UNIVERSITAS INDONESIA

IMPLEMENTASI PLATFORM GENERAL PURPOSE GRAPHIC PROCESSING UNIT UNTUK PROSES SINGULAR VALUE DECOMPOSITION PADA SIMPLE-O

SKRIPSI

BOMA ANANTASATYAADHI

0606029372

FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK ELEKTRO
DEPOK
JUNI 2010

UNIVERSITAS INDONESIA

IMPLEMENTASI PLATFORM GENERAL PURPOSE GRAPHIC PROCESSING UNIT UNTUK PROSES SINGULAR VALUE DECOMPOSITION PADA SIMPLE-O

SKRIPSI

Diajukansebagaisalahsatusyaratmemperolehgelarsarjana

BOMA ANANTASATYA ADHI

0606029372

FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK ELEKTRO
DEPOK
JUNI 2010

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama : BomaAnantasatyaAdhi
NPM : 0606029372
Tanda Tangan :
Tanggal : 8Juni 2010

HALAMAN PENGESAHAN

Skripsiinidiajukanoleh	:
Nama	: BomaAnantasatyaAdhi
NPM	: 0606029372
Program Studi	: TeknikElektro
JudulSkripsi	: Implementasi Platform General Purpose Graphic Processing Unit untuk Proses Singular Value Decomposition pada Simple-O

Telahberhasildipertahankan di hadapanDewanPengujidanditerimasebagaibagianpersyaratan yang diperlukanuntukmemperolehgelarSarjanaTeknikpada Program StudiTeknikElektro, FakultasTeknik, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing	: Dr. Ir. AnakAgungPutriRatnaM.Eng	( )
Penguji	: Ir. EndangSriningsih MT, Si	( )
Penguji	: Muhammad Salman ST., MIT	( )

Ditetapkan di : Depok

Tanggal : 28 Juni 2010

UCAPAN TERIMA KASIH

Puji syukur saya panjatkan kehadirat Allah SWT, karena atas segala rahmat dan penyertaan-Nya saya dapat menyelesaikan skripsi ini. Saya menyadari bahwa skripsi ini tidak akan terselesaikan tanpa bantuan dari berbagai pihak. Oleh karena itu, saya mengucapkan terima kasih kepada :

1. IbuDr. Ir. Anak Agung Putri Ratna M.Eng, selaku pembimbing yang membantumemberikanarahandannasihatsehinggasayadapatmenyelesaikanskripsi ini;
2. Para peneliti sebelum ini yang juga memberikan sumber bacaan yang banyak bagi saya;
3. Orang tua saya yang selalu memberikan dukungan kepada saya;
4. Dan seluruh Sivitas akademik Departemen Teknik Elektro yang tidak dapat saya sebutkan satu persatu.

Akhir kata, semoga Allah SWTberkenan membalas kebaikan semua pihak yang telah membantu. Semoga skripsi ini bermanfaat bagi perkembangan ilmu pengetahuan.

Depok, Juni 2010

BomaAnantasatyaAdhi

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI
TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademika Universitas Indonesia, saya bertanda tangan di bawah ini :

Nama : BomaAnantasatyaAdhi
NPM : 0606029372
Program studi : Teknik Elektro
Departemen : Teknik Elektro
Fakultas : Teknik
Jenis karya : Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Nonoksklusif (*Non-exclusive Royalty Free Right*)** atas karya ilmiah saya yang berjudul :

IMPLEMENTASI PLATFORM GENERAL PURPOSE GRAPHIC PROCESSING UNIT UNTUK PROSES SINGULAR VALUE DECOMPOSITION PADA SIMPLE-O

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non Eksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/format-kan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta sebagai pemegang Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 8Juni 2010
Yang menyatakan

BomaAnantasatyaAdhi

ABSTRAK

Nama : BomaAnantasatyaAdhi
Program studi : Teknik Elektro
Judul :Implementasi Platform General Purpose Graphic Processing Unit untuk Proses Singular Value Decomposition pada Simple-O

SIMPLE-O merupakansistempenilaianesaiotomatisberbasis*latent semantic analysis* (LSA) yang bergantungpada Java Matrix untukmelakukanperhitungan*singular value decomposition* (SVD) dalammelakukanpenilaian.Padaskripsiiniakandibahasmengenaiimplementasi proses SVD pada platform *general purposes graphic processing unit* (GPGPU)pada SIMPLE-O yang lebihcepatdaripadaalgoritmasekuensialbiasa yang adapada JAMA.GPGPU merupakansuatu platform komputasiparalelperformatinggi yang berbasiskan*Graphic Processing Unit*komersialbiasa. Implementasiakandilakukandengancaramemindahkan proses eksekusi SVD pada SIMPLE-O kemoduleksternal yang ditulisdalambahasaCdengan*Application Programming Interface* (API) untuk GPGPU seperti CUDA, CULA tools, danOpenCL. Performadiukurdenganpeningkatankecepatanwaktukalkulasi SVD danjumlahkalkulasi yang dapatdilakukansetiapdetiknya.Implementasi GPGPU meningkatkan performa pada matriks ukuran 512x512 berkisar antara lebih dari 200 kali lipat (CULA tools) hingga 4200 kali lipat (OpenCL).

Kata kunci :CUDA, CULA tools, GPU, GPGPU, LSA, Open-CL,SIMPLE-O,SVD.

ABSTRACT

Name : BomaAnantasatyaAdhi
Study program: Electrical Engineering
Title : Implementation of General Purpose Graphic Processing Unit Platform for Singular Value Decomposition in Simple-O

Simple-O is an automated essay grading system based on latent semantic analysis (LSA) which depends on Java Matrix (JAMA) for singular value decomposition (SVD) calculation.This paper will present an implementation of SVD calculation on General Purpose Graphic Processing Unit (GPGPU) platformin SIMPLE-O, which is essentially faster and more efficient than standard sequential algorithm found in JAMA. GPGPU is a high performance parallel computing platform based on commercially available 3D Graphic Processing Unit. Implementation will be done by altering the SVD execution unit to pipe an external module written in C with GPGPU Application Programming Interface (API) such as CUDA, CULA tools and OpenCL. Performance will be measured in terms of SVD calculation time improvements and numbers of calculation per second. Over 200 times (CULA tools) up to 4200 times (OpenCL) performance gain were measured in 512 x 512 matrix.

Keyword : CUDA, CULA tools, GPU, GPGPU, LSA, Open-CL,SIMPLE-O,SVD.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Tren ‘serba *online*’ saat ini semakin meningkat. Semua orang berlomba-lomba untuk meng-*online*-kan bidangnya masing-masing. Hampir semua bidang saat ini sudah memiliki versi online nya. Demikian pula dalam bidang pendidikan. Kebutuhan akan pendidikan secara *online* yang dapat dilangsungkan secara praktis dan efisien juga semakin meningkat.

Salah satu komponen pendukung dalam suatu sistem pendidikan *online* adalah sistem untuk melakukan *assessment* atau penilaian peserta didik sebagai evaluasi hasil proses pembelajaran. Agar dapat dilangsungkan dengan efisien, proses *assessment* juga harus dilakukan secara *online*. Metode *assessment* yang digunakan pada umumnya dapat berupa esai maupun bentuk objektif, salah satunya adalah pilihan ganda.

Pilihan ganda lebih mudah untuk dinilai namun bentuknya sangat terbatas. Penilaian *assessment* dengan format pilihan ganda cukup sederhana. Sistem cukup membandingkan pilihan *user* dengan jawaban yang benar, kemudian menghitung nilainya.

Di lain pihak, esai dianggap jauh lebih luwes, namun karena keluwesannya tersebut, menjadikan esai cukup rumit untuk dikoreksi secara otomatis oleh mesin. Oleh karena itu perkembangan sistem penilaian esai secara otomatis berlangsung cukup pesat. Bermacam-macam metode penilaian telah dikembangkan, salah satunya adalah *Latent Semantic Analysis* (LSA).

LSA merupakan metode matematis untuk permodelan dan simulasi arti suatu tulisan dengan menganalisa bentuk dari tulisan tersebut pada tes dengan jumlah yang besar[1]. Dalam proses LSA diperlukan proses aljabar linear yang disebut *Singular Value Decomposition* (SVD), yang merupakan proses analisis vektor pada suatu matriks. Proses SVD ini sendiri jika dibandingkan dengan kemampuan CPU pada masa kini sebenarnya bukan merupakan proses yang sangat rumit. Namun dalam aplikasinya sering sekali diperlukan untuk melakukan

perhitungan SVD dalam jumlah besar secara *realtime* yang dapat membebani performa sistem.

SIMPLE-O merupakan salah satu sistem penilaian esai yang menerapkan algoritma LSA didalamnya [2], [3], [4]. Pada SIMPLE-O, proses SVD dilakukan menggunakan Matlab atau Java Matrix. Dari hasil pengamatan awal, proses perhitungan nilai masih menghabiskan waktu eksekusi yang cukup lama. Pada implementasi dengan MATLAB, ditemui *overhead* waktu yang sangat besar pada proses perhitungannya. Semantara itu, pada implementasi selanjutnya, yaitu menggunakan JAMA, overhead waktu tersebut sudah teratasi, namun performa sistem semakin buruk ketika jumlah data yang harus dikalkulasi bertambah banyak. Oleh karena itu perlu ditemukan solusi penanganan perhitungan SVD secara lebih cepat tanpa harus menambah mesin yang digunakan dalam penilaian.

Di sisi lain, perkembangan *clock* prosesor atau CPU telah berhenti sejak tahun 2003 [5]. Tren komputasi saat ini telah mengarah ke parallelisme. Hampir seluruh produsen CPU telah beralih menuju paralellisme, baik berupa superscalar CPU, maupun menambah jumlah inti dalam sebuah CPU. Dengan parallelisme tersebut pekerjaan yang dapat dilakukan setiap saat menjadi lebih banyak. Walaupun demikian, hingga saat ini CPU *high-end* dengan inti terbanyak pun hanya mencapai 12 inti dengan performa masih pada orde ratusan Giga FLOPS.

Sebagai alternatif, suatu platform baru untuk pemrosesan data secara paralel yang disebut *General Purpose Graphic Processing Unit* (GPGPU) mulai diperkenalkan pada SIGGRAPH 2005. Platform ini memanfaatkan *Graphic Processing Unit* (GPU) berperforma tinggi yang tersedia secara komersial untuk melakukan perhitungan non-grafis secara *massively parallel* dan intensif dengan operasi aritmatika. Perkembangan teknologi GPU berlangsung pesat karena dukungan dari dunia gaming yang semakin menuntut kinerja tinggi untuk pemrosesan grafis. Sebuah GPU modern dapat memiliki 1600 inti pemrosesan dalam sebuah chip dengan kemampuan menghitung hingga 2,7 Terra FLOPS pada harga yang terjangkau.

Pemrograman GPU untuk GPGPU tidak seperti pemrograman CPU pada umumnya yang sekuensial. Diperlukan algoritma dan teknik pemrograman khusus untuk mengimplementasikan SVD dalam GPGPU.

Oleh karena itu, dalam skripsi ini akan dibahas mengenai implementasi SVD untuk SIMPLE-O pada GPGPU, baik menggunakan API CULA pada platform CUDA maupun menggunakan algoritma single-side Jacobi rotation pada platform OpenCL, dengan harapan akan memperoleh peningkatan kinerja dibandingkan dengan sistem terdahulu (implementasi dalam JAMA maupun MATLAB).

1.2 Tujuan Penulisan

Tujuan dari penulisan skripsi ini adalah untuk memaparkan implementasi suatu metode untuk meningkatkan performansi perhitungan SVD dalam SIMPLE-O dengan menggunakan platform GPGPU terutama menggunakan nVidia CUDA dan OpenCL. Pada tahap selanjutnya dilakukan perbandingan kinerja dari masing-masing jenis implementasi dan kemungkinan pengembangan di masa depan.

1.3 Pembatasan Masalah

Topik yang akan dibahas dalam skripsi ini terbatas hanya dalam lingkup teknis implementasi perhitungan SVD pada SIMPLE-O menggunakan platform GPGPU, khususnya CUDA dan OpenCL.

1.4 Metodologi Penulisan

Metode penulisan yang digunakan pada buku skripsi ini adalah:

1. Studi literatur, yaitu dengan mencari sumber-sumber yang digunakan untuk referensi.
2. Teknik observasi, yaitu melakukan pengamatan langsung pada *code* Simple-O yang sudah ada dan pada API terkait GPGPU yang sudah tersedia dipasaran.
3. Teknik eksperimen, yaitu dengan cara melakukan implementasi langsung algoritma SVD menggunakan API CULA pada platform CUDA dan algoritma *single-side Jacobi rotation* pada platform OpenCL. Kemudian dilakukan perbandingan sederhana antara desain sistem lama dan yang telah menggunakan platform GPGPU.

1.5 Sistematika Penulisan

Skripsi ini terdiri dari lima bab, dimana masing-masing bab akan menjelaskan sebagai berikut:

a. Bab 1: Pendahuluan

   Pada bab ini, akan dijelasan mengenai Latar Belakang, Tujuan, Pembatasan Masalah, Metodologi Penulisan, dan Sistematika Penulisan.

b. Bab 2: LSA, SVD, SIMPLE-O dan GPGPU

   Pada bab ini, akan dijelaskan mengenai *Essay Grading* dengan metode LSA, SVD, SIMPLE-O, GPU, GPGPU beserta berbagai API dan platformnya serta program-program yang terkait.

c. Bab 3: Rancang Bangun SIMPLE-O Menggunakan Platform GPGPU

   Pada bab ini akan dipaparkan bagaimana merealisasikan proses LSA pada SIMPLE-O dengan platform perhitungan GPGPU dengan menggunakan API CULA pada platform nVidia CUDA dan OpenCL, baik dari sisi *software* maupun *hardware*.

d. Bab 4: Implementasi, Pengujian dan Analisa Performa SIMPLE-O Menggunakan Platform GPGPU

   Pada bab ini, akan paparkan proses pengujian performa sistem yang telah didesain beserta hasil pengujian dan analisanya.

e. Bab 5: Kesimpulan

   Bab ini berisi kesimpulan yang dapat diambil dari pengujian ini. Selain itu juga akan dipaparkan mengenai kemungkinan pengembangan dimasa yang akan datang.

BAB 2

LSA, SVD, SIMPLE-O dan GPGPU

2.1 *Essay Grading* dengan metode LSA (*Latent Semantic Analysis*)

Dalam melakukan evaluasi terhadap ujian esai diperlukan suatu metode dalam melakukan penilaiannya. Terdapat beberapa metode dalam melakukan penilaian ujian esai, salah satu diantaranya adalah LSA (*Latent Semantic Analysis*).

Latent Semantic Analysis (LSA) merupakan metode matematis untuk permodelan dan simulasi arti suatu tulisan dengan menganalisa bentuk dari tulisan tersebut [1]. LSA merepresentasikan jumlah dan kebolehjadian kata untuk dibandingkan secara geometris (matriks) dalam suatu tulisan dalam matriks dua dimensi yang besar. Bagian terpenting dari pemrosesan dari LSA adalah analisa SVD (*Singular Value Decomposition*) yang mengkompresi informasi yang berkaitan dalam jumlah besar ke dalam ruang yang lebih kecil tetapi mewakili arti sebenarnya. Dengan membandingkan nilai *singular* kedua matriks dari dokumen yang berbeda, maka dapat diketahui kaitan antara dua matriks tersebut.

2.2 *Singular Value Decomposition* (SVD)

2.2.1 Definisi SVD

SVD merupakan salah satu metode dekomposisi matriks dalam aljabar linier. Teknik SVD sering digunakan untuk melakukan perkiraan struktur penggunaan kata dalam dokumen-dokumen. SVD merupakan teknik untuk melakukan estimasi *rank* dari matriks. Jika diketahui matriks A dengan dimensi m x n, dimana nilai m > n dan *rank* (A) = r maka *Singular Value Decomposition* dari A, dinotasikan sebagai SVD (A), didefinisikan melalui persamaaan

$$A = U \sum V^T \tag{2.1}$$

dimana,

$$U^T U = V^T V = I_n \tag{2.2}$$

dan memenuhi kondisi,

$$\sum = \text{diag}\ (\sigma_1, ..., \sigma_n) \tag{2.3}$$

dimana,

$\sigma_i > 0$ untuk $1 \leq i \leq r$

$\sigma_j = 0$ untuk $j \geq r + 1$

Kolom r pertama dari matriks U dan V mendefinisikan vektor eigen ortonormal yang bersesuaian dengan r nilai vektor eigen tidak-nol dari matriks AA^T dan A^TA berturut-turut. Kolom dari matriks U dan V berisi vektor, masing-masing disebut vektor singular kiri dan kanan.

Nilai singular dari A merupakan elemen diagonal dari matriks $\sum$, dimana nilai singular didapat dari akar pangkat dua dari nilai absolut dari sejumlah n nilai eigen dari AA^T [6].

2.2.2 *Jacobi Rotation*

Dalam proses SVD hampir selalu berkaitan dengan *Jacobi Rotation* [7]. Untuk itu, berikut akan dibahas beberapa hal yang berkaitan dengan *Jacobi Rotation*.

Jacobi Rotation didefinisikan $J(i,j,\theta)$ dengan indeks (i,j) dan sudut θ sebagai matriks persegi I (identitas) ditambah elemen tambahan sin dan cos θ pada lokasi perpotongan (i,j).

$$J(i,j,\theta) = \begin{array}{c} \\ i \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} i & & & j & & \end{array} \\ \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & c & & s & \\ & & & \ddots & & \\ & & s & & c & \\ & & & & & \ddots \\ & & & & & & 1 \end{pmatrix} \end{array}$$

Dengan $c = \cos\theta$ dan $s = \sin\theta$. Maka jelas bahwa $J(i,j,\theta)^T J(i,j,\theta) = I$. Pada algoritma SVD klasik, maka akan dipilih nilai θ yang menyebabkan elemen $a_{ij} = a_{ji}$ pada matriks A menjadi bernilai 0 ketika dilakukan rotasi bidang dengan melakukan perkalian $\hat{A} = J^T A J$.

Berikut adalah contoh pada sebuah matriks 2 × 2.

$$\begin{pmatrix} \hat{x} & 0 \\ 0 & \hat{y} \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} x & w \\ w & y \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

Maka akan dicari J yang akan menghilangkan elemen w yang tidak berada di diagonal A. Maka, untuk mencari c dan s yang sesuai dapat dilakukan dengan cara berikut:

$$\alpha = \frac{y - x}{2w} \tag{2.4}$$

$$\tau = \frac{\text{sign}(\alpha)}{|\alpha| + \sqrt{1 + \alpha^2}} \tag{2.5}$$

maka,

$$c = \frac{1}{\sqrt{1 + \tau^2}} \tag{2.5}$$

dan

$$s = \tau c \tag{2.6}$$

2.2.3 Algoritma SVD

2.2.3.1 Algoritma SVD Klasik

Algoritma ini hanya dapat digunakan pada matriks A yang persegi. Pada algoritma ini akan dilakukan transformasi Jacobi pada matriks A hingga matriks tersebut menjadi matriks diagonal.

$$A^{(k+1)} = J^T A^{(k)} J \tag{2.7}$$

dengan $A^{(0)} = A$ hingga:

$$\lim_{k \to \infty} A^{(k)} = \Sigma \tag{2.8}$$

Setiap transformasi Jacobi dipilih untuk menghilangkan elemen non diagonal dari matriks A $a_{ij}^{(k)} = a_{ji}^{(k)}$ yang memiliki nilai absolut terbesar. Untuk itu, dipilih nilai α seperti pada persamaan 2.4 dengan cara berikut:

$$\alpha = \frac{a_{jj} - a_{ii}}{2a_{ij}} \tag{2.9}$$

Setiap transformasi akan membuat matriks A semakin mendekati matriks diagonal, dan setelah transformasi dilakukan beberapa kali pada akhirnya menjadi matriks Σ .

2.2.3.2 Algoritma SVD Golub-Kahan-Reinsch

Algoritma ini adalah algoritma yang umum digunakan pada komputer sekuensial biasa. Algoritma ini diadopsi oleh fungsi SVD pada LAPACK, LINPACK, MATLAB, dan JAMA.

Algoritma ini secara umum melakukan dekomposisi QR dan QL secara bergantian untuk menghilangkan komponen non diagonal dari matriks A satu persatu. Proses ini sama sekali tidak dapat diparalelkan. Algoritma dekomposisi QR itu sendiri sebenarnya dapat diparalelkan, namun waktu eksekusinya tetap tidak efisien [7]. Dapat diambil kesimpulan bahwa algoritma ini tidak tepat untuk diimplementasikan pada platform GPGPU.

2.2.3.3 Algoritma SVD Hestenes-Jacobi

Hestenes menemukan kesamaan dengan melakukan operasi Jacobi hanya pada satu sisi dan melakukan rotasi bidang ortogonal untuk menghilangkan satu elemen matriks [8]. Berikut adalah langkah-langkahnya.

Metode Hestenes-Jacobi menghasilkan suatu matriks ortogonal U. Apabila U dikalikan dengan matriks A, maka akan diperoleh matriks B. Dimana matriks B tidak sama dengan matriks yang dihasilkan rotasi Jacobi biasa. Sehingga dapat ditulis sebagai persamaan berikut:

$$UA = B$$

Dimana B memiliki seluruh baris yang ortogonal, sehingga $b_i^T b_j = 0$ untuk $i \neq j$.

Matriks B dapat dinormalkan dengan menghitung matriks S dimana $s_i = b_i^T b_j$ dan kemudian menulis ulang matriks B sebagai $B = SS^{-1}B = SV$. Dengan matriks V diperoleh dengan cara membagi setiap baris b_i dengan $s_i = b_i^T b_j$. Demikian diperoleh matriks V yang ortonormal.

Karena matriks U adalah matriks ortogonal, maka:

$$UA = SV \Leftrightarrow A = U^T SV$$

Dengan demikian, maka bentuk yang ada di sebelah kanan sudah sesuai dengan definisi SVD. Matriks S merupakan nilai singular dari matriks A.

Algoritma ini adalah dasar algoritma yang akan digunakan dalam implementasi menggunakan OpenCL.

2.3 Simple-O

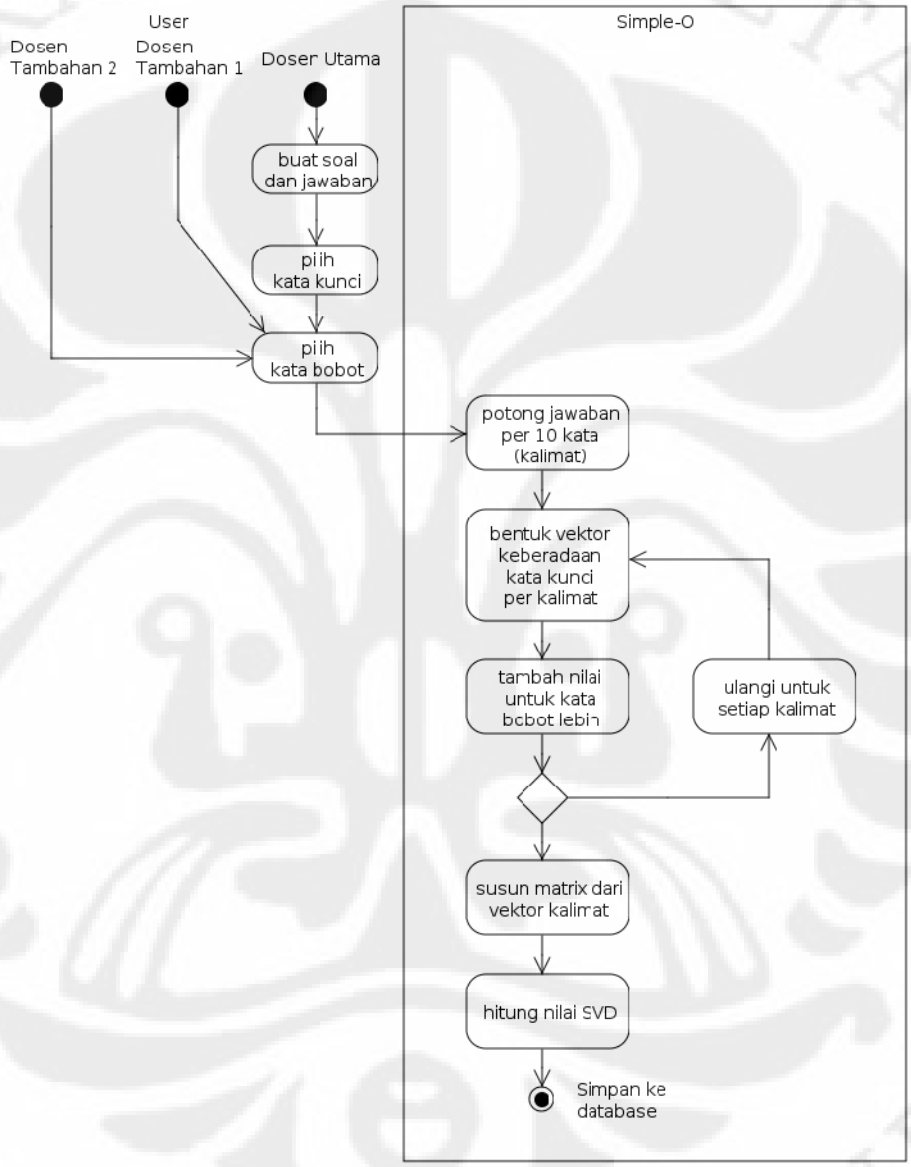
SIMPLE-O merupakan Sistem Penilaian Esei Otomatis yang didesain dengan menggunakan metode LSA [2],[3],[4]. Pada sistem ini proses SVD dilakukan pada Matlab dan kemudian pada versi terbaru dilakukan pada JAMA. Pada sub bab ini, akan dijelaskan bagian-bagian Simple-O yang berkaitan dengan proses LSA saja. Penjelasan detail mengenai Simple-O terdapat pada lampiran 1.

Proses LSA dalam Simple-O dilakukan pada 2 modul. Modul pertama adalah modul dosen, terutama pada saat dosen akan memasukkan soal baru. Modul kedua adalah modul mahasiswa saat nilai dari jawaban mahasiswa akan dihitung.

Dalam modul dosen, seorang dosen utama akan memasukkan soal beserta jawabannya. Setelah itu, dosen diharuskan untuk memilih beberapa kata kunci dan menentukan kata kunci mana yang memiliki bobot lebih. Pemilihan kata kunci bobot lebih dilakukan oleh seorang dosen utama dan 2 orang dosen tambahan yang berbeda. Hanya kata kunci yang dipilih oleh minimal 2 dosen yang akan mendapat bobot lebih. Dengan demikian, maka keterlibatan manusia dalam modul dosen sudah selesai. Proses ini dijelaskan pada Gambar 2.1. Proses selanjutnya akan berlangsung secara otomatis.

Sistem kemudian akan menyiapkan matriks referensi berdasarkan jawaban dosen tersebut. Jawaban akan dipotong-potong menjadi kalimat yang masing-masing terdiri dari 10 kata. Masing-masing kalimat akan dibuatkan sebuah vektor dengan n elemen dimana n adalah jumlah kata kunci. Masing-masing elemen merepresentasikan 1 kata kunci. Bila suatu kata kunci terdapat pada pada kalimat tersebut, maka elemen vektor yang bersangkutan akan diberi nilai, untuk kata kunci yang memiliki bobot lebih, maka nilainya akan lebih tinggi. Kumpulan vektor tersebut kemudian digabungkan menjadi suatu matriks yang menjelaskan komposisi kata dalam jawaban tersebut. Matriks tersebut kemudian dicari nilai singularnya melalu proses SVD. SVD dilakukan dengan bantuan Matlab pada versi terdahulu, sementara saat ini SVD dilakukan menggunakan *library* PHP

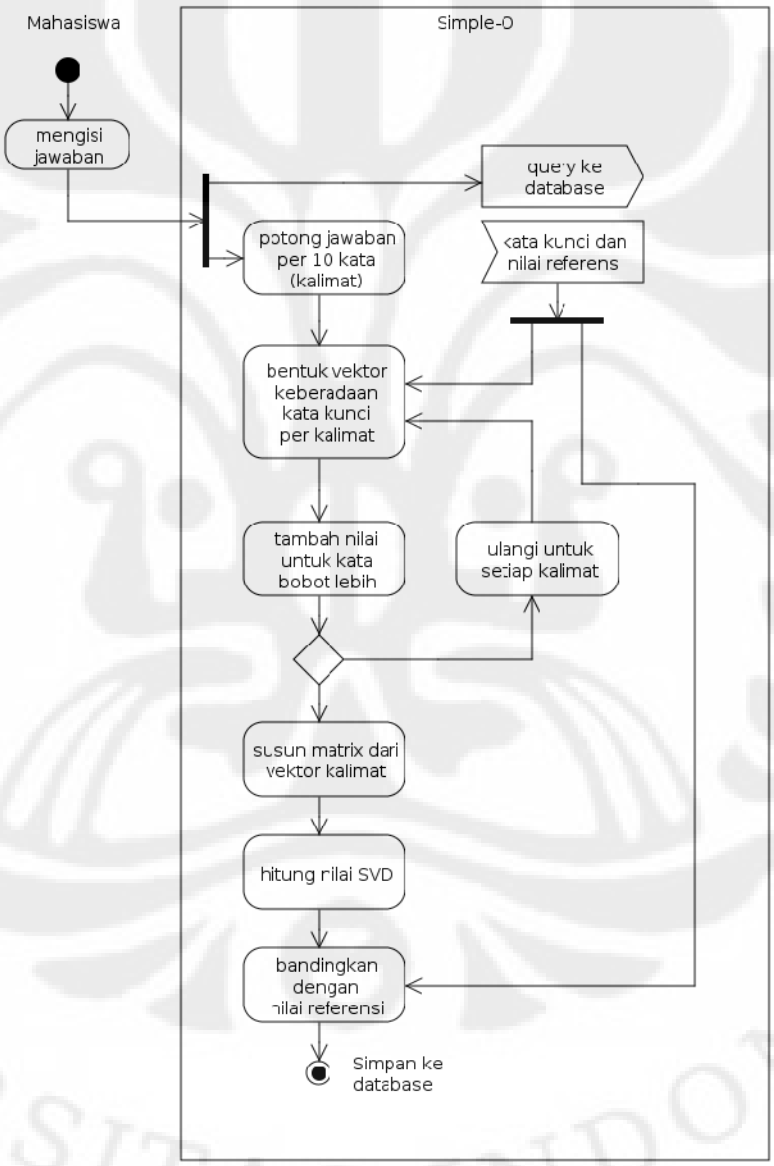
yang di-*porting* dari Java Matrix (JAMA). Nilai singular ini mewakili komposisi kata kunci dalam jawaban dosen tersebut. Nilai singular tersebut beserta kata kunci selanjutnya disimpan dalam database untuk dibandingkan dengan jawaban mahasiswa.



Gambar2. 1 UML *Activity Diagram* dosen menambah soal

Modul yang ke dua adalah modul mahasiswa ketika jawaban yang telah dibuat akan diproses oleh system. Jawaban tersebut kemudian juga dibuat matriksnya seperti pada modul dosen. Namun pada saat pencarian kata kunci,

jawaban mahasiswa akan dibandingkan dengan tabel persamaan kata agar kata-kata yang berbeda namun memiliki arti yang sama dapat dikenali dengan baik (LSA sebenarnya tidak dapat mengenali arti dari masing-masing kata tersebut). Selanjutnya, matriks yang telah dihasilkan juga akan dicari nilai singularnya. Nilai singular ini dibandingkan dengan nilai singular dari jawaban referensi yang sebelumnya sudah dimasukkan oleh dosen pembuat soal. Algoritma proses ini dapat dilihat pada Gambar 2.1.



Gambar2. 2 UML *Activity Diagram* perhitungan nilai mahasiswa

Sebagai catatan, versi Simple-O yang telah menggunakan JAMA memiliki beberapa *bug* dalam perhitungan SVD karena belum selesainya pengimplementasian seluruh fungsi pada versi terdahulu yang menggunakan MATLAB. Sebelum dapat digunakan dalam pengujian ini, telah dilakukan perbaikan pada sebagian *bug* yang ada.

2.4 Graphic Processing Unit

Graphic Processing Unit (GPU) adalah suatu prosesor khusus yang membebastugaskan prosesor utama dalam suatu mesin dari perhitungan yang berkaitan dengan grafis [10].

GPU pada awalnya dirancang untuk *workstation* berperforma tinggi. Harganya sangat mahal dan belum umum ditemukan pada PC. GPU versi awal hanya menyediakan akselerasi perhitungan 2D secara *hardware.*

Pada awal tahun 1990-an mulai marak kemunculan *game* 3D yang masih mengandalkan prosesor untuk melakukan *rendering.* Sejak saat itu, mulai timbul permintaan untuk membuat akselerator 3D secara *hardware*. API OpenGL yang berasal dari aplikasi grafis pada *workstation professional* pun mulai diadopsi untuk melakukan pemrograman grafis *game* 3D. Demikian pula hingga kemunculan DirectX dengan Direct3D-nya. Dengan dorongan semakin berkembangnya bisnis gaming pada PC, GPU semakin lama menjadi lebih terjangkau dan lebih *powerful* dari sebelumnya.

GPU terus berevolusi, pada awal tahun 2001-an, muncul GPU keluarga GeForece3 dari NVIDIA. GPU ini merupakan pelopor GPU yang memiliki *programmable shader*. *Shader* adalah unit pemrosesan data dalam sebuah GPU. Setiap GPU umumnya memiliki lebih dari satu *shader*. *Shader* memungkinkan suatu pemrograman sederhana terhadap *texture* maupun *vertex* sebelum di-*render* ke layar. Pada masa ini lah GPU untuk pertama kalinya dapat di program [5]. Tujuannya untuk meningkatkan performa GPU dalam memproses data grafis secara paralel.

Pada tahun 2002, muncul GPU ATI Radeon 9700 dan GeForce FX 5800 dari NVIDIA. Untuk pertama kalinya GPU mensupport data *floating point.* Pada

tahun 2004 mulai tersedia GPU yang mampu melakukan *shading* dengan *branching*, *looping* dan operasi intensif aritmatika di dalamnya.

2.5 *General Purpose Graphic Processing Unit*

Pada tahun 2006, muncul GPU ATI Radeon HD 2900. GPU tersebut memiliki *unified shader*, yaitu *shader* multi fungsi yang dapat berfungsi sebagai *texture* maupun *vertex shader*. *Unified shader* membuat pemrograman pada GPU menjadi sangat fleksibel [5]. Maka muncul istilah GPGPU, dimana GPU dapat diprogram tidak hanya untuk keperluan grafis saja, namun juga untuk perhitungan lainnya. GPU telah berubah menjadi suatu divais yang dapat diprogram dengan kemampuan perhitungan yang luar biasa paralel. Mulai saat itu, juga diluncurkan suatu keluarga GPU yang dikhususkan untuk GPGPU yang bahkan tidak dioptimalkan untuk pemrosesan grafis. Misalnya NVIDIA Tesla dan ATI FireStream. Kedua *card* tersebut murni ditujukan untuk kepentingan GPGPU.

Pada masa itu, setiap GPU tercanggih biasanya memiliki puluhan hingga ratusan *shader*. Jumlah *shader* yang begitu banyak membuat GPU memiliki kemampuan kalkulasi paralel yang luar biasa dan kemampuan diprogram yang fleksibel. Saat itu kemampuan GPU telah jauh melampaui CPU dalam menangani perhitungan yang bersifat paralel. Yang dimaksud bersifat paralel adalah tidak ada ketergantungan langsung antar data yang diproses pada suatu *shader* dengan data yang diproses pada *shader* lainnya.

Setiap *shader* mampu mengambil data dari memori, kemudian melakukan perhitungan sesuai yang telah diprogramkan pada *shader*, kemudian mengalirkannya keluar. Beberapa GPU juga dapat menulis kembali aliran hasil perhitungannya ke memori. Dengan demikian, maka tiap-tiap *shader* mampu melakukan banyak perhitungan yang independen pada suatu waktu.

Pada GPU modern, jumlah *shader*, atau sering disebut *stream processor* (karena *input* dan *output*-nya berupa aliran/*stream*), telah mencapai ratusan bahkan ribuan. Kemapuan kalkulasi GPU dapat mencapai orde Terra FLOPS, ratusan kali lebih cepat ketimbang CPU. Kemampuan tersebut sudah mulai dimanfaatkan untuk melakukan berbagai perhitungan lain selain grafis. Oleh

karena itu, platform yang melakukan perhitungan non-grafis pada GPU biasa disebut GPGPU.

Pada aplikasinya, pemrograman *stream processor* tidak semudah yang dibayangkan. Terdapat perbedaan karakteristik arsitektur antara GPU dan CPU, sehingga diperlukan perancangan program dari level algoritma agar peningkatan performa dari CPU ke GPU dapat diperoleh.

2.5.1 GPGPU Programming

Secara umum, hampir semua GPU modern yang ditemui dipasaran dapat digunakan untuk GPGPU, baik *card* kelas konsumen (ATI Radeon dan NVIDIA GeForce), kelas profesional (ATI FirePro & FireGL dan NVIDIA Quadro), maupun prosesor khusus GPGPU (ATI FireStream dan NVIDIA Tesla) [10] - [18].

Pemrograman terhadap GPGPU dapat dilakukan dengan 3 cara. Masing-masing cara memiliki kelebihan dan kekurangan masing-masing [14], [15], [19].

Cara pertama adalah dengan melakukan pemrograman *general purposes* pada API 3D pada umumnya seperti Direct3D maupun OpenGL. Pemrograman ini sangat fleksibel, dapat dijalankan pada *platform*, *device*, dan *Operating System* (OS) apa saja. Selama API 3D tersebut dapat dijalankan, maka GPGPU dapat dilakukan.

Proses perhitungan dalam pemrograman cara ini dimodelkan dalam bentuk 3D *primitives*. Misal, data dalam bentuk *array* harus diubah ke dalam *textures,* operasi matematis harus dinyatakan dalam *triangle,* dan lain sebagainya. Sehingga, walaupun *platform independent,* namun diperlukan pengetahuan yang memadai dalam 3D programming. Tidak hanya itu, menyatakan suatu perhitungan matematis menjadi 3D *primitives* bukan merupakan hal yang sederhana. Karena beberapa kekurangannya ini, cara pertama ini sudah mulai ditinggalkan.

Cara yang kedua adalah dengan menggunakan API *highlevel language* yang dikhususkan untuk GPGPU. Bahasanya cenderung didesain untuk melakukan perhitungan, sehingga lebih mudah digunakan. Bahasa ini biasanya diturunkan dari bahasa C/C++ yang telah dimodifikasi [13] - [15], [17], [19].

Highlevel languages ini ada dua jenis, *platform dependent (vendor specific)* dan *platforn independent.* Contoh *highlevel languages* yang *platform independent* adalah OpenCL dari Khronos Group dan DirectCompute dari Microsoft. Cara ini tidak menuntut pengetahuan mengenai 3D *primitives* sama sekali [10], [13], [19].

API yang pertama adalah OpenCL dari Khronos Group. API ini *platform independent*, artinya dapat dijalankan di GPU buatan ATI maupun NVIDIA. OpenCL baru pertama kali diperkenalkan pada Desember 2008. API ini akan dibahas secara detail pada sub bab berikutnya [19].

API yang kedua adalah DirectCompute dari Microsoft yang merupakan bagian dari DirectX11. API ini juga *platform independent*. Referensi mengenai API ini ketika skripsi ini dibuat masih minim, Sehingga tidak akan digunakan dalam implementasi sistem ini.

Selain dua API diatas, ada dua API *high level languages* yang *vendor specific*. Misalnya Brook+ dalam paket ATI Stream Computing maupun C for CUDA dalam paket NVIDIA CUDA [14], [15]. Keduanya akan dibahas lebih lanjut pada subbab berikutnya.

Kelemahan dari cara kedua ini adalah kelemahan pada bahasa *high level* pada umumnya, yaitu kekurangan kemampuan untuk mengakses *hardware* secara detail.

Cara ketiga adalah dengan menggunakan *low level* API yang disediakan oleh masing-masing produsen GPU. Misalnya menggunakan driver ATI CAL atau NVIDIA CUDA secara langsung. Komunikasi antara program *host* dan GPGPU terjadi pada *driver/binary level* [14], [15]. Dengan cara ini, maka dimungkinkan dilakukan optimasi secara radikal untuk *hardware* tertentu, sehingga peningkatan performa secara signifikan dapat diperoleh. Bahasa pemrograman yang digunakan pun bebas karena bekerja pada level *driver/binary*. Pembahasan detail mengenai cara ini akan dilakukan pada subbab selanjutnya.

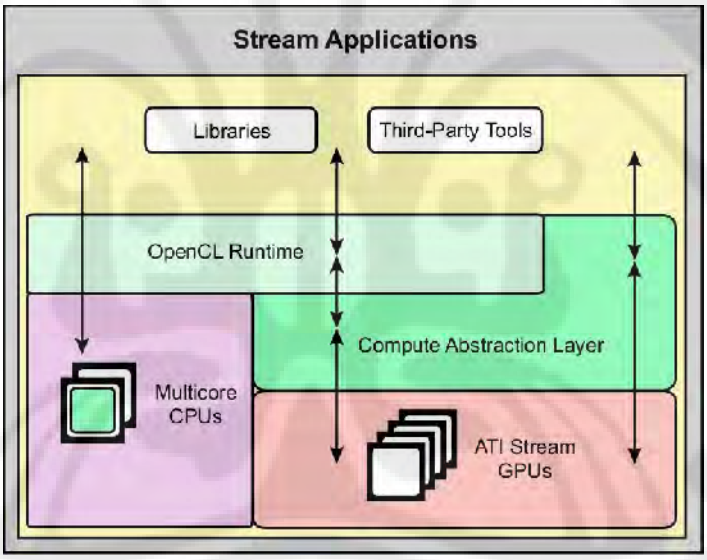
Kelemahannya adalah diperlukan pengetahuan mengenai *hardware* yang digunakan secara mendetail. Selain itu, program yang ditulis untuk suatu devais yang spesifik, tidak akan dapat digunakan pada devais yang berbeda.

2.5.2 ATI Stream SDK

ATI Stream Computing merupakan suatu kesatuan antara ATI Stream Processor dan perangkat lunak pendukungnya [11]. ATI Stream Computing terdiri dari 3 bagian:

- *OpenCL compiler* dan *runtime*
- File *header* dan *compile time library* untuk ATI CAL
- Dokumentasi

Pemrograman pada ATI Stream SDK dapat dilakukan dalam 2 cara. Yaitu menggunakan *driver level* ATI CAL atau menggunakan OpenCL seperti pada Gambar 2.3. Penggunaan *meta programming languages* seperti Brook++ sudah tidak disertakan lagi [14].



Gambar2. 3 ATI Stream Environment

2.5.2.1 ATI CAL

ATI CAL adalah API yang dapat digunakan oleh *developer* untuk mengembangkan *software* yang menggunakan ATI Stream Processor [10].

CAL secara umum terdiri dari 2 bagian:

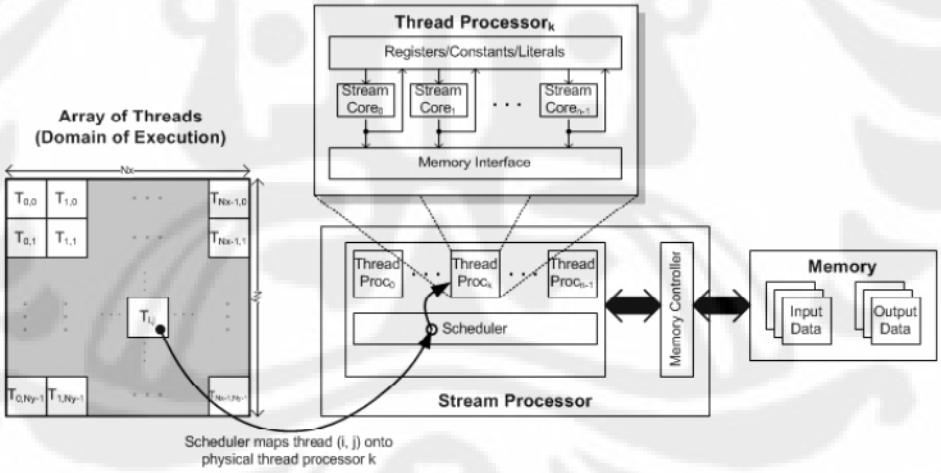
- program yang berjalan di CPU yang ditulis dalam C/C++, atau aplikasi *host*

- dan program yang berjalan di GPU/ATI Stream Processor, atau disebut *kernel*.

ATI CAL merupakan bahasa pemrograman terendah yang dapat digunakan untuk mengakses GPU. Selain dapat digunakan secara langsung, ATI CAL berfungsi untuk menghubungkan API yang lebih tinggi seperti OpenCL ke GPU.

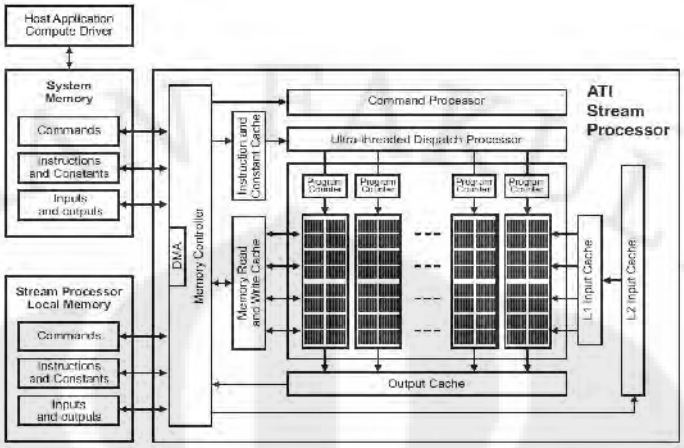
2.5.2.2 ATI Stream Processor

ATI Stream Processor adalah sebutan untuk produk GPU dan turunannya dari ATI yang mendukung *Stream Processing* [12]. Pada gambar 2.4, satu atau lebih *stream processor* terhubung ke CPU melalui *bus* berkecepatan tinggi, misal PCI-E. CPU menjalankan CAL dan mengirim perintah ke GPU menggunakan CAL API. Sementara itu, *stream processor*/GPU menjalankan *kernel* yang diperintahkan oleh CPU. Pada ATI CAL, *stream processor* memiliki akses baik ke memori utama sistem maupun memori lokalnya, sehingga *kernel* dapat diletakkan dimana saja.



Gambar2. 4 Eksekusi Stream Computing
(ATI Stream Computing User Guide , 2009)

ATI Stream Processor dioptimalkan untuk grafis maupun untuk perhitungan umum yang bersifat paralel. Strukturnya merupakan sekumpulan SIMD processor (Gambar 2.5).



Gambar2. 5 Arsitektur GPU ATI R700

(ATI R700 Architecture, 2008)

2.5.3 NVIDIA CUDA

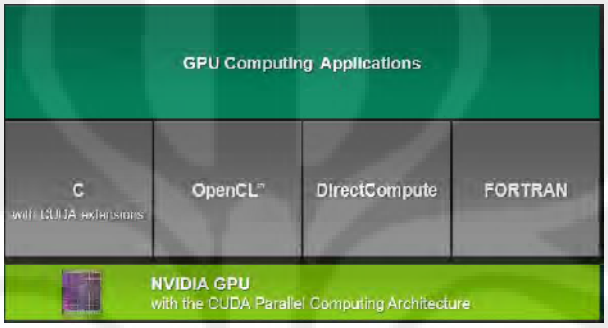
Pada tahun 2006 NVIDIA merilis Compute Unified Device Architecture (CUDA) yang merupakan arsitektur *general purpose parallel computing*. CUDA merupakan satu kesatuan antara GPU NVIDIA yang sudah CUDA *enabled* dan API [15], [18]. Secara umum, konsep penggunaan API-nya mirip dengan ATI CAL. CUDA mengekspos arsitektur pada GPU yang paralel, sementara tetap akan membagi pekerjaan ke CPU untuk tugas-tugas serial, terlihat pada Gambar 2.6.

Gambar2. 6 Perbandingan CPU dan GPU

(NVIDIA CUDA Programming Guide, 2009)

Secara *default* NVIDIA CUDA menyediakan dua bentuk implementasi. Pertama adalah *C for CUDA* yang merupakan suatu API dalam bentuk bahasa C yang diperluas untuk mempermudah implementasi paralelisme dalam suatu algoritma. Dengan CUDA, *developer* dapat lebih fokus mengembangkan

algoritma paralel daripada memikirkan implementasinya. Pada Gambar 2.7, selain dari C for CUDA, terdapat pula alternatif implementasi dalam bahasa lainnya (Fortran, OpenCL, DirectCompute) atau pun menggunakan cara kedua, yaitu melakukan pemrograman menggunakan CUDA driver API yang merupakan low level programming [15]. Berikutnya, akan dibahas lebih lanjut mengenai C for CUDA.



Gambar2. 7 Lingkungan Aplikasi CUDA
(NVIDIA CUDA Programming Guide, 2009)

2.5.3.1 *Kernel*

Dalam C for CUDA dikenal istilah *kernel*. *Kernel* di sini bukan merupakan *kernel* seperti yang ada pada sebuah sistem operasi. *Kernel* merupakan suatu fungsi yang akan dipanggil sejumlah N kali sesuai level paralelisme yang dimungkinkan. *Kernel* memungkinkan eksekusi secara paralel terhadap suatu fungsi [16]. Berikut adalah contoh pemanggilan *kernel* pada gambar 2.8.

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
int i = threadIdx.x;
C[i] = A[i] + B[i];
}
int main()
{
...
// Kernel invocation
VecAdd<<<1, N>>>(A, B, C);
}
```

Gambar2. 8 Kernel CUDA
(NVIDIA CUDA Programming Guide, 2009)

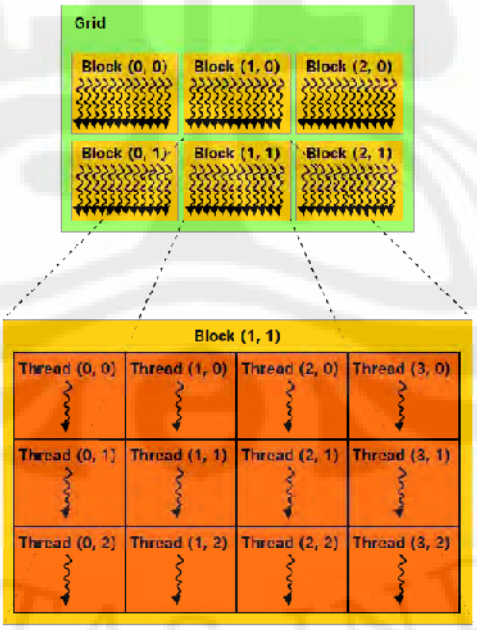
Bentuk tersebut merupakan bentuk umum dari program yang dibuat dalam C for CUDA. Program tersebut bukan merupakan program bahasa C biasa. Program tersebut harus di *compile* dengan menggunakan nvcc.

2.5.3.2 *Thread* dan Hirarkinya

Thread merupakan suatu *kernel* yang sudah dijalankan [15] - [18]. Pada Gambar 2.17 terdapat hirarki thread. Setiap *thread* dalam suatu blok memiliki *id* yang disebut **threadIdx** yang bisa dinyatakan dalam 1,2, maupun 3 dimensi. Dengan demikian suatu *problem* dengan domain eksekusi vektor, matriks atau medan, dapat diselesaikan dengan mudah.

Thread dalam satu blok dapat berbagi memori yang sama. Pada suatu batas tertentu, dapat diberikan perintah **__synctrheads()** untuk melakukan sinkronasi operasi paralel sebelum melanjutkan ke operasi selanjutnya.

Beberapa *block* tergabung dalam suatu *grid*. Setiap *block* dalam suatu *grid* memiliki *id* **blockIdx** 1-2 dimensi yang unik. Masing-masing *grid* dieksekusi secara independen dari *grid* lainnya, tidak boleh ada perubahan ketika dioperasikan secara paralel maupun secara serial. Jumlah *grid* yang dapat dieksekusi sesuai dengan jumlah *core* yang tersedia.

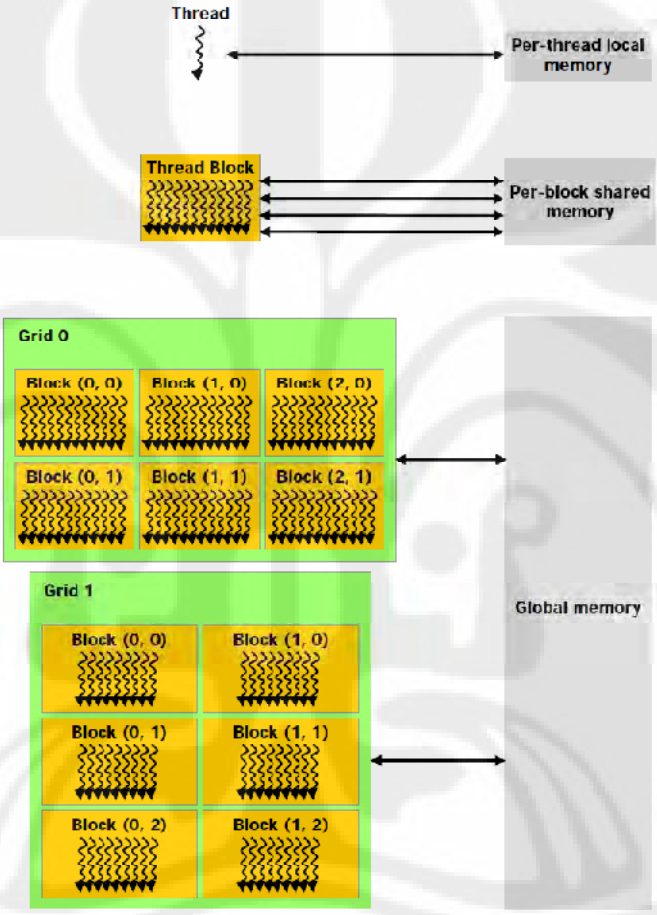


Gambar2. 9 Hirarki *Thread*

(NVIDIA CUDA Programming Guide, 2009)

2.5.3.3 Memori

Setiap *thread* memiliki memori lokal yang hanya dapat diakses oleh mereka sendiri. Setiap *block* memiliki memori yang di-*share* oleh semua *thread* yang ada di dalamnya. Dan setiap *grid* dapat mengakses memori utama sistem. Gambar 2.10 merupakan hirarki memori.



Gambar2. 10 Hirarki Memori
(NVIDIA CUDA Programming Guide, 2009)

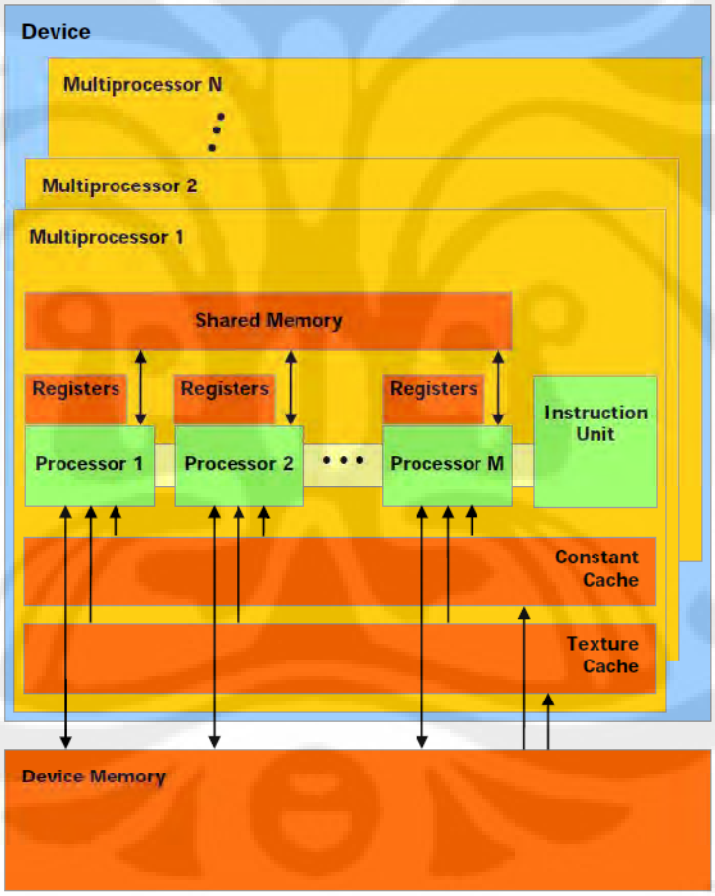
2.5.3.4 CUDA Device

Setiap *thread* dalam CUDA dianggap dieksekusi dalam divais CUDA yang terpisah [15]. Divais tersebut dianggap sebagai koprosesor (dalam hal ini GPU core) dari suatu *host* yang menjalankan program dalam bahasa C (dalam hal ini CPU).

Baik CPU maupun GPU dianggap memiliki memori yang terpisah. Satu-satunya cara untuk mentransfer data adalah memalui pemanggilan *kernel*.

Setiap divais CUDA memiliki beberapa multi prosesor (Gambar 2.11). Setiap multi prosesor memiliki 8 *scalar processor* (SP), 2 unit spesial untuk fungsi *trancedental* (sin cos, dan lain-lain), *instruction unit* dan memori.

Masing-masing SP memiliki 32 register. Setiap multi prosesor memiliki *shared memory* yang dapat diakses oleh seluruh SP dalam multi prosesor tersebut. Ada dua bagian tambahan dari *shared memory* tersebut yang merupakan *read only*. *Constant cache* dan *Texture cache*. *Texture chace* dapat diakses melalui *texture unit* yang mampu melakukan *addressing* secara lebih kompleks.



Gambar2. 11 Arsitektur Divais CUDA
(NVIDIA CUDA Programming Guide, 2009)

2.5.4 OpenCL

Open Computing Language (OpenCL) merupakan suatu *framework* terintegerasi yang digunakan untuk membuat program yang dapat dijalankan pada berbagai hardware. OpenCL dicetuskan oleh Apple, dan dikembangkan oleh Khronos Group. Spesifikasi OpenCL pertama kali dirilis pada Novenber 2008 [20].

Struktur eksekusi OpenCL terdiri dari suatu host yang terhubung ke satu atau lebih *compute devices* yang dapat berupa CPU, GPU, DSP, atau akselerator proses lainnya yang memungkinkan eksekusi secara data atau *task* parallel [19].

OpenCL secara umum terdiri dari set bahasa C99 untuk membuat *kernel*, dan *runtime* API yang dijalankan pada *host* untuk menjalankan dan mengatur platform. *Kernel* pada OpenCL mirip dengan bahasa C standar. Kernel dieksekusi pada *compute devices*, dalam hal ini CPU dan GPU. Sementara itu *runtime* API menyediakan pengaturan memori, *platform*, dan manajemen *thread* yang akan dieksekusi pada compute device. Berbeda dengan Brook+ dan C for CUDA yang merupakan *meta programming language* sehingga tidak memerlukan akses langsung untuk melakukan manajemen divais dan memori, OpenCL memungkinkan *developer* untuk melakukan manajemen memory dan thread secara manual [13], [17], [19].

Berdasarkan pengamatan, sejak spesifikasi OpenCL dirilis, hingga skripsi ini diajukan, masih sedikit jumlah *vendor* hardware yang menyediakan SDK versi *release*. Sebagian besar masih dalam tahap beta. Hal ini menyebabkan masih minimnya aplikasi dan komunitas yang menggunakan OpenCL Namun, sejak awal tahun 2010, telah ada beberapa SDK OpenCL yang sudah beredar dan mulai banyak ditemukan berbagai aplikasi yang mengeksploitasi OpenCL.

2.6 Program Pendukung Lainnya

SIMPLE-O dibangun dengan bahasa *scripting* PHP yang menggandeng *database* MySQL dan *scripting* HTML. *Web server* yang digunakan adalah Apache. Berikut adalah penjelasannya.

2.6.1 PHP (Hypertext Preprocessor)

PHP banyak digunakan oleh *programmer* dengan alasan memiliki kemiripan *syntax* dengan bahasa C/C++, Java, dan Perl. PHP merupakan bahasa *scripting open source* dengan menggunakan lisensi GPL (GNU Public Licence) yang dapat digunakan dengan gratis dan bebas. PHP biasa digunakan untuk membuat aplikasi web yang bersifat dinamis [21].

PHP menyerupai dengan HTML dimana kode-kode yang kita buat tidak perlu di-*compile* sebelum digunakan. Kode yang dibuat akan diproses saat diperlukan.

2.6.2 MySQL

MySQL merupakan *software* yang tergolong sebagai DBMS *(Database Management System)* yang bersifat *Open Source* [22]. MySQL awalnya dibuat oleh perusahaan konsultan bernama TcX yang berlokasi di Swedia. Selanjutnya pengembangan MySQL berada dibawah naungan perusahaan MySQL AB.

Pada tanggal 16 Januari 2008 Sun Microsystems, Inc mengumumkan aksi akuisisi terhadap MySQL AB sehingga menjadikan Sun sebagai salah satu perusahaan dengan produk *platform open source* terbesar seperti Java, OpenSolaris dan akhirnya MySQL.

Berselang setahun kemudian, tepatnya pada tanggal 20 April 2009 giliran Oracle melakukan akusisi terhadap Sun Microsystems.

2.6.3 Apache

Server HTTP Daemon Apache atau Server Web/WWW Apache adalah *web server* yang dapat dijalankan di banyak sistem operasi seperti Unix, BSD, Linux, Microsoft Windows dan Novell Netware serta platform lainnya, yang berguna untuk melayani dan memfungsikan situs web [23]. Protokol yang digunakan untuk melayani fasilitas web ini menggunakan HTTP. *Web Server* Apache berbasiskan *open source* dan mulai populer di internet semenjak tahun 1996.

2.6.4 CULA tools

CULA tools merupakan implementasi sebagian kecil fungsi yang terdapat LAPACK pada platform CUDA [24]. CULA menyediakan API untuk bahasa C. CULA tools basic tersedia tersedia secara gratis dan berisi implementasi BLAS level satu dan sebuah fungsi BLAS level 2, yaitu SGESVD yang digunakan dalam skripsi ini. Dalam versi *basic* ini CULA tools hanya menyediakan tingkat kepresisian *single* saja.

BAB 3
RANCANG BANGUN SIMPLE-O MENGGUNAKAN PLATFORM GPGPU

3.1 Rancang Bangun Software

Sistem ini akan disusun dalam 2 modul.

- Modul Web *Interface* dan LSA
- Modul SVD

Masing-masing modul memiliki fungsinya masing-masing. Berikut adalah penjelasan masing-masing modul.

3.1.1 Modul Web Interface dan LSA

Modul ini berfungsi untuk menyediakan *user interface* bagi pengguna dan melakukan pengolahan string jawaban dosen dan mahasiswa hingga siap untuk dilakukan operasi matematis.

Modul ini ditulis dalam bahasa PHP. Modul ini akan berkomunikasi dengan *database* MySQL untuk kepentingan pengarsipan data. *Script* HTML yang dihasilkan oleh modul ini akan dibuat tersedia secara *online* oleh *web server*httpd Apache.

```
if pil = Pilih Kata Bobot then
                    {bagian untuk menambah soal}
          [pilih matkul]
          [pilih soal]
          [input kata kunci bobot]
          [bentuk matriks]
          [Proses SVD]
          [Simpan nilai frobenius/cos Alfa yang sesuai]
```

Gambar 3.1 Algoritma penambahan soal SIMPLE-O asli

Dalam sistem ini, modul ini merupakan adaptasi dari program SIMPLE-O yang sudah ada. Hanya saja, proses perhitungan SVD tidak akan dilangsungkan

dalam PHP menggunakan JAMA, namun modul ini akan menggunakan beberapa metode tertentu untuk berkomunikasi dengan modul perhitungan SVD. Dilakukan 2 perubahan pada sistem SIMPLE-O yang sudah ada.

Pertama pada modul dosen. Ketika suatu soal telah selesai dibuat, dan kata bobot lebihnya juga telah dipilih oleh lebih dari 2 dosen, maka akan dilakukan blok algoritma pada Gambar 3.1.

Blok algoritma tersebut akan diubah menjadi sebagai berikut pada sistem yang baru:

```
if pil = Pilih Kata Bobot then
                    {bagian untuk menambah soal}
        [pilih matkul]
        [pilih soal]
        [input kata kunci bobot]
        [bentuk matriks]
        [komunikasi dengan Modul SVD]
        [Simpan nilai frobenius/cos Alfa yang sesuai]
```

Gambar 3.2 Algoritma penambahan soal SIMPLE-O yang telah dimodifikasi

Pada algoritma yang baru terlihat bahwa proses SVD digantikan komunikasi ke modul SVD. Komunikasi tersebut dapat berupa COM *invoke* pada sistem operasi windows untuk terhubung ke MATLAB, prosedur proc_open pada PHP untuk memanggil program lokal, maupun SSH untuk terhubung dengan *remote server* tempat dieksekusinya modul SVD.

Perubahan yang kedua berada pada modul mahasiswa. Tujuan perubahan yang kedua tetap sama, yaitu memindahkan proses SVD dari pemanggilan fungsi lokal di PHP menjadi proses komunikasi seperti yang dijelaskan di atas. Detail modifikasi pada program SIMPLE-O dapat dilihat pada lampiran.

3.1.2 Modul SVD

Sesuai dengan namanya, modul SVD berfungsi untuk melakukan perhitungan SVD secepat dan seefisien mungkin. Modul SVD akan

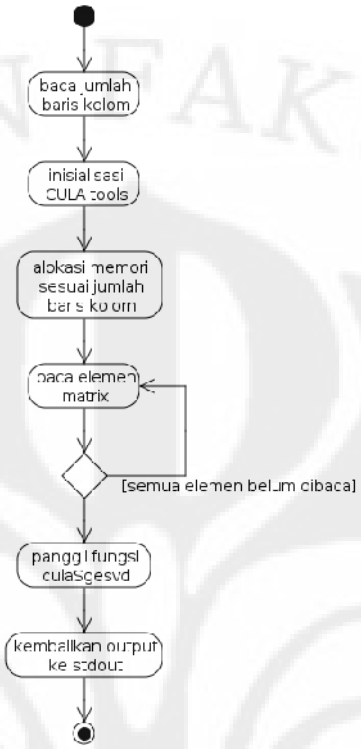
berkomunikasi dengan modul web interface dan LSA melalui *standard input* dan *output* agar dapat diterapkan di berbagai platform.

Secara umum, modul ini akan terdiri dari 2 bagian, bagian yang dieksekusi pada CPU dan yang dieksekusi pada GPU. Program pada CPU berfungsi untuk berkomunikasi dengan Modul *Web Interface*dan sebagai aplikasi *host* yang akan mengatur jalannya program pada GPU. Sementara program pada GPU murni untuk melakukan perhitungan aritmatika saja.

Implementasi program ini akan dilakukan pada bahasa C dengan beberapa pilihan API untuk ke GPGPU.Pilihan API yang pertama adalah CULA tools yang berjalan pada platform CUDA, sedangkan implementasi kedua menggunakan OpenCL.

3.1.2.1 CULA tools

Implementasi dengan menggunakan CULA tools cukup sederhana. API ini telah menyediakan fungsi culaSgesvd yang merupakan implementasi fungsi LAPACK SGESVD dalam platform CUDA. Modul SVD hanya berfungsi untuk menerima data melalui stdin, inisialisasi API, pemanggilan fungsi culaSgesvd, dan mengembalikan nilai keluaran fungsi tersebut ke stdout. Manajemen memori, divais, *kernel*, *thread* dan lain sebagainya sudah ditangani secara otomatis oleh API tersebut.Gambar 3.3 adalah UML *activity diagram* dari modul ini.



Gambar 3.3 Activity Diagram Modul SVD dengan CULA tools

3.1.2.2 OpenCL

Sementara itu untuk implementasi pada platform OpenCL, jauh lebih kompleks dibandingkan dengan implementasi pada CULA tools.Seperti yang telah dijelaskan pada bab2, OpenCL hanya menyediakan frame work untuk melakukan komputasi pada *compute device.*Algoritma perhitungan, manajemen divais, memori dan lain sebagainya dilakukan secara manual.

Algoritma yang digunakan berasal dari prorgam CUSVD oleh ZhangShu dan DouHeng pada tahun 2009 yang merupakan implementasi perhitungan nilai SVD dengan metode *single side jacobirotation* untuk matrix 512 x 512 dalam platform C for CUDA [25]. Algoritma tersebut dimodifikasi agar sesuai dengan kebutuhan modul ini kemudian diimplementasikan dalam bahasa C dengan API OpenCL. *Kernel* yang ada juga diterjemahkan kedalam kernel OpenCL.

Perlu diperhatikan bahwasannya algoritma ini hanya mampu menangani ukuran matriks tepat 512 x 512. Untuk ukuran matriks yang lebih kecil tetap dapat dilakukan perhitungan dengan mengisi sisa kolom dan baris yang kosong dengan

nilai 0. Haltersebut tetap dapat dilakukan dengan aman karena nilai 0 tidak akan mengubah hasil SVD.

Berikut *pseudocode* untuk algoritma modul ini dengan penekanan pada teknis implementasi daripada unsur matematisnya.

```
{
baca input tipe divais
baca ukuran matriks yang akan dihitung
dapatkan id platform
dapatkan id divais pada platform tsb yg sesuai dg tipe
divais masukan
buat cl_context untuk divais tersebut
buat command queue untuk konteks tersebut
buka source file untuk kernel dari file teks
baca source untuk kernel
buat program dari source tersebut
built program tersebut
buat kernel1 dari program tersebut untuk fungsi bjrot
buat kernel2 dari program tersebut untuk fungsi bjrot8
inisialisasi variabel untuk konstanta, iterasi dan ukuran
global work
alokasi memori untuk host
baca input elemen matriks ke lokasi memori yang sudah
disiapkan
siapkan buffer untuk memori pada divais
looping bi diagonalisasi
{
     set kernel2 argument
     eksekusi kernel2
}
Looping rotation 1
{
```

```
      Inner loop 1
      {
            set kernel1 argument
            eksekusi kernel1
      }
      Inner loop 2
      {
            set kernel1 argument
            eksekusi kernel1
      }
}
Looping rotation 2
{
      Inner loop 1
      {
            set kernel1 argument
            eksekusi kernel1
      }
      Inner loop 2
      {
            set kernel1 argument
            eksekusi kernel1
      }
}
looping rotation
{
      set kernel2 argument
      eksekusi kernel2
}
tunggu proses selesai
```

```
baca data dari buffer divais ke host
bentuk vektor s
sorting nilai singular dari yang terbesar
pembersihan memori
kembalikan nilai ke pemanggil
}
```

Gambar 3.4 Algoritma Modul SVD dengan OpenCL

Pada skripsi ini akan digunakan dua jenis implementasi OpenCL. Pertamayang dilakukan oleh AMD dan yang kedua NVIDIA. Untuk AMD, ada dua divais yang dapat digunakan, yaitu CPU dan GPU. Sementara untuk NVIDIA hanya GPU saja.

Untuk alasan performa dan ketepatan pengukuran waktu, modul SVD pada implementasi OpenCL dilakukan pada dedicated server yang terpisah dari *web server* yang menjalankan modul *web interface* + LSA. Kedua modul tersebut akan berkomunikasi melalui SSH.

3.2 Rancang Bangun Hardware

Untuk implementasi dengan API CULA tools, diperlukan hardware yang telah mendukung CUDA. Yaitu komputer yang memiliki GPU NVIDIA GeForce seri 8 atau lebih baru, beberapa seri Quadro dan Tesla.

Sementara itu, untuk menggunakan API OpenCL, diperlukan hardware sebagai berikut:

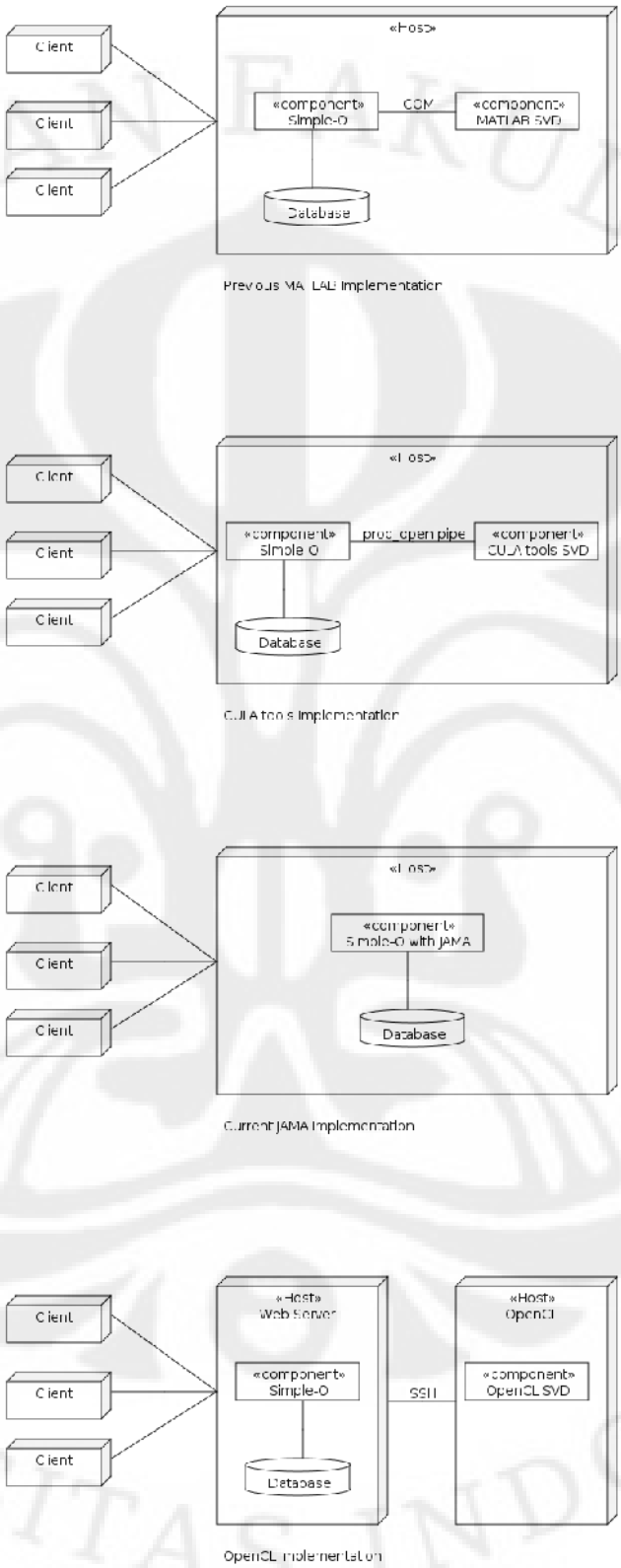
- OpenCL CPU : CPU x86/x86_64 yang mendukung instruksi SSE3, disarankan bermerek AMD. Hal tersebut dikarenakan implementasi OpenCL untuk CPU dibuat oleh AMD dengan mengeksploitasi kemampuan SSE3 pada prosesor tersebut. Semakin banyak core, semakin baik.
- OpenCL GPU : GPU yang telah mendukung OpenCL. Diantaranya GPU NVIDIA yang mendukung CUDA, GPU ATI seri 4000 atau lebih baru, ATI FireGL dan FirePro, Ati FireStream.GPU yang digunakan pada sistem ini akan berada pada kelas konsumen (ATI

Radeon maupun NVIDIA GeForce) untuk alasan ekonomi. GPU kelas profesional dan prosesor khusus GPGPU memiliki harga yang sangat tinggi sehingga menyebabkan pengembangan sistem menjadi tidak efisien dibandingkan dengan performance gain-nya.

- OpenCL ACCELERATOR : Akselerator khusus OpenCL seperti IBM Cell Broadband Engine atau DSP.

Memori utama (RAM) dalam sistem yang digunakan tidak harus berukuran besar. Ukuran blok data yang akan dikirim dari CPU ke GPU berukuran kecil, hanya beberapa ratus kilobita untuk setiap operasi SVD. Namun memori ini harus berjalan dengan kecepatan yang sangat tinggi untuk menghindari *bottleneck* dalam komunikasi CPU - GPU. Bus interkoneksi antara divais OpenCL dan host juga harus cepat, karena pertukaran data akan berlangsung dengan sangat intensif. PCIe 16 lanes Gen. 2.0 biasanya sudah cukup memadai walaupun masih sering terjadi bottleneck.

Khusus untuk implementasi dalam OpenCL, digunakan *dedicated webserver* khusus menangani modul *web interface* + LSA dan server terpisah untuk menangani implementasi OpenCL pada modul SVD.Gambar 3.5 menggambarkan sistem secara keseluruhan.



Gambar 3.5 Desain sistem secara keseluruhan

BAB 4
IMPLEMENTASI
PENGUJIAN DAN ANALISA PERFORMA SIMPLE-O MENGGUNAKAN PLATFORM GPGPU

4.1 Implementasi

Seperti yang telah dijelaskan pada bab 3, sistem akan dibentuk dari dua modul. Modul web interface + LSA dan modul SVD. Berikut adalah penjelasan implementasi pada masing-masing modul.

4.1.1 Implementasi Modul SVD dengan API CULA tools

Algoritma yang telah dijelaskan pada bab 3 dimplementasikan dalam bahasa C. Kode sumber yang dibuat dapat dilihat pada lampiran. Berikut adalah cuplikan kode sumber yang digunakan untuk inisialisasi CULA tools, memanggil fungsi culaSgesvd dan mengakhiri sesi cula.

```
...
status = culaInitialize();
...
status = culaSgesvd(jobu, jobvt, M, N, A, LDA, S, U, LDU,
VT, LDVT);
...
culaShutdown();
...
```

Gambar 4.1Cuplikan kode modul SVD dengan API CULA tools

Kode tersebut kemudian di *compile* menggunakan Visual Studio 2008 dan di *link* kepada *static* library cula.lib yang merupakan compile time library untuk API CULA tools. Setelah *compiling* dan *linking*, maka akan dihasilkan file .exe yang merupakan file *executable* yang dapat dipanggil dari command line di Windows. File .exe yang dihasilkan kemudian diletakkan pada folder C:\culasvd\ bersama dengan *dynamic library* yang dibutuhkan. Maka modul SVD telah siap untuk dipanggil.

Modul ini akan menerima *input/output* melalui *stdio*. *Input* yang dibaca berupa ukuran matriks dan nilai matriksnya. Sementara *output* yang dihasilkan adalah *string* yang berisi seluruh nilai SVD matrix tersebut dengan dipisahkan oleh spasi. Seperti yang telah dijelaskan pada bab 3, modul web interface + LSA akan menggunakan fungsi proc_open() untuk membuka file ini melalui *command line*.

Modul dijalankan pada sistem sebagai berikut:

4.1.1.1 Hardware

Berikut adalah konfigurasi hardware yang digunakan:

GPU : NVIDIA GeForce 8400GS DDR2 512MB

- Shader clock 1400MHz
- CUDA Compute capability: 1.1
- Multi Prosesors : 1; 8 cores

Host :

- CPU : AMD Phenom 9500 @ 2.2 GHz
- Memori : 2048 MB DDR2 Dual Channel
- GPU interface : PCIe 16x Gen. 2.0
- Chipset : AMD 770 + SB600
- HDD : 160 GB 7200RPM

Spesifikasi lengkap terdapat pada lampiran.

4.1.1.2 Software

Berikut adalah spesifikasi software yang digunakan:

- OS : Microsoft Windows XP SP3 32-bit (build 2600)
- GPU Driver : Forceware 6.14.11.9713 (3-15-2010)
- CUDA : 3.0.1
- CULA tools : 1.3a Win32
- IDE : Microsoft Visual Studio 2008 v 9.0.21022.8 RTM
- Compiler : MSVC 15.00.21022.08 for 80x86
- Ketergantungan tambahan: cula.dll, cublas.dll, cudart.dll
- Compile time library : cula.lib

- Web server : Apache/2.2.14 (Win32)
- PHP : 5.3.1
- MySQL : 5.1.41

4.1.2 Implementasi Modul SVD dengan OpenCL

Serupa dengan pembuatan modul SVD yang menggunakan CULA tools, untuk implementasi menggunakan OpenCL algoritma yang sudah ada ditulis dalam bahasa C. Kode sumber asli dapat dilihat pada bagian lampiran. Berikut cuplikan kode sumber untuk implementasi OpenCL.

```
...
//mencari platform
  clGetPlatformIDs( 1, &platform, NULL );
//mencari device
  clGetDeviceIDs( platform, device_type,
                            1,
&device,
                            NULL);
//membuat konteks
  cl_context context = clCreateContext( NULL,
                                        1,
&device,
                                        NULL, NULL, NULL);
//membuat queue untuk konteks
  cl_command_queue queue = clCreateCommandQueue( context,
                                                 device,
                                                 0, NULL );
//buat program untuk kernel
  cl_program program = clCreateProgramWithSource( context,
                                                  1,
```

```
                                                    (const
char**)&source2,
                                                    NULL, NULL
);
//compile program kernel tersebut
  clBuildProgram( program, 1, &device, NULL, NULL, NULL );
  cl_kernel kernel = clCreateKernel( program, "bjrot", NULL
);
  cl_kernel kernel2 = clCreateKernel( program, "bjrot8",
NULL );
//load memori host ke divais
  cl_mem d_u = clCreateBuffer( context,
                               CL_MEM_READ_WRITE |
CL_MEM_COPY_HOST_PTR,
                               num2 * sizeof(cl_float),
                               unit, NULL );
...
//contoh pengesetan argumen ke kernel
        clSetKernelArg(kernel2, 0, sizeof(d_w), (void*)
&d_w);
...
//eksekusi kernel
     clEnqueueNDRangeKernel( queue,
     kernel2,
     1,
     NULL,
     &grid,
     NULL, 0, NULL, NULL);
...
//menunggu kernel selesai dieksekusi
```

```
  clFinish( queue );
//pembacaan memori divais ke host
  w = (cl_float *) clEnqueueMapBuffer( queue,
                                       d_w,
                                       CL_TRUE,
                                       CL_MAP_READ,
                                       0,
                                       num2 *
sizeof(cl_float),
                                       0, NULL, NULL, NULL
);
...
```

Gambar 4.2Cuplikan kode modul SVD dengan OpenCL

Berikut adalah cuplikan kode kernel yang digunakan.

```
...
//contoh deklarasi kernel
__kernel  void bjrot(__global float * d_w_o, __global float
* d_w_i,__global float * d_u_o, __global float * d_u_i, int
level, int offset)
{
... // kernel code
}
...
```

Gambar 4.3 Cuplikan kernel OpenCL yang digunakan

Kode tersebut kemudian di *compile* menjadi sebuah *object files* untuk kemudian di*link* dengan library yang dibutuhkan menjadi suatu file *executable binary*. File tersebut kemudian dipindahkan pada lokasi yang sama dengan *http root* untuk mempermudah operasi.

Ketika file *binary* tersebut dipanggil, maka ia akan mencari beberapa file *shared object* yang berisi implementasi OpenCL dan juga OpenCL ICD yang

vendor spesifik. Proses pemanggilan melalui PHP sama dengan implementasi menggunakan CULA tools.

Modul dijalankan pada sistem sebagai berikut:

4.1.2.1 Hardware

Berikut adalah konfigurasi hardware yang digunakan:

GPU 1 : ATI/AMD Radeon HD 5850 GDDR5 1024 MB

- Core Clock 725MHz
- Memory Clock1000MHz
- Usable Memory : 256MB
- Compute Units: 18; 1440 cores

GPU 2 : NVIDIA GeForce 8400 GS DDR2 512 MB

- Core Clock 567 MHz; Shader Clock 1400 MHz
- Memory Clock 400 MHz
- Usable Memory : 511MB
- Compute Units: 1; 8 cores

CPU 1 : AMD Phenom II X6 1055T DDR3 8192 GB

- Core Clock 3200 MHz
- Memory Clock 1200 MHz
- Usable Memory : 1024MB
- Compute Units: 6; 6 cores

CPU 2 : AMD Phenom 9500 DDR2 2048 GB

- Core Clock 2200 MHz
- Memory Clock 667 MHz
- Usable Memory : 1024MB
- Compute Units: 4; 4 cores

Host 1 (untuk GPU 1 dan CPU 1):

- Memori : 8192 MB DDR3 Dual Channel
- GPU interface : PCIe 16x Gen. 2.0
- Chipset : AMD 890FX + SB850
- HDD : SSD 128 GB

Host 2 (untuk GPU 2 dan CPU 2):

- Memori : 2048 MB DDR2 Dual Channel
- GPU interface : PCIe 16x Gen. 2.0
- Chipset : AMD 770 + SB700
- HDD : RAID 0 2x 160 GB 7200RPM

Web server :

- CPU : Intel Pentium 4 640 3.2 GHz
- Memori : 1024 MB DDR2 Dual Channel
- HDD : 320 GB 7200RPM

Spesifikasi lengkap terdapat pada lampiran.

4.1.2.2 Software

Berikut adalah spesifikasi software yang digunakan:

- OS : Linux 2.6.32-22-generic #36-Ubuntu SMP x86_64
- GPU Driver : NVIDIA 195.36.15;ATI 8.723
- GPGPU Platform :CUDA 3.0.1; ATI-Stream-v2.0.1
- OpenCL 1.0
- Editor : gedit 2.30.2
- Compiler : g++ (Ubuntu 4.4.3-4ubuntu5) 4.4.3
- Web server : Apache/2.2.14 (Ubuntu)
- PHP : 5.3.2-1ubuntu4.2
- MySQL : 5.1.41-3ubuntu12.1

4.1.3 Modul *Web Interface* + LSA

Pada bab 3 dijelaskan bahwa modul *web interface* + LSA akan menggunakan versi asli dari Simple-O dengan beberapa perubahan. Pada sub bab ini akan dibahas perubahan apa saja yang dilakukan agar Simple-O dapat terhubung ke Modul SVD.

Seperti yang telah dijelaskan dalam bab 2 dan 3, dalam Simple-O penggunaan fungsi SVD dilakukan pada dua lokasi. Pertama, pada modul dosen untuk menambah soal. Yang kedua pada modul mahasiswa ketika akan menghitung nilai.

Berikut adalah kutipan bagian program PHP yang identik dari file dosen_soaladd4.php dan mhs_hitungscore.php yang sama-sama mengandung pemanggilan fungsi SVD pada JAMA:

```
...
$matrix = substr($matrix, 0, -1);
$aku = explode(';', $matrix);
for ($i = 0; $i < count($aku); $i++) {
$aku[$i] = trim($aku[$i]);
   $aku[$i] = explode(' ', $aku[$i]);
   $jumjawab += array_sum($aku[$i]);
}
if ($idx < 2) {
   $MTX = new Matrix($aku);
   $MTXdata = $MTX->svd();
   $S_matrix = $MTX->normF();
}$total_nilai1 =
...
```

Gambar 4.4 Cuplikan kode asli SIMPLE-O untuk perhitungan SVD

Pada kode php di gambar 4.4, variabel matrix berisi array yang akan dihitung nilai SVD nya dengan format penulisan antar kolom dipisahkan oleh spasi dan antar baris dipisahkan oleh titik koma. Kemudian, isi variabel matrix dipindah ke array aku dengan format *row major*. Selanjutnya variabel aku dibuat suatu *object* Matrix yang bernama MTX. Kemudian MTX di hitung SVD nya dan di normal frobenius kan dengan menggunakan JAMA. Untuk lebih jelas, pada lampiran terdapat kode php lengkap dari kedua file diatas.

Berikut adalah penggalan kode yang telah dimodisikasi untuk pemanggilan modul SVD dengan CULA tools:

```
include_once "../lib/strlib.php";

...

culasvd(1,$bykkunci*$idx,str_replace(';','',$matrix));

...
```

Gambar 4.5 Cuplikan kode SIMPLE-O yang telah dimodifikasi

Yang akan memanggil fungsi berikut dalam strlib.php

```
...
function culasvd($x, $y, $input)
{
   $input=$x.' '.$y.' '.$input;
   $descriptorspec = array(
      0 => array("pipe", "r"),
      1 => array("pipe", "w"),
      2 => array("file", "error-output.txt", "a")
   );
   $cwd = 'C:/culasvd/';
   $process = proc_open('TestCula.exe', $descriptorspec,
$pipes, $cwd);
   fwrite($pipes[0], $input);
   fclose($pipes[0]);
   $progout2 = stream_get_contents($pipes[1]);
   fclose($pipes[1]);
   $return_value = proc_close($process);
   return $progout2;
}
...
```

Gambar 4.6 Fungsi culasvd pada strlib.php

Pada Gambar 4.5 terlihat bahwa pada proses diatas matriks terlebih dahulu dijadikan 1 baris untuk menghilangkan keharusan untuk melakukan normalisasi. Fungsi culasvd memiliki 3 argument. Urutan argument tersebut adalah jumlah kolom matriks, jumlah baris matriks dan isi matriks.

Pada fungsi culasvd terlihat bahwa file TestCula.exe dieksekusi dengan *pipe input* dari variabel input dan keluarannya di masukkan ke variabel progout2. Nilai progout2 kemudian akan dikembalikan ke pemanggil fungsi culasvd.

Sementara itu untuk OpenCL, digunakan fungsi-fungsi SSH pada modul libssh2-php untuk berkomunikasi dengan *dedicated* OpenCL server:

```
...
function clsvd($device, $x, $y, $input)
{
$input=$device.' '.$x.' '.$y.' '.$input.' ';
   $con=ssh2_connect('CLSERV',22);
   ssh2_auth_password($con, 'cluser', 'clpass');
   $stream=ssh2_exec($con,'/var/www/cliseng3bin');
   fwrite( $stream, $input.PHP_EOL);
   stream_set_blocking($stream, true);
   return fgets($stream);
}
...
```

Gambar 4.7 Fungsi clsvd pada strlib.php

CLSERV merupakan *host name* dari *dedicated* OpenCL server. Sementara program yang dieksekusi berada di /var/www/cliseng3bin. Kemudian input dikirim melalui fungsi fwrite. Dilanjutkan dengan penghentian *stream* dan pemngembalian stream yang di dapat ke pemanggil fungsi clsvd.

Dalam penggunaannya, perbedaan yang signifikan hanya pada 1 argument tambahan untuk inputnya. Argumen pertama digunakan untuk memilih divais yang akan digunakan, 0 untuk CPU dan 1 untuk GPU. Argument kedua hingga keempat mengikuti implementasi dari CULA tools.

4.2 Pengujian

Untuk mengetahui performa dari sistem ini, maka dilakukan dua jenis pengujian. Pengujian pertama akan melihat kemampuan dari sistem untuk menangani jawaban dengan jumlah kata yang sangat banyak. Sementara pengujian kedua digunakan untuk mengetahui kemampuan sistem dalam menghadapi banyak permintaan grading secara serempak.

Pada saat pengujian, faktor-faktor selain yang disebutkan dianggap ideal. Kondisi jaringan, utilisasi CPU, penggunaan memori oleh program lain dan lain sebagainya telah diminimalkan. Untuk menjaga keadilan dalam pengujian, maka seluruh algoritma diset ke *single precision* termasuk untuk OpenCL dan MATLAB. Hal ini dikarenakan JAMA hanya mendukung kalkulasi *single precision* dan CULA tools hanya menyediakan *double precision* pada versi berbayarnya, CULA tools basic yang digunakan terbatas pada *single precision.*

Berikut akan dijelaskan masing-masing skenario pengujian.

4.2.1 Pengujian Ukuran Matriks

Dari penjelasan mengenai Simple-O pada bab2, dapat disimpulkan bahwa jika jumlah kata yang akan dihitung nilai SVD-nya semakin banyak, maka akan semakin banyak pula jumlah elemen dalam matriks yang akan dihitung. Dengan semakin besarnya ukuran matriks, maka semakin banyak perhitungan yang harus dilakukan sehingga akan menjadikan proses SVD menjadi lambat. Oleh karena itu pengujian berikut akan mengukur kemampuan sistem dalam menangani jumlah kata yang sangat banyak dan melihat dampaknya terhadap waktu eksekusi.

Untuk melakukan pengujian, disusun suatu *script* yang bernama drone1.php beserta databasenya. *Script*ini bertugas untuk memberikan simulasi input berupa matriks mulai dari yang berukuran 1x1 hingga maksimum 512x512 kemudian mengukur waktu eksekusi masing-masing modul dan mencatatnya dalam database. Matriks yang diberikan dibuat semirip mungkin dengan kondisi nyata pada LSA, yaitu merupakan *sparse matrix* dengan nilai antara 0-4. Waktu eksekusi yang dicatat, dihitung mulai dari fungsi perhitungan SVD dipanggil melalui PHP hingga nilai tersebut dikembalikan ke pemanggilnya.

Pengujian ini hanya terbatas hingga ukuran matriks 512x512. Pembatasan ukuran ini disebabkan oleh beberapa hal, terutama adalah keterbatasan algoritma yang digunakan pada implementasi OpenCL. Algoritma yang digunakan hanya mampu menangani matriks berukuran 512x512. Selain itu implementasi SVD menggunakan JAMA masih berbasis operasi serial, sehingga akan memakan waktu eksekusi yang sangat lama sekali. Dari hasil pengujian awal, matriks 1024x1024 pada JAMA membutuhkan waktu lebih dari 8000 detik sementara implementasi dalam CULA tools hanya membutuhkan waktu 21 detik.

4.2.2 Pengujian Permintaan Serempak

Pada pengujian ini dilakukan permintaan perhitungan untuk matriks berukuran sedang (16x16) dalam jumlah banyak sekaligus. Pengujian ini mensimulasikan banyak permintaan grading yang terjadi dalam waktu yang bersamaan.

Pengujian dilakukan dengan membuat *script*drone2.php yang merupakan modifikasi dari drone1.php. Prinsip kerja drone2.php sama seperti drone1.php, hanya saja ukuran matriks yang diuji sama yaitu 16x16 dan dilakukan secara berulang-ulang. *Script* drone2.php ini akan dipanggil secara jamak untuk mensimulasikan banyak user. Jumlah perhitungan yang berhasil diselesaikan setiap detiknya akan dicatat dalam database.

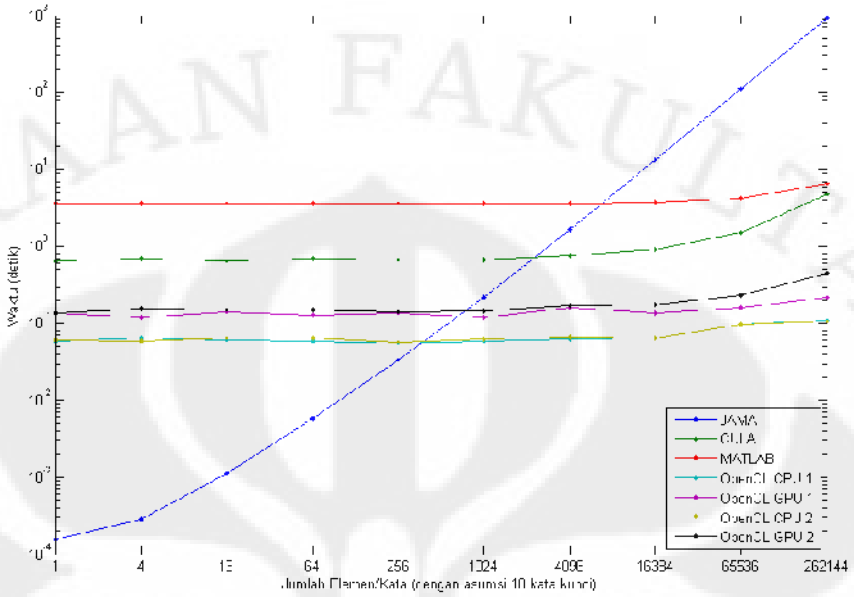
4.3 Hasil dan Analisa

4.3.1 Hasil dan Analisa Uji Ukuran Matriks

Setelah dilakukan pengujian sebanyak 10 kali, kemudian dirata-ratakan, maka diperoleh data sebagai berikut:

Tabel4.1Perbandingan kecepatan kalkulasi setiap implementasi

	Waktu Eksekusi dalam detik						
Ukuran matriks	JAMA	CULA	MATLAB	OLC CPU1	OCL GPU1	OCL CPU2	OCL GPU2
1	0.000159025	0.651933908	3.566739082	0.059289932	0.13754034	0.060971932	0.137735897
2	0.000287056	0.681726933	3.585502863	0.063547134	0.120116234	0.058532821	0.153912235
4	0.001126051	0.65737915	3.589609861	0.060782909	0.140929699	0.064553506	0.148968381
8	0.005867958	0.696954966	3.568249941	0.058986902	0.129019737	0.063674029	0.150076969
16	0.033119917	0.669210196	3.572027922	0.055332899	0.137988091	0.057263239	0.139223556
32	0.220705986	0.671563148	3.607639074	0.059155941	0.120699883	0.061939534	0.144218723
64	1.649186134	0.756123066	3.625951052	0.062973976	0.160515785	0.065599807	0.17104088
128	13.41542387	0.926187992	3.736470938	0.065312147	0.136774063	0.064148199	0.175670681
256	109.4701259	1.524574995	4.225842953	0.096190929	0.157886028	0.097870091	0.228486875
512	933.4352429	4.722379923	6.505066156	0.108977079	0.218689919	0.105425896	0.452175016



Gambar 4.8 Grafik perbandingan kecepatan kalkulasi setiap implementasi

Tabel4.2 Peningkatan kecepatan relatif terhadap JAMA

	Peningkatan kecepatan relatif terhadap JAMA					
Ukuran matriks	CULA	MATLAB	OLC CPU1	OCL GPU1	OCL CPU2	OCL GPU2
1	0.000243928	4.45856E-05	0.002682162	0.001156208	0.00260817	0.001154566
2	0.000421072	8.00602E-05	0.004517213	0.002389818	0.004904188	0.001865063
4	0.00171294	0.000313697	0.018525782	0.007990161	0.017443684	0.007558993
8	0.008419422	0.001644492	0.099479	0.045481088	0.092156223	0.039099657
16	0.049491052	0.009272021	0.598557413	0.240020112	0.578380085	0.237890181
32	0.328645171	0.061177402	3.730918353	1.828551788	3.563249031	1.530355991
64	2.181108088	0.454828571	26.18837551	10.27429254	25.14010658	9.6420583
128	14.48455819	3.590399629	205.4047288	98.08456052	209.1317307	76.36689173
256	71.80370022	25.90492054	1138.050402	693.3490385	1118.524824	479.1090342
512	197.6620387	143.4935818	8565.427227	4268.304864	8853.946532	2064.322908

Lebih lambat | Hampir sama | Lebih cepat

Berikut akan dijelaskan masing-masing implementasi.

4.3.1.1 JAMA

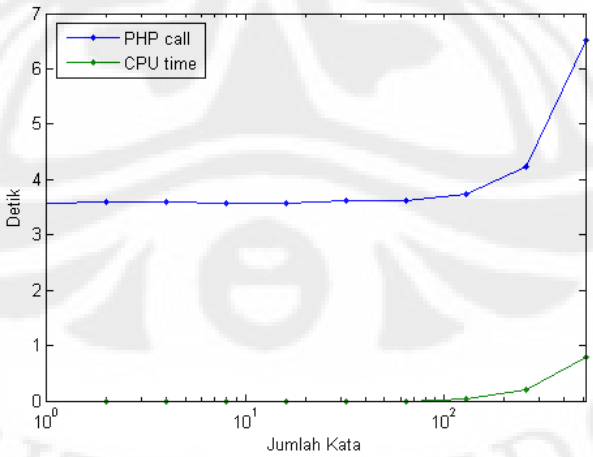
Dari Gambar 4.8 terlihat bahwa hampir semua jenis implementasi memiliki waktu eksekusi yang tidak terlalu dipengaruhi oleh ukuran matriks hingga 4096 kata kecuali implementasi menggunakan JAMA.JAMA sangat efisien untuk ukuran matriks yang kecil. Bahkan implementasi ini berjalan dengan waktu dibawah 1 mili detik untuk perhitungan LSA dibawah 16 kata.Hal ini

sangat jelas alasannya, JAMA merupakan *library* yang diimplementasikan langsung dalam PHP, sehingga pemanggilan fungsi ini sama sekali tidak memiliki *overhead*. Namun waktu yang diperlukan meningkat secara eksponensial untuk setiap penambahan jumlah elemen. Dari Tabel 4.1 untuk matrix 512x512, implementasi dalam JAMA bahkan menghabiskan waktu hingga 933 detik untuk melakukan perhitungan, hampir 200 kali lebih lambat dari implementasi CULA dan lebih dari8500 kali lebih lambat dari OpenCL. Peningkatan waktu ini disebabkan oleh algoritma yang digunakan adalah algoritma yang murni sekuensial.

4.3.1.2 MATLAB

Pada implementasi yang menggunakan MATLAB, terlihat bahwa overhead waktu yang terjadi sangat besar sekali. Bahkan untuk matriks 1x1 pun dari Tabel 4.1 memerlukan waktu hingga 3,5 detik dan meningkat hingga 6 detik untuk matriks 512x512.

Dari hasil penyelidikan diketahui bahwa overhead waktu disebabkan oleh mekanisme pemanggilan MATLAB dari PHP melalui Component Object Model (COM) pada sistem operasi Windows.Melalui pengamatan sederhana, pemanggilan program melalui COM seringkali memiliki *overhead* waktu yang cukup besar. Hal tersebut tergambar pada gambar berikut.



Gambar 4.9 Grafik perbandingan waktu pemanggilan fungsi MATLAB dari PHP dan waktu eksekusi MATLAB yang sesungguhnya

Pada gambar di atas, CPU time merupakan waktu yang sesungguhnya yang digunakan matlab dalam melakukan eksekusi perhitungan SVD. Sementara PHP call merupakan waktu sejak fungsi dipanggil pada PHP hingga hasil SVD diterima kembali oleh PHP. Pada grafik terlihat sekitar 90% waktu dihabiskan oleh COM interface. Oleh karena itu, untuk implementasi SVD selanjutnya tidak akan menggunakan COM inteface pada sistem operasi Windows.

Tabel4.3Perbandingan waktu pemanggilan fungsi MATLAB dari PHP dan waktu eksekusi MATLAB yang sesungguhnya (nilai 0 adalah nilai dibawah 1 mili detik yang tidak terukur oleh Windows)

	Waktu dalam detik	
Ukuran matriks	PHP call	MATLAB CPU time
1	3.566739082	0
2	3.585502863	0
4	3.589609861	0
8	3.568249941	0
16	3.572027922	0
32	3.607639074	0
64	3.625951052	0
128	3.736470938	0.03125
256	4.225842953	0.2031
512	6.505066156	0.3969

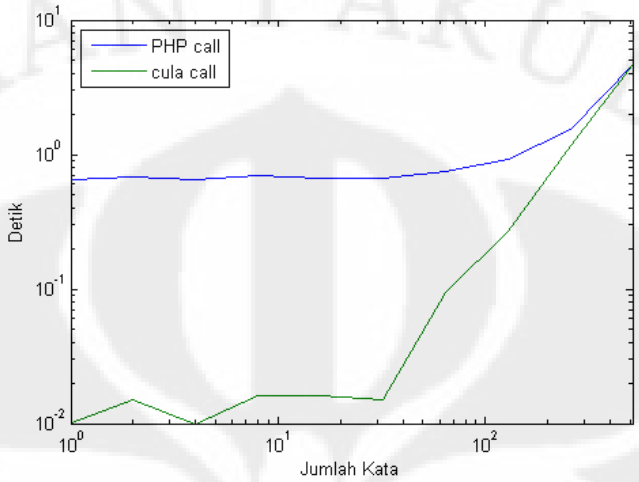
4.3.1.3 CULA tools pada Platform CUDA

Selanjutnya bila diperhatikan pada Tabel 4.1 untuk implementasi menggunakan CULA tools, overhead terlihat lebih rendah. Untuk matriks berukuran kecil waktu eksekusi hanya sekitar 0.65 detik. Sementara itu untuk matriks yang lebih besar mencapai 4.7 detik. Waktu tersebut masih jauh lebih cepat dari JAMA dan sedikit mengungguli MATLAB.

Untuk lebih mengetahui lama *overhead* yang terjadi pada saat pemanggilan fungsi culasvd pada PHP, maka data tersebut dibandingkan dengan waktu eksekusi culaSgesvd pada API CULA tools. Berikut adalah grafik perbandingannya.

Walaupun *overhead* pada matriks ukuran kecil mencapai 99% waktu panggil PHP, namun besarannya masih lebih rendah dari pemanggilan MATLAB. Sementara itu, pada matriks ukuran besar, overhead semakin tidak nampak, hal tersebut dikarenakan pada matriks besar operasi matematis semakin intensif sehingga memerlukan waktu lebih lama dalam eksekusinya, sehingga waktu

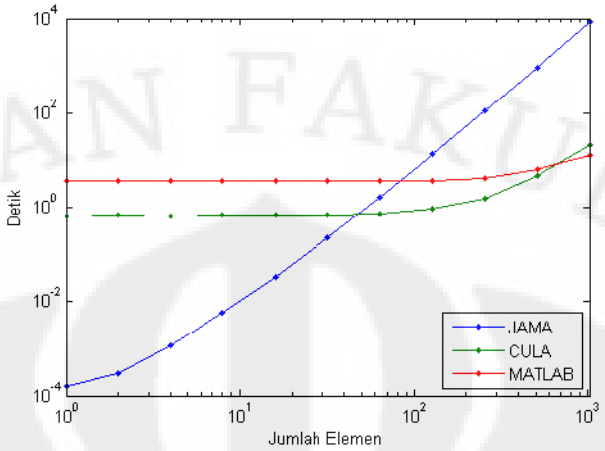
tunggu yang muncul karena fungsi proc_open pada PHP dapat dimanfaatkan dengan baik.



Gambar 4.10 Grafik perbandingan waktu pemanggilan fungsi culasvd dari PHP dan waktu eksekusi modul SVD yang sesungguhnya

Dari ketiga implementasi ini, dapat ditarik kesimpulan sementara bahwa implementasi SVD dalam CULA menggabungkan keunggulan JAMA yang rendah *overhead* untuk matriks kecil, dan MATLAB yang memiliki waktu eksekusi tidak eksponensial untuk matriks berukuran besar.

Sebelum membahas mengenai OpenCL, pada gambar 4.8terlihat suatu hal yang menarik. Bila diperhatikan, grafik waktu eksekusi CULA menaik lebih tajam dibandingkan dengan MATLAB pada ukuran matriks 256x256 ke 512x512. Diduga pada matrix dengan ukuran lebih besar kondisinya dapat berbalik. Berikut adalah grafik perbandingan ketiga platform selain OpenCL hingga matrix ukuran 1024x1024.



Gambar 4.11 Grafik perbandingan waktu pemanggilan fungsi MATLAB, CULA dan JAMA untuk matriks hingga ukuran 1024x1024

Ternyata dugaan tersebut terbukti. Pada matriks dengan ukuran 1024x1024 terlihat bahwa waktu eksekusi CULA telah melampaui MATLAB. Setelah dilakukan analisa, diperkirakan ada 2 kemungkinan penyebab. Pertama, algoritma yang digunakan MATLAB memang sudah baik sekali. MATLAB mampu mengeksploitasi banyak core pada CPU. Kedua, penyebabnya muncul dari *hardware* itu sendiri.

Tabel4.4Waktu eksekusi pada matriks berukuran 1024x1024

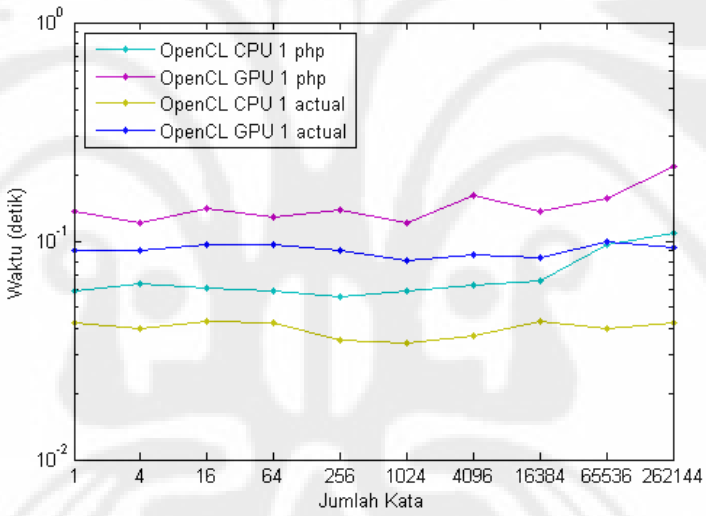
	JAMA	CULA	MATLAB
1024x1024	8450	21.36	13.3

Hardware yang digunakan oleh CULA tools adalah GeForce 8400GS dengan 8 unit pemrosesan data. Secara kasar dapat dihitung kemampuan pemrosesannya dari perhitungan sederhana berikut. *Clock*900MHz dikalikan dengan 8 unit *float* prosesor, sehingga didapat 7.2 GFlops. Sementara itu, MATLAB menggunakan prosesor Phenom II x6 dengan 6 core*clock* 3200 MHz, didapat teoritis maksimum 19.2 GFlops (Efektif hanya sekitar 8GFlops dari hasil benchmark). Sehingga *raw* computing power yang dapat digunakan oleh MATLAB lebih besar. Dengan *raw computing power* yang lebih besar, maka ketika menghitung matriks lebih besar, waktu *overhead*untuk pemanggilan program dan lain sebagainya menjadi tertutup.

4.3.1.4 OpenCL

Dari Tabel 4.2, implementasi pada OpenCLlebih lambat 1,6 kali dari JAMA pada kondisi normal. Namun kondisi berbalik saat kasus ekstrim, OpenCL 8565 kali lebih cepat dari implementasi JAMA saat menghitung matriks berukuran 512x512.

Jika diperhatikan, terlihat bahwa overhead waktu yang terjadi tidak terlalu besar, tetapi cenderung untuk konstan disemua ukuran matriks. Hal tersebut dikarenakan algoritma yang digunakan sebenarnya memiliki ukuran tetap, yaitu 512x512. Walaupun data yang dihitung lebih sedikit dari itu, kalkulasi yang dilakukan tetap berada pada 512x512 elemen matriks. Hal tersebut akan semakin tampak pada grafik berikut.



Gambar 4.12 Grafik perbandingan waktu pemanggilan fungsi clsvd dari PHP dan eksekusi modul SVD yang sesungguhnya

Gambar diatas merupakan grafik perbandingan antara waktu eksekusi platform OpenCL secara langsung dan pemanggilan melalui PHP pada sistem operasi linux. Dari grafik diatas dapat disimpulkan beberapa hal sebagai berikut.

Pertama, waktu untuk eksekusi langsung tanpa melalui php tidak berubah terhadap ukuran matriks. Hal ini jelas disebabkan oleh algoritma yang digunakan memiliki ukuran matriks yang tetap, yaitu 512x512.

Tabel4.5Perbandingan waktu pemanggilan fungsi clsvd dari PHP dan eksekusi modul SVD yang sesungguhnya

	Actual		PHP call	
UkuranMatriks	CPU1	GPU1	CPU1	GPU1
1x1	0.042	0.090	0.059	0.137
2x2	0.040	0.090	0.063	0.120
4x4	0.043	0.095	0.060	0.140
8x8	0.042	0.095	0.058	0.129
16x16	0.035	0.090	0.055	0.137
32x32	0.034	0.081	0.059	0.120
64x64	0.037	0.087	0.062	0.160
128x128	0.043	0.086	0.065	0.136
256x256	0.040	0.098	0.096	0.157
512x512	0.042	0.092	0.108	0.218

Kedua, dibandingkan dengan implementasi pada platform lainnya, kecuali JAMA, *overhead*waktu OpenCL yang muncul lebih rendah. Diperkirakan penyebabnya adalah digunakannya platform linux yang menurut pengamatan biasanya memiliki *latency* yang lebih rendah untuk pemanggilan program melalui *shell* dibandingkan Windows melalui cmd.exe.

Ketiga, pada Gambar 4.12 diatas terlihat jika waktu pemanggilan melalui PHP tetap semakin meningkat seiring peningkatan jumlah komponen walaupun waktu eksekusi aktualnya tetap sama. Diperkirakan hal tersebut disebabkan oleh proses pengiriman data melalui *stdio*pada modul SVD maupun pengiriman data melalui SSH. Hal tersebut dibuktikan dengan melakukan pengukuran waktu yang diperlukan PHP untuk mengiriman data *dummy*melalui SSH ke sebuah program *loop back* yang memeneruskan *stdin* langsung menuju *stdout.* Waktu yang diperoleh memiliki karakteristik yang sama dengan overhead waktu pada implementasi OpenCL ini. Semakin banyak nilai yang ditransfer, maka semakin lama waktu yang diperlukan.

Yang terakhir, terlihat bahwa implementasi pada CPU ternyata lebih cepat dibandingkan implementasi pada GPU. Pada bab 2 dijelaskan bahwa sebuah program OpenCL terdiri *kernel* yang dieksekusi pada suatu *compute device* dan sebuah program*host* yang berfungsi untuk melakukan pengaturan terkait eksekusi *kernel*.*Kernel* tersebut dapat dieksekusi pada semua jenis*compute device*, dalam

hal ini CPU atau GPU. Dengan demikian jenis kalkulasi yang dilakukan oleh CPU dan GPU dalam kasus ini tetaplah sama.Di lain pihak, *computing power* yang dimiliki GPU jauh lebih besar ketimbang CPU yang digunakan. CPU Phenom II x6 1055T yang digunakan secara teoritis hanya memiliki kemampuan sebesar 38.4 GFlops, sementara Radeon 5850 memiki kemampuan hingga mencapai 2.09 TFlops.Hal ini merupakan suatu anomali yang harus dianalisa.

Kecurigaan pertama ditujukan kepada desain *kernel* yang belum sempurna. Pada bab 3 disebutkan bahwa kernel yang digunakan merupakan *porting* dari algoritma *kernel* yang didesain untuk C for CUDA. Diduga desain tersebut telah dioptimalkan untuk GPU berbasis chip NVIDIA. Selanjutnya *kernel* yang telah dibuat akan di analisa menggunakan program System Kernel Analyzer v 1.5 (SKA) dari AMD. Program ini berfungsi untuk menganalisa efisiensi suatu kernel untuk dijalankan pada divais AMD. Berikut analisa untuk kedua kernel:

Tabel4.6Output SKA untuk kernel bjrot dan bjrot8

Name	Min	Max	Avg	ALU	Fetch	Write	Est Cycles	ALU:Fetch	BottleNeck	Thrd\Clk	Throughput
bjrot	15	57	40.5	483	76	75	40.5	1.5	Global Write	0.79	672 M Threads\Sec
bjrot8	14	62	44	530	83	86	44	1.51	Global Write	0.73	618 M Threads\Sec

Dari hasil analisis tersebut terlihat bahwa perbandingan operasi matematis (ALU) dengan operasi memori (fetch) sangat kecil. Hal tersebut bertentangan dengan desain kernel yang baik yang harus memiliki rasio ALU:Fetch yang tinggi. Walaupun kemampuan kalkulasi pada GPU 1 (2.09 Terra Flops) sangat tinggi, namun kemampuan tersebut belum diimbangi oleh kecepatan memori dan bus PCIe (hanya sekitar 2 Giga Float per detik) yang digunakan dalam operasi *fetch* ke momori utama sistem.Jelas bahwa operasi *fetch*dan *write* merupakan *bottleneck* pada implementasi ini.

Sementara itu, untuk implementasi dengan CPU, operasi *fetch* bukan merupakan masalah. CPU dapat melakukan komunikasi dengan memori utama sistem dengan mudah. *Bottleneck* yang terjadi hanya pada batasan kemampuan kalkulasi CPU itu sendiri.

Dari fakta tersebut, dapat disimpulkan bahwa desain kernel yang digunakan belum mampu meng eksploitasi GPU 1 karena belum terlalu intensif

operasi matematis. Untuk dapat mengeksploitasi kemampuan sebuah GPU lebih jauh, maka diperlukan perhitungan matematis yang lebih berat atau jumlah data yang jauh lebih banyak lagi untuk dihitung. GPU 1 masih *overkill* untuk kernel ini. Hal ini terlihat dari utilisasi GPU saat kernel dijalankan hanya sekitar 10% saja.

Desain kernel ini mungkin lebih tepat untuk diimplementasikan pada GPU yang lebih rendah kemampuannya. Walaupun tidak akan menjadi lebih cepat, setidaknya GPU tersebut dapat digunakan secara optimal dan tidak mubazir.

Tabel4.7Output SKA untuk kernel bjrot pada GPU AMD yang berbeda

Name	Min	Max	Avg	ALU	Fetch	Write	Est Cycles	ALU:Fetch	BottleNeck	Thrd\Clk	Throughput
Radeon HD 4890	15	808.8	71.01	672	74	75	71.01	5	ALU Ops	0.23	192 M Threads\Sec
Radeon HD 4770	17.25	1011	88.76	672	74	75	88.76	5	ALU Ops	0.18	135 M Threads\Sec
Radeon HD 4870	15	808.8	71.01	672	74	75	71.01	5	ALU Ops	0.23	169 M Threads\Sec
FireStream 9250	15	808.8	71.01	672	74	75	71.01	5	ALU Ops	0.23	141 M Threads\Sec
FireStream 9270	15	808.8	71.01	672	74	75	71.01	5	ALU Ops	0.23	169 M Threads\Sec
Radeon HD 5870	15	57	40.5	483	76	75	40.5	1.5	Global Write	0.79	672 M Threads\Sec

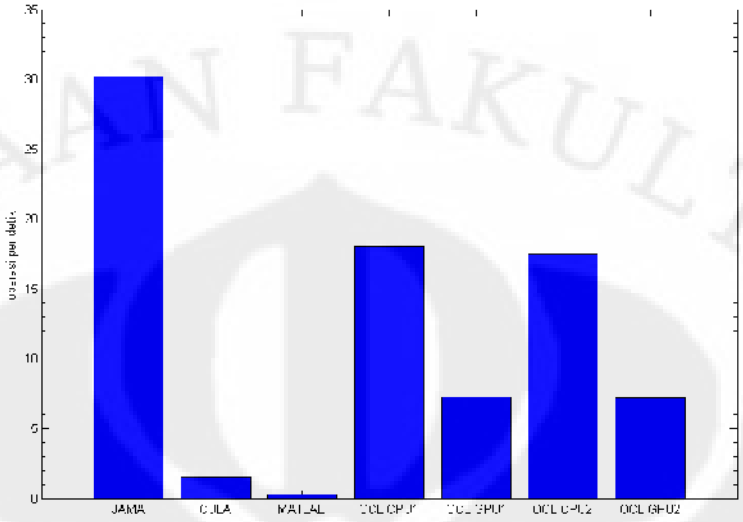
Hal ini juga menjelaskan alasan GPU 2 (GeForce 8400GS, 7.2 Giga Flops) yang memiliki *computing power* jauh dibawah GPU 1 memiliki kinerja yang masih serupa dengan GPU 1.

4.3.1 Hasil dan Analisa Uji Permintaan Serempak

Pengujian Permintaan serempak merupakan pengujian yang lebih mencerminkan penggunaan sistem secara normal sehari-hari. Pengujian ini menuntut sistem melakukan perhitungan untuk ukuran matrix 16x16 yang sering ditemui pada jawaban esay pada umumnya dengan jumlah yang sangat banyak. Berikut adalah data yang diperoleh.

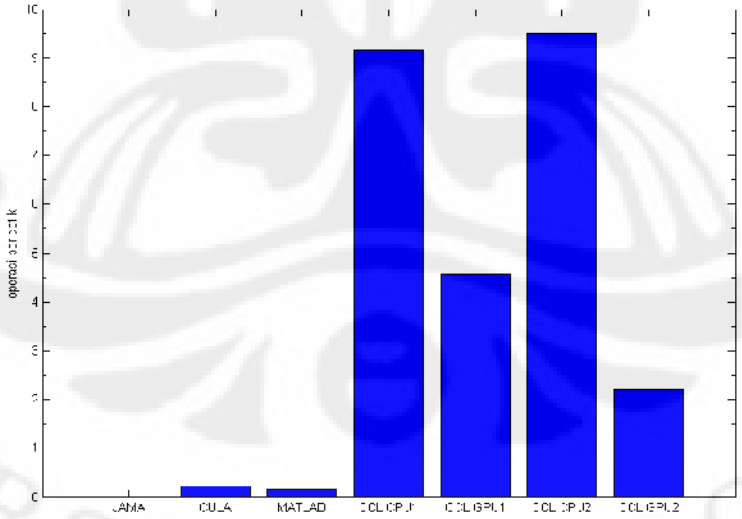
Tabel4.8Perbandingan Jumlah Kalkulasi SVD per detik untuk masing-masing platform

	Jumlah Operasi SVD per detik						
Ukuran matriks	JAMA	CULA	MATLAB	OLC CPU1	OCL GPU1	OCL CPU2	OCL GPU2
16x16	30.19331246	1.494298812	0.279953019	18.07243098	7.247002232	17.46321063	7.182692574
512x512	0.001071312	0.211757634	0.153726338	9.176241514	4.572684497	9.485335592	2.211533069



Gambar 4.13 Grafik perbandingan jumlah kalkulasi SVD untuk setiap detiknya pada matrix 16x16

Dari grafik tersebut dapat ditarik kesimpulan bahwa untuk penggunaan normal, JAMA masih jauh lebih baik. Implementasi ini mampu melayani lebih dari 30 perhitungan setiap detiknya. Namun kondisi tersebut akan berubah apabila jawaban yang akan dikoreksi oleh sistem sangat panjang. Berikut adalah kasus ekstrim untuk kalkulasi SVD mencapai 250 ribu kata.



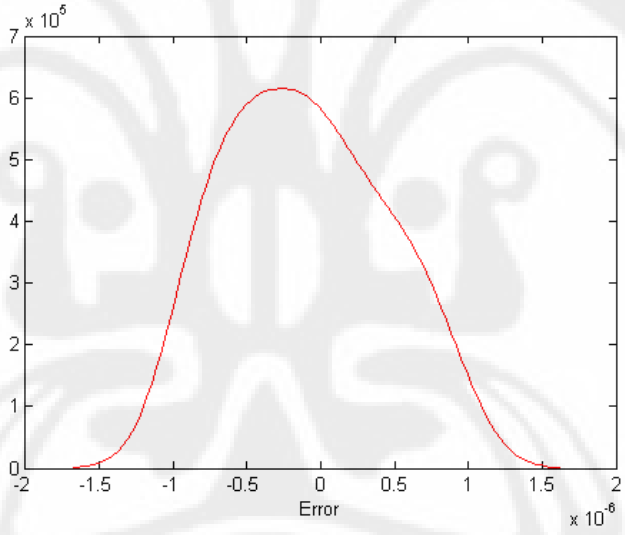
Gambar 4.14 Grafik perbandingan jumlah kalkulasi SVD untuk setiap detiknya pada matrix 512x512

Pada kasus ini kondisi berbalik. Implementasi SVD dalam JAMA, CULA dan MATLAB tidak mampu melakukan kalkulasi dengan tepat waktu. Implementasi OpenCL CPU dengan masih memimpin.Implementasi dengan OpenCL GPU juga masih dapat digunakan.

4.4 Analisa Ketepatan Matematis

Pada implementasi CULA tidak dilakukan pengujian ketepatan matematis dari implementasi ini. Ketepatan matematis implementasi ini telah dinyatakan oleh pembuat API. Karena tidak dilakukan perubahan selain implementasi teknis, maka ketepatan matematis implementasi ini tidak akan berubah.

Untuk implementasi OpenCL, dilakukan pengukuran sederhana perbandingan nilai SVD dari 100 matriks menggunakan OpenCL dan Matlab. Maka diperoleh sebaran kesalahan sebagai berikut:



Gambar 4.15 Grafik kepadatan probabilitas kesalahan

Kesalahan yang paling signifikan bersumber dari tingkat ketelitian tunggal yang digunakan. Semantara itu, kalkulasi referensi pada MATLAB menggunakan tingkat ketelitian ganda. Sumber kesalahan lain diperkirakan berasal dari pemotongan nilai float ketika bilangan ditampilkan melalui *stdout*yang hanya menampilkan 7 digit signifikan. Untuk mengatasi dapat dilakukan dengan cara memperpanjang format keluaran agar sesuai dengan presisi float.

BAB 5
KESIMPULAN

Berikutkesimpulan yang dapat diambil dari skripsi ini:

1. Implementasi perhitungan SVD pada Simple-O menggunakan platform GPGPU berhasil dilakukan, yaitu menggunakan CULA tools pada platform CUDA dan algoritma *single side Jacobi rotaion* oleh ZhangShu dan DouHeng pada platform OpenCL.
2. Pada matriks berukuran besar, implementasi SVD dalam GPGPU mampu meningkatkan performa sistem antara 200 kali (CULA tools pada GeForce 8400 GS) – 4200 kali (OpenCL pada Radeon 5850) dan terus meningkat untuk ukuran matriks yang lebih besar dibandingkan dengan JAMA.
3. Untukpenggunaan normal, implementasi dalam JAMA masih yang terbaik. Dengan menggunakan implementasi ini, Simple-O mampu menggunakan hingga 30 operasi LSA setiapdetiknya. OpenCL menyusul di posisike dua. Implementasidalam MATLAB menempatiposisi terakhir.
4. Implementasi OpenCL dalam GPU masih belum optimal, ditandai dengan lebih rendahnya performa GPU1 (0.21 detik untuk matriks 512x512) dibandingkan dengan CPU1 (0.1 detikuntuk matriks 512x512). Diperlukan desain kernel yang lebih cocok untuk GPU yang digunakan.
5. Kalkulasi SVD dalam skripsi ini hanya mampu untuk memanfaatkan 10% *computing power* yang adapada GPU Radeon 5850 (2.09TFlops) yang digunakan dalam pengujian. Diperlukan permasalahan matematis yang lebih kompleks dan besar untuk dapat memanfaatkan seluruh kemampuan GPU ini.

DAFTAR ACUAN

[1] T. K. Landauer, S. Dumais.“Latent semantic analysis”.Scholarpedia, 3(11):4356.[Online]. Available: http://www.scholarpedia.org/article/Latent_semantic_analysis [Diakses: 8 Juni 2010]

[2] A. A. P. Ratna, A. W. Astato, B. Budiarjo, D. Hartanto, “Simple-O: Web Based Automated Essay Grading System Using Latent Semantic Analysis method for Indonesian Language Considering Weight Word and Word Synonym”, The 10th International Conference on Quality in Research, Faculty of Engineering, University of Indonesia, 4 – 6 December 2007, Depok, Indonesia.

[3] AnakAgungPutriRatna, M. Salman, B. Budiardjo, D. HartantodanSeinosuke Narita, “SIMPLE: SistemPenilaianEseiOtomatisBerbasis WEB DenganMetodeLatemt Semantic Analysis Yang DigunakanPadaBahasaInsonesiaDenganPenambahan Kata Bobot”, Journal of Technology Edisi No. 3 Thn XX, September 2006, ISSN: 0215-1685.

[4] AnakAgungPutriRatna, BagioBudiardjodanDjokoHartanto, “SIMPLE: SistemPenilaianEseiOtomatisuntukMenilaiUjiandalamBahasaInsonesia”, JurnalMakara Seri Teknologi, volume 11, April 2007, ISSN : 1693-6698

[5] I. Buck, A. Lefohn, et al. “General Purpose Computation on Graphics Hardware”*IEEE GPUVis Course 2005*, 2005. [Online].http://gpgpu.org/static/s2005/FullCourseNotes.pdf [Diakses: 8 Juni 2010]

[6] G. H. Golub& W. Kahan, “Calculating the singular Values and Pseudoinverse of a Matrix”,*Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis*, 2(2):205-224, 1965.

[7] V. Strumpen, H. Hoffmann, A. Agarwal. “A Stream Algorithm for the SVD”,*Computer Science and Artificial Intelligence Laboratory Technical Report*. MIT-CSAIL-TR-2003-024. 2003.

[8] M. R.Hestenes, “Inversion of Matrices by Biorthogonalization and Related Result”, *Journal of the Society for Industrial and Applied Mathematics*, 6(1):51-90, Mar 1958.

[9] NVIDIA, “Graphics Processing Unit (GPU)”. [Online] http://www.nvidia.com/object/gpu.html31 Aug 1999.

[10] AMD. *ATI Stream Computing Technical Overview*. 2009.

[11] AMD. *ATI Stream Computing User Guide*. 2009.

[12] AMD. *Evergreen-Family ISA-Instruction and Microcode*. 2010.

[13] AMD. *ATI Stream SDK OpenCLProgramming Guide*, 2010.

[14] AMD. *CALProgramming Guide*, 2010.

[15] NVIDIA.*CUDA Programming Guide*. 2010.

[16] NVIDIA.*CUDA Refference Manual Version 3.0*. 2010.

[17] NVIDIA.*OpenCL Programming Guide* 2010.

[18] NVIDIA.*OpenCL Best Practice Guide 3.0*. 2010.

[19] KhronosOpenCL Working Group,*TheOpenCL Specification*, Version: 1.0Document Revision: 48

[20] D. Roe. “OpenCL gets touted in Texas”. MacWorld.com.[Online]http://www.macworld.com/article/136921/2008/11/opencl.html?lsrc=top_2[Diakses: 8 Juni 2010]

[21] PHP, *PHP General Information* [Online] http://id2.php.net/manual/en/faq.general.php [Diakses: 8 Juni 2010]

[22] MySQL, *About MySQL*, [Online]http://www.mysql.com/about/ [Diakses: 8 Juni 2010]

[23] Apache, *About Apache*, [Online] http://httpd.apache.org/ABOUT_APACHE.html [Diakses: 8 Juni 2010]

[24] CULA tools, *CULA tools FAQ General*[Online] http://www.culatools.com/faq#faq_general[Diakses: 8 Juni 2010]

[25] ZhangShu, DouHeng, “Matrix Singular Value Decomposition Based On Computing UnifiedDevice Architecture”, *supplementary issue of Application Research of Computers*,ChengDu, Jun 2009

SIMPLE-O

PenilaianEseiOtomatisdenganmenggunakanmetode*Latent Semantic Analysis*terdiridaribeberapamodul.Berikutakandijelaskanmasing-masingmodultersebut.

Modul*Login*

Modul*login*/menu utamadigunakanuntukmembedakan*login*mahasiswa, dosenatau admin.Bila*login*dengan*login* ID dosenmendapatfasilitassebagaidosen, begitujugabila*login*dengan*login* ID mahasiswa.Selainitu, terdapatjuga*ID root* yang berfungsisebagai*super user*darisistemini. *Root*memilikikemampuanuntukmeng-*asignuser*barusebagaidosenataumahasiswadanmenentukanmatakuliahnya. Adapunalgoritmatampilanuntukmodul*login*/menu utamaditunjukkanpadagambar L1.3.

```
prog();
   [Pengecekan idetifikasi user]
   if idlogin = 1 then                    {login dosen}
   (
     [tampilan utama untuk dosen]
     if userName = root then
     (
          [tampilkan menu tambahan untuk root]
     }
     read (pil);
   [ke sub modul sesuai pilihan : listing nilai, tampilkan
dan mengisisoal, registrasiatau logout]
    )
   ifidlogin = 2 then {login mahasiswa}
   (
```

```
      [tampilan utama untuk mahasiswa]

      read (pil);

      [ke sub modul sesuai pilihan : listing nilai,
tampilkan dan menjawabsoal, registrasiatau logout]

    )

Eprog
```

Gambar L1.1Pseudocode Menu Utama
(Ratna, Budiarjo, Hartanto, 2007)

ModulDosen

Moduluntukdosenterdiridari 4 (empat) modul :

1. Modul List Nilai, yang merupakanmoduluntukmelihatnilaidarimahasiswa yang mengambilujianuntukmatakuliahtersebut,
2. ModulSoal, yang merupakanmoduluntukmemasukkansoal yang baru, mengeditdanmenghapussoal yang lama,
3. Modul Mata Kuliah, yang merupakanmoduluntukmelihatmatakuliah yang adapadasistem,
4. ModulRegistrasi, yang merupakanmoduluntuk admin untukmelakukanregistrasipadasistem.

Modul List Nilai

Padamodul List Nilaiinidosendapatmelihatnilaidarimahasiswa yang mengambilujianpadamatakuliah yang dikelolaolehdosen yang bersangkutan.Adapunalgoritmauntukmodul list nilaiterdapatpadagambar L1.4.

```
Proc listnilai ()
      [load nilai_mhs untuk matkul dari database]
      i=0;
      while(score =! EOF)
            i++;
```

```
            write ("idmk");
            write ("nama_mhs");
            write ("score");
      ewhile
Eproc
```

Gambar L1.2PseudocodeListnilai
(Ratna, Budiarjo, Hartanto, 2007)

ModulSoal

Padamodulsoaldapatdilakukanbeberapahalsepertiuraian di bawahini.

1. Mengeditsoal.
2. Menghapussoal.
3. Meng-*input*soal.
4. Memilih kata bobot.

Padamodulsoaldapatdilakukanmengedit, menghapusdanmeng-*input*soalolehdosen yang besangkutan.Algoritmaglobal untukmodulsoaldapatditunjukkanpadagambar L1.5.

```
Procsoal ()

      [loadmatakuliahuntuk user dari database, dapat di
      delete, edit dan input soalmatakuliahtersebut]
      read (pil);

      ifpil = Delete then   {bagianuntukmenghapussoal}
            [hapus record dari database]
      else

      ifpil = Edit then           {bagianuntukmengeditsoal}
```

```
            [mengeditsoal]

      ifpil = Input Soal then     {bagianuntukmenambahsoal}

            [mengisi soal]
Eproc
```

Gambar L1.3PseudocodeModulSoal
(Ratna, Budiarjo, Hartanto, 2007)

Padamodulsoaldapatdibuatbeberapafituruntukmeningkatkankinerjadarimetode*Latent Semantic Analysis*.Fiturtersebutdiuraikan dibawahini.

Fiturpertamaadalahpenambahanbobotdari*keyword*. *Keyword*akandipilih 2 kali olehdosen yang bersangkutan. Untukujicobatahapanpertamaadalah*keyword*biasa.Untuksatujawaban, *keyword*biasadapatterdiridari 10 – 20 *keyword*untukpercobaanpertama.Untukpercobaankedua, *keyword*biasaditentukanoleh minimal 3 dosen yang kompeten.Yang keduaadalah*keyword*bobot, yang mencakuphal yang dianggappentingsekali.Padarisetini*keyword*bobotterdiridari 5 – 8 kata *keyword*perjawabanuntukpercobaanpertama.Untukpercobaankedua, *keyword*biasajugaditentukanoleh minimal 3 dosen yang kompeten.Untuk*keyword*biasapembobotanmatriksadalah 1.Sedangkanuntuk*keyword*bobotpembobotannyaadalahdikalidengan 2.

```
      ifpil = Input Soal then
                        {bagianuntukmenambahsoal}
            [inputsoal]
            [inputjawaban]
            [input kata kunci]
            [input kata kuncibobot]
```

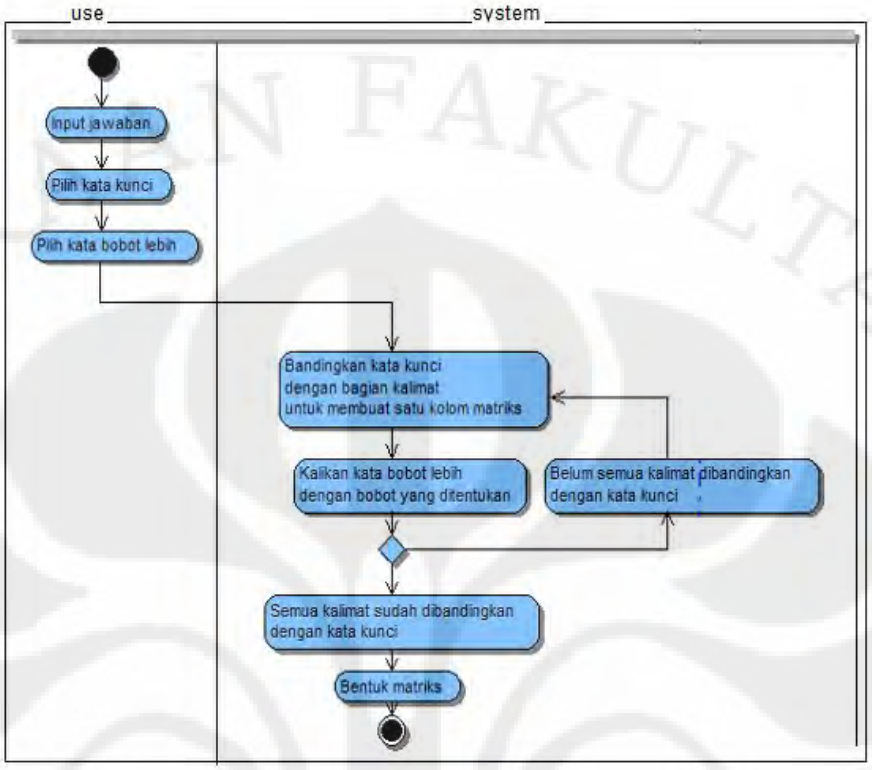
```
[simpansoaldan kata kuncike database]
```

Gambar L1.4Pseudocode Input Soal
(Ratna, Budiarjo, Hartanto, 2007)

```
ifpil = Pilih Kata Bobot then
                {bagianuntukmenambahsoal}
      [pilihmatkul]
      [pilihsoal]
      [input kata kuncibobot]
      [bentuk matriks]
      [Proses SVD]
      [Simpan nilai frobenius/cos Alfa yang sesuai]
```

Gambar L1.5PseudocodePilih Kata Bobot
(Ratna, Budiarjo, Hartanto, 2007)

Fiturkeduaadalahfiturpersamaan kata.Padafiturpersamaan kata, kata yang samaatau yang memilikiarti yang samaakandianggapsama. Persamaaniniberdasarkantabel yang dibentukseperticontohpadaTabel L1.1. *Activity diagram*konversimatriksdanpembobotanditunjukkandi bawahini.



Gambar L1.6*Activity Diagram*Konversi Matrix

Tabel L1.1Persamaan Kata

no	Kata	kata dasar	kode kata	kodepersamaan
1	memiliki	miliki	1	1
2	mempunyai	punya	2	1
3	berkesempatan	sempat	0	2
4	Bisa	bisa	9	2
5	dapat	dapat	9	2

Tabel L1.1Persamaan Kata (lanjutan)

6	kemungkinan	mungkin	0	2
7	mampu	mampu	0	2
8	mungkin	mungkin	0	2
9	butuhkan	Butuh	3	3
10	dibutuhkan	Butuh	3	3
11	diperlukan	Perlu	3	3
12	perlukan	Perlu	3	3
13	butuh	Butuh	2	4
14	membutuhkan	Butuh	2	4
15	memerlukan	Perlu	2	4
16	perlu	Perlu	2	4
17	kebutuhan	Butuh	1	5
18	keperluan	Perlu	1	5
19	dimiliki	Miliki	2	6
20	dipunyai	Punya	3	6
21	menambah	Tambah	2	7
22	menambahkan	Tambah	2	7
23	menjumlah	Jumlah	2	7
24	menjumlahkan	Jumlah	2	7
25	modem	Modem	0	8
26	connector	Connect	0	9
27	konektor	Konektor	0	9
28	penyambung	Sambung	0	9
29	dijumlah	Jumlah	3	10
30	dijumlahkan	Jumlah	3	10
31	ditambah	Tambah	3	10
32	ditambahkan	Tambah	3	10

Tabel L1.1Persamaan Kata (lanjutan)

33	ditingkatkan	Tingkat	3	10
34	penambahan	Tambah	1	11
35	penjumlahan	Jumlah	1	11
36	penambah	Tambah	1	12
37	penjumlah	Jumlah	0	12
38	standar	Standar	1	13
39	standard	Standard	0	13
40	standart	Standar	1	13
41	kirim	Kirim	0	14
42	penghantaran	Hantar	0	14
43	pengirim	Kirim	0	14
44	pengiriman	Kirim	1	14
45	pengirimannya	Kirim	0	14
46	pentransferan	Transfer	1	14
47	sender	Send	0	14
48	sending	Send	0	14
49	transfer	Transfer	0	14

KolompertamapadaTabel L1.1, adalahkolom no yang menunjukkanurutan kata. Kolomkeduaadalahkolom kata yang kemungkinandigunakanpadaujianolehsiswa.Kolominiterdiridarisemua kata yang kemungkinandigunakanolehsiswa, baikdalambentuk kata kerjatransitif, kata kerjaintransitif, benda, sifatdan lain sebagainya,

Kolomketigaadalahkolom kata dasar.Semua kata yang munculpadakolomkeduadikembalikanlagidalambentuk kata dasarnya.Kolomkeempatadalahkolomkode kata.Padakolomkeempatinisemua kata dikategorikanpadadaftar di bawahini.

Kata benda : 1.

Kata kerjaaktif : 2.

Kata kerjapasif : 3.

Kata sifat : 4.

Kata keterangan : 5.

Benda satuan : 6.

Kata majemuk : 7.

Ajektif (kata keterangan) : 8.

Adverb (kata sifat) : 9.

Kata sambung : 10.

Untuksistem yang saatinidikembangkan, kolomke 4 tidakdigunakan.

Kolomkelimaadalahkolomuntukmenyatakankodepersamaan kata. Proses yang dilakukanuntukpersamaan kata adalahsebagaiberikut. Pertama-tama kalimat yang dimasukkanolehsiswasebagaijawabandiuraikanmenjadi kata-kata.Kata-kata tersebutkemudiandisesuaikandengan kata kuncidarijawabanreferensidosen yang telahdimasukkansebelumnyakesistem.Kata-kata tersebutkemudiandiceklagike database persamaan kata.Bilaada kata yang kodepersamaannyasama, maka kata tersebutdiprosessamadengan kata kunci yang adapadajawabanreferensidosen. Pembobotandiberikanapakah kata tersebutsesuai kata kunciatau kata bobot, SebagaicontohterlihatpadaTabelL1.1, untuk no. 22 dan no. 23, yaitu ’menambahkan’ dan ’menjumlah’. Bilasiswamenjawabdengan kata ’menambahkan’ danjawabanreferensiadalah ’menjumlah’ makasiswatersebutmendapatkannilaipenuhuntuk kata tersebut.Algoritmauntukmodulpengecekanpersamaan kata dapatditunjukkanpadagambar L1.9..

```
Proc ujian ()
     [load matkul untuk user dari database]
     whilematkul =! EOF
          [tampilkan mata kuliah yang dipilih]
     ewhile
     read (pil);
     ifpil = Back then
```

```
            gotoHalaman_Muka;
      else


      ifpil = LihatSoal then
            write (“Soal”, soal[ ]);
            write (“Jawaban”);
            readjawab_mhs[ ];
            if submit = true then
                  [cek persamaan kata-kata jawaban mahasiswa]
                  [bentukmatriksdarijawabanmahasiswa];
                  [bandingkan matriks jawaban dengan
                        referensi];
                  [kirim nilai ke tabel database];
            else
            ( );
      else
      ( );
Eproc
```

Gambar L1.7PseudocodePengecekanPersamaan Kata

(Ratna, Budiarjo, Hartanto, 2007)

GeForece 8400 GS

```
==================================[ Graphics Adapter / GPU ]
- SLI: disabled
- GPUs: 1
- Logical GPUs: 1
- OpenGL Renderer: GeForce 8400 GS/PCI/SSE2/3DNOW!
- Drivers Renderer: NVIDIA GeForce 8400 GS
- DB Renderer: NVIDIA GeForce 8400 GS
- Device Description: NVIDIA GeForce 8400 GS
- Adapter String: GeForce 8400 GS
- Vendor: NVIDIA Corporation
- Vendor ID: 0x10DE
- Device ID: 0x06E4
- Sub device ID: 0x1163
- Sub vendor ID: 0x1462
- Drivers Version: Forceware 6.14.11.9713 (3-15-2010)
- GPU Codename: G98
- GPU Unified Shader Processors: 8
- GPU Vertex Shader Processors: 0
- GPU Pixel Shader Processors: 0
- SM / SIMD: 1
- TPC: 1
- Video Memory Size: 512 MB
- Video Memory Type: DDR2
- Clocks level #0: Core: 567MHz - Memory: 400MHz - Shader: 1400MHz
- BIOS String:  62.98.47.00.93
- Current Display Mode: 1280x1024 @ 60 Hz - 32 bpp
==================================[ NVIDIA CUDA Capabilities ]
- CUDA Device 0
        - Device name: GeForce 8400 GS
        - Compute Capability: 1.1
        - Total Memory: 511 MB
        - Shader Clock Rate: 1400 MHz
        - Multiprocessors: 1
        - Warp Size: 32
        - Max Threads Per Block: 512
        - Threads Per Block: 512 x 512 x 64
        - Grid Size: 65535 x 65535 x 1
        - Registers Per Block: 8192
        - Texture Alignment: 256 byte
        - Total Constant Memory: 64 Kb


==================================[ OpenCL Capabilities ]
- Num OpenCL platforms: 1
- Name: NVIDIA CUDA
- Version: OpenCL 1.0 CUDA 3.0.1
- Profile: FULL_PROFILE
- Vendor: NVIDIA Corporation
- Num devices: 1

        - CL_DEVICE_NAME: GeForce 8400 GS
        - CL_DEVICE_VENDOR: NVIDIA Corporation
        - CL_DRIVER_VERSION: 197.13
        - CL_DEVICE_PROFILE: FULL_PROFILE
        - CL_DEVICE_VERSION: OpenCL 1.0 CUDA
        - CL_DEVICE_TYPE: GPU
        - CL_DEVICE_VENDOR_ID: 0x10DE
        - CL_DEVICE_MAX_COMPUTE_UNITS: 1
        - CL_DEVICE_MAX_CLOCK_FREQUENCY: 1400MHz
        - CL_NV_DEVICE_COMPUTE_CAPABILITY_MAJOR: 1
        - CL_NV_DEVICE_COMPUTE_CAPABILITY_MINOR: 1
        - CL_NV_DEVICE_REGISTERS_PER_BLOCK: 8192
        - CL_NV_DEVICE_WARP_SIZE: 32
        - CL_NV_DEVICE_GPU_OVERLAP: 0
        - CL_NV_DEVICE_KERNEL_EXEC_TIMEOUT: 1
        - CL_NV_DEVICE_INTEGRATED_MEMORY: 0
        - CL_DEVICE_ADDRESS_BITS: 32
        - CL_DEVICE_MAX_MEM_ALLOC_SIZE: 131072KB
        - CL_DEVICE_GLOBAL_MEM_SIZE: 511MB
        - CL_DEVICE_MAX_PARAMETER_SIZE: 4352
        - CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE: 0 Bytes
        - CL_DEVICE_GLOBAL_MEM_CACHE_SIZE: 0KB
```

```
- CL_DEVICE_ERROR_CORRECTION_SUPPORT: NO
- CL_DEVICE_LOCAL_MEM_TYPE: Local (scratchpad)
- CL_DEVICE_LOCAL_MEM_SIZE: 16KB
- CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE: 64KB
- CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS: 3
- CL_DEVICE_MAX_WORK_ITEM_SIZES: [512 ; 512 ; 64]
- CL_DEVICE_MAX_WORK_GROUP_SIZE: 512
- CL_EXEC_NATIVE_KERNEL: 4645096
- CL_DEVICE_IMAGE_SUPPORT: YES
- CL_DEVICE_MAX_READ_IMAGE_ARGS: 128
- CL_DEVICE_MAX_WRITE_IMAGE_ARGS: 8
- CL_DEVICE_IMAGE2D_MAX_WIDTH: 8192
- CL_DEVICE_IMAGE2D_MAX_HEIGHT: 8192
- CL_DEVICE_IMAGE3D_MAX_WIDTH: 2048
- CL_DEVICE_IMAGE3D_MAX_HEIGHT: 2048
- CL_DEVICE_IMAGE3D_MAX_DEPTH: 16
- CL_DEVICE_MAX_SAMPLERS: 16
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR: 1
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT: 1
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT: 1
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG: 1
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT: 1
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE: 0
- CL_DEVICE_EXTENSIONS: 10
- Extensions:
        - cl_khr_byte_addressable_store
        - cl_khr_icd
        - cl_khr_gl_sharing
        - cl_nv_d3d9_sharing
        - cl_nv_compiler_options
        - cl_nv_device_attribute_query
        - cl_nv_pragma_unroll
        -
        - cl_khr_global_int32_base_atomics
        - cl_khr_global_int32_extended_atomics
```

Radeon 5850 & Phenom x6 1055XT

```
==================================[ Graphics Adapter / GPU ]
- CrossFire: disabled
- GPUs: 1
- Physical adapters: 1
- OpenGL Renderer: ATI Radeon HD 5800 Series
- Drivers Renderer: ATI Radeon HD 5800 Series
- DB Renderer: ATI Radeon HD 5850
- Device Description: ATI Radeon HD 5800 Series
- Adapter String: ATI Radeon HD 5800 Series
- Vendor: ATI Technologies Inc.
- Vendor ID: 0x1002
- Device ID: 0x6899
- Sub device ID: 0x0B00
- Sub vendor ID: 0x1002
- Perf Level 0 - GPU: 157 MHz
- Perf Level 0 - Memory: 300 MHz
- Perf Level 1 - GPU: 550 MHz
- Perf Level 1 - Memory: 900 MHz
- Perf Level 2 - GPU: 725 MHz
- Perf Level 2 - Memory: 1000 MHz
- Drivers Version: 8.712.0.0 - Catalyst 10.3 (3-2-2010) - atig6pxx.dll
- ATI Catalyst Version String: 10.3
- ATI Catalyst Release Version String: 8.712-100302b-096979C-ATI
- GPU Codename: Cypress
- GPU Unified Shader Processors: 1440
- GPU Vertex Shader Processors: 0
- GPU Pixel Shader Processors: 0
- SM / SIMD: 18
- TPD (Watts): 151
- Video Memory Size: 1024 MB
- Video Memory Type: unknown
- Clocks: Level 157 - GPU: 300MHz - Memory: 1565160MHz
- Clocks: Level 550 - GPU: 900MHz - Memory: 1565160MHz
- Clocks: Level 725 - GPU: 1000MHz - Memory: 1565160MHz
- BIOS String: 113-C00201-100
- Current Display Mode: 1920x1080 @ 60 Hz - 32 bpp
==================================[ OpenCL Capabilities ]
- Num OpenCL platforms: 1
- Name: ATI Stream
- Version: OpenCL 1.0 ATI-Stream-v2.0.1
- Profile: FULL_PROFILE
- Vendor: Advanced Micro Devices, Inc.
- Num devices: 2

        - CL_DEVICE_NAME: AMD Phenom(tm) II X6 1055T Processor
        - CL_DEVICE_VENDOR: AuthenticAMD
        - CL_DRIVER_VERSION: 1.0
        - CL_DEVICE_PROFILE: FULL_PROFILE
        - CL_DEVICE_VERSION: OpenCL 1.0 ATI-Stream-v2.0.1
        - CL_DEVICE_TYPE: CPU
        - CL_DEVICE_VENDOR_ID: 0x1002
        - CL_DEVICE_MAX_COMPUTE_UNITS: 6
        - CL_DEVICE_MAX_CLOCK_FREQUENCY: 3262MHz
        - CL_DEVICE_ADDRESS_BITS: 32
        - CL_DEVICE_MAX_MEM_ALLOC_SIZE: 524288KB
        - CL_DEVICE_GLOBAL_MEM_SIZE: 1024MB
        - CL_DEVICE_MAX_PARAMETER_SIZE: 4096
        - CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE: 64 Bytes
        - CL_DEVICE_GLOBAL_MEM_CACHE_SIZE: 64KB
        - CL_DEVICE_ERROR_CORRECTION_SUPPORT: NO
        - CL_DEVICE_LOCAL_MEM_TYPE: Global
        - CL_DEVICE_LOCAL_MEM_SIZE: 32KB
        - CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE: 64KB
        - CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS: 3
        - CL_DEVICE_MAX_WORK_ITEM_SIZES: [1024 ; 1024 ; 1024]
        - CL_DEVICE_MAX_WORK_GROUP_SIZE: 1024
        - CL_EXEC_NATIVE_KERNEL: 4645096
        - CL_DEVICE_IMAGE_SUPPORT: NO
        - CL_DEVICE_MAX_READ_IMAGE_ARGS: 0
        - CL_DEVICE_MAX_WRITE_IMAGE_ARGS: 0
        - CL_DEVICE_IMAGE2D_MAX_WIDTH: 0
        - CL_DEVICE_IMAGE2D_MAX_HEIGHT: 0
```

```
- CL_DEVICE_IMAGE3D_MAX_WIDTH: 0
- CL_DEVICE_IMAGE3D_MAX_HEIGHT: 0
- CL_DEVICE_IMAGE3D_MAX_DEPTH: 0
- CL_DEVICE_MAX_SAMPLERS: 0
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR: 16
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT: 8
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT: 4
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG: 2
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT: 4
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE: 0
- CL_DEVICE_EXTENSIONS: 6
- Extensions:
      - cl_khr_icd
      - cl_khr_global_int32_base_atomics
      - cl_khr_global_int32_extended_atomics
      - cl_khr_local_int32_base_atomics
      - cl_khr_local_int32_extended_atomics
      - cl_khr_byte_addressable_store

- CL_DEVICE_NAME: Cypress
- CL_DEVICE_VENDOR: Advanced Micro Devices, Inc.
- CL_DRIVER_VERSION: CAL 1.4.556
- CL_DEVICE_PROFILE: FULL_PROFILE
- CL_DEVICE_VERSION: OpenCL 1.0 ATI-Stream-v2.0.1
- CL_DEVICE_TYPE: GPU
- CL_DEVICE_VENDOR_ID: 0x1002
- CL_DEVICE_MAX_COMPUTE_UNITS: 18
- CL_DEVICE_MAX_CLOCK_FREQUENCY: 725MHz
- CL_DEVICE_ADDRESS_BITS: 32
- CL_DEVICE_MAX_MEM_ALLOC_SIZE: 262144KB
- CL_DEVICE_GLOBAL_MEM_SIZE: 256MB
- CL_DEVICE_MAX_PARAMETER_SIZE: 1024
- CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE: 0 Bytes
- CL_DEVICE_GLOBAL_MEM_CACHE_SIZE: 0KB
- CL_DEVICE_ERROR_CORRECTION_SUPPORT: NO
- CL_DEVICE_LOCAL_MEM_TYPE: Local (scratchpad)
- CL_DEVICE_LOCAL_MEM_SIZE: 32KB
- CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE: 64KB
- CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS: 3
- CL_DEVICE_MAX_WORK_ITEM_SIZES: [256 ; 256 ; 256]
- CL_DEVICE_MAX_WORK_GROUP_SIZE: 256
- CL_EXEC_NATIVE_KERNEL: 4645096
- CL_DEVICE_IMAGE_SUPPORT: NO
- CL_DEVICE_MAX_READ_IMAGE_ARGS: 0
- CL_DEVICE_MAX_WRITE_IMAGE_ARGS: 0
- CL_DEVICE_IMAGE2D_MAX_WIDTH: 0
- CL_DEVICE_IMAGE2D_MAX_HEIGHT: 0
- CL_DEVICE_IMAGE3D_MAX_WIDTH: 0
- CL_DEVICE_IMAGE3D_MAX_HEIGHT: 0
- CL_DEVICE_IMAGE3D_MAX_DEPTH: 0
- CL_DEVICE_MAX_SAMPLERS: 0
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR: 16
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT: 8
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT: 4
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG: 2
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT: 4
- CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE: 0
- CL_DEVICE_EXTENSIONS: 4
- Extensions:
      - cl_khr_global_int32_base_atomics
      - cl_khr_global_int32_extended_atomics
      - cl_khr_local_int32_base_atomics
      - cl_khr_local_int32_extended_atomics
```