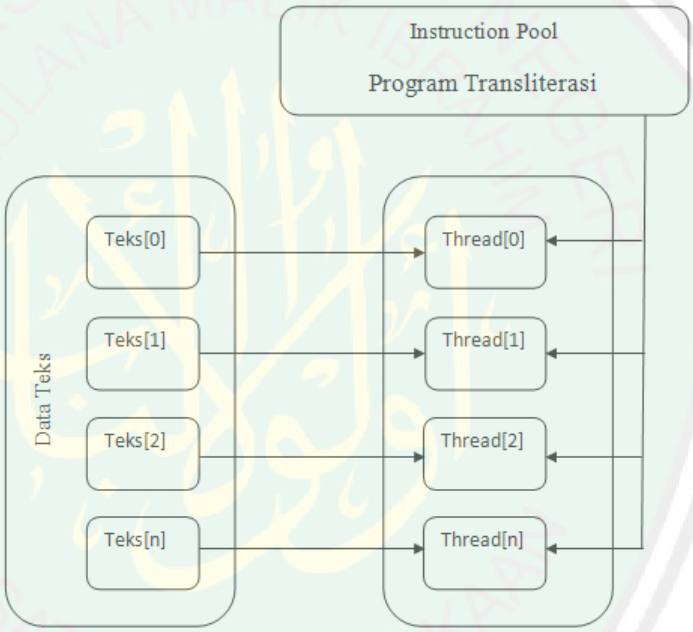
Berikut adalah penjelasan dari diagram alur pada Gambar 3.8.

a. Pembuatan *thread*

Pembuatan *thread* dilakukan setelah mendapatkan jumlah halaman. Setelah jumlah halaman didapatkan maka dilakukan perulangan berdasarkan indeks halaman dari halaman pertama sampai halaman terakhir. Setelah *thread* dibuat maka *thread* akan dihidupkan (*start*).

b. Transliterasi

Setelah *thread* dengan indeks i hidup langkah selanjutnya adalah mentransliterasikan teks dengan indeks i, indeks yang sama dengan indeks *thread*. Proses transliterasi dengan menggunakan *multithread* dilakukan dengan menambahkan *thread* sebelum masuk ke dalam metode *decision tree*. Alur kerja dari *multithread* tersebut dapat dilihat pada Gambar 3.9.



Gambar 3.9 Alur Kerja Komputasi Paralel Menggunakan *Multihread*

Dari Gambar 3.9 dapat dilihat bahwa komputasi yang menjalankan satu instruksi dan memproses banyak data. Komputasi ini merupakan salah klasifikasi komputasi paralel milik Flynn yaitu *Single Instruction, Multiple Data Stream* (SIMD).

c. Penyimpanan hasil

Setelah transliterasi selesai maka teks yang terbagi akan digabungkan kembali dan ditampilkan sebagai hasil dari proses transliterasi.

3.2.4 Uji Coba dan Evaluasi

Uji Coba dilakukan untuk mengetahui apakah metode dapat mengoptimalkan proses komputasi pada aplikasi transliterasi huruf latin ke aksara Jawa. Evaluasi dilakukan dengan mengukur waktu yang dibutuhkan aplikasi sebelum mengimplementasikan *multithread* dan setelah mengimplementasikan *multithread*. Waktu tersebut dicatat sehingga didapatkan perbandingan waktu sebelum dan sesudah pengimplementasian *multithread.* Dari perbandingan tersebut didapatkan data persentase selisih yang dan akan menjadi salah satu dari kesimpulan dari penelitian ini.

3.2.5 Dokumentasi

Penulisan laporan skripsi merupakan dokumentasi keseluruhan penelitian yang dikerjakan. Diharapkan dokumentasi penelitian berguna dan bermanfaat untuk penelitian dan pengembangan lebih lanjut.

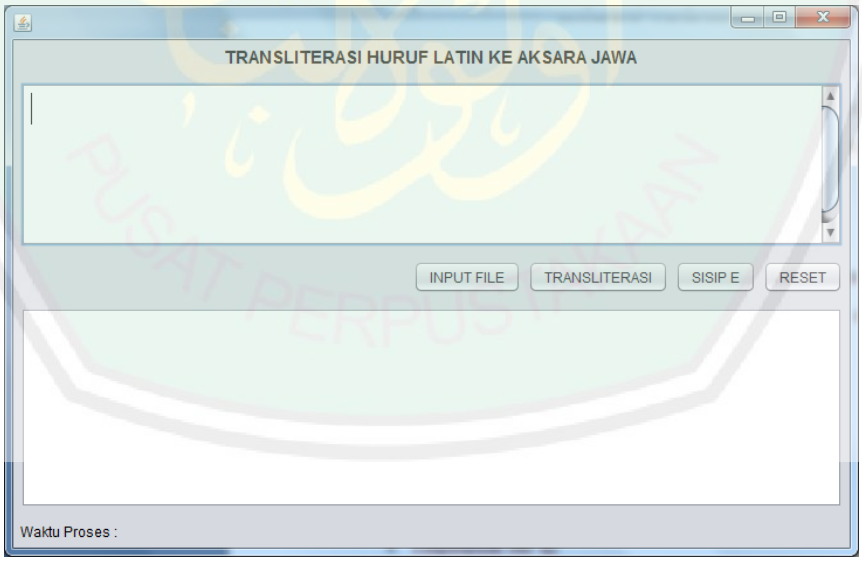
BAB IV

HASIL DAN PEMBAHASAN

Dalam bab ini dibahas mengenai implementasi dari hasil perancangan yang telah dibuat. Implementasi meliputi empat tahap yaitu implementasi antarmuka, implementasi sistem, uji coba dan analisa. Berikut penjelasan tiap tahapan.

4.1 Implementasi Antarmuka

Di dalam implementasi antarmuka, dijelaskan kegunaan dari komponen komponen yang ada pada aplikasi transliterasi huruf latin ke aksara Jawa menggunakan parallel computing ini. Tampilan utama dari aplikasi bisa dilihat pada Gambar 4.1



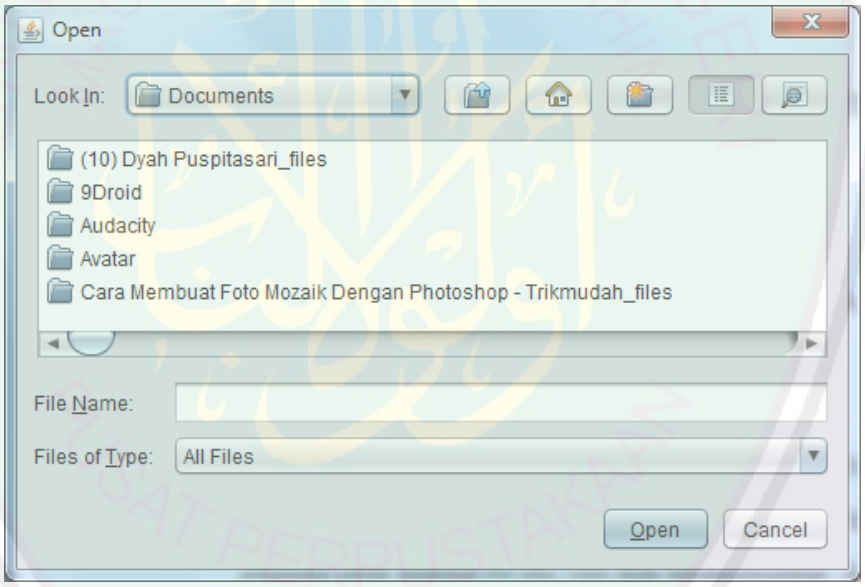
Gambar 4.1 Tampilan Utama

g. *TextArea* Input

TextArea ini berfungsi sebagai tempat untuk memasukkan teks huruf Latin yang akan ditransliterasikan ke dalam aksara Jawa.

h. Tombol Input *File*

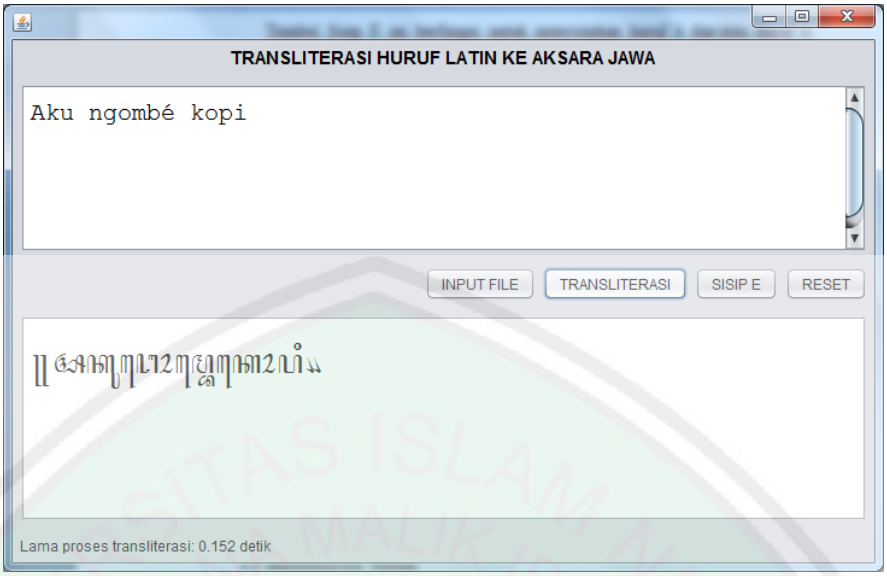
Tombol Input *File* ini berfungsi untuk megambil inputan dari *file* dengan format *txt*. Tombol ini menampilkan kotak dialog *open file* yang dapat digunakan pengguna untuk memilih *file* teks yang akan ditransliterasikan. Kotak dialog yang ditampilkan adalah seperti pada Gambar 4.2. Pengguna dapat memilih *file* yang dikehendaki melalui kotak dialog tersebut dan dapat menekan tombol *Open* setelah memilih *file* yang akan ditransliterasikan untuk menjadikan *file* tersebut sebagai teks masukan yang akan ditransliterasikan ke dalam aksara Jawa.



Gambar 4.2 Tampilan kotak dialog pembuka file

i. Tombol Alih Aksara

Tombol Alih Aksara ini berfungsi untuk mentransliterasikan teks dengan huruf Latin ke aksara Jawa yang mana komputasinya menggunakan model pemrograman *multithread*. Hasil yang dikeluarkan oleh aplikasi ini adalah teks beraksara jawa yang terdapat pada textarea output serta lama proses yang dibutuhkan oleh aplikasi untuk mentransliterasikan teks, yang dapat dilihat pada Gambar 4.3.

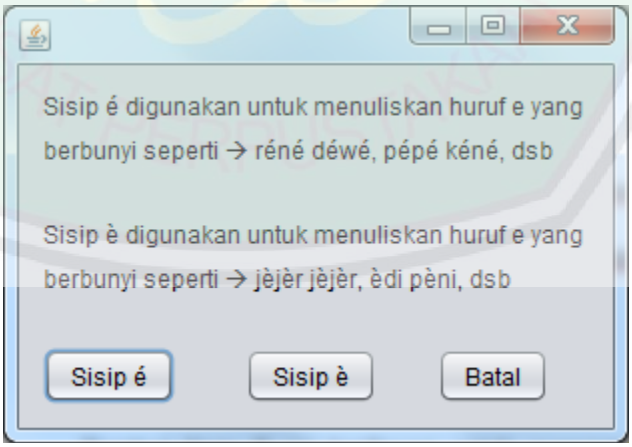


Gambar 4.3 Hasil transliterasi

j. Tombol Sisip E

Tombol Sisip E ini berfungsi untuk menyisipkan huruf é dan/atau huruf è. Tombol ini mengarahkan pengguna ke jendela seperti pada Gambar 4.4. Terdapat dua tombol untuk menyisipkan huruf e taling, yaitu:

- è untuk bunyi e seperti pada kata rènè dan kènè
- é untuk bunyi e seperti pada kata péyék dan téh.



Gambar 4.4 Tampilan Jendela Sisip E

k. Tombol Reset

Tombol Reset ini berfungsi untuk mengosongkan *textarea* seperti keadaan semula.

l. *TextArea* Output

Textarea Output ini adalah tempat ditampilkannya keluaran atau hasil dari transliterasi berupa teks beraksara Jawa.

4.2 Implementasi Sistem

Sistem yang akan di implementasikan pada aplikasi transliterasi huruf latin ke dalam aksara Jawa ini menggunakan bahasa pemrograman Java sebagai platformnya.

4.2.1 Pengaturan dan Penanganan *Font* Aksara Jawa Sebagai Output

Dalam berinteraksi dengan aksara Jawa di komputer, pertama diperlukan penginstalan font aksara Jawa. Selanjutnya dilakukan pengaturan *textarea* agar outputnya berupa aksara jawa. Potongan kode program dari pengaturan font tersebut dapat dilihat pada Gambar 4.5.

```
t_jawa.setFont(new java.awt.Font("Hanacaraka", 0, 20));
```

Gambar 4.5 *Source code* untuk mengatur font aksara jawa pada output

4.2.2 Penghitungan Jumlah Kata

Penghitungan ini dilakukan untuk mendapatkan jumlah kata di dalam teks masukan, dikarenakan sifat kesukukataan pada aturan penulisan aksara Jawa. Pemecahan teks dilakukan berdasarkan spasi menggunakan fungsi *split* dan disimpan ke dalam sebuah *array*. Jumlah kata didapatkan dari panjang array dengan menggunakan fungsi *array.length*. Kode program untuk mendapatkan jumlah kata dapat dilihat pada Gambar 4.6.

```
int jumlahKata = tempKata.length;
```

Gambar 4.6 *Source code* untuk mendapatkan jumlah kata

4.2.3 Pembagian Halaman

Proses penghitungan jumlah halaman, sebagai pembagi, dilakukan untuk menentukan jumlah *thread* yang akan dibuat sehingga banyaknya halaman sama dengan banyaknya *thread* yang dibuat. Jumlah kata yang telah didapatkan dari proses pemecahan kata (*splitting*) akan dibagi dengan 380, yang diasumsikan sebagai rata-rata jumlah kata dalam satu halaman dokumen seperti pada batasan masalah, dan akan menghasilkan variabel jumlah halaman. Karena hasil pembagian tidak selalu habis dibagi dengan 380, maka dilakukan pembagian dengan pembulatan untuk mengatasi hal tersebut, yaitu pembulatan keatas. Potongan dari kode program untuk pembagian halaman dengan pembulatan keatas dapat dilihat pada Gambar 4.7.

```
//pembagian dengan pembulatan keatas
BigDecimal halaman = jumKata.divide
        (BigDecimal.valueOf(ambilKata),RoundingMode.CEILING);
```

Gambar 4.7 *Source code* untuk pembagian halaman dan pembulatan

Setelah mendapatkan jumlah halaman dari pembagian diatas maka tahap berikutnya adalah pegecekan apakah jumlah teks lebih dari 380. potongan program untuk pengecekan jumlah kata apakah sama dengan 380 dapat dilihat pada Gambar 4.8.

```
if (jumlahKata >= ambilKata) {
…
}else {
setIndeke(hal);
setHasilsplit(new String[getIndeke()]);
}
```

Gambar 4.8 *Source code* pengecekan jumlah kata

Setelah pengecekan jumlah kata, akan ada dua kondisi yaitu

1. Jika jumlah kata kurang dari 380 maka teks akan langsung dimasukkan sebagai parameter masukan untuk selanjutnya dilakukan transliterasi.
2. Jika jumlah kata lebih dari 380 maka dilakukan pembagian teks menjadi halaman-halaman setiap 380 kata.

Setelah pengecekan tersebut, terdapat perulangan untuk membagi teks setiap 380 kata, yang dapat dilihat pada Gambar 4.9.

```
while ((jumlahKata - counter) != (jumlahKata % ambilKata)) {
      …
}
```

Gambar 4.9 *Source code* untuk membagi teks tiap 380 kata

Kode diatas digunakan untuk mengatasi pembagian halaman pada teks kata, setiap 380 kata. Variabel jumlahKata adalah jumlah seluruh kata pada teks. Variabel counter adalah variabel yang akan bertambah nilainya berdasarkan variabel batas. Variabel batas adalah variabel yang membatasi pembagian halaman, batas default adalah kelipatan 380, sehingga setiap kata yang memiliki indeks sama dengan variabel batas merupakan kata terakhir pada suatu halaman. Namun variabel batas ini dapat berubah jika akhir dari kata yang memiliki indeks sama dengan batas bukan karakter titik. Perulangan pada kode di atas akan berjalan untuk membagi halaman ketika variabel jumlahKata – counter tidak sama dengan modulo jumlahKata dengan 380. Jika jumlahKata – counter sama dengan modulo jumlahKata dengan 380 maka program akan keluar dari perulangan tersebut dan akan mengeksekusi kode pada Gambar 4.10 berikut.

```
for (int i = counterb; i < batasb; i++) {
     text += tempKata[i] + " "; //gabungkan teks
}
halamanDok[indb] = text; //masukkan ke array
```

Gambar 4.10 *Source code* untuk menyimpan teks halaman terakhir

Variabel counterb diatas didapat dari variabel batas terakhir dari perulangan sebelumnya. Variabel batasb didapat dari jumlahKata. Perulangan pada potongan kode diatas berfungsi untuk menampung teks kata-kata terakhir untuk menempati halaman terakhir pada array halamanDok yang menyimpan teks perhalaman.

Pembagian halaman haruslah dibagi tepat setelah tanda titik, dikarenakan aturan penulisan aksara Jawa yang menggunakan aturan kesukukataan. Cuplikan kode program dari pengecekan karakter titik dapat dilihat pada Gambar 4.11.

```
public char CheckCharLastWord(String LastWord) {
        int LastWordLength = LastWord.length();
        char LastWordChar = LastWord.charAt(LastWordLength - 1);
        return LastWordChar;
  }
```

Gambar 4.11 *Source code method* pengecekan karakter titik

Kode program pada Gambar 4.11 adalah *method* yang digunakan untuk mengecek karakter terakhir dari kata terakhir pada satu halaman. Jika tidak ditemukan karakter titik pada kata terakhir suatu halaman, maka program akan mencari karakter titik pada kata setelah kata terakhir yang telah dicek tersebut. potongan kode untuk mencari kata berikutnya adalah seperti pada Gambar 4.12.

```
public int cariKataBerikutnya(int i) throws IOException {
        String[] tempKata = getHasilsplit();
        int x;
        for (x = i + 1; x < tempKata.length; x++) {
            String kata = tempKata[x];
            char huruf = CheckCharLastWord(kata);
            if (huruf == '.') {
                break;
            }
        }
        System.out.println("x " + x);
        return x;
}
```

Gambar 4.12 *Source code method* pencarian kata berikutnya yang berakhiran titik

Kode program di atas adalah sebuah *method* untuk mencari kata berikutnya yang berakhiran dengan titik. Dan method tersebut mengembalikan

nilai kembalian yaitu indeks dari kata yang memiliki karakter terakhir titik. Setelah didapatkan karakter titik pada akhir kata, maka perulangan berhenti dan method mengembalikan nilai kembalian berupa indeks dari kata yang berakhiran titik tersebut.

```
while ((jumlahKata - counter) != (jumlahKata % ambilKata)) {
      char kar = new mcTh().CheckCharLastWord(tempKata[batas
      - 1]);
      if (kar != '.') {
         batasBaru = cariKataBerikutnya(batas);
         batas = batasBaru + 1;
      }
      …
}
```

Gambar 4.13 *Source code* pemeriksaan karakter titik

Kode pada Gambar 4.13 adalah percabangan yang memeriksa jika karakter dari kata terakhir sebuah halaman adalah bukan titik, maka dilakukanlah pencarian titik pada kata berikutnya. Kode diatas ada di dalam sebuah perulangan sehingga akan mengulang perintah di atas sampai dengan batas dari perulangan.

4.2.4 Proses Transliterasi Secara Parallel Menggunakan *Multithread*

Proses transliterasi dilakukan dengan menggunakan aplikasi transliterasi dengan metode *Decision Tree* yang telah dibangun pada penelitian Ayumitha (2014). Namun proses transliterasi ini akan dilakukan secara parallel yaitu dengan menggunakan multithread. Sebelum proses inti transliterasi program akan menjalankan perintah untuk membuat thread terlebih dahulu.

```
Thread threadnya = new Thread(a);
        threadnya.start();
        try {
            threadnya.join();
        } catch (InterruptedException ex) {

Logger.getLogger(dua.class.getName()).log(Level.SEVERE,     null,
ex);
        }
```

Gambar 4.14 *Source code* untuk membuat dan memulai thread

Pada potongan kode pada Gambar 4.13 *thread* dibuat dengan perintah new Thread(a). Variabel a adalah variabel yang mewakili kelas mcTh dimana kelas tersebut mengimplementasikan *interface* Runnable dan di dalam kelas tersebut terdapat *method* run() yang berisikan perintah untuk menjalankan transliterasi secara parallel menggunakan *multithread*. Perintah threadnya.join(); digunakan agar *thread* lain berjalan setelah *thread* ini selesai dikerjakan, sehingga teks dapat disimpan secara berurutan. Transliterasi dengan *multithread* dapat dilihat pada pada Gambar 4.15.

```
public void run() {
…
        for (int i = 0; i < getIndeke(); i++) {
            try {
                ct.setName("Thread " + i);
                Trans n = new Trans(hasilsplit[i]);
                n.alihaksara();
                hasil[i] = n.getSusunanOke();
            } catch (Exception e) {
                System.out.println(e);
            }
        }
…
    }
```

Gambar 4.15 *Source code* transliterasi secara parallel dengan multithread

Pada method *run()* pada Gambar 4.15 terdapat perulangan berdasarkan indeks halaman yang didapatkan dari function *getIndeke()*. Transliterasi dijalankan pada baris kode *Trans n = new Trans(hasilsplit[i]);*.

Setelah ditransliterasikan teks akan memiliki awalan “pada” atau ? setiap awal halaman. Pada penelitian sebelumnya letak tanda “pada” atau ? hanya terdapat pada awal cerita atau teks.

Sehingga selain halaman pertama maka tanda “pada” akan dihapuskan agar tidak mengurangi akurasi dari transliterasi sebelumnya. Potongan kode untuk penghapusan tanda “pada” adalah seperti pada Gambar 4.16.

```
public static String removeCharAt(String s, int pos) {
      return s.substring(0, pos) + s.substring(pos + 1);
}

Public void run(){
…
…
      for (int i = 0; i < getIndeke(); i++) {
      …
          if (i!=0){
                okokok = n.getSusunanOk();
                okokok = removeCharAt(okokok, 0);
                hasil[i] = okokok.trim();
          }
      }
      …
}
```

Gambar 4.16 *Source code* penghapusan karakter "pada"

Function removeCharAt merupakan fungsi yang menghilangkan karakter indeks 0 pada String okokok yang merupakan tanda baca “pada”.

Setelah ditransliterasikan maka teks digabungkan dan diurutkan berdasarkan indeksnya. Potongan kode penggabungan hasil dari transliterasi dapat dilihat pada Gambar 4.17.

```
Public void run(){
…
…
//gabung hasil
        StringBuffer bf = new StringBuffer();
        for (int i = 0; i < indeke; i++) {
            bf.append(hasil[i]).append("");
        }
        hasilgabung = bf.toString();
        setHasilgabung(hasilgabung);

//menyimpan file output dalam file txt
        try {
            PrintWriter filehasil = new PrintWriter(new
            FileWriter("D:/output.txt"));

            filehasil.print(hasilgabung);

            filehasil.close();
```

```
        } catch (IOException ex) {

Logger.getLogger(mcTh.class.getName()).log(Level.SEVERE,     null,
ex);
            System.out.println("gabung");
        }
}
```

Gambar 4.17 *Source code* penggabungan hasil transliterasi

Hasil teks yang digabungkan ditampung pada variabel string hasilgabung dan kemudian dimasukkan sebagai parameter dalam method setter setHasilgabung() yang nantinya dapat dipanggil dari.

4.2.5 Pencetakan Hasil Pada Tampilan Antarmuka

Setelah teks disimpan dalam variabel maka perintah dikembalikan pada kelas dua.java yang menangani antarmuka tampilan utama. Perintah pencetakan hasil dapat dilihat pada potongan kode program pada Gambar 4.18.

```
t_jawa.append("\n" + masukan);
```

Gambar 4.18 *Source code* untuk menampilkan hasil pada antarmuka

4.3 Pengujian

Untuk mengetahui sejauh mana kemampuan multithreading pada aplikasi dalam mempercepat komputasi alih aksara dari huruf latin ke dalam aksara Jawa maka perlu dilakukan uji coba. Berikut adalah tahapan-tahapan yang dilakukan dalam pengujian pada aplikasi ini.

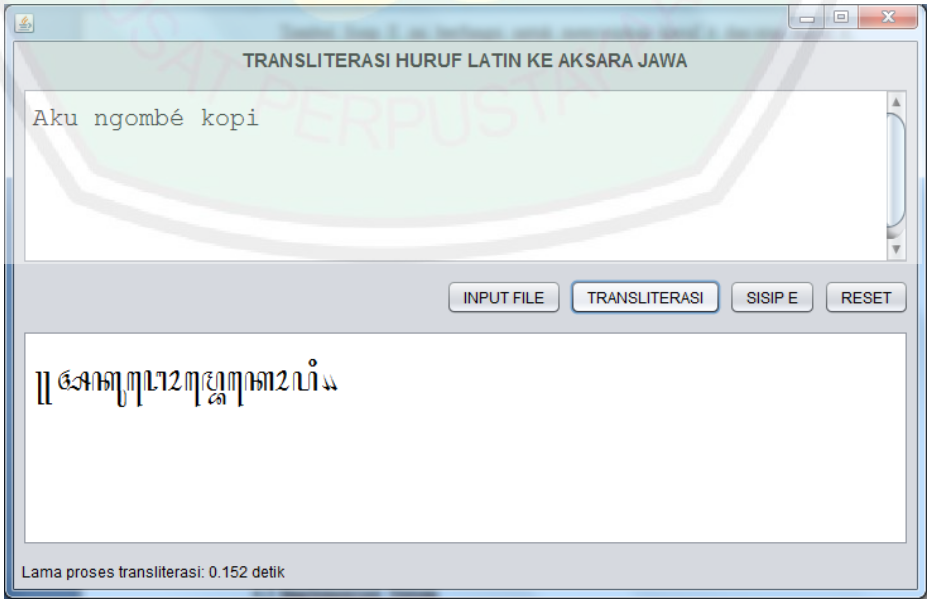
1. Pengumpulan teks uji.

   Pengujian pada penelitian ini menggunakan teks latin berbahasa Jawa. Teks dikumpulkan dari website www.sastra.org, yang kemudian dikelompokkan menjadi 10 halaman, 20 halaman, 30 halaman sampai 100 halaman yang setiap kelompok tersebut dijadikan satu dokumen.

2. Uji coba.

   Pengujian dilakukan dengan memasukkan teks ke dalam aplikasi, lalu aplikasi akan memproses teks dan menghasilkan teks beraksara Jawa dan menampilkan waktu komputasi proses yang telah dilakukan.

   Pengujian mula-mula dilakukan dengan menjalankan aplikasi yang telah dibuat. Ketika aplikasi berjalan, setiap teks uji yang telah disiapkan diketikkan di *TextArea* input. Setelah memasukkan kalimat di *TextArea* input, klik tombol transliterasi maka akan muncul hasil alih aksara di *TextArea* output. Pengujian dilakukan dengan menjalankan hal yang sama pada teks 10 halaman, 20 halaman, sampai dengan 100 halaman.



Gambar 4.19 Contoh pengujian dengan menggunakan *multithread*

Pengujian ini akan membandingkan waktu yang diperlukan dua komputasi yang berbeda, yaitu menggunakan multithread dan tanpa multithread dengan mencatat waktu yang komputasinya.

3. Pencatatan hasil uji coba.

   Waktu komputasi yang ditampilkan dicatat kemudian dibuat perbandingannya sehingga didapatkan kesimpulan dari penelitian ini.

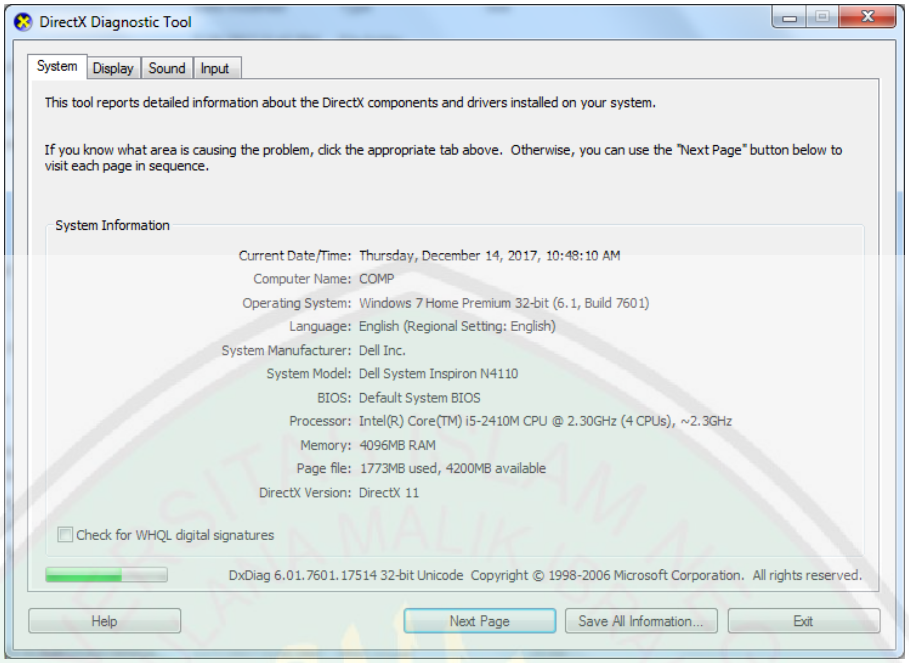
4.4 Hasil dan Analisa

Pengujian dilakukan sebanyak satu kali per dokumen pada setiap komputer, satu kali pengujian menggunakan komputasi secara paralel (dengan *multithread*) dan satu kali sekuensial (tanpa *multithread*). Pengujian pertama menggunakan komputer dengan spesifikasi sebagai berikut, atau dapat dilihat pada Gambar 4.20.

Sistem Operasi : Windows 7 Home Premium 32-bit

Processor : Intel(R) Core(TM) i5-2410M CPU @2.30GHz (4CPUs)

Memory : 4096MB RAM



Gambar 4.20 Spesifikasi Komputer Percobaan I

Hasil dari pengujian pertama ini dapat dilihat pada Tabel 4.1. Dalam Tabel 4.1 percobaan dengan proses tanpa *multithread* pada teks 60 halaman sampai 100 halaman mengalami kekurangan *memory* sehingga proses tidak selesai, atau berhenti saat proses berlangsung. *Error* tersebut terjadi karena tidak cukupnya *memory* untuk mengalokasikan objek dalam ***Java heap***.

Ketika sebuah program Java dimulai, *Java Virtual Machine* mendapatkan sejumlah memori dari Sistem Operasi. *Java Virtual Machine* atau disingkat *JVM* menggunakan memori ini untuk semua kebutuhan. Sebagian dari memori ini disebut memori *java heap*. *Heap* di dalam java umumnya terletak ruang alamat paling bawah dan kemudian menaik keatas. Setiap kali kita membuat objek menggunakan operator baru atau dengan yang lain, berarti sebuah objek dialokasikan dari memori *Heap* dan ketika objek tersebut telah mati dan dikumpulkan oleh *Garbage collector*, maka sejumlah memori ini dikembalikan ke *heap space*. (Oracle and/or its affiliates, 2010)

Tabel 4.1 Hasil Pengujian Pada Komputer dengan empat inti dan memory 4GB RAM

No	Banyak Halaman	Waktu Komputasi (detik)		Selisih Waktu (detik)
		Dengan Multithread	Tanpa Multithread	
1	10	239.4	2690	2450.6
2	20	482.8	17036	16553.2
3	30	859.5	41722	40862.5
4	40	780.9	61913	61132.1
5	50	1651	121080	119429
6	60	2583	*Out of Memory*	2583
7	70	2704	*Out of Memory*	2704
8	80	3666	*Out of Memory*	3666
9	90	5407	*Out of Memory*	5407
10	100	5849	*Out of Memory*	5849

Ukuran standar heap space di java umumnya adalah 128Mb pada sebagian besar JVM Sun 32 bit, namun bisa sangat bervariasi dari satu jenis JVM ke JVM lain. Misalnya maximum heap space dan initial heap untuk Sistem Operasi 32-bit Solaris (Platform SPARC Edition) adalah -Xms = 3670K dan -Xmx = 64M dan nilai-nilai default dari parameter *heap space* pada sistem 64-bit telah meningkat sekitar 30%. Juga jika menggunakan *garbage collector* pada *Java* 1.5 ukuran maximum heap space *JVM* adalah Memori Fisik / 4 dan ukuran *initial heap space* adalah sama dengan ukuran Memori fisik /16. Cara lain untuk mendapatkan *heap space default* pada *JVM* adalah dengan menjalankan aplikasi menggunakan parameter *heap default* dan memonitor dengan menggunakan *JConsole* yang

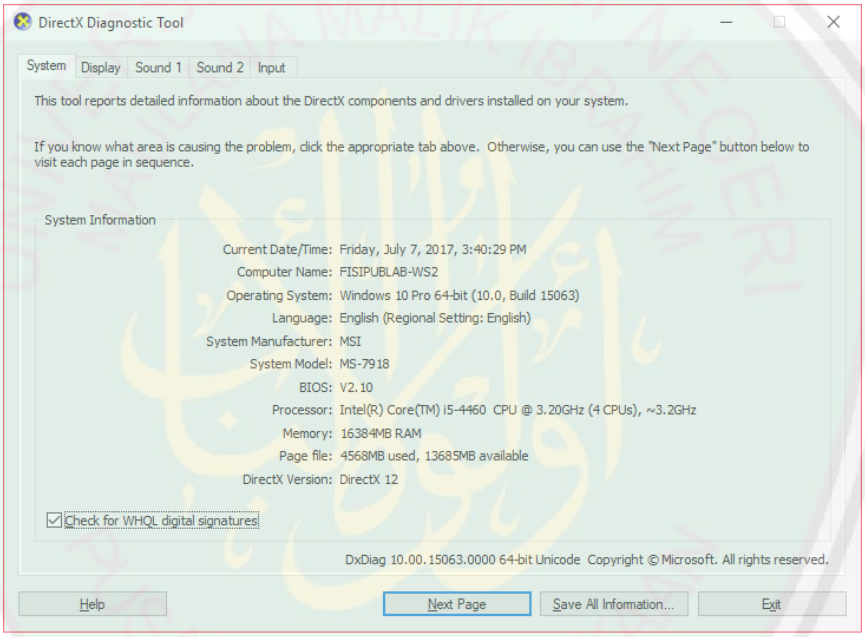
tersedia di JDK 1,5 ke atas, pada tab *VMSummary* akan dapat terlihat ukuran *maximum heap*. (Oracle and/or its affiliates, 2010)

Pengujian juga dilakukan pada komputer dengan spesifikasi sebagai berikut atau dapat dilihat pada Gambar 4.7.

Sistem Operasi : Windows 10 Pro 64-bit

Processor : Intel(R) Core(TM) i5-4460 CPU @3.20GHz (4CPUs)

Memory : 16384MB RAM



Gambar 4.21 Spesifikasi Komputer Percobaan II

Hasil dari pengujian dengan komputer dengan spesifikasi seperti pada Gambar 4.21 dapat dilihat pada Tabel 4.2.

Tabel 4.2 Hasil Pengujian Pada Komputer dengan empat inti dan memory 16GB RAM

No	Banyak Halaman	Waktu Komputasi (detik)		Selisih Waktu (detik)
		Dengan Multithread	Tanpa Multithread	
1	10	81.21	895.3	814.09

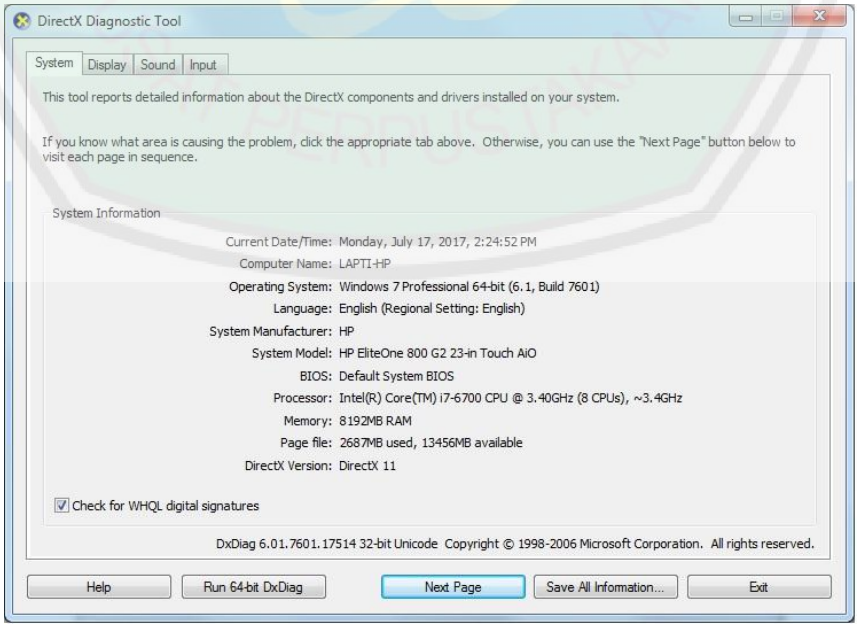
2	20	186.5	3715	3528.5
3	30	314.6	11242	10927.4
4	40	336.4	13146	12809.6
5	50	539.5	26501	25961.5
6	60	651.6	35618	34966.4
7	70	727.6	49033	48305.4
8	80	936.5	55396	54459.5
9	90	1211	78647	77436
10	100	1289	103060	101771

Pengujian ketiga dilakukan pada komputer dengan spesifikasi sebagai berikut atau dapat dilihat pada Gambar 6.8.

Sistem Operasi : Windows 7 Professional 64-bit

Processor : Intel(R) Core(TM) i7-6700 CPU @3.40GHz (8 CPUs)

Memory : 8192MB RAM



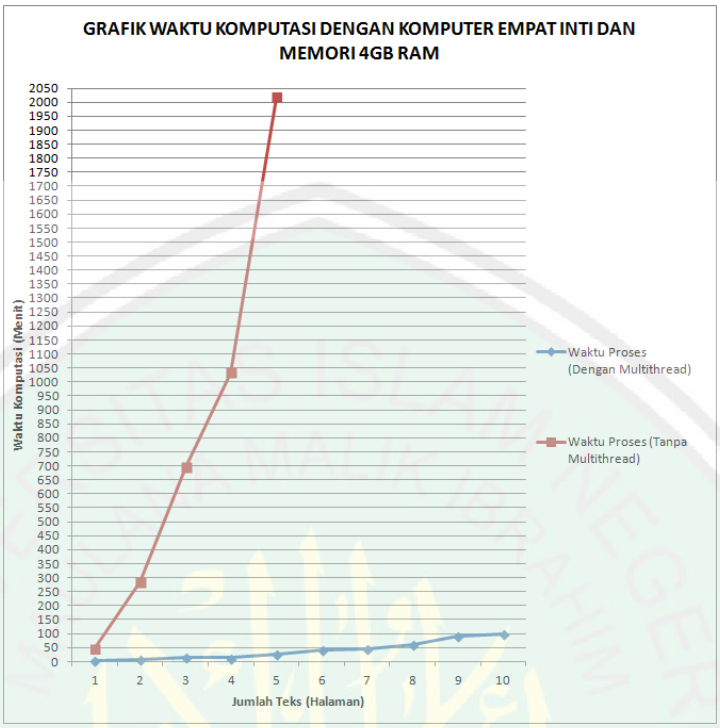
Gambar 4.22 Spesifikasi Komputer Percobaan III

Hasil pengujian dari komputer dengan spesifikasi di atas dapat dilihat pada Tabel 4.3

Tabel 4.3 Hasil Pengujian Pada Komputer dengan delapan inti dan memory 8GB RAM

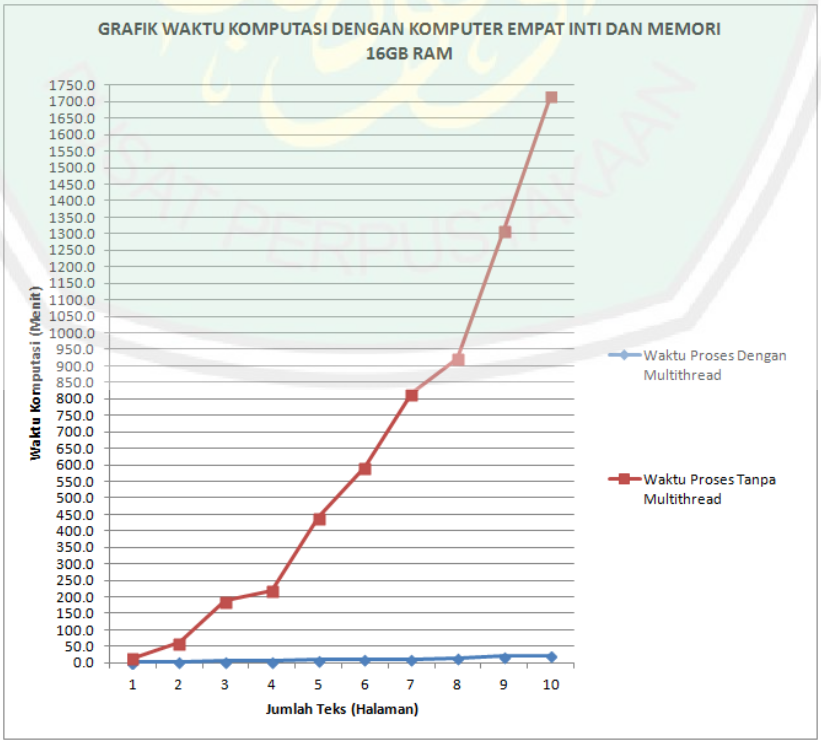
No	Banyak Halaman	Waktu Komputasi (detik)		Selisih Waktu (detik)
		Dengan Multithread	Tanpa Multithread	
1	10	58.14	634.5	576.36
2	20	141.6	2340	2198.4
3	30	215.6	6043	5827.4
4	40	244.4	9474	9229.6
5	50	394.5	17607	17212.5
6	60	432.2	22263	21830.8
7	70	487	26130	25643
8	80	676.2	43119	42442.8
9	90	818.5	57843	57024.5
10	100	889.2	59548	58658.8

Dari Tabel 4.1 diperoleh grafik seperti pada Gambar 4.23



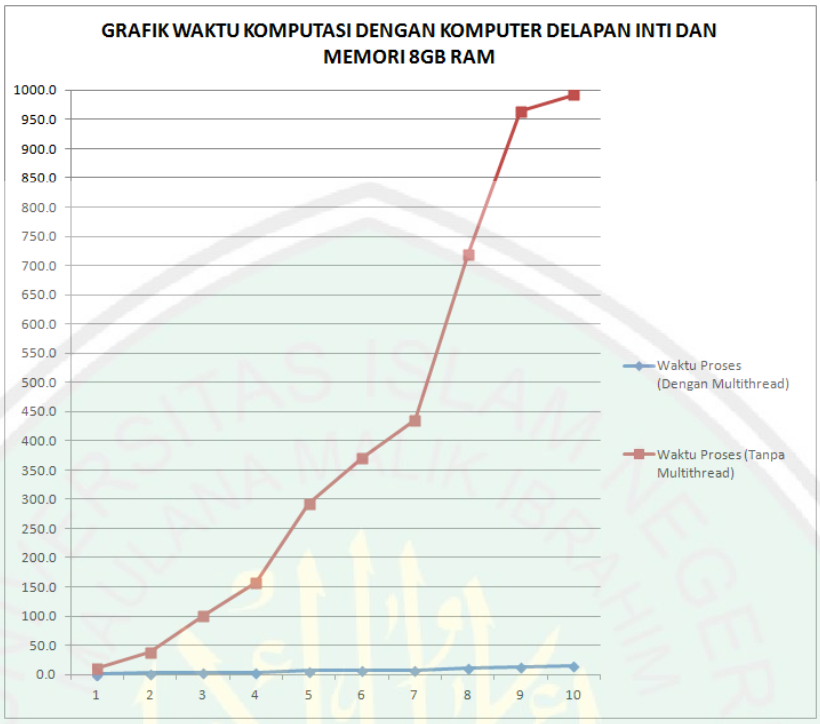
Gambar 4.23 Grafik waktu komputasi dengan komputer empat inti dan memori 4GB RAM

Dari Tabel 4.2 diperoleh grafik seperti pada Gambar 4.24



Gambar 4.24 Grafik waktu komputasi dengan komputer 4 inti dan 16GB RAM

Dari Tabel 4.3 diperoleh grafik seperti pada Gambar 4.25.



Gambar 4.25 Grafik waktu komputasi dengan komputer 8 inti 8GB RAM

Pada grafik yang ditunjukkan oleh Gambar 4.1 sampai Gambar 4.3 terdapat perbedaan garis merah dan biru yang sangat jauh berbeda. Garis merah (yang menunjukkan waktu proses tanpa *multithread*) pada grafik menunjukkan kenaikan waktu yang signifikan dibanding garis biru (yang menunjukkan waktu proses menggunakan *multithread*). Semakin tinggi garis semakin banyak pula waktu yang dibutuhkan oleh CPU untuk melakukan eksekusi program. Selain itu perbedaan juga dipengaruhi jumlah memori dan jenis cpu yang digunakan. Semakin besar memori maka semakin kecil kemungkinan terjadi *Out of memory* pada komputer. Hal ini dikarenakan *memory* yang dibutuhkan untuk proses semakin besar.

Untuk menghitung seberapa efisien implementasi *parallel computing multithread* maka dihitunglah persentase dari selisih waktu dari hasil percobaan yang telah dilakukan. Penghitungan presentase dilakukan dengan membandingkan

nilai dari selisih waktu komputasi dengan nilai dari waktu komputasi sebelum implementasi *multithread* yang mana dapat dilihat pada rumus berikut (Microsoft, 2018a):

$$\text{Persentase Selisih Waktu} = \left|\frac{N2-N1}{N1} \times 100\%\right|$$

Dimana,

N1 = Nilai Waktu Komputasi tanpa *multithread*

N2 = Nilai Waktu Komputasi dengan *multithread*

Tabel 4.4 Rata-rata selisih waktu komputasi per jumlah halaman

Rata-Rata Selisih Waktu Komputasi Per Jumlah Halaman								
Jumlah Halaman	corei5 4GB		corei5 16GB		corei7 8GB		Rata-rata	
	Selisih Waktu	Persentase	Selisih Waktu	Persentase	Selisih Waktu (detik)	Persentase Selisih Waktu	Selisih Waktu (detik)	Persentase Selisih Waktu
10	2450.6	91.10%	2440.6	90.93%	490.98	90.84%	1794.06	90.95%
20	16553.2	97.17%	16533.2	94.98%	2294.5	93.95%	11793.63	95.36%
30	40862.5	97.94%	40832.5	97.20%	6815.3	96.43%	29503.43	97.19%
40	61132.1	98.74%	61092.1	97.44%	8467.9	97.42%	43564.03	97.87%
50	119429	98.64%	119379	97.96%	15849.7	97.76%	84885.9	98.12%
60	2583	100%	2523	98.17%	22710.8	98.06%	9272.27	98.74%
70	2704	100%	2634	98.52%	30423.5	98.17%	11920.5	98.88%
80	3666	100%	3586	98.31%	44563	98.43%	17271.67	98.91%
90	5407	100%	5317	98.46%	58707.7	98.58%	23143.9	99.01%
100	5849	100%	5749	98.75%	61834.4	98.50%	24477.47	99.08%

Dari Tabel 4.4 didapatkan rata-rata persentase selisih waktu komputasi per halaman. Dari rata-rata tersebut dapat digunakan untuk mendapatkan rata-rata selisih seluruh waktu komputasi antara komputasi dengan menggunakan multithread dan tanpa multithread. Untuk mendapatkan persentase rata-rata selisih keseluruhan digunakan rumus berikut (Microsoft, 2018b):

$$Rata - rata\ persentase\ = \frac{P1 + P2 + P3 + \cdots + Pn}{n}$$

Dimana,

P1, P2, P3 …Pn = Persentase selisih waktu pada data ke 1 sampai *n*

n = jumlah data uji

Dari rumus nilai rata-rata tersebut dapat dihitung nilai rata-rata persentase dari selisih waktu komputasi sebagaimana berikut:

$$= \frac{90.95\% + 95.36\% + 97.19\% + 97.87\% + 98.12\% + 98.12\% + 98.74\% + 98.88\% + 98.91\% + 99.01\% + 99.08\%}{10}$$

$$= \frac{974.14\%}{10}$$

$$= 97.41\%$$

Dari penghitungan dengan rumus nilai rata-rata didapatkan bahwa nilai rata-rata persentase selisih waktu komputasi dengan menggunakan multithread dan tanpa multithread adalah 97.41%.

4.5 Integrasi Implementasi *Parallel Computing* pada Aplikasi Transliterasi Aksara Jawa dengan Al-Qur'an

Al-qur'an merupakan salah satu pedoman hidup bagi manusia, dimana Al-qur'an mengajarkan berbagai macam aspek seperti halnya agama, ekonomi, sosial

dan lain sebagainya. Bahkan keberagaman bahasa dan suku – suku telah diterangkan di dalam Al-qur'an Q.S Ar-Ruum ayat 22.

وَمِنْ آيَاتِهِ خَلْقُ السَّمَوَاتِ وَالأَرْضِ وَاخْتِلافُ أَلْسِنَتِكُمْ وَأَلْوَانِكُمْ إِنَّ فِي ذَلِكَ لآيَاتٍ لِلْعَالِمِينَ

Artinya: Dan di antara tanda-tanda kekuasaan-Nya ialah menciptakan langit dan bumi dan berlain-lainan bahasamu dan warna kulitmu. Sesungguhnya pada yang demikan itu benar-benar terdapat tanda-tanda bagi orang-orang yang Mengetahui.

Didalam tafsir Hidayatu Insan dijelaskan bahwa maksud dari kalimat وَاخْتِلافُ أَلْسِنَتِكُمْ وَأَلْوَانِكُمْ adalah meskipun asalnya hanya satu dan tempat keluarnya huruf juga satu, namun terdapat sedikit atau banyak perbedaan antara suara dan warna kulit yang membedakan antara yang satu dengan yang lain, hal ini merupakan bentuk dari kesempurnaan kekuasaan Allah SWT. Berdasarkan ayat dan tafsir tersebut dapat diartikan bahwa aksara jawa merupakan salah satu dari sekian banyak bahasa dan aksara yang telah di kehendaki oleh Allah SWT sebagai salah satu bahasa dalam berkomunikasi antar sesama.

Dalam kehidupan ini waktu juga merupakan salah satu aspek yang begitu penting. Dengan waktu segala pekerjaan dapat dikerjakan dan diselesaikan. Allah Swt. berfirman dalam al-Quran surat al-Furqan ayat 62

وَهُوَ الَّذِي جَعَلَ اللَّيْلَ وَالنَّهَارَ خِلْفَةً لِمَنْ أَرَادَ أَنْ يَذَّكَّرَ أَوْ أَرَادَ شُكُورًا (٦٢)

Artinya: Dan Dia (pula) yang menjadikan malam dan siang silih berganti bagi orang yang ingin mengambil pelajaran atau orang yang ingin bersyukur.(Q.S. AlFurqan/25:62)

Ayat ini menerangkan bahwa dengan waktu tersebut dapat mengambil pelajaran dan bersyukur. Allah juga berfirman dalam al-Quran surat Yunus ayat 24:

إِنَّمَا مَثَلُ الْحَيَاةِ الدُّنْيَا كَمَاءٍ أَنزَلْنَاهُ مِنَ السَّمَاءِ فَاخْتَلَطَ بِهِ نَبَاتُ الأَرْضِ مِمَّا يَأْكُلُ النَّاسُ وَالأَنْعَامُ حَتَّى إِذَا أَخَذَتِ الأَرْضُ زُخْرُفَهَا وَازَّيَّنَتْ وَظَنَّ أَهْلُهَا أَنَّهُمْ قَادِرُونَ عَلَيْهَا أَتَاهَا أَمْرُنَا لَيْلا أَوْ نَهَارًا فَجَعَلْنَاهَا حَصِيدًا كَأَنْ لَمْ تَغْنَ بِالأَمْسِ كَذَلِكَ نُفَصِّلُ الآيَاتِ لِقَوْمٍ يَتَفَكَّرُونَ (٢٤)

Artinya: Sesungguhnya perumpamaan kehidupan duniawi itu adalah seperti air (hujan) yang Kami turunkan dari langit, lalu tumbuhlah dengan suburnya karena air itu tanam-tanaman bumi, di antaranya ada yang dimakan manusia dan binatang ternak. Hingga apabila bumi itu telah sempurna keindahannya, dan memakai (pula) perhiasannya, dan pemilik-pemiliknya mengira bahwa mereka pasti menguasainya, tiba-tiba datanglah kepadanya azab Kami di waktu malam atau siang, lalu Kami jadikan (tanam-tanaman) laksana tanam-tanaman yang sudah disabit, seakan-akan belum pernah tumbuh kemarin. Demikianlah Kami menjelaskan tanda-tanda kekuasaan (Kami) kepada orang-orang yang berpikir. (Q.S. Yunus/10:24)

Dalam ayat ini manusia dianjurkan untuk terus berfikir mengenai pemanfaatan segala sesuatu yang telah Allah ciptakan di bumi. Ayat tersebut juga menganjurkan kepada manusia untuk terus menggali dan memperhatikan apa saja yang ada di dalam semesta ini. Berdasarkan ayat ini dengan memanfaatkan teknologi informasi yang semakin berkembang kita dapat menerapkannya seperti untuk mengoptimalkan komputasi pada aplikasi transliterasi aksara Jawa dengan

menggunakan *parallel computing* agar waktu prosesnya menjadi lebih cepat. Dalam al-Quran surat Ali Imran ayat 146 Allah Swt. berfirman,

وَكَأَيِّنْ مِنْ نَبِيٍّ قَاتَلَ مَعَهُ رِبِّيُّونَ كَثِيرٌ فَمَا وَهَنُوا لِمَا أَصَابَهُمْ فِي سَبِيلِ اللَّهِ وَمَا ضَعُفُوا وَمَا اسْتَكَانُوا وَاللَّهُ يُحِبُّ الصَّابِرِينَ (١٤٦)

Artinya: Dan berapa banyak nabi yang berperang bersama-sama mereka sejumlah besar dari pengikut (nya) yang bertakwa. Mereka tidak menjadi lemah karena bencana yang menimpa mereka di jalan Allah, dan tidak lesu dan tidak (pula) menyerah kepada musuh). Allah menyukai orang-orang yang sabar. (Q.S. Ali Imran/3:146)

Ayat diatas menceritakan bahwa dengan mengerjakan sesuatu secara bersama-sama menjadikan pekerjaan menjadi lebih mudah diatasi dan hasil pekerjaan menjadi lebih baik, yang dalam penelitian ini diimplementasikan menggunakan *parallel computing*. *Parallel computing* menjadikan proses menjadi lebih cepat dan dapat memproses lebih banyak teks dari pada tidak menggunakan *parallel computing*. Allah juga berfirman dalam Al-Qur'an surat Al-Insyirah/94 ayat 7.

فَإِذَا فَرَغْتَ فَانْصَبْ (7)

Artinya: Maka apabila kamu telah selesai (dari sesuatu urusan), kerjakanlah dengan sungguh-sungguh (urusan) yang lain.

Ayat diatas juga mengungkapkan betapa pentingnya waktu. Untuk itu setiap kita selesai melakukan suatu urusan atau pekerjaan segeralah menuju ke urusan dan pekerjaan berikutnya. Ayat diatas sesuai dengan konsep *multithread* yang langsung mengerjakan satu pekerjaan setelah pekerjaan lain terselesaikan.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan penelitian yang telah dikerjakan serta uji coba yang telah dilakukan, maka didapatkan kesimpulan sebagai berikut:

1. *Parallel computing* model pemrograman *multithread* dapat diterapkan untuk mempercepat waktu komputasi proses pada aplikasi alih aksara Latin ke dalam aksara Jawa.
2. Berdasarkan hasil pengujian, penerapan parallel computing pada aplikasi transliterasi aksara Jawa ini dapat mengurangi waktu proses sebanyak 97.41%.

5.2 Saran

Penerapan parallel computing pada aplikasi transliterasi aksara Jawa pada penelitian ini tentunya tidaklah luput dari kesalahan dan kekurangan. Oleh karena itu penulis menyertakan beberapa saran untuk dapat digunakan sebagai pengembangan sistem yang lebih baik dikemudian hari, diantaranya:

1. Pemotongan kata masih terbatas pada titik setelah batas pembagian, sehingga jumlah kata perbagian banyak yang melebihi batasnya, akan lebih baik jika pemotongan kata dapat dilakukan juga ke belakang agar jumlah kata tidak banyak melebihi batas.
2. Dapat dikembangkan pada *interface* agar lebih menarik dan lebih nyaman digunakan. Penelitian ini dapat dikembangkan lagi dengan pembuatan versi mobile atau web agar dapat digunakan secara lebih fleksible dan efisien.

DAFTAR PUSTAKA

Atina, Vihi dkk. 2012. *Program Transliterasi Antara Aksara Latin dan Aksara Jawa dengan Metode FSA*. Jurnal ITSMART, Vol 1, No. 2, Desember 2012, ISSN: 2301-7201.

Ayumitha, Fevi Henda. 2014. *Transliterasi Huruf Latin ke Dalam Aksara Jawa Dengan Menggunakan Decision Tree*. Skripsi. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Baeney, Blaise. 2014. *Introduction to Parallel Computing*. https://computing.llnl.gov/tutorials/parallel_comp/. Tanggal akses 12 Januari 2015 pukul 21.44 wib.

Gozali, Ferrianto dan Lagusto, Dimas. 2005. *Analisis Untuk Kerja Komputasi Distributed Shared Memory Pada Sistem Cluster Komputer Personal*. Jurnal JETri Universitas Trisakti, Volume 4, Nomor 2, Februari 2005, Halaman 25-44, ISSN 1412-0372.

Kadir, Abdul. 2004. *Dasar Pemrograman Java 2*. Yogyakarta: Andi Offset.

Microsoft. 2018a. *Menghitung persentase.* https://support.office.com/id-id/article/menghitung-persentase-6b5506e9-125a-4aba-a638-d6b40e603981. Tanggal akses 10 Januari 2018 pukul 20.09 wib.

Microsoft. 2018b. *Menghitung rata-rata dari grup angka.* https://support.office.com/id-id/article/Menghitung-rata-rata-dari-grup-angka-e158ef61-421c-4839-8290-34d7b1e68283?ui=id-ID&rs=id-ID&ad=ID. Tanggal akses 10 Januari 2018 pukul 20.21 wib.

Mulya, Megah dan Abdiansyah. 2013. *Penerapan Multi-threading untuk Meningkatkan Kinerja Pengolahan Citra Digital*. Jurnal Generic, Vol. 8, No.2, September 2013, pp.230~237 ISSN: 1907-4093, e-ISSN: 2087-9814.

Oracle and/or its affiliates. 2014. *Thread (Java Platform SE 7)*. http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html. Tanggal akses 12 Januari 2015 pukul 22.28 wib.

Oaks, Scott dan Wong. 1999. *Java Threads, 2nd Edition*. ISBN: 1-56592-418-5.

Panca, I Gede Putu Agus Wiranata Panca. 2015. Komputasi Paralel Berbasis GPU CUDA Untuk Pengembangan Image Inpainting Dengan Metode Perona-Malik. Tugas Akhir. Program Studi Teknik Informatika Fakultas Teknologi Industri Universitas Atma Jay Yogyakarta.

Pardede, Jasman. 2014. *Implementasi Multithreading Untuk Meningkatkan Kinerja Information Retrieval Dengan Metode GVSM*. Jurnal Sistem Komputer – Vol 4, No.1, Mei 2014, ISSN: 2087-4685, e-ISSN: 2252-3456.

Salambue, Roni. 2013. *Pengenalan Pola Tanda Tangan dengan Metode Moment Ivariant dan Euclidian Distance.* Prosiding Semirata FMIPA Universitas Lampung.

Santosa, Budi dan Widyarini, Tiananda. 2009. *Aplikasi Metode Cross Entropy Untuk Support Vector Machines*. Jurnal Teknik Industri Universitas Muhammadiyah Malang – Vol 10, No 2, Agustus 2009, ISSN: 1978-1431, e-ISSN: 2527-4112.

Schildt, Herbert. 2014. *Java: The Complete Referece, 9th Edition*. New York: McGraw-Hill Education.

Seminar, Kudang B, Buono, Agus, dan Alim, Khawarismie. 2006. Uji dan Aplikasi Komputasi Paralel Pada Jaringan Syaraf Probabilistik (PNN) Untuk Proses Klasifikasi Mutu Tomat. Jurnal Teknologi, Edisi No 1, Tahun XX, Maret 2006, Halaman 34-45, ISSN 0215-1685.

Supeno, Handoko. 2015. Kombinasi Teknik Algoritma Genetika Untuk Mempercepat Munculnya Offspring (Studi Kasus Game Simulator Pelabuhan). Jurnal INFORMATEK Universitas Pasundan, Volume 17, Nomor 2, Desember 2015, Halaman 97 – 106, ISSN 1411-0865.

Utami, Ema. 2013. *Penerapan Rule Based dalam Membangun Transliterator JawaTex*. Berkala MIPA, 23(1), Januari 2013.