IMPLEMENTASI METODE GAUSSIAN FILTER UNTUK PENGHAPUSAN NOISE PADA CITRA MENGGUNAKAN GPU

SKRIPSI

Oleh :
ZULIATUL AFIFA
NIM. 12650081

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIMMALANG
2016

LEMBAR PENGAJUAN

IMPLEMENTASI METODE GAUSSIAN FILTER UNTUK PENGHAPUSAN NOISE PADA CITRA MENGGUNAKAN GPU

SKRIPSI

Diajukan Kepada:
Fakultas Sains dan Teknologi
Universitas Islam Negeri
Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh :
ZULIATUL AFIFA
NIM. 12650081

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

LEMBAR PERSETUJUAN

IMPLEMENTASI METODE GAUSSIAN FILTER UNTUK PENGHAPUSAN NOISE PADA CITRA MENGGUNAKAN GPU

SKRIPSI

Oleh :

ZULIATUL AFIFA
NIM. 12650081

Telah Diperiksa dan Disetujui untuk Diuji
Tanggal Desember 2016

Pembimbing I,	Pembimbing II,
Fatchurrochman, M. Kom	M. Imamudin, Lc. MA
NIP. 19700731 200501 1 002	NIP. 197406022009011010

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

Dr. Cahyo Crysdian
NIP. 197404242009011008

LEMBAR PENGESAHAN

IMPLEMENTASI METODE GAUSSIAN FILTER
UNTUK PENGHAPUSAN NOISE PADA CITRA
MENGGUNAKAN GPU

SKRIPSI

Oleh :

ZULIATUL AFIFA
NIM. 12650081

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

Penguji Utama	Dr. M.Amin Hariyadi, M.T NIP. 19670118 200501 1 001	.....
Ketua Penguji	Dr. Cahyo Crysdian NIP. 19740424 200901 1 008	.....
Sekretaris Penguji	Fatchurrochman, M. Kom NIP. 19700731 200501 1 002	.....
Anggota Penguji	M. Imamudin, Lc. MA NIP. 197406022009011010	.....

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

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

HALAMAN PERNYATAAN
ORISINALITAS PENELITIAN

Saya yang bertanda tangan dibawah ini:

Nama : Zuliatul Afifa

NIM : 12650081

Jurusan : Teknik Informatika

Fakultas : Sains dan Teknologi

Judul Skripsi : Implementasi Metode Gaussian Filter Untuk Penghapusan Noise pada Citra Menggunakan GPU

Menyatakan dengan sebenarnya bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan, atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri., kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka. Apabila dikemudian hari terbukti atau dapat dibuktikan skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, Desember 2016

Yang membuat pernyataan,

Zuliatul Afifa

12650081

MOTTO

Pergunakanlah waktumu sebaik mungkin karena hidup itu pendek

dan cepat berlalu dan terus berlalu begitu cepat.

HALAMAN PERSEMBAHAN

Alhamdulillah puji syukur kehadirat Allah SWT yang memberikan kekuatan dan kemudahan kepada saya hingga bisa sampai menyelesaikan kuliah S1 di kampus UIN malang. Sholawat serta salam kepada Nabi Muhammad SAW yang membawa kabar baik kepada seluruh umat manusia.

Terima kasih kepada kedua orang tua dan keluarga saya yang selalu mendidik dan menyayangi saya dan tak lupa untuk mendo'akan saya agar menjadi manusia yang berguna bagi dunia utamanya untuk agama.

Terima kasih kepada Dosen-dosen yang telah sabar dan ikhlas dalam mendidik saya hingga mampu melewati seluruh ujian dari semua mata kuliah yang saya tempuh, terutama kepada Bapak Zainal Abidin, M.Kom, Fatchurrochman, M. Kom dan M. Imamuddin, Lc. MA yang telah mengamalkan ilmunya dan selalu memotivasi saya.

Terima kasih kepada seluruh teman-teman saya yang telah menemani saya selama kuliah, mendukung saya, membantu saya, men-*support*saya setiap saat. Khususnya ke-3 Sahabat saya tamy, kiki & wasa yang selalu ada dalam semua ceritaku selama kuliah. Dan juga terima kasih kepada adik kosku ulfah yang sudah rela meinjamkan laptopnya untuk menyelesaikan skripsi ini.

KATA PENGANTAR

Segala puji bagi Allah SWT yang maha pengasih dan penyayang, bahwa atas taufiq dan hidayah-Nya maka penulis dapat menyelesaikan penyusunan skripsi ini.

Shalawat dan salam semoga tetap tercurahkan kepada junjungan kita Nabi Muhammad saw kekasih Allah sang pemberi syafa'at beserta seluruh keluarga, sahabat dan para pengikutnya.

Skripsi yang berjudul "**Implementasi Metode *Gaussian Filter* untuk Penghapusan *Noise* pada Citra Menggunakan GPU",** ini disusun untuk memenuhi salah satu syarat guna memperoleh gelar Sarjana Strata Satu (S.1) Fakultas Sain dan Teknologi Universitas Maulana Malik Ibrahim Malang. Dalam penyusunan skripsi ini penulis menyadari bahwa penulisan ini tidak mungkin terlaksana tanpa adanya bantuan baik moral maupun spiritual dari berbagai pihak. Maka patut rasanya penulis menyampaikan terimakasih yang sedalamnya terutama kepada :

1. Fatchurrochman ,M.Kom, selaku dosen pembimbing I dan Zainal Abidin M,Kom selaku dosen pembimbing penelitian dan Bapak M. Imamuddin, Lc. MA selaku dosen pembimbing II, yang telah banyak memberikan bimbingan dan pengarahan selama proses pembuatan skripsi.
2. Seluruh jajaran dosen Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang, atas segala ilmu yang telah diberikan selama ini.

3. Seluruh Civitas Akademika Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.
4. Teman-teman Jurusan Teknik Informatika angkatan 2012, terim kasih atas persahabatan yang telah terjalin selama ini.
5. Serta seluruh pihak yang secara langsung maupun tidak langsung membantu proses penyusunan skripsi ini yang tidak bisa penulis sebutkan satu persatu. Semoga Allah SWT membalas segala kebaikan dan bantuan yang telah diberikan.

Penulis menyadari bahwa skripsi ini masih banyak kekurangan dan jauh dari sempurna. Oleh karena itu saran dan kritik yang bersifat membangun sangat penulis harapkan untuk perbaikan ke depan. Semoga skripsi ini dapat bermanfaat bagi pembaca dan dapat menambah pengetahuan kita semua, Amin.

Malang, Desember 2016

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LISTING KODE

ABSTRAK

Afifa, Zuliatul. 2016. **Implementasi Metode Gaussian Filter untuk Penghapusan Noise pada Citra Menggunakan GPU**. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: Fatchurrochman, M.Kom.

Kata Kunci: GPU, OpenCL, *noise*, pengolahan citra.

Citra yang kita miliki terkadang mempunyai kualitas mutu yang kurang bagus atau tidak memuaskan, misalnya mengandung *noise* atau derau. Sehingga harus dilakukan penghapusan *noise* untuk memperbaiki kualitas citra. Penghapusan *noise* pada citra (*image filtering*) adalah salah satu bagian terpenting dalam pengolahan citra. Selama ini komputasi dilakukan pada CPU yaitu komputasi tunggal atau sekuensial. Pada komputasi tunggal, instruksi memproses data secara bergantian. Sehingga membutuhkan waktu yang lama untuk menyelesaikan data yang banyak. Sedangkan komputasi pararel dapat mengerjakan beberapa data dalam waktu yang sama. Komputasi pararel menggunakan banyak *processor*untuk mengolah data. Setiap *processor* mengolah data yang berbeda. Pada penelitian sebelumnya juga dikatakan bahwa perbandingan kinerja antara implementasi berbasis GPU lebih cepat dari implementasi berbasis CPU. Penelitian ini menyajikan hasil dari perbandingan kinerja antara implementasi metode *gaussian filter* untuk penghapusan *noise* pada citra menggunakan GPU dan CPU. *Framework* yang digunakan untuk komputasi pararel dalam penelitian ini adalah OpenCL. Ujicoba pada penelitian ini dilakukan pada 30 citra yang telah diberi *noise salt and pepper* dengan intensitas *noise* sebesar 5%, 10%, dan 50% pada setiap citra. Berdasarkan hasil pengujian, hasil ujicoba data dengan intensitas *noise* 5% yang dijalankan pada CPU-GPU mengalami kenaikan kecepatan waktu sebesar 86% lebih cepat untuk perbesaran windowing 3×3, 76% perbesaran windowing 5×5, 66% perbesaran windowing 7×7. Begitu juga untuk intensitas *noise* 10% dan 50%, rata-rata kecepatan waktu yang dibutuhkan untuk proses penghapusan *noise* pada GPU lebih cepat dari CPU.

ABSTRACT

Afifa, Zuliatul. 2016. **Implementation Methods Gaussian Noise Removal Filter for images on the Smartphone Using GPU**. Department of Informatics Faculty of Science and Technology of the State Islamic University of Maulana Malik Ibrahim Malang. Supervisor: Fatchurrochman, M.Kom.

Keywords: GPU, OpenCL, noise, image filtering.

The image that we have sometimes have a bad quality and unsatisfying, for example, it contains noise.So the noise removal must be done to improve the image quality.The noise removal (image filtering) is an important part in image processing. Nowadays, the computation process has been done on the CPU through single or sequential computing. In a single computing, the instruction presesses the data replaceable. So it takes a long time to resolve a lot of data.On the other hand, parallel computing can compute several data in the same time. Parallel computing using many processors to compute a large data. In the previous research has also been stated that the performance comparison of a GPU-based implementations faster than CPU-based implementations. This study presents the results of a performance comparison between gaussian filter implementation methods for noise removal in the image using GPU and CPU. In this study, the framework used for parallel computing is OpenCL. Experiments in this study was performed on 30 images that have been given a salt and pepper noise with noise intensity by 5%, 10%, and 50% on each image. Based on the test results, test results data with the noise intensity 5% run on the CPU-GPU increases time speed by 86% faster for windowing 3 × 3, 76%windowing 5 × 5, 66% windowing 7 × 7. Likewise, for noise intensity of 10% and 50%, the average speed of the time required for noise removal process on the GPU is faster than the CPU.

الملخص

عفيفة، زلية. ٢٠١٦. **تنفيذ أسلوب "تصفية الضبابي" "القضاء على الضوضاء" في الصورة استخدام الحوسبة المتوازية**. قسم المعلوماتية كلية العلوم والتكنولوجية في جامعة مولانا مالك إبراهيم مالانج الإسلامية الحكومية بمالانج.المشرف: فتح الرحمن الماجستير

الكلمة الأساسية: الحوسبة المتوازية،OpenCL، والضوضاء، ومعالجة الصور.

تحتوي الصورة في بعض الأحيان ذات نوعية جيدة أقل أو غير مرضية، على الضجيج أو الضوضاء. حتنقوم للقيام بالقضاء على الضوضاء بتحسين جودة الصورة. والقضاء على الضوضاء في الصورة (تصفية الصورة) واحد من أهم جزء في معالجة الصور. ويتم خلال هذه العملية الحسابية على وحدة المعالجة المركزية التي هي تقوم بمفرده أو متسلسلة الحوسبة. بناء على تعليمات الحوسبة الطويلة ولمعالجة البيانات التبادل. لذا يستغرق وقتاً طويلاً لإكمال البيانات. وفي حين الحوسبة المتوازية يمكن العمل بها بعض البيانات في نفس الوقت. وبموازاة الحوسبة باستخدام المعالج الكثير من البيانات العملية. كل معالج لمعالجة البيانات المختلفة. في البحوث السابقة بيّن أن مقارنة بين أداء تنفيذ المستندة إلىGPUأسرع من تنفيذ المستندة إلى وحدة المعالجة المركزية (CPU). تقدم هذه الدراسة نتائج المقارنة بين أداء أسلوب تنفيذ تصفية الضبابي للقضاء على الضوضاء في الصورة باستخدام الحوسبة المتوازية والحوسبة متسلسلة. الإطار المستخدم للحوسبة المتوازية في هذا البحث هو .OpenCLالمحاكمة الحرة في هذا البحث على٣٠صورة أعطيت الضوضاء كثافة الضجيج *noise salt & pepper*٥%،١٠%، و ٥٠% على كل صورة. وبناء على نتائج الاختبار، نتائج بيانات الاختبار مع شدة الضوضاء ٥% تعمل على زيادة سرعة الجرافيك وحدة المعالجة المركزية من٨٦% في المائة أسرع بالنسبة ٧٦% التكبير عمل إطارات عمل إطارات التكبير ٣×٣، ٥×٥، ٦٦% التكبير عمل إطارات ٧×٧. وبالمثل لشدة الضوضاء ١٠% و ٥٠%، سرعة المتوسط الوقت الذي تستغرقه عملية إزالة الضوضاء في الجرافيك أسرع من وحدة المعالجة المركزية(CPU).

BAB I

PENDAHULUAN

4.1. Latar Belakang

Citra(*image*) merupakan salah satu bentuk informasi yang memegang peranan sangat penting. Citra bisa berwujud gambar dua dimensi, seperti lukisan, foto, dan berwujud tiga dimensi, seperti patung.Citra terbagi menjadi 2 yaitu ada citra yang bersifat analog dan ada citra yang bersifat digital. Citra analog tidak dapat dipresentasikan dalam komputer sehingga tidak bisa diproses di komputer secara langsung. Sehingga masyarakat mulai banyak yang meninggalkan citra analog dan beralih pada citra digital, karena citra digital dapat di olah komputer. Citra digital sangat mudah dipergunakan baik dari segi pengiriman sebagai data, pengolahan maupun pemrosesan citra itu sendiri.

Citra yang kita miliki terkadang mempunyai kualitas mutu yang kurang bagus atau tidak memuaskan, misalnya mengandung noise atau derau. Hal itu biasanya terjadi pada proses pengambilan ataupun penyimpanan citra digital serta proses pengiriman citra digital baik melalui satelit maupun melalui kabel. Menurut Lipeng Wang(2011), jenis Noise yang paling sering merusak citra adalah s*alt and pepper noise*(atau *noise impluse)* dan *gaussian noise*.*Salt and pepper noise* disebabkan oleh gangguan yang tiba-tiba dan tajam pada proses perolehan isyarat citra. Bentuknya berupa bintik-bintik hitam atau putih dalam citra.Pada noise jenis ini *pixel*mengalami gangguan yang memiliki *pixel* intensitas tinggi bernilai 255 atau intensitas rendah bernilai 0. Pada nilai 0 diibaratkan sebagai *pepper* (lada) yang ditunjukkan dengan titik hitam dan bernilai 255 sebagai *salt* (garam) yang ditunjukkan dengan titik putih. Sedangkan*gaussian*

noise disebabkan oleh sumber-sumber alam seperti getaran termal atom dan sifat diskrit dari radiasi benda hangat.*Gaussian noise* umumnya mengganggu nilai abu-abu dalam citra digital.

Untuk menghilangkan *noise* tersebut, ada beberapa metode yang dapat dilakukan agar kualitas citra lebih baik, di antaranya yaitu *mean filter, median filter, gaussian filter, alpha trimmed mean filter, adaptive median filter, minimum mean square error (wiener) filter, order-statistics filters, max and min filters,* dll. Perbaikan citra disebut juga dengan *image restoration.Image restoration* digunakan untuk menghilangkan atau meminimumkan noise.*Image restoration* di proses menggunakan kernel dan non-kernel. Kernel merupakan matrik berukuran kecil yang berisi angka-angka. Dalam pengolahan citra kernel hanyalah sebuah matriks 2 dimensi angka yang digunakan untuk proses konvolusi. Ukuran kernel dapat berbeda-beda, seperti 3 x 3, 5 x 5, dan sebagainya.Filter yang termasuk menggunakan kernel yaitu *Gaussian Filter* dan *Mean Filter*. Sedangkan filter yang tidak menggunakan kernel(Non-Kernel) yaitu *Median Filter* dan *Wiener Filter*.

Gaussian Filter atau filter gaussian blur merupakan salah satu teknik pengolahan citra yang mendasar dan banyak digunakan. *Gaussian Filter* di dapat dari operasi konvolusi. Operasi konvolusi yang dilakukan adalah perkalian antara matriks kernel dengan matriks gambar asli. Pada pelaksanaan konvolusi, kernel digeser sepanjang baris dan kolom dalam citra sehingga diperoleh nilai citra yang baru. Tidak seperti metode filtering lainnya, seperti *Wiener,* yang membutuhkan pengetahuan signal yang lengkap untuk menentukan filter yang optimal. Pada

skripsi ini akan dilakukan perbaikan citra yang sudah dikenai noise dengan menggunakan *gaussian filter*.

Komputasi pada konvolusi dapat menjadi lama jika ukuran citrabesar. Untuk kernel dengan ukuran n x n, proses konvolusi akan dilakukan n x n kali. Kalau dinyatakan dengan ukuran Big O, prosesnya memerlukan $O(n^2)$. Untuk mempercepat komputasi, perlu dicari solusi yang proses komputasinya kurang dari $O(n^2)$. Hal ini dapat dilakukan dengan memecah kernel yang berupa matriks menjadi dua buah vektor. Jika konvolusi dilakukan dengan menggunakan komputasi sekuensial, maka akan membutuhkan waktu yang lama untuk memproses konvolusi dengan ukuran citra yang sangat besar.

Komputasi pararel dapat mengeksekusi perintah secara bersamaan namun dengan data yang berbeda. Komputasi pararel menggunakan beberapa prosesor (*multiprocessor* atau arsitektur komputer dengan banyak prosesor) agar kinerja komputer semakin cepat. Komputasi pararel lebih efektif dan dapat menghemat waktu untuk pemrosesan data yang besar daripada komputasi tunggal atau sekuensial.

Hasil Penelitian menunjukkan bahwa menggunakan komputasi pararel memperoleh kinerja terbaik(Sanchez, Maria G, *et all.* 2014). Kecepatan komputasi dapat memungkinkan efisiensi filter citra. Dalam penelitian lain, implementasi menggunakan *Graphics Processing Unit*(GPU) terbukti lebih efisien dibandingkan dengan *Central Processing Unit*(CPU)(Ravibabu, P, *et all*. 2014). Dengan menggunakan algoritma yang sama dan dijalankan pada GPU 48 core menunjukkan bahwa kecepatannya lebih baik(0,58ms untuk resolusi *Hight*

Definition(HD)). Sedangkan pada CPU 1 *core* membutuhkan waktu 4ms untuk resolusi tersebut.

Perbandingan kinerja antara implementasi berbasis CPU dan implementasi berbasis GPU juga telah dilakukan oleh Chang Won Lee (2014).*Multi-threaded two-way* rekursif filter gaussian diimplementasikan pada PC dengan Intel Quad Core i5-750 Prosesor 2,67 GHz. Implementasi berbasis GPU mengkonfigurasi 32 baris per block. Implementasi berbasis CPU memiliki kinerja yang lebih baik pada gambar berukuran *Standard Definition*(SD), namun terbalik ketika digunakan pada gambar berukuran HD dan *Full Hight Definition*(Full HD).

Platform pemrograman pararel yang paling sering digunakan adalah CUDA (*compute unified device architecture*) yang dikembangkan oleh NVIDIA. CUDA hadir pertama kali sebagai paltform yang tidak hanya menangani tentang grafika komputer tetapi juga untuk segala jenis aplikasi. Pada tahun 2008 OpenCL (*open computing language*) dihadirkan oleh Khronos™ yang kemudian diadopsi untuk dipasangkan pada produk perusahaan AMD. Dengan berjalannya waktu dikembangkan device untuk pemrograman pararel yang saling berintegrasi tidak hanya berjalan pada GPU, namun juga CPU. AMD Radeon™ menghadirkan CPU dengan GPU terintergrasi dalam satu paket yang sempurna.Teknologi baru ini memberikan kinerja dan efisiensi dengan HSA (*Heterogeneous System Architecture*) yang memungkinkan CPU dan GPU untuk bekerja bersama pada aplikasi yang didukung dengan membagi dan mengarahkan tugas secara cepat dan otomatis ke *core* yang tepat. Dengan AMD dapat dilakukan komputasi paralel menggunakan bahasa pemrograman OpenCL. OpenCL merupakan *framework* untuk memenuhi tujuan dari semua pemrograman paralel dan *portable* dengan

GPU maupun CPU(*Kazuya Matsumoto et al*,2012). OpenCL dirancang untuk memenuhi hal tersebut. Dikembangkan oleh Khronos,OpenCL merupakan gabungan konsep CUDA, CAL,CTM, dan mendukung berbagai tingkat paralelisme dan efisien untuk homogen atau heterogen(Gaster, Benedict R, *et all* .2013).OpenCL merupakan teknologi yang akan berjalan pada hampir semua platform.Usaha dalam peningkatan kecepatan ini dapat diterapkan untuk berbagai aplikasi seperti audio,pengenalan suara,video,database gambar,pengolahan citra ,dan masih banyak lainnya *(J. A. Fraire et al*,2013). Luasnya lingkup penerapan OpenCL peneliti akan coba terapkan pada citra untuk penghapusan noise.

Pada skripsi ini akan diterapkan algoritma*gaussian filter* untuk menghilangkan *noise* menggunakan OpenCL pada CPU-GPU, sehingga diharapkan memperoleh hasil yang memuaskan dengan efisiensi waktu yang tinggi. Dengan menggunakan komputasi pararel, proses komputasi akan lebih cepat dan tidak membutuhkan waktu yang lama. Sehingga dapat mengolah citra dengan ukuran gambar yang sangat besar.

4.2. Rumusan Masalah

1. Seberapaefisien waktu yang dibutuhkan untuk proses penghapusan*noise* menggunakanGPU dan CPU?
2. Seberapa akurat metode *gaussian filter* untuk menghilangkan *noise* menggunakan GPU dan CPU?

4.3. Batasan Masalah

Agar penelitian ini tidak keluar dari pokok permasalahan yang dirumuskan, maka ruang lingkup pembahasan diberi batasan sebagai berikut:

1. Ukuran gambar yang digunakan adalah 3264× 1836.
2. Smartphone yang digunakan adalah ASUS Zenfone5 (kamera sebesar 8 MP).
3. Bahasa pemrograman yang digunakan adalah bahasa pemrograman OpenCL.
4. Jenis *noise* yang digunakan adalah *salt and pepper*.
5. Spesifikasi laptop yang digunakan;

   Platform : Laptop Toshiba Satellite C855D

   Memori : 4096MB RAM

   Harddisk : 320GB HDD

   Processor : AMD A6-4400M APU with Radeon(tm) HD Graphics (2CPUs), ~2.7GHz

4.4. Tujuan Penelitian

1. Mengukur efisiensi waktu yang dibutuhkan untuk penghapusan*noise* pada GPU dan CPU.
2. Mengukur akurasi metode *gaussian filter* untuk menghilangkan *noise* pada GPU dan CPU.

4.5. Manfaat Penelitian

Manfaat yang diharapkan dari penulisan skripsi ini adalah untuk memperkaya pengetahuan tentang penerapan metode *gaussian filter* untuk menghapus *gaussian noise* menggunakan GPU. Selain itu juga dapat mengetahui perbedaan kecepatan waktu komputasi menggunakan GPU dan CPU. Dan juga mengetahuiperbedaan akurasi citra yang dihasilkan menggunakan GPU dan CPU.

BAB II

LANDASAN TEORI

2.1 Pengolahan Citra Digital

Citra adalah gambar dua dimensi yang dihasilkan dari gambar analog dua dimensi yang kontinu menjadi gambar diskrit melalui proses sampling. Gambar analog dibagi menjadi N baris dan M kolom sehingga menjadi gambar diskrit. Dimana setiap pasangan indeks baris dan kolom menyatakan suatu titik pada citra. Nilai matriksnya menyatakan nilai kecerahan titik tersebut.Titik-titik tersebut dinamakan sebagai elemen citra, atau *pixel* (*picture elemen*). Dalam kamus komputer gambar atau foto diistilahkan sebagai citra digital sebagai fungsi representasi matematis berua matriks $Cm \times n = (cij)$.

Menurut Efford (2000), pengolahan citra adalah istilah umum untuk berbagai teknik yang keberadaannya untuk memanipulasi dan memodifikasi citra dengan berbagai cara. Foto adalah contoh gambar berdimensi dua yang bisa diolah dengan mudah. Setiap foto dalam bentuk citra digital dapat diolah melalui perangkat-lunak tertentu. Sebagai contoh, apabila hasil bidikan kamera terlihat agak gelap, citra dapat diolah agar menjadi lebih terang. Dimungkinkan pula untuk memisahkan foto orang dari latar belakangnya. Gambaran tersebut menunjukkan hal sederhana yang dapat dilakukan melalui pengolahan citra digital. Tentu saja, banyak hal lain yang lebih pelik yang dapat dilakukan melaluui pengolahan citra digital.

Menurut R. C Gonzalez dan Woods(2002), gambar dapat didefinisikan sebagai fungsi dua dimensi, f(x,y), dimana x dan y adalah spasial koordinat, dan amplitudo f pada setiap sepasang koordinat (x,y) disebut intensitas atau gray level

dari citra pada suatu titik. Ketika x, y, dan nilai-nilai amplitudo f semua terbatas jumlah diskrit, kita sebut gambar itu adalah gambar digital. Sedangkan Zhou, Huiyu dkk dalam bukunya *Digital Image Processing: Part I*, mengatakan bahwa pengolahan citra digital adalah teknologi menerapkan sejumlah algoritma komputer untuk memproses gambar digital. Hasil dari proses ini dapat berupa gambar atau satu set karakteristik perwakilan atau properti dari gambar asli.

2.2 Filter

Filter adalah alat untuk memproses data yang mempunyai data asli untuk memproduksi data hasil sebagaimana yang diinginkan. Dalam pengolahan citra, respon perambatan filter memberikan gambaran bagaimana *pixel-pixel* pada citra diproses.

2.2.1 Konvolusi

Konvolusi adalah operasi yang mendasar dalam pengolahan citra. Konvolusi didefinisikan secara sederhana sebagai operasi penjumlahan dari perkalian dengan notasi operasi (*), yang mengalikan sebbuah citra dengan sebuah *mask* atau *kernel*. Konvolusi 2 buah fungsi *f*(x) dan *g*(x) didefinisikan sebagai berikut:

$$h(x) = f(x) \times g(x) = \sum_{x=-n}^{n} \sum_{y=-N}^{N} f(u,v)g(x+u, y+u) \quad ...(2.1)$$

dimana: f(x,y) : Citra asli
h(x,y) : *Linier-position invariant operator*
g(x,y) : Citra hasil konvolusi
x, y, u dan v : Posisi titik dalam citra

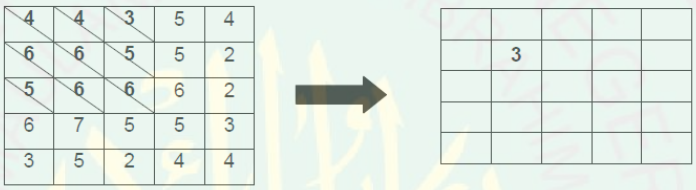
Misalkan citra *f*(x,y) yang berukuran 5 × 5 dan sebuah kernel atau mask ukuran 3 × 3 masing-masing prosesnya adalah sebagai berikut:

$$f(x,y) = \begin{bmatrix} 4 & 4 & 3 & 5 & 4 \\ 6 & 6 & 5 & 5 & 2 \\ 5 & 6 & 6 & 6 & 2 \\ 6 & 7 & 5 & 5 & 3 \\ 3 & 5 & 2 & 4 & 4 \end{bmatrix} \qquad g(x,y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Posisi (0,0) dari kernel

Berikut operasi konvolusi antara citra $f(x,y)$ dengan kernel $g(x,y)$, dapat diilustrasikan sebagai berikut:

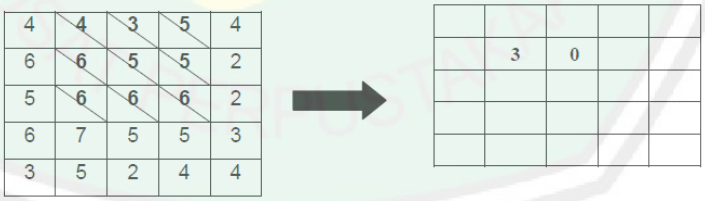
(1) Tempatkan kernel pad sudut kiri atas, kemudian hitung nilai *pixel* pada posisi(0,0) dari kernel, berikut prosesnya:



Hasil konvolusi = 3. Nilai ini dihitung dengan cara berikut:

(0×4)+(-1×4)+(0×3)+(-1×6)+(4×6)+(-1×5)+(0×5)+(-1×6)+(0×6)=3

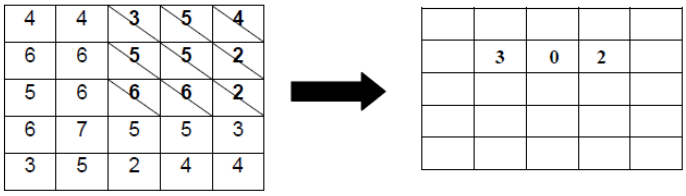
(2) Geser kernel satu *pixel* ke kanan, kemudian hitung nilai *pixel* pada posisi(0,0) dari kernel.



Hasil konvolusi = 0. Nilai ini dihitung dengan cara berikut:

(0×4)+(-1×3)+(0×5)+(-1×6)+(4×5)+(-1×5)+(0×6)+(-1×6)+(0×6)=0

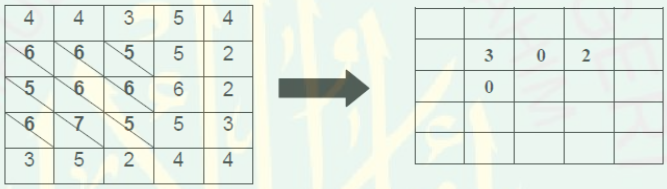
(3) Geser kernel satu *pixel* ke kanan, kemudian hitung nilai *pixel* pada posisi (0,0) dari kernel.



Hasil konvolusi = 2. Nilai ini dihitung dengan cara berikut:

(0×3)+(-1×5)+(0×4)+(-1×5)+(4×5)+(-1×2)+(0×6)+(-1×6)+(0×2)=2

(4) Selanjutnya, geser kernel satu *pixel* ke bawah, lalu mulai lagi melakukan konvolusi dari sisi kiri citra. Setiap kali konvolusi, geser kernel satu *pixel* ke kanan.



Hasil konvolusi = 0. Nilai ini dihitung dengan cara berikut:

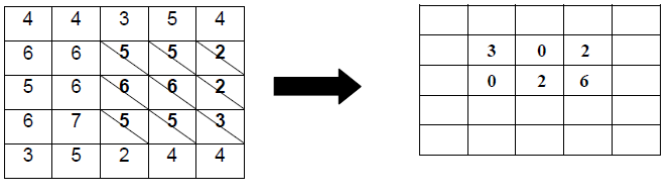
(0×6)+(-1×6)+(0×5)+(-1×5)+(4×6)+(-1×6)+(0×6)+(-1×7)+(0×5)=0

(5) Proses berikut sama dengan sebelumnya:

Hasil konvolusi = 2. Nilai ini dihitung dengan cara berikut:

(0×6)+(-1×5)+(0×5)+(-1×6)+(4×6)+(-1×6)+(0×7)+(-1×5)+(0×5)=2

(6)



Hasil konvolusi = 6. Nilai ini dihitung dengan cara berikut:

(0×5)+(-1×5)+(0×2)+(-1×6)+(4×6)+(-1×2)+(0×5)+(-1×5)+(0×3)=6

Dengan cara yang sama seperti di atas, maka *pixel-pixel* pada baris ketiga dikonvolusikan sehingga menghasilkan citra hasil sebagai berikut:

	3	0	2	
	0	2	6	
	6	0	2	

2.2.2 Kernel Filter

Kernel atau *mask* memberikan petunjuk tentang apa yang harus dilakukan filter terhadap data. Pada umumnya kernel mempunyai panjang dan lebar ganjil. Pola bilangan ganjil n bertujuan agar matriks kernel mempunyai jari-jari r sehingga n=2r-1. Contoh cara penentuan lokasi entri-entri matriks dapat dilihat pada contoh gambar 2.1 dengan (*i*,*j*) yang berjalan dari -2 hingga 2 dan (x,y) yang berjalan dari 0 sampai 4.

2.2.3 Filter Gaussian

Filter Gaussian adalah salah satu filter linier dengan nilai pembobotanuntuk setiap anggotanya dipilih berdasarkan bentuk fungsi Gaussian. FilterGaussian dipilih sebagai filter penghalusan berdasarkan pertimbangan bahwafilter ini mempunyai pusat kernel.

Menurut Usman Ahmad (2005:70), filter Gaussian sangat baik untuk menghilangkan *noise* yang bersifat sebaran nomal, yang banyak di jumpai pada sebaran citra hasil proses digitasi menggunakan kamera karena merupakan fenomena alamiah akibat sifat pantulan cahaya dan kepekaan sensor cahaya pada kamera itu sendiri.

-2	97	152	143	108	165
-1	160	146	107	124	125
0	143	153	146	120	122
1	139	213	119	85	255
2	108	121	128	54	185
	-2	-1	0	1	2

Gambar 2.1 Penentuan lokasi entri pada kernel filter

Untuk menghitung atau menentukan nilai-nilai setiap elemen dalam filter penghalus Gasussian yang akan dibentuk berlaku persamaan (2.2):

$$G(i,j) = c.e - \frac{(i-u)^2+(j-v)^2}{2\sigma^2} \quad \text{.....(2.2)}$$

(Sumber: Gonzalez, R.C.; Woods, R.E. 2002)

dimana: c dan σ = konstanta

G(i,j) = elemen matriks kernel *gauss* pada posisi(i,j)
u,v = indeks tengah dari matriks kernel *gauss*

Dalam hal ini, σ adalah deviasi standar dan piksel pada pusat (y, x) mendapatkan bobot terbesar berupa 1.

Filter *Gaussian* paling tidak berukuran 5x5. Sebagai contoh, bobot-bobotnya dapat diperoleh dengan membuat σ^2 bernilai 1. Dengan demikian:

$G(0,0) = e^{-0} = 1$

$G(1,0) = G(0,1) = G(-1,0) = G(0,-1) = e^{-1/2} = 0{,}6065$

$G(1,1) = G(1,-1) = G(-1,1) = G(-1,-1) = e^{-1} = 0{,}3679$

$G(2,1) = G(1,2) = G(-2,1) = G(-2,-1) = e^{-5/2} = 0{,}0821$

$G(2,0) = G(0,2) = G(0,-2) = G(-2,0) = e^{-2} = 0.1353$

$G(2,2) = G(-2,-2) = G(-2,2) = G(2,-2) = e^{-4} = 0{,}0183$

Dengan mengatur nilai terkecil menjadi 1, maka setiap nilai di atas perlu dikalikan dengan 55 (diperoleh dari 1/0,0183 dan kemudian hasilnya dibulatkan

ke atas). Dengan demikian, diperoleh hasil seperti berikut, yang diperoleh dengan mengalikan nilai G(x,y) di depan dengan 55.

1	5	7	5	1
5	20	33	20	5
7	33	55	33	7
5	20	33	20	5
1	5	7	5	1

Setelah dinormalisasi diperoleh filter seperti berikut:

1/339

1	5	7	5	1
5	20	33	20	5
7	33	55	33	7
5	20	33	20	5
1	5	7	5	1

Gambar 2.2 memberikan contoh penerapan filter *Gaussian*(Sumber: Kadir, Abdul dan Adhi Susanto. 2013)

Hasilnya, terjadi sedikit penghalusan pada daerah yang intensitasnya berbeda jauh, seperti yang ditunjukkan pada gambar 2.3.

2.3 *Noise*

Noise adalah citra atau gambar atau piksel yang mengganggu kualitas citra. *Noise* dapat disebabkan oleh gangguan fisis (optik) pada alat akuisisi maupun secara disengaja akibat proses pengolahan yang tidak sesuai, selain itu *noise* juga dapat disebabkan oleh kotoran-kotoran yang terjadi pada citra. Terdapat beberapa *noise* sesuai dengan bentuk dan karakteristik jenis, yaitu *salt&pepper*, *gaussian*, *uniform*, dan *noise speckle*. Banyak metode yang ada dalam pengolahan citra yang bertujuan untuk mengurangi atau menghilangkan *noise*. *Noise* muncul biasanya sebagai akibat dari pembelokkan yang tidak bagus

(sensor *noise*, *photographic gainnoise*). Gangguan tersebut umumnya berupa variasi intensitas suatu piksel dengan piksel-piksel tetangganya. Secara visual, gangguan mudah dilihat oleh mata karena tampak berbeda dengan piksel tetangganya. Piksel yang mengalami gangguan umumnya memiliki frekuensi tinggi. Komponen citra yang berfrekuensi rendah umumnya mempunyai nilai piksel konstan atau berubah sangat lambat. Operasi *denoise* dilakukan untuk menekan komponen yang berfrekuensi tinggi dan meloloskan komponen yang berfrekuensi rendah (Munir, 2004) Reduksi *noise* adalah suatu proses menghilangkan atau mengurangi *noise* dari suatu signal. Reduksi *noise* secara konsep hampir sama penerapannya pada setiap jenis signal, tetapi untuk implementasinya, reduksi *noise* tergantung dari jenis signal yang akan diproses.

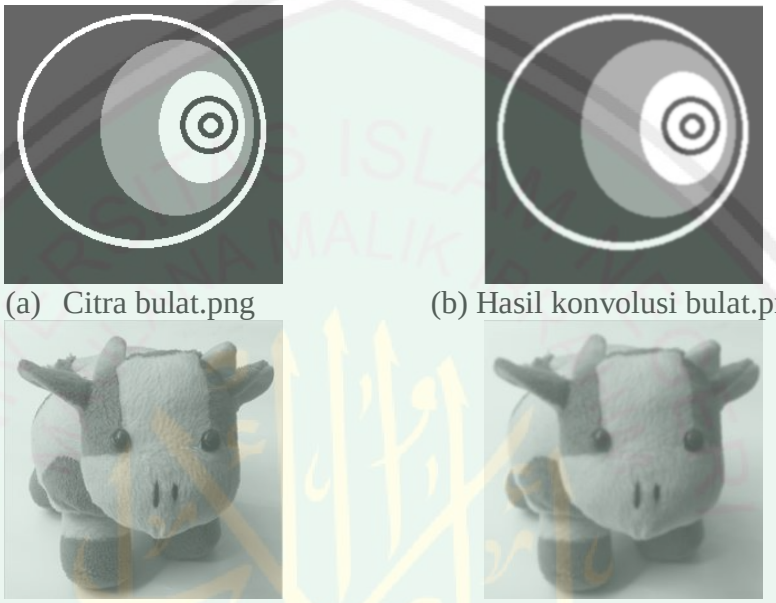
Secara umum metode untuk mereduksi *noise* dapat dilakukan dengan cara melakukan operasi pada citra digital dengan menggunakan suatu jendela ketetanggan, kemudian jendela tersebut diterapkan dalam citra. Proses tersebut dapat juga disebut proses filtering.

Berdasarkan bentuk dan karakteristiknya, *noise* pada citra dibedakan menjadi beberapa macam yaitu:

2.3.1 *Gaussian Noise*

Noise / Derau merupakan informasi yang tidak diinginkan dalam gambar digital. *Noise* menghasilkan efek yang tidak diinginkan seperti arrtefak, tepi realistis, garis tak terlihat, sudut, objek kabur dan menunggu adegan latar belakang. Berikut ini adalah jenis-jenis *noise* yang hadir merusak gambar asli.

Salah satu jenis *noise* yang mengganggu adalah *gaussian noise*.*Gaussian noise* disebut juga sebagai *noise* elektronik karena muncul di amplifier atau detektor. *Gaussian Noise* disebabkan oleh sumber-sumber alam seperti getaran termal atom dan sifat diskrit dari radiasi benda hangat.



(a) Citra bulat.png (b) Hasil konvolusi bulat.png

(c) Citra boneka.png (d) Hasil konvolusi boneka.png

Gambar 2.3 Efek filter *Gaussian*(Sumber: Kadir, Abdul dan Adhi Susanto. 2013)

Gaussian noise umumnya mengganggu nilai abu-abu dalam gambar digital. Itulah sebabnya *gaussian noise* dasarnya dirancang dan karakteristik oleh PDF atau menormalkan histogram dengan nilai abu-abu. PDF yang mewakili sifat paling acak dalam bentuk satu dimensi seperti berikut:

$$p(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(z-\mu)^2}{2\sigma^2}} \quad \text{..... (2.3)}$$

(Sumber: Gonzalez, R.C.; Woods, R.E. 2002)

Dalam hal ini, μ adalah nilai rerata dan σ adalah deviasi standar (atau akar varians) variabel random. PDF-nya ditunjukkan pada Gambar 2.4.

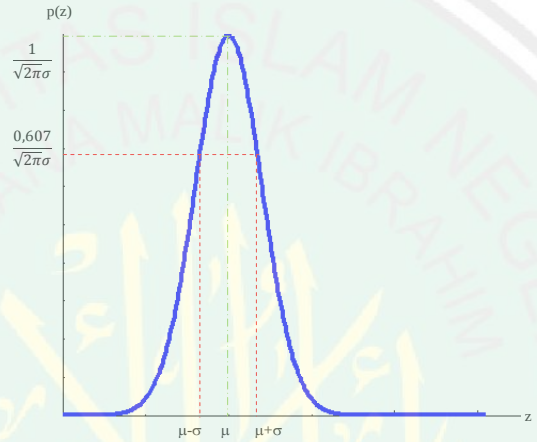
Derau *Gaussian* dapat dilakukan dengan menggunakan fungsi pembangkit bilangan acak. Sebagai contoh, pada *Octave* terdapat fungsi bernama **randn** yang berguna untuk menghasilkan bilangan acak yang terdistribusi secara normal

dengan nilai berkisar antara 0 dan 1. Nah, rumus untuk mendapatkan derau *Gaussian* yang acak dengan deviasi standar sebesar σ dan rerata sama dengan μ adalah seperti berikut:

$$d = randn * \sigma + \mu \quad \text{.....(2.4)}$$

(Sumber: Gonzalez, R.C.; Woods, R.E. 2002)

Apabila μ berupa nol, rumus di atas dapat disederhanakan menjadi d = randn * σ.



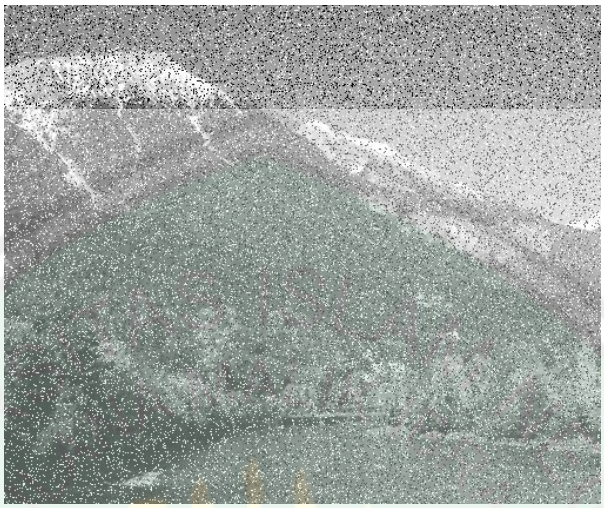
Gambar 2.4 Fungsi kepadatan probabilitas derau *Gaussian*(Sumber: Kadir, Abdul dan Adhi Susanto. 2013)

Contoh penambahan derau pada citra berskala keabuan ditunjukkan pada fungsi drgaussian. Argumen pertama berupa citra berskala keabuan. Argumen kedua bersifat opsional dan menyatakan nilai deviasi standar. Nilai bawaannya berupa 1. Argumen ketiga bersifat opsional dan menyatakan nilai rerata. Nilai bawaannya berupa 0.

2.3.2 *Salt and Pepper Noise*

Salt and Pepper Noise adalah bentuk noise yang biasanya terlihat titik-titik hitam dan putih pada citra seperti tebaran salt dan pepper. Penerapan dari noise ini tergantung nilai pembangkit yang di pakai . Nilai yang dibangkitkan

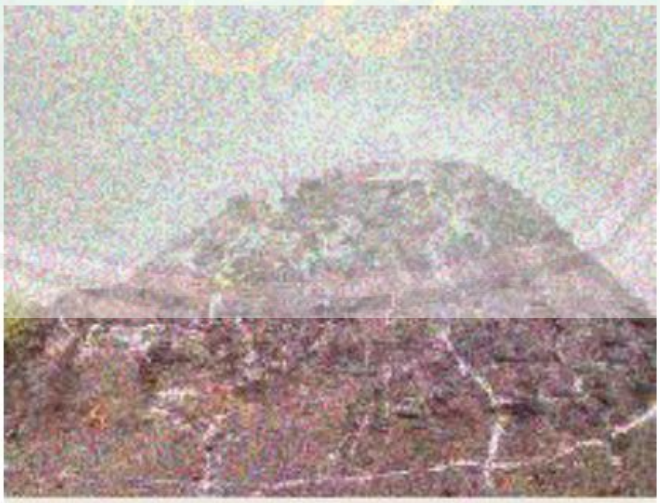
adalah nilai 0 dan 255 yang menunjukkan warna hitam dan putih.*Noise* jenis ini sering muncul pada citra yang diperoleh melalui kamera.



Gambar 2.5 Contoh *noise salt & pepper* dengan probabilitas 0,2**(**Sumber: Kadir, Abdul dan Adhi Susanto. 2013)

2.3.3 *Speckle Noise*

Noise ini dapat dibangkitkan dengan cara membangkitkan bilangan 0 (warna hitam) pada titik-titik yang secara probabilitas lebih kecil dari nilai probabilitas *noise*.



Gambar 2.6 Speckle noise dengan probabilitas 0,04(Sarode, Milindkumar V dan Prashant R. Deshmukh. 2011)

2.4 Komputasi Pararel

Menurut Wilkinson dan Allen (2005) mengatakan bahwa komputasi pararel merupakan komputasi yang melakukan operasi yang sama pada elemen-elemen data yang berbeda secara serentak. Komputasi pararel digunakan untuk menyelesaikan suatu permasalahan besar, dengan memecah-mecah permasalahan menjadi bagian-bagian dari permasalahan yang lebih kecil(sub-masalah). Kemudian sub-masalah tersebut diselesaikan oleh kumpulan-kumpulan dari prosesor(*multi-processors*) yang nantinya terlibat dalam pengeksekusian masalah. Setiap sub-masalah diselesaikan oleh satu prosesor. Sehingga penyelesaian permasalahan menggunakan multi-prosesor dapat lebih cepat dibandingkan dengan penyelesaian permasalahan menggunakan satu prosesor. Tujuan utama komputasi pararel adalah untuk mempersingkat waktu eksekusi program yang menggunakan komputasi serial. Selain itu, komputasi pararel digunakan untuk menyelesaikan komputasi yang sangat kompleks. Komputasi serial yang ada saat ini belum cukup mampu menyelesaikan komputasi yang sangat kompleks.

Secara umum, paradigma komputasi pararel ada dua jenis, yaitu *parallelism implisit* dan *parallelism eksplisit*. *Parallelism implisit* merupakan suatu pendekatan dimana penulis program tidak perlu memperhatikan masalah pembagian *task* ke beberapa prosesor dan memori beserta sinkronisasinya. Pembagian *task* dan sinkronisasi ditangani sepenuhnya oleh sistem divawah program(sistem operasi atau mesin virtual) atau *compiler*.

Sedangkan paradigma *parallelism eksplisit* harus memperhatikan dan menangani masalah pembagian *task* dan komunikasinya. Komunikasi antar dua atau lebih *task* pada umumnya dilakukan dengan menggunakan *shared-*

*memory*atau *message-passing*. Pada s*hared-memory*, *task-task* menggunakan memori(*address space*) yang sama, dimana masing-masing *task* dapat menulis atau membaca secara *asinkron*. Dan pada *message passing*, setiap *task* mempunyai memori(*address space)* lokal masing-masing.

2.5 GPU

GPU adalah sirkuit khusus yang dirancang untuk cepat memanipulasi dan mengubah memori yang sedemikian rupa sehingga mempercepat pembangunan gambar dalam frame buffer yang dimaksudkan untuk output untuk tampilan. GPU saat ini digunakan dalam sistem tertanam, ponsel, komputer, workstation, dan konsol game. GPU modern sangat efisien dalam memanipulasi grafis komputer, dan struktur mereka sangat pararel membuat mereka lebih efektif daripada tujuan umum CPU untuk algoritma di mana pengolahan blok besar data dilakukan secara pararel. Dalam sebuah komputer pribadi, GPU dapat hadir pada kartu video (kartu grafis), atau dapat pada motherboard, atau dalam CPU tertentu. Saat ini lebih dari 90% komputer desktop baru dan notebook telah terintegrasi GPU, yang biasanya jauh lebih kuat daripada yang ada pada kartu video khusus.

Istilah GPU ini didefinisikan dan dipopulerkan oleh Nvidia pada tahun 1999, yang memasarkan kartu video GeForce 256 sebagai "GPU pertama di dunia , yang merupakan sebuah chip prosesor tunggal yang terintegrasi dengan transformasi, pencahayaan, segitiga setup / kliping, dan mesin render yang mampu memproses minimal 10 juta poligon per detik ". Sedangkan perusahaan saingan nvidia yaitu ATI Technologies tidak ingin menggunakan istilah yang dibuat oleh nvidia, ATI menciptakan istilah yang berbeda yaitu VPU (singkatan

dari Virtual Processing Unit) dengan menghadirkan kartu grafis Radedon 9700 pada tahun 2002.

Pada awalnya kartu grafis hanya berfungsi sebagai media untuk menampilkan keluaran ke layar monitor. Kemudian masyarakat mulai mengenal apa yang disebut dengan game komputer dan semakin banyak masyarakat yang memainkannya. Mereka butuh tampilan yang sangat real atau mendekati nyata. Mereka ingin bermain dengan lancar, dalam arti tidak terganggu dengan tampilan yang berjalan tersendat-sendat. Untuk itulah para produsen GPU berlomba-lomba untuk menyajikan teknologi pengolah grafis yang paling mutakhir untuk memenuhi kebutuhan tersebut. Seolah-olah GPU diciptakan khusus hanya untuk melayani para pengguna game. Dengan adanya GPU, game menjadi semakin cepat dan semakin realistis, dan secara otomatis semakin banyak bermunculan game-game baru yang mengisi pasaran industri game.

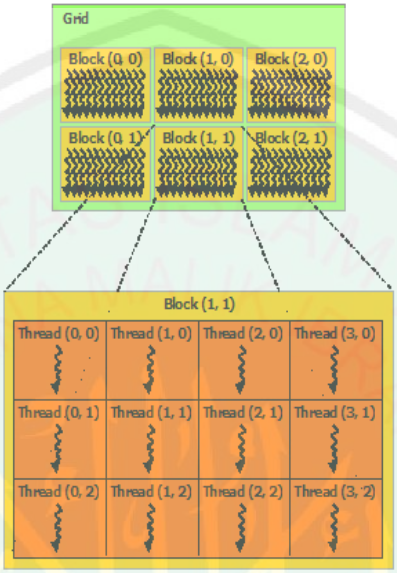
Seiring dengan perkembangan teknologi dan semakin besarnya kebutuhan manusia, kartu grafis juga mengalami perkembangan mulai dari ukuran, kapasitas, konsumsi energi, teknologi serta penerapannya. Tingginya kebutuhan akan grafis, GPU yang dulu hanya diterapkan pada komputer desktop atau laptop, kini dapat ditemukan pada perangkat elektronik lain, misalnya handphone, PDA, iPod dan perangkat mobile lain. Teknologi diciptakan untuk memudahkan pekerjaan manusia, begitu pula dengan GPU. Penerapannya mengalami perkembangan untuk memenuhi kebutuhan manusia yang lain. Tidak lagi hanya untuk memproses grafis saja, namun juga diharapkan dapat memenuhi kebutuhan lain yang diinginkan. Misalnya GPU juga dapat memproses seperti layaknya sebuah prosesor, mengeluarkan hasil, bahkan sampai memprediksi suatu keadaan.

2.6 Nvidia CUDA

CUDA (*Compute Unified Device Architecture*) merupakan API yang dikembangkan oleh NVIDIA. CUDA adalah model pemrograman untuk komputasi paralel yang memungkinkan peningkatan dalam kinerja komputasi dengan memanfaatkan kelebihan dari GPU (NVIDIA, 2014). Pada tahun 2006 , NVIDIA meluncurkan bahasa pemrograman yang bernama CUDA, dan bahasa pemrograman tersebut secara khusus untuk melakukan komputasi yang umum atau sering disebut dengan GPGPU(*General Purpose Graphic Processing Unit*). CUDA adalah ekstensi dari bahasa pemrograman C yang ditambahkan dengan beberapa sintax untuk bekerja dengan GPU (*Jackson*, 2009).

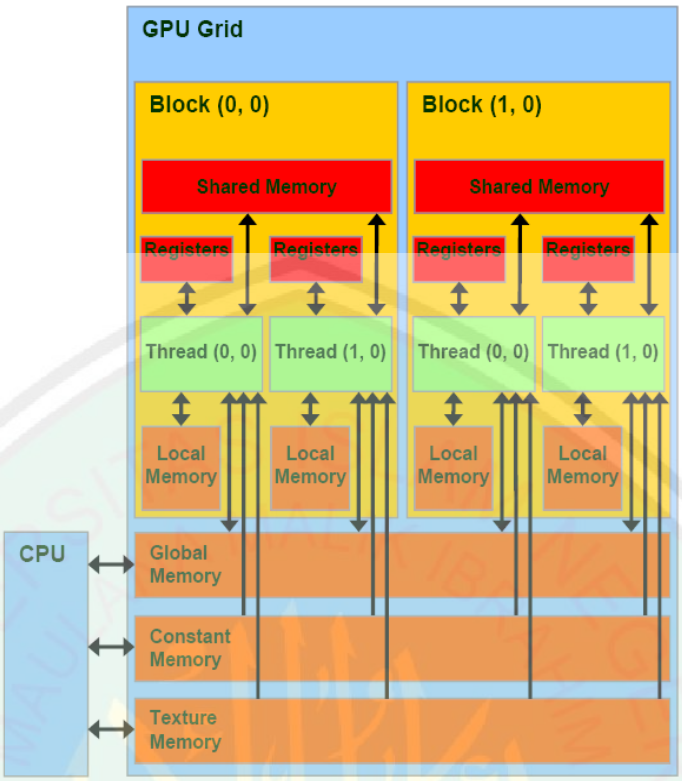
CUDA pada dasarnya terbagi menjadi dua bagian utama , yaitu *Thread* dan *Block(Grid)*. *Thread* memiliki *memory* sendiri dan akan mengerjakan unit pekerjaan yang kecil dan akan berjalan secara bersamaan dengan *thread* lain, sehingga waktu yang dibutuhkan untuk mengerjakan pekerjaan seluruhnya akan menjadi lebih singkat. *Thread-thread* tersebut dikelompokkan menjadi *block*, yaitu kumpulan *thread* yang memiliki satu *memory* yang dapat digunakan oleh setiap *thread* dalam *block* tersebut secara bersama-sama untuk media komunikasi antar thread tersebut yang dinamakan dengan *shared memory*. Setiap *block* tersebut akan dikelompokkan lagi menjadi sebuah *grid* yang merupakan kumpulan dari semua *block* yang digunakan dalam suatu komputasi.

Dalam proses komputasim terdapat proses antrian data yang disebut sebagai *warp*.*Warp* memiliki efek pada kinerja GPU, sehingga jumlah *block* dan *thread* yang digunakan memiliki efek pada kecepatan proses komputasi.



Gambar 2.7 Struktur Unit Pemroses pada CUDA.
Sumber :Doc.nvidia.com,2016.

CUDA memiliki beberapa macam *memory* yang dapat digunakan untuk proses komputasi yaitu; *Global Memory*, *Shared Memory*, *Texture Memory*, *Register*, *Local Memory*, dan *Constant Memory*.



Gambar 2.8 Struktur Memori pada CUDA (Sumber :Doc.nvidia.com,2016)

Cuda memiliki wilayah memory yang terbagi shared memory,local memory, global memory dan constant memory dengan aliran data sesuai pada mekanisme memory cuda seperti yang terlihat pada gambar(doc.nvidia.com).

2.7 OpenCL

OpenCL(*Open Computing Language*) adalah standar kerangka industri untuk pemrograman komputer yang terdiri dari kombinasi CPU, GPU, dan perangkat komputasi diskrit lainnya diatur dalam satu platform (Munshi, Aaftab, *et all.*2012). OpenCL lebih dar sebuah bahasa pemrograman. Karena di dalam OpenCL terdapat bahasa, API, *library*, dan sistem runtime untuk mendukung pengembangan perangkat lunak. OpenCL termasuk *framework* yang berbasis bahasa C. API runtime untuk mengendalikan platform dan device OpenCL.

Pemrograman OpenCL memungkinkan program untuk dijalankan pada CPU, GPU, proseosr sel, DSP, dan device lainnya. Berikut adalah dasar-dasar OpenCL :

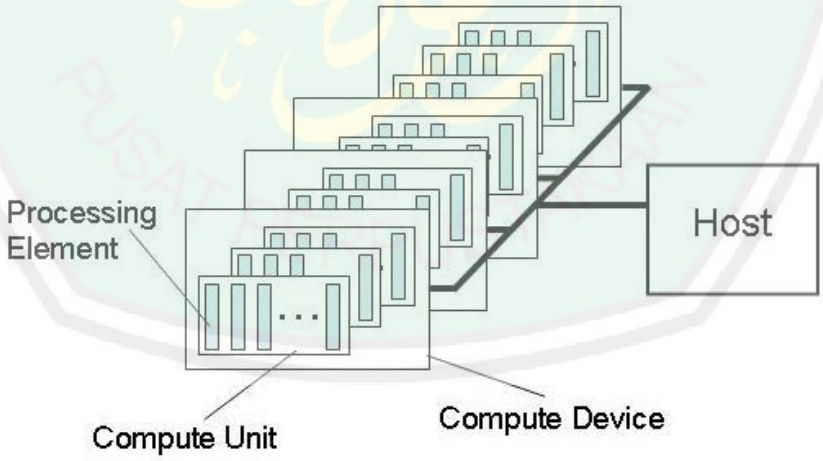
a. *Platform Model*

b. *Memory Model*

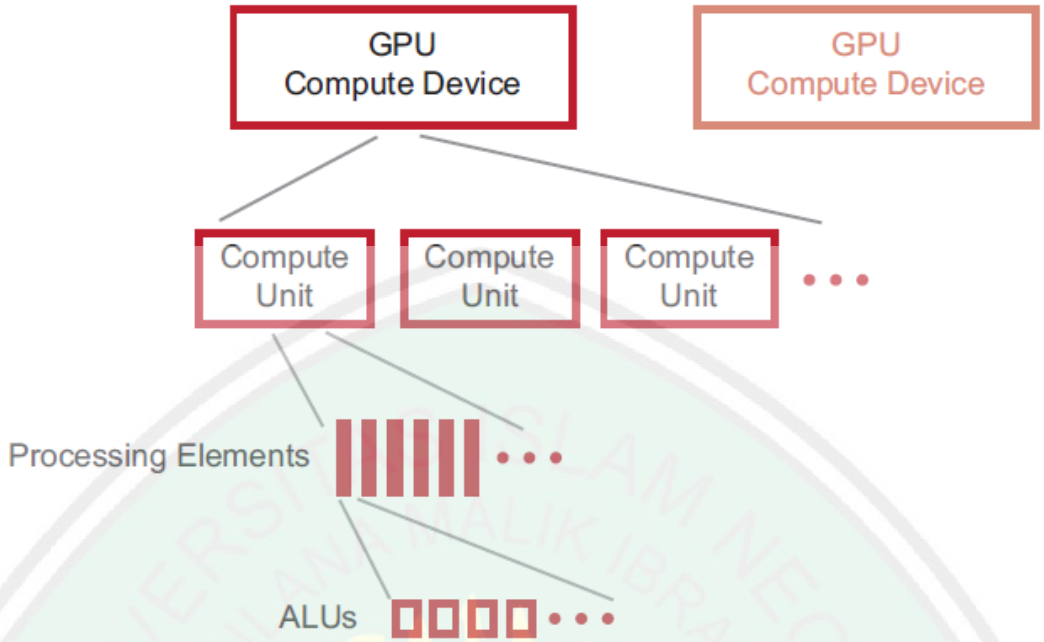
c. *Execution Model*

d. *Programming Model*

Model platform (*platform model*) OpenCL terdiri dari sebuah host yang terhubung ke satu atau lebih perangkat OpenCL. Sebuah perangkat OpenCL dibagi menjadi beberapa *compute uni* (CU). Dalam CU terdapat beberapa *processing element* (PE). Perhitungan atau eksekusi program terjadi dalam PE. Aplikasi OpenCL menyampaikan perintah dari host ke perangkat untuk melaksanakan perhitungan dalam PE. Model platform untuk OpenCL ditunjukkan pada gambar 2.9.

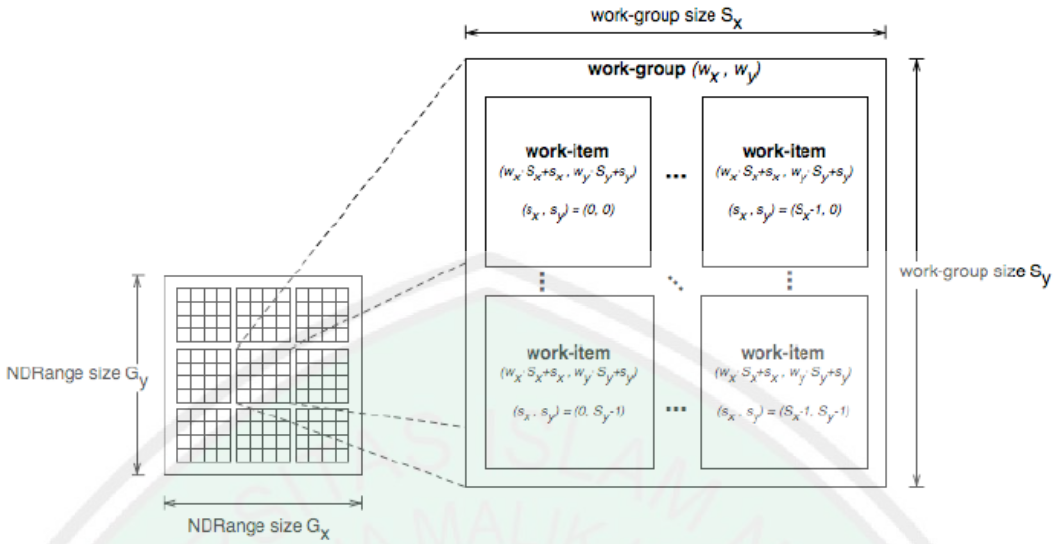


Gambar 2.9 Model platform **(**Sumber: Munshi, Aaftab. 2012)



Gambar 2.10Struktur generalisasi AMD GPU *compute device* **(**Sumber:)

Pelaksanaan program OpenCL terjadi dalam dua bagian: kernel yang mengeksekusi pada satu atau beberapa perangkat OpenCL dan program host yang mengeksekusi pada host. Program host mendefinisikan konteks untuk kernel dan mengelola eksekusi. Eksekusi model OpenCL ditentukan oleh bagaimana kernel mengeksekusi. Ketika kernel diajukan untuk eksekusi oleh host, *index space* didefinisikan. Kernel dalam OpenCL disebut dengan istilah *work-item*, kernel mengeksekusi setiap titik dalam *index space* seperti yang ditunjukkan pada gambar 2.11. Setiap *work-item* mengeksekusi kode yang sama dan data yang berbeda. *Work-item* tersusun dalam *work-group*. *Work-group* memberikan ID *work-group* yang unik dengan dimensi yang sama sebagai *index space* yang diguakan *work-item*. *Work-item* memberikan ID lokal yang unik dalam *work-group* sehingga *work-item* dapat diidentifikasi secara unik oleh ID global.



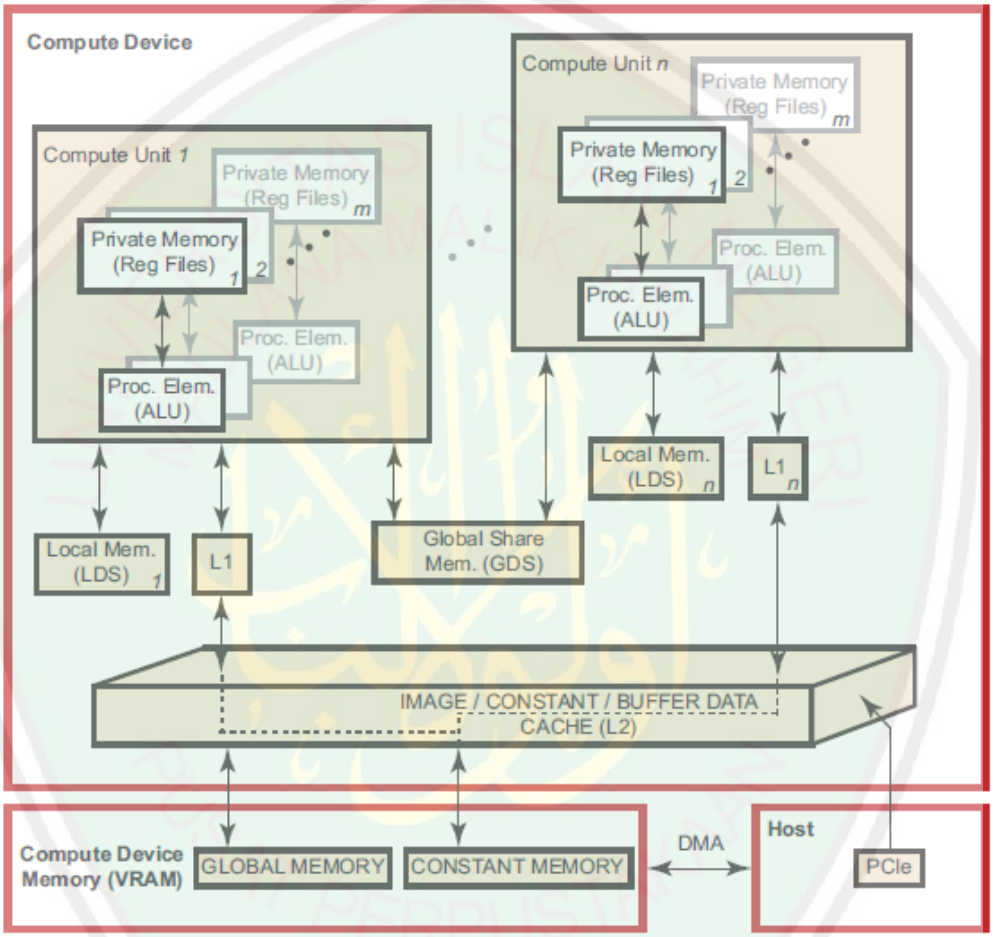
Gambar 2.11 Contoh *NDRange* yang menampilkan *work-item*(Sumber: Banger, Ravishekhar *and* Koushik Bhattcharyya. 2013)

OpenCL memiliki empat daerah memori yang berbeda untuk mengakses data. Empat daerah tersebut ditunjukkan pada gambar 2.12.

a. *Private memory* merupakan wilayah memori khusus untuk *work-item*. Data dalam memori ini hanya bisa di akses oleh sebuah *work-item*.
b. *Local memory* merupakan wilayah memori khusus untuk *work-group*. Memori lokal dapat mengakses data dari *work-item satu dengan work-item* lainnya.
c. *Global memory* merupakan wilayah memori dimana data dapat dibaca / ditulis oleh semua *work-group*.
d. *Constant memory* merupakan wilayah dimana data tidak dapat berubah selama eksekusi, karena memori ini hanya bisa membaca memori global.

Model eksekusi OpenCL mendukung model pemrograman data pararel dan model pemrograman *task* pararel. Model pemrograman data pararel mendefinisikan urutan instruksi yang diterapkan ke beberapa elemen dari objek memori. *Index space* yang terkait dengan model eksekusi OpenCL mendefinisikan *work-item* dan bagaimana pemetaan data ke *work-item.* Dalam model

pemrograman data pararel, pemetaan data satu-ke-satu bukanlah suatu keharusan. Sedangkan pada model pemrograman *task* pararel, kernel dieksekusi pada unit komputasi dengan *work-group* yang berisi *work-item* tunggal. Dengan model ini, pengguna mengungkapkan pararelisme dengan menggunakan jenis data vektor.



Gambar 2.12 Model memori **(**Sumber:)

BAB III

ANALISIS DAN PERANCANGAN

Bab ini membahas tentang analisis dan perancangan penelitian. Analisis dilakukan dengan mencari dan menentukan beberapa kebutuhan seperti masukan, fungsi-fungsi yang dibutuhkan, dan keluaran dari program yang dibangun. Analisis dan perancangan ini meliputi analisis permasalahan, desain data, desain proses, dan metode analisis data.

3.1 Analisis Permasalahan

Subbab ini akan membahas mengenai analisis permasalahan yang dikerjakan dalam skripsi ini. Tujuan pembuatan program ini adalah untuk memperbaiki citra ber-*noise* dengan menggunakan metode *gaussian filter* pada komputasi paralel. Komputasi paralel dapat menyelesaikan suatu permasalahan dengan memecah-mecah permasalahan menjadi bagian-bagian yang lebih kecil(sub-masalah). Kemudian sub-masalah tersebut diselesaikan oleh kumpulan-kumpulan dari prosesor(*multi-processors*) yang nantinya terlibat dalam pengeksekusian masalah. Setiap sub-masalah diselesaikan oleh satu prosesor. Sehingga penyelesaian permasalahan menggunakan multi-prosesor dapat lebih cepat dibandingkan dengan penyelesaian permasalahan menggunakan satu prosesor.

3.2 Desain Data

Data yang digunakan untuk penghapusan *noise* menggunakan metode *gaussian filter* pada komputasi paralel terbagi menjadi tiga bagian utama, yaitu

data masukan, data yang digunakan selama proses perbaikan citra ber-*noise* dan data keluaran.

3.2.1 Data Masukan

Data masukan yang pertama adalah citra asli dengan format .jpg. Citra yang digunakan dalam program ini berasal dari 30 foto yang diperoleh dari hasil kamera smartphone ASUS Zenfone5. Kemudian 30 foto tersebut diberi *noise* dengan intensitas *noise* sebanyak 5%, 10%, dan 50% pada setiap data.

3.2.2 Data Selama Proses

Data masukan yang ada akan di alamatkan pada memori sementara yang di transfer ke *device* untuk di proses dalam GPU. Kemudian data tersebut akan dilakukan windowing berupa matriks yang berukuran 3 × 3 atau lebih. Selanjutnya yaitu konvolusi matriks windowing dengan matriks kernel. Kemudian data ini akan disimpan sebagai nilai *pixel* baru yang sudah dilakukan proses reduksi *noise*.

3.2.3 Data Keluaran

Data keluaran yang dihasilkan dari program ini adalah citra hasil *filtering* dengan metode *gasussian filter* pada komputasi paralel. Data lain yang didapatkan adalah *running-time* selama proses *filtering* hingga hasil dari *filtering* pada komputasi paralel. *Running-time* digunakan untuk menghitung waktu yang dibutuhkan untuk memproses suatu program mulai dari awal proses hingga akhir proses. Dengan adanya data perhitungan waktu proses akan dapat diketahui

perbedaan lama waktu yang dibutuhkan antara komputasi paralel dan komputasi serial.

3.3 Desain Proses

Desain proses digunakan untuk mengetahui proses apa saja yang berlangsung pada sistem. Desain proses untuk program ini menggunakan *activity diagram*. *Activity diagram*yang menggambarkan berbagai alir aktivitas dalam program yang sedang dirancang, bagaimana masing-masing alir berawal dan bagaimana alir berakhir. Dalam desain proses ini juga akan dijelaskan bagaimana pengaturan *thread* untuk pemetaan data yang akan diproses dalam komputasi paralel. Jika diilustrasikan secara umum *block diagram* penelitian ini ditunjukkan pada gambar 3.1.

Gambar 3.1 *Block diagram* penelitian

3.3.1 Citra Masukan

Citra masukan yang digunakan adalah citra dua dimensi dengan ukuran XY. Kemudian citra tersebut akan dibagi menjadi tiga bagian, yaitu layer merah, hijau, dan biru. Sehingga nantinya akan terdapat 3 data masukan dari citra yang sama. Pada akhir proses, 3 data masukan tersebut akan digabungkan kembali menjadi satu data yaitu citra hasil filter.

3.3.2 Menyisipkan *Noise*

Pada proses ini dilakukan penambahan *noise* terhadap data citra yang telah dibagi menjadi 3 bagian. Penambahan *noise* akan dilakukan pada masing-masing citra yang telah dibagi menjadi tiga bagian. Pada skripsi ini jenis *noise* yang ditambahkan adalah *salt and pepper noise*. Jenis *noise salt and pepper* merupakan jenis *noise* yang sering terjadi pada citra.

3.3.3 Menghapus *Noise* Menggunakan Metode *Gaussian Filter* pada Komputasi Paralel

Pada skripsi ini diterapkan 2 skenario yang berbeda agar dapat mengetahui perbedaan nilai *running-time* pada masing-masing skenario. Skenario 1 ditunjukkan pada gambar 3.2, pada gambar tersebut digambarkan bagaimana alirberawal hingga berakhir menjadi citra baru. Sedangkan pada gambar 3.3 digambarkan bagaimana jalannya skenario 2. Perbedaannya skenario 1 dengan skenario 2 terletak pada proses komputasi. Pada skenario 1 proses pembagian layer dan penambahan noise dilakukan dalam CPU. Proses windowing sampai dengan proses pemfilteran di lakukan dalam GPU . Sedangkan pada skenario 2 semua komputasi di proses dalam CPU.

Tahap-tahap proses komputasi paralel yang tergambar pada skenario 1 adalah sebagai berikut:

3.3.3.1. Mengambil Nilai Pixel

Data citra diambil nilai pixelnya untuk dialokasikan kedalam memori host buffer (CPU). Data citra diambil dan diubah dalam tipe RGB agar kemudian mudah diolah dalam GPU. Kode program untuk mengambil nilai pixel dapat dilihat pada listing 3.1.

```
int sizeX = gambar.getWidth();
int sizeY = gambar.getHeight();
BufferedImage hasil = new BufferedImage(sizeX, sizeY,
                      BufferedImage.TYPE_INT_RGB);
```

Listing 3.1 Mengambil nilai pixel

3.3.3.2. Pengaturan Platform

Memilih OpenCL device sebagai platform yang akan kita gunakan untuk implementasi. Untuk mendapatkan platform dan OpenCL device dapat menggunakan `clGetPlatfornIDs()` dan `clGetDeviceIDs()`.Kode program untuk mengatur platform dapat dilihat pada listing 3.2.

```
//memperoleh jumlah platform-platform
int numPlatformsArray[] = new int[1];
clGetPlatformIDs(0, null, numPlatformsArray);
int numPlatforms = numPlatformsArray[0];
//memperoleh ID Platform
cl_platform_id platforms[] = new cl_platform_id[numPlatforms];
clGetPlatformIDs(platforms.length, platforms, null);
cl_platform_id platform = platforms[indexPlatform];
//memperoleh jumlah device pada platform
int jumlahDevicesArray[] = new int[1];
clGetDeviceIDs(platform, tipeDevice, 0, null,
           jumlahDevicesArray);
int jumlahDevices = jumlahDevicesArray[0];
//memperoleh ID device
```

```
cl_device_id devices[] = new cl_device_id[jumlahDevices];
clGetDeviceIDs(platform, tipeDevice, jumlahDevices, devices,
          null);
cl_device_id device = devices[indexDevice];
```

Listing 3.2 Mengatur platform yang akan digunakan

3.3.3.3. Membuat OpenCL *Context*

Setelah OpenCL device terpilih, maka OpenCL context dibuat. *Context* mendefinisikan seluruh lingkungan OpenCL, termasuk perangkat, obyek program, kernel OpenCL, hingga *memory object*. Sebuah konteks dapat dikaitkan dengan beberapa perangkat atau dengan hanya satu perangkat. Antrian perintah dan kernel harus dari konteks OpenCL yang sama, tidak bisa dari konteks yang berbeda. Sebelum dapat membuat suatu konteks, dilakukan*queryruntime* OpenCL untuk menentukan vendor platform yang tersedia dalam sistem. Setelah vendor platform dipilih, inisialisasi pelaksanaan OpenCL untuk membuat konteks. Sebuah konteks dapat memiliki sejumlah perangkat terkait, yang dapat berupa CPU atau GPU atau keduanya.*Context* dibuat dengan menggunakan `clCreateContext()`. Kode program untuk membuat *context* dapat dilihat pada listing 3.3.

```
//inisialisasi properties context
cl_context_properties contextProperties = new
              cl_context_properties();
contextProperties.addProperty(CL_CONTEXT_PLATFORM, platform);
//mmbuat context untuk device yang dipilih
cl_context context = clCreateContext(contextProperties, 1, new
              cl_device_id[]{device}, null, null, null);
```

Listing 3.3 Membuat *context*

3.3.3.4. Membuat *Command Queue*

Membuat *command queue* dapat dilakukan dengan menggunakan `clCreateCommandQueue()`. Command queue diciptakan agar dapat mengkoordinasi eksekusi kernel pada device. Sehingga dapat menjalankan tugas-tugas secara independen pada antrian perintah yang berbeda. Kode program untuk membuat *command queue* ditunjukkan pada listing 3.4.

```
//mmbuat command-queue untuk device yang dipilih
cl_command_queue cQueue = clCreateCommandQueue(context, device,
                0, null);
```

Listing 3.4 Membuat *command queue*

Sebuah perintah merupakan sebuah transfer data, atau perintah eksekusi kernel atau hambatan (barrier) dalam antrian perintah. Host akan mengantrikan perintah tersebut pada antrian perintah. Setiap perintah atau tugas dihubungkan dengan sebuah OpenCL event. Event ini dapat digunakan sebagai mekanisme sinkronisasi untuk mengkoordinasikan eksekusi antara host dan perangkat.

3.3.3.5. Membuat *MemoryObject*

Membuat *memoryobject* dengan menggunakan `clCreateBuffer()`. *Memory object* yang dibuat meliputi input dan output. Objek input berisi objek gambar yang akan diproses dan objek output disediakan untuk menampung output yang sudah terproses. Kode program untuk membuat *memory object* dapat dilihat pada listing 3.5.

```
//mmbuat memory object untuk input dan output image
DataBufferInt dataBufferSrc = (DataBufferInt)
          src.getRaster().getDataBuffer();
int dataSrc[] = dataBufferSrc.getData();
inputImageMem = clCreateBuffer(context, CL_MEM_READ_ONLY |
          CL_MEM_USE_HOST_PTR, dataSrc.length * Sizeof.cl_uint,
          Pointer.to(dataSrc), null);
outputImageMem = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
```

```
            imageSizeX * imageSizeY * Sizeof.cl_uint, null,
            null);
```

Listing 3.5 Membuat *memory object*

3.3.3.6. Membuat perintah-perintah OpenCL (OpenCL kernel).

Kernel ini berisi perintah pemrosesan filter dengan metode *gaussian filter*. Berikut ini adalah langkah-langkah untuk menghapus *noise* dengan metode *gaussian filter* pada GPU:

a. Mengirim data ke memori global GPU.

Pada langkah ini data dari CPU dikirim ke GPU. Data yang dikirim merupakan data-data yang dibutuhkan untuk proses penghapusan *noise*, diantaranya adalah nilai pixel, ukuran gambar, ukuran perbesaran kernel. Untuk melakukan pengiriman data dapat dilakukan dengan menggunakan `clSetKernelArg()`.Adapun argumen yang ada pada fungsi `clSetKernelArg` meliputi objek kernel yang argumennya akan diatur ; index dari argumen dimulai dari indeks ke 0 hingga akhir argumen ; spesifikasi ukuran dari `arg_value ; arg_value`sebagai pointer untuk data. Kode program untuk pengirian data ini dapat dilihat pada listing 3.6.

```
long localWorkSize[] = new long[2];
int kernelSizeX = window;
int kernelSizeY = window;
localWorkSize[0] = 16;
localWorkSize[1] = 16;
long globalWorkSize[] = new long[2];
globalWorkSize[0] = round(localWorkSize[0], imageSizeX);
globalWorkSize[1] = round(localWorkSize[1], imageSizeY);
int imageSize[] = new int[]{imageSizeX, imageSizeY};
int kernelSize[] = new int[]{kernelSizeX, kernelSizeY};
clSetKernelArg(clKernel, 0, Sizeof.cl_mem,
        Pointer.to(inputImageMem));
```

```
clSetKernelArg(clKernel, 1, Sizeof.cl_mem,
        Pointer.to(outputImageMem));
clSetKernelArg(clKernel, 2, Sizeof.cl_int2,
        Pointer.to(imageSize));
clSetKernelArg(clKernel, 3, Sizeof.cl_int2,
        Pointer.to(kernelSize));
```

Listing 3.6 Membuat Argumen

b. Membuat Matriks Windowing.

Matriks windowing digunakan sebagai data yang akan dikonvolusikan dengan matriks mask. Pada penelitian ini, akan dilakukan perbesaran matriks windowing 3×3, 5×5, dan 7×7. Perbesaran ini dilakukan untuk mengetahui perbedaan waktu yang dituhkan pada perbesaran matriks windowing yang berbeda. Kode program untuk membuat matriks windowing dapat dilihat pada listing 3.7.

```
  int uiRGB[9];
    int maskOrigin = window/2;
    int ind = 0;
    for(int mx=0; mx<window; mx++){
        for(int my=0; my<window; my++){
            int ix = x - maskOrigin + mx;
            int iy = y - maskOrigin + my;
            int i = mul24(iy, imageSize.x) + ix;
            if(i<0){
                uiRGB[ind] = 0;
            }else{
                uiRGB[ind] = pSrc[i];
            }
            ind++;
        }
    }
```

Listing 3.7 Membuat matriks *windowing*

c. Menghitung Nilai Kernel Mask

Matriks kernel *gauss* di dapat dari fungsi komputasi dari distribusi *gaussian*, seperti pada persamaan 3.1. Kode program untuk perhitungan kernel mask pada OpenCL dapat dilihat pada listing 3.8.

$$G(i,j) = \frac{1}{2\pi\sigma^2} . e - \frac{x^2+y^2}{2\sigma^2} \quad \text{.....(3.1)}$$

(Sumber: Putra, Darma. 2010)

Dimana: σ = konstanta

G(i,j) = elemen matriks kernel *gauss* pada posisi(i,j)

x,y = indeks tengah dari matriks kernel *gauss*

d. Penghapusan *Noise* Menggunakan Metode *Gaussian Filter*

Data yang telah dikirim ke GPU akan dilakukan proses penghapusan *noise* dengan menggunakan OpenCL. Pada proses ini sama saja ketika kita melakukan proses penghapusan *noise* dengan menggunakan bahasa java yaitu dilakukan konvolusi antara matriks windowing dengan kernel mask. Konvolusi tersebut dapat dirumuskan seperti pada persamaan 3.2.*Pixel* yang dicari nilai barunya sebagai *pixel* tengah dan bobotnya dikalikan dengan bobot pada *pixel* tengah matrik kernel, lalu dijumlahkan dengan hasil perkalian antara bobot *pixel-pixel* tetangganya dengan bobot matrik kernel. Kemudian hasil penjumlahan dikalikan dengan 1/K dimana K adalah jumlah seluruh bobot matrik kernel. Kode program untuk konvolusi ini ditunjukkan pada listing 3.9.

```
float mask[3][3];
    float mask1D[9];
        float phi = M_PI_F;
        for (int i = 0; i < window; i++) {
            for (int j = 0; j < window; j++) {
                float maskk = (1/(2 * phi * pow(sigma,2))) *
                              exp(-((i*i)+(j*j))/2*
                              pow(sigma,2));
```

```
                mask[i][j] = maskk;
                mask1D[i*window+j]=mask[i][j];
            }
        }
```

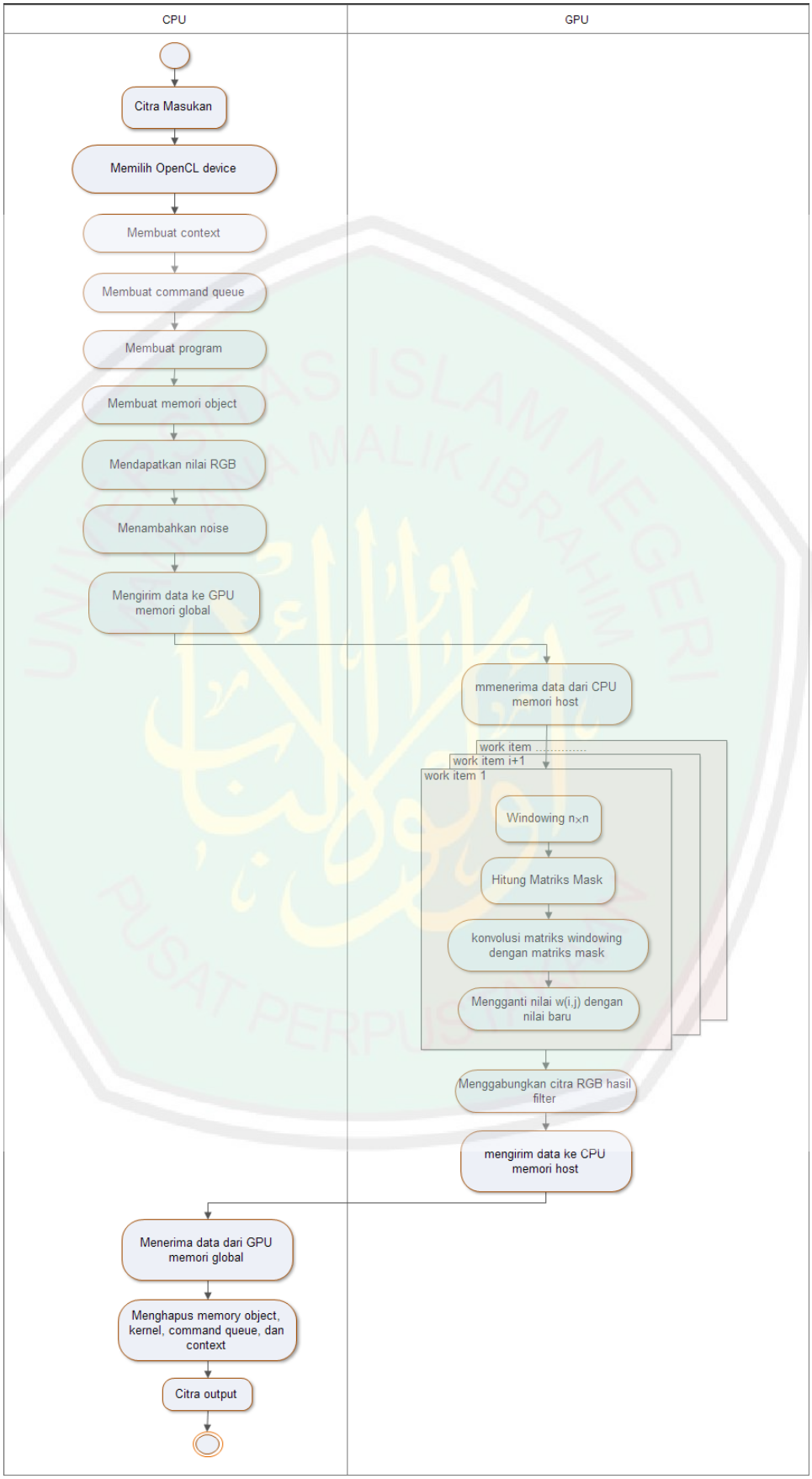
Listing 3.8 Menghitung nilai kernel mask

$$W(i,j) = \frac{1}{K} . \sum_{p=0}^{N-1} \left(\sum_{q=0}^{M-1} G(p,q). Pixel\ A \left(i + p - \frac{(N-1)}{2}, j + q - \frac{(M-1)}{2} \right) \right) \quad (3.2)$$

Dimana: *Pixel* A = citra asli
W(i,j) = bobot hasil perkalian pada posisi(i,j)
N = jumlah kolom matriks kernel
M = jumlah baris matriks kernel
K = penjumlahan semua bobot di G
G(p,q) = elemen matriks kernel *gauss* pada posisi(p,q)

```
for(int k = 0; k<wind; ++k){
        sum += array[k] * mask1D[k];
        sum_mask += mask1D[k];
        }
total = sum/sum_mask;
```

Listing 3.9 Proses *gaussianfilter*



Gambar 3.2 *Activity diagram* Skenario 1

3.3.3.7. Source kernel yang sudah dibuat dijadikan objek program dengan menggunakan fungsi `clCreateProgramWithSource`.

Objek program dibuat untuk device yang berkaitan dengan *context* Opencl. Setelah objek program baru diciptakan untuk sebuah *context*,baik menggunakan biner atau *source* OpenCL, langkah berikutnya adalah membangun program. Program kernel perlu untuk dibangun dan dihubungkan pada saat runtime. Untuk membangun program, fungsi yang digunakan yaitu `clBuildProgram`. Tahap pembangunan (*build*) ini melibatkan kompilasi source code karena program dibuat menggunakan fungsi `clCreateProgramWithSource`. Jika program dibuat menggunakan fungsi `clCreateProgramWithBinary` maka hanya langkah menghubungkan runtime yang dijalankan. listing program untuk implementasinya dapat dilihat pada listing 3.10.

```
cl_program program = clCreateProgramWithSource(context, 1, new
                     String[]{sumber}, null, null);
String compileOption = "-cl-mad-enable";
clBuildProgram(program, 0, null, compileOption, null, null);
```

Listing 3.10 Source code untuk *build* program

3.3.3.8. Setelah proses *building* program, tahap selanjutnya adalah membuat objek kernel dengan fungsi clCreateKernel.

Setiap program adalah kumpulan kernel. Objek program dapat dikatakan sebagai *library* kernel-kernel. Sebuah kernel ketika diantrekan pada antrian perintah, *runtime* OpenCL menghasilkan biner untuk eksekusi pada perangkat. Jika kernel lebih dari satu, maka setiap kernel dijalankan pada perangkat yang berbeda. Dalam hal ini kernel yang dibuat hanya satu kernel. Sebuah objek kernel adalah enkapsulasi untuk satu kesatuan yang dieksekusi secara paralel. Objek

kernel digunakan sebagai jalan untuk melewatkan argumen menggunakan API clSetKernelArg, sebelum *running* kernel menggunakan API clEnqueueNDRangeKernel. Kode untuk membuat objek kernel ini ditunjukkan pada listing 3.11.

```
clKernel = clCreateKernel(program, "Gausss", null);
```

Listing 3.11 Membuat objek kernel

3.3.3.9. Mengeksekusi Kernel.

Untuk eksekusi objek kernel, digunakan fungsi `clEnqueueNDRangeKernel`dimana fungsi tersebut untuk menyebarkan opencl kernel ke perangkat dan mendefinisikan berapa banyak work item yang dibuat untuk mengeksekusi kernel (global_work_size) dan jumlah work item dalam setiap work group (local_work_size). Adapun fungsi `clEnqueueNDRangeKernel`meliputi `command_queue` ; objek Opencl Kernel ; dimensi dari NDRange ; `global_work_offset` yang juga digunakan untuk menghitung global_id dari work item, jika argumen ini bernilai null, maka nilai default adalah 0 ; `global_work_size`yang mendefinisikan global work item dalam setiap dimensi ; `local_work_size`untuk mndefinisikan local work item dalam setiap dimensi ; event_wait_list ; event.

Selanjutnya proses yang dilakukan adalah membaca kembali memori dari device (GPU) ke host buffer (CPU) menggunakan fungsi `clEnqueueReadBuffer`dengan argumen yang berisi `command_queue` ; objek buffer `cl_mem` yang akan dibaca dan akan ditulis pada pointer ; `blocking_read;` `offset` ; total bytes yang dibaca dari device yang dipointerkan oleh buffer ; host memory pointer dari dimana data dibaca ;

`event_wait_list ; event`. Buffer yang dibaca dari perangkat ke host adalah memori buffer untuk output dari memori input yang sudah diproses pada kernel code. Kode program untuk eksekusi kernel dapat dilihat pada listing 3.12.

```
clEnqueueNDRangeKernel(queue, clKernel, 2, null, globalWorkSize,
        localWorkSize, 0, null, null);
DataBufferInt       dataBufferDest       =       (DataBufferInt)
        dest.getRaster().getDataBuffer();
int dataDest[] = dataBufferDest.getData();
clEnqueueReadBuffer(queue,     outputImageMem,     CL_TRUE,     0,
        dataDest.length * Sizeof.cl_uint, Pointer.to(dataDest),
        0, null, null);
```

Listing 3.12 Eksekusi kernel

3.3.3.10. Menghapus *memory object*, *kernel*, *command queue*, dan *context* yang telah dibuat sebelumnya.

Setiap objek program seharusnya dilepas (*released*) dari penyimpanan Opencl setelah digunakan. Untuk melepas objek program, fungsi yang digunakan adalah `clRetainProgram`.Listing program untuk implementasinya dapat dilihat pada listing 3.13.

```
clRetainProgram(program);
```

Listing 3. 13 Source code membersihkan memori objek program

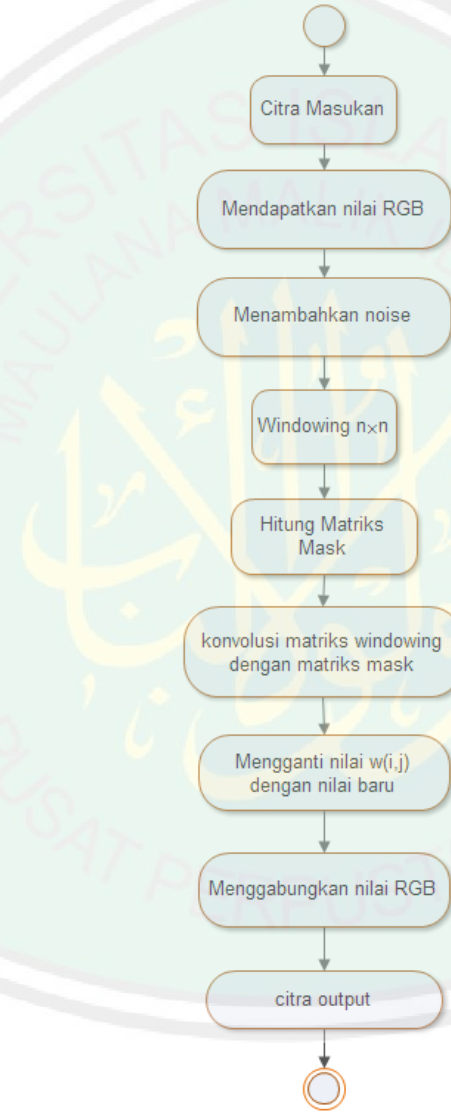
```
Void shutdown() {
    clReleaseKernel(clKernel);
    clReleaseCommandQueue(queue);
    clReleaseContext(context);
}
```

Listing 3. 14 Source code menghapus *memory object*, *kernel*, *command queue*, dan *context*.

Sama halnya dengan objek program, objek kernel juga seharusnya dilepas (*released*) setelah proses selesai. Berbeda dengan objek program, objek

kernel menggunakan fungsi `clReleaseKernel`. Adapun untuk *release command queue* menggunakan fungsi `clReleaseCommandQueue`dan untuk *release context* menggunakan fungsi `clReleaseContext`.

3.3.4 Menghapus *Noise* Menggunakan Metode *Gaussian Filter* pada Komputasi Sekuensial



Gambar 3.3*Activity diagram* Skenario 2

Pada skenario 2 yang digambarkan pada gambar 3.3. Komputasi pada skenario 2 dilakukan dalam CPU. Sehingga komutasi berjalan secara serial. Jika dalam skenario 1 harus menentukan device, membuat context, membuat command queue, membuat program, dan membuat memori object, maka dalam

skenario 2 tidak perlu melakukan langkah-langkah tersebut. Karena dalam skenario 2 komputasi berjalan secara serial, sedangkan pada skenario 1 komputasi berjalan secara paralel. Tahap-tahap proses komputasi yang tergambar pada skenario 2 adalah sebagai berikut:

3.3.4.1 Membaca nilai pixel dari data inputan. Kemudian mendapatkan nilai RGB dari citra input. Sehingga citra di bagi menjadi tiga layer, yaitu layer merah, hijau, dan biru. Kode program untuk mendapatkan nilai RGB dapat dilihat pada listing 3.15

```
int width = image.getWidth();
int height = image.getHeight();
int[][] gbr = new int[width][height];
Color c = null;
        int[][] R = new int[width][height];
        int[][] G = new int[width][height];
        int[][] B = new int[width][height];

        for (int row = 0; row < width; row++) {
            for (int col = 0; col < height; col++) {
                gbr[row][col] = image.getRGB(row, col);
                c = new Color(image.getRGB(row, col));

                R[row][col] = c.getRed();
                G[row][col] = c.getGreen();
                B[row][col] = c.getBlue();
            }
        }
```

Listing 3.15 Mendapatkan nilai RGB

3.3.4.2 Selanjutnya dilakukan windowing berupa matriks yang berukuran 3 × 3 atau lebih. Kode program untuk mencari matriks windowing ditunjukkan pada listing 3.16.

3.3.4.3 Tahapan berikutnya yaitu menghitung matriks mask atau matriks kernel. Matriks kernel *gauss* di dapat dari fungsi komputasi dari distribusi *gaussian*, seperti pada persamaan 3.1. Kode program untuk mencari matriks mask dapat dilihat pada listing 3.17.

```
public float[][] tetanggaan(int[][] input, int window, int
width, int height, int x, int y) {
        float Mwindowing[][] = new float[window][window];
        int variabel = (window-1)/2;
        for (int i = 0; i < window; ++i) {
            for (int j = 0; j < window; ++j) {
                if (((x - variabel + i) >= 0) && ((y -
variabel + j) >= 0) && ((x - variabel + i) < width) && ((y -
variabel + j) < height)) {
                    Mwindowing[i][j] = input[x - variabel + i][y
- variabel + j];
                }
            }
        }
        return Mwindowing;
    }
```

Listing 3.16 Mencari matriks windowing

```
public double[][] Kernelmask(double sigma, int window) {
        double mask[][] = new double[window][window];
        double phi = 3.14;
        for (int i = 0; i < window; i++) {
            for (int j = 0; j < window; j++) {
                double maskk = (1/(2*phi*Math.pow(sigma,2)))*
                  Math.exp(-((i*i)+(j*j))/2*Math.pow(sigma,2));
                mask[i][j] = maskk;
            }
        }
        return mask;
    }
```

Listing 3.17 Mencari matriks mask

3.3.4.4 Konvolusi antara matriks windowing dengan matriks kernel. Konvolusi tersebut dapat dirumuskan seperti pada persamaan 3.2. Kode program untuk konvolusi matriks ditunjukkan pada listing 3.18.

```
for (int k = 0; k < window; k++) {
    for (int l = 0; l < window; l++) {
        konvol[k][l] = tetangga[k][l] * maskk[k][l];
        sum += konvol[k][l];
        sum_mask += maskk[k][l];
     }
}
total = sum / sum_mask;
outputArrays[i][j] = (float) total;
```

Listing 3.18 Konvolusi matriks

3.3.4.5 Menggabungkan nilai RGB yang telah di *denoising* menjadi satu output hasil filter.Kode program untuk penggabungan nilai RGB ditunjukkan pada listing 3.19

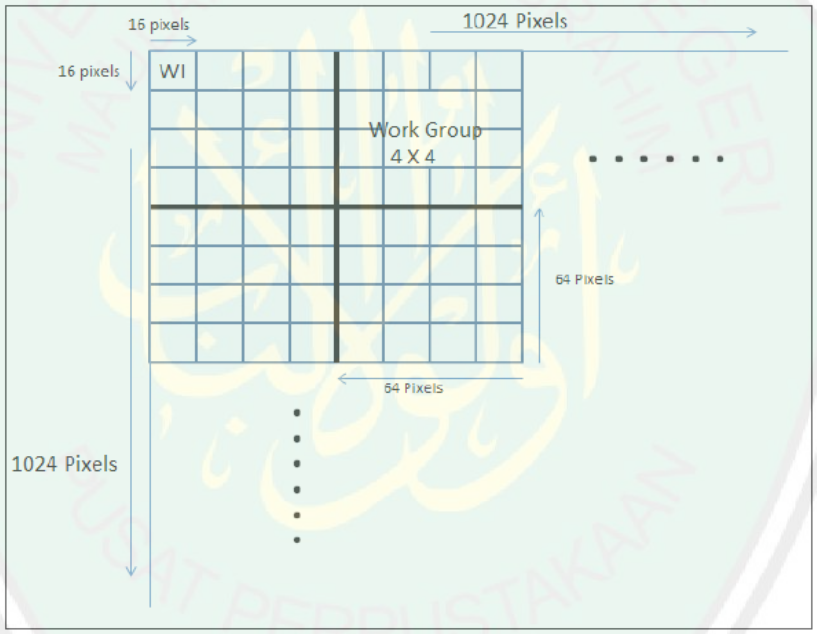
```
for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                Color warna = new Color((int)hasilred[i][j],
(int)hasilgreen[i][j], (int)hasilblue[i][j]); // Color white
                int rgb = warna.getRGB();
                output.setRGB(i, j, rgb);
            }
        }
```

Listing 3.19 Penggabungan nilai RGB

3.4 Manajemen *Thread* untuk Pemetaan data

Pemetaan data dapat memberikan peningkatan kinerja yang signifikan. Data dipetakan melalui host pointer ke device. OpenCL menyediakan dua jenis model pemetaan data, yaitu model pemrograman secara eksplisit dan model pemrograman secara implisit (Munshi, Aaftab. 2012). Secara eksplisit, *programmer* harus mendefinisikan ukuran *work-item* yang akan dieksekusi dalam

pararel dan juga bagaimana jumlah *work-item* dalam satu *work-group*. Secara implisit, *programmer* hanya mendefinisikan ukuran *work-item* yang akan dieksekusi dalam pararel dan OpenCL akan mengatur bagaimana jumlah *work-item* dalam satu *work-group*.Pemetaan data secara eksplisit dapat menggunakan Fungsi `clEnqueueMapImage` akan memetakan data dari device ke space alamat host. Pemetaan ini adalah tugas yang diantrikan pada *device command_queue*. Model pemetaan data secara eksplisit dapat dilihat pada gambar 3.4.



Gambar 3.4 Model eksekusi pada pemetaan data gambar(Sumber : Banger, Ravishekhar dan Koushik Bhattcharyya. 2013)

Data yang diproses merupakan citra asli yang telah ditambahkan *noise*. Kemudian dari data tersebut didapatkan komponen nilai RGB. Sehingga data yang diolah adalah tiga komponen nilai RGB tersebut. Dalam OpenCL gambar masukan dibaca kedalam buffer yang berdekatan dan *image object* dibuat menggunakan fungsi `clCreateImage`.

Pada skripsi ini digunakan model pemrograman secara implisit. Sehingga peneliti hanya menentukan ukuran *work-item* untuk pemetaan data. Dengan ukuran gambar 3264 × 1836 yang nantinya akan dikonvolusi dengan perbesaran kernel sampai 7×7, maka ditentukan ukuran *work-item* 16×16. Sehingga setiap *work-item* memproses 256 piksel.

```
//perhitungan proses penghapusan noise di CPU
before = System.nanoTime();
bufferGbrOutputJV=f.baca(bufferGbrInput,bufferGbrOutputJV,
window, sigma);
after = System.nanoTime();
//diambil durasi waktu selama filter dengan milisecond
durationMS = (after - before) / 1e6;
//perhitungan proses penghapusan noise di GPU
before = System.nanoTime();
bufferGbrOutputCL=jop.filter(bufferGbrInput,  bufferGbrOutputCL,
window);
after = System.nanoTime();
//diambil durasi waktu selama filter dengan milisecond
durationMS = (after - before) / 1e6;
```

Listing 3.20 Perhitungan waktu proses penghapusan *noise*

3.5 Metode Analisa Data

3.5.1 *Running-time*

Subbab ini menjelaskan metode analisa data yang digunakan untuk menghitung waktu yang dibutuhkan dalam mereduksi *noise* menggunakan metode *gaussian filter* pada komputasi paralel dan komputasi serial. Menghitung waktu proses sangat diperlukan untuk mengetahui efisiensi proses komputasi paralel dan

komputasi serial. Waktu mulai dan waktu akhir bekerja secara bersamaan untuk menghitung total waktu yang diperlukan dan ditampilkan dalam satuan detik. Semakin kecil nilai *running-time* semakin cepat waktu yang digunakan untuk proses, dan semakin besar nilai *running-time* semakin lama waktu yang digunakan untuk proses. Kode program untuk perhitungan waktu komputasi dapat dilihat pada listing 3.20.

3.5.2 Penilaian Kualitas *Image*

Penilaian kualitas gambar digunakan untuk menunjukkan kualitas *image* yang dihasilkan. Menurut Yusra A. Y. Al-Najjar (2012), ada dua metode untuk mengevaluasi kualitas *image*, yaitu evaluasi metode secara subjektif dan objektif. Evaluasi metode secara subjektif telah digunakan bertahun-tahun. Namun, evaluasi metode secara subjektif dianggap mahal, lama dan susah.

Evaluasi metode secara objektif bertujuan untuk mengukur nilai similaritas dan perbedaan error. Menurut Yusra A. Y. Al-Najjar (2012), *Mean Square Error* (MSE) adalah metode pengukuran kualitas gambar secara objektif yang paling banyak digunakan. MSE didapatkan dengan menghitung rata-rata kuadrat nilai error antara citra asli (input) dan citra hasil (output). Secara matematis dapat dirumuskan seperti pada pernyataan 3.1.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \quad \text{..... (3.1)}$$

(Sumber: M. Kudelka JR, 2012)

dimana n merupakan jumlah banyaknya pixel, $(x_i - y_i)^2$ adalah perbedaan nilai error antara citra asli dan sitra hasil. Semakin kecil nilai MSE, maka kualitas citra semakin baik.

BAB IV

HASIL DAN PEMBAHASAN

Bab ini menjelaskan tentang hasil uji coba program yang telah dirancang dan dibuat. Uji coba dilakukan untuk mengetahui apakah program dapat berjalan sesuai dengan skenario implementasi metode *gaussian filter* untuk penghapusan citra bernoise menggunakan komputasi pararel.

4.1. Data Uji

Citra yang digunakan dalam uji coba ini berformat jpg dengan ukuran citra 3264 × 1836. Data yang digunakan untuk uji coba adalah sebanyak 30 citra indoor yang diambil menggunakan kamera *smartphone* Asus Zenfone5. Kemudian data tersebut diberi *noise salt and pepper* 5% sejumlah 30 citra, *noise salt and pepper* 10% sejumlah 30 citra, dan *noise salt and pepper* 50% sejumlah 30 citra. Contoh data citra uji dapat dilihat pada gambar 4.1. Semua data citra uji dapat dilihat pada lampiran.

a b c d

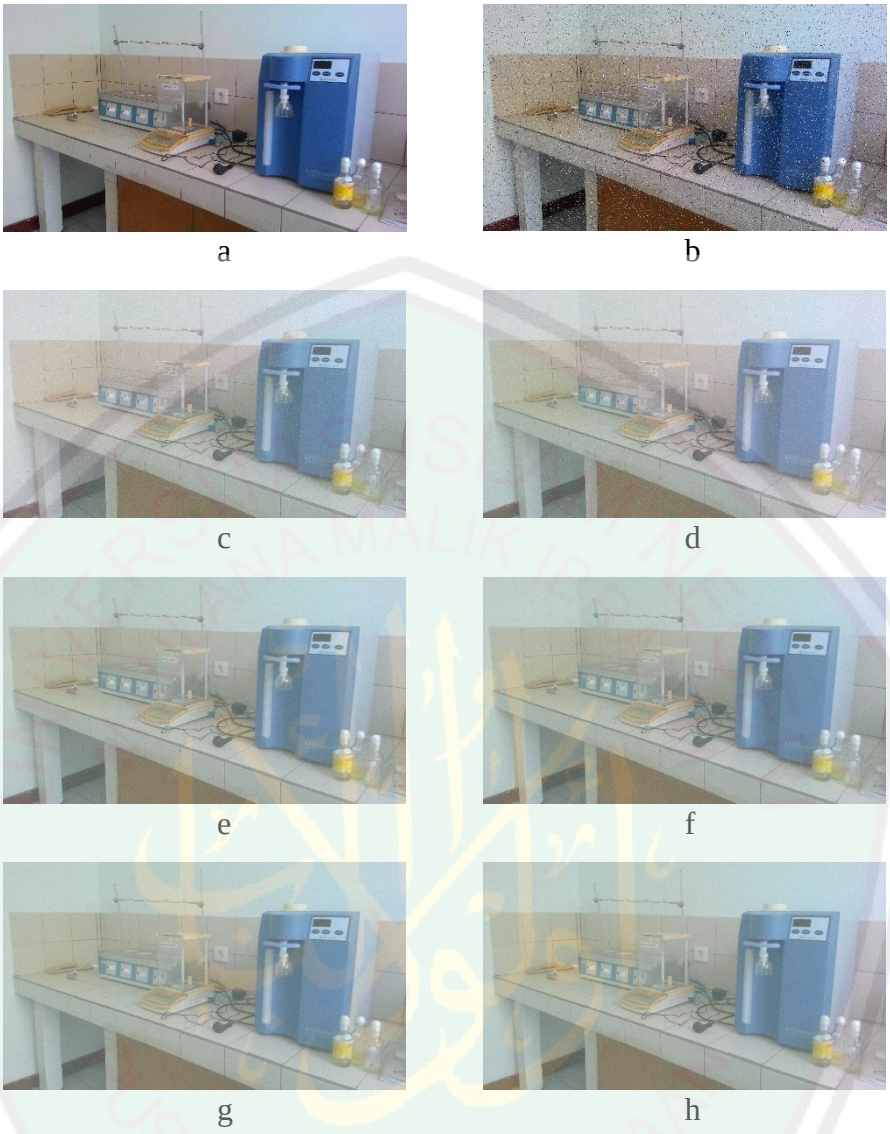
Gambar 4.1 Citra masukan. (a) Citra asli; (b) citra dengan *noise* 5%; (c) citra dengan *noise* 10%; dan (d) citra dengan *noise* 50%.

4.2. Tahapan Proses

Implementasi metode *gaussian filter* yang sudah dilakukan, diuji coba untuk didapatkan data efisiensinya. Data efisiensi didapatkan karena adanya perbandingan antara pemrosesan pada CPU dan GPU. Sehingga pengujian dilakukan pada CPU dan GPU. Pengujian dilakukan pada setiap data uji seperti yang telah dijelaskan pada subbab 4.1. Adapun langkah pengujian implementasi metode *gaussian filter* untuk penghapusan *noise* dapat dilihat pada skenario yang telah dirancang yaitu pada gambar 3.2 dan 3.7.

Alur pengujian untuk proses penghapusan *noise* pada CPU dan GPU tidaklah berbeda. Citra masukan sebanyak 30 dengan penambahan *noise salt and pepper* sebanyak 5%, 10%, dan 50%. Kemudian dilakukan penghapusan *noise* pada setiap citra masukan dengan menggunakan metode *gaussian filter*. Penghausan *noise* dilakukan dengan dua model pemrograman, yaitu pemrograman sekuensial dan pararel. Pemrograman sekuensial diproses dalam CPU. Sedangkan pemrograman pararel diproses dalam CPU (semua proses dilakukan dalam CPU) dan GPU (mengirim data dari CPU ke GPU, penghapusan *noise* di GPU, mengirim hasil ke CPU).

Perhitungan waktu dijalankan pada awal proses pengiriman data untuk proses penghapusan *noise* sampe dengan data dikirim kembali setelah proses penghapusan *noise* untuk dijadikan citra RGB berformat jpg. Data waktu dicatat dalam satuan *milisecond* (ms) .

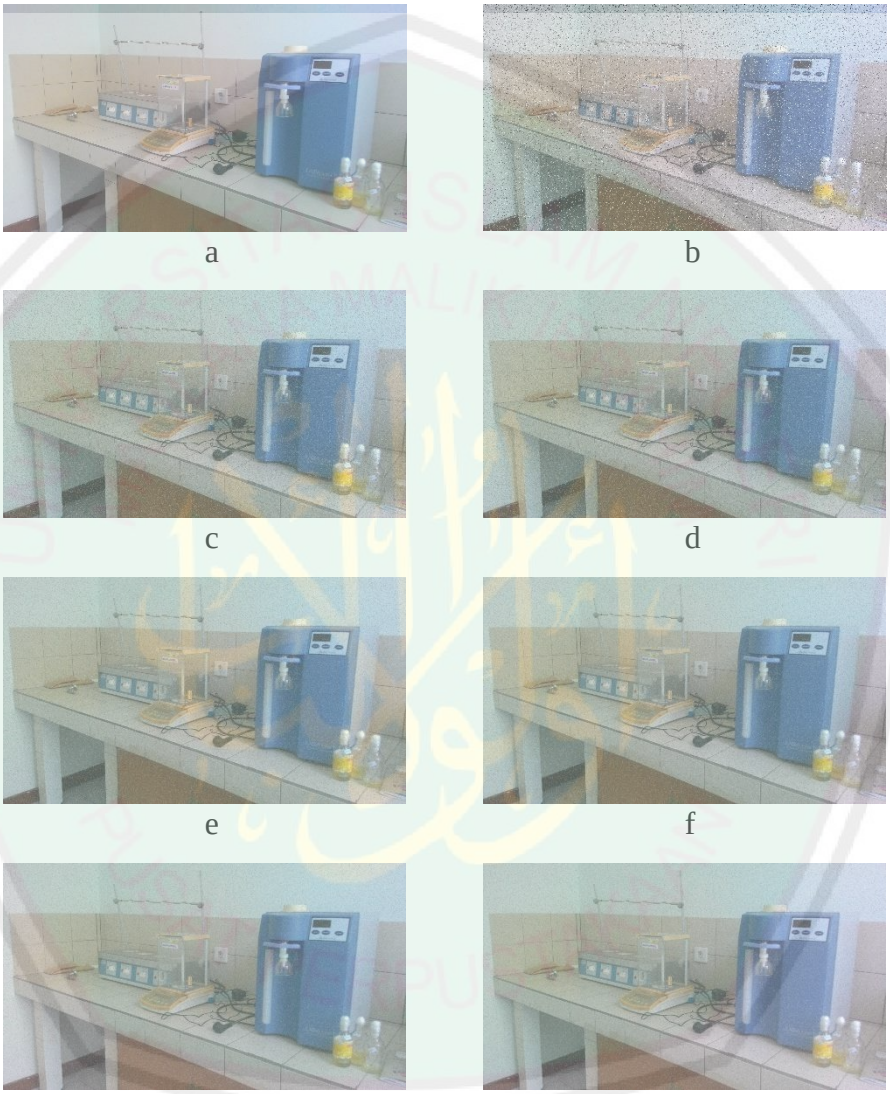


Gambar 4.2 Citra hasil penghapusan *noise* dengan probabilitas 0,05 menggunakan metode *gaussian filter* menggunakan komputasi pararel. (a) Citra asli; (b) citra masukan dengan intensitas *noise* 5%; (c) hasil filter pada CPU dengan kernel 3×3; (d) hasil filter pada GPU dengan kernel 3×3; (e) hasil filter pada CPU dengan kernel 5×5; (f) hasil filter pada GPU dengan kernel 5×5; (g) hasil filter pada CPU dengan kernel 7×7; (h)hasil filter pada GPU dengan kernel 7×7;

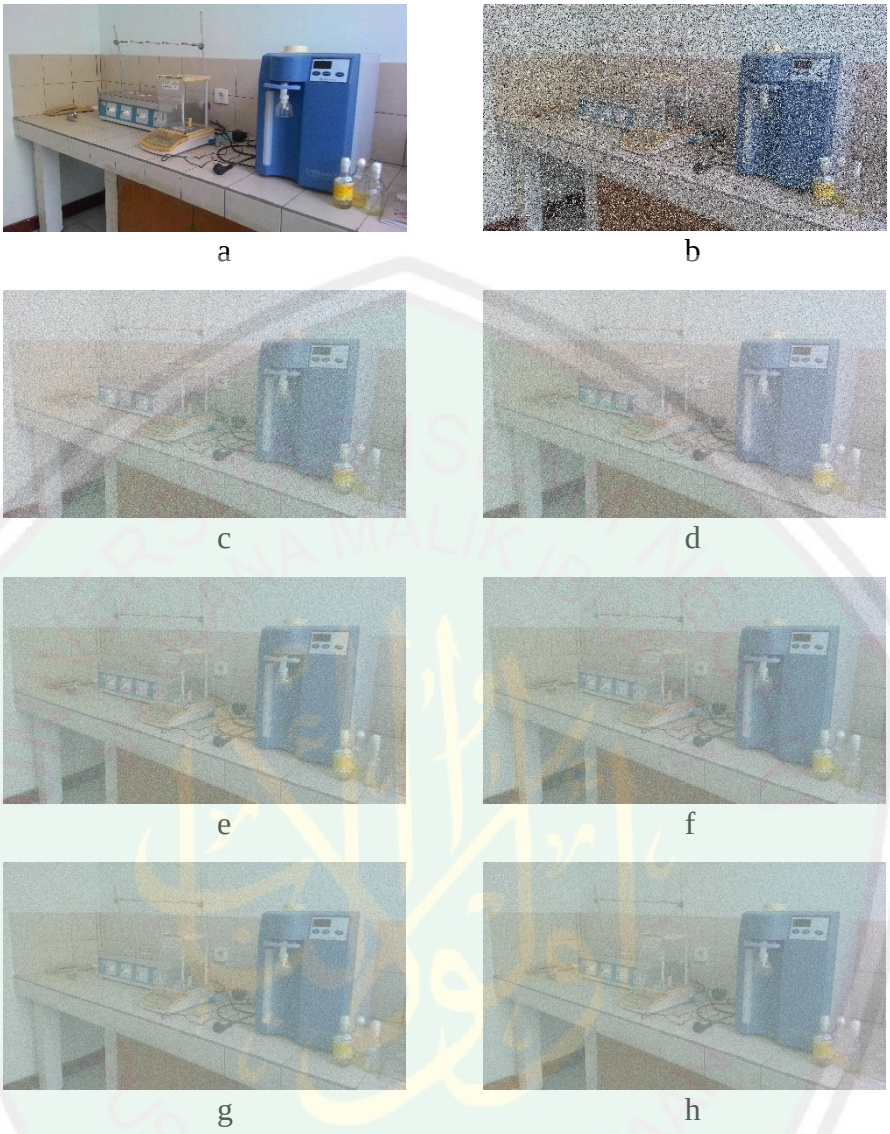
4.3. Hasil Uji Coba

Berdasarkan dari implementasi skenario yang telah dilakukan. Skripsi ini membedakan hasil waktu yang dibutuhkan untuk memproses penghapusan *noise* menggunakan komputasi pararel dengan metode *gaussian filter*. Dengan perbedaan waktu tersebut, maka dapat diketahui perbedaan kecepatan dari total

proses yang dilakukan pada CPU dan GPU. Hasil citra yang telah dilakukan penghapusan *noise* menggunakan metode *gaussian filter* dapat dilihat pada gambar 4.2, 4.3 dan 4.4.



Gambar 4.3 Citra hasil penghapusan *noise* dengan probabilitas 0,1 menggunakan metode *gaussian filter* menggunakan komputasi pararel. (a) Citra asli; (b) citra masukan dengan intensitas *noise* 10%; (c) hasil filter pada CPU dengan kernel 3×3 ; (d) hasil filter pada GPU dengan kernel 3×3; (e) hasil filter pada CPU dengan kernel 5×5; (f) hasil filter pada GPU dengan kernel 5×5; (g) hasil filter pada CPU dengan kernel 7×7; (h)hasil filter pada GPU dengan kernel 7×7;



Gambar 4.4 Citra hasil penghapusan *noise* dengan probabilitas 0,5 menggunakan metode *gaussian filter* menggunakan komputasi pararel. (a) Citra asli; (b) citra masukan dengan intensitas *noise* 50%; (c) hasil filter pada CPU dengan kernel 3×3 ; (d) hasil filter pada GPU dengan kernel 3×3; (e) hasil filter pada CPU dengan kernel 5×5; (f) hasil filter pada GPU dengan kernel 5×5; (g) hasil filter pada CPU dengan kernel 7×7; (h)hasil filter pada GPU dengan kernel 7×7;

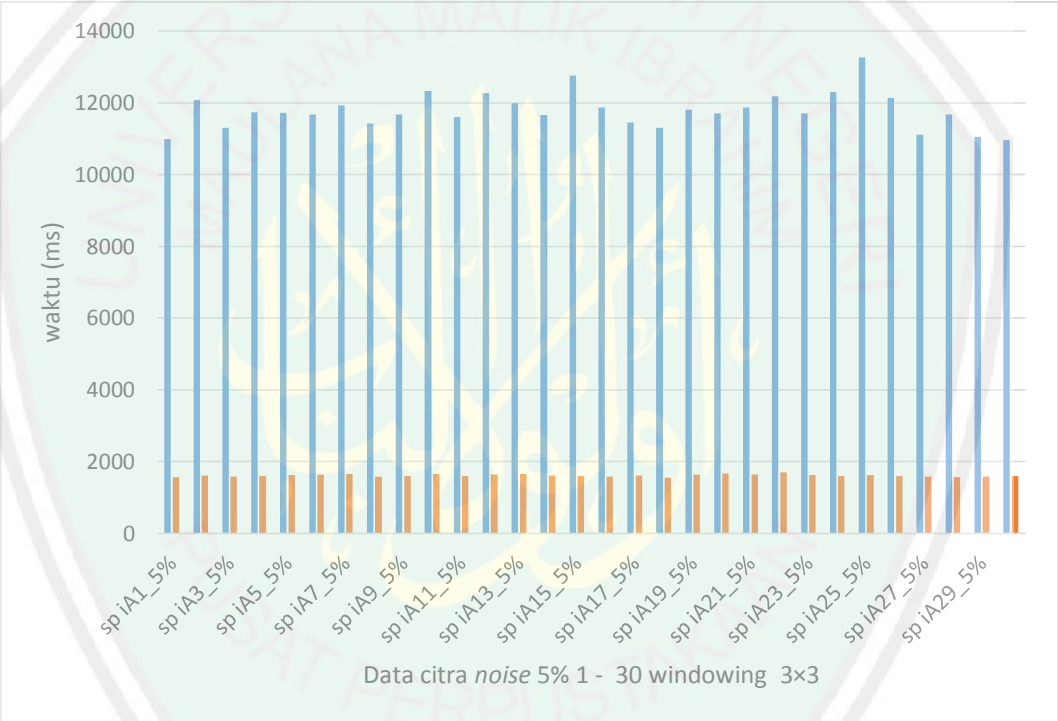
Hasil waktu yang dibutuhkan untuk memproses penghapusan *noise salt and pepper* 5% pada CPU dengan menggunakan metode *gaussian filter* ditunjukkan pada tabel 4.1. Untuk hasil ujicoba data dengan *noise salt and pepper* 10% ditunjukkan pada tabel 4.2. Sedangkan hasil ujicoba data dengan *noise salt and pepper* 50% dapat dilihat pada tabel 4.3.

Tabel 4.1 Hasil kecepatan pada uji coba data dengan *noise salt and pepper* 5%.

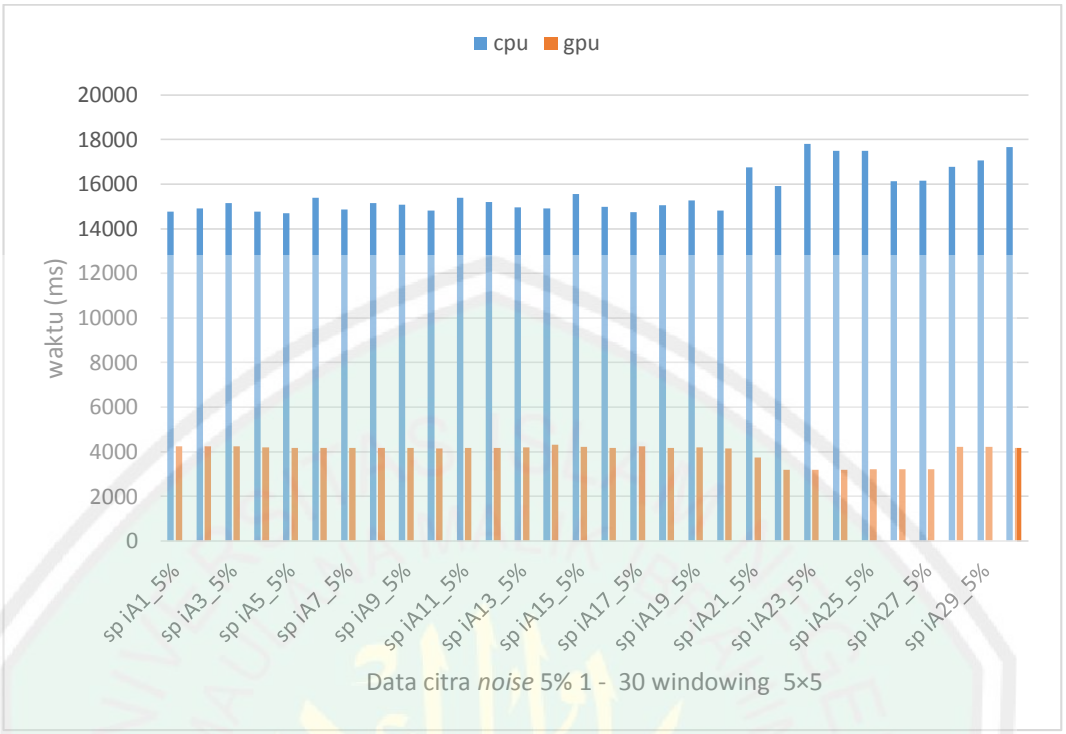
Nama Data	kernel 3		kernel 5		kernel 7	
	CPU	GPU	CPU	GPU	CPU	GPU
sp iA1_5%	9262.6	1199.58	14807.32	3280.72	19370.57	6633.6
sp iA2_5%	9303.07	1192.14	14255.62	3279.42	20206.08	6376.63
sp iA3_5%	9576.28	1239.4	14734.54	3289.53	19456.82	6516.42
sp iA4_5%	9963.34	1220.14	15511.81	3259.19	19646.66	6383.41
sp iA5_5%	9447.76	1225.91	14242.93	3447.54	19504.55	6388.59
sp iA6_5%	11409.34	1595.2	12433.45	3349.34	18949.77	6344.51
sp iA7_5%	11750.85	1592.02	14874.71	3331.47	19554.33	6297.79
sp iA8_5%	11318.06	1571.23	14091.39	3252.18	18279.88	6363.1
sp iA9_5%	11936.53	1626.06	13654.93	3234.38	19967.49	6456.56
sp iA10_5%	11009.77	1653.49	13294.41	3289.41	18892.72	6271.88
sp iA11_5%	11620.25	1626.06	13041.13	3290.85	18812.34	6336.35
sp iA12_5%	11362.12	1569.42	13824.84	3272.85	17357.2	6317.31
sp iA13_5%	11268.09	1648.72	14726.78	3223.64	17695.14	6286.37
sp iA14_5%	11181.93	1584.1	15636.39	3209.04	17004.46	6324.59
sp iA15_5%	11543.97	1625.28	12686.51	3269.08	17270.5	6399.56
sp iA16_5%	11285.01	1566.22	13257.59	3216.96	16764.4	6355.67
sp iA17_5%	12004.07	1574.29	12346.16	3215.24	18648.9	6384.85
sp iA18_5%	11405.24	1645.81	12347.74	3217.18	18909.08	6292.14
sp iA19_5%	11542.82	1709.4	12473.11	3271.82	19062.16	6450.43
sp iA20_5%	11257.46	1634.13	12837.12	3233.28	19899.85	6392.26
sp iA21_5%	11219.02	1601.75	12533.32	3288.09	19865.68	6277.65
sp iA22_5%	11628.78	1632.97	13267.19	3269.14	17385.13	6350.17
sp iA23_5%	11540.31	1570.77	13303.04	3318.44	20015.32	6400.15
sp iA24_5%	11293.57	1576.56	14019	3503.66	19238.65	6472.8
sp iA25_5%	11538.56	1552.91	13897.24	3283.74	6472.8	6383.91
sp iA26_5%	11547.17	1600.78	14176.05	3294.21	20976.81	6277.33
sp iA27_5%	11596.13	1567.91	14990.41	3249.53	19086.26	6302.64
sp iA28_5%	11562.91	1615.25	14536.47	3249.25	20349.77	6387.8
sp iA29_5%	11272.35	1565.64	14368.18	3265.9	20963.91	6358.27
sp iA30_5%	11100.6	1563.89	14272.97	3268.95	21514.87	6407.62

Berdasarkan pada tabel 4.1 dapat dilihat bahwa hasil ujicoba data dengan *noise salt and pepper* sebesar 5% yang dijalankan pada CPU-GPU mengalami kenaikan kecepatan waktu sebesar 86,17% dengan perbesaran kernel 3×3. Prosentase ini diambil dari selisih rata-rata waktu yang dibutuhkan untuk memproses data dengan perbesaran kernel 3×3.

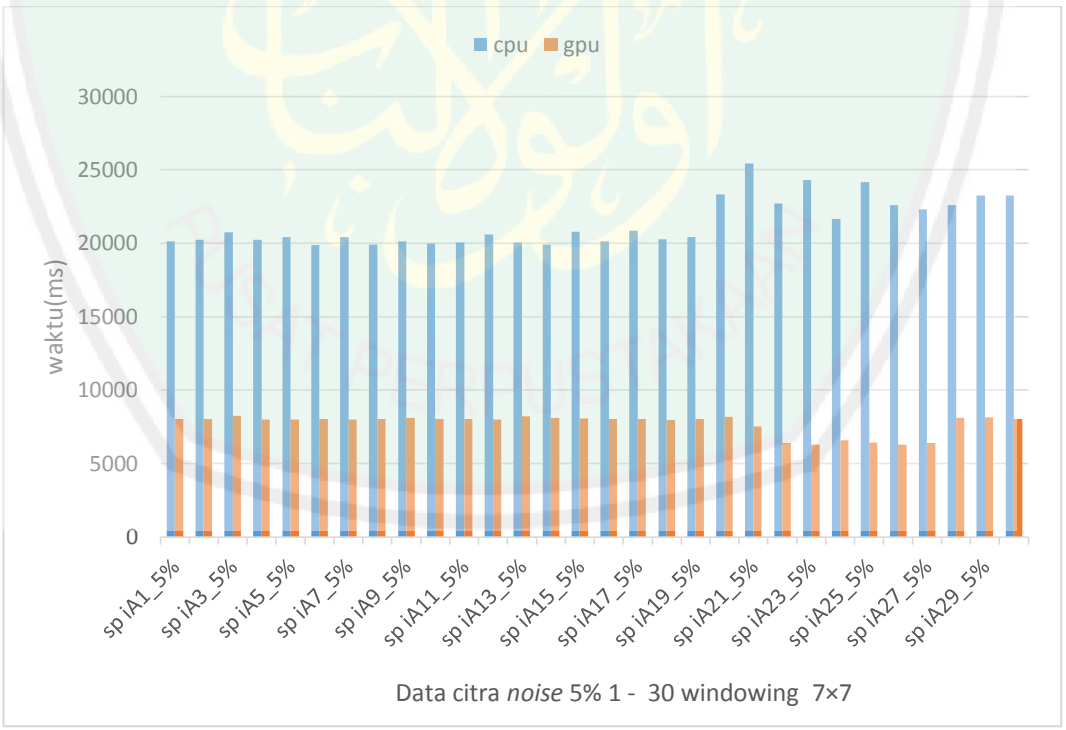
Untuk hasil ujicoba data dengan *noise salt and pepper* sebesar 5% dengan perbesaran kernel 5×5 yang dijalankan pada CPU-GPU mengalami kenaikan kecepatan waktu sebesar 76,25%. Sedangkan untuk hasil ujicoba data dengan *noise salt and pepper* sebesar 5% dengan perbesaran kernel 7×7 yang dijalankan pada CPU-GPU mengalami kenaikan kecepatan waktu sebesar 65,93%.



Gambar 4.5 Grafik waktu pemrosesan penghapusan *noise* 5% dengan perbesaran windowing 3×3.



Gambar 4.6 Grafik waktu pemrosesan penghapusan *noise* 5% dengan perbesaran windowing 5×5



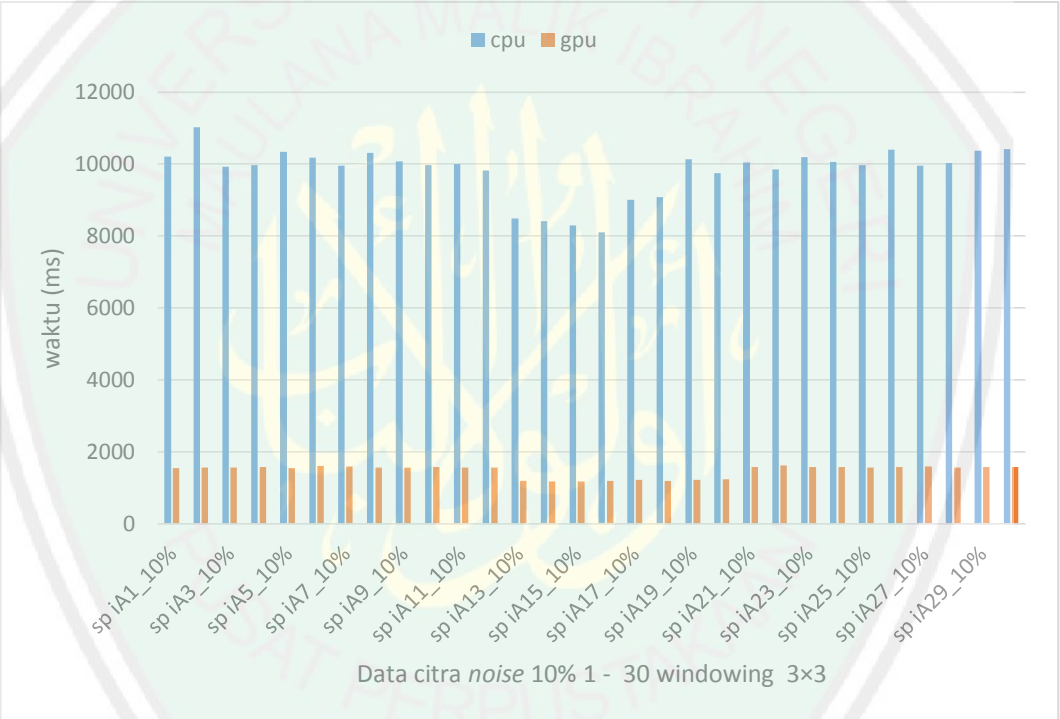
Gambar 4.7 Grafik waktu pemrosesan penghapusan *noise* 5% dengan perbesaran windowing 7×7

Tabel 4.2 Hasil kecepatan pada uji coba data dengan *noise salt and pepper* 10%.

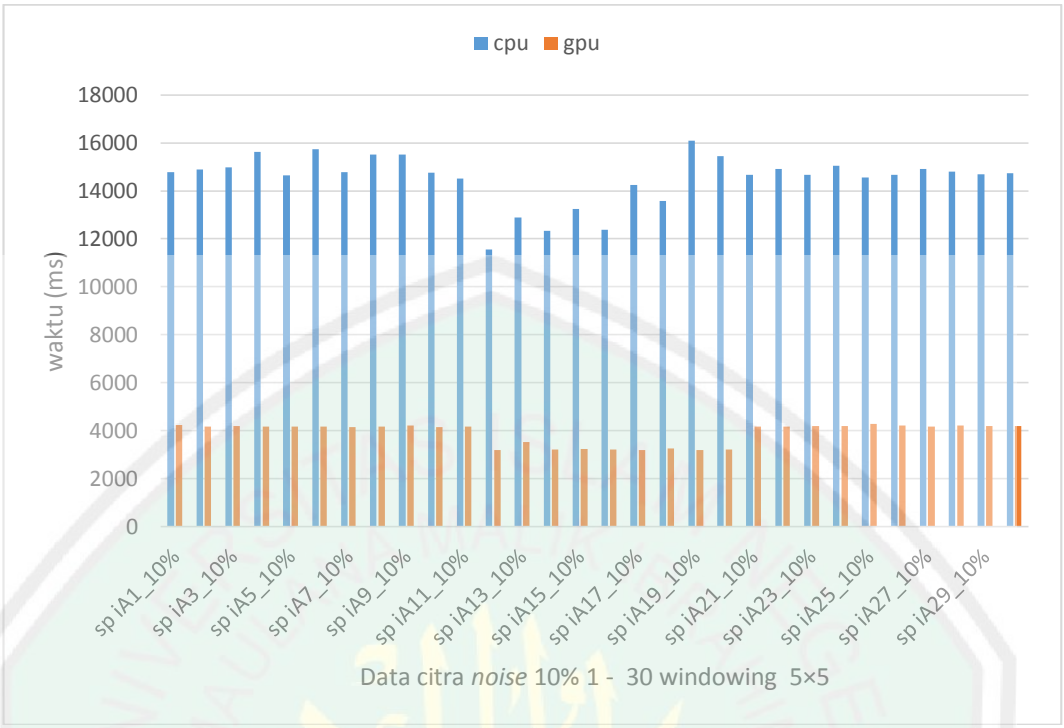
Nama Data	kernel 3		kernel 5		kernel 7	
	CPU	GPU	CPU	GPU	CPU	GPU
sp iA1_10%	9305.81	1192.18	13825.52	3254.05	22297.84	6684.57
sp iA2_10%	9242.19	1238.4	14231.93	3325.39	21799.01	6529.56
sp iA3_10%	9075.44	1221.27	14262.81	3253.91	20011.68	6476.27
sp iA4_10%	9345.86	1214.11	14390.04	3250.17	20636.5	6377.11
sp iA5_10%	9239.84	1196.12	14601.49	3259.84	19858.44	6375.25
sp iA6_10%	11113.3	1570.11	17274.82	4154.8	22524.9	7982.84
sp iA7_10%	11268.48	1605.7	18146.44	4298.09	22542.1	7898
sp iA8_10%	10888.52	1557.64	16524.59	4195.04	22347.06	8048.74
sp iA9_10%	11059.76	1555.59	16623.93	4272.31	22782.9	8133.34
sp iA10_10%	11177.46	1550.84	16726.42	4239.89	22464.75	8116
sp iA11_10%	11320.74	1555.07	16336.02	4176.73	22139.79	8125.49
sp iA12_10%	11179.15	1554.67	16749.15	4182.38	22721.91	8033.14
sp iA13_10%	10929.4	1554.48	16708.54	4180.27	22222.58	8050.52
sp iA14_10%	11080.81	1561.55	16527.06	4270.32	22524.45	8095.26
sp iA15_10%	11053.48	1586.71	16556.01	4165.35	22374.21	8143.16
sp iA16_10%	10949.82	1554.49	16636.42	4237.52	22208	8186.74
sp iA17_10%	11390.99	1539.82	16565.8	4233.95	22319.63	8120.49
sp iA18_10%	11436.01	1563.6	16969.23	4170.99	22386.02	8108.08
sp iA19_10%	11178.89	1574.95	17618.2	4163.92	22956.84	8065.64
sp iA20_10%	11390.7	1570.07	17884.93	4195.31	22599.84	8104.03
sp iA21_10%	11771.23	1562.43	16879.35	4224.98	23072.27	8119.12
sp iA22_10%	12031.97	1565.06	16954.74	4233.95	22319.35	8120.76
sp iA23_10%	11270.12	1560.85	16613.97	4271.84	22126.83	7975.91
sp iA24_10%	11780.2	1566.96	16419.68	4283.53	22408.24	8072.21
sp iA25_10%	11362.53	1589.98	16357.18	4323.81	22248.29	8041.76
sp iA26_10%	10957.57	1594.88	16435.17	4269.43	22207.8	8112.59
sp iA27_10%	11656.68	1602.67	17044.21	4169.87	22487.19	7971.66
sp iA28_10%	11107.28	1602.38	16972.48	4165.51	22308.89	8017.21
sp iA29_10%	11682.16	1612.21	16565.26	4253.27	22202.98	8031.01
sp iA30_10%	11020.8	1573.58	16270.18	4187.44	22176.02	7979.04

Berdasarkan pada tabel 4.2 dapat dilihat bahwa hasil ujicoba data dengan *noise salt and pepper* sebesar 10% yang dijalankan pada CPU-GPU mengalami kenaikan kecepatan waktu sebesar 86,18% dengan perbesaran kernel 3×3. Prosentase ini diambil dari selisih rata-rata waktu yang dibutuhkan untuk memproses data dengan perbesaran kernel 3×3.

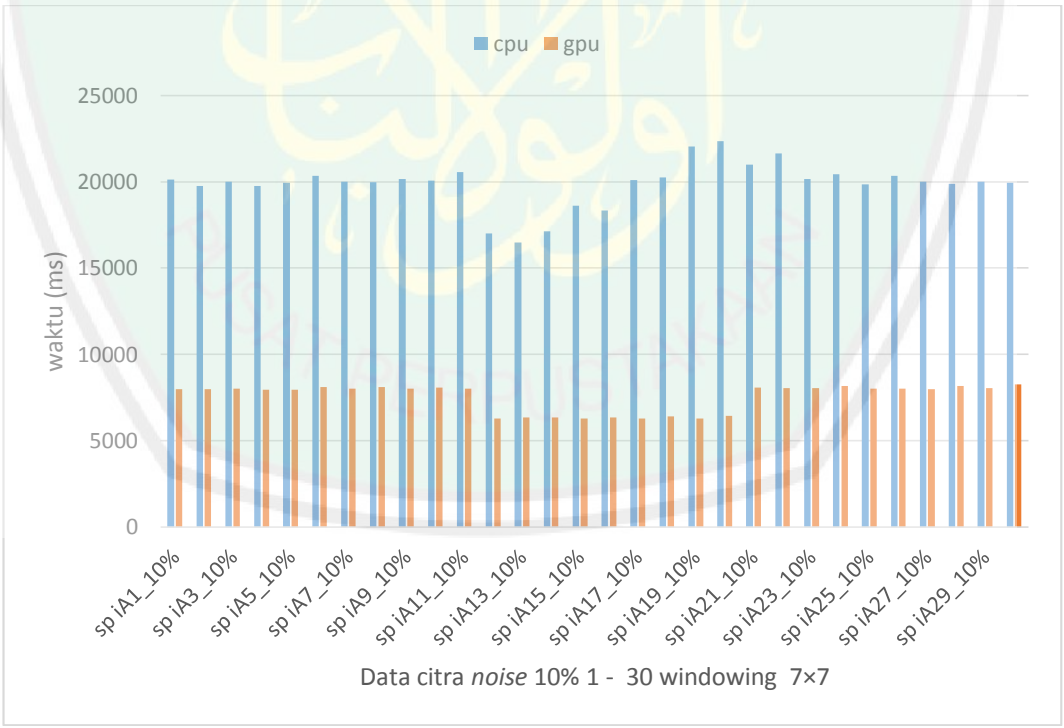
Untuk hasil ujicoba data dengan *noise salt and pepper* sebesar 10% dengan perbesaran kernel 5×5 yang dijalankan pada CPU-GPU mengalami kenaikan kecepatan waktu sebesar 75,21%. Sedangkan untuk hasil ujicoba data dengan *noise salt and pepper* sebesar 10% dengan perbesaran kernel 7×7 yang dijalankan pada CPU-GPU mengalami kenaikan kecepatan waktu sebesar 64,81%.



Gambar 4.8 Grafik waktu pemrosesan penghapusan *noise* 10% dengan perbesaran windowing 3×3



Gambar 4.9 Grafik waktu pemrosesan penghapusan *noise* 10% dengan perbesaran windowing 5×5

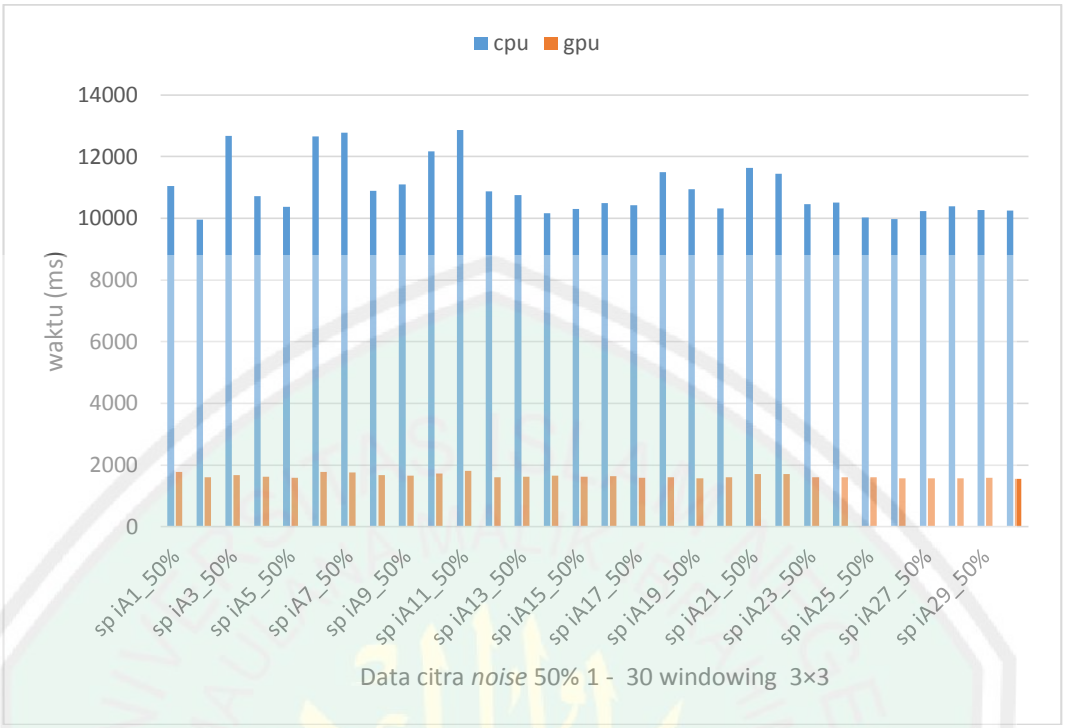


Gambar 4.10 Grafik waktu pemrosesan penghapusan *noise* 10% dengan perbesaran windowing 7×7

Tabel 4.3 Hasil kecepatan pada uji coba data dengan *noise salt and pepper* 50%.

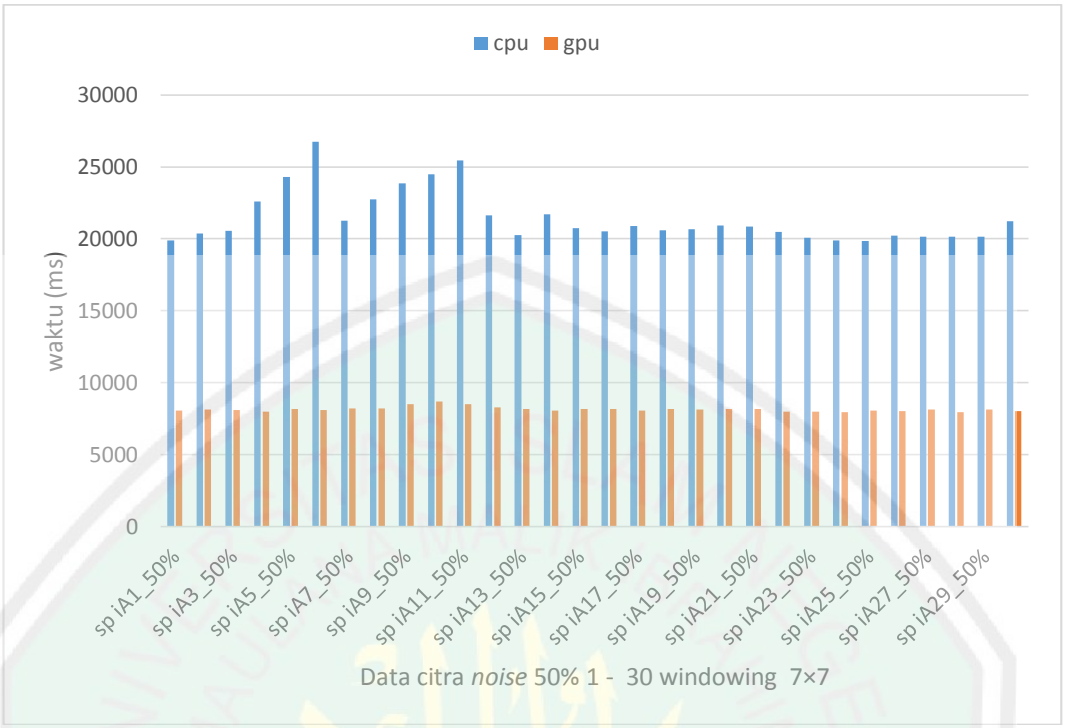
Nama Data	kernel 3		kernel 5		kernel 7	
	CPU	GPU	CPU	GPU	CPU	GPU
sp iA1_50%	9164.51	1232.24	13895.51	3255.71	22236.74	6487.89
sp iA2_50%	9129.68	1195.87	14047.01	3297.13	22884.21	6438.45
sp iA3_50%	9468.56	1223.64	13845.13	3261.84	21803.51	6382.67
sp iA4_50%	9531.15	1253.88	14067.38	3263.42	21548.39	6364.04
sp iA5_50%	9244.57	1247.21	14604.61	3294.83	20445.24	6358.15
sp iA6_50%	8383.43	1191.86	12329.93	3215.38	19884.65	8072.68
sp iA7_50%	8266.65	1194.11	14858.11	4203.93	20851.73	8529.11
sp iA8_50%	8216.36	1189.55	15320.63	4210.03	20528.36	8174.53
sp iA9_50%	8204.87	1234.13	15092.65	4309.85	19914.32	8131.88
sp iA10_50%	8131.38	1195.05	14990.63	4195.17	20350.21	8131.75
sp iA11_50%	8264.51	1190.93	15083.06	4197.61	19875.8	8135.07
sp iA12_50%	8102.59	1190.25	15288.34	4193.56	20182.49	8221.92
sp iA13_50%	8197.8	1240.28	15159.92	4297.54	20553.01	8131.08
sp iA14_50%	8577.1	1188.75	15026.95	4251.02	20517.92	8070.17
sp iA15_50%	8262.79	1234.31	14934.67	4302.64	19848.71	8126.52
sp iA16_50%	8285.18	1221.03	15216.77	4274.74	19858.67	8090.89
sp iA17_50%	8588.95	1206.38	15015.48	4259.6	20335.96	8098.15
sp iA18_50%	8567.14	1188.8	15011.68	4324.68	20184.17	8071.43
sp iA19_50%	8304.21	1191.78	15269.29	4342.07	20477.14	8294.73
sp iA20_50%	8298.27	1220.58	15097.61	4257.68	19976.13	8126.92
sp iA21_50%	8992.39	1254.13	14969.31	4242.39	21925.45	8162.5
sp iA22_50%	8382.96	1212.63	15153.56	4260.5	19775.83	8154.3
sp iA23_50%	8591.35	1207.27	15346.8	4262.04	19844.26	8100.5
sp iA24_50%	8297.79	1220.46	14699.63	4293.4	19915.7	8119.13
sp iA25_50%	8737.82	1221.54	15211.91	4280.33	20204.26	8021.97
sp iA26_50%	8647.01	1215.38	15250.58	4262.69	20439.56	8157.07
sp iA27_50%	8266.98	1191.01	15163.97	4190.67	19853.41	8074.42
sp iA28_50%	8350.81	1190.89	15181.18	4405.49	19843.52	8158.08
sp iA29_50%	8294.93	1228.53	15041.52	4382.07	20007.44	8153.48
sp iA30_50%	8825.1	1232.24	14974.31	4303.9	20301.89	8185.48

Berdasarkan pada tabel 4.3 dapat dilihat bahwa hasil ujicoba data dengan *noise salt and pepper* sebesar 50% yang dijalankan pada CPU-GPU mengalami kenaikan kecepatan waktu sebesar 85,81% dengan perbesaran kernel 3×3. Prosentase ini diambil dari selisih rata-rata waktu yang dibutuhkan untuk memproses data dengan perbesaran kernel 3×3.



Gambar 4.11 Grafik waktu pemrosesan penghapusan *noise* 50% dengan perbesaran windowing 3×3

Gambar 4.12 Grafik waktu pemrosesan penghapusan *noise* 50% dengan perbesaran windowing 5×5



Gambar 4.13 Grafik waktu pemrosesan penghapusan *noise* 50% dengan perbesaran windowing 7×7

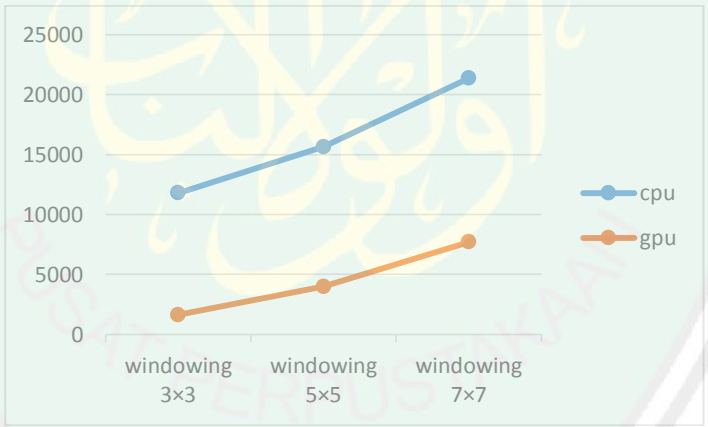
Untuk hasil ujicoba data dengan *noise salt and pepper* sebesar 50% dengan perbesaran kernel 5×5 yang dijalankan pada CPU-GPU mengalami kenaikan kecepatan waktu sebesar 72,57%. Sedangkan untuk hasil ujicoba data dengan *noise salt and pepper* sebesar 50% dengan perbesaran kernel 7×7 yang dijalankan pada CPU-GPU mengalami kenaikan kecepatan waktu sebesar 61,63%.

Tabel 4.4 Prosentase rata-rata waktu penghapusan *noise*

Noise	kernel 3×3	kernel 5×5	kernel 7×7
Salt & pepper 5%	86.17 %	76.25 %	65.93 %
Salt & pepper 10%	86.18 %	75.21 %	64.81 %
Salt & pepper 50%	85.81 %	72.57 %	61.63 %

4.4. Pembahasan

Hasil waktu proses ujicoba data pada CPU dan CPU-GPU memang sangat berbeda. Proses penghapusan *noise* yang dijalankan pada CPU memang membutuhkan waktu yang lama dibandingkan dengan proses penghapusan *noise* yang dijalankan pada CPU-GPU. Seperti yang telah ditampilkan pada tabel 4.5, hasil rata-rata waktu yang dibutuhkan untuk penghapusan *noise salt and pepper* sebesar 5% yang ditampilkan pada gambar 4.14. Rata-rata waktu proses yang diperoleh dari hasil ujicoba data penghapusan *noise* dengan perbesaran kernel 3×3 yang dijalankan pada CPU sebesar 11.124,93ms. Sedangkan rata-rata waktu proses pada CPU-GPU sebesar 1.538,23ms. Sehingga diperoleh selisih rata-rata waktu proses sebesar 9.586,7ms. Dari hasil ini diketahui bahwa waktu proses pada CPU-GPU 7 kali lebih cepat daripada pemrosesan data pada CPU.

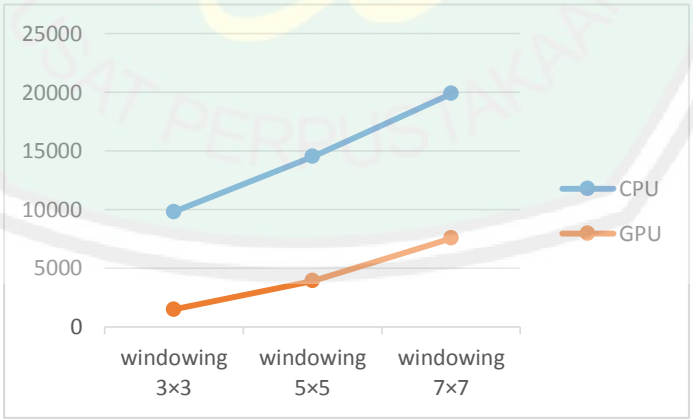


Gambar 4.14 Grafik rata-rata waktu yang dibutuhkan untuk proses penghapusan *noise salt and pepper* sebesar 5%

Rata-rata waktu proses yang diperoleh dari hasil ujicoba data penghapusan *noise* dengan perbesaran kernel 5×5 yang dijalankan pada CPU sebesar 13.814,75ms. Sedangkan rata-rata waktu proses pada CPU-GPU sebesar 3.280,8ms. Sehingga diperoleh selisih rata-rata waktu proses sebesar 10.533,95ms. Dari hasil ini diketahui bahwa waktu proses pada CPU-GPU 4 kali lebih cepat daripada pemrosesan data pada CPU.

Sedangkan rata-rata waktu proses yang diperoleh dari hasil ujicoba data penghapusan *noise* dengan perbesaran kernel 7×7 yang dijalankan pada CPU sebesar 18.704,07ms. Sedangkan rata-rata waktu proses pada CPU-GPU sebesar 6.373,01ms. Sehingga diperoleh selisih rata-rata waktu proses sebesar 12.331,06ms. Dari hasil ini diketahui bahwa waktu proses pada CPU-GPU 3 kali lebih cepat daripada pemrosesan data pada CPU.

Seperti yang telah ditampilkan pada tabel 4.4, hasil rata-rata waktu yang dibutuhkan untuk penghapusan *noise salt and pepper* sebesar 10% yang ditampilkan pada gambar 4.15. Rata-rata waktu proses yang diperoleh dari hasil ujicoba data penghapusan *noise* dengan perbesaran kernel 3×3 yang dijalankan pada CPU sebesar 10.942,24ms. Sedangkan rata-rata waktu proses pada CPU-GPU sebesar 1.511,61ms. Sehingga diperoleh selisih rata-rata waktu proses sebesar 9.430,63ms. Dari hasil ini diketahui bahwa waktu proses pada CPU-GPU 7 kali lebih cepat daripada pemrosesan data pada CPU.

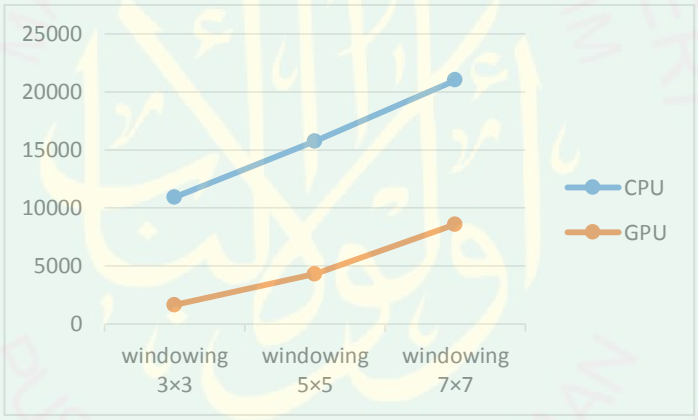


Gambar 4.15 Grafik rata-rata waktu yang dibutuhkan untuk proses penghapusan *noise salt and pepper* sebesar 10%

Rata-rata waktu proses yang diperoleh dari hasil ujicoba data penghapusan *noise* dengan perbesaran kernel 5×5 yang dijalankan pada CPU sebesar 16.389,05ms. Sedangkan rata-rata waktu proses pada CPU-GPU sebesar

4.062,13ms. Sehingga diperoleh selisih rata-rata waktu proses sebesar 12.326,92ms. Dari hasil ini diketahui bahwa waktu proses pada CPU-GPU 4 kali lebih cepat daripada pemrosesan data pada CPU.

Sedangkan rata-rata waktu proses yang diperoleh dari hasil ujicoba data penghapusan *noise* dengan perbesaran kernel 7×7 yang dijalankan pada CPU sebesar 22.175,88ms. Sedangkan rata-rata waktu proses pada CPU-GPU sebesar 7.803,18ms. Sehingga diperoleh selisih rata-rata waktu proses sebesar 14.372,7ms. Dari hasil ini diketahui bahwa waktu proses pada CPU-GPU 3 kali lebih cepat daripada pemrosesan data pada CPU.



Gambar 4.16 Grafik rata-rata waktu yang dibutuhkan untuk proses penghapusan *noise salt and pepper* sebesar 50%

Untuk hasil rata-rata waktu yang dibutuhkan untuk penghapusan *noise salt and pepper* sebesar 50% ditunjukkan pada gambar 4.16. Rata-rata waktu proses penghapusan *noise* dengan perbesaran kernel 3×3 yang diperoleh dari hasil ujicoba data yang dijalankan pada CPU sebesar 8.552,56ms. Sedangkan rata-rata waktu proses pada CPU-GPU sebesar 1.213,49ms. Sehingga diperoleh selisih rata-rata waktu proses sebesar 7.339,07ms. Dari hasil ini diketahui bahwa waktu proses pada CPU-GPU 7 kali lebih cepat daripada pemrosesan data pada CPU.

Rata-rata waktu proses yang diperoleh dari hasil ujicoba data penghapusan *noise* dengan perbesaran kernel 5×5 yang dijalankan pada CPU sebesar 14.838,27ms. Sedangkan rata-rata waktu proses pada CPU-GPU sebesar 4.069,73ms. Sehingga diperoleh selisih rata-rata waktu proses sebesar 10.768,54ms. Dari hasil ini diketahui bahwa waktu proses pada CPU-GPU 4 kali lebih cepat daripada pemrosesan data pada CPU.

Sedangkan rata-rata waktu proses yang diperoleh dari hasil ujicoba data penghapusan *noise* dengan perbesaran kernel 7×7 yang dijalankan pada CPU sebesar 20.478,96ms. Sedangkan rata-rata waktu proses pada CPU-GPU sebesar 7.857,5ms. Sehingga diperoleh selisih rata-rata waktu proses sebesar 12.621,46ms. Dari hasil ini diketahui bahwa waktu proses pada CPU-GPU 2 kali lebih cepat daripada pemrosesan data pada CPU.

Tabel 4.5 Hasil rata-rata waktu penghapusan *noise*

Noise	kernel 3		kernel 5		kernel 7	
	cpu	gpu	cpu	gpu	cpu	gpu
Salt & pepper 5%	11124.93	1538.234	13814.75	3280.801	18704.07	6373.012
Salt & pepper 10%	10942.24	1511.612	16389.05	4062.129	22175.88	7803.183
Salt & pepper 50%	8552.561	1213.49	14838.27	4069.73	20478.96	7857.499

Semakin tinggi angka yang dihasilkan, maka semakin lama waktu yang dibutuhkan untuk proses penghapusan *noise*. Begitu pula sebaliknya, semakin kecil angka yang dihasilkan maka semakin cepat waktu yang dibutuhkan untuk proses penghapusan *noise*. Pada tabel 4.4 ditunjukkan hasil prosentase dari skala hasil kecepatan rata-rata pada setiap perbesaran *windowing*. Pada tabel tersebut

ditunjukkan bahwa skala terbesar dimiliki oleh data dengan intensitas *noise* 5% dan dengan perbesaran *windowing* 3×3. Sehingga dapat disimpulkan bahwa untuk perbesaran kernel terbesar, implementasi metode *gaussian filter* menggunakan komputasi pararel tidak bekerja begitu cepat dibandingkan dengan perbesaran kernel terkecil. Semakin kecil pebesaran kernel yang dilakukan, maka semakin cepat proses komputasi pada GPU.

Setelah mendapatkan hasil pemfilteran citra bernoise, maka dilakukan penghitungan nilai *Mean Square* error (MSE) untuk mengukur kualitas citra yang dihasilkan dengan menghitung rata-rata kuadrat perbedaan nilai error antara citra asli dengan citra hasil. Sehingga dapat diketahui intensitas *noise* yang sudah dihapus. Secara matematis dapat dirumuskan seperti pada pernyataan 3.1.

Semakin kecil nilai MSE berarti semakin baik kualitas citra yang dihasilkan, sebaliknya jika nilai MSE lebih besar berarti kualitas citra yang dihasilkan kurang baik. Perhitungan nilai MSE dilakukan pada setiap citra yang telah diberi *noise* dengan intensitas *noise* 5%, 10%, dan 50%. Dan pada setiap data dengan perbesaran kernel 3×3, 5×5, dan 7×7.

Nilai MSE dapat dilihat pada tabel 4.6, 4.7, dan 4.8. Berdasarkan pada tabel tersebut maka diperoleh rata-rata nilai MSE yang ditunjukkan pada tabel 4.9. Nilai MSE untuk citra yang di proses menggunakan GPU lebih kecil daripada CPU. Pada perbesaran kernel 3×3 untuk citra yang berintensitas *noise* 5% memiliki selisih nilai MSE sebesar 4,77, untuk citra yang berintensitas *noise* 10% memiliki selisih nilai MSE sebesar 4,43, dan untuk citra yang berintensitas *noise* 50% memiliki selisih nilai MSE sebesar 22,2.

Tabel 4.6 Nilai MSE untuk citra berintensitas *noise* sebesar 5%

Nama	Image ber-Noise	windowing 3		windowing 5		windowing 7	
		cpu	gpu	cpu	gpu	cpu	gpu
sp iA1_5%	2603.71	329.44	323.59	224.60	213.56	258.59	239.69
sp iA2_5%	2712.13	341.29	338.52	251.81	247.76	315.29	309.48
sp iA3_5%	2922.88	410.72	406.25	326.05	320.16	398.70	390.74
sp iA4_5%	2770.33	375.60	367.58	278.70	263.87	315.89	290.89
sp iA5_5%	2632.82	354.68	348.58	265.91	255.27	315.96	298.59
sp iA6_5%	2701.69	323.43	318.72	199.08	190.59	216.84	202.19
sp iA7_5%	3332.84	434.23	433.04	274.71	275.83	302.31	308.05
sp iA8_5%	2986.04	393.96	393.65	264.99	262.55	281.96	273.65
sp iA9_5%	2593.16	315.32	310.04	207.76	199.53	233.85	221.38
sp iA10_5%	3765.66	500.86	496.20	306.00	300.32	303.98	296.49
sp iA11_5%	2650.85	328.03	321.74	200.68	190.69	208.38	192.30
sp iA12_5%	2646.54	302.84	298.44	167.09	160.83	160.05	151.53
sp iA13_5%	3479.90	458.86	452.91	277.70	269.72	278.51	267.01
sp iA14_5%	2998.84	404.08	397.45	252.23	241.88	252.65	237.57
sp iA15_5%	3611.21	518.39	512.16	366.94	359.39	391.21	380.66
sp iA16_5%	2815.52	328.97	324.49	184.75	177.28	177.55	165.16
sp iA17_5%	2917.83	370.30	364.08	242.31	231.81	260.67	243.65
sp iA18_5%	2675.50	368.37	364.16	316.64	310.19	416.87	406.99
sp iA19_5%	2814.59	350.20	343.43	224.87	212.84	250.00	229.65
sp iA20_5%	3128.94	430.38	425.89	346.05	338.77	466.44	454.78
sp iA21_5%	2701.75	330.56	325.16	219.66	212.10	243.53	232.68
sp iA22_5%	2668.10	326.32	321.72	202.25	196.13	212.87	205.68
sp iA23_5%	2948.16	399.75	394.52	265.56	258.31	305.10	293.78
sp iA24_5%	3274.50	442.10	438.79	292.71	285.96	301.98	287.51
sp iA25_5%	3001.26	364.88	361.18	211.10	207.14	213.86	209.90
sp iA26_5%	2975.36	384.18	382.15	252.50	250.56	278.06	275.86
sp iA27_5%	3106.00	399.49	397.73	245.96	246.96	249.75	256.09
sp iA28_5%	3750.63	487.66	479.23	291.38	280.36	287.54	271.30
sp iA29_5%	3127.80	385.69	379.52	230.19	220.44	234.35	218.93
sp iA30_5%	3019.20	371.75	368.29	233.05	227.82	262.81	254.48

Untuk perbesaran kernel 5×5 dan 7×7 tidak jauh berbeda dengan perbesaran kernel 3×3. Pada perbesaran kernel 5×5 untuk citra yang berintensitas *noise* 5% memiliki selisih nilai MSE sebesar 7,15, untuk citra yang berintensitas *noise* 10% memiliki selisih nilai MSE sebesar 6,41, dan untuk citra yang berintensitas *noise* 50% memiliki selisih nilai MSE sebesar 28,8.

Tabel 4.7 Nilai MSE untuk citra berintensitas *noise* sebesar 10%

Nama	Image ber-Noise	windowing 3		windowing 5		windowing 7	
		cpu	gpu	cpu	gpu	cpu	gpu
sp iA1_10%	5032.55	662.76	659.11	392.33	384.48	390.13	376.38
sp iA2_10%	5083.27	657.41	656.16	404.93	403.10	432.24	430.29
sp iA3_10%	5331.26	759.69	755.15	511.08	505.44	545.41	538.07
sp iA4_10%	5199.33	703.30	696.49	440.83	427.66	441.66	419.40
sp iA5_10%	5092.69	698.54	694.59	444.25	436.74	457.78	444.99
sp iA6_10%	5006.09	615.98	612.03	334.64	327.63	318.70	306.65
sp iA7_10%	6101.50	856.87	856.03	512.60	514.55	500.44	507.29
sp iA8_10%	5607.74	764.90	766.21	459.13	458.88	438.20	433.28
sp iA9_10%	4838.37	596.68	592.41	334.51	328.29	326.80	317.42
sp iA10_10%	6542.40	945.59	938.70	565.16	556.95	522.93	511.58
sp iA11_10%	4949.38	644.00	638.75	359.51	351.24	332.93	319.65
sp iA12_10%	5055.33	607.46	604.82	307.19	303.96	265.51	261.84
sp iA13_10%	5955.77	845.29	836.67	495.27	484.21	459.39	443.27
sp iA14_10%	5378.66	787.40	780.38	475.65	464.98	440.99	425.53
sp iA15_10%	6323.81	944.34	936.10	610.27	600.33	594.71	580.62
sp iA16_10%	5150.01	633.41	629.54	329.76	323.11	288.52	277.49
sp iA17_10%	5450.70	713.05	707.75	412.96	404.04	394.38	379.91
sp iA18_10%	5029.92	685.00	682.05	472.81	468.21	534.92	527.99
sp iA19_10%	5376.01	699.04	694.07	401.51	392.25	389.66	373.60
sp iA20_10%	5745.17	823.78	819.54	565.00	558.26	645.77	635.02
sp iA21_10%	4970.57	611.78	607.22	343.66	337.65	332.95	324.49
sp iA22_10%	5093.97	658.26	655.24	369.64	365.92	344.44	341.30
sp iA23_10%	5208.28	758.08	751.83	470.46	461.67	476.42	462.81
sp iA24_10%	6058.08	855.57	852.82	520.45	514.54	489.62	476.78
sp iA25_10%	5522.17	730.94	727.57	406.42	403.19	372.41	370.11
sp iA26_10%	5601.44	759.93	759.49	451.66	452.06	438.98	440.53
sp iA27_10%	5561.42	772.66	770.45	454.69	455.43	422.94	429.14
sp iA28_10%	6321.08	882.70	870.42	514.06	498.39	473.08	449.85
sp iA29_10%	5550.63	718.55	711.39	398.32	387.75	367.20	350.69
sp iA30_10%	5502.85	714.63	711.75	408.33	403.87	401.92	394.76

Pada perbesaran kernel 7×7 untuk citra yang berintensitas *noise* 5% memiliki selisih nilai MSE sebesar 10,96, untuk citra yang berintensitas *noise* 10% memiliki selisih nilai MSE sebesar 9,68, dan untuk citra yang berintensitas *noise* 50% memiliki selisih nilai MSE sebesar 43,5. Hal ini menunjukkan bahwa komputasi menggunakan GPU tidak hanya dapat meningkatkan waktu proses,

namun juga dapat menghasilkan kualitas citra yang lebih baik dibandingkan dengan komputasi menggunakan CPU.

Tabel 4.8 Nilai MSE untuk citra berintensitas *noise* sebesar 50%

Nama	Image ber-Noise	windowing 3		windowing 5		windowing 7	
		cpu	gpu	cpu	gpu	cpu	gpu
sp iA1_50%	20122.3	3439.5	3430.4	2165.2	2150.8	1936.9	1912.1
sp iA2_50%	20509.7	3478.6	3466.8	2200.0	2186.1	1984.9	1964.5
sp iA3_50%	22268.4	4545.9	4522.6	3218.2	3189.0	2990.5	2947.8
sp iA4_50%	20874.3	3613.8	3594.0	2317.5	2287.3	2080.5	2032.1
sp iA5_50%	20487.5	3704.7	3694.8	2446.0	2430.3	2228.9	2204.5
sp iA6_50%	20198.6	3235.9	3218.1	1939.7	1914.6	1696.7	1656.1
sp iA7_50%	25187.3	5619.7	5599.5	4128.3	4106.8	3849.7	3822.7
sp iA8_50%	22587.0	4356.6	4344.4	2974.9	2956.8	2712.0	2677.6
sp iA9_50%	19483.2	2974.3	2957.4	1722.3	1700.2	1485.4	1451.9
sp iA10_50%	27889.6	6838.2	6797.3	5221.8	5171.7	4894.1	4818.9
sp iA11_50%	20313.7	3622.4	3604.3	2330.1	2305.6	2073.6	2036.0
sp iA12_50%	19916.1	3011.4	3001.7	1707.0	1695.3	1441.8	1425.7
sp iA13_50%	25881.2	6169.0	6125.2	4642.1	4586.4	4334.7	4250.2
sp iA14_50%	23294.8	5448.7	5419.8	4056.5	4017.5	3777.8	3720.9
sp iA15_50%	26881.3	6470.6	6429.6	4932.2	4880.5	4635.5	4559.8
sp iA16_50%	20989.8	3630.6	3610.0	2292.3	2263.8	2021.4	1977.6
sp iA17_50%	21887.5	3967.5	3947.9	2606.7	2579.7	2348.4	2306.7
sp iA18_50%	20308.4	3510.9	3497.1	2278.4	2260.5	2086.1	2058.2
sp iA19_50%	21348.4	3752.7	3737.5	2416.6	2394.4	2169.4	2135.6
sp iA20_50%	23626.5	4999.6	4977.6	3640.9	3611.0	3453.6	3407.1
sp iA21_50%	19972.8	3058.8	3039.0	1777.8	1752.9	1534.8	1498.7
sp iA22_50%	20423.9	3542.6	3529.4	2245.5	2230.0	1990.3	1970.8
sp iA23_50%	22767.8	5247.2	5215.5	3886.4	3844.3	3642.8	3577.8
sp iA24_50%	24730.7	5334.7	5313.6	3863.6	3834.1	3578.4	3528.2
sp iA25_50%	22916.6	4733.5	4712.5	3335.1	3311.9	3059.7	3027.0
sp iA26_50%	22660.9	4462.3	4450.5	3077.0	3064.4	2815.4	2797.4
sp iA27_50%	23721.3	5332.1	5307.3	3920.4	3892.7	3638.3	3602.6
sp iA28_50%	27459.8	6679.0	6623.4	5086.7	5015.1	4763.0	4656.8
sp iA29_50%	23113.7	4585.9	4554.5	3166.4	3124.7	2887.7	2823.4
sp iA30_50%	22532.7	4300.3	4279.0	2916.1	2888.8	2662.2	2619.0

Berdasarkan hasil rata-rata nilai MSE yang ditunjukkan pada tabel 4.9, nilai rata-rata MSE untuk data yang diproses pada CPU-GPU lebih kecil daripada data yang diproses pada CPU. Hal ini menunjukkan bahwa komputasi pararel

(pemrosesan filter pada CPU-GPU) memiliki peningkatan kualitas lebih baik daripada komputasi sekuensial (pemrosesan filter pada CPU). Namun, nilai MSE terkecil terdapat pada data dengan intensitas *noise* terkecil yaitu 5% dan data dengan perbesaran *windowing* terbesar yaitu 7×7. Dalam tabel 4.9 juga ditampilkan rata-rata nilai MSE pada citra ber-*noise* sebelum dilakukan penghapusan *noise*. Sehingga terlihat adanya pengurangan *noise* setelah dilakukan proses penghapusan *noise*. Hal ini dinyatakan dengan adanya perbedaan nilai MSE antara citra sebelum dan sesudah dilakukan penghapusan *noise*.

Tabel 4.9 Rata-rata nilai MSE

Noise	Noise	kernel 3		kernel 5		kernel 7	
		cpu	gpu	cpu	gpu	cpu	gpu
Salt & pepper 5%	2977.79	384.41	379.64	254.11	246.95	279.85	268.89
Salt & pepper 10%	5454.68	736.92	732.49	438.90	432.49	428.03	418.36
Salt & pepper 50%	22478.5	4455.6	4433.4	3083.7	3054.9	2825.8	2782.3

Hal ini menunjukkan bahwa data yang diproses dengan pebesaran *windowing* lebih besar memiliki kualitas citra lebih baik dibandingkan dengan data yang diproses dengan perbesaran *windowing* yang lebih kecil. Dan juga sebaliknya, data yang diproses dengan pebesaran *windowing* lebih kecil memiliki kualitas citra tidak begitu baik dibandingkan dengan data yang diproses dengan perbesaan *windowing* yang lebih besar.

Berdasarkan pada hasil MSE terdapat perbedaan antara nilai hasil citra yang diproses menggunakan GPU dan CPU. Pada dasarnya, algoritma yang digunakan untuk penghapusan *noise* pada GPU dan CPU adalah sama. Namun output yang dihasilkan berbeda. Hal ini disebabkan karena perbedaan pemetaan data. Pada GPU, umumnya data diolah secara paralel yaitu dibagi kedalam *work-*

goup, kemudian data diproses secara serial dalam setiap *work-item*. Sehingga pada proses konvolusi terdapat lebih banyak tepi *pixel*.Untuk pengolahan *pixel* yang berada ditepi dan tidak memiliki *pixel* tetangga, maka nilai *pixel* tetangganya adalah 0. Perubahan nilai *pixel* ini dapat merubah hasil konvolusi. Sedangkan pada CPU, semua data diolah secara serial. Sehingga tepi *pixel* pada data ini adalah tepi citra itu sendiri.

Berdasarkan dari pembahasan diatas, maka dapat disimpulkan bahwa komputasi pararel lebih cepat dan lebih banyak peningkatan kualitas dibandingkan dengan komputasi sekuensial. Namun, hal ini tidak dalam satu tipe data masukan. Kecepatan waktu proses tertinggi terjadi pada data dengan intensitas *noise* dan perbesaran *windowing* terbesar. Sedangkan peningkatan kualitas paling banyak terjadi pada data dengan intensitas *noise* terkecil dan perbesaran *windowing* terbesar.

4.5. Integrasi Program dan Al-Qur'an

Skripsi ini mengimplementasikan metode *Gaussianfilter* untuk penghapusan *noise* citra pada smartphone menggunakan komputasi pararel. Dengan komputasi pararel waktu yang dibutuhkan untuk proses komputasi lebih sedikit dibandingkan dengan komputasi sekuensial. Sehingga dengan komputasi pararel kita dapat menghemat waktu untuk memproses sebuah gambar. Dalam Islam kita dianjurkan untuk menghemat waktu. Telah dikatakan dalam Al-Qur'an Surah *Al-Ashr* (103) ayat 1-3 Allah SWT berfirman :

ٱلۡحَقِّ وَتَوَاصَوۡاْ ٱلصَّٰلِحَٰتِ وَعَمِلُواْ ءَامَنُواْ ٱلَّذِينَ إِلَّا ٣ خُسۡرٖ لَفِي ٱلۡإِنسَٰنَ إِنَّ ٢ وَٱلۡعَصۡرِ ١
٣ بِٱلصَّبۡرِ وَتَوَاصَوۡاْ بِ

"Demi masa; Sesungguhnya manusia itu benar-benar dalam kerugian; Kecuali orang-orang yang beriman dan mengerjakan amal saleh dan nasehat menasehati supaya mentaati kebenaran dan nasehat menasehati supaya menetapi kesabaran." (*Q.SAl-Ashr*/103: 1-3)

Menurut Quraish Shihab (2002), dalam ayat tersebut dijelaskan bahwa waktu adalah modal utama manusia, apabila tidak diisi dengan kegiatan yang positif, maka ia akan berlalu begitu saja. Ia akan hilang dan ketika itu jangankan keuntungan yang diperoleh, modal pun telah hilang. Sayyidina 'Ali ra. Pernah berkata: "Rezeki yang tidak diperoleh hari ini masih dapat diharapkan lebih dari itu diperoleh hari esok, tetapi waktu yang berlalu hari ini tidak mungkin dapat diharapkan kembali esok." Jika demikian waktu harus dimanfaatkan. Apabila tidak diisi maka kita merugi, bahkan kalau pun diisi tetapi dengan hal-hal yang negatif maka manusia pun diliputi oleh kerugian.

Ayat diatas menjelaskan bahwa manusia memang benar-benar berada dalam kerugian apabila tidak memanfaatkan waktu yang telah diberikan oleh Allah secara optimal. Dimana Allah SWT memberikan input waktu yang sama kepada manusia, namun diantara mereka menghasilkan pencapaian yang berbeda. Hal tersebut dikarenakan kesempatan waktu yang dilakukan manusia berbeda-beda. Selain itu komputasi pararel juga dapat menghemat penggunaan listrik. Energi listrik yang kita nikmati sehari-hari pada umumnya berasal dari bahan bakar fosil, seperti gas, batubara, dan minyak bumi. Ketersediaan bahan bakar fosil tersebut pada umumnya sangat terbatas. Jika kita tidak memanfaatkan energi listrik dengan baik, maka bukanlah tidak mungkin jika persediaan listrik habis.

Selama ini penggunaan teknologi menjadi faktor utama penyumbang emisi gas rumah kaca (Stern, 2007). Saat ini, Efek gas rumah kaca sudah kita rasakan. Contohnya jika kita keluar rumah pada siang hari pasti akan merasakan panas yang luar biasa. Selain itu, dampak ini menimbulkan perubahan cuaca yang tidak menentu, naiknya air dipermukaan laut, punahnya hewan-hewan karena ekosistem yang tidak baik. Padahal Allah telah memerintahkan manusia untuk memelihara, melestarikan, serta menjaga seluruh isi alam dari kerusakan. Seperti yang terkandung dalam *Q.S Al-A'raaf* (07) ayat 56.

مِّنَ قَرِيبٌ ٱللَّهِ رَحۡمَتَ إِنَّ وَطَمَعٗاۚ خَوۡفٗا وَٱدۡعُوهُ إِصۡلَٰحِهَا بَعۡدَ ٱلۡأَرۡضِ فِي تُفۡسِدُواْ وَلَا
٥٦ ٱلۡمُحۡسِنِينَ

"*Dan janganlah kamu membuat kerusakan di muka bumi, sesudah (Allah) memperbaikinya dan berdoalah kepada-Nya dengan rasa takut (Tidak akan diterima) dan harapan (akan dikabulkan). Sesungguhnya rahmat Allah amat dekat kepada orang-orang yang berbuat baik."* (*Q.S Al-A'raaf*/07: 56)

Dalam ayat ini Allah melarang manusia agar tidak membuat kerusakan di muka bumi. Larangan membuat kerusakan ini mencakup semua bidang, seperti merusak pergaulan, jasmani dan rohani orang lain, kehidupan dan sumber-sumber penghidupan (pertanian, perdagangan, dan lain-lain), merusak lingkungan dan lain sebagainya. Bumi ini sudah diciptakan Allah dengan segala kelengkapannya, seperti gunung, lembah, sungai, lautan, daratan, hutan dan lain-lain, yang semuanya ditujukan untuk keperluan manusia, agar dapat diolah dan dimanfaatkan dengan sebaik-baiknya untuk kesejahteraan mereka. Oleh karena itu, manusia dilarang membuat kerusakan di muka bumi (Departemen Agama RI, 2010).

Menurut tafsir Ibnu Katsir (2002), Allah Ta'ala melarang dari melakukan perusakan dan hal-hal yang membahayakannya, setelah dilakukan perbaikan atasnya. Karena jika berbagai macam usrusan sudah berjalan dengan baik dan setelah itu terjadi perusakan, maka yang demikian itu lebih berbahaya bagi umat manusia. Maka Allah Ta'ala melarang hal itu, dan memerintahkan hamba-hambanya untuk beribadah, berdo'a dan merendahkan diri kepada-Nya, serta menundukkan diri di hadapan-Nya.

Pada *Q.S Ar-Rum* (30) ayat 41 juga diterangkan bahwa kerusakan di darat dan di laut diakibatkan oleh ulah tangan orang-orang musyrik, kafir, dan muslim yang tidak sadar bahwa alam semesta adalah juga milik Allah yang harus dijaga dan dipelihara seperti menjaga diri sendiri.

ظَهَرَ ٱلْفَسَادُ فِى ٱلْبَرِّ وَٱلْبَحْرِ بِمَا كَسَبَتْ أَيْدِى ٱلنَّاسِ لِيُذِيقَهُم بَعْضَ ٱلَّذِى عَمِلُوا۟ لَعَلَّهُمْ يَرْجِعُونَ ﴿٤١﴾

"Telah nampak kerusakan di darat dan di laut disebabkan Karena perbuatan tangan manusia, supaya Allah merasakan kepada mereka sebahagian dari (akibat) perbuatan mereka, agar mereka kembali (ke jalan yang benar)."(*Q.S Ar-Rum*/30:41)

Menurut tafsir Al-Qur'an oleh departemen Agama RI (2010), dalam ayat ini diterangkan bahwa telah terjadi *al-fasad* di daratan dan lautan. *Al-fasad* adalah segala bentuk pelanggaran atas sistem atau hukum yang dibuat Allah, yang diterjemahkan dengan "perusakan". Perusakan itu bisa berupa pencemaraan alam sehingga tidak layak lagi didiami, atau bahkan penghancuran alam sehingga tidak bisa lagi dimanfaatkan. Di daratan, misalnya hancurnya flora dan fauna, di laut seperti rusaknya biota laut. Perusakan itu terjadi akibat perilaku manusia, misalnya eksploitasi alam yang berlebihan.

Menurut Quraish Shihab (2002) kata *al-fasad* menurut al-Ashfahani adalah keluarnya sesuatu dari keseimbangan, baik sedikit maupun banyak. Kata ini digunakan menunjuk apa saja, baik jasmani, jiwa, maupun hal-hal lain. Dosa dan pelanggaran (*al-fasad*) yang dilakukan manusia yang mengakibatkan kerusakan di darat dan di laut. Ketiadaan keseimbangan di darat dan di laut, mengakibatkan siksaan kepada manusia. Semakin banyak perusakan terhadap lingkungan, semakin besar pula dampak buruknya terhadap manusia. Semakin banyak dan beraneka ragam dosa manusia, semakin parah pula kerusakan lingkungan. Hakikat ini merupakan kenyataan yang tidak dapat dipungkiri lebih-lebih dewasa ini. Allah menciptakan semua makhluk, saling kait terkait. Dalam keterkaitan itu, lahir keserasian dan keseimbangan dari yang terkecil hingga yang terbesar, dan semua tunduk dalam pengaturan Allah Yang Maha Besar. Bila terjadi gangguan pada keharmonissan dan keseimbangan itu, maka kerusakan terjadi, dan ini kecil atau besar, pasti berdampak pada seluruh bagian alam, termasuk manusia, baik yang merusak maupun yang merestui kerusakan itu.

Pemrograman pararel ditujukan untuk meningkatkan performa komputasi. Performa dalam pemrograman pararel diukur dari berapa banyak peningkatan kecepatan (waktu proses) yang dibutuhkan untuk memproses program. Sehingga teknologi ini dapat mengurangi kerusakan lingkungan akibat penggunaan energi listrik yang berlebihan.

BAB V

PENUTUP

5.1. Kesimpulan

Berdasarkan penelitian yang telah dilakukan, maka dapat disimpulkan bahwa:

5.1.1. Implementasi metode *gaussian filter* untuk penghapusan *noise* pada citra menggunakan GPU mampu berjalan lebih cepat dibandingkan implementasi pada CPU, karena implementasi pada GPU menggunakan platform OpenCL yang melibatkan beberapa prosesor dari GPU.

5.1.2. Data yang memiliki intensitas *noise* lebih kecil dan dengan perbesaran *windowing* lebih besar memiliki nilai MSE lebih kecil (banyak peningkatan kualitas). Sebaliknya, data yang memiliki intensitas *noise* lebih besar dan dengan perbesaran *windowing* lebih kecil menghasilkan kualitas citra yang kurang baik.

5.2. Saran

Beberapa saran untuk penelitian dan pengembangan selanjutnya adalah sebagai berikut:

5.2.1. Menggunakan metode yang berbeda untuk penghapusan noise pada citra berwarna dengan menggunakan komputasi pararel pada platform yang lain.

5.2.2. Implementasi pada metode yang sama, namun dengan pengaturan *thread* yang berbeda untuk menangani data.

5.2.3. Meningkatkan kemampuan kecepatan, namun dengan akurasi yang tinggi. Sehingga akan tercipta produk *filter* berkualitas tinggi dan cepat.

DAFTAR PUSTAKA

Abdullah bin Muhammad. 1994. *Tafsir Ibn Katsir, Jilid 3*. Diterjemahkan oleh: M. Abdul Ghoffar E.M. Bogor: Pustaka Imam Asy-Syafi'i.

Ahmad, Usman. 2005. *Pengolahan Citra Digital dan Teknik Pemrogramannya*. Edisi Pertama. Yogyakarta : Graha Ilmu.

Al-Najjar, Yusra A. Y dan Dr. Der Chen Soong. 2012. *Comparison of Image Quality Assessment: PSNR, HVS, SSIM, UIQI*. *International Journal of Scientific & Engineering Research*. Volume 3.

Banger, Ravishekhar dan Koushik Bhattcharyya. 2013. *OpenCL Programming by Example*. Birmingham: Packt Publishing.

Departemen Agama RI. 2010. *Al-Qur'an dan Tafsirnya*. Jakarta: Lentera Abadi

docs.nvidia.com/cuda/. Dankses tanggal 26 September 2016.

Efford, N. 2000. *Digital Image Processing a Practical Introduction Using Java*. Essex: Pearson Education Limited.

Gaster, Benedict R, *et all* . 2013. *Heterogeneous Computing With Opencl*. Usa: elsevier inc

Gonzalez, R.C.; Woods, R.E. 2002. *DigitalImage Processing. Prentice Hall.*

J. A. Fraire, p. Ferreyra and c. Marques.2013.*Opencl Overview, Implementation, And Performance Comparison*. IEEE latin america transactions, vol. 11.

Jr, M. Kudelka. 2012. *Image Quality Assessment*. Proceedings of Contributed Papers, 2012: 94-99

Kadir, Abdul dan Adhi Susanto. 2013. *Teori dan Aplikasi Pengolahan Citra*. Yogyakarta: Andi Publisher.

Khronos.org/registry/cl/sdk/1.0/docs.Diakses tanggal 22 desember 2015.

Matsumoto, Kazuya, *et all*. 2012. *Performance Tuning of Matrix Multiplication in OpenCL on Different GPUs and CPUs*. IEEE Computer Society,2012,pp.396-405.

Munir Rinaldi. 2004. *Pengolahan Citra Digital dengan Pendekatan Algoritmik*. Bandung : Informatika.

Munshi, Aaftab, e*t all*. 2012. *OpenCL Programming Guide*. United States of America: Pearson Education, Inc.

Munshi, Aaftab. 2012. *The OpenCL Specification*. Amerika: The Khronos Group Inc.

Putra, Darma. 2010. *Pengolahan Citra Digital*. Yogyakarta: Andi Publisher.

Ravibabu, P. et.al. 2014 *GPU Implementation of Belief Propagation Method in Image Restoration Using OpenCL*. Computer and Communications Technologies (ICCCT).

Sanchez, Maria G, *et all.* 2014. *Image Noise Removal on Heterogenous CPU-GPU Configuration*. Elsevier B. V. 29,(2014),2219-2229

Sarode, Milindkumar V dan Prashant R. Deshmukh. 2011. *Reduction of Speckle Noise and Image Enhancement of Image Using Filtering Technique*. International Journal of Advancements in Technology, Vol 2.

Shihab, M. Quraish. 2002. *Tafsir Al Mishbah : Pesan, Kesan dan Keserasian Al-Qur'an*. Jakarta: Lentera Hati.

Stern, N. H., & Great Britain. 2007. *The economics of climate change: The Stern review*. Cambridge, UK: Cambridge University Press.

Wang, Lipeng dan Hanbin Wang. 2011. *Implementation of a Soft Morphological Filter Based on GPU Framework*. IEEE,2011,pp.1-4.

Wilkinson, Barry dan Michael Allen. 2005. *Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers-second Edition*. New Jersey: Pearson Education, Inc.

Won Lee, Chang, *et all.* 2014. *Two-way Partitioning of a Recursive Gaussian Filter in CUDA. EURASIP Journal on Image and Video Processing* 2014, 2014:33

LAMPIRAN

1. Data Masukan dan Data Hasil Uji dengan Intensitas *Noise* 5%.

Nama Citra	Citra Masukan	Citra Hasil CPU			Citra Hasil GPU		
		Kernel 3×3	Kernel 5×5	Kernel 7×7	Kernel 3×3	Kernel 5×5	Kernel 7×7
sp iA1_5%							
sp iA2_5%							
sp iA3_5%							
sp iA4_5%							
sp iA5_5%							

sp iA6_5%							
sp iA7_5%							
sp iA8_5%							
sp iA9_5%							
sp iA10_5%							
sp iA11_5%							
sp iA12_5%							
sp iA13_5%							

sp iA14_5%							
sp iA15_5%							
sp iA16_5%							
sp iA17_5%							
sp iA18_5%							
sp iA19_5%							
sp iA20_5%							
sp iA21_5%							

sp iA22_5%							
sp iA23_5%							
sp iA24_5%							
sp iA25_5%							
sp iA26_5%							
sp iA27_5%							
sp iA28_5%							
sp iA29_5%							

sp iA30_5%							

2. Data Masukan dan Data Hasil Uji dengan Intensitas *Noise* 10%.

Nama Citra	Citra Masukan	Citra Hasil			Citra Hasil		
		Kernel 3×3	Kernel 5×5	Kernel 7×7	Kernel 3×3	Kernel 5×5	Kernel 7×7
sp iA1_10%							
sp iA2_10%							
sp iA3_10%							
sp iA4_10%							
sp iA5_10%							

sp iA6_10%							
sp iA7_10%							
sp iA8_10%							
sp iA9_10%							
sp iA10_10%							
sp iA11_10%							
sp iA12_10%							
sp iA13_10%							

sp iA14_10%							
sp iA15_10%							
sp iA16_10%							
sp iA17_10%							
sp iA18_10%							
sp iA19_10%							
sp iA20_10%							
sp iA21_10%							

sp iA22_10%							
sp iA23_10%							
sp iA24_10%							
sp iA25_10%							
sp iA26_10%							
sp iA27_10%							
sp iA28_10%							
sp iA29_10%							

sp iA30_10%							

3. Data Masukan dan Data Hasil Uji dengan Intensitas *Noise* 50%.

Nama Citra	Citra Masukan	Citra Hasil CPU			Citra Hasil GPU		
		Kernel 3×3	Kernel 5×5	Kernel 7×7	Kernel 3×3	Kernel 5×5	Kernel 7×7
sp iA1_50%							
sp iA2_50%							
sp iA3_50%							
sp iA4_50%							
sp iA5_50%							

sp iA6_50%							
sp iA7_50%							
sp iA8_50%							
sp iA9_50%							
sp iA10_50%							
sp iA11_50%							
sp iA12_50%							
sp iA13_50%							

sp iA14_50%							
sp iA15_50%							
sp iA16_50%							
sp iA17_50%							
sp iA18_50%							
sp iA19_50%							
sp iA20_50%							
sp iA21_50%							

sp iA22_50%							
sp iA23_50%							
sp iA24_50%							
sp iA25_50%							
sp iA26_50%							
sp iA27_50%							
sp iA28_50%							
sp iA29_50%							

sp iA30_50%							