# IMPLEMENTASI KONVOLUSI BERBASIS KERNEL LAWS MENGGUNAKAN RENDERSCRIPT PADA ANDROID

**SKRIPSI**

**Oleh :**
**MUHAMMAD BAHRUL H**
**NIM. 12650070**

**JURUSAN TEKNIK INFORMATIKA**
**FAKULTAS SAINS DAN TEKNOLOGI**
**UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM**
**MALANG**
**2016**

**IMPLEMENTASI KONVOLUSI BERBASIS KERNEL LAWS MENGGUNAKAN RENDERSCRIPT PADA ANDROID**

# SKRIPSI

**Diajukan kepada:**

**Universitas Islam Negeri Maulana Malik Ibrahim Malang**

**Untuk memenuhi Salah Satu Persyaratan dalam**

**Memperoleh Gelar Sarjana Komputer (S.Kom)**

**Oleh :**

**MUHAMMAD BAHRUL H**

**NIM. 12650070**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS SAINS DAN TEKNOLOGI**

**UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG**

**2016**

## LEMBAR PERSETUJUAN

# IMPLEMENTASI KONVOLUSI BERBASIS KERNEL LAWS MENGGUNAKAN RENDERSCRIPT PADA ANDROID

**SKRIPSI**


**Oleh :**

**Muhammad Bahrul H**
**NIM. 12650070**


Telah Diperiksa dan Disetujui untuk Diuji
Tanggal : 1 September 2016

| Pembimbing I, | Pembimbing II, |
|---|---|
| Fatchurrochman, M.Kom | A'la Syauqi, M.Kom |
| NIP. 19700731 200501 1 002 | NIP. 19771201 200801 1 007 |

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

## LEMBAR PENGESAHAN

## IMPLEMENTASI KONVOLUSI BERBASIS KERNEL LAWS MENGGUNAKAN RENDERSCRIPT PADA ANDROID

**SKRIPSI**

**Oleh :**

**Muhammad Bahrul H**
**NIM. 12650070**

Telah Dipertahankan di Depan Dewan Penguji Skripsi
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Tanggal: 9 September 2016

| | | | |
|---|---|---|---|
| Penguji Utama | : | Dr. Cahyo Crysdian<br>NIP. 19740424 200901 1 008 | (.....) |
| Ketua Penguji | : | Dr. M. Amin Hariyadi, M.T<br>NIP. 19670118 200501 1 001 | (.....) |
| Sekertaris Penguji | : | Fatchurrochman, M.Kom<br>NIP. 19700731 200501 1 002 | (.....) |
| Anggota Penguji | : | A'la Syauqi, M.Kom<br>NIP. 19771201 200801 1 007 | (.....) |

Mengesahkan,
Ketua Jurusan Teknik Informatika
*Fakultas Sains dan Teknologi*
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

**SURAT PERNYATAAN**

**ORISINALITAS TULISAN**

Saya yang bertanda tangan di bawah ini:

| | |
|---|---|
| Nama | : Muhammad Bahrul H. |
| NIM | : 12650070 |
| Jurusan | : Teknik Informatika |
| Fakultas | : Sains dan Teknologi |
| Judul Penelitian | : Implementasi Konvolusi Berbasis Kernel LAWS Menggunakan Renderscript Pada Android |

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

Malang, 4 September 2016

Yang membuat pernyataan,


Muhammad Bahrul H.

NIM. 12650070

# *MOTTO*

*"Be Your Self"*

فَإِنَّ مَعَ الْعُسْرِ يُسْرًا ۝ إِنَّ مَعَ الْعُسْرِ يُسْرًا

*"Karena sesungguhnya sesudah kesulitan itu ada kemudahan.*

*Sesungguhnya sesudah kesulitan itu ada kemudahan."*

**(Asy Syarh: 5-**

# LEMBAR PERSEMBAHAN

*Dengan mengucap syukur Alhamdulillahirobbil'alamin…*

*Karya kecil ini kupersembahkan dan terima kasih untuk:*

*Ayah Ismanto dan Ibu Sudjiati yang telah mendidik dan menuntunku dengan penuh kesabaran dan kasih sayang, selalu memberikan dukungan moral dan do'a yang tiada henti. Semoga Rahmat dan Rahman Allah ﷻ selalu terlimpahkan kepada beliau.*

*Kakak dan adik tersayang, mbak Khilmi, dek Aan, dan dek Ismi yang memberikan doa, dorongan dan ejekan motivasi serta menjadi hiburan untukku.*

*Teman-teman jurusan Teknik Informatika angkatan 2012, terima kasih atas kerja sama, dukungan, dan bantuannya selama bangku perkuliahan dan penelitian. Teman-teman asrama SJK dan futsal Ti, terima kasih telah memberikan warna hidup dengan beribu kisah, kebersamaan, persaudaraan, pertengkaran dan ejekan selama perkuliahan. Dan Intan Fakhrun Ni'am yang selalu memberikan semangat, nasehat, motivasi dan meluangkan waktu untuk mendengarkan keluh kesah dan membantu selama ini. SEE YOU ON TOP.*

*Terima kasih kepada pembimbing bapak Zainal Abidin dan A'la Syauqi. Semoga senantiasa diberi umur panjang, sehat, dan ilmu yang telah diberikan mendapat barokah dari Allah ﷻ*

## KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

Puji syukur kehadirat Allah ﷻ yang telah melimpahkan rahmat, taufiq dan hidayah serta inayah-Nya tiada henti dan tiada terbatas kepada penulis, sehingga penulis dapat menyelesaikan skripsi ini dengan baik dan lancar. Shalawat dan salam semoga tetap terucap kepada Nabi Muhammad ﷺ yang telah membimbing dan menuntun manusia ke jalan yang benar.

Skripsi dengan judul "Implementasi Komputasi Paralel Metode Laws untuk Ekstraksi Fitur Tekstur Citra menggunakan Renderscript pada Android" dapat disusun dan diselesaikan dengan baik karena dukungan, motivasi serta bimbingan dari berbagai pihak. Tidak ada kata dan perbuatan yang patut terucap dan terlihat untuk menguntai sedikit makna kebahagiaan diri. Oleh karena itu, izinkan penulis mengucapkan banyak terimakasih kepada:

1. Prof. Dr. H. Mudjia Rahardjo, M.Si selaku Rektor Universitas Islam Negeri Maulana Malik Ibrahim Malang.
2. Dr. drh. Hj. Bayyinatul Muchtaromah, M. Si selaku Dekan Fakultas Sains dan Teknologi UIN Maliki Malang.
3. Dr. Cahyo Crysdian, M.Cs selaku Ketua Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maliki Malang.
4. Zainal Abidin, M.Kom selaku Dosen Pembimbing I yang telah memberikan bimbingan, arahan, motivasi dan meluangkan waktu sehingga penulis dapat menyelesaikan penulisan skripsi ini.

5. A'la Syauqi, M.Kom selaku Dosen Pembimbing II yang telah meluangkan waktunya, menyalurkan ilmunya serta bimbingannya sehingga penulisan skripsi ini dapat terselesaikan.
6. Muhammad Faisal, M.T selaku Dosen Wali yang telah memberikan arahan dan nasehat selama perkuliahan sehingga penulisn skripsi ini dapat terselesaikan.
7. Segenap Dosen Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maliki Malang yang telah membimbing dan memberikan ilmunya dengan penuh kesabaran dan keikhlasan.
8. Seluruh staf laboratorium Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maliki Malang yang telah meluangkan waktunya untuk membantu dan menemani kinerja selama penelitian berlangsung sehingga skripsi ini dapat terselesaikan.
9. Ayah dan Ibu tercinta, Ayah Ismanto, Ibu Sudjiati yang selalu memberikan dukungan moril maupun spiritual serta ketulusan do'anya sehingga penulisan skripsi dapat terselesaikan. Semoga Rahman dan Rahim Allah ﷻ selalu menaungi mereka. Amiin.
10. Kakak dan adik tersayang, mbak Khilmi, dek Aan, dan dek Ismi yang memberikan doa, dorongan dan ejekan motivasi serta memberikan hiburan kepada penulis dalam proses penyelesaian skripsi ini. Semoga Rahman dan Rahim Allah ﷻ selalu menaungi mereka. Amiin.
11. Intan Fakhrun Ni'am yang selalu memberikan semangat, nasehat, motivasi dan meluangkan waktu untuk mendengarkan keluh kesah dan membantu penulis sehingga skripsi ini dapat terselesaikan.

12. Teman-teman jurusan Teknik Informatika angkatan 2012 khususnya Bindar, Imam, Misbah, Faik, Sakinah, Kiki, dan Afifah, terima kasih atas kerja sama, dukungan, dan bantuannya selama menempuh studi di Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Malang. Semoga kita semua menjadi insan yang bermanfaat. Amiin.
13. Teman-teman asrama SJK yang tidak bisa saya sebutkan satu persatu, terima kasih telah memberikan warna hidup dengan beribu kisah, kebersamaan, persaudaraan, pertengkaran dan ejekan selama perkuliahan. Semoga kita sukses bersama.
14. Teman-teman futsal Ti, terima kasih telah memberikan keceriaan dan guyonan selama jeda perkuliahan. Semoga pertemanan kita tetap utuh walau terpisah waktu dan tempat.

Tidak ada kata yang patut terucap selain ucapan terimakasih yang sebesar-besarnya dan doa semoga amal baik mereka mendapat ridho dari Allah ﷻ. Penulis menyadari akan banyaknya kekurangan dalam skripsi ini dan semoga skripsi ini dapat bermanfaat bagi diri penulis dan semua pembaca.

Wassalamu'alaikum Wr. Wb.

Malang, 4 September 2016

Penulis

## DAFTAR ISI

## DAFTAR TABEL

## DAFTAR GAMBAR

## DAFTAR LAMPIRAN

## ABSTRAK

Hidayatullah, Muhammad Bahrul. 2016. Implementasi Konvolusi Berbasis Kernel LAWS Menggunakan Renderscript Pada Android. Skripsi. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Pembimbing : Fatchurrochman, M.Kom, A'la Syauqi, M.Kom.

**Kata Kunci :** *Renderscript*, LAWS.

Permasalahan kecepatan dalam komputasi telah menjadi topik utama dalam penelitian-penelitian terdahulu. Sebagai solusi dalam permasalahan kecepatan proses komputasi, beberapa pendekatan dilakukan dalam penelitian, salah satunya dengan sistem pemrosesan paralel. Paralelisasi atau penggunaan sistem pemrosesan paralel bisa diterapkan dalam permasalahan komputasi dalam bidang pengolahan citra, komputasi fotografi dan *computer vision.* Model pemrograman yang diusulkan dalam penelitian ini adalah *Renderscript* dengan menggunakan perangkat mobile berbasis android. Pada penelitian ini penulis mencoba mengimplementasikan filter tekstur LAWS untuk ekstraksi fitur energi tekstur pada citra. Implementasi Metode LAWS pada komputasi paralel dibagi menjadi tiga tahap yaitu tahap konvolusi citra dengan 25 pasangan kernel LAWS, tahap pergeseran *window* dengan ukuran 15x15 dan penormalan nilai piksel. Pembagian *thread* yang digunakan dalam pemrosesan citra disesuaikan dengan jumlah inti pemroses GPU perangkat android ponsel pintar Xiamo Redmi 2. Data yang digunakan pada penelitian ini berupa citra tunggal daun cocor bebek dengan tiga ukuran berbeda, yaitu 512x512, 1024x1024 dan 2048x2048. Hasil yang didapatkan dari data menunjukkan waktu komputasi paralel pada ukuran 512x512, 1024x1024 dan 2048x2048 dengan nilai rata-rata pengukuran 470,7ms, 1731,8ms dan 5416,06. Sedangkan hasil pengukuran komputasi sekuensial dengan nilai rata-rata 5695,3ms, 25103,6ms dan 82558,53ms. Dari hasil perbandingan pengukuran waktu komputasi menunjukkan komputasi paralel lebih cepat dibandingkan dengan komputasi sekuensial dengan peningkatan kecepatan pada ukuran 512x512,1024x1024 dan 2048x2048 mencapai 12,72 , 14,54, 15,24. Efisiensi waktu komputasi paralel menunjukkan nilai peningkatan konsumsi waktu 92% pada semua ukuran dari komputasi sekuensial.

## ABSTRACT

Hidayatullah, Muhammad Bahrul. 2016. Implementation of Convolution LAWS Kernel Based Using Renderscript for Android. Thesis. Department of Informatics Faculty of Science and Technology State Islamic University of Maulana Malik Ibrahim Malang.

Advisor : Fatchurrochman, M.Kom, A'la Syauqi, M.Kom.

**keyword :** *Renderscript*, LAWS.

Problems in computing speed has become a major topic in previous studies. Some of approaches are made in the research, one of them is a system of parallel processing as a solution to the problem of computing processing speed. Parallelization or the use of parallel processing system can be applied in computing problems in the field of image processing, computer vision and computational photography. The programming model proposed in this study is Renderscript using android based mobile devices. In this study, the author tried to implement LAWS texture filter for texture extraction energy features in the image. Implementation LAWS method on parallel computing is divided into three phases: the convolution kernel image with 25 pairs LAWS, phase shift in the window with a size of 15x15 and normalizing the pixel values. The division of thread used in image processing adapted to the number of processors GPU core android smart phone devices Xiaomi redmi 2. The data used in this study of a single image cocor bebek leaf with three different sizes, ie 512x512, 1024x1024 and 2048x2048. The results obtained from the data indicating the time of parallel computing on the size of 512x512, 1024x1024 and 2048x2048 with an average value of measurement 470,7ms, 1731,8ms and 5416.06. While the sequential computing measurement results with the average value 5695,3ms, 25103,6ms and 82558,53ms. From the comparison of computing time measurements shows that parallel computing faster than the sequential computing with speed increases in size 512x512,1024x1024 and 2048x2048 reached 12.72, 14.54, 15.24. The efficiency of parallel computing time shows the value of a 92% increase in consumption in all sizes from sequential computing.

# مستخلص البحث

هداية الله، محمّد بحرل. 2016. تنفيد الحوسبة المتوازية بطريقة قوانين (LAWS) لميزات طاقة استخراج نسيج الصورة باستخدام (*Renderscript*) في الروبوت. البحث الجامعي. قسم الهندسة المعلوماتية بكليّة العلوم والتكنولوجيا بجامعة مولانا مالك إبراهيم الإسلامية الحكومية بمالانج.

المشرف: الدكتور زين العابدين الماجستير، الدكتور أعلى شوقي الماجستير.

---

**الكلمات المفتاحية**: الحوسبة المتوازية، (*Renderscript*)، قوانين (LAWS)، استخراج ميزة النص.

أصبحت مشكلات السرعة في الحوسبة موضوعا رئيسيا في الدراسات السابقة. لحلّ في مشكلات سرعة العملية الحاسوبية، جعلت بعض المقاربة في البحث، أحدها بنظام العملية المتوازية. المتوازية أو استخدام نظام العملية المتوازية يمكن تطبيقها في المشكلات الحاسوبية في مجال معالجة الصور والتصوير الحسابي و(*computer vision*). اسلوب البرمجة المقترحة في هذه البحث هو *Renderscript* باستخدام الجهاز المحلول الروبوت. في هذا البحث، الباحث يحاول أن يطبّق ميزات الملمس LAWS لميزات طاقة استخراج نسيج في الصورة. وينقسم تطبيق أسلوب القوانين على الحاسوبية المتوازية إلى ثلاث مراحل: مرحلة صورة تفاف النواة مع 25 قرينة LAWS، مرحلة تحول نافذة بحجم 15x15وتطبيع قيم بكسل (piksel). تقسيم موضوع المستخدمة في معالجة الصور وفقا بعدد من المعالجات الأساسية GPU الروبوت الهواتف الذكية XIAOMI redmi 2. البيانات المستخدمة في هذا البحث من ورقة صورة بطة واحدة مع ثلاثة أحجام مختلفة، هي 512 x512، 1024 x1024، 2048 x2048.، الاختبارات التي يقوم بها أحد القوانين مضروبا بيانات الصورة 30 ورقة في جميع الأحجام. النتائج من البيانات تشير إلى الوقت الحوسبة المتوازية على حجم 512 x512، 1024 x1024، و2048 x2048.، بقيمة متوسط من 470،7ms، 1731،8ms، و5416،06. وأما نتائج قياس الحوسبة المتتابعة بقيمة متوسط 5695،3ms، 25013،6ms، 82558،53ms. من نتائج مقارنة قياس وقت الحوسبة تدلّ الحوسبة المتوازية أسرع من زيادة سرعة حساب متتابعة في حجم 512 x512، 1024 x1024، و2048 x2048.، صلت 12،72. 14،54. 15،24. كفاءة من الوقت الحوسبة المتوازية تدلّ قيمة الزيادة 92% في استهلاك في جميع الأحجام من الحوسبة المتتابعة.

# BAB I

# PENDAHULUAN

## 1.1 Latar Belakang

Waktu merupakan sesuatu yang sangat penting bagi kehidupan manusia. Hal ini ditunjukkan dengan firman Allah ﷻ yang bersumpah dengan nama waktu dalam banyak ayat Al-Qur'an. Salah satunya terdapat pada ayat Al-Quran surat Al-'Ashar 1-2 yang berbunyi :

وَٱلۡعَصۡرِ ۝١ إِنَّ ٱلۡإِنسَٰنَ لَفِي خُسۡرٍ ۝٢

Artinya : "*1. demi masa. 2. Sesungguhnya manusia itu benar-benar dalam kerugian*"

Ayat diatas menunjukkan bahwa kemulian dan keagungan diberikan oleh Allah ﷻ kepada waktu. Dari ayat diatas dapat diambil pelajaran bahwa waktu sangat berharga bagi kehidupan manusia. Dalam hal komputasi, waktu menjadi bagian penting dan telah menjadi topik penelitian-penelitian terdahulu dalam usaha untuk meningkatkan efisiensi waktu komputasi.

Permasalahan kecepatan dan konsumsi waktu dalam komputasi telah menjadi topik utama dalam penelitian-penelitian terdahulu (Matloff Norman, 2008). Sebagai solusi dalam permasalahan kecepatan proses komputasi, beberapa pendekatan dilakukan dalam penelitian. Salah satunya dengan sistem pemrosesan paralel (Z.A. Ahmad dkk, 2012). Penambahan kecepatan dalam proses komputasi adalah kunci utama dalam penggunaan sistem pemrosesan paralel. Pada penelitian yang dilakukan oleh Z.A. Ahmad dkk (2012) mengemukakan salah satu pendekatan dalam sistem pemrosesan paralel adalah dengan

mengimplementasikan *multiprocessing* dalam satu *processor* (perangkat pemroses). Dalam perkembangannya, teknologi *processor* dalam perangkat Android telah berevolusi. Dimana arsitektur *multicore processor* dan GPU (Graphics Processing Unit) telah diterapkan (Alejandro dan Fransisco, 2015). Oleh karena itu penerapan sistem pemrosesan paralel dalam perangkat Android sudah bisa diimplementasikan.

Paralelisasi atau penggunaan sistem pemrosesan paralel bisa diterapkan dalam permasalahan komputasi dalam bidang pengolahan citra, komputasi fotografi dan *computer vision* (Alejandro dan Fransisco, 2015). Pada bidang pengolahan citra metode analisa tekstur telah dikembangkan selama dua dekade sebelum tahun 1980 (Dong-Cheon dan Toni, 1992). Penelitian yang dilakukan Dong-Cheon dan Toni (1992) menyebutkan pendekatan Laws pada tekstur telah menjadi metode yang memadai dalam metode analisa tekstur, hal ini dikemukakan juga oleh Ballard dan Grown pada tahun 1982, Gool dkk tahun 1985, Gong dan Huang tahun 1988, Unser dan Eden tahun 1990 yang mengembangkan metode laws dari disertasi yang dilakukan oleh Laws (1980). Dalam penerapannya metode laws merupakan sebuah kernel yang diproses menggunakan konvolusi pada citra sehingga didapatkan energi tekstur dari citra. Masalah akan timbul dalam proses konvolusi apabila matriks kernel berukuran besar yang membutuhkan waktu komputasi yang lama (Abdul Kadir dan Adhi Susanto, 2013).

Beberapa model pemrograman telah diusulkan dalam penelitian dalam penggunaan sistem pemrosesan paralel seperti *Compute Unified Device Architecture* (CUDA) dan *Open Computing Language* (OpenCL), akan tetapi tidak semua perangkat android sesuai dengan dua *framework language* tersebut

(Alejandro dan Fransisco, 2015). Pada penelitian lainnya, Roelof Kemp dkk (2013), mengusulkan RCUDA (Remote CUDA) berbasis *computation offloading* dengan mendistribusikan komputasi dari perangkat android pada server CUDA, akan tetapi RCUDA masih belum bisa menambah kecepatan komputasi pada algoritma yang diimplementasikan.

Untuk mengatasi permasalahan keberagaman arsitektur pemrosesan di dalam perangkat android, Google mengenalkan sebuah model pemrograman yang ditujukan untuk memanfaatkan kemampuan komputasi tinggi pada perangkat android yaitu *Renderscript*. *Renderscript* mengijinkan eksekusi aplikasi paralel pada beberapa *processor* seperti *Central Processing Unit* (CPU), *Graphics Processing Unit* (GPUs), dan *Digital Signal Processors* (DSPs). *Renderscript* melakukan eksekusi secara otomatis terdistribusi pada *workload* melewati semua *core* pemroses pada perangkat android (Alejandro dan Fransisco, 2015).

Model pemrograman yang diusulkan dalam penelitian ini adalah *Renderscript* dengan menggunakan perangkat mobile berbasis android. Pada penelitian ini mencoba mengimplementasikan filter tekstur LAWS untuk pengukuran energi tekstur pada citra.

## 1.2 Identifikasi Masalah

a. Bagaimana mengimplementasikan komputasi paralel pada Filter Tekstur LAWS berbasis *Renderscript*?
b. Berapa perbandingan kecepatan komputasi pada *Renderscript* pada citra dengan perbedaan ukuran citra?

## 1.3 Tujuan Penelitian

a. Mengimplementasikan komputasi paralel pada metode LAWS berbasis *Renderscript.*
b. Membandingkan kecepatan komputasi pada *Renderscript* pada citra dengan ukuran yang berbeda.

## 1.4 Batasan Masalah

a. Pada penelitian ini masalah komputasi / algoritma/ metode yang diangkat adalah komputasi pada bidang pengolahan citra untuk penentuan ukuran energi tekstur  tekstur.
b. Perangkat Android yang digunakan adalah Xiaomi Redmi 2 dengan spesifikasi :

   CPU : quad-core 1,21 Ghz

   GPU : adreno 306

   OS : Android OS, v 4.4 Kitkat
c. Data citra daun yang digunakan merupakan data citra tunggal.

## 1.5 Manfaat Penelitian

Penelitian ini bermanfaat sebagai referensi kecepatan proses komputasi paralel pada filter tekstur LAWS menggunakan *Renderscript* di perangkat mobile Android.

# BAB II

# KAJIAN PUSTAKA

## 2.1 Penelitian Terkait

Penelitian yang dilakukan oleh Lee dan Toni (1992) mengemukakan teknik segmentasi citra berdasarkan ukuran tekstur citra. Pada penelitian ini fitur tekstir yang coba diekstrak oleh Lee dan Toni adalah fitur energi tekstur. Dalam pendekatannya Lee dan Toni menggunakan metode LAWS (*Laws Texture Energy Measurement*) sebagai metode untuk menentukan ukuran energi tekstur yang digunakan sebagai segmentasi citra. Metode ini pertaman kali dikemukakan oleh Laws dalam disertasinya. Salah satu keungulan metode ini adalah untuk menyediakan beberapa bidang fitur tekstur pada citra asli. Keunngulan ini menjadi lebih bagus ketika digunakan khusus pada image dengan komponen satu warna karena untuk ekstraksi informasi tekstur yang bermanfaat dari citra mentah merupakan tugas yang sulit pada sistem pengelihatan manusia. Banyak informasi dan hasil segmentasi bisa diperoleh dengan mengintegrasikan bidang fitur tekstur tambahan. Lee dan Toni mengemukakan karakteristik tekstur adalah pergeseran, orientasi, momen, kontras dan pencahayaan invarian. Persepsi manusia cenderung pada perhitungan statistik seperti rata-rata, variansi dan fungsi auto-korelasi. Karakteristik lainnya yaitu tekstur merupakan sebuah hirarki, fitur global mengolongkan keselruhuan tekstur daripada elemen tekstur. Karakteristik – karakteristik tersebut sangat membantu digunakan sebagai garis bantu untuk merancang sistem analisis tekstur. Pada dasarnya analisis tekstur dibagi menjadi dua pendekatan utama yaitu pendekatan statistik dan pendekatan struktural.

Transformasi energy tekstur yang dekembangkan oleh Laws termasuk dalam jenis pendekatan spasial-statistik. Karakteristik pada metode ini lebih cocok dengan intuisi pada fitur tekstur, lebih condong mirip dengan pemrosesan pengelihatan manusia. Metode ini telah dikembangkan setelah investigasi dan evaluasi beberapa metode yang telah tersedia termasuk metode statistic, structural, ko-okurensi, frekuensi spasial dan pendekatan korelasi otomatis. Pengekstrakan fitur energy didapatkan dengan mengkonvolusi filter tekstur-mikro untuk mendapatkan fitur tekstur-mikro.

$$f'_k(i,j) = f(i,j) * h_k \quad (1)$$

dimana filter tekstur mikro bisa didapatkan 4 kernel 1 dimensi; L5R5, E5L5, …, R5R5, L5L5 tidak digunakan karena hasil penjumlahan elemen filter tidak berupa nol. Untuk mendapatkan fitur tekstur makro (f'$_k$), setiap tektur mikro(f'$_k$) ditransformasi ke dalam citra energy tekstur dengan menjalankan *window* tekstur makro

$$f''_k(i,j) = \frac{1}{w^2} \sum_{n=-w/2}^{w/2} \sum_{m=-w/2}^{w/2} |f'_k(n,m)| \quad (2)$$

dimana w adalah *window* makro. Nilai fitur tekstur makro diganti dengan rata-rata dari nilai absolut pada *window* makro. Ukuran optimal *window* makro berdasarkan kekasaran, sebagus kualtias fitur mikro yang tersedia. Tekstur mikro didesain untuk mengukur sifat-sifat tekstur lokal, dimana fitur tekstur makro mengukur sifat-sifat pada tekstur secara keseluruhan. Dalam penelitian yang dilakukan oleh Lee dan Toni ini telah berhasil menerapkan metode LAWS pada pencitraan pemandangan.

Pada penelitian yang dilakukan oleh Virmani dan Kumar (2011) menggunakan metode LAWS untuk mendeteksi perbedaan antara normal dan

sirhosis liver disegmentasi berdasarkan SROIs. Virmani dan Kumar menjelaskan fitur tekstur Laws' telah sering digunakan untuk perhitungan tekstur. Banyak peneliti telah menggunakan metode LAWS untuk klasifikasi pada penyakit liver. Peneliti telah menginvestigasi potensi metode LAWS untuk penggunaan pada analisa citra medis lainnya termasuk analisa tekstur tulang sebagai prediksi osteoporosis. Potensi lain dari metode LAWS telah dieskplor pada inspeksi kecacatan keramik. Varmani dan Kumar pada penelitiannya, pertama menggunakan potensi 75 rotasi invarians dari fitur tekstur Laws' untuk membedakan SROIs antara 82 normal dan 39 penyakit telah diinvestigasi. Klasifikasi pada penelitian Varmani dan Kumar telah mencapai akurasi 90.80% dengan menggunakan pengklasifikasi Jaringan Syaraf dan akurasi terbesar dengan perolehan akurasi 90.90% diperoleh dengan menggunakan pengklasifikasi *Support Vector Machine* (SVM). Langkah kedua, Varmani dan Kumar mencoba performa dari jaringan syaraf dan *Support Vector Machine* pada bagian fitur tekstur Laws' yang optimal terdiri dari 8 fitur, yaitu LLmean, LLsd, LEsd, SSskewness, RRenergy, LSenergy dan LWenergy, dipilih menggunakan *correlation based feature selection* (CFS). Proses penentuan ukuran energi tekstur menggunakan lima kernel LAWS 1 dimensi yaitu L = [1 4 6 4 1], E = [-1 -2 0 2 1], S =[-1 0 2 0 -1], W = [-1 2 0 -2 1] dan R = [1 -4 6 -4 1]. Dimana kernel tersebut bekerja pada pererataan lokal, deteksi tepi, deteksi titik, deteksi gelombang dan deteksi perata. 25 kernel LAWS dua dimensi didapatkan dengan *outer product* dua kernel LAWS satu demensi dengan kernel lain atau dengan kernel itu sendiri. Metode pengklasisfikasian pada penelitian ini diabgai menjadi dua pendekatan yaitu pengklasisfikasian jaringan syaraf dan pengklasisfikasian

SVM. Setiap pendekatan pengeklasifikasian dibagi menjadi dua model, model pertama menggunakan vektor fitur yang terdiri dari 75 fitur tekstur LAWS diambil dari 85 SROIs normal dan 35 SROIs cacat. Model kedua menggunakan vektor fitur yang terdiri 8 fitur tekstur LAWS yaitu : LLmean, LLsd, LEsd, SSskewness, RRenergy, LEenergy, LSenergy dan LWenergy. Model satu pada pendekatan jaringan syaraf menggunakan 75 neuron pada lapisan input, 38 neuron pada lapisan tersembunyi, dan 2 neuron pada lapisan keluaran. Model dua pada pendekatan jaringan syaraf menggunakan 8 neuron pada lapisan inputan, 5 neuron pada lapisan tersembunyi dan 2 neuron pada lapisan keluaran. 2 model pada pendekatan SVM menggunakan implementasi algortima SMO pada pengklasifikasian SVM menggunakan tool WEKA (*Waikato Environment for Knowledge Analysis*). Akurasi pengklasisfikasi pada pendekatan SVM diperoleh dengan menggunakan metode 10FCV. Pada percobaan performa jaringan syaraf dan *Support Vector Machine* menghasilkan akurasi klasifikasi pada jaringan syaraf sebesar 91.73% dan pada *Support Vector Machine* dengan akurasi sebesar 92.56%. Pada kesimpulannya dari 75 rotasi invarians fitur tekstur Laws' hanya 8 fitur tekstur dengan nama LLmean, LLsd, SSskewness, RRenergy, LEenergy, LSenergy dan LWenergy merupakan yang paling efektif untuk deteksi variasi tekstur antara SROIs normal dan SROIs cacat.

Qureshi dan M.Deriche pada penelitiannya di tahun 2014 mengemukakan penggunaan metode LAWS untuk percepatan penaksiran kualitas citra. Pada penelitiannya Qureshi dan M.Deriche algoritma NR-IQA pada domain spasial dikemukakan berdasarkan kernel filter Laws' diikuti dengan pemfilteran jarak. Untuk prediksi nilai subjektif tunggal, algortima *Generalized Regression Neural*

*Network* (GRNN) digunakan. Citra hasil filter Laws dibuat dengan oleh penguraian kernel yang mudah untuk diimplementasikan. Pemfilteran jarak merupakan contoh dari pemfilteran lokal yang membutuhkan sedikit komputasi. Algoritma GRNN menyediakan pembelajaran secara cepat dan prediksi yang lembut. Teknik yang diusulkan pada penelitian ini pada penentuan ukuran energi tekstur menggunakan metode LAWS yang kemudian di lanjutkan dengan *range filtering*. Teknik selanjutnya sebagai tambahan, sebuah algotima panggabungan fitur yang efisien digunakan untuk memprediksi nilai kualitas tunggal, yaitu dengan menggunakan algoritma GRNN (*Generalized Regression Neural Network*). GRNN merupakan jaringan syaraf probabilistik yang membutuhkan sedikit sampel latihan guna keefektifan pembelajaran, yang dibandingkan dengan jaringan syaraf konvesional Backpropation (BPNN). Pada penerapannya arsitektur GRNN yang digunakan meliputi 4 lapisan yaitu lapisan masukan, lapisan tersembunyi, lapisan penjumlahan dan lapisankeluaran. Vektor dari fitur dimasukkan ked dalam lapisan masukan. Banyaknya neuron masukan bergantung pada jumlah fitur. Jumlah neuron pada lapisan tersembunyi bergantung pada vektor sampel. Fitur masukan digunakan dengan Gaussian pdf (*probability density function*) pada setiap lapisan tersembunyi, kemudian hubungan antara lapisan masukan dengan lapisan tersembunyi dimasukkan kedalan unit neuron. Lapisan penjumlahan memiliki 2 unit neuron setiap unit neuron menghitung jumlah bobot keluaran dari lapisan tersembunyi. Ketika performa bisa dibandingkan dengan teknik yang sudah tersedia, Qureshi dan M.Deriche menunjukkan algoritma yang diusulkan merupakan pengkomputasian yang lebih efisien. Data yang digunakan dalam penelitian Qureshi dan M.Deriche berupa open data yang disediakan oleh

*Laboratory for Image and Video Engineering (LIVE release 2).* Set pada data berupa 808 citra dengan uraian 29 citra asli dan 779 citra distorsi. Citra asli memiliki resolusi tinggi, 24-bit tiap piksel dengan komponen warna RGB berdimensi 768 x 512. Dari citra asli dihasilkan 5 tipe citra distorsi yang berbeda. Distorsi yang digunakan adalah *Gaussian Blur, White Noise, JPEG Compression, JPEG 2000 compression(J2K)* dan eror transmisi pada J2K *bit stream* menggunakan *Fast Fading (FF) Rayleigh Channel.* Pada kesimpulannya metode penaskiran kualitas citra telah diusulkan. Sejak korespondensi persepsi pengelihatan manusia ditujukan pada *bandpass filter,* metode LAWS digunakan untuk ekstraksi citra hasil *bandpass filter* yang berbeda. Teknik yang diusulkan mudah untuk diimplementsikan pada ekstraksi fitur yang berbasis proses konvolusi dan bisa diimplementasikan secara efisien pada perangkat keras yang menjadikan teknik yang diusulkan sangat bisa digunakan untuk aplikasi multimedia secara real-time.

Penelitian yang dilakukan oleh Setiawan dkk (2015) mengimplementasikan metode LAWS (*Laws Teksture Energy Measurement*) sebagai ekstraksi fitur pada pendekatan *mammography* untuk pendeteksain dini kanker payudara. Pada pengklasifikasian *mammography,* tingkat akurasi pendeteksian kanker payudara berdasarkan pada metode ekstraksi fitur dan metode pengklasifikasian. Sebagai metode pengklasifikasi Setiawan dkk menggunakan jaringan syaraf untuk menentukan citra normal, citra tidak normal, citra kanker lunak, dan citra kanker ganas. Kanker payudara merupakan tipe kanker yang hanya terjadi pada wanita. Beberapa cara untuk mendeteksi dini kanker payudara yaitu *mammography, biopsy, ultra sound image* dan

*thermography*. Dari beberapa cara yang sudah disebutkan, *mammography* merupakan cara terbaik yang bisa mendeteksi kanker payudara sejak dini. Dalam *mammography* tingkat akurasi deteksi ditentukan oleh metode ekstraksi fitur. Ada beberapa metode untuk pemrosesan ekstraksi fitur. Studi sebelumnya menggunakan matriks *Spatial grey level dependence* (SGLD) untuk menganalisa tekstur. Metode yang lain yaitu *Gray Level Co-occurrence Matrix* (GLCM). GLCM mengkalkulasi kemungkinan dari dua piksel yang mempunyai detail tingkat keabuan pada detail hubungan spasial. Pada penelitian yang dilakukan oleh Setiawan dkk ini membuat pengkalsifikasi *mammogram* menggunakan metode LAWS sebagai ekstraski fitur dan jaringan syaraf tiruan sebagai pengklasifikasi. Alur metode klasifikasi yang diterapkan dimulai dengan pengambilan citra dilanjutkan dengan pra-pemrosesan citra, ekstraksi fitur menggunakan metode LAWS, pengklasifikasian menggunakan jaringan syaraf tiruan dan yang terkahir pengeavaluasian. Data yang digunakan pada penelitian ini menggunakan data yang disediakan oleh *Mammographic Image Analysis Society* (MIAS). MIAS merupakan organisasi peneliti Inggris yang tertarik pada bidang *mammograms* dan telah menggenarate basis data dari *mammograms* digital. Pra pemrosesan citra dilakukan dengan pemotongan citra asli menjadi citra dari bagian yang menarik. 128 x 128 ukuran piksel digunakan sebagai ukuran terbaik. Proses ektraksi fitur pada penelitian ini menggunakan 4 kernel LAWS yaitu : L5 (*level*) = [1 4 6 4 1], E5 (*edge*) = [-1 2 0 2 1], S5 (*spot*) = [-1 0 2 0 01] dan R5 (*ripple*) = [1 -4 6 -4 1]. Proses ekstraksi fitur metode LAWS yaitu dengan mengkonvolusi kernel LAWS dua dimensi ke citra masukan kemudian dilanjutkan dengan proses windowing, pada penelitian Setiawan *window* yang

digunakan adalah *window* dengan ukuran 15 x 15. Kernel konvolusi yang digunakan merupakan hasil *outer product* dari dua kernel LAWS satu dimensi sehingga menghasilkan kernel LAWS dua dimensi dengan ukuran 5x5. Pada penelitian ini 9 pasangan kernel LAWS dipilih sebagai kernel ekstraksi fitur yaitu : L5E5/ E5L5, L5S5/S5L5, L5R5/S5E5, E5S5/S5E5, E5R5/R5E5, R5S5/S5R5, S5S5, E5E5 dan R5R5. Arsitektur jaringan syaraf tiruan yang digunakan pada penelitian Setiawan dkk dengan 3 lapisan backpropagation dengan fungsi sigmoid pada lapisan tesembunyi dan lapisan keluaran. Komposisi data pada proses ini dibagi menjadi 70% data pembelajaran 15% data validasi dan 15 % data uji. Pada proses klasifikasi dibagi menjadi dua langkah, pertama pengklasifikasian data normal dan tidak normal. Kedu, pengklasifikasian data jinak dan ganas. Hasil dari pengujian pada penelitian Setiawan dkk menunjukkan 93.90% pada pengklasifikasian data normal dan tidak normal, 83.30% data jinak dan ganas yang diperoleh dari fitur LAWS. Pada GLCM untuk pengklasifikasian normal dan tidak normal : 63.30% pada 0 derajat, 73.50% pada 45 derajat, 71.40% pada 90 deraja, dan 63.30% pada 135 derajat. Untuk klasifikasi jinak dan ganas : 66.70% pada 0 derajat, 72.20% pada 45 derajat, 66.70% pada 90 derajat, dan 66.10% pada 135 derajat. Kesimpulan pada penelitian Setiawan dkk mengemukakan bahwa fitur LAWS memberikan akurasi lebih baik pada klasifikasi citra *mammograms* dibandingkan dengan fitur FLCM. Hal ini bisa dilihat dari perolehan tingkat akurasi dari fitur LAWS menunjukkan 78.21% pada klasifikasi kanker jinak dan ganas, sedangkan fitur GLCM hanya berada pada angka kurang dari 55% di setiap derajat.

Z. A. Ahmad dkk (2012) melakukan penelitian pada bidang sistem pemrosesan paralel untuk aplikasi pengolahan citra, penelitiannya mengemukakan sistem pemrosesan paralel dengan biaya rendah. Hal didasarkan pada perkembangan kompleksitas sistem pengolahan citra dan kerumitan pada perkembengan desain sistem teknologi baru. Metode yang diusulkan diimplementasikan menjadi dua tahapan. Tahap pertama difokuskan pada implementasi perangkat keras paralel prosesor, dimana prosesor dengan teknologi ARM digunakan untuk kebutuhan biaya rendah, dan sebagai standar teknologi pada prosesor. Pada tahap ini hanya fokus pada keseluruhan kebutuhan pondasi sistem dan integrasi sistem. Tahap kedua, desain tugas pada prosesor secara paralel, dan keseluruhan penampilan sistem. Paralelisasi bisa dieksploitasi pada arsitektur SISD (*Single Instruction Single Data*), SIMD (*Single Instruction Multiple Data*), MIMD (*Multiple Instruction Multiple Data*) mode pengintruksian berdasarkan dua prisnsip. Prinsip pertama, implementasi paralel dengan menumpang tindih operasi pada paralelisme *time-temporal.* Prinsip kedua, implementasi paralel dengan mereplika sumberdaya pada ruang bebas yang disebut dengan paralelisme spasial. 4 tipe pada arsitektur paralel. Pertama, penyaluran yang menyisipkan langkah-langah yang berturur-turut pada eksekusi sebuah intruksi atau operasi antar banyak tahap pada unit saluran. Langkah eksekusi pada intruksi ini bisa ditumpang tindih ketika saluran terisi penuh. Secara konsep, implementsi ini menggunakan paralelisasi temporal. Kedua, larik prosesor diman banyak prosesor terhubung bersamaan pada satu larik. Menggeneralisasi sebuah saluran ke dalam algortima, kemudian algoritma dipecah menjadi beberapa tahap diman setiap tahap bisa dimasukkan dalam sebuah

prosesor pada larik. Model biasa disebut paralisasi algoritma. Ketiga, arsitektur larik prosesor dimana banyak prosesor selalu terhubung pada larik dua dimensi dan dikendalikan oleh saru unit kontrol. Keempat, arsitektur multi-prosesor digunakan dengan menghubungkan multi-prosesor bersama-sama untuk berbagi memori umum. Metodologi yang diterapkan pada penelitian Z.A. Ahmad dkk secara teknik membagi data citra yang berupa martriks menjadi blok-blok kecil matriks. Tahap implementasi dan desain dibagi menjadi dua yaitu tahap satu yang berkonsentrasi pada desain dan implementasi perangkat keras dan tahap dua difokuskan pada implementasi perangkat pabrik. Pada kesimpulannya Z.A Ahmad berhasil mengintegrasikan sistem yang diusulkan ke dalam prosesor tunggal. Sistem pemrosesan paralel yang diusulkan berhasil mereduksi *wall clock* mencapai angka 28.31%.

Roelof Kemp dkk (2013) dalam penelitiannya memberikan sebuah studi kasus pada tenaga pemrosesan pada perangkat *mobile*. Tenaga pemrosesan pada perangkat *mobile* mengalami peningkatan yang terus-menerus. Oleh karena itu Roelof Kemp dkk dalam penelitiannya menyajikan sebuah studi kasus 3 model pemrograman yang bisa digunakan untuk memperbesar tenaga pemrosesan untuk komputasi tugas intensif. Algoritma yang digunakan berupa algoritma pencitraan dan perbandingan referensi implementasi pada algoritma berbasis OpenCV dengan *multithreded* Renderscript dan komputasi berbasis *computarion offloading* dengan Remote CUDA. Perangkat yang digunakan untuk ekspreimen adalah Tegra 3 Quad core. Dua tahun sebelum melakukan penelitian Roelof Kemp melakukan observasi pada prosesor perangkat *mobile* yang tidak hanya berkembang pada *clock speed,* akan tetapi juga pada multicore. Dengan

berkembangya tenaga pemroses maka berkembang juga kompleksitas pada perangkat *mobile.* Pada penelitian Roelof Kemp dkk menyajicoba membandingkan tiga implementasi *High Dynamic Range Photography* menggunakan tiga model pemrograman yang berbeda yaitu : *low level language, specialized language* dan *computation offloading.* Hasil perbandingan didapatkan berdasarkan waktu eksekusi dan penggunaan energi. Aplikasi yang difokuskan pada penelitian ini adalaha aplikasi demo kamera dengan mempertinggi kegunaan gotografi seperti *Negative Shutter Lag* dan *High Dynamic Range.* Secara spesifik Roelof Kemp dkk difokuskan pada *exposure fusion.* Sebuah algoritma *computer vision* dengan proses *High Dynamic Range.* Relevansi pada penelitian Roelof Kemp dkk pada komputasi algortima yang berbasis matriks, filter dan operasi pyramid yang semuanya merupakan operasi data paralel. Bagian implementasi dijelaskan bahwa Renderscript merupakan *host.device language* yang sama dengan bahasa seperti OpenCL dan CUDA. *Device Code* pada Renderscript bisa dijalankan pada satu thread atau banyak thread. Jika *device code* dijalankan secara banyak thread maka eksekusi semua kernel dijalankan pada setiap elemen pada larik 1,2 atau 3 dimensi. Renderscript secara otomatis mendistribusikan elemen-elemen ke seluruh prosesor dan juga melakukan *load balancing.* Implementasi selanjutnya *computation offloading* menggunakan Remote CUDA. RCUDA menjalankan *computation offloading* klasik dengan mengirimkan sebuah proxy pada sisi klien yang kemudian memanggil kepada server. RCUDA mengoperasikan lapisan abstraksi CUDA dan aksekusi komputasi GPGPU pada host secara paralel. Pada perangkat *mobile* ada versi modifikasi dari CUDA *shared library* (libcuda.so) yang bisa mengkomunikasikan antara TCP dengan

cudaserver. Target kunci pada eksperiman penelitian Roelog Kemp dkk ini adalah waktu eksekusi dan penggunaan energy pada seluruh implementasi. Pada kesimpulannya Roelof Kemp dkk mengemukakan hasil diskusi dan evaluasi dua alternatif model pemrograman pada implementasi *native* untuk komputasi tugas intensif pada perangkat *mobile* dengan studi kasus algoritma *exposure fusion* yang digunakan pada fotografi HDR. Pada Renderscript, Reolof Kemp dkk mengemukakan bisa memperbaiki waktu ekskusi hingga 2.2 kali lebih cepat dengan menjaga penggunaan energi pada CPU.

Pada penelitian Alejandro dan Fransisco (2015) mengemukakan sistem komputer saku (*smartphones, tabltes,…*) telah menjadi perangkat paling digunakan dengan penyebaran pasar yang penting. Jaminan teknologi yang mendukung, dengan kontribusi perkembangan kemampuan komputasi secara signifikan pada setiap generasi baru. Ekosistem Android menyediakan kerangka kerja pengembangan yang menjamin portablitas pada kode hingga mencakup jangkauan platform yang luas. Walaupun efisiensi pada setiap perangkat merupakan sebuah tantangan permanen dikarenakan keberagaman sifat dan peningkatan komputasi berdasarkan permintaan aplikasi pengguna. Alejandro dan Fransisco menunjukkan bahwa masalah portabilitas dalam efisiensi mencoba juga untuk menjamin bahwa kode akan berjalan secara efisien mencakup platform yang berbeda. Alejandro dan Fransisco dalam penelitian ini mengusulkan metode yang mengijinkan untuk memperoleh sebuah model analitik guna mendapatkan parameter-parameter untuk eksekusi optimal pada kode paralel Renderscript. Model yang diusulkan oleh Alejandro dan Fransisco telah berhasil divalidasi pada banyak tolak ukur aplikasi. Kesederhanaan dari metode yang diusulkan oleh

Alejandro dan Fransisco membuat model tersebut mudah diterapkan ke dalam ekosistem Android yang menjadikan portabilitas pada efisiensi juga tergaransi. Alejandro dan Fransisco memaparkan perkembangan teknologi seperti internet, teknologi wireless mobile telah bisa dikaitkan dengan evolusi teknologi SoC (*System on Chip*). Pada perannya, teknologi informasi telah mencetuskan revolusi komunikasi global. Berkaitan dengan revolusi ini, perengkat *mobile* telah meningkatkan kemampuan komputasi untuk beradaptasi dengan skenario baru. Seperti perangkat *mobile* telah mengimplementasikan ketersediaan teknologi seperti pada komputer dekstop. Penggunaan prosesor baru dengan arsitekrut *multicore* dan GPU. Seperti Nvidia Tegra dengan 2 dan 5 *core* ARM dan GPU dengan power rendah, dan Qualcomm snapdragon, Samsung Exynos dan OMAP 5 merupakan prosesor yang dikembangakan. Secara konsep, model arsitektural bisa dipandang sebagai keberagaman sistem CPU/GPU sistem, dimana memori digunakan bersama-sama antara CPU dan GPU dan berkerja sebagai saluran komunikasi dengan *bandwidth* tinggi. Teknologi seperti *Algorithmic Memory*, *GPUDirect, Unified Virtual Addressing* (Standar pada CUDA 6.0) dari Nvidia dan HSA dari AMD mengimplementasikan *unified memory system* untuk CPU dan GPU pada arsitektur memori tradisional. Performa memori terus menerus terlebihi oleh peningkatan tuntutan prosesor yang lebih cepat, multiprosesor core dan arsitektur paralel. Penelitian Alejandro dan Fransisco terstruktur menjadi analisa mendalam pada model eksekusi renderscript di sesi 2, analisa manfaat beberapa parameter eksekusi pada waktu running dan bagaimana parameter tersebut harus dioptimasi pada sesi 3. Penampilan metodologi untuk memodelkan parameter pada sesi 4, pengaplikasian metodolgi pada sesi 4 di sesi 5, dan pada

sesi 6 Alejandro dan Fransisco memvalidasi model ke berbagai aplikasi tolak ukur Renderscript berbasis pengolahan citra. Eksekusi default pada Renderscript menggunakan $EPK_x = 2^0$ dan $EPK_y = 2^0$ (EPK = *Element per Kernel*). Pada sesi 4 metodolgi yang digunakan Alejandro dan Fransisco yang pertama dilakukan adalah *Cluster Analysis* pengklusteran dibagi menjadi 3 bagian yaitu *fine granularity, medium granularity,* dan *coarse granularity*. Tahap selanjutnya dengan memilih EPK yang optimal, tahap terakhir yaitu memasukkan EPK optimal ke dalam masalah baru. Kesimpulan dari penelitian Alejandro dan Fransisco menyimpulkan bahwa model analitikal merupakan bisa diterpkan pada banyak aturan penting pada banyak bidang ilmu komputer. Model ini bisa digunakan untuk merepresentasikan distribusi optimal pada beban kerja meliputi unit pemroses pada keberagaman sistem. Berdasarkan kompleksitas pada beragam sistem sebuah perbuahan kecil pada parameter eksekusi pada kode bisa menunutun penampakan beban kerja pada tugas yang sulit. Model yang diusulkan telah divalidasi pada 17 belas tolak ukur masalah. Hasil yang didapatkan menunjukkan bahwa model yang diusulkan meningkatkan eksekusi default Renderscript pada semua kasus.

## 2.2 Komputasi Paralel

Pada dasarnya komputasi paralel digunakan untuk menyelesaikan suatu permasalahan besar, dengan memecah-mecah permasalahan tersebut menjadi bagian-bagian dari permasalahan yang lebih kecil (sub-masalah). Kemudian sub-masalah tersebut diselesaikan oleh kumpulan-kumpulan dari prosesor (*multi-processor*) yang nantinya terlibat dalam pengeksekusian masalah tersebut.

Dimana setiap bagian dari sub-masalah diselesaikan oleh satu prosesor (*single-processor*). Sehingga kita dapat mengambil kesimpulan jika sebuah masalah yang diselesaikan oleh satu prosesor membutuhkan beberapa banyak sub-masalah dab berapa lama waktu yang dibutuhkan oleh prosesor tersebut. Kemudian dilakukan perbandingan dengan masalah yang sama, jika masalah tersebut diselesaikan oleh banyak prosesor.

Tujuan utama komputasi paralel adalah untuk mempersingkat waktu eksekusi program yang menggunakan komputasi serial. Beberapa alas an lain yang menjadikan satu program menggunakan komputasi paralel adalah :

a. Untuk komputasi yang sangkat kompleks, terkadang sumberdaya (*resource*) yang tersedia belum cukup mampu untuk mendukung penyelesaian terhadap permasalahan secara cepat.
b. Adanya keterbatasan memori pada mesin untuk komputasi serial.
c. Adanya sumber daya non-lokal yang dapat digunakan melalui jaringan lokal atau internet.
d. Penghematan biaya pengadaan perangkat keras, dengan menggunakan beberapa mesin yang murah sebagai alternatif penggunaan mesin yang bagus akan tetapi mahal, walaupun menggunakan P-Processor(*multicore*).

Penggunaan komputasi paralel sebagai solusi untuk mempersingkat waktu yang dibutuhkan, namun eksekusi program mempunyai beberapa hambatan. Hambatan – hambatan tersebut antara lain adalah :

a. Hukum *Amdhal*, yaitu percepatan waktu eksekusi program dengan menggunakan komputasi paralel tidak akan pernah mencapai kesempurnaan karena selalu ada bagian program yang dieksekusi secara serial.

b. Hambatan yang diakibatkan karena beban jaringan, dalam eksekusi program secara paralel, prosesor yang berada di mesin yang berbeda memerlukan pertukaran data melalui jaringan. Untuk program yang dibagi manjadi tugas-tugas membutuhkan sinkornasi, *network latency* (keterlambatan jaringan) menjadi masalah utama. Permasalahan ini muncul ketka siatu tugas membutuhkan data dari tugas yang lain, bagian ini dikirimkan memalui jaringan dimana kecepatan transfer data kurang dari kecepatan prosesor yang mengeksekusi intruksi tugas tersebut. Hal ini menyebabkan tugas tersebut harus menunggu sampai data tiba terlebih dahlu sebelum mengeksekusi intruksi selanjutnya.
c. Hambatan yang terkait dengan beban waktu untuk inisilisasi tugas, terminasi tugas, dan sinkornasi.

### 2.2.1 Definisi dan Kosakata Penting

#### 2.2.1.1 Konkurensi dan Paralel

Walaupun sering digunakan sebagai sinonim, kata konkurensi dan paralel merupakan kata yang berbeda. Pada kenyataannya paralel merupakan bagian dari sebuah konkurensi, dimana penggunaannya diperuntukkan untuk menandingi model eksekusi serial. Eksekusi paralel adalah dua atau lebih aksi yang dieksekusi sekaligus, sedangkan konkurensi mengijinkan eksekusi berjalan pada dua atau lebih aksi dalam suatu progress pada waktu yang sama. Dalam prakteknya, program berjalan menggunakan dua *thread* pada *single-core* merupakan konkurensi bukan paralel.

#### 2.2.1.2 *Worker, Tasks, Data* dan *Dependencies*

Algortima mengimplementasikan prosedur untuk menyelesaikan masalah dengan menggunakan *task* dan data pada kebanyakan kasus. *Task* tersebut diproses oleh satu atau beberapa *worker*, yang merupakan sebuah abstraksi dari unit pemroses tersedia pada perangkat. Perbedaan pada program serial, algoritma konkurensi mengeksekusi intruksi pada pemesanan *nondeterministic.* Dalam waktu yang sama dua jenis *dependency* bias muncul diantara *task* pada algoritma yang membatasi pesanan yang berjalan, yaitu : *data dependency* dan *control dependency. Data dependency* adalah masalah pada satu *task* mungkin dibutuhkan untuk berkerja pada pada yang digenerasi oleh *task* dan membutuhkan untuk menunggu untuk penyelesaian atau memory yang diaksek membutuhkan untuk disinkornasi guna menjaga konsistensi memory. *Control dependency* adalah *dependency* pada *task* pada even, keadaan atau efek pada bagian lain yang dibuat oleh *task* lain.

#### 2.2.1.3 Paralel Mandatory dan Paralel Opsional

Paralel mandatory adalah model pemrograman paralel dimana sistem memaksa konkurensi eksekusi intruksi. Contohnya pada POSIX *thread,* dimana pembangkitan dan penghancuran *thread* langsung terspesifikasi pada program. Berbeda dengan paralel opsional yang mendefinisakan keuntungan pada eksekusi paralel. Sistem secara mandiri memutuskan jika eksekusi paralel bermanfaat pada keadaan sekarang.

### 2.2.2 Model Mesin

#### 2.2.2.1 Taksonomi Flynn

Dalam sebuah artikel yang direferensikan oleh Flynn (Z.A Ahmad dkk, 2012), dalam mendesain sebuah komputer dikarakteristikkan oleh alur dari instruksi-instruksi yang akan diselesaikan oleh suatu arsitektur komputer. Taksonomi ini diklasifikasikan sesuai alur dari gabungan intruksi dan data. Taksonomi ini menghasilkan empat kemungkinan kombinasi dari pengoprasian intruksi. Yang terlihat pada table dibawah ini :

**Tabel 2.1** Arsitektur Taksonomi Flyn (Konrad Markus, 2014)

| | |
|---|---|
| SISD<br>(*Single Instruction, Single Data*) | SIMD<br>(*Single Instruction, Multiple Data*) |
| MISD<br>(*Multiple Instruction, Single Data*) | MIMD<br>(*Multiple Instruction, Multiple Data*) |

Keterangan :

a. SISD (*Single Instruction, Single Data*) : Pada arsitektur ini adalah yang mewakili komputasi serial, dimana hanya ada satu tugas yang dapat dieksekusi pada suatu waktu.
b. SIMD (*Single Instruction, Multiple Data*) : Pada arsitektur ini, eksekusi sebuah intruksi dilakukan bersemaan oleh beberapa prosesor, dmana sebuah prosesor dapat menggunakan data yang berbeda dengan prosesor lain.
c. MISD (*Multiple Instruction, Single Data*) : Pada arsitektur ini, berbagai instruksi akan dieksekusi secara bersamaan oleh beberapa prosesor dengan menggunakan data yang sama.

d. MIMD (*Multiple Instruction, Multiple Data*) : Pada arsitektur ini, berbagai instruksi dapat dieksekusi oleh beberapa prosesor diman masing-masing prosesor dapat menggunakan data yang berbeda.

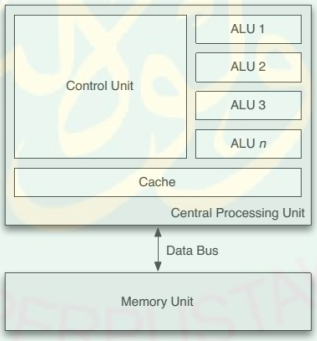
**2.2.2.2 Arsitektur Umum CPU dan GPU**

Pengguna CPU modern talah mengalami pertumbuhan pada bidang performa dan efisiensi, selama desain utama masih berdasarkan arsitektur Von-Neumann klasik. Menunjukkan bahwa arsitektur ini berhasil dalam penggunaan umum komputer. Sebuah CPU terhubung melalui bus data ke unit memori yang mengijinkan untuk membaca dan menulis data. CPU terdiri dari unit kontrol dan sebuah *aritmethic/logic unit* (ALU). Kontrol unit menginterpretasikan intruksi pada program dan mengontrol komunikasi ke dan dari ALU yang mengeksekusi intruksi logic dan aritmatik. Modern CPU memiliki penambahan pada *data cache* dan terdiri dari banyak ALU. Walaupun basis desain pada Von-Neumann untuk berjalan secara sekuensial, dimana bertempat pada *Single Instrucsion, Single Data (SISD)* kategori Flynn. Gambar 2.1 menunjukkan arsitektur CPU.

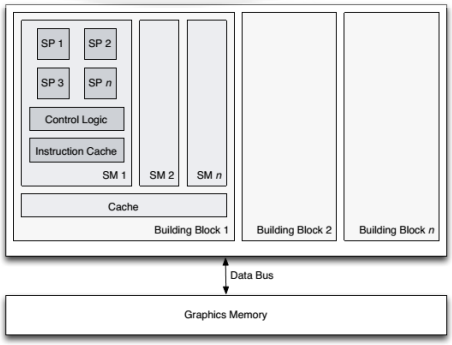
GPU memiliki arsitektur perangkat yang berbeda. Dari perangkat *graphic pipeline* dengan beberapa prosesor terspesialisasi ke sebuah *unified grid of processor* atau *shader* dengan array besar pada unit pemroses yang bias deprogram secara bebas. Unit pemroses ini dirancang untuk perforna perhitungan jumlah besar *floating point number* secara paralel dengan mengeksekusi banyak nomor dari *thread* yang beroperasi secara beregu pada memori dengan intruksi sama untuk meminimalisi *control logic* dan *long-latency memory accesses.* GPU modern terdiri dari beberapa *streaming multiprocessors (SMs)* yang secara bersama-sama terorganisasi pada sebuah *building block.* Setiap

SM memiliki banyak *streaming processors (SPs)* terhubung pada *shared instruction cache* dan *control logic unit.* Semua SMs memiliki bandwith akses tinggi ke memori video pada GPU, dengan kapasitas 128 MB sampai dengan 8 GB pada kartu grafis modern. Gambar 2.2 menunjukkan arsitektur GPU modern.

Paralelisasi tingkat tinggi menuntun pada kekuatan impresif komputasi pada GPU modern. Perbedaan yang mencolok pada perbandingan pada CPU bahwa GPU didesain untuk mencapai nilai tinggi untuk perenderan grafis ketika CPU didesain untuk *low latency.* Sebuah CPU modern mengoperasikan pada skala nano detik, tapi pada sistem pengelihatan manusia berkerja hanya pada mili detik. Oleh karena itu GPU didesain untuk menyelesaikan pemrosesan pada ukuran data yang besar pada dengan waktu panjang, sedangkan CPU memroses sedikit data pada waktu yang sangat pendek.




**Gambar 2.1** Arsitektur CPU (Konrad Markus, 2014)




**Gambar 2.2** Arsitektur GPU Modern (Konrad Markus, 2014)

## 2.3 *Renderscript*

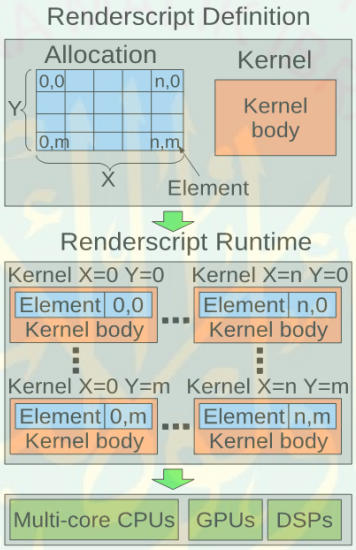
Untuk memanfaatkan kemampuan komputasi tinggi pada perangkat *mobile*, Android menyediakan Renderscript, sebuah kerangka kerja untuk menjalankan tugas komputasi secara intensif pada kinerja tinggi. Kode Renderscript dikompilasi ke bytecode menengah dengan llvm compiler yang berjalan sebagai bagian dari membangun Android. Ketika aplikasi berjalan pada perangkat, bytecode tersebut kemudian dikompilasi (*Just-in-time*) ke kode mesin oleh compiler llvm lain yang berada pada perangkat. Kode mesin dioptimalkan untuk penggunaan perangkat dan juga cache, sehingga berikutnya ketika Renderscript diaktifkan aplikasi tidak mengulang bytecode. Kode Renderscript beroperasi pada tingkat asli tetapi aplikasi yang menggunakan Renderscript masih berjalan dalam Android Runtime. Renderscript memberikan lapisan menengah untuk memfasilitasi komunikasi dan memori manajemen antara dua tingkat kode. Renderscript memungkinkan pelaksanaan aplikasi paralel di bawah beberapa jenis prosesor seperti CPU, GPU atau DSP, melakukan distribusi otomatis beban kerja di seluruh *processing core* yang tersedia pada perangkat. Renderscript terutama berorientasi untuk digunakan dengan perhitungan data paralel dan sangat berguna untuk aplikasi pengolahan citra, komputasi fotografi atau *computer vision*.

Model pemrograman Renderscript mirip dengan bahasa lain seperti CUDA atau OpenCL. Jadi, kelemahan utama Renderscript adalah kompleksitas. Perhatikan bahwa sebagian besar programmer Android yang Java programmer dengan sedikit atau tanpa pengetahuan tentang pemrograman paralel. Selanjutnya, peningkatan kinerja yang diperoleh Renderscript terkait dengan implementasi Java bisa dianggap keuntungan utama.

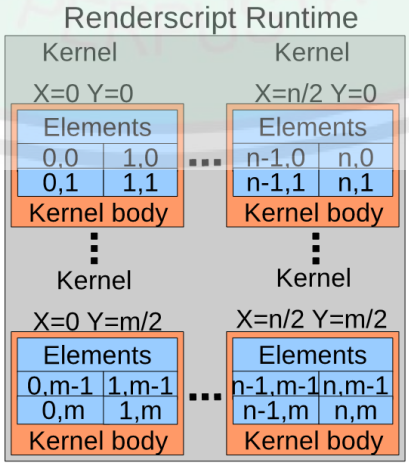
Untuk mengembangkan aplikasi paralel dengan Renderscript dua konsep utama harus diperhatikan, yaiutu Alokasi Memori dan Kernel komputasi tinggi:

- Alokasi memori: Kode Renderscript berjalan native dan membutuhkan akses ke memori yang dialokasikan di Android Runtime. Untuk mencapai hal ini, memori yang dialokasikan dalam Runtime Android harus melekat pada runtime Renderscript. Proses ini, memungkinkan runtime Renderscript untuk bekerja dengan memori yang meminta tetapi tidak dapat secara eksplisit mengalokasikan (seperti malloc dalam C). Untuk mendukung sistem pengalokasian memori ini, ada set API yang memungkinkan Android Runtime untuk mengalokasikan memori dan menawarkan fungsionalitas mirip dengan panggilan malloc. API ini terdiri dari:
  - Elemen mewakili (sizeof ()...) Sebagian dari panggilan malloc dan merangkum satu sel dari alokasi memori, seperti sebagai nilai float tunggal atau struct.
  - *Type* menjelaskan tata letak memori (pada dasarnya sebuah larik Elemen) tetapi tidak mengalokasikan memori untuk data yang menggambarkan. Sebuah *Type* adalah template alokasi memori dan terdiri dari Elemen dan satu atau lebih ukuran.
  - Alokasi memberikan memori untuk aplikasi berdasarkan deskripsi memori yang diwakili oleh Mengetik.
- Kernel komputasi tinggi: Kernel ditulis dalam bahasa C99. Sebuah API Java digunakan untuk mengelola *liftime* sumber daya Renderscript dan mengendalikan pelaksanaan Kernel. Sebuah Kernel adalah fungsi paralel yang mengeksekusi seluruh setiap Elemen dalam sebuah Alokasi. Gambar 2.1

menunjukkan model eksekusi Renderscript. Setiap Kernel telah ditetapkan dengan satu atau dua Alokasi. Secara default, Kernel berjalan di seluruh Alokasi, dengan satu pelaksanaan tubuh kernel per Elemen dalam Alokasi (Renderscript runtime pada Gambar. 2.3). Sebuah Kernel dapat mengakses koordinat eksekusi saat menggunakan argumen x, y, dan z. *Runtime* Renderscript berjalan secara paralel pada seluruh rangkaian *kernel body* di semua prosesor yang tersedia pada perangkat.

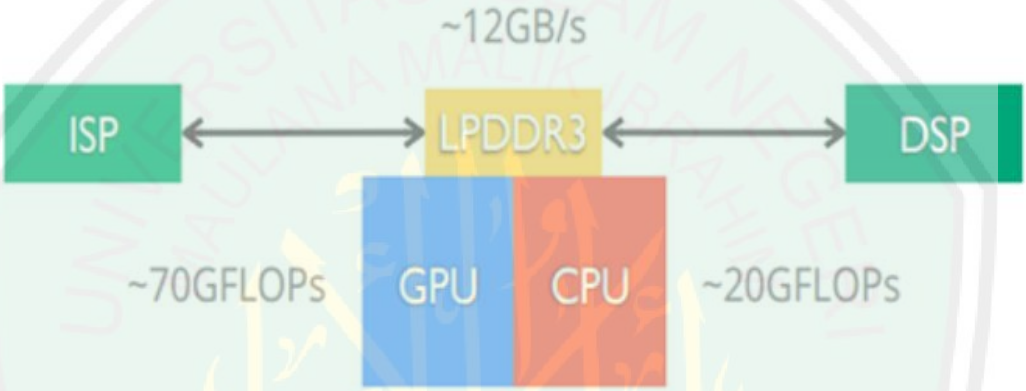



**Gambar 2.3** Model Eksekusi Renderscript (Alejandro dan Fransisco, 2015)




**Gambar 2.4** *Runtime* Renderscript (Alejandro dan Fransisco, 2015)

Model eksekusi Renderscript bergantung pada arsitektur perangkat *mobile* yang sangat berbeda dengan arsitektur yang dimiliki oleh perangkat komputer desktop yang memiliki sistem dengan ketinggian kemampuan. Gambar 2.4 menerangkan arsiektur pada perangkat *mobile.*




**Gambar 2.5** Arsitektur Perangkat *Mobile* (Rafael Fransisco dkk, 2014)

Komunikasi antara CPU dan GPU berjalan pada memori DDR, LPDDR atau MDDR pada 12 Gb/s dan pada set intruksi msndukung hingga 70 GFLOPS pada GPU dan 20 GFLOPS pada CPU.

Renderscript telah dikembangkan pada paradigma *Low Level Virtual Machine.* Model eksekusi android bisa dilihat pada Gambar 2.6.

Operasi yang dijalankan adalah :

1. Eksekusi daftar program dibuat pada bahasa C99, kemudian decompile dengan format :
    a. Enkapsulasi kode sumber atau kernel, itu diproses dan membuat daftar binary arsitektur.

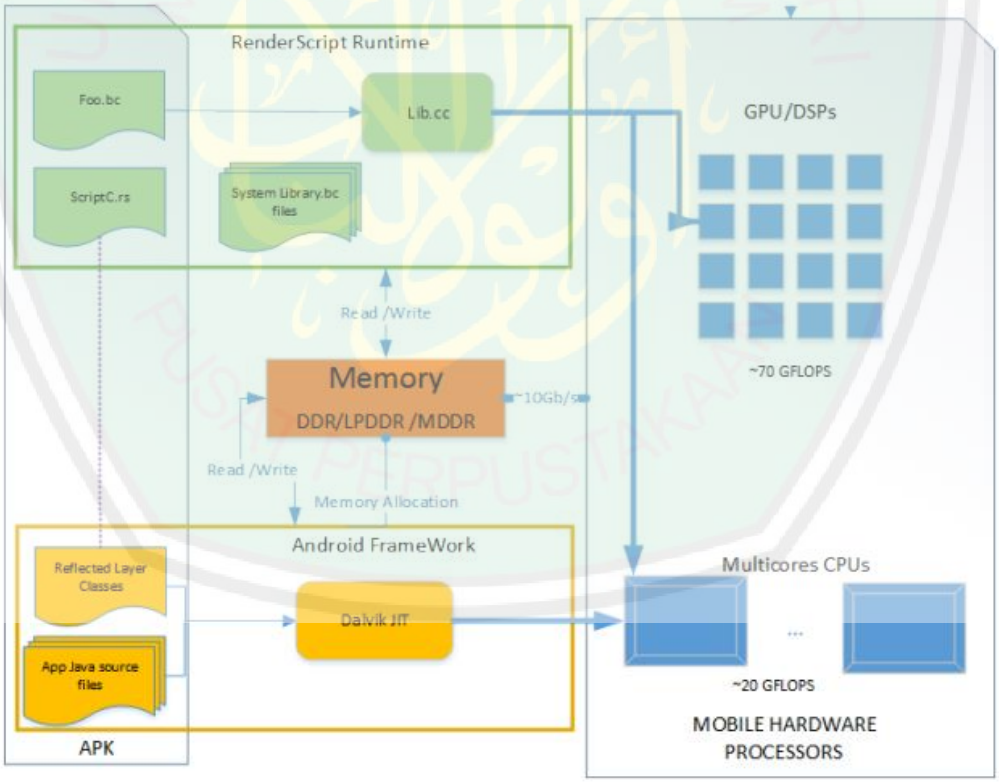
b. Sebuah rutinitas spesial untuk mengcompile daftar binary pada satu atau lebih prosesor.

c. Enkapsulasi spesial yang terintegrasi bergantung pada programmer yang membutuhkan beberapa operasi.

2. Kelas JAVA yang dibuat secara otomatis oleh Development Kit.

3. Sumber-sumber administrasi dan penyelenggaraan yang dikendalikan oleh *Java Application Interface.*

Rangkaian di atas terintegrasi dengan *virtual machine,* dimana penambahan binary yang bisa dieksekusi dan daftar pengecekan seperti *Java Application.*




**Gambar 2.6** Model Eksekusi Renderscript (Rafael Fransisco dkk, 2014)

# BAB III

# METODE PENELITIAN

Pada bab ini akan diuraikan mengenai metode penelitian untuk perancangan implementasi komputasi paralel pada metode LAWS sebagai penentuan ukuran energi tekstur menggunakan *Renderscript*. Pada bab ini akan dijelaskan mengenai data yang digunakan dalam penelitian, prosedur pengerjaan penelitian dan metode untuk analisa dan perancangan.

## 3.1 Data

Data yang digunakan dalam penelitian ini berupa data citra digital daun dengan komponen warna RGB. Rincian jumlah dan ukuran data citra adalah sebagai berikut :

1. 30 citra daun cocor bebek dengan ukuran 512x512.
2. 30 citra daun cocor bebek dengan ukuran 1024x1024.
3. 30 citra daun cocor bebek dengan ukuran 2048x2048.

Contoh data citra daun bisa dilihat pada Gambar 3.1.

**Gambar 3.1** Data Citra Daun

## 3.2 Prosedur Penelitian

Pada subbab ini akan diuraikan tahap-tahap penelitian implementasi komputasi paralel pada metode LAWS. Tahapan penelitian dengan rincian :

a. Penelitian dimulai dengan pengambilan dan pengumpulan data, teknik pengumpulan data telah dijelaskan di subbab 3.1.
b. Tahap selanjutnya setiap data citra dikelompokkan sesuai dengan resolusi citra.
c. Analisa dan Perancangan model pemrograman yang dapat diterapkan dalam data berupa citra digital.
d. Penulisan *source code* dan penerapan metode LAWS ke dalam renderscript.
e. Pencatatan dan perhitungan waktu (*consuming time*) yang dibutuhkan dalam proses komputasi.
f. Penyajian hasil dan perbandingan perhitungan waktu komputasi paralel dan sekuens yang disajikan dalam sebuah diagram batang.

## 3.3 Analisis dan Perancangan

Pada subbab ini akan diuraikan alur penentuan ukuran energi tekstur menggunakan metode LAWS, proses perhitungan metode LAWS, teknik pemecahan data matriks ke dalam komputasi parallel, teknik paralelisasi konvolusi dan pergeseran *window*.

### 3.3.1 Metode

Metode yang diimplementasikan untuk komputasi paralel pada penelitian ini yaitu metode LAWS dengan optimasi pemrosesan kecepatan filter. Kernel LAWS terdiri dari 4 matriks 1 x 5 yaitu :

a. L5 (*Level*) = [1 4 6 4 1]

b. E5 (*Edge*) = [-1 -2 0 2 1]

c. S5 (*Spot*) = [-1 0 2 0 -1]

d. R5 (*Ripple*) = [1 -4 6 -4 1]

e. W5 (*Wave*) = [-1 2 0 -2 1]

Kernel LAWS didapatkan dengan mengalikan (*outer product*) pasangan dua vektor sehingga didapatkan matriks kernel LAWS. Contoh matriks pasangan dua kernel LAWS bisa dilihat pada Gambar 3.4.

Menurut Kolter Zico (2015), *outer product* dari dua vektor x dan y adalah matriks A diperoleh dari rumus (1). setiap elemen dari vektor x dikalikan dengan setiap elemen dari vektor y, proses *outer product* vektor x dan y bisa dilihat pada Gambar 3.2.

$$A = x \otimes \text{y}, \text{Aij} = \text{XiYj} \quad (1)$$

$$xy^T \in \mathbb{R}^{m \times n} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_my_1 & x_my_2 & \cdots & x_my_n \end{bmatrix}$$

**Gambar 3.2** *Outer Product* dua vektor

$$\text{E5L5} = \begin{bmatrix} -1 \\ -2 \\ 0 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -8 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

**Gambar 3.3** Proses *Outer Product* Dua Kernel LAWS 1-Dimensi

$$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

(a) L5L5 (b) E5L5

**Gambar 3.4** Matriks Kernel LAWS 2-Dimensi

Citra hasil filter dari metode LAWS diperoleh dengan proses konvolusi citra asli dengan pasangan kernel LAWS.

$$f'_k(i,j) = f(i,j) * h_k \tag{2}$$

dimana :

- f(j,k) = citra asli
- f'(j,k) = citra hasil filter
- $h_k$ = kernel LAWS

Citra hasil filter pasangan kernel LAWS kemudian diubah menjadi citra energi tekstur dengan menggerakan *window*. *Window* yang digunakan pada penelitian ini yaitu *window* dengan ukuran 15. Ukuran ini telah banyak diterapkan pada penelitian – penelitian terdahulu (Lee dan Toni, 1990) (Setiawan dkk, 2015).

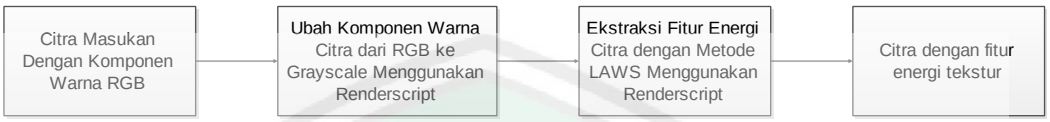
$$f''_k(i,j) = \frac{1}{w^2} \sum_{n=-\frac{w}{2}}^{\frac{w}{2}} \sum_{m=-\frac{w}{2}}^{\frac{w}{2}} |f'_k(n,m)| \tag{3}$$

dimana :

- f'$_k$(i,j) = citra energi tekstur
- f'$_k$ = citra hasil filter kernel LAWS
- w = *window*

### 3.3.2 Proses Filter LAWS

Proses filter LAWS dengan menggunakan metode LAWS bisa dilihat pada Gambar 3.5.




**Gambar 3.5** Diagram Blok Filter LAWS

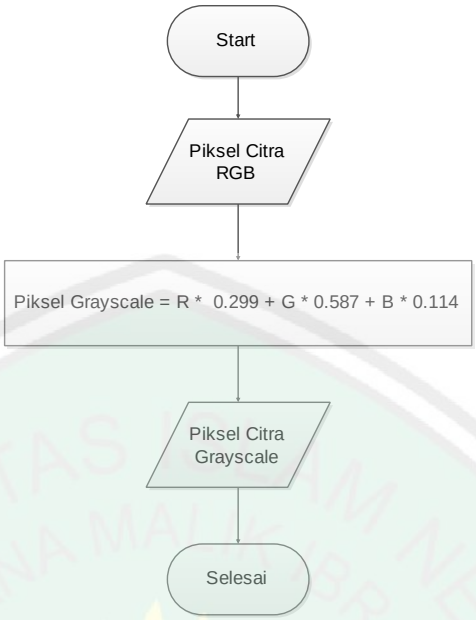
Penjelasan proses penentuan ukuran energi tekstur menggunakan metode LAWS adalah sebagai berikut :

a. Citra yang digunakan masukan adalah citra dengan komponen warna RGB (Red, Green, Blue).
b. Citra dengan komponen warna RGB diubah menjadi citra dengan komponen warna *grayscale* menggunakan *Renderscript.*
c. Citra yang sudah diubah menjadi *grayscale* kemudian di eksatrasi fitur dengan proses konvolusi menggunakan kernel LAWS, proses konvolusi pada citra dioptimasi dengan komputasi paralel menggunakan *Renderscript.*
d. Hasil dari proses konvolusi menggunakan kernel LAWS berupa citra dengan fitur energi tekstur.

#### 3.3.2.1 Rincian Proses Filter LAWS

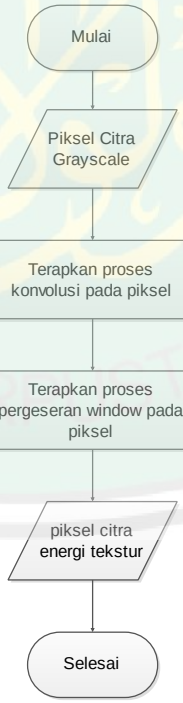
Rincian proses ektraksi fitur menjelaskan dua blok dari diagram blok 3.5.

a. Rincian Proses Pengubahan Komponen Warna Citra




**Gambar 3.6** Diagram Alir Proses Pengubahan Komponen Warna Citra

b. Rincian Proses Penerapan Metode LAWS




**Gambar 3.7** Diagram Alir Penerapan Metode LAWS

Perincian proses konvolusi dijelaskan pada *psuedo-code* di bawah ini:

```
citra_masukan; kernel_konvolusi; m2 = floor(panjang kernel_konvolusi / 2);
n2 = floor(lebar kernel_konvolusi / 2);
For i = 0 : panjang citra_masukan { For j = 0 : lebar citra_masukan {
        For p = - m2 :m2 { For q = - n2 : n2 {
        Citra_masukan(i,j) += kernel_konvolusi(p + m2 - 1, q + n2 –
    1) * citra_masukan(i – p, j – q);
        }
        }
    }
}
```

***Psuedo-code* 3.1**

Perincian proses pergeseran *window* dijelaskan pada *psuedo-code* di bawah ini :

```
citra_masukan;
window;
m2 = floor(panjang window / 2);
n2 = floor(lebar window / 2);
for i = 0 : panjang citra_masukan {
    for j = 0 : lebar citra_masukan {
        for p = - m2 : m2 {
            for q = - n2 : n2 {
                sum + = |citra_masukan(p,q)|;
            }
        }
        citra_masukan(i,j) = 1 / w^2 * sum;
    }
}
```

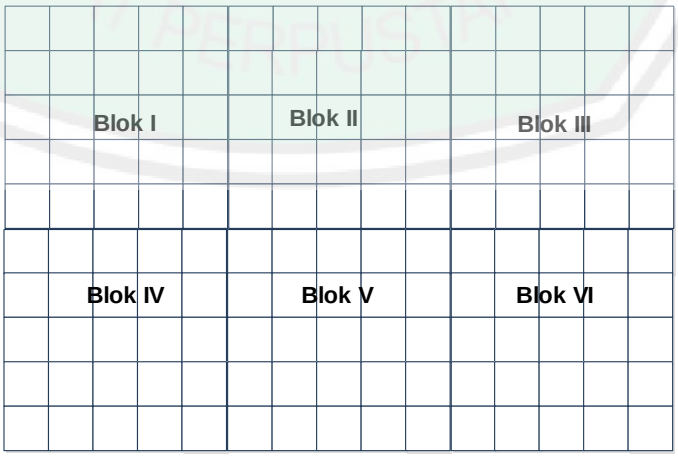
***Psuedo-code* 3.2**

### 3.3.3 Desain Komputasi Paralel

Pada penelitian ini peneliti menggunakan model pemrograman *Data Parallelism* sebagai model untuk optimasi ekstraksi fitur menggunakan metode LAWS. Teknik *data decomposition* dipilih sebagai teknik pemecahan suatu masalah sehingga bisa dijalankan secara paralel. Teknik *data decomposition* merupakan teknik yang cocok digunakan dalam pemecahan struktur data vektor, matriks dan pada bidang pengolahan citra. Citra sendiri merupakan sebuah data matrik dengan nilai piksel pada nilai koordinat tertentu. Teknik pemecahan data / dekomposisi data adalah teknik pemrosesan satu tugas dengan banyak data dalam satu kali waktu pemrosesan.

#### 3.3.3.1 Teknik Dekomposisi Data

Teknik dekomposisi data atau teknik pemecahan data citra yang digunakanan pada penelitian ini yaitu dengan membagi citra asli menjadi blok-blok citra dengan ukuran sesuai dengan kernel LAWS berupa matriks berukuran 5x5. Blok-blok data citra mengandung 25 piksel setiap satu blok, 25 piksel terdiri 4 komponen warna *Alpha, Red, Green* dan *Blue* dalam setiap satu piksel.



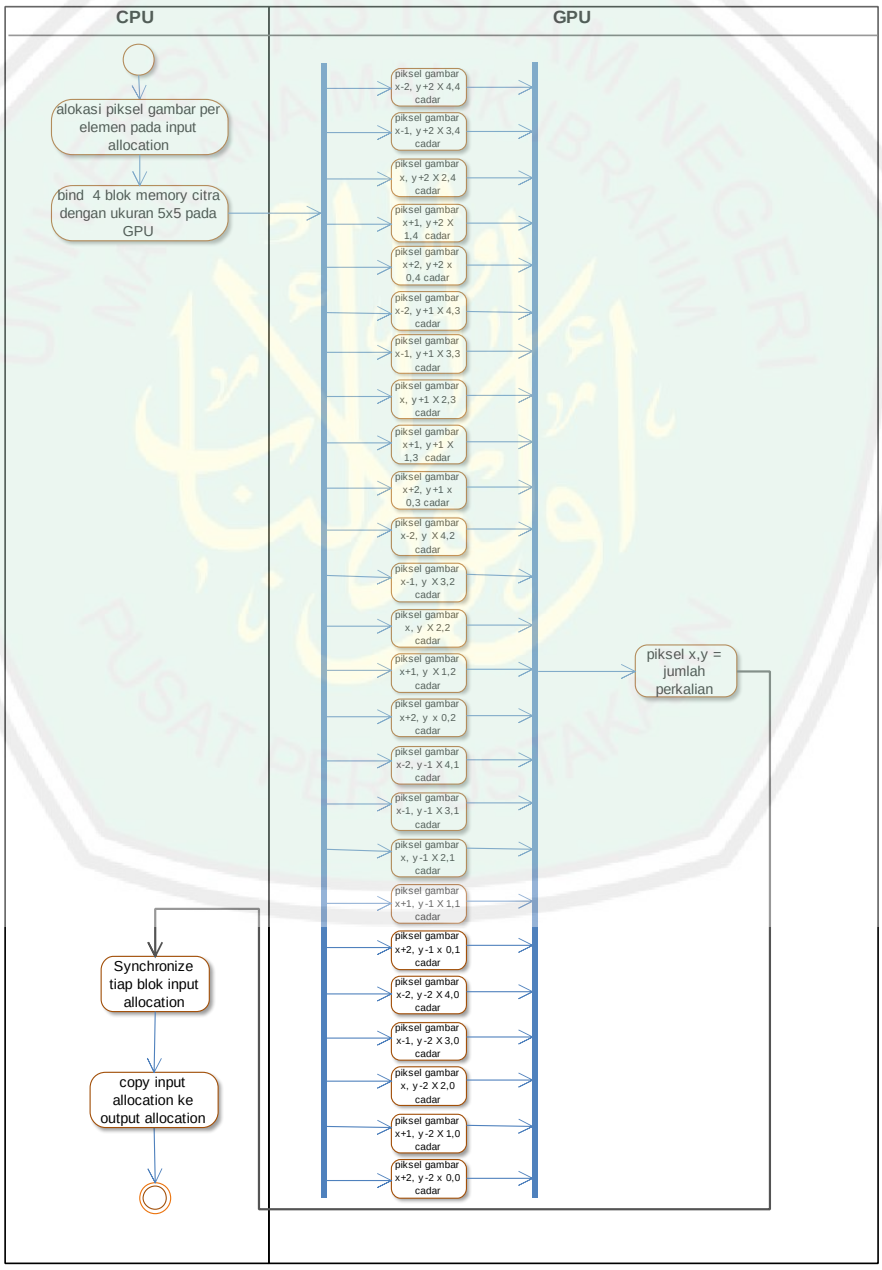

**Gambar 3.8** Pemecahan data citra ke dalam blok citra**.**

### 3.3.3.2 Implementasi pada Renderscript

Dalam sub-bab ini menjelaskan tentang teknik pemecahan konvolusi, pemecahan window dan teknik pengalokasian memori.

#### 3.3.3.2.1 Teknik Pemecahan Konvolusi

Proses pemecahan proses konvolusi ke dalam komputasi paralel pada metode LAWS secara paralel bisa dilihat pada activity diagram pada Gambar 3.9.



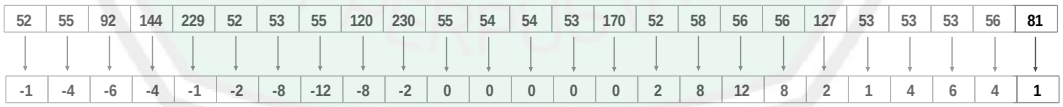

**Gambar 3.9** Activity Diagram Komputasi Paralel Konvolusi

Perhitungan nilai piksel pada proses konvolusi menggunakan kernel matriks LAWS dipecah menjadi 25 proses perkalian. 25 proses ini didapatkan dari jumlah elemen atau piksel dalam matriks kernel LAWS berukuran 5 x 5. Hasil 25 proses perkalian data piksel citra dengan elemen matriks kernel LAWS ukuran 5 x 5 kemudian dijumlahkan dan dimasukkan ke dalam nilai piksel data citra.

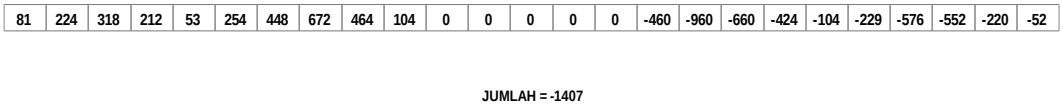
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 230 | 212 | 92 | 57 | 54 | 49 | 51 | 64 | 54 | 52 |
| 230 | 229 | 81 | 56 | 53 | 53 | 53 | 57 | 54 | 54 |
| 230 | 230 | 127 | 56 | 56 | 58 | 52 | 53 | 58 | 53 |
| 230 | 230 | 170 | 53 | 54 | 54 | 55 | 52 | 55 | 53 |
| 230 | 230 | 230 | 120 | 55 | 53 | 52 | 52 | 54 | 51 |
| 230 | 231 | 239 | 144 | 92 | 55 | 52 | 46 | 54 | 55 |
| 230 | 230 | 230 | 230 | 111 | 53 | 53 | 50 | 48 | 51 |
| 230 | 230 | 229 | 229 | 220 | 110 | 47 | 51 | 48 | 46 |
| 230 | 229 | 231 | 229 | 229 | 181 | 112 | 86 | 48 | 47 |
| 230 | 230 | 229 | 230 | 230 | 229 | 230 | 166 | 113 | 118 |

**Gambar 3.10** 10 x 10 Data Piksel Citra

25 proses perkalian dikenakan pada setiap elemen citra sebesar ukuran kernel LAWS. Proses perkalian elemen data citra dan elemen kernel LAWS dilakukan secara paralel.



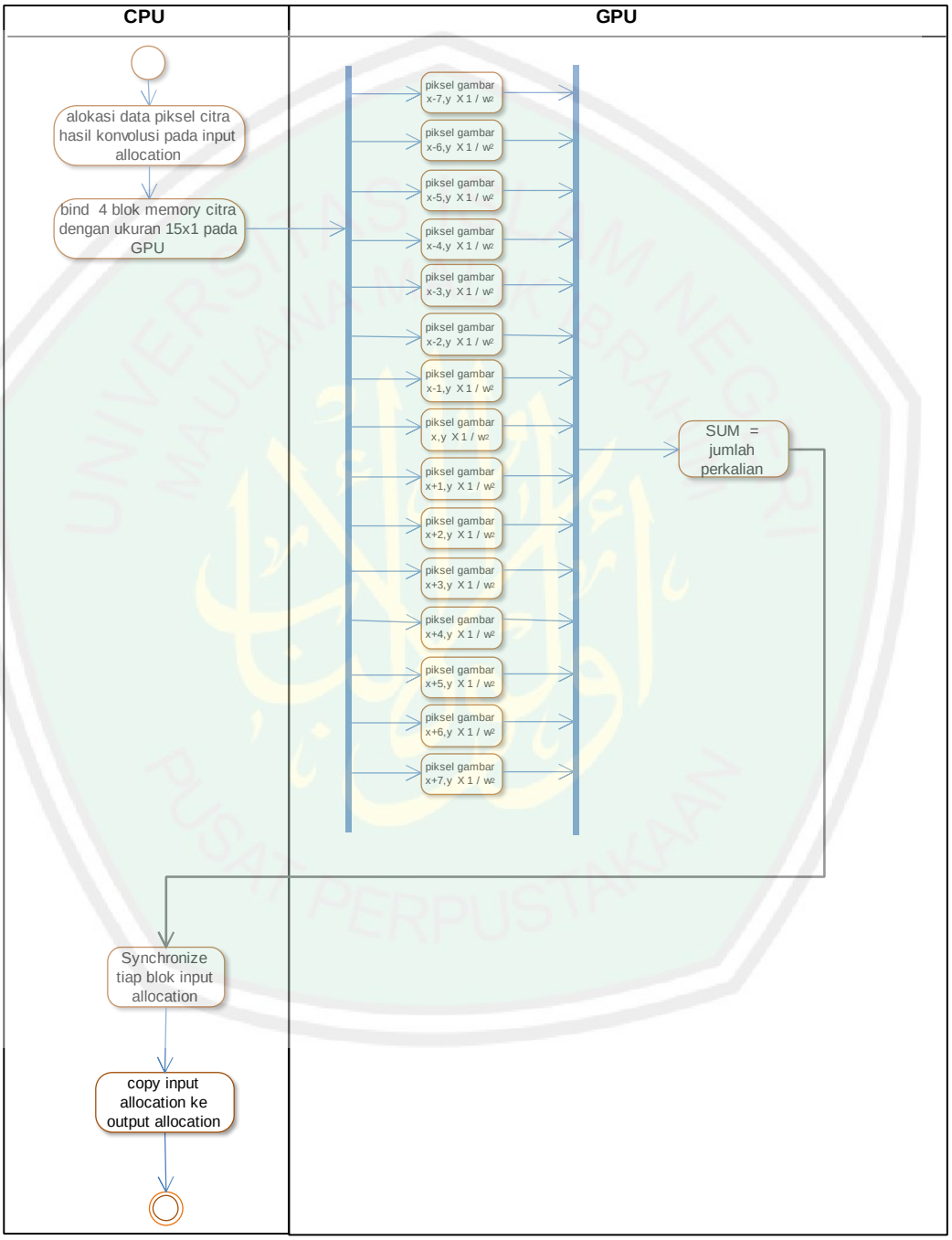

**Gambar 3.11** 25 Proses Perkalian Elemen Data Citra Dengan Kernel LAWS




**Gambar 3.12** Proses Penjumlahan Hasil Perkalian

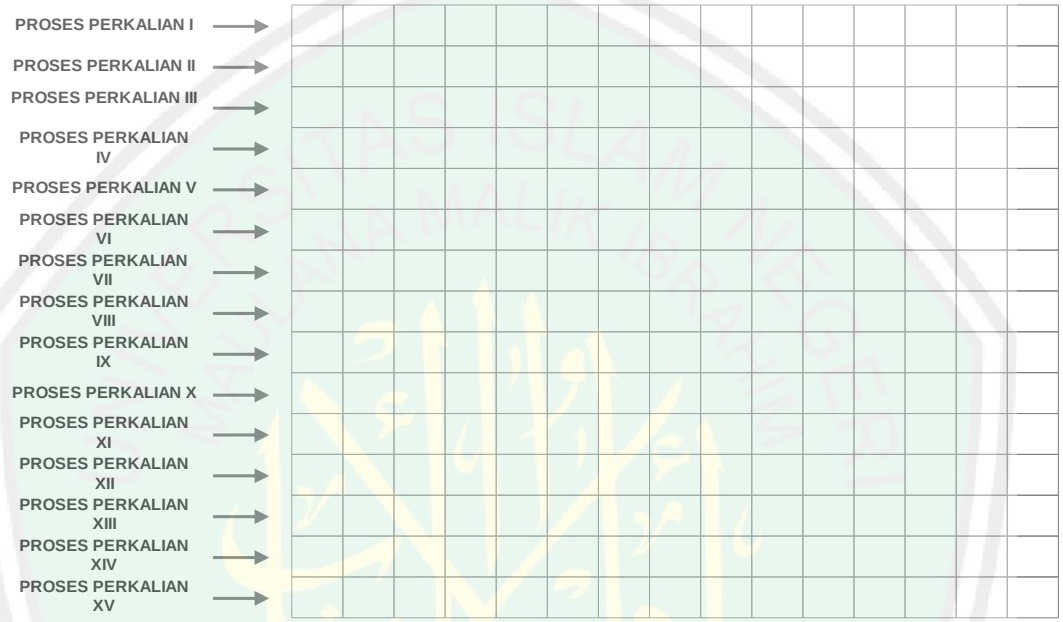
#### 3.3.3.2.2 Teknik Pemecahan *Window*

Pemecahan proses filter *window* dilakukan setiap 15 proses perkalian elemen pada tiap baris data citra .




**Gambar 3.13** Activity Diagram Pemecahan Proses Pergeseran *Window*

Proses pergeseran *window* dengan ukuran 15 x 15 dipecah menjadi 15 proses perkalian 1 baris elemen citra dengan $1/w^2$. Perkalian elemen setiap baris data citra dilakukas secara parallel. setiap satu data piksel dikalikan secara bersama dengan 15 data piksel lain yang berada dalam saru baris.




**Gambar 3.14** Pemecahan pemrosesan *window* berdasarkan baris *window*

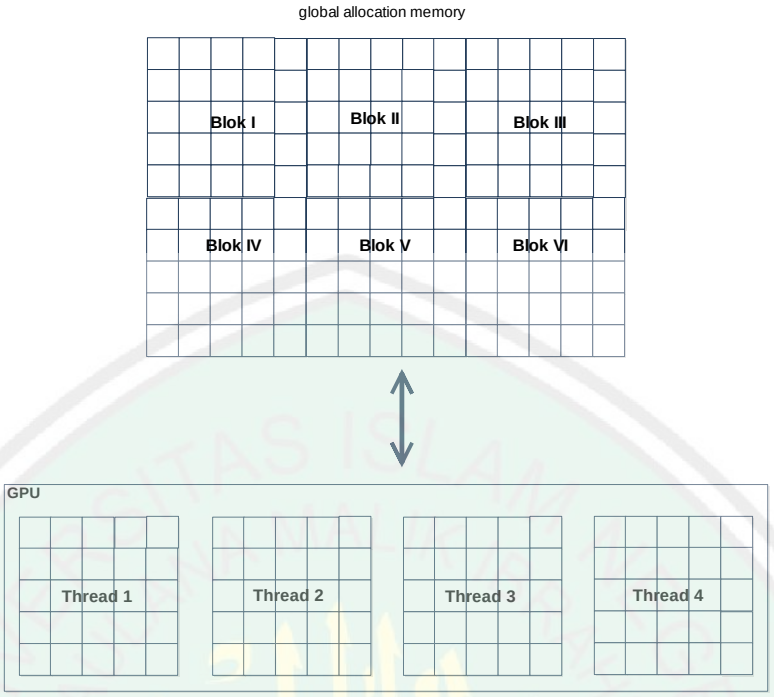
### 3.3.3.2.3 Teknik Pengalokasian Memori

Teknik pengalokasikan memori berdasarkan *data decomposition* dengan mengalokasikan setiap piksel ke dalam setiap elemen pada satu alokasi memory. Setiap elemen pada alokasi memory mewakili satu data piksel citra digital. Sehingga banyaknya elemen pada alokasi memory sebanyak NxM ukuran citra digital. Jenis elemen yang dialokasikan berupa data pixel, format ini telah disediakan oleh *Renderscript* API sehingga satu elemen memori bisa menampung data pixel citra berwarna dengan komponen ARGB (*Alpha, Red, Green, Blue*). Tipe elemen yang dialokasikan berupa data elemen multidimensi karena data yang

dialokasikan ke dalam sebuah matriks dengan ukuran NxM. Proses pengalokasian memori dari citra digital ke dalam elemen *memory allocation* bisa dilihat pada Gambar 3.15.


**Gambar 3.15** Alokasi Data Piksel Pada Memori Renderscript

Alokasi memori data piksel citra dipecah menjadi blok-blok memori. Setiap 4 blok memori dikirim ke inti pemroses GPU. Pada penelitian ini peneliti menggunakan GPU Adreno 306 dengan 4 inti pemroses. Pada *Renderscript* ,jumlah *thread* dibangkitkan otomatis mengikuti inti pemroses yang dimiliki perangkat prosesor, kernel *Renderscript* berjalan secara *multithreading* jika perangkat pemroses memiliki *multicores* (Sams Jason, 2013). 4 *thread* digunakan mengikuti jumlah inti pemroses yang dimiliki GPU Adreno 306. Setiap *thread* pada penelitian ini mengelola 25 piksel dari setiap blok memori data citra. 4 blok data piksel citra dikirim dari global memory allocation ke 4 inti pemroses GPU. Setiap elemen dari blok data citra dikalikan dengan konstanta kernel LAWS dan kemudian dijumlahkan. Proses tersebut dijalankan secara bersama – sama oleh 4 inti pemroses GPU.




**Gambar 3.16** Transfer Data Global Memory ke *Core* GPU

# BAB IV

# UJI COBA DAN PEMBAHASAN

Rangkaian uji coba dan evaluasi terhadap penelitian yang dikerjakan untuk mengetahui tingkat keberhasilan implementasi penentuan ukuran energi tekstur citra metode LAWS dalam komputasi paralel menggunakan bahasa pemrograman *Renderscript*. Rangkaian uji coba dan evaluasi ditujukan untuk tingkat keberhasilan implementasi ekstraksi fitur tekstur citra metode LAWS pada bahasa pemrograman komputasi paralel *Renderscript* dan pengujian perbedaan kecepatan pada ukuran citra yang berbeda. Hasil uji coba berupa citra hasil penentuan ukuran energi tekstur dan hasil perhitungan waktu komputasi secara sekuensial dan paralel dipaparkan secara rinci dalam bentuk diagram batang, tabel data, dan bagan.

## 4.1 Uji Coba

Uji coba proses ekstraksi fitur tekstur menggunakan metode LAWS menghasilkan citra dengan penentuan ukuran energi tekstur. Metode LAWS memiliki 25 pasangan kernel dua dimensi, sehingga citra yang dihasilkan sebanyak 25 dengan penentuan ukuran energi tekstur. Pembahasan citra hasil secara rinci akan dibahas satu persatu menurut pasanagan kernel LAWS dua dimensi yang digunakan yaitu, L5L5, L5E5, L5S5, L5R5, L5W5, E5L5, E5E5, E5S5, E5R5, E5W5, S5L5, S5E5, S5S5, S5R5, S5W5, R5L5, R5E5, R5S5, R5R5, R5W5, W5L5, W5E5, W5S5, W5R5, W5W5. Huruf depan pada pasangan kernel LAWS dua dimensi menunjukkan proses eksekusi secara vertikal pada citra,

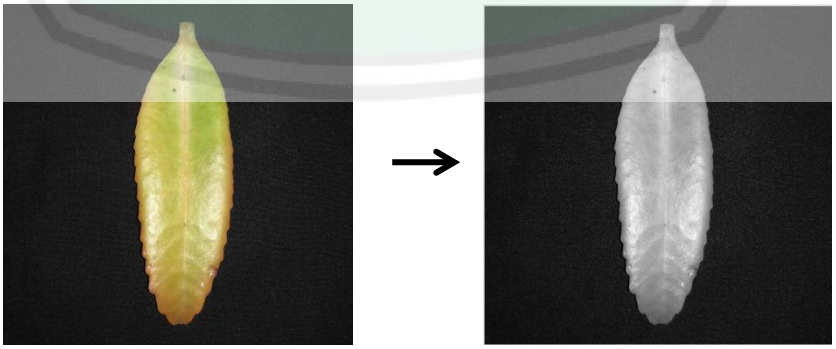
sedangkan huruf belakang menyatakan proses eksekusi secara horisontal pada citra.

### 4.1.1 Alur Proses Pengujian

Alur Pengujian proses penentuan ukuran energi tekstur pada citra daun cocor bebek dapat dilihat pada blok diagram di bawah ini.

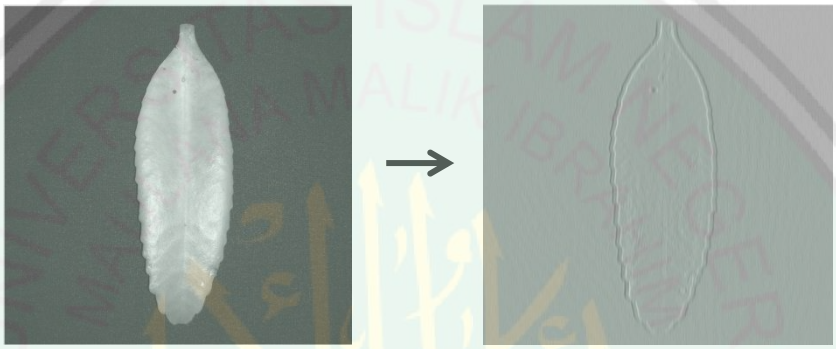

**Gambar 4.1** Alur Proses Pengujian

Alur pengujian dimulai dengan mengambil citra masukan yang sudah tersimpan pada perangkat android, kemudian citra masukan dengan komponen warna rgb diubah menjadi citra dengan komponen warna grayscale. Hasil pengubahan komponen warna citra bisa dilihat pada gambar di bawah ini.

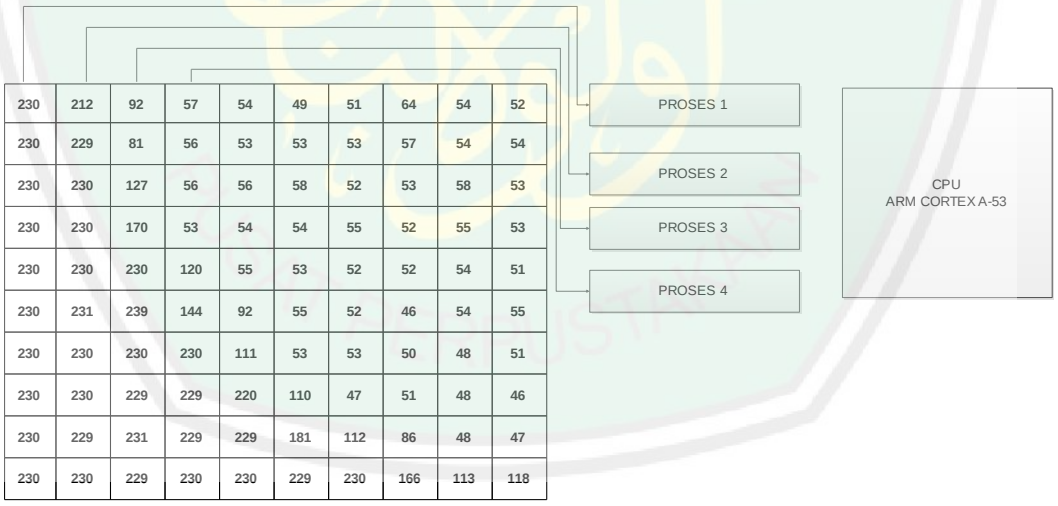


**Gambar 4.2** Proses Grayscale

Proses selanjutnya yaitu pemrosesan citra dengan konvolusi menggunakan 25 pasangan kernel LAWS. Nilai piksel hasil proses konvolusi kemudian diproses menggunakan pergeseran window dengan ukuran 15x15. Nilai piksel hasil proses pergeseran window kemudian dinormalkan dihingga didapatkan nilai piksel normal dengan rentang 0-255. Hasil citra dengan fitur energi tekstur bisa dilihat pada gambar di bawah ini.



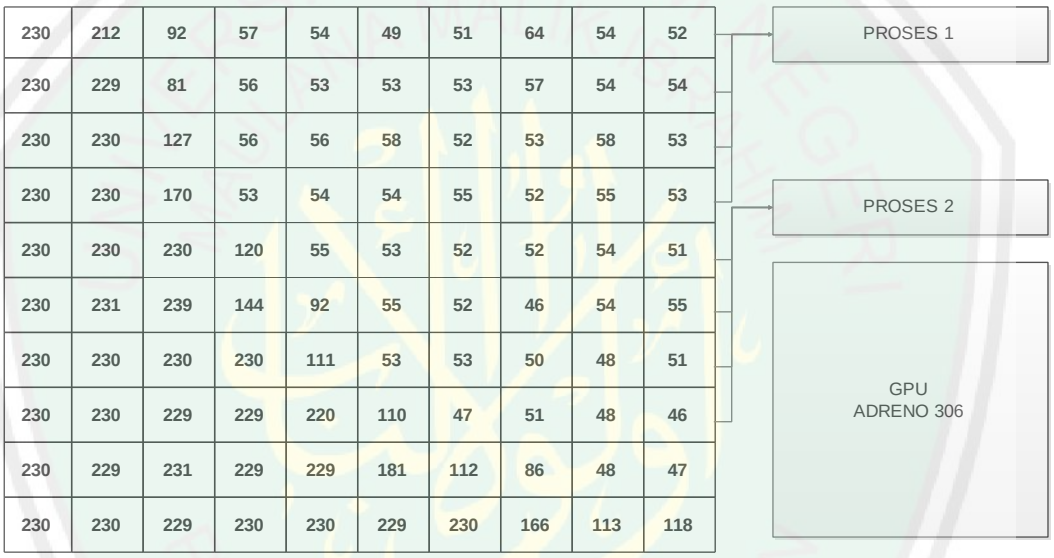
**Gambar 4.3** Proses LAWS Filter




**Gambar 4.4** Proses Uji Coba Komputasi Sekuensial

Proses pengujian pada komputasi sekuensial diterapkan menggunakan cpu perangkat android. Dalam hal ini cpu yang digunakan adalah cpu arsitketur ARM Cortex-A53 dengan kecepatan 1,21 Ghz. Proses uji coba pada komputasi sekuensial dilakukan satu persatu pada setiap piksel citra. Proses pengujian

komputasi sekuensial dilakukan berurutan satu persatu oleh cpu sesuai dengan koordinat piksel. Proses piksel ke 1 dilakukan sampai tugas yang diberikan ke cpu selesai, kemudian beralih ke proses piksel ke 2 dilakukan sampai tugas yang diberikan ke cpu pada proses piksel ke 2 selesai sampai proses piksel ke n. Proses piksel yang berada di atas atau diantrian setelah proses piksel terdepan tidak akan diproses sebelum proses-proses piksel sebelumnya selesai diproses oleh cpu. Hal tersebut mengakibatkan memakan waktu lama.




**Gambar 4.5** Proses Uji Coba Komputasi Paralel

Proses pengujian pada komputasi sekuensial diterapkan menggunakan gpu perangkat android. Dalam hal ini menggunakan gpu dengan arsitketur Adreno 306 dengan 4 inti pemroses. Proses uji coba pada komputasi paralel dilakukan bersama-sama pada setiap 4 piksel citra. Proses pengujian komputasi paralel dilakukan berurutan setiap 4 piksel citra oleh gpu sesuai dengan koordinat piksel. Proses piksel ke 1 - 4 dilakukan sampai tugas yang diberikan ke gpu selesai, kemudian beralih ke proses piksel ke 4-8 dilakukan sampai tugas yang diberikan ke gpu pada proses piksel ke 4-8 selesai sampai proses piksel ke n. Proses piksel

yang berada di atas atau diantrian setelah proses piksel terdepan tidak akan diproses sebelum proses-proses piksel sebelumnya selesai diproses oleh gpu.

### 4.1.2 Data Uji

Data uji pada penelitian ini yaitu data citra daun cocor bebek dengan tiga ukuran berbeda, meliputi 512x512, 1024x1024 dan 2048x2048. Data uji pada penenilitian ini didapatkan dengan cara mengambil gambar citra daun cocor bebek menggunakan kamera depan ponsel cerdas Xiaomi Redmi 2 dengan spesifikasi kamerea 8 megapiksel. Teknik pengambilan gambar citra daun cocor bebek diambil tegak lurus dengan titik tengan daun dengan jarak 25 cm dari permukaan daun cocor bebek. Daun cocor bebek diprotret sebanyak satu kali kemudian digandakan sebanyak 30 kali sehingga didapatkan data citra daun cocor bebek sebanyak 30 citra daun cocor bebek. Posisi duan dalam pengambilan gambar yaitu sama dengan posisi daun tegak lurus dengan batang daun berada pada bagian atas citra. Kemudian hasil citra protret kamera dirubah ukurannya menjadi citra dengan ukuran 512x512, citra ukuran 1024x1024 dan citra ukuran 2048x2048. Ukuran citra maksimal yang digunakan disesuaikan dengan kapasitas RAM ponsel pintar, karena ukuran citra lebih dari 2048x2048 akan melebihi kapasitas RAM ponsel pintar. Pengambilan 30 data citra duan cocor bebek berdasarkan ukuran sampel yang layak dalam penelitian adalah antara 30 sampai dengan 500 (Sugiyono, 2012). Jumlah sampel yang diambil pada penlitian ini mengambil sampel meninimal dari rentang ukuran sampel yang sudah dijelaskan oleh sugiyono.

**Tabel 4.1** Data Citra Daun Cocor Bebek

| NO | NAMA | UKURAN | | |
|---|---|---|---|---|
| | | 512x512 | 1024x1024 | 2048x2048 |
| 1 | Data Citra 01 | Citra 01 | Citra 01 | Citra 01 |
| 2 | Data Citra 02 | Citra 02 | Citra 02 | Citra 02 |
| 3 | Data Citra 03 | Citra 03 | Citra 03 | Citra 03 |
| 4 | Data Citra 04 | Citra 04 | Citra 04 | Citra 04 |
| 5 | Data Citra 05 | Citra 05 | Citra 05 | Citra 05 |
| 6 | Data Citra 06 | Citra 06 | Citra 06 | Citra 06 |
| 7 | Data Citra 07 | Citra 07 | Citra 07 | Citra 07 |
| 8 | Data Citra 08 | Citra 08 | Citra 08 | Citra 08 |
| 9 | Data Citra 09 | Citra 09 | Citra 09 | Citra 09 |
| 10 | Data Citra 10 | Citra 10 | Citra 10 | Citra 10 |
| 11 | Data Citra 11 | Citra 11 | Citra 11 | Citra 11 |
| 12 | Data Citra 12 | Citra 12 | Citra 12 | Citra 12 |
| 13 | Data Citra 13 | Citra 13 | Citra 13 | Citra 13 |
| 14 | Data Citra 14 | Citra 14 | Citra 14 | Citra 14 |
| 15 | Data Citra 15 | Citra 15 | Citra 15 | Citra 15 |
| 16 | Data Citra 16 | Citra 16 | Citra 16 | Citra 16 |
| 17 | Data Citra 17 | Citra 17 | Citra 17 | Citra 17 |
| 18 | Data Citra 18 | Citra 18 | Citra 18 | Citra 18 |
| 19 | Data Citra 19 | Citra 19 | Citra 19 | Citra 19 |
| 20 | Data Citra 20 | Citra 20 | Citra 20 | Citra 20 |
| 21 | Data Citra 21 | Citra 21 | Citra 21 | Citra 21 |
| 22 | Data Citra 22 | Citra 22 | Citra 22 | Citra 22 |
| 23 | Data Citra 23 | Citra 23 | Citra 23 | Citra 23 |
| 24 | Data Citra 24 | Citra 24 | Citra 24 | Citra 24 |
| 25 | Data Citra 25 | Citra 25 | Citra 25 | Citra 25 |
| 26 | Data Citra 26 | Citra 26 | Citra 26 | Citra 26 |
| 27 | Data Citra 27 | Citra 27 | Citra 27 | Citra 27 |
| 28 | Data Citra 28 | Citra 28 | Citra 28 | Citra 28 |
| 29 | Data Citra 29 | Citra 29 | Citra 29 | Citra 29 |
| 30 | Data Citra 30 | Citra 30 | Citra 30 | Citra 30 |

## 4.2 Hasil Uji Coba

Pemaparan hasil uji coba diterangkan menggunakan bagan citra hasil pemrosesan penentuan ukuran energi tekstur menggunakan metode LAWS. Hasil pengukuran konsumsi waktu komputasi dipaparkan dengan tabel dan diperinci ke

dalam bagan grafik perbandingan konsumsi waktu komputasi secara sekusensial dan paralel.

### 4.2.1 Hasil Implementasi Metode LAWS

Berikut dijelaskan citra hasil penentuan ukuran energi tekstur menggunakan metode LAWS. Citra hasil antara pemrosesan menggunakan komputasi sekusensial dan paralel mempunyai citra hasil yang sama. Pemabahasan citra diurutkan mulai dari citra hasil pemrosesan menggunakan kernel LAWS L5L5 sampai dengan kernel LAWS W5W5.

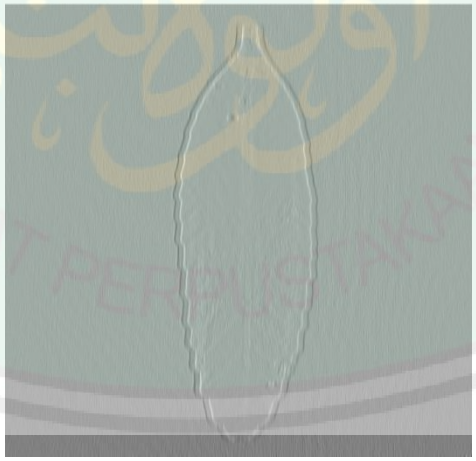
**Gambar 4.6** Citra Hasil Kernel LAWS L5L5

Citra hasil pada gambar 4.6 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS L5L5 yang merupakan kernel untuk pererataan lokal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5978, 25156, dan 83722 menggunakan eksekusi sekuensial dan angka 417, 1773, dan 5401 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

Citra hasil pada gambar 4.7 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS L5E5 yang merupakan kernel untuk pererataan lokal

pada eksekusi vertikal dan pendeteksian tepi pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5865, 24935, dan 81997 menggunakan eksekusi sekuensial dan angka 443, 1708, dan 5688 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.7** Citra Hasil Kernel LAWS L5E5



**Gambar 4.8** Citra Hasil Kernel LAWS L5S5

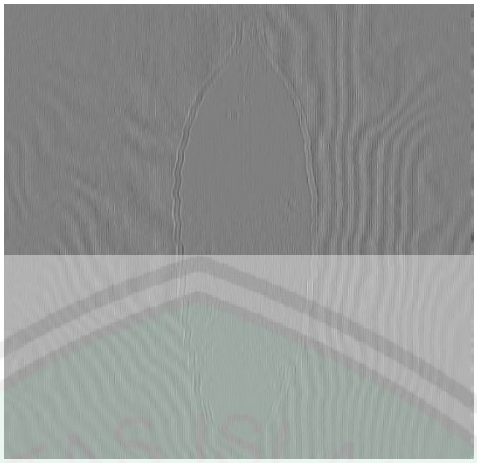
Citra hasil pada gambar 4.8 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS L5S5 yang merupakan kernel untuk pererataan lokal pada eksekusi vertikal dan pendeteksian titik pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5865, 24935,

dan 81997 menggunakan eksekusi sekuensial dan angka 443, 1708, dan 5688 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.9** Citra Hasil Kernel LAWS L5R5

Citra hasil pada gambar 4.9 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS L5R5 yang merupakan kernel untuk pererataan lokal pada eksekusi vertikal dan pendeteksian riak pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5913, 24892, dan 81140 menggunakan eksekusi sekuensial dan angka 437, 1735, dan 5756 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

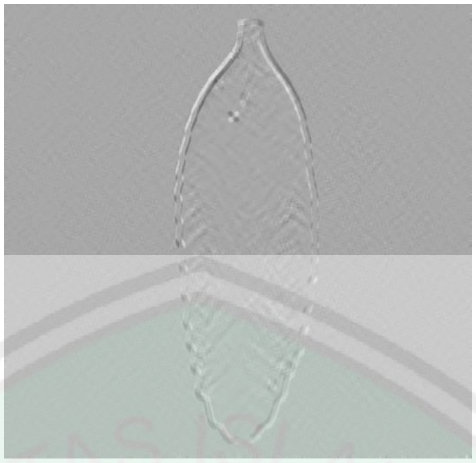
Citra hasil pada gambar 4.10 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS L5W5 yang merupakan kernel untuk pererataan lokal pada eksekusi vertikal dan pendeteksian gelombang pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5839, 24937, dan 82060 menggunakan eksekusi sekuensial dan angka 435, 1728, dan 5899 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.



**Gambar 4.10** Citra Hasil Kernel LAWS L5W5

**Gambar 4.11** Citra Hasil Kernel LAWS E5L5

Citra hasil pada gambar 4.11 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS E5L5 yang merupakan kernel untuk pendeteksian tepi pada eksekusi vertikal dan pererataan lokal pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 6176, 26861, dan 80913 menggunakan eksekusi sekuensial dan angka 447, 1735, dan 5426 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.



**Gambar 4.12** Citra Hasil Kernel LAWS E5E5

Citra hasil pada gambar 4.12 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS E5E5 yang merupakan kernel untuk pendeteksian tepi pada eksekusi vertikal dan pendeteksian tepi pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5861, 25072, dan 80954 menggunakan eksekusi sekuensial dan angka 435, 2183, dan 6156 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.13** Citra Hasil Kernel LAWS E5S5

Citra hasil pada gambar 4.13 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS E5S5 yang merupakan kernel untuk pendeteksian

tepi pada eksekusi vertikal dan pendeteksian titik pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5947, 25080, dan 80826 menggunakan eksekusi sekuensial dan angka 432, 1690, dan 5665 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.14** Citra Hasil Kernel LAWS E5R5

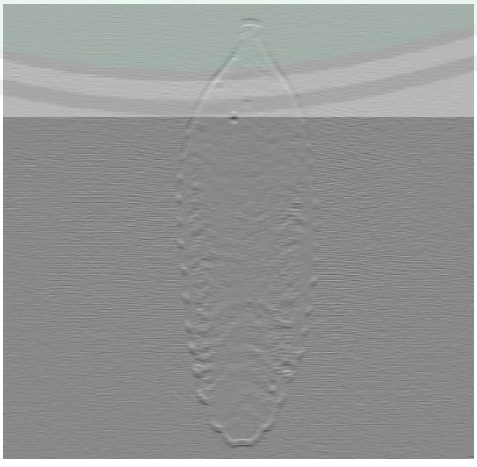
**Gambar 4.15** Citra Hasil Kernel LAWS E5W5

Citra hasil pada gambar 4.14 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS E5R5 yang merupakan kernel untuk pendeteksian tepi pada eksekusi vertikal dan pendeteksian riak pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka

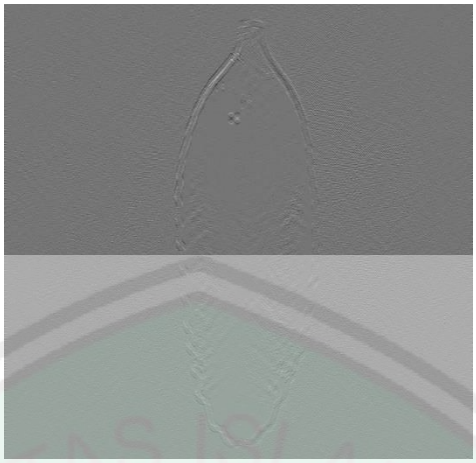
5862, 25080, dan 82243 menggunakan eksekusi sekuensial dan angka 467, 1701, dan 5737 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

Citra hasil pada gambar 4.15 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS E5W5 yang merupakan kernel untuk pendeteksian tepi pada eksekusi vertikal dan pendeteksian gelombang pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5948, 25390, dan 80826 menggunakan eksekusi sekuensial dan angka 490, 1764, dan 5451 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

Citra hasil pada gambar 4.16 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS S5L5 yang merupakan kernel untuk pendeteksian titik pada eksekusi vertikal dan pererataan lokal pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5883, 25178, dan 81860 menggunakan eksekusi sekuensial dan angka 433, 1729, dan 5380 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.



**Gambar 4.16** Citra Hasil Kernel LAWS S5L5



**Gambar 4.17** Citra Hasil Kernel LAWS S5E5

Citra hasil pada gambar 4.17 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS S5E5 yang merupakan kernel untuk pendeteksian titik pada eksekusi vertikal dan pendeteksian tepi pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5975, 24931, dan 81427 menggunakan eksekusi sekuensial dan angka 432, 1716, dan 5461 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

Citra hasil pada gambar 4.18 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS S5S5 yang merupakan kernel untuk pendeteksian titik pada eksekusi vertikal dan pendeteksian titik' pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5882, 25025, dan 83260 menggunakan eksekusi sekuensial dan angka 450, 1700, dan 5747 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

Citra hasil pada gambar 4.19 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS S5R5 yang merupakan kernel untuk pendeteksian titik pada eksekusi vertikal dan pendeteksian riak pada eksekusi horisontal.

Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5995, 24974, dan 88309 menggunakan eksekusi sekuensial dan angka 463, 1786, dan 6094 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.18** Citra Hasil Kernel LAWS S5S5

**Gambar 4.19** Citra Hasil Kernel LAWS S5R5

Citra hasil pada gambar 4.20 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS S5W5 yang merupakan kernel untuk pendeteksian titik pada eksekusi vertikal dan pendeteksian gelombang pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5853, 25122, dan 83753 menggunakan eksekusi sekuensial dan angka 469, 1700,

dan 5601 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.20** Citra Hasil Kernel LAWS S5W5

**Gambar 4.21** Citra Hasil Kernel LAWS R5L5

Citra hasil pada gambar 4.21 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS R5L5 yang merupakan kernel untuk pendeteksian riak pada eksekusi vertikal dan pererataan lokal pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5826, 25154, dan 83842 menggunakan eksekusi sekuensial dan angka 481, 1736, dan 5538 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

Citra hasil pada gambar 4.22 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS R5E5 yang merupakan kernel untuk pendeteksian riak pada eksekusi vertikal dan pendeteksian tepi pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5811, 25261, dan 84654 menggunakan eksekusi sekuensial dan angka 503, 1737, dan 5684 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.22** Citra Hasil Kernel LAWS R5E5

**Gambar 4.23** Citra Hasil Kernel LAWS R5S5

Citra hasil pada gambar 4.23 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS R5S5 yang merupakan kernel untuk pendeteksian

riak pada eksekusi vertikal dan pendeteksian titik pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5851, 25150, dan 83589 menggunakan eksekusi sekuensial dan angka 490, 1785, dan 5404 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.24** Citra Hasil Kernel LAWS R5R5

Citra hasil pada gambar 4.24 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS R5W5 yang merupakan kernel untuk pendeteksian riak pada eksekusi vertikal dan pendeteksian riak pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5827, 24961, dan 83986 menggunakan eksekusi sekuensial dan angka 423, 1735, dan 5566 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

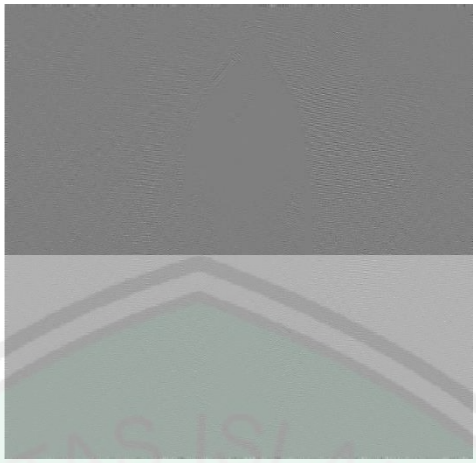
Citra hasil pada gambar 4.25 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS R5W5 yang merupakan kernel untuk pendeteksian riak pada eksekusi vertikal dan pendeteksian gelombang pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5859, 25026, dan 85206 menggunakan eksekusi sekuensial dan angka 449, 1724,

dan 5551 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.25** Citra Hasil Kernel LAWS R5W5

**Gambar 4.26** Citra Hasil Kernel LAWS W5L5

Citra hasil pada gambar 4.26 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS W5L5 yang merupakan kernel untuk pererataan lokal pada eksekusi vertikal dan pendeteksian gelombang pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5981, 25077, dan 89067 menggunakan eksekusi sekuensial dan angka 442, 1699, dan 5543 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.



**Gambar 4.27** Citra Hasil Kernel LAWS W5E5

Citra hasil pada gambar 4.27 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS W5E5 yang merupakan kernel untuk pendeteksian gelombang pada eksekusi vertikal dan pendeteksian tepi pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5955, 24997, dan 86218 menggunakan eksekusi sekuensial dan angka 446, 1783, dan 5886 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.28** Citra Hasil Kernel LAWS W5S5

Citra hasil pada gambar 4.28 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS W5S5 yang merupakan kernel untuk pendeteksian

gelombang pada eksekusi vertikal dan pendeteksian titik pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5992, 25175, dan 87170 menggunakan eksekusi sekuensial dan angka 421, 1654, dan 5527 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

**Gambar 4.29** Citra Hasil Kernel LAWS W5R5

**Gambar 4.30** Citra Hasil Kernel LAWS W5W5

Citra hasil pada gambar 4.29 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS W5R5 yang merupakan kernel untuk pendeteksian gelombang pada eksekusi vertikal dan pendeteksian riak pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka

5592, 25040, dan 85065 menggunakan eksekusi sekuensial dan angka 469, 1745, dan 5496 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

Citra hasil pada gambar 4.30 merupakan hasil proses esktraksi fitur menggunakan kernel LAWS W5W5 yang merupakan kernel untuk pendeteksian gelombang pada eksekusi vertikal dan pendeteksian gelombang pada eksekusi horisontal. Konsumsi waktu yang diperlukan untuk proses ekstraksi menunjukkan angka 5951, 24980, dan 84345 menggunakan eksekusi sekuensial dan angka 472, 1720, dan 5508 menggunakan eksekusi paralel pada satuan *milisecond*(ms) dengan ukuran citra secara berurutan dari kiri 512x512, 1024x1024 dan 2048x2048.

Implementasi metode LAWS membutuhkan 3 tahap pemrosesan gambar yaitu konvolusi citra menggunakan kernel LAWS dengan ukuran 5x5, proses pergerseran window dengan ukuran window 15x15 dan kemudian penormalan nilai hasil dua proses sebelumnya ke nilai *range* piksel. Implementasi sekuensial metode LAWS membutuhkan tiga kali perulangan pemrosesan tiap piksel citra berdasarkan pada ukuran citra. Proses konvolusi membutuhkan 25 perulangan untuk satu kali pemrosesan piksel citra. Proses pergeseran window membutuhkan 225 perulangan untuk satu kali pemrosesan piksel citra.

Implementasi metode LAWS secara paralel membutuhkan perulangan pemrosesan piksel lebih sedikit karena beban pemrosesan dialokasikan ke banyak prosssor dalam GPU. Perbandingan beban pemrosesan mencapai 4:1 sesuai dengan inti pemroses GPU yang digunakan. Dua puluh lima perulangan untuk satu pemrosesan konvolusi piksel dialokasikan ke 4 inti pemroses, dalam satu kali

eksekusi mencapai 100 perulangan terselesaikan. Perulangan proses pergeseran window mencapai 1000 perulangan dalam satu kali eksekusi.

Berdasarkan hasil yang sudah didapatkan dari proses konvolusi berbasis kernel LAWS, dapat dilihat hasil yang dipaparkan dalam bentuk citra merupakan hasil pengukuran energi tekstur dari citra daun cocor bebek. Garis-garis yang ditimbulkan pada latarbelakang citra daun dikarenakan efek dari tekstur kain yang digunakan pada saat pengambilan gambar. Efek tersebut terbentuk karena proses pengambilan foto menggunakan cahaya kamera ponsel pintar menimbulkan detail dari tekstur kain yang digunakan.

## 4.2 Hasil Pengukuran Konsumsi Waktu Komputasi

Setelah pemaparan hasil implementasi metode LAWS secara sekuensial dan paralel, pemaparan hasil pengukuran konsumsi waktu komputasi metode LAWS secara sekuensial dan paralel. Hasil pengukuran konsumsi waktu berdasarkan ukuran citra yang diproses yaitu 512x512, 1024x1024 dan 2048x2048. Data hasil pengukuran citra berdasarkan kernel LAWS yang digunakan dan ukuran citra disajikan dalam sebuah tabel. Dalam tabel tersebut menyajikan citra hasil pemrosesan berdasarkan kernel LAWS, konsumsi waktu komputasi secara sekuensial dan paralel berdasarkan ukuran citra. Data yang ditampilkan pada tabel merupakan data rata-rata hasil pengukuran setiap satu kali percobaan pada data citra daun cocor bebek. Selain dalam bentuk tabel, pemaparan hasil pengukuran konsumsi waktu komputasi metode LAWS dalam bentuk grafik. Peningkatan kecepatan komputasi metode LAWS dari eksekusi sekuensial ke ekskusi paralel dipaparkan dalam bentuk grafik.

**Tabel 4.2** Hasil Rata-rata Pengukuran Waktu Komputasi Citra Ukuran 512

| NO | KERNEL LAWS | CITRA HASIL | WAKTU KOMPUTASI (ms) | |
|---|---|---|---|---|
| | | | SEKUENSIAL | PARALEL |
| 1 | L5L5 | Citra 31 | 5896,5 | 470,7 |
| 2 | L5E5 | Citra 32 | 5913,1 | 464,7 |
| 3 | L5S5 | Citra 33 | 5946 | 475,6 |
| 4 | L5R5 | Citra 34 | 5938,6 | 473,1 |
| 5 | L5W5 | Citra 35 | 5919,7 | 464,9 |
| 6 | E5L5 | Citra 36 | 5889,9 | 450 |
| 7 | E5E5 | Citra 37 | 5843,6 | 472,4 |
| 8 | E5S5 | Citra 38 | 5899,7 | 463,6 |
| 9 | E5R5 | Citra 39 | 5927,2 | 468,5 |
| 10 | E5W5 | Citra 40 | 5930,1 | 466,4 |
| 11 | S5L5 | Citra 41 | 5920,8 | 465,8 |
| 12 | S5E5 | Citra 42 | 5921,3 | 467.3 |
| 13 | S5S5 | Citra 43 | 6089,1 | 473,2 |
| 14 | S5R5 | Citra 44 | 5929,1 | 468,8 |
| 15 | S5W5 | Citra 45 | 5923,9 | 464,7 |
| 16 | R5L5 | Citra 46 | 5892,1 | 459 |
| 17 | R5E5 | Citra 47 | 5919,5 | 470,7 |
| 18 | R5S5 | Citra 48 | 5919,3 | 466,8 |
| 19 | R5R5 | Citra 49 | 5898,9 | 467,5 |
| 20 | R5W5 | Citra 50 | 5906,6 | 465,9 |
| 21 | W5L5 | Citra 51 | 5953,1 | 465,3 |
| 22 | W5E5 | Citra 52 | 5974,3 | 467,6 |
| 23 | W5S5 | Citra 53 | 5980,3 | 466,7 |
| 24 | W5R5 | Citra 54 | 5979,8 | 462,8 |
| 25 | W5W5 | Citra 55 | 5966,2 | 458,1 |

**Tabel 4.3** Hasil Rata-rata Pengukuran Waktu Komputasi Citra Ukuran 1024

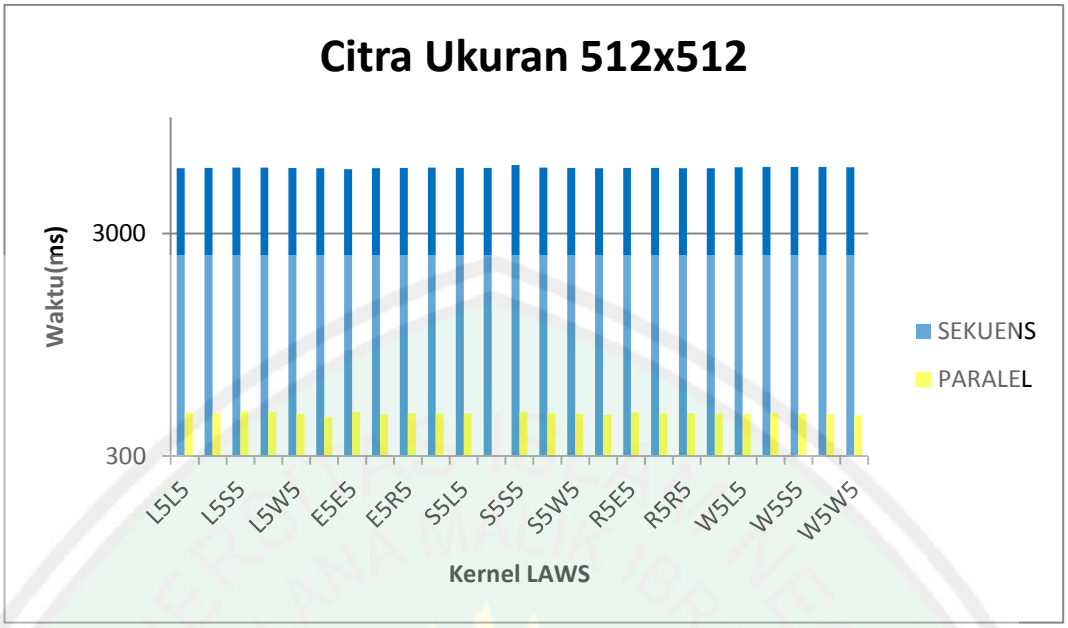
| NO | KERNEL LAWS | CITRA HASIL | WAKTU KOMPUTASI (ms) | |
|---|---|---|---|---|
| | | | SEKUENSIAL | PARALEL |
| 1 | L5L5 | Citra 31 | 25235,5 | 1731,8 |
| 2 | L5E5 | Citra 32 | 25052 | 1714,8 |
| 3 | L5S5 | Citra 33 | 25027,7 | 1726,4 |
| 4 | L5R5 | Citra 34 | 24881,3 | 1734,6 |
| 5 | L5W5 | Citra 35 | 24861,8 | 1728,3 |
| 6 | E5L5 | Citra 36 | 25301,4 | 1737,9 |
| 7 | E5E5 | Citra 37 | 25036,2 | 1769,2 |
| 8 | E5S5 | Citra 38 | 25152,9 | 1717,6 |
| 9 | E5R5 | Citra 39 | 25122 | 1721,7 |
| 10 | E5W5 | Citra 40 | 25142,7 | 1727,4 |
| 11 | S5L5 | Citra 41 | 25174,5 | 1716,6 |
| 12 | S5E5 | Citra 42 | 25069,3 | 1724,2 |
| 13 | S5S5 | Citra 43 | 25059 | 1725,3 |
| 14 | S5R5 | Citra 44 | 25108,7 | 1722,8 |
| 15 | S5W5 | Citra 45 | 25106,9 | 1720,4 |
| 16 | R5L5 | Citra 46 | 25056,6 | 1715,6 |
| 17 | R5E5 | Citra 47 | 25138,7 | 1725,9 |
| 18 | R5S5 | Citra 48 | 25179,7 | 1741,2 |
| 19 | R5R5 | Citra 49 | 25203,1 | 1724 |
| 20 | R5W5 | Citra 50 | 25152,3 | 1717 |
| 21 | W5L5 | Citra 51 | 25080,4 | 1712,4 |
| 22 | W5E5 | Citra 52 | 25112,2 | 1733,3 |
| 23 | W5S5 | Citra 53 | 25139,5 | 1704,6 |
| 24 | W5R5 | Citra 54 | 25111,3 | 1733,2 |
| 25 | W5W5 | Citra 55 | 25067,8 | 1721,2 |

**Tabel 4.4** Hasil Rata-rata Pengukuran Waktu Komputasi Citra Ukuran 2048

| NO | KERNEL LAWS | CITRA HASIL | WAKTU KOMPUTASI (ms) | |
|---|---|---|---|---|
| | | | SEKUENSIAL | PARALEL |
| 1 | L5L5 | Citra 31 | 84088,3 | 5488,7 |
| 2 | L5E5 | Citra 32 | 82933,8 | 5474,5 |
| 3 | L5S5 | Citra 33 | 8289,3 | 5501,9 |
| 4 | L5R5 | Citra 34 | 82739,9 | 5416,3 |
| 5 | L5W5 | Citra 35 | 82708,6 | 5452,3 |
| 6 | E5L5 | Citra 36 | 82640,6 | 5373 |
| 7 | E5E5 | Citra 37 | 82543,2 | 5473,8 |
| 8 | E5S5 | Citra 38 | 82583,8 | 5400,8 |
| 9 | E5R5 | Citra 39 | 82755,6 | 5416,3 |
| 10 | E5W5 | Citra 40 | 81674,1 | 5488 |
| 11 | S5L5 | Citra 41 | 81996,7 | 5388,5 |
| 12 | S5E5 | Citra 42 | 82256,4 | 5374,3 |
| 13 | S5S5 | Citra 43 | 82287,3 | 5412,5 |
| 14 | S5R5 | Citra 44 | 82578,3 | 5410,1 |
| 15 | S5W5 | Citra 45 | 81970,4 | 5397,8 |
| 16 | R5L5 | Citra 46 | 82703,1 | 5373,9 |
| 17 | R5E5 | Citra 47 | 82259,6 | 5452,4 |
| 18 | R5S5 | Citra 48 | 82531,7 | 5365,2 |
| 19 | R5R5 | Citra 49 | 82080,4 | 5375,5 |
| 20 | R5W5 | Citra 50 | 82427,5 | 5370,9 |
| 21 | W5L5 | Citra 51 | 82891,5 | 5359,8 |
| 22 | W5E5 | Citra 52 | 82581,7 | 5417,1 |
| 23 | W5S5 | Citra 53 | 82686,9 | 5446,6 |
| 24 | W5R5 | Citra 54 | 82579,6 | 5435,4 |
| 25 | W5W5 | Citra 55 | 82475 | 5390 |

Pengukuran konsumsi waktu komputasi dilakukan dua puluh lima kali pada setiap ukuran citra berdasarkan jumlah pasangan 25 kernel LAWS yang digunakan. Pengukuran konsumsi waktu komputasi setiap 25 pasangan kernel LAWS dilakukan satu kali dikarenakan proses komputasi 25 pasangan kernel LAWS tidak dipengaruhi oleh kernel LAWS yang digunakan. Kernel LAWS L5L5 dengan kernel LAWS lainnya memiliki waktu komputasi relatif sama. Konsumsi waktu komputasi dipengaruhi oleh besarnya ukuran citra. Pengukuran konsumsi waktu komputasi menggunakan satuan *milisecond* (ms) didapat dari pemanggilan method *System.nanoTime();* dari kelas *System* dalam bahasa pemrograman Java. Waktu hasil pemanggilan method *System.nanoTime()* dengan satuan *nanosecond* kemudian dibagi dengan 1000000 sehingga didapatkan waktu dengan satuan *milisecond.*

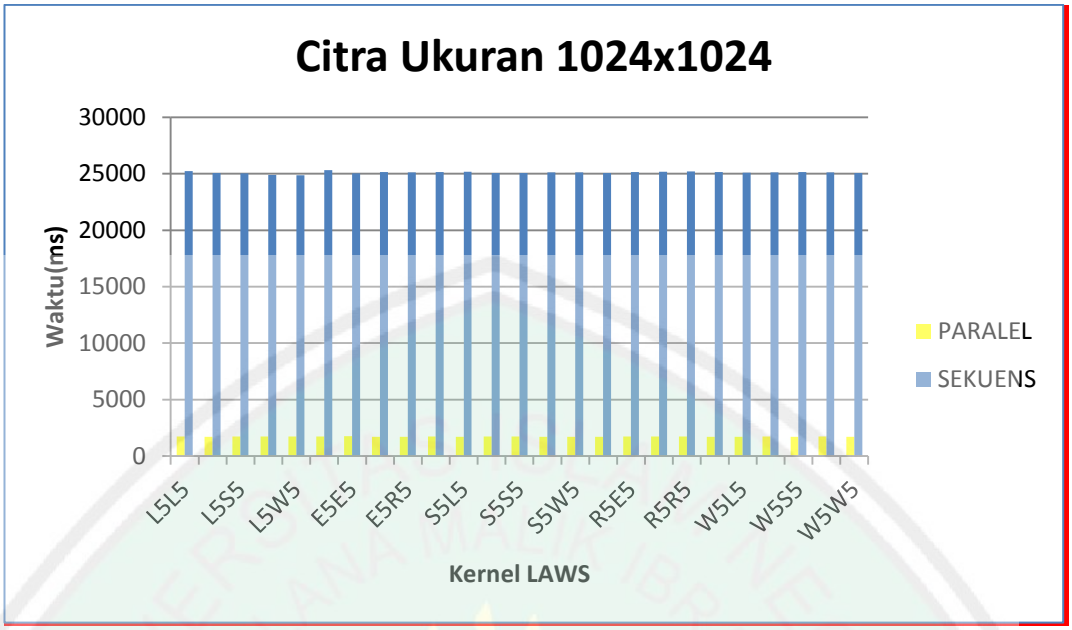
Citra dengan ukuran 512x512 memiliki waktu komputasi dengan rata-rata nilai 5917,32 ms pada komputasi sekuens dan 450,72 ms pada komputasi paralel. Citra dengan ukuran 1024x1024 memiliki waktu komputasi dengan rata-rata nilai 15170,84 ms pada komputasi sekuens dan 1746 ms pada komputasi paralel. Citra dengan ukuran 512x512 memiliki waktu komputasi dengan rata-rata nilai 83521,68 ms pada komputasi sekuens dan 5654,04 ms pada komputasi paralel. Perbandingan hasil pengukuran konsumsi waktu komputasi antara hasil pengukuran komputasi sekuens dengan hasil pengukuran komputasi paralel menunjukkan hasil yang jauh berbeda. Perbandingan hasil pengukuran mencapai 13 berbanding 1 pada ukuran citra 512x512, 14 berbanding 1 pada ukuran citra 1024x1024 dan 14 berbanding 1 pada ukuran citra 2048x2048.



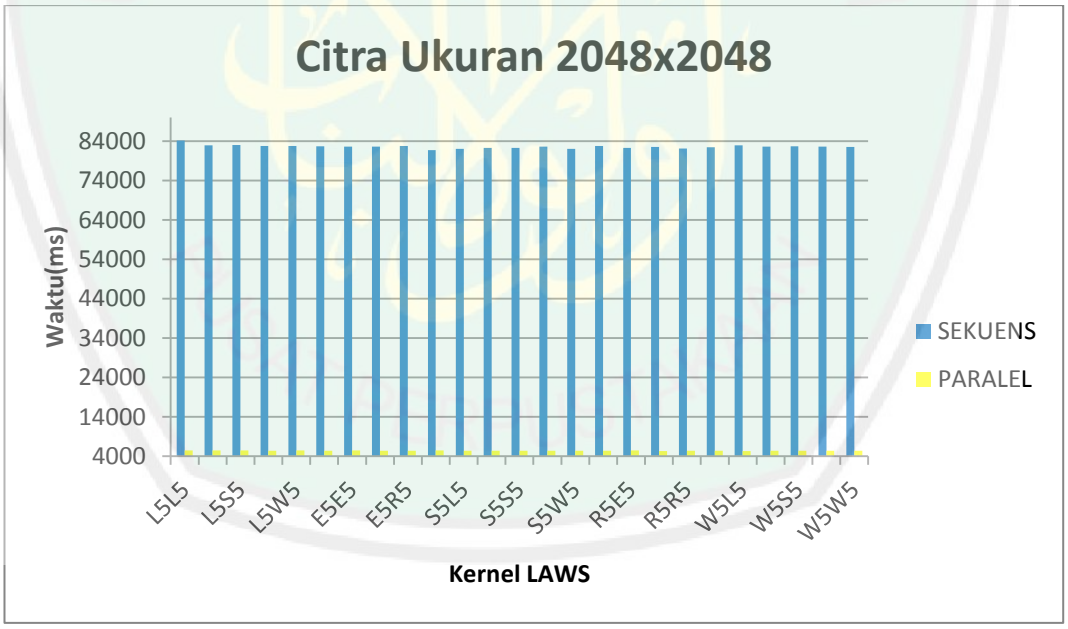

**Gambar 4.31** Grafik Hasil Pengukuran Konsumsi Waktu Komputasi Citra 512x512

Hasil pengukuran konsumsi waktu komputasi sekuens dan paralel pada citra dengan ukuran 512x512 pada grafik gambar 4.31 menunjukkan perbedaan yang signifikan dari hasil pengukuran antara komputasi paralel dengan komputasi sekuens. Hasil pengukuran 25 pasangan kernel LAWS pada komputasi sekuens cenderung berada pada angka 5695 ms, hasil pengukuran ini stabil dalam 25 kali percobaan berdasarkan kernel LAWS. Sedangkan pada komputasi paralel nilai hasil pengukuran stabil pada angka 447 ms.

Grafik pada gambar 4.32 mempresentasikan hasil pengukuran konsumsi waktu komputasi sekuens dan paralel pada citra dengan ukuran 1024x1024. Hasil pengukuran dari 25 kali percobaan menunjukkan nilai komsumsi waktu komputasi sekuens yang stabil pada angka 25103 ms, sedangkan nilai konsumsi waktu komputasi paralel stabil pada angka 1725 ms.



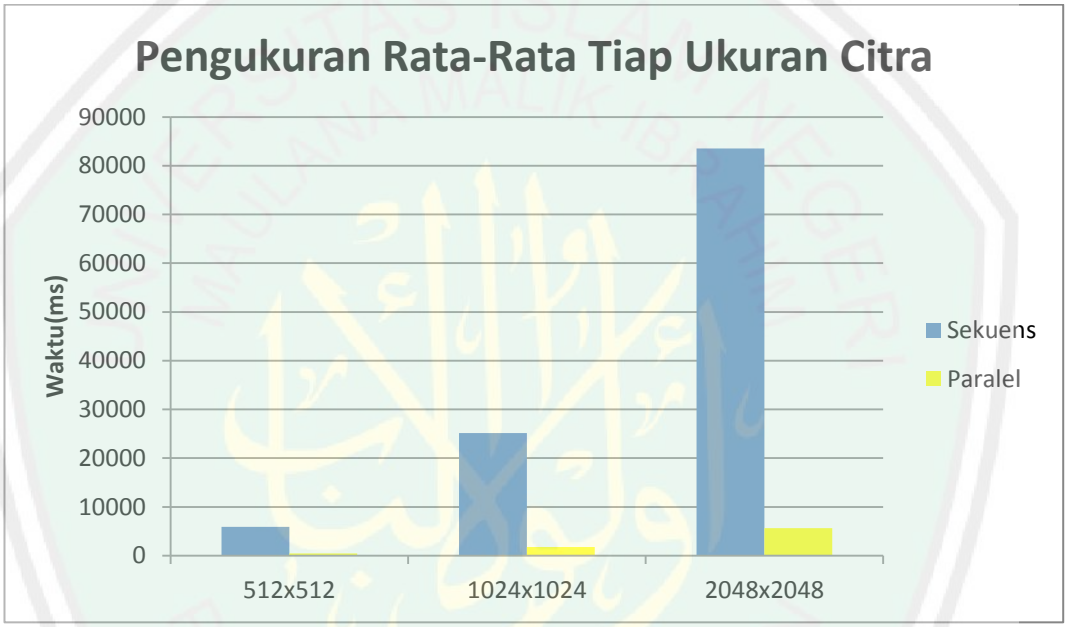

**Gambar 4.31** Grafik Hasil Pengukuran Konsumsi Waktu Komputasi Citra 1024x1024




**Gambar 4.33** Grafik Hasil Pengukuran Konsumsi Waktu Komputasi Citra 2048x2048

Grafik hasil pengukuran konsumsi waktu komputasi sekuens dan paralel pada citra dengan ukuran 2048x2048 yang ditunjukkan pada gambar 4.33

menggambarkan hasil pengukuran konsumsi waktu komputasi sekuens stabil pada angka 82558 ms sedangkan konsumsi waktu komputasi paralel stabil pada angka 5416 ms. Perbedaan hasil pengukuran antara 25 pasangan kernel LAWS yang digunakan tidak besar. Hasil pengukuran antara 25 pasangan kernel yang digunakan cenderung stabil dan presisi pada hasil pengukuran antara kernel satu dengan kernel lainnya.




**Gambar 4.34** Grafik Rata-rata Waktu Komputasi Metode LAWS

Peningkatan kecepatan komputasi metode LAWS pada citra dengan ukuran 512x512, 1024x1024 dan 2048x2048 mencapai angka +92% dengan menggunakan komputasi paralel. Peningkatan komputasi metode LAWS secara sekuens dengan menggunakan komputasi paralel menunjukkan hasil yang signifikan. Peningkatan kecepatan komputasi metode LAWS secara sekuens dengan komputasi paralel hampir mencapai angka 100 persen yang menunjukkan bahwa komputasi paralel hampir sempurna sebagai metode untuk peningkatan

kecepatan komputasi sekuens. Prosentase peningkatan kecepatan didapatkan denga rumus 4.1 (Pierce, 2016).

$$\frac{x'-x}{|x|} * 100\% \qquad (4.1)$$

Dimana :

X = Nilai Lama

X' = Nilai Baru

Berdasarkan penelitian yang telah dilakukan, hasil pengukuran pemrosesan komputasi paralel metode LAWS menunjukkan efisiensi waktu yang lebih baik dibanding dengan hasil pengukuran waktu komputasi sekuens. Efisiensi waktu yang telah didapatkan dari hasil penelitian sesuai dengan perintah Allah ﷻ agar tidak melakukan pemborosan yang di terangkan pada surat Al-Isra' ayat 26-27 yang berbunyi :

وَءَاتِ ذَا ٱلْقُرْبَىٰ حَقَّهُۥ وَٱلْمِسْكِينَ وَٱبْنَ ٱلسَّبِيلِ وَلَا تُبَذِّرْ تَبْذِيرًا ﴿٢٦﴾ إِنَّ ٱلْمُبَذِّرِينَ كَانُوٓا۟ إِخْوَٰنَ ٱلشَّيَٰطِينِ ۖ وَكَانَ ٱلشَّيْطَٰنُ لِرَبِّهِۦ كَفُورًا ﴿٢٧﴾

Artinya : "*26. dan berikanlah kepada keluarga-keluarga yang dekat akan haknya, kepada orang miskin dan orang yang dalam perjalanan dan janganlah kamu menghambur-hamburkan (hartamu) secara boros.*
*27. Sesungguhnya pemboros-pemboros itu adalah saudara-saudara syaitan dan syaitan itu adalah sangat ingkar kepada Tuhannya."*

Ayat diatas menjelaskan perintah untuk tidak melakukan pemborosan terhadap pembelanjaan harta, karena pemborosan atau menghamburkan-hamburkan harta termasuk dalam perbuatan syaitan (Ibnu Kasir, 2000).

Permasalahan mengenai pentingnya memperhatikan efesiensi waktu juga diterangkan oleh Nabi Muhammad ﷺ dalam sebuah hadits :

نِعْمَتَانِ مَغْبُوْنٌ فِيْهِمَا كَثِيْرٌ مِنَ النَّاسِ: الصِّحَةُ وَالْفَرَاغُ

Artinya : *"Ada dua nikmat yang kebanyakan orang tertipu padanya: Kesehatan dan waktu luang."* (HR. Al-Bukhari no. 6412)

Dimana hadits di atas menjelaskan dua nikmat dari Allah ﷻ yang disia-siakan oleh manusia, yaitu nikmat sehat dan waktu luang. Hal tersebut menunjukkan pentingnya memperhatikan efisiensi waktu dalam mengerjakan sesuatu. Dalam segi komputasi, peningkatan efisiensi waktu komputasi telah menjadi topik-topik penelitian terdahulu (Z.A Ahmad, 2012).

# BAB V

# PENUTUP

## 5.1 Kesimpulan

Berdasarkan uji coba dan pemabahasan pada bab sebelumnya, dapat ditarik kesimpulan bahwa implementasi komputasi paralel pada metode LAWS untuk menentukan ukuran energi tekstur tekstur citra daun cocor bebek dapat meningkatkan kecepatan proses komputasi metode LAWS dibandingkan dengan implementasi metode LAWS secara sekuens. Kesimpulan lain yang dapat ditarik adalah implementasi komputasi paralel pada metode LAWS memiliki konsumsi waktu komputasi lebih sedikit dibandingkan dengan implementasi komputasi sekuensial pada metode LAWS. Hasil perbandingan menunjukkan angka 92% peningkatan konsumsi waktu yang dihasilkan dari implementasi komputasi paralel. Kesimpulan terakhir yang dapat ditarik dari hail uji coba dan pembahasan pada bab sebelumnya yaitu konsumsi waktu komputasi citra sangat bergantung dengan besarnya ukuran citra.

## 5.2 Saran

Dalam penelitian ini masih ada beberapa kekurangan yang dapat dilengkapi pada penelitian selanjutnya. Berdasarkan review, ada beberapa kekurangan yang perlu ditambahkan pada penelitan selanjutnya, yaitu:

A. Metode yang diimplementasikan pada komputasi paralel ditambah dengan metode pengolahan citra yang lain.
B. Penelitian ini perlu diterapkan langsung pada aplikasi pengolahan citra berbasis perangkat android.

# DAFTAR PUSTAKA


Ahmad Z.A, dkk.2012. *Low Cost Parallel Processing System for Image Processing Applications.* Procedia Engineering 41 (2012) 771-776.

Acosta Alejandro, Almeida Fransisco. 2015. *Toward the optimal execution of Renderscript applications in Android devices.* Simulation Modelling Practice and Theory xxx (2015) xxx-xxx.

Lee Dong-Cheon, Schenk Toni. 1990. *Image Segmentation from Texture Measurement.* Department of Geodetic and Surveying The Ohio State University.

Kadir Abdul, Susanto Adhi. 2013. Pengolahan Citra : Teori dan Aplikasi. Penerbit Andi.

Kemp Roelof, Palmer Nicholas, Kielmann Thilo, Bal Henri, Aarts Bastiaan, Ghuloum Anwar. 2013. *Using RenderScript and RCUDA for Compute Intensive taks on Mobile Devices : a Case Study.*

Kolter Zico. 2015. *Linear Algebra Review and Reference.*

Konrad Markus. 2014. Master's Thesis. *Parallel Computing for Digital Signal Processing on Mobile Devices GPUs.* University of Apllied Sciences.

Pierce Rod. *Percentage Change.* www.mathisfun.com Diakses tanggal 11 Agustus 2016

Qureshi M.Ali, M.Deriche. 2014. *A Fast No Reference Image Quality Assessment using Laws Texture Moments.* GlobalSIP 2014.

Rafael Fransisco dkk. 2014. *Parallelization Strategy Based on Renderscript Reduction.* Research In Computer Science 78 (2014).

Sams R.Jason.2013. *Renderscript Part 2.* http://android-developers.blogspot.co.id/2011/03/renderscript.html diakses tanggal 28 Maret 2013.

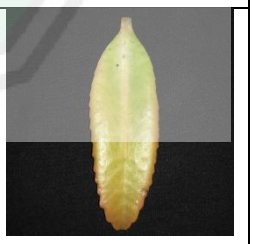
Setiawan Arden Sagiterry, Elysia, Wesley Julian, Purnama Yudi. 2015. *Mammogram Clasification using Law's Texture Energy Measure and Neural Network.* Procedia Computer Science 59 (2015) 92 – 97.

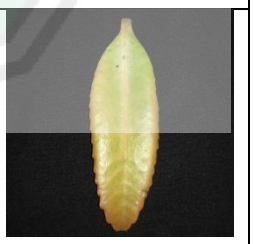
Sugiyono. 2012. Metode Penelitian Pendidikan Pendekatan Kualitatif dan R&D. Bandung: Alfabeta.

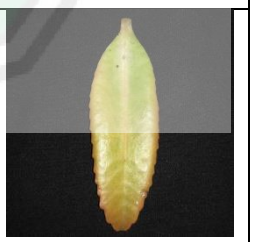
Virmani Jitendra, Kumar Vinod, Kalra Naveen, Khandelwal Niranjan . 2012. *Prediction of Cirrhosis from Liver Ultrasound B-Mode Images based on Laws' Mask Analysis.* 2012 International Conference on Image Information Processing (ICIIP 2012).

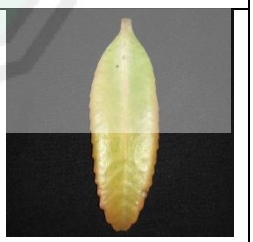
# LAMPIRAN

## Lampiran 1. Data Citra Daun

| NO | NAMA | UKURAN | | |
|---|---|---|---|---|
| | | 512x512 | 1024x1024 | 2048x2048 |
| 1 | Citra 01 |  |  |  |
| 2 | Citra 02 |  |  |  |
| 3 | Citra 03 |  |  |  |
| 4 | Citra 04 |  |  |  |
| 5 | Citra 05 |  |  |  |
| 6 | Citra 06 |  |  |  |

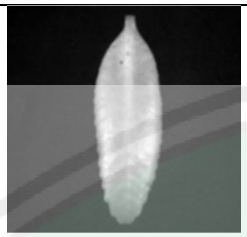
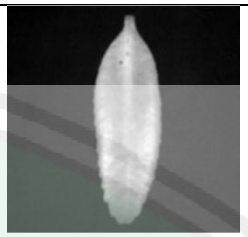
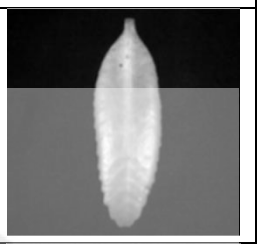
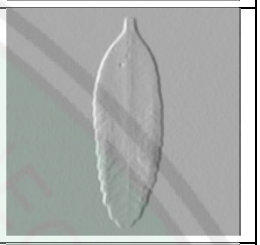
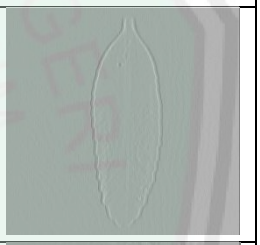
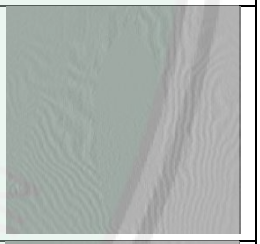
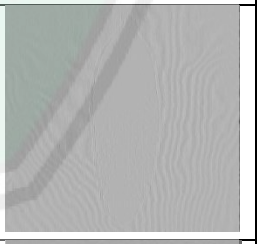
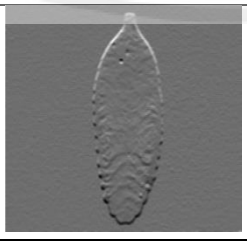
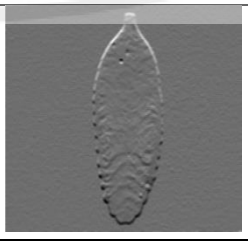
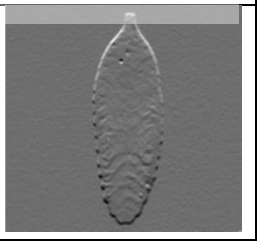
| | | | | |
|---|---|---|---|---|
| 7 | Citra 07 |  |  |  |
| 8 | Citra 08 |  |  |  |
| 9 | Citra 09 |  |  |  |
| 10 | Citra 10 |  |  |  |
| 11 | Citra 11 |  |  |  |
| 12 | Citra 12 |  |  |  |
| 13 | Citra 13 |  |  |  |

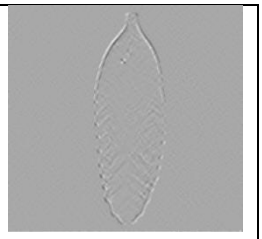
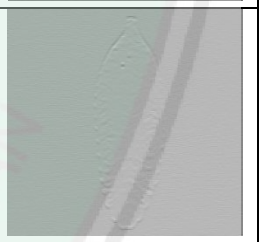
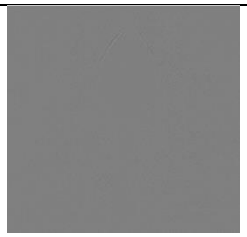
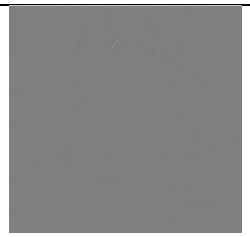
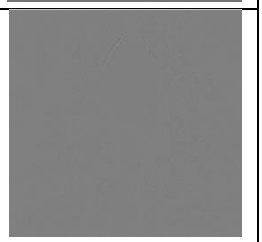
| | | | | |
|---|---|---|---|---|
| 14 | Citra 14 |  |  |  |
| 15 | Citra 15 |  |  |  |
| 16 | Citra 16 |  |  |  |
| 17 | Citra 17 |  |  |  |
| 18 | Citra 18 |  |  |  |
| 19 | Citra 19 |  |  |  |
| 20 | Citra 20 |  |  |  |

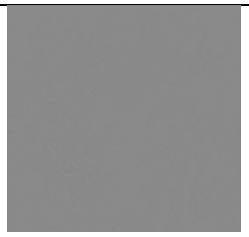
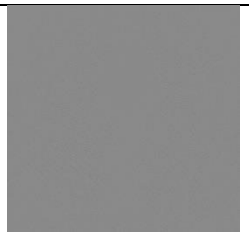
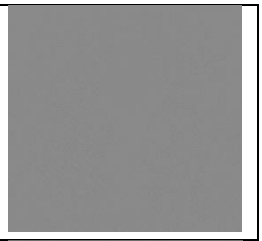
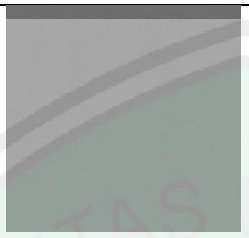
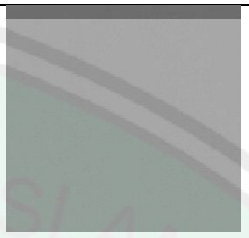
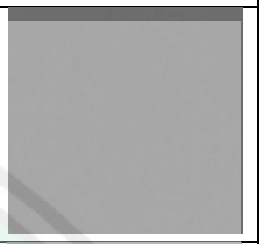
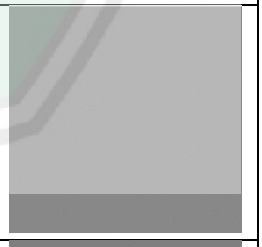
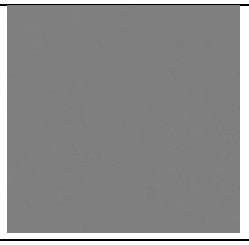
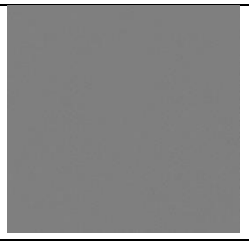
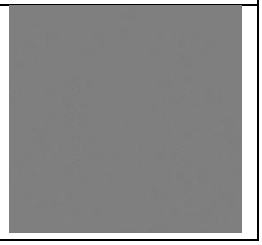
| | | | | |
|---|---|---|---|---|
| 21 | Citra 21 |  |  |  |
| 22 | Citra 22 |  |  |  |
| 23 | Citra 23 |  |  |  |
| 24 | Citra 24 |  |  |  |
| 25 | Citra 25 |  |  |  |
| 26 | Citra 26 |  |  |  |
| 27 | Citra 27 |  |  |  |

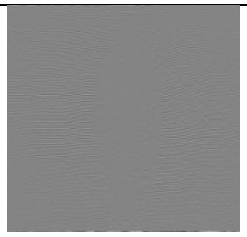
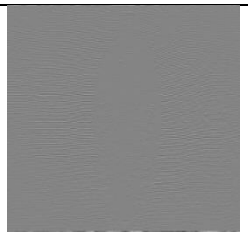
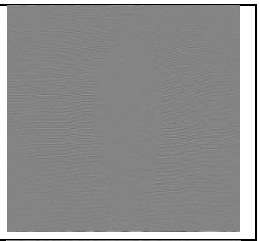
| | | | | |
|---|---|---|---|---|
| 28 | Citra 28 |  |  |  |
| 29 | Citra 29 |  |  |  |
| 30 | Citra 30 |  |  |  |

**Lampiran 2. Citra Hasil Ekstraksi Fitur**

| NO | NAMA | UKURAN | | |
|---|---|---|---|---|
| | | 512x512 | 1024x1024 | 2048x2048 |
| 1 | Citra 31 |  |  |  |
| 2 | Citra 32 |  |  |  |
| 3 | Citra 33 |  |  |  |
| 4 | Citra 34 |  |  |  |
| 5 | Citra 35 |  |  |  |
| 6 | Citra 36 |  |  |  |

| | | | | |
|---|---|---|---|---|
| 7 | Citra 37 |  |  |  |
| 8 | Citra 38 |  |  |  |
| 9 | Citra 39 |  |  |  |
| 10 | Citra 40 |  |  |  |
| 11 | Citra 41 |  |  |  |
| 12 | Citra 42 |  |  |  |
| 13 | Citra 43 |  |  |  |

| | | | | |
|---|---|---|---|---|
| 14 | Citra 44 |  |  |  |
| 15 | Citra 45 |  |  |  |
| 16 | Citra 46 |  |  |  |
| 17 | Citra 47 |  |  |  |
| 18 | Citra 48 |  |  |  |
| 19 | Citra 49 |  |  |  |
| 20 | Citra 40 |  |  |  |

| | | | | |
|---|---|---|---|---|
| 21 | Citra 41 |  |  |  |
| 22 | Citra 42 |  |  |  |
| 23 | Citra 43 |  |  |  |
| 24 | Citra 44 |  |  |  |
| 25 | Citra 45 |  |  |  |

**Lampiran 3. Kode Sumber Program**

**3.1 Kode Sumber Generate Kernel LAWS**

```
final int KERNELS [][]=
            {
                    {1,4,6,4,1},
                    {-1,2,0,2,1},
                    {-1,0,2,0,-1},
                    {1,-4,6,-4,1},
                    {-1,2,0,-2,1}
            };
    int matrikKernel [] = new int[25];
    final String [] stringKernel = {"L5", "E5", "S5",
"R5", "W5"};
public void GenerateKernel () {
        String pilihanSatu =
pilihanKernel1.getSelectedItem().toString();
        String pilihanDua =
pilihanKernel2.getSelectedItem().toString();
        int [] kernelSatu = null;
        int [] kernelDua = null;
        for (int i = 0; i < stringKernel.length; i++) {
            if (pilihanSatu == stringKernel[i]) {
                kernelSatu = KERNELS[i];
            }

            if (pilihanDua == stringKernel[i]) {
                kernelDua = KERNELS[i];
            }
        }

        for (int i = 0; i < matrikKernel.length; i++) {
            int kSatu = i / 5;
            int kDua = i % 5;
            matrikKernel[i] = kernelSatu[kSatu] *
kernelDua[kDua];
        }
    }
```

**3.2 Kode Sumber PreProcess**

```
public void Preprocess() {

   renderStart = System.nanoTime();
        mAllocIn = Allocation.createFromBitmap(rs,
image, Allocation.MipmapControl.MIPMAP_NONE,
                Allocation.USAGE_SCRIPT);
        mAllocOut = Allocation.createTyped(rs,
mAllocIn.getType());

        script.forEach_grayScale(mAllocIn, mAllocOut);
```

```
        mAllocOut.copyTo(image);

        gambar.setImageBitmap(image);
        renderStop = System.nanoTime();
        Toast.makeText(this,
                String.valueOf((renderStop-renderStart)
/ 1000000) + "ms",
                Toast.LENGTH_SHORT).show();

    }

#pragma version(1)
#pragma rs java_package_name(com.example.bb.render)

const float3 grayKonstanta = {0.299f,0.587f,0.114f};

uchar4 __attribute__((kernel)) root(const uchar4 in,
uint32_t x, uint32_t y) {
    float4 f4 = rsUnpackColor8888(in);
    uchar4 out;
    float3 mono = dot(f4.rgb, grayKonstanta);
    out = rsPackColorTo8888(mono);
    return out;
}
```

### 3.3 Kode Sumber Metode LAWS Sekuensial

```
public void sequence() {
        renderStart = System.nanoTime();
        GenerateKernel();
        int arrPikselIn [][] = new int[lebar][tinggi];
        int arrPikselOut [][] = new int
[lebar][tinggi];
        float arrPikselF[][] = new
float[lebar][tinggi];
        for (int i = 0; i < arrPikselIn.length; i++) {
            for (int j = 0; j < arrPikselIn[0].length;
j++) {
                arrPikselIn[i][j] = image.getPixel(i,j)
& 0xff;
            }
        }
        long piksel = 0;
        for (int x = 0; x < lebar; x++) {
            for (int y = 0; y < tinggi; y++) {
                int sum = 0;
                for (int kId = matrikKernel.length - 1;
kId >= 0; kId-- ) {
                    int kX = kId % 5 - 2;
                    int kY = kId / 5 - 2;
```

```
                    int x2 = x + kX;
                    int y2 = y + kY;
                    if (x2 < 0) {
                        x2 = -x2;
                    } else if (x2 > lebar - 1) {
                        x2 = (2 * (lebar - 1)) - x2;
                    }

                    if (y2 < 0) {
                        y2 = -y2;
                    } else if (y2 > tinggi - 1) {
                        y2 = (2 * (tinggi - 1)) - y2;
                    }

                    piksel = arrPikselIn[x2][y2];

                    sum += piksel * matrikKernel[kId];
                }
                //int alpha = arrPikselIn[x][y] >>> 24;
                //int konvolusi = (alpha << 24) | (sum
<< 16) | (sum << 8) | (sum);
                arrPikselOut[x][y] = sum;
                //System.out.println("piksel sequence
: "+arrPikselIn[x][y]);
                //pikselSequence[x][y] =
arrPikselIn[x][y];
            }
        }
        for (int x = 0; x < lebar; x++) {
            for (int y = 0; y < tinggi; y++) {
                int sum = 0;
                for (int wX = -7; wX <= 7; wX++) {
                    for (int wY = -7; wY <= 7; wY++) {
                        int x2 = x - wX;
                        int y2 = y - wY;
                        if (x2 < 0) {
                            x2 = -x2;
                        } else if (x2 > lebar - 1) {
                            x2 = (2 * (lebar - 1)) -
x2;
                        }

                        if (y2 < 0) {
                            y2 = -y2;
                        } else if (y2 > tinggi - 1) {
                            y2 = (2 * (tinggi - 1)) -
y2;
                        }
                        sum += arrPikselOut[x2][ y2];
```

```
                    }
                }

                arrPikselF[x][y] = sum * 0.0044f;
            }
        }

        float max = arrPikselF[0][0], min =
arrPikselF[0][0];
        for (int x = 1; x < lebar; x++) {
            for (int y = 1; y < tinggi; y++) {
                if (arrPikselF[x][y] > max) {
                    max = arrPikselF[x][y];
                } else if (arrPikselF[x][y] < min) {
                    min = arrPikselF[x][y];
                }
            }
        }

        System.out.println("max : "+max);
        System.out.println("min : "+min);
        for (int x = 0; x < lebar; x++) {
            for (int y = 0; y < tinggi; y++) {
                arrPikselIn[x][y] =
(int)(((arrPikselF[x][y] - min) / (max - min)) *255);
                int alpha = image.getPixel(x,y) >>> 24;
                int hasil = (alpha << 24) |
(arrPikselIn[x][y] << 16)
                        | (arrPikselIn[x][y] << 8) |
arrPikselIn[x][y];
                //System.out.println("hasil sequence :
"+arrPikselIn[x][y]);
                image.setPixel(x,y,hasil);
            }
        }
        gambar.setImageBitmap(image);
        renderStop = System.nanoTime();
        /**Toast.makeText(this,
                String.valueOf((renderStop-renderStart)
/ 1000000) + "ms",
                Toast.LENGTH_LONG).show();**/
        teksSekuens.setText(String.valueOf((renderStop-
renderStart) / 1000000)+"ms",null);
}
```

**2.4 Kode Sumber Metode LAWS Paralel**

```
Allocation mAllocIn;
    Allocation mAllocOut;
    ScriptC_LAWS script;
```

```
public void renderScript() {
        renderStart = System.nanoTime();
//        int [] arrPikselIn = new int [lebar *
tinggi]; //array input
        int [] arrPikselOut = new int [tinggi  *
lebar]; //array output
//        float [] arrPikselF = new float[tinggi *
lebar]; //array penampung bilangan pecah

        GenerateKernel();
        script.set_k(matrikKernel);
        script.set_lebar(lebar);
        script.set_tinggi(tinggi);

        //pembuatan elemen tipe alokasi
        Type.Builder builder = new
Type.Builder(rs,Element.I32(rs));
        builder.setX(lebar);
        builder.setY(tinggi);
        Type tipe = builder.create();

        mAllocIn = Allocation.createFromBitmap(rs,
image, Allocation.MipmapControl.MIPMAP_NONE,
Allocation.USAGE_GRAPHICS_TEXTURE);
        //pengisian nilai piksel dari array input ke
alokasi input
  //      mAllocIn = Allocation.createTyped(rs, tipe,
Allocation.USAGE_SCRIPT);
//        mAllocIn.copy2DRangeFrom(0, 0,lebar, tinggi,
arrPikselIn);
        //menyiapkan alokasi output sebesar alokasi
input
        mAllocOut = Allocation.createTyped(rs, tipe,
Allocation.USAGE_GRAPHICS_TEXTURE);
        script.set_in(mAllocIn); //binding alokasi
input ke dalam renderscript
        script.forEach_konvolusi(mAllocOut);
//pemanggilan kernel konvolusi
        //mAllocOut.syncAll(Allocation.USAGE_SCRIPT);
        mAllocOut.copyTo(arrPikselOut); //pengisian
nilai array output dari alokasi output

        //pembuatan alokasi memori bertipe float
        //copy array output ke alokasi input
        mAllocIn = Allocation.createTyped(rs, tipe,
Allocation.USAGE_GRAPHICS_TEXTURE);
        mAllocIn.copy2DRangeFrom(0, 0, lebar, tinggi,
arrPikselOut);
        script.set_in(mAllocIn);
```

```
        script.forEach_geserWindow(mAllocOut);
//pemanggilan kernel window
        //mAllocOut.syncAll(Allocation.USAGE_SCRIPT);
        mAllocOut.copyTo(arrPikselOut);

        float max = arrPikselOut[0], min =
arrPikselOut[0];
        for (int x = 1; x < arrPikselOut.length; x++) {
            if (arrPikselOut[x] > max) {
                max = arrPikselOut[x];
            } else if (arrPikselOut[x] < min) {
                min = arrPikselOut[x];
            }
        }

        System.out.println("max render : " + max);
        System.out.println("min render : " + min);
        script.set_mins(min);
        script.set_maks(max);
        builder = new Type.Builder(rs,
Element.RGBA_8888(rs));
        builder.setX(lebar);
        builder.setY(tinggi);
        tipe  = builder.create();
        mAllocOut = Allocation.createTyped(rs,tipe,
Allocation.USAGE_GRAPHICS_TEXTURE );

        mAllocIn.copy2DRangeFrom(0, 0, lebar, tinggi,
arrPikselOut);
        script.set_in(mAllocIn);
        script.forEach_penormalan(mAllocOut);
        mAllocOut.copyTo(image);
        gambar.setImageBitmap(image);
        renderStop = System.nanoTime();
        teksParalel.setText(String.valueOf((renderStop-
renderStart) / 1000000+"ms"), null);
    }
#pragma version(1)
#pragma rs java_package_name(cpm.example.bb.render)

const float3 grayKonstanta = {0.299f,0.587f,0.114f};
int k [] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float mins = 0, maks = 0;
int lebar = 0, tinggi = 0;
rs_allocation in;

int32_t __attribute__((kernel)) konvolusi(uint32_t x,
uint32_t y){
```

```
    int32_t sum = 0, piksel = 0;
    int4 temp = {0,0,0,0};

    for (int kIdx = 24; kIdx >= 0; kIdx--) {
        int kX = kIdx % 5 - 2;
        int kY = kIdx / 5 - 2;
        int x2 = x + kX;
        int y2 = y + kY;
        if (x2 < 0) {
           x2 = -x2;
        } else if (x2 > lebar - 1) {
           x2 = (2 * (lebar - 1)) - x2;
        }

        if (y2 < 0) {
            y2 = -y2;
        } else if (y2 > tinggi - 1) {
            y2 = (2 * (tinggi - 1)) - y2;
        }
        temp = convert_int4(rsGetElementAt_uchar4(in,
x2, y2));
        piksel = temp.r;
        sum += piksel * k[kIdx];
        //sum += rsGetElementAt_int(in, x2, y2) *
k[kIdx];
    }
    return sum;
}
int32_t __attribute__((kernel)) geserWindow(uint32_t x,
uint32_t y){
    int32_t sum = 0, temp = 0;
    int32_t hasil = 0;

    for (int32_t wX = -7; wX <= 7; wX++) {
        for (int32_t wY = -7; wY <= 7; wY++) {
            int x2 = x + wX;
            int y2 = y + wY;
            if (x2 < 0) {
                x2 = -x2;
            } else if (x2 > lebar - 1) {
                x2 = (2 * (lebar - 1)) - x2;
            }

            if (y2 < 0) {
                y2 = -y2;
            } else if (y2 > tinggi - 1) {
                y2 = (2 * (tinggi - 1)) - y2;
            }
```

```
                temp = rsGetElementAt_int(in, x2, y2);
                sum += temp;
            }
        }

        hasil = sum *  0.0044f;

        return hasil;
}

uchar4 __attribute__((kernel)) penormalan(uint32_t x,
uint32_t y) {
        int32_t hasil = 0;
        //float4 piksel = {0,0,0,0};
        hasil = ((rsGetElementAt_int(in, x, y) - mins) /
(maks - mins)) * 255;
        //piksel.rgb = hasil;
        //piksel.a = 255;
        uchar4 out = (255 << 24) | (hasil << 16) | (hasil
<< 8) | hasil;

        return out;
}

uchar4 __attribute__((kernel)) grayScale(const uchar4
in, uint32_t x, uint32_t y) {
        float4 f4 = rsUnpackColor8888(in);
        uchar4 out;
        float3 mono = dot(f4.rgb, grayKonstanta);
        out = rsPackColorTo8888(mono);
        return out;
}
```