Донецкий Национальный Технический Университет

Лабораторная работа № 4

«Знакомство с оптимизацией памяти в GPGPU

на базе технологии Nvidia Cuda»

Выполнил:

ст. группы ИПОИм -17

Лысенко А. С.

Проверила:

доцент каф. ПМИ

Назарова И. А.

Покровск 2017

**Задание к лабораторной работе**

1. Практически проверить изложенные в теоретической работе сведения и наглядно продемонстрировать полученные результаты

2. Модифицировать созданные в рамках 3 лабораторной работы с точки зрения рассматриваемой теоретической информации. Сравнить результаты работы соответствующих программ

1. Практическая проверка изложенных в теоретической работе сведений и наглядно продемонстрированы полученные результаты.

Выравнивание размеров используемых типов.

Обращение к глобальной памяти происходит через чтение/запись 32/64/128-битовых слов. Важно, что адрес, по которому происходит доступ, должен быть выровнен по размеру слова (кратен размеру слова в байтах).



Рис 1. - Пример выровненного и невыровненного 4-х байтного слова.

Все функции выделяющие глобальную память выделяют её выровненной по 256 байтам. Пусть есть массив из следующих структур в глобальной памяти:

struct vector {

float x;

float y;

float z;

};

Каждый элемент массива — 12 байт. Адрес первого элемента выровнен по 16 байтам, но адрес второго элемента — нет, и его чтение потребует двух обращений.

Выравнивание:

struct \_\_align\_\_(16) vector3 {

float x;

float y;

float z;

};

Теперь все элементы массива будут хранится по адресам кратным 16 байтам.

Для проверки оптимизации при использовании структур с выравниванием и без, использую функции ядра в которых произвожу вычисления с заданными структурами. Замеры времени будут проводиться с помощью cuda event's в cuda runtime API.

Примеры кода функции ядра:

\_\_global\_\_ void testFunc()

{

vector3 d;

d.x = 257;

d.y = 663;

d.z = 771;

int idx = d.x \* d.y + d.z;

for(int i = 0; i < ITERATIONS; i++)

{

int g =+ (d.x \* d.x \* d.z) / d.x;

idx += g;

}

};

\_\_global\_\_ void testFunc1()

{

vector b;

b.x = 257;

b.y = 663;

b.z = 771;

int idx = b.x \* b.y + b.z;

for(int i = 0; i < ITERATIONS; i++)

{

int c = (b.x \* b.x \* b.z) / b.x;

idx += c;

}

};

В самих функциях выполняются обычные операции при вычислении индекса потока с операциями со структурами данных определенных типов.

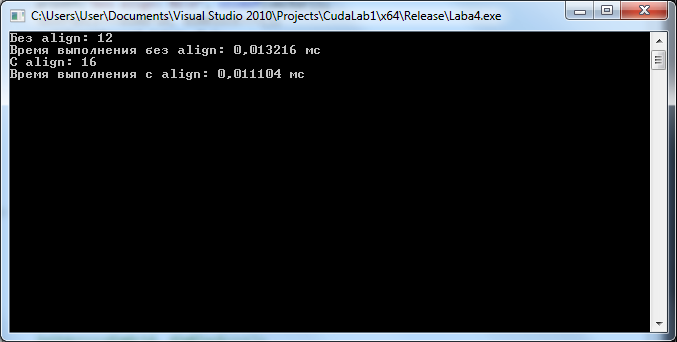


Рис. 2 – Пример выполнения программы с выравниванием размеров используемых типов

Продемонстрирую ещё один пример с выравниванием размеров используемых типов.

Пример кода функции ядра:

\_\_device\_\_ float3 data[512];

\_\_global\_\_ void initData()

{

int idx = threadIdx.x;

data[idx] = make\_float3(idx, idx, idx);

};

\_\_device\_\_ float4 dataFast[512];

\_\_global\_\_ void initDataFast()

{

int idx = threadIdx.x;

dataFast[idx] = make\_float4(idx, idx, idx, 0);

};

cudaEvent\_t start , stop;

cudaEventCreate (&start);

cudaEventCreate (&stop);

dim3 dimGrid(256,2,1); // кол-во блоков

dim3 dimBlock(16,16,1); // кол-во потоков

cudaEventRecord (start, 0 );

initData<<<dimGrid, dimBlock>>>();

cudaEventRecord (stop, 0 );

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("Время выполнения int3: %.4f мс\n", gpuTime);

cudaEventRecord (start, 0 );

initDataFast<<<dimGrid, dimBlock>>>();

cudaEventRecord (stop, 0 );

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("Время выполнения int4: %.4f мс\n", gpuTime);

В данном примере сравнивается время работы выравненного и невыравненного вектора в глобальной памяти.

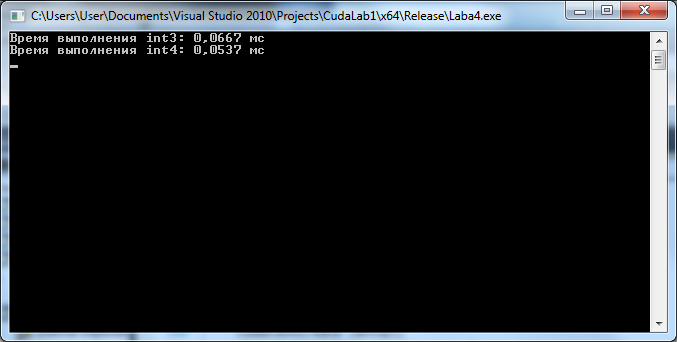


Рис. 3 – Пример работы выравненного и невыравненного вектора в глобальной памяти.

Использование объединенных запросов.

GPU имеет возможность объединять несколько запросов к глобальной памяти в один (coalescing). Все обращения MP к памяти происходят независимо для каждой половины warp’a. Максимальное объединение — все запросы одного полу-warp’a удается объединить в один большой запрос на чтение из глобальной памяти.

Все нити обращаются к 32(Computing Platform от 1.2) /32/64-битовым словам, давая в результате один 32(CP от 1.2) /64/128-байтный блок;

Получившийся блок выровнен по своему размеру (адрес кратен 32(CP от 1.2) /64/128);

Все 16 слов, к которым обращаются нити лежат в пределах одного блока;

Нити обращаются к словам последовательно: k–ая нить обращается к k–му слову; допускается, что отдельные нити пропустят обращение к соответствующим словам.

Если нити полу-warp’a не удовлетворяют любому условию, то каждое обращение к памяти происходит как отдельная транзакция!

Ещё один пример оптимизации это использование структур массивов а не массивов структур. Позволит уменьшить количество запросов в глобальную память за счет объединений.

Пример кода функции ядра:

\_\_device\_\_ vec3 data1[SIZE]; // инициализация массива структур

\_\_global\_\_ void initDataStruct()

{

int idx = blockDim.x \* blockIdx.x + threadIdx.x;

data1[idx].x = idx;

data1[idx].y = idx \* 2;

data1[idx].z = idx \* 3;

}

\_\_device\_\_ float x[SIZE]; //инициализация структур массивов

\_\_device\_\_ float y[SIZE]; //инициализация структур массивов

\_\_device\_\_ float z[SIZE]; //инициализация структур массивов

\_\_global\_\_ void initArr()

{

int idx = blockDim.x \* blockIdx.x + threadIdx.x;

x[idx] = idx;

y[idx] = idx \* 2;

z[idx] = idx \* 3;

};

cudaEventRecord (start, 0 );

initDataStruct<<<dimGrid, dimBlock>>>();

cudaEventRecord (stop, 0 );

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("Время выполнения массива структур: %.4f мс\n", gpuTime);

gpuTime = 0;

cudaEventRecord (start, 0 );

initArr<<<dimGrid, dimBlock>>>();

cudaEventRecord (stop, 0 );

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("Время выполнения структур массивов: %.4f мс\n", gpuTime);\*/

cudaDeviceSynchronize();

Комментариями обозначены места объявления данных

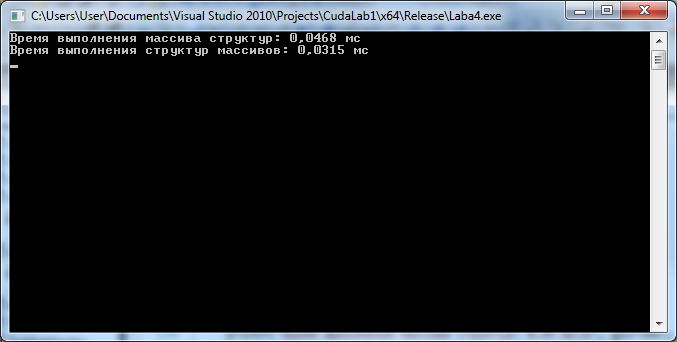


Рис 4. – Пример работы программы с использованием объединенных запросов.

2. Модифицировать созданные в рамках 3 лабораторной работы с точки зрения рассматриваемой теоретической информации. Сравнить результаты работы соответствующих программ.

В лабораторной работе 3 происходит оптимизация с помощью использования структуры массивов.

Пример кода с объявлением и инициализацией:

// Функція транспонування матриці без використання глобальної пам'яті

// \* inputMatrix - покажчик на вихідну матрицю

// \* outputMatrix - покажчик на матрицю результат

// \* width - ширина вихідної матриці (вона ж висота матриці-результату)

// \* height - висота вихідної матриці (вона ж ширина матриці-результату)

\_\_global\_\_ void transposeMatrixGlobal(float\* inputMatrix, float\* outputMatrix, int width, int height)

{

// Розрахунок індексів матриці

int xIndex = blockDim.x \* blockIdx.x + threadIdx.x;

int yIndex = blockDim.y \* blockIdx.y + threadIdx.y;

if ((xIndex < width) && (yIndex < height)) {

// Лінійний індекс елемента рядки вихідної матриці

int inputIdx = xIndex + width \* yIndex;

// Лінійний індекс елемента стовпця матриці-результату

int outputIdx = yIndex + height \* xIndex;

// Встановлення елементу

outputMatrix[outputIdx] = inputMatrix[inputIdx];

}

}

Ещё хочу рассказать о топ что оптимизацию можно осуществить на этапе компиляции программы для этого нужно выбирать конфигурацию Release а не Debug при использовании Release компилятор перестаёт выполнять некоторые проверки на итерации и выключает из списка ненужные пространства имён что приводит к оптимизации проекта. Демонстрации примера приведена ниже:

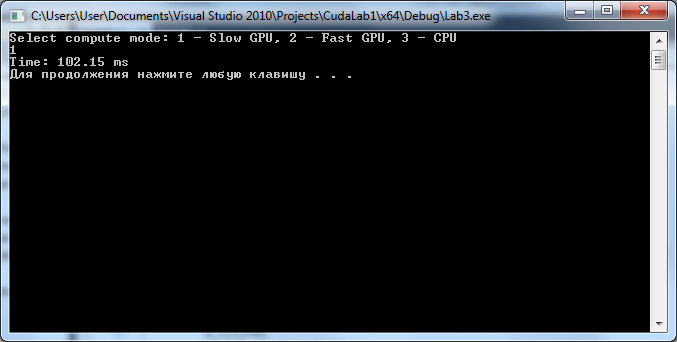


Рис. 5 – Пример выполнения конфигурации Debug

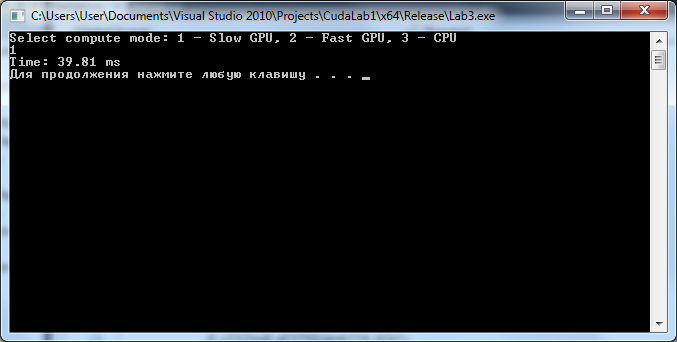


Рис. 5 – Пример выполнения конфигурации Release

Как видим увеличение скорости выполнения 2,57 раза.

Листинг программы

Оптимизация памяти с помощью запросов и типов

Код на Си

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

#include <conio.h>

#include <clocale>

#include <ctime>

#define SIZE 256

#define ITERATIONS 20000

struct \_\_align\_\_(16) vec3

{

float x;

float y;

float z;

};

\_\_device\_\_ vec3 data1[SIZE]; // инициализация массива струтур

\_\_global\_\_ void initDataStruct()

{

int idx = blockDim.x \* blockIdx.x + threadIdx.x;

data1[idx].x = idx;

data1[idx].y = idx \* 2;

data1[idx].z = idx \* 3;

}

\_\_device\_\_ float x[SIZE]; //инициализация структур массивов

\_\_device\_\_ float y[SIZE]; //инициализация структур массивов

\_\_device\_\_ float z[SIZE]; //инициализация структур массивов

\_\_global\_\_ void initArr()

{

int idx = blockDim.x \* blockIdx.x + threadIdx.x;

x[idx] = idx;

y[idx] = idx \* 2;

z[idx] = idx \* 3;

};

\_\_device\_\_ float3 data[512];

\_\_global\_\_ void initData()

{

int idx = threadIdx.x;

data[idx] = make\_float3(idx, idx, idx);

};

\_\_device\_\_ float4 dataFast[512];

\_\_global\_\_ void initDataFast()

{

int idx = threadIdx.x;

dataFast[idx] = make\_float4(idx, idx, idx, 0);

};

struct vector {

float x;

float y;

float z;

};

struct \_\_align\_\_(16) vector3 {

float x;

float y;

float z;

};

\_\_global\_\_ void testFunc()

{

vector3 d;

d.x = 257;

d.y = 663;

d.z = 771;

int idx = d.x \* d.y + d.z;

for(int i = 0; i < ITERATIONS; i++)

{

int g =+ (d.x \* d.x \* d.z) / d.x;

idx += g;

}

};

\_\_global\_\_ void testFunc1()

{

vector b;

b.x = 257;

b.y = 663;

b.z = 771;

int idx = b.x \* b.y + b.z;

for(int i = 0; i < ITERATIONS; i++)

{

int c = (b.x \* b.x \* b.z) / b.x;

idx += c;

}

};

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Rus");

float gpuTime;

float number;

cudaEvent\_t start , stop;

cudaEventCreate (&start);

cudaEventCreate (&stop);

dim3 dimGrid(256,2,1); // кол-во блоков

dim3 dimBlock(16,16,1); // кол-во потоков

cudaEventRecord (start, 0 );

initData<<<dimGrid, dimBlock>>>();

cudaEventRecord (stop, 0 );

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("Время выполнения int3: %.4f мс\n", gpuTime);

cudaEventRecord (start, 0 );

initDataFast<<<dimGrid, dimBlock>>>();

cudaEventRecord (stop, 0 );

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("Время выполнения int4: %.4f мс\n", gpuTime);

cudaEventRecord (start, 0 );

printf("Без align: %i\n", sizeof(vector));

testFunc1<<<dimGrid, dimBlock>>>();

cudaEventRecord (stop, 0 );

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("Время выполнения без align: %.6f мс\n", gpuTime);

cudaEventRecord (start, 0 );

printf("С align: %i\n", sizeof(vector3));

testFunc<<<dimGrid, dimBlock>>>();

cudaEventRecord (stop, 0 );

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("Время выполнения c align: %.6f мс\n", gpuTime);

cudaEventRecord (start, 0 );

initDataStruct<<<dimGrid, dimBlock>>>();

cudaEventRecord (stop, 0 );

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("Время выполнения массива структур: %.4f мс\n", gpuTime);

gpuTime = 0;

cudaEventRecord (start, 0 );

initArr<<<dimGrid, dimBlock>>>();

cudaEventRecord (stop, 0 );

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("Время выполнения структур массивов: %.4f мс\n", gpuTime);

cudaDeviceSynchronize();

\_getch();

return 0;

}