Национальный Исследовательский Университет
Московский Авиационный Институт(МАИ)

Лабораторная работа №1

Освоение программного обеспечения среды программирования OPENCL

Выполнила Волушкова Александра Юрьевна
группа 08-406

Москва 2012

Содержание

Постановка задачи

1. Установить OPENCL
2. Интегрировать OPENCL в Visual Studio 2008
3. Выполнить задачу - перемножить два массива

Цель работы

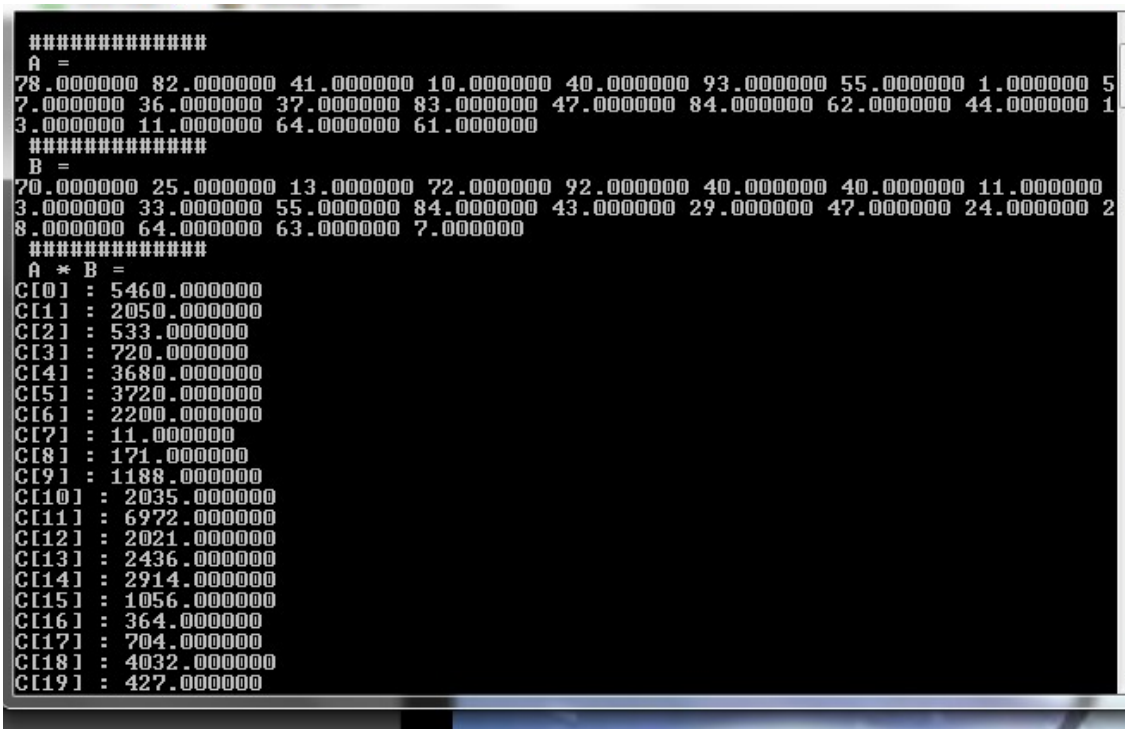
Ознакомиться с программным обеспечение для параллельного программирование на GPU.

Результаты

Сгенерируем 2 массива, размерностью MEM_SIZE A,B со случайных числами. Результат выполнения — третий массив C.

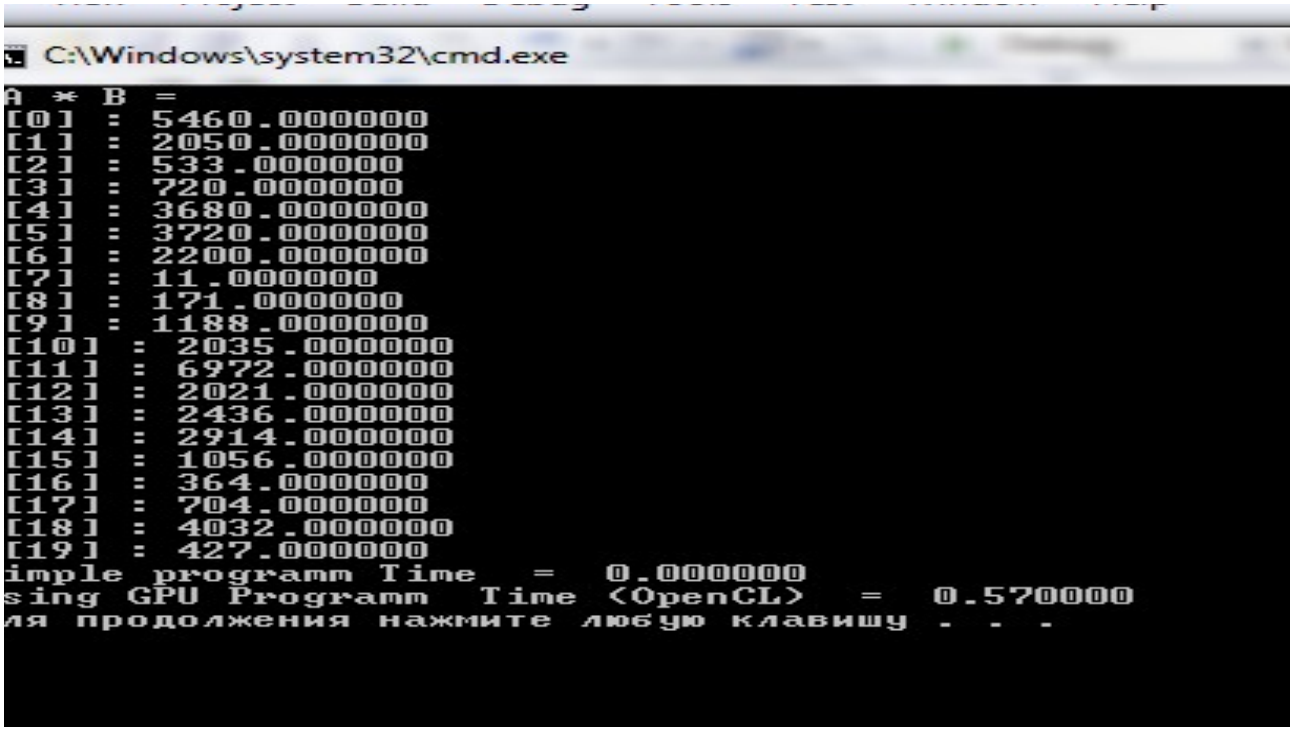
$C[i] = A[i]*B[i], i = \overline{1..MEM\_SIZE}$

Пример -



Сравнение производительности CPU и GPU

Тип процессора	A6-4455M 2.1 ГГц
Оперативная память (RAM)	6 ГБ
Графический контроллер	Radeon HD 7500G



Видно, что использование видеокарты было бессмысленным. Это объясняется довольно большой мощностью процессора, а так же поддержкой многопоточных вычислений(4 ядра). Так же стоит отметить крайне малый объем задачи, что не оправдывает копирование памяти.

Вывод

Использование GPU имеет смысл при решении задач с большим количеством вычислений.Если задача решается за приемлемое время на CPU, то использовать GPU не нужно. При малых объемах задачи использование GPU иногда дает даже отрицательный результат.

Список литературы

1. OpenCL Programming Guide,*Aaftab Munshi, Benedict R. Gaster, Timothy G., Mattson James, Fung Dan Ginsburg, Edwards Brothers* in Ann Arbor, Michigan, July 2011
2. http://opencl.codeplex.com/wikipage?title=OpenCL%20Tutorials%20-%201
3. *Heterogeneous* Computing with OpenCL Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, Dana Schaa,2012 Advanced Micro Devices, Inc. Published by Elsevier Inc.
4. *OpenCL in Action,* MATTHEW SCARPINO*, ©2012 by Manning Publications Co.*

Приложение

Ядро

```
#pragma OPENCL EXTENSION cl_khr_byte_addressable_store : enable

__kernel void hello(__global const float* a,__global const float* b,__global float* c)
{
  int gid = get_global_id(0);
```

```
  c[gid] = a[gid]*b[gid];
}
```

Основная программа

```
#include <stdio.h>
#include <stdlib.h>

#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif

#define MEM_SIZE (20)
#define MAX_SOURCE_SIZE (0x100000)

#include <time.h>
#include <ctime>

int main()
{
        cl_mem Amobj = NULL;
        cl_mem Bmobj = NULL;
        cl_mem Cmobj = NULL;
        cl_platform_id platform_id = NULL;
        cl_device_id device_id = NULL;
        cl_context context = NULL;
        cl_command_queue command_queue = NULL;
        cl_mem memobj = NULL;
        cl_program program = NULL;
        cl_kernel kernel = NULL;
        cl_uint ret_num_devices;
        cl_uint ret_num_platforms;
        cl_int ret;

        float mem[MEM_SIZE];

        FILE *fp;
        const char fileName[] = "hello.cl";
        size_t source_size;
        char *source_str;
        cl_int i;

        /* Load kernel source code */
        fp = fopen(fileName, "r");
        if (!fp) {
                fprintf(stderr, "Failed to load kernel.\n");
                exit(1);
        }
        source_str = (char *)malloc(MAX_SOURCE_SIZE);
        source_size = fread( source_str, 1, MAX_SOURCE_SIZE, fp );
        fclose( fp );

        /*Initialize Data */
        float *A;
        float *B;
        float *C;
        float *C1;
        A = new float [MEM_SIZE];
```

```
	B = new float [MEM_SIZE];
	C = new float [MEM_SIZE];
	C1 = new float [MEM_SIZE];
	srand(time(0));// áåç ýòîãî ÷ìñëà áóäóò îäèíàêîâûå
	for(int i=0;i<MEM_SIZE;i++){
		A[i] = rand()%100;
		B[i] = rand()%100;
	}
	/*Print input data*/

	printf("\n ############# \n A = \n");
	for(int i=0;i<MEM_SIZE;i++){
		printf("%f ", A[i]);
	}
	printf("\n ############# \n B =  \n");
	for(int i=0;i<MEM_SIZE;i++){
		printf("%f ", B[i]);
	}
	printf("\n ############# \n A * B =  \n");

	//		double start_opencl = GetTickCount();
	/* Get platform/device information */
	double duration_cl;
	clock_t start_cl, finish_cl;
	start_cl = clock(); //âðåìÿ íà÷àëà âûïîëíåíèÿ ïðîãðàììû
	ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);
	ret = clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_DEFAULT, 1, &device_id, &ret_num_devices);
	/* Create OpenCL Context */
	context = clCreateContext( NULL, 1, &device_id, NULL, NULL, &ret);
	/* Create Command Queue */
	command_queue = clCreateCommandQueue(context, device_id, 0, &ret);
	/* Create Buffer Object */
	Amobj = clCreateBuffer(context, CL_MEM_READ_WRITE, MEM_SIZE * sizeof(float), NULL, &ret);
	Bmobj = clCreateBuffer(context, CL_MEM_READ_WRITE,MEM_SIZE * sizeof(float), NULL, &ret);
	Cmobj = clCreateBuffer(context, CL_MEM_READ_WRITE,MEM_SIZE * sizeof(float), NULL, &ret);
	/* Transfer data to memory buffer */
	ret = clEnqueueWriteBuffer(command_queue, Amobj, CL_TRUE, 0, MEM_SIZE * sizeof(float), A, 0, NULL,
NULL);
	ret = clEnqueueWriteBuffer(command_queue, Bmobj, CL_TRUE, 0, MEM_SIZE * sizeof(float), B, 0, NULL,
NULL);
	/* Create Kernel program from the read in source */
	program = clCreateProgramWithSource(context, 1, (const char **)&source_str, (const size_t *)&source_size,
&ret);
	/* Build Kernel Program */
	ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
	/* Create OpenCL Kernel */
	kernel = clCreateKernel(program, "hello", &ret);
	/* Set OpenCL kernel arguments */
	ret = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *)&Amobj);
	ret = clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *)&Bmobj);
	ret = clSetKernelArg(kernel, 2, sizeof(cl_mem), (void *)&Cmobj);
	size_t global_work_size[1] = {MEM_SIZE};
	size_t local_work_size[1]  = {1};
	size_t global_item_size = 1;
	size_t local_item_size = 1;
	/* Execute OpenCL kernel */
	ret = clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL,
		global_work_size, local_work_size, 0, NULL, NULL);
	/* Transfer result from the memory buffer */
	ret = clEnqueueReadBuffer(command_queue, Cmobj, CL_TRUE, 0, MEM_SIZE * sizeof(float), C, 0, NULL,
```

```
NULL);
	finish_cl = clock(); //âðåìÿ îêîí÷àíèÿ âûïîëíåíèÿ ïðîãðàììû
	/* Display result */
	duration_cl = (double)(finish_cl - start_cl) / CLOCKS_PER_SEC; //âðåìÿ âûïîëíåíèÿ ïðîãðàììû
	for(i=0; i<MEM_SIZE; i++) {
		printf("C[%d] : %f\n", i, C[i]);
	}
	/*Simple programm*/
	float duration;
	clock_t start, finish;
	start = clock();
	for(i=0; i<MEM_SIZE; i++) {
		C1[i] = A[i]*B[i];
	}
	finish = clock();
	duration = (float)(finish - start) / CLOCKS_PER_SEC; //âðåìÿ âûïîëíåíèÿ ïðîãðàììû
	printf("simple programm Time  = % lf \n" ,duration);
	printf("Using GPU Programm  Time (OpenCL)  = % lf \n", duration_cl);
	/* Finalization */
	ret = clFlush(command_queue);
	ret = clFinish(command_queue);
	ret = clReleaseKernel(kernel);
	ret = clReleaseProgram(program);
	ret = clReleaseMemObject(Amobj);
	ret = clReleaseMemObject(Bmobj);
	ret = clReleaseMemObject(Cmobj);
	ret = clReleaseCommandQueue(command_queue);
	ret = clReleaseContext(context);
	free(A);
	free(B);
	return 0;
}
```