



UNIVERSITAS INDONESIA

PR 2 KAJIAN BAHASA PEMROGRAMAN PARALEL

LAPORAN TUGAS PEMROGRAMAN PARALEL

KELOMPOK III

Muhammad Fathurachman 1506706276

Otniel Yosi Viktorisa 1506706295

Yohanes Gultom 1506706345

FAKULTAS ILMU KOMPUTER

PROGRAM STUDI MAGISTER ILMU KOMPUTER

DEPOK

MEI 2016

DAFTAR ISI

Daftar Isi	ii
Daftar Gambar	iii
Daftar Tabel	iv
1 LINGKUNGAN PERCOBAAN	1
2 R	2
2.1 Pendahuluan	2
2.2 Fitur Bahasa R	2
2.3 Instalasi Bahasa R	3
2.3.1 Instalasi Bahasa R pada Linux	3
2.3.2 Instalasi Bahasa R pada Windows	3
2.3.3 Instalasi Package Bahasa R	3
2.4 Komputasi Paralel pada Bahasa R	4
2.4.1 Komputasi Paralel dengan GPU pada Bahasa R	4
2.4.2 CUDA pada Bahasa R	4
3 OPENCL	6
3.1 Pendahuluan	6
3.1.1 Instalasi	6
3.1.2 Struktur Program	8
3.1.3 Perbandingan Terminologi dengan CUDA	9
3.1.4 Library BLAS	11
3.1.5 Eksperimen	13
3.1.5.1 Device Query	13
3.1.5.2 Single-Precision AX Plus Y (SAXPY)	14
3.1.5.3 Perkalian Matriks Bujursangkar	15
3.1.5.4 Gaussian Filter Blurring pada Gambar Bitmap	16
4 KONTRIBUSI	18
Daftar Referensi	19

DAFTAR GAMBAR

3.1	Cara pemanggilan clblasSgemm pada clBLAS	12
3.2	Kinerja library BLAS pada OpenCL	13
3.3	Program untuk mendapatkan informasi platform dan device OpenCL	14
3.4	Program SAXPY 1024 elemen	15
3.5	Perbandingan perkalian matriks bujursangkar OpenCL dengan CUDA	16
3.6	Proses Gaussian Filter Blurring pada gambar	16
3.7	Kinerja Gaussian Blurring CPU vs GPU	17

DAFTAR TABEL

3.1	Terminologi Perangkat Keras	9
3.2	Qualifiers untuk fungsi Kernel	10
3.3	Indeks pada Kernel	10
3.4	Pemanggilan API	11

BAB 1

LINGKUNGAN PERCOBAAN

Eksperimen yang dilakukan pada kajian ini menggunakan laptop/personal PC dengan spesifikasi sebagai berikut:

- Notebook Core i7 5500U
 - RAM 8 GB, GPU NVIDIA 940M, SSD 250GB
 - Sistem operasi Ubuntu 15.10 (64-bit)
 - Driver NVIDIA 352
 - CUDA 7.5.3
 - NVIDIA OpenCL 1.2
 - GCC & G++ 4.9.3

Sedangkan kode program dari eksperimen OpenCL dapat diakses pada *public repository* <https://github.com/yohanesgultom/parallel-programming-assignment/tree/master/PR2>

BAB 2

R

2.1 Pendahuluan

Bahasa Pemrograman R adalah salah satu bahasa pemrograman yang digunakan untuk Komputasi Statistik dan Graphics. R banyak digunakan oleh Data Scientist untuk membuat software untuk mengolah data. R dibuat oleh Ross Ihaka dan Robert Gentleman di University of Auckland. Nama R kemudian diambil dari huruf pertama dua pembuat R. Pada Bahasa R terdapat library statistik seperti linear dan non linear model, classification, time-series analysis, dan lain-lain. Selain itu, R didesain agar pengguna mudah dalam memberikan kontribusi bagi pengembangannya, di mana pengguna dapat membuat package yang dapat digunakan oleh komunitas R lainnya. R didukung oleh banyak repository package yang digunakan untuk data manipulasi, perhitungan, visualisas. Dalam suatu package terdapat fungsi-fungsi untuk memproses data, penyimpanan, operator untuk menghitung array atau matriks. Bahasa R juga dilengkapi dengan fungsi-fungsi dasar pemrograman lainnya seperti perulangan, rekursif, percabangan dan lain-lain.

2.2 Fitur Bahasa R

Bahasa R didukung oleh berbagai fitur diantaranya adalah :

- Interpreted language (dapat dioperasikan menggunakan command line).
- Dukungan terhadap beberapa jenis struktur data seperti vector, matriks, array, dan data frame yang merupakan struktur data menyerupai matrik yang mampu menyimpan data dengan tipe data yang berbeda.
- Mendukung fungsi pemrograman prosedural dan berorientasi obyek.
- Memiliki performa komputasi statistik yang setara dengan Matlab dan Octave.
- Didukung dengan IDE, beberapa yang populer adalah Rstudio dan Visual Studio.

2.3 Instalasi Bahasa R

2.3.1 Instalasi Bahasa R pada Linux

Untuk menginstal Bahasa Pemrograman R pada Linux cukup dengan menuliskan perintah pada terminal seperti berikut

Jika proses instalasi telah selesai, pada terminal linux, silahkan mengetikan command R untuk memulai menulis program R. Untuk menggunakan IDE Rstudio pada Ubuntu, RStudio dapat diunduh pada situs <https://www.rstudio.com> dan di instal pada Ubuntu.

2.3.2 Instalasi Bahasa R pada Windows

Untuk Bahasa R pada Windows, Silahkan download file instalasi pada situs <https://cran.r-project.org/bin/windows/base/> dan instal R-3.x.x-win.exe sebagaimana menginstal software seperti biasa.

2.3.3 Instalasi Package Bahasa R

Packages pada R berisi fungsi-fungsi komputasi statistik tertentu, fungsi grafik, analisa, dan lain-lain yang ditulis menggunakan bahasa R dan dapat juga diintegrasikan dengan bahasa pemrograman lainnya seperti, Java, C, C++, FORTRAN. Jumlah package yang terdapat repository cran adalah 7.801 (January 2016). Dengan jumlah package yang sangat besar dan bervariasi, CRAN repository membagi dalam beberapa Task atau bidang agar mempermudah pengguna dalam memilih dan menggunakan package yang sesuai dengan pekerjaannya.

Cara memasang package yang ingin diinstal pada bahasa R, yaitu dengan mengetikan syntax `install.packages(_nama_package_)` pada command line R, atau dapat juga dengan mengunduh file package terlebih dahulu dan dengan perintah `install.packages(directory_package)`, tunggu hingga proses instalasi selesai. Setelah proses instalasi selesai untuk memanggil package yang telah terpasang, gunakan perintah `library(_nama_package_)`, maka fungsi-fungsi yang terdapat pada package tersebut sudah dapat digunakan.

2.4 Komputasi Paralel pada Bahasa R

2.4.1 Komputasi Paralel dengan GPU pada Bahasa R

GPU atau Graphical Processing Unit, adalah merupakan single-chip processor yang melakukan komputasi khusus untuk aplikasi 3D. Berbeda dengan CPU yang hanya memiliki beberapa core pada satu chip, GPU memiliki ratusan-bahkan ribuan core dalam satu chip. Karena sebagian besar proses komputasi pada GPU mencakup operasi vektor dan matriks, maka GPU dapat digunakan untuk mengeksekusi proses lain yang berbeda dengan pemrosesan pada graphics. Dalam hal ini GPU digunakan untuk melakukan komputasi non-graphical process, seperti menjalankan algoritma, komputasi FFT, persamaan linear.

Untuk mengakses GPU menggunakan R, terdapat dua cara yaitu :

- Menggunakan package yang tersedia oleh CRAN.
- Mengakses GPU melalui CUDA Library / CUDA Accelerated Programming Language C, C++, FORTRAN.

Untuk menggunakan package yang mendukung komputasi GPU dapat menggunakan package pada tabel berikut.

Contoh program R menggunakan GPU :

Simulasi program diatas menggunakan package gputools, dengan melakukan deklarasi fungsi untuk melakukan perkalian matriks. Fungsi tersebut memberikan output berupa hasil waktu CPU dan GPU yang dibutuhkan untuk melakukan perkalian matriks. Dengan menggunakan fungsi `gpuMatMul(A,B)` pada package gputools, proses perkalian matriks A dan B dilakukan dengan menggunakan GPU.

Hasil eksekusi program terlihat bahwa, untuk ukuran matriks yang kecil, CPU melakukan komputasi lebih baik dari GPU, namun untuk ukuran matriks yang lebih besar dari 2000, maka terlihat GPU jauh lebih cepat dibandingkan CPU.

2.4.2 CUDA pada Bahasa R

Mengakses GPU dengan menggunakan library CUDA dengan mengintegrasikan menggunakan bahasa C, pastikan bahwa pada komputer anda dilengkapi dengan hardware GPU Nvidia, dan telah terinstal CUDA. Langkah-Langkah untuk menggunakan library CUDA adalah sebagai berikut :

- Membuat interface sebagai penghubung antara R dan library CUDA.

- Compile dan membuat link shared object. Shared object berisi fungsi-fungsi pada bahasa C yang akan diakses oleh R.
- Load shared object.
- Eksekusi dan tes.

BAB 3

OPENCL

3.1 Pendahuluan

OpenCL *Open Computing Language* merupakan *library General Purpose Graphics Processing Unit* Computing (GPGPU) yang dikembangkan oleh Khronos (yang disponsori oleh Apple). OpenCL juga disebut sebagai sebuah *open standard* untuk pemrograman paralel pada sistem heterogen karena mendukung berbagai vendor GPU (*integrated* maupun *dedicated*) seperti Intel, AMD, NVIDIA, Apple dan ARM.

OpenCL merupakan *library* yang dapat berjalan di kebanyakan sistem karena *kernel* bahasanya merupakan subset dari C++ 14. Selain itu, OpenCL juga telah memiliki *language binding* dari bahasa pemrograman *high-level* seperti Microsoft.Net (NOpenCL dan OpenCL.Net), Erlang dan Python (PyOpenCL).

OpenCL saat ini sudah mencapai versi 2.0 dengan sejarah pengembangan [3] sebagai berikut:

- OpenCL 1.0
 - Model pemrograman dasar
- OpenCL 1.1 and 1.2
 - Teknik manajemen *memory*
 - Kontrol *resources* yang lebih baik
- OpenCL 2.0
 - Memaksimalkan penggunaan kapabilitas baru *hardware*
 - API pemrograman yang lebih baik
 - Kontrol *resources* yang lebih baik

3.1.1 Instalasi

Salah satu kelebihan yang dimiliki OpenCL dibanding *hardware-specific library* seperti NVIDIA CUDA adalah dukungan ke banyak vendor *hardware*. Untuk mencapai hal ini, OpenCL beradaptasi dengan karakteristik instalasi masing-masing

vendor sehingga setiap vendor memiliki prosedur instalasi OpenCL yang berbeda. Berikut daftar tautan panduan instalasi untuk beberapa vendor ternama:

- AMD <http://developer.amd.com/tools-and-sdks/opencl-zone>
- Intel <https://software.intel.com/en-us/intel-opencl>
- NVIDIA <https://developer.nvidia.com/opencl>

Contoh langkah-langkah instalasi OpenCL SDK pada Ubuntu 15.10 64-bit dengan NVIDIA 940M [1] adalah sebagai berikut:

1. Instal *driver* yang disarankan oleh versi Ubuntu 15.10 yaitu NVIDIA *driver* versi 352. Instalasi dapat dilakukan melalui menu *Additional Drivers* atau dengan mengetikkan perintah pada terminal:

```
$ sudo apt-get install nvidia-352
```

2. Setelah itu, instal CUDA dengan mengunduh *repository* CUDA Toolkit versi 7.5 untuk Ubuntu 15.04 (*.deb) di <https://developer.nvidia.com/cuda-downloads> dan menjalankan perintah berikut di *terminal*:

```
$ sudo dpkg -i cuda-repo-ubuntu1504-7-5-*_amd64.deb
$ sudo apt-get update
$ sudo apt-get install cuda-toolkit
```

Pastikan juga baris-baris ini ada di dalam file `~/.bashrc` (baris terbawah):

```
export CUDA_HOME=/usr/local/cuda-7.5
export LD_LIBRARY_PATH=${CUDA_HOME}/lib64
PATH=${CUDA_HOME}/bin:${PATH}
export PATH
```

Untuk memastikan bahwa driver dan CUDA sudah terinstal sempurna nama GPU (contoh NVIDIA 940M) harus terlihat ketika 2 kelompok perintah ini dipanggil:

Pertama:

```
$ nvidia-smi
```

Kedua:

```
$ cd $CUDA_HOME/samples/1_Utilities/deviceQuery
$ sudo make run
```

3. Terakhir, instal header OpenCL dengan perintah:

```
$ sudo apt-get install nvidia-352-dev nvidia-prime nvidia-modprobe nvidia-opencl-dev
```

3.1.2 Struktur Program

Struktur program OpenCL cukup berbeda dengan struktur program CUDA. Perbedaan mendasar adalah adanya proses kompilasi kernel di dalam program OpenCL di mana untuk CUDA proses tersebut tidak perlu dilakukan secara eksplisit pada program. Oleh karena itu, kernel pada program OpenCL biasanya diletakkan di file terpisah dengan ekstensi `*.cl`.

Struktur umum atau langkah-langkah pada program OpenCL adalah sebagai berikut:

1. Memilih *platform* yang tersedia
2. Memilih *device* pada *platform* yang tersedia
3. Membuat *Context*
4. Membuat *command queue*
5. Membuat *memory objects*
6. Membaca file *kernel*
7. Membuat *program object*
8. Mengkompilasi *kernel*
9. Membuat *kernel object*
10. Memasukkan *kernel arguments*
11. Menjalankan *kernel*

12. Membaca *memory object* (hasil proses *kernel*)

13. *Free memory objects*

Contoh program sederhana OpenCL *Single-Precision AX Plus Y* (SAXPY) dapat dilihat pada tautan berikut:

- Program utama <https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/PR2/opencl/saxpy.c>
- Kernel <https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/PR2/opencl/saxpy.cl>

3.1.3 Perbandingan Terminologi dengan CUDA

Bagi *programmer* yang sudah terbiasa dengan CUDA, pada bagian ini akan dipaparkan tabel-tabel konversi terminologi antara CUDA dan OpenCL [11]. Dengan tabel-tabel ini diharapkan *programmer* CUDA dapat lebih cepat memahami OpenCL dan mengkonversi program CUDA ke OpenCL.

Tabel 3.1: Terminologi Perangkat Keras

CUDA	OpenCL
Stream Multiprocessor (SM)	CU (Compute Unit)
Thread	Work-item
Block	Work-group
Global Memory	Global Memory
Constant Memory	Constant Memory
Shared Memory	Local Memory
Local Memory	Private Memory

Tabel 3.2: Qualifiers untuk fungsi Kernel

CUDA	OpenCL
<code>__global__</code> function	<code>__kernel</code> function
<code>__device__</code> function	N/A
<code>__constant__</code> variable	<code>__constant</code> variable
<code>__device__</code> variable	<code>__global</code> variable
<code>__shared__</code> variable	<code>__local</code> variable

Tabel 3.3: Indeks pada Kernel

CUDA	OpenCL
<code>gridDim</code>	<code>get_num_groups()</code>
<code>blockDim</code>	<code>get_local_size()</code>
<code>blockIdx</code>	<code>get_group_id()</code>
<code>threadIdx</code>	<code>get_local_id()</code>
<code>blockIdx * blockDim + threadIdx</code>	<code>get_global_id()</code>
<code>gridDim * blockDim</code>	<code>get_global_size()</code>

Tabel 3.4: Pemanggilan API

CUDA	OpenCL
<code>cudaGetDeviceProperties()</code>	<code>clGetDeviceInfo()</code>
<code>cudaMalloc()</code>	<code>clCreateBuffer()</code>
<code>cudaMemcpy()</code>	<code>clEnqueueReadBuffer()</code> <code>clEnqueueWriteBuffer()</code>
<code>cudaFree()</code>	<code>clReleaseMemObj()</code>
<code>kernel<<<...>>>()</code>	<code>clEnqueueNDRangeKernel()</code>

3.1.4 Library BLAS

Basic Linear Algebra Subprograms (BLAS) adalah *library* yang umum digunakan pada pemrograman paralel karena berisi subprogram perkalian matriks dan vektor dasar. BLAS [7] awalnya merupakan bagian dari *library* Fortran tetapi kemudian dikembangkan secara terpisah dan terbuka untuk bahasa C dan bahasa lainnya. BLAS terdiri dari 3 tingkat atau kelompok subprogram [6]:

- Level 1 BLAS: operasi skalar, vektor and vektor-vektor
- Level 2 BLAS: operasi matriks-vektor
- Level 3 BLAS: operasi matriks-matriks

OpenCL memiliki beberapa alternatif implementasi BLAS yang dikembangkan oleh beberapa pihak, yaitu:

1. CIBLAS

CIBLAS [8] merupakan implementasi *library* BLAS untuk OpenCL yang bersifat *opensource* yang dikembangkan oleh clMath¹. *Library* ini sudah mengimplementasikan BLAS secara lengkap (*level* 1, 2 dan 3) dan juga memiliki fitur optimasi khusus untuk AMD GPU. Sistem operasi yang didukung oleh *library* ini adalah Windows 7/8, Linux dan Mac OSX. Cara pemanggilan fungsi pada clBLAS dapat dilihat pada gambar 3.1.

¹<https://github.com/clMathLibraries>

```

/* Setup clBLAS */
err = clblasSetup( );

/* Prepare OpenCL memory objects and place matrices inside them. */
bufA = clCreateBuffer( ctx, CL_MEM_READ_ONLY, M * K * sizeof(*A),
                      NULL, &err );
bufB = clCreateBuffer( ctx, CL_MEM_READ_ONLY, K * N * sizeof(*B),
                      NULL, &err );
bufC = clCreateBuffer( ctx, CL_MEM_READ_WRITE, M * N * sizeof(*C),
                      NULL, &err );

err = clEnqueueWriteBuffer( queue, bufA, CL_TRUE, 0,
                          M * K * sizeof( *A ), A, 0, NULL, NULL );
err = clEnqueueWriteBuffer( queue, bufB, CL_TRUE, 0,
                          K * N * sizeof( *B ), B, 0, NULL, NULL );
err = clEnqueueWriteBuffer( queue, bufC, CL_TRUE, 0,
                          M * N * sizeof( *C ), C, 0, NULL, NULL );

/* Call clBLAS extended function. Perform gemm for the lower right sub-matrices */
err = clblasSgemm( clblasRowMajor, clblasNoTrans, clblasNoTrans,
                  M, N, K,
                  alpha, bufA, 0, lda,
                  bufB, 0, ldb, beta,
                  bufC, 0, ldc,
                  1, &queue, 0, NULL, &event );

/* Wait for calculations to be finished. */
err = clWaitForEvents( 1, &event );

/* Fetch results of calculations from GPU memory. */
err = clEnqueueReadBuffer( queue, bufC, CL_TRUE, 0,
                          M * N * sizeof(*result),
                          result, 0, NULL, NULL );

/* Release OpenCL memory objects. */
clReleaseMemObject( bufC );
clReleaseMemObject( bufB );
clReleaseMemObject( bufA );

/* Finalize work with clBLAS */
clblasTeardown( );

```

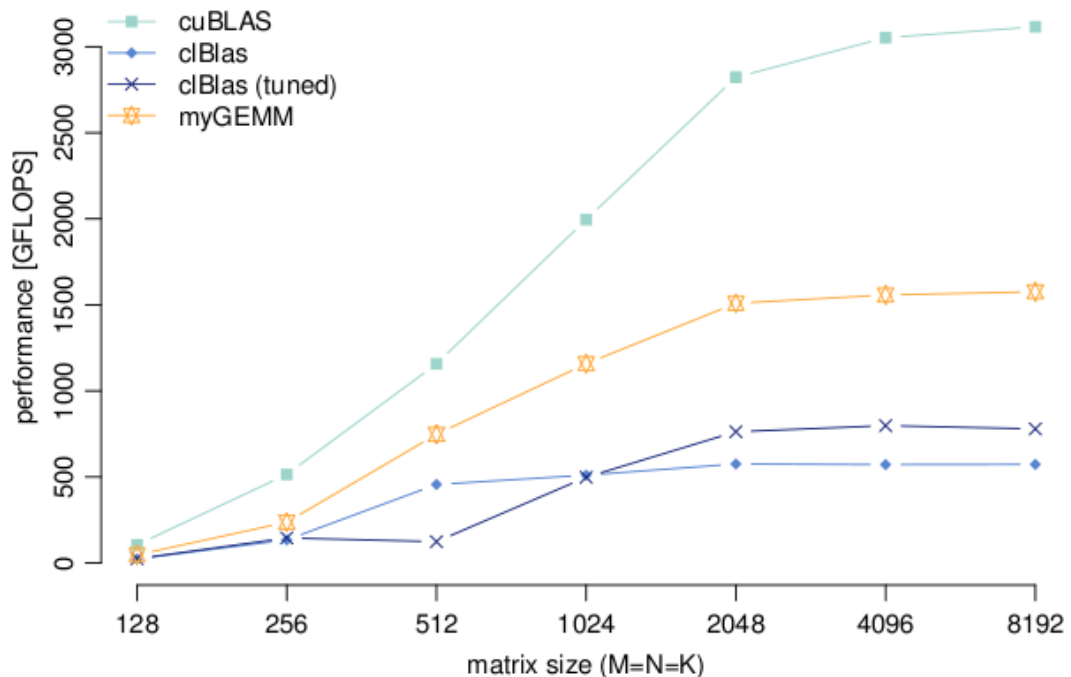
Gambar 3.1: Cara pemanggilan clblasSgemm pada clBLAS

2. MyGEMM

MyGEMM [10] adalah *library* yang dikembangkan oleh Cedric Nugteren yang juga bersifat *opensource*. Library ini dikembangkan karena ketidakpuasannya terhadap kinerja clBLAS [9] pada NVIDIA GPU. Sekalipun *library* ini dapat dioptimasi untuk NVIDIA GPU, implementasi BLAS yang ada baru *single-precision generalised matrix-multiplication* (SGEMM) saja. Cara pemakaiannya juga sama persis dengan memakai *kernel* OpenCL pada umumnya karena semua fungsi-fungsi BLAS diimplementasikan dalam bentuk file *kernel* OpenCL biasa.

Perbandingan kinerja clBLAS, MyGEMM dan CUBLAS (CUDA) [9] pada GPU NVIDIA Tesla K40 dapat dilihat pada gambar 3.2. Pada grafik tersebut terlihat bahwa kinerja CUDA dengan *library* CUBLAS jauh lebih baik dari semua

library OpenCL. Di sini Nugteren juga menunjukkan bahwa implementasi SGEMM pada myGEMM lebih baik dari clBLAS.



Gambar 3.2: Kinerja library BLAS pada OpenCL

3.1.5 Eksperimen

3.1.5.1 Device Query

OpenCL menyediakan *Application Programming Interface* (API) untuk mengecek *platform* dan *device* OpenCL yang ada di dalam sebuah sistem [4] [5], yaitu:

- `clGetPlatformID()`: mendapatkan daftar *platform* pada mesin
- `clGetPlatformInfo()`: mendapatkan informasi dari suatu *platform*
- `clGetDeviceID()`: mendapatkan daftar *device* pada *platform*
- `clGetDeviceInfo()`: mendapatkan informasi dari suatu *device*

API ini juga digunakan pada saat menjalankan *kernel* untuk memilih *platform* dan *device* yang ingin digunakan untuk menjalankan *kernel* tersebut.

Pada eksperimen ini, program `device_query.c`² akan memanggil API untuk mendapatkan informasi *platform* dan *device* kemudian menampilkannya seperti pada gambar 3.3.

²https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/PR2/opencl/device_query.c

```

yohanesgultom@Asus-K401LB: ~/Workspace/parallel-programming-assignment/PR2/opencv
yohanesgultom@Asus-K401LB:~/Workspace/parallel-programming-assignment/PR2/opencv$ ./device_query.o
Platform - 1
  1.1 CL_PLATFORM_NAME: NVIDIA CUDA
  1.2 CL_PLATFORM_VENDOR: NVIDIA Corporation
  1.3 CL_PLATFORM_VERSION: OpenCL 1.2 CUDA 7.5.23
  1.4 CL_PLATFORM_PROFILE: FULL_PROFILE
  1.5 CL_PLATFORM_EXTENSIONS: cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_options cl_nv_device_attribute_query cl_nv_pragma_unroll cl_nv_copy_opts
Device - 1:
  CL_DEVICE_NAME: GeForce 940M
  CL_DEVICE_VENDOR: NVIDIA Corporation
  CL_DRIVER_VERSION: 352.63
  CL_DEVICE_VERSION: OpenCL 1.2 CUDA
  CL_DEVICE_MAX_COMPUTE_UNITS: 3
yohanesgultom@Asus-K401LB:~/Workspace/parallel-programming-assignment/PR2/opencv$

```

Gambar 3.3: Program untuk mendapatkan informasi platform dan device OpenCL

3.1.5.2 Single-Precision AX Plus Y (SAXPY)

Single-Precision AX Plus Y (SAXPY) adalah program yang melakukan kombinasi perkalian skalar dan penjumlahan vektor $z = \alpha x + y$ di mana x, y, z : vektor dan α : skalar. Program ini merupakan contoh program yang sederhana tapi cukup merepresentasikan sintaks operasi aljabar linear dari sebuah bahasa program atau *library* sehingga sering dianggap sebagai "*hello world*" untuk program aljabar linear.

Program SAXPY pada eksperimen ini terdiri dari dua buah file yaitu `saxpy.c`³ (program utama) dan `saxpy.cl`⁴ (*kernel*). Program ini akan menghitung SAXPY dengan vektor yang berukuran 1.024 elemen dan mencetak hasilnya seperti pada gambar 3.4.

³<https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/PR2/opencv/saxpy.c>

⁴<https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/PR2/opencv/saxpy.cl>

```

yohanesgultom@Asus-K401LB: ~/Workspace/parallel-programming-assignment/PR2/opencl$ gcc -o saxpy.o saxpy.c -lOpenCL
saxpy.c: In function 'main':
saxpy.c:65:5: warning: 'clCreateCommandQueue' is deprecated (declared at /usr/include/CL/cl.h:1359) [-Wdeprecated-declarations]
    cl_command_queue command_queue = clCreateCommandQueue(context, device_list[0], 0, &clStatus);
    ^
yohanesgultom@Asus-K401LB: ~/Workspace/parallel-programming-assignment/PR2/opencl$ ./saxpy.o
2.000000 * 0.000000 + 1024.000000 = 1024.000000
2.000000 * 1.000000 + 1023.000000 = 1025.000000
2.000000 * 2.000000 + 1022.000000 = 1026.000000
2.000000 * 3.000000 + 1021.000000 = 1027.000000
2.000000 * 4.000000 + 1020.000000 = 1028.000000
2.000000 * 5.000000 + 1019.000000 = 1029.000000
2.000000 * 6.000000 + 1018.000000 = 1030.000000
2.000000 * 7.000000 + 1017.000000 = 1031.000000
2.000000 * 8.000000 + 1016.000000 = 1032.000000
2.000000 * 9.000000 + 1015.000000 = 1033.000000
2.000000 * 10.000000 + 1014.000000 = 1034.000000
2.000000 * 11.000000 + 1013.000000 = 1035.000000
2.000000 * 12.000000 + 1012.000000 = 1036.000000
2.000000 * 13.000000 + 1011.000000 = 1037.000000
2.000000 * 14.000000 + 1010.000000 = 1038.000000
2.000000 * 15.000000 + 1009.000000 = 1039.000000
2.000000 * 16.000000 + 1008.000000 = 1040.000000
2.000000 * 17.000000 + 1007.000000 = 1041.000000
2.000000 * 18.000000 + 1006.000000 = 1042.000000
2.000000 * 19.000000 + 1005.000000 = 1043.000000
2.000000 * 20.000000 + 1004.000000 = 1044.000000
2.000000 * 21.000000 + 1003.000000 = 1045.000000
2.000000 * 22.000000 + 1002.000000 = 1046.000000
2.000000 * 23.000000 + 1001.000000 = 1047.000000
2.000000 * 24.000000 + 1000.000000 = 1048.000000
2.000000 * 25.000000 + 999.000000 = 1049.000000
2.000000 * 26.000000 + 998.000000 = 1050.000000
2.000000 * 27.000000 + 997.000000 = 1051.000000
2.000000 * 28.000000 + 996.000000 = 1052.000000
2.000000 * 29.000000 + 995.000000 = 1053.000000
2.000000 * 30.000000 + 994.000000 = 1054.000000
2.000000 * 31.000000 + 993.000000 = 1055.000000
2.000000 * 32.000000 + 992.000000 = 1056.000000
2.000000 * 33.000000 + 991.000000 = 1057.000000
2.000000 * 34.000000 + 990.000000 = 1058.000000
2.000000 * 35.000000 + 989.000000 = 1059.000000
2.000000 * 36.000000 + 988.000000 = 1060.000000
2.000000 * 37.000000 + 987.000000 = 1061.000000
2.000000 * 38.000000 + 986.000000 = 1062.000000
2.000000 * 39.000000 + 985.000000 = 1063.000000
2.000000 * 40.000000 + 984.000000 = 1064.000000
2.000000 * 41.000000 + 983.000000 = 1065.000000
2.000000 * 42.000000 + 982.000000 = 1066.000000
2.000000 * 43.000000 + 981.000000 = 1067.000000

```

Gambar 3.4: Program SAXPY 1024 elemen

3.1.5.3 Perkalian Matriks Bujursangkar

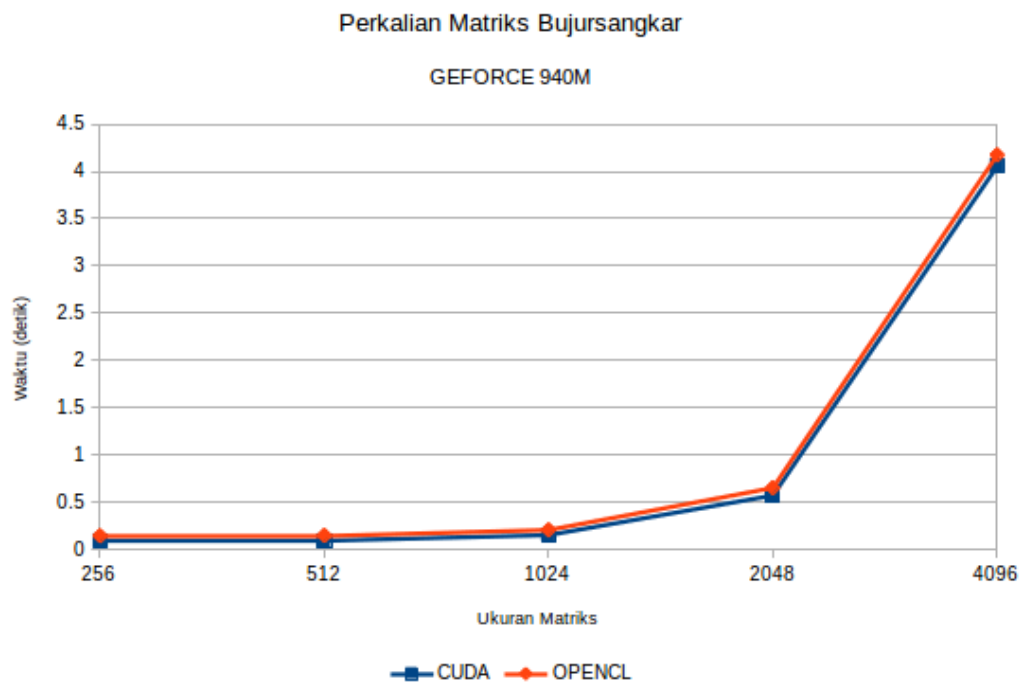
Eksperimen ini mencoba membandingkan kinerja perkalian matriks bujursangkar OpenCL [2] dengan CUDA. Program yang digunakan adalah `mmul_cuda.cu`⁵ yang melakukan perkalian matriks bujursangkar dengan CUDA. Sedangkan perkalian matriks bujursangkar OpenCL terdiri dari dua file yaitu `mmul_openc1.c`⁶ (program utama) dan `mmul_openc1.cl`⁷ (*kernel*).

Hasil eksperimen pada mesin dengan NVIDIA 940M, memberikan hasil seperti grafik pada gambar 3.5. Pada grafik tersebut bahwa untuk ukuran matriks 256x256 sampai 4096x2096, program perkalian matriks dengan OpenCL selalu sedikit lebih lambat dari program yang dibuat dengan CUDA. Sekalipun demikian untuk kasus ini, perbedaan kecepatan antara OpenCL dan CUDA sangatlah kecil, yaitu di bawah 0.3 detik.

⁵https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/PR2/opencl/mmul_cuda.cu

⁶https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/PR2/opencl/mmul_openc1.c

⁷https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/PR2/opencl/mmul_openc1.cl



Gambar 3.5: Perbandingan perkalian matriks bujursangkar OpenCL dengan CUDA

3.1.5.4 Gaussian Filter Blurring pada Gambar Bitmap

Gaussian filter blurring adalah teknik untuk membuat gambar menjadi *blur* (kabur) (seperti gambar 3.6) dengan memanfaatkan perkalian dengan matriks yang dibangun menggunakan persamaan Gaussian 3.1.

$$g(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (3.1)$$

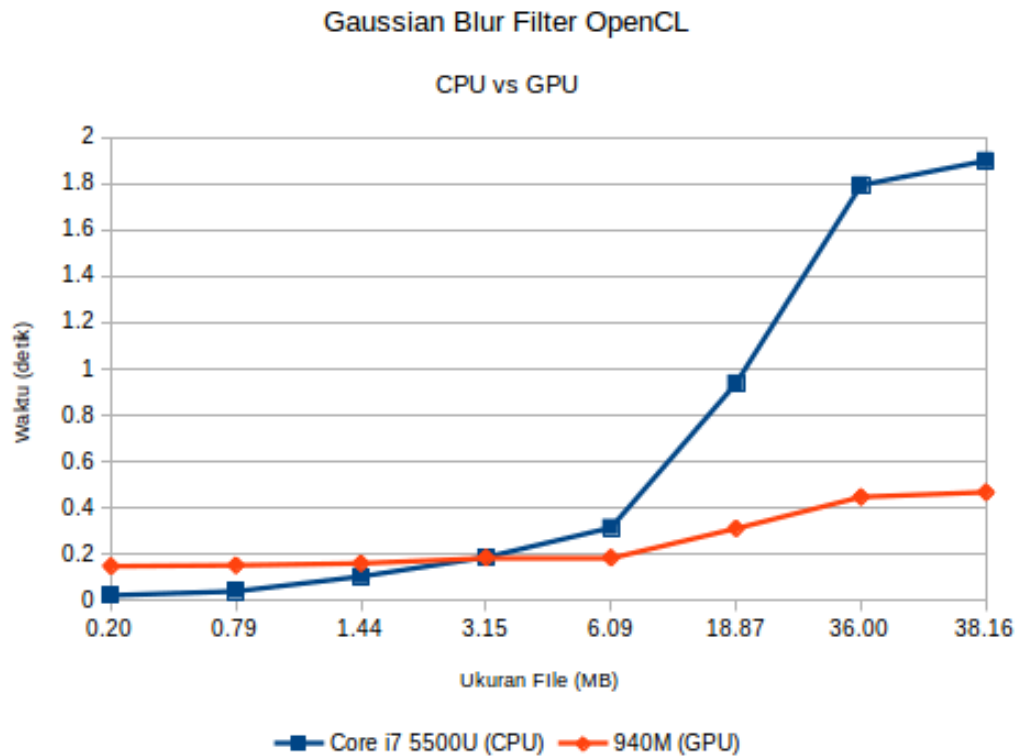


Gambar 3.6: Proses Gaussian Filter Blurring pada gambar

Eksperimen ini mencoba membandingkan kinerja Gaussian *blurring* dengan CPU dan dengan GPU (menggunakan OpenCL) [12]. Program yang digunakan terdiri dari kumpulan *library file*⁸ dengan file utama `main.c` (program utama) dan

⁸<https://github.com/yohanesgultom/parallel-programming-assignment/tree/master/PR2/opencl/>

`kernel.cl` (*kernel*) yang akan melakukan Gaussian *blurring* dengan CPU dan GPU serta menghitung waktu eksekusinya.



Gambar 3.7: Kinerja Gaussian Blurring CPU vs GPU

Hasil eksperimen yang dilakukan menggunakan gambar BMP dengan ukuran 0,2 MB - 38,16 MB memberikan hasil seperti grafik pada gambar 3.7. Pada hasil eksperimen tersebut terlihat bahwa pada gambar BMP dengan ukuran di bawah 3,15 MB, CPU (Intel Core i7 5500U) dapat melakukan Gaussian *blurring* lebih cepat dari GPU (NVIDIA 940M). Tetapi ketika gambar BMP yang diproses sudah lebih besar dari 3,15 MB, terlihat bahwa GPU dapat menyelesaikan proses dengan lebih cepat. Bahkan untuk ukuran gambar paling besar (38,16 MB), waktu yang dibutuhkan GPU untuk melakukan blurring hanya 0,4 detik atau sekitar $\frac{1}{5}$ dari waktu yang dibutuhkan CPU.

BAB 4

KONTRIBUSI

Kontribusi tiap anggota kelompok pada tugas ini adalah sebagai berikut:

Muhammad Fathurachman:

- Kajian dan eksperimen pemrograman paralel R

Otniel Yosi Viktorisa:

- Kajian pemrograman paralel R

Yohanes Gultom:

- Kajian dan eksperimen pemrograman OpenCL

DAFTAR REFERENSI

- [1] Askubuntu.com. *How to make OpenCL work on 14.10 + Nvidia 331.89 drivers?*. 10 Oktober 2014. <http://askubuntu.com/questions/541114/how-to-make-opencl-work-on-14-10-nvidia-331-89-drivers>.
- [2] Zaius. *Matrix Multiplication 1 (OpenCL)*. 22 September 2009. <http://gpgpu-computing4.blogspot.co.id/2009/09/matrix-multiplication-1.html>.
- [3] Mukherjee, S et al. *Exploring the Features of OpenCL 2.0*. 2015. IWOCL
- [4] Banger, R, Bhattacharyya .K. *OpenCL Programming by Example*. 2013. Packt publishing
- [5] Stackoverflow. *What is the right way to call clGetPlatformInfo?*. 21 Juni 2013. <http://stackoverflow.com/questions/17240071/what-is-the-right-way-to-call-clgetplatforminfo>.
- [6] Netlib.org. *About BLAS*. 15 November 2015. <http://www.netlib.org/blas/>.
- [7] Wikipedia. *Basic Linear Algebra Subprograms*. 10 Mei 2016. https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms.
- [8] ClMath. *ClBLAS*. 2016. <https://github.com/clMathLibraries/clBLAS>.
- [9] Nugteren, C. *ClBLAS Tutorial*. 2014. <http://www.cedricnugteren.nl/tutorial.php?page=1>.
- [10] Nugteren, C. *MyGEMM*. 2014. <https://github.com/cnugteren/myGEMM>.
- [11] sharcnet.ca. *Porting CUDA to OpenCL*. 2014. https://www.sharcnet.ca/help/index.php/Porting_CUDA_to_OpenCL.
- [12] Karapetsas, L. *Playing with OpenCL: Gaussian Blurring*. 2014. <http://blog.refu.co/?p=663>.