МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики

Кафедра технической кибернетики

ЛАБОРАТОРНАЯ РАБОТА № 5

по курсу   
Параллельное программирование

Группа 6409

Студент П.А. Маштаков

(*подпись*)

Преподаватель,

к.ф.-м.н. Е.С. Козлова

(*подпись*)

Самара 2023

ЗАДАНИЕ

**Произвести запуск программы с использованием библиотеки CUBLAS** для **умножения квадратных матриц размерностей N×N, сгенерированных случайно. В ходе анализа работы программы оценить время ее выполнения на различном количестве исполняющих нитей.**

**Реализовать последовательный вариант программы. Оценить время ее выполнения. Рассчитать ускорение параллельных программ относительно последовательного варианта.**

Таблица 1 – Исходные данные на ЛР № 5

|  |  |
| --- | --- |
| Тип | float |
| N | 256, 1024, 4096 |
| Параметры транспонирования | без Т, без Т |

ВВЕДЕНИЕ

Программная модель CUDA, разработанная компанией Nvidia в 2006 году, является комплексом аппаратного и программного обеспечения, который упрощает создание программ для графических адаптеров. Nvidia гарантирует, что все её графические карты, независимо от серии, будут поддерживать архитектуру, совместимую с CUDA. Эта технология включает в себя необходимые инструменты для разработки, такие как расширения языка C, компилятор, API для взаимодействия с графическими адаптерами и библиотеки. В API предусмотрены как синхронные, так и асинхронные функции. Синхронные вызовы блокируют выполнение, в то время как асинхронные позволяют выполнять другие задачи параллельно [1]. Асинхронные процессы, включая копирование, запуск ядра и инициализацию памяти, требуют синхронизации перед использованием результатов.

В данной лабораторной работе было произведено перемножение двух квадратных матриц различных размерностей с использованием библиотеки CUBLAS.

.

ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ

2.1 Результаты работы программ без параметра Q

В ходе исследования времени работы программ здесь и далее проводилось усреднение не менее чем по 12 запускам.

На рисунке 1 представлен скрин запуска и работы программы.

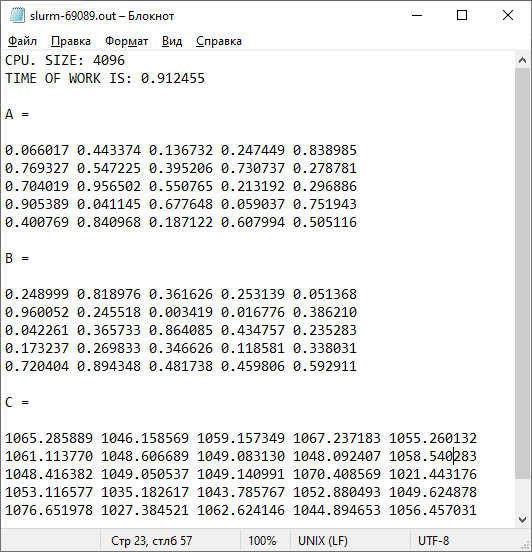


Рисунок 1 – Пример работы программы для N = 4096.

Последовательная программа представляла собой перемножение двух случайно сгенерированных матриц, размерностью **N×N**.

В таблицах 2-3 представлено время выполнения параллельных программ и их ускорение по сравнению с последовательным вариантом.

Таблица 2 – Время работы программ

|  |  |  |  |
| --- | --- | --- | --- |
| N | Время последовательной программы, с | Время параллельной программы, с | |
| Генерация на GPU | Генерация CPU |
| 256 | 0,1283 | 0,0342 | 0,0133 |
| 1024 | 11,1033 | 0,0411 | 0,0609 |
| 4096 | 2198,7900 | 0,2942 | 0,9124 |

Таблица 4 – Ускорение параллельных программ

|  |  |  |
| --- | --- | --- |
| N | Ускорение параллельной программы | |
| Генерация на GPU | Генерация CPU |
| 256 | 3,7514 | 9,6466 |
| 1024 | 270,1532 | 182,3201 |
| 4096 | 7473,7933 | 2409,8969 |

На рисунке 2 приведен график зависимости времени работы программ от размерности матрицы при различных вариантах генерации данных. На рисунке 3 приведен график зависимости ускорения программ от размерности матрицы при различных вариантах генерации данных.

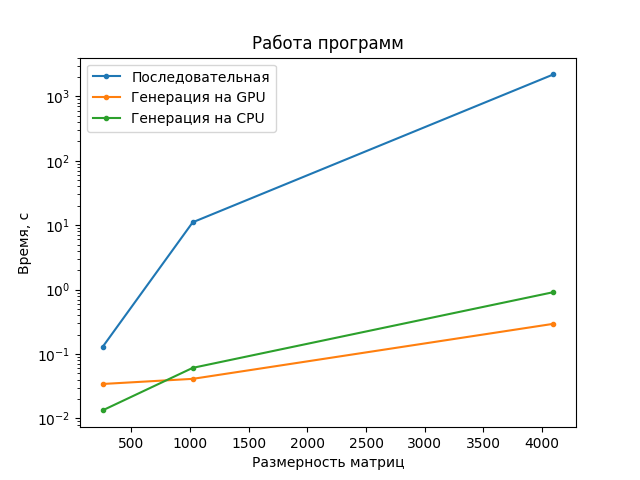


Рисунок 2 – Время работы программ

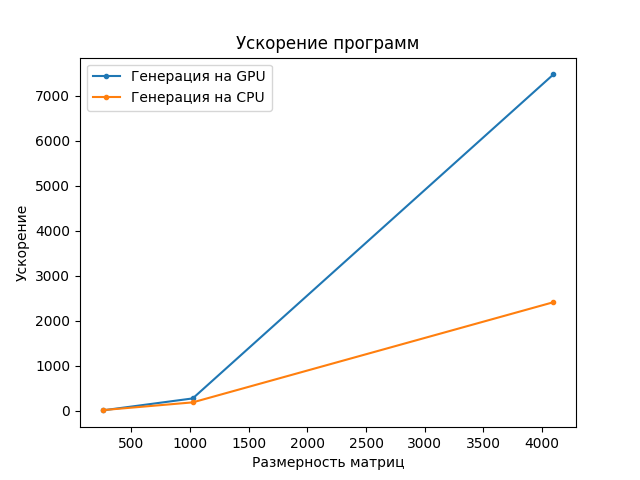


Рисунок 3 – Ускорение программ

**ВЫВОДЫ:**

Из полученных результатов видно, что:

1. Вариант генерации матриц на GPU при помощи специальной функции оказывается намного эффективнее для матриц большого размера, в то время как для матриц небольшого размера генерация случайных чисел занимает большой процент времени от общего времени работы программы. Поэтому для небольших матриц эффективнее оказывается вариант генерации матриц на CPU, об этом говорит ускорение обоих вариантов (3,7514 – генерация на GPU и 9,6466 – генерация на CPU) при размерности матриц 256**×256.**
2. Максимальное ускорение – 7473,7933 было получено параллельной программой с генерацией матриц на GPU при максимальной размерности матриц **4096×4096. Это можно объяснить тем, что при увеличении размерностей матриц алгоритм с генерацией матриц на GPU оказывается эффективнее.**
3. С увеличением размерности задачи некратно увеличивается время выполнения и ускорение параллельных программ, поскольку время, затраченное на копирование данных и генерацию матриц отличается с каждым запуском программ.

ЗАКЛЮЧЕНИЕ

Цель лабораторной работы – написать параллельную программу **с использованием библиотеки CUBLAS** для **умножения квадратных матриц** и сравнить время выполнения с длительностью последовательной программы достигнута. Показано, что использование параллельных технологий для данного типа программ обосновано, в виду того, что оба варианта параллельной программы показывают ускорение, к тому же с увеличением размерностей матриц оно существенно растет.

В ходе выполнения лабораторной работы я изучил(а) основы использования библиотеки CUBLAS, приобрел(а) навыки по написанию параллельных программ с ее использованием. Наиболее сложной частью выполнения лабораторной работы была реализация последовательной программы. Интерес вызвали возможности библиотеки CUBLAS.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сапелкин, А. С. CUDA и удалённый GPU / А. С. Сапелкин // Хабр : [сайт]. – Москва, 2018. – URL: https://habr.com/ru/articles/416127/ (дата обращения: 20.11.2023).
2. Козлова, Е.С. Лабораторные работы по курсу «Параллельное программирование»: Методические указания [Текст] / Сост. Е.С. Козлова, А.С. Широканев − Самара, 2019. – 61 с.
3. оеводин, В. В. Параллельные вычисления [Текст] / В. В. Воеводин, Вл. В. Воеводин. — СПб.: БХВ-Петербург, 2002. — 608 с.
4. Богачёв К.Ю. Основы параллельного программирования: учебное пособие, 2-е изд. [Текст] / К. Ю. Богачёв ‑ М. : БИНОМ. Лаборатория знаний, 2013. ‑ 344 с.
5. Гергель, В. П. Теория и практика параллельных вычислений, 2-е изд. [Текст] / B. II. Гергель. — М.: Интуит. 2016. - 500 с.
6. Боресков А.В. Параллельные вычисления на GPU. Архитектура и программная модель CUDА Учеб. пособие [Текст] / А.В. Боресков ‑ М.: Издательство Московского университета, 2012. - 336 с.
7. Библиографическое описание документа. Общие требования и правила составления [Электронный ресурс] / сост.: В.С. Крылова, С.М. Григорьевская, Е.Ю. Кичигина // Официальный интернет-сайт научной библиотеки Томского государственного университета. – Электрон. дан. – Томск, [2010]. – <http://www.lib.tsu.ru/win/produkzija/metodichka/metodich.html> (дата обращения: 10.09.2019).

ПРИЛОЖЕНИЕ А

Код программы с технологией CUDA

#include <cstdlib>

#include <curand.h>

#include <cstdlib>

#include <curand.h>

#include <cublas\_v2.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 1024 // 256, 1024, 4096

// C(m,n) = A(m,k) \* B(k,n)

void gpu\_blas\_mmul(const float\* A, const float\* B, float\* C, const int m\_length) {

int lda = m\_length, ldb = m\_length, ldc = m\_length;

const float alf = 1;

const float bet = 0;

const float\* alpha = &alf;

const float\* beta = &bet;

cublasHandle\_t handle;

cublasCreate(&handle);

cublasSgemm(handle, CUBLAS\_OP\_N, CUBLAS\_OP\_N, m\_length, m\_length, m\_length, alpha, A, lda, B, ldb, beta, C, ldc);

cublasDestroy(handle);

}

void print\_matrix(float\* matrix, int m\_length) {

printf("\n");

for (int i = 0; i < m\_length; ++i) {

for (int j = 0; j < m\_length; ++j) {

printf("%f ", matrix[j \* m\_length + i]);

}

printf("\n");

}

printf("\n");

}

int main() {

int m\_length = SIZE, n2b = m\_length \* m\_length \* sizeof(float);

float\* h\_A = (float\*)malloc(n2b);

float\* h\_B = (float\*)malloc(n2b);

float\* h\_C = (float\*)malloc(n2b);

float\* d\_A, \* d\_B, \* d\_C;

cudaMalloc(&d\_A, n2b);

cudaMalloc(&d\_B, n2b);

cudaMalloc(&d\_C, n2b);

cudaEvent\_t start, stop;

cudaEventCreate(&start);

cudaEventCreate(&stop);

float gpuTime = 0.0f;

srand(time(0));

cudaEventRecord(start, 0);

for (int i = 0; i < 12; ++i) {

for (int i = 0; i < m\_length \* m\_length; i++) {

h\_A[i] = (float)rand() / RAND\_MAX;

h\_B[i] = (float)rand() / RAND\_MAX;

}

cudaMemcpy(d\_A, h\_A, n2b, cudaMemcpyHostToDevice);

cudaMemcpy(d\_B, h\_B, n2b, cudaMemcpyHostToDevice);

gpu\_blas\_mmul(d\_A, d\_B, d\_C, m\_length);

cudaMemcpy(h\_C, d\_C, n2b, cudaMemcpyDeviceToHost);

}

cudaEventRecord(stop, 0);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("CPU. SIZE: %d\n", SIZE);

printf("TIME OF WORK IS: %.6f\n\n", gpuTime / 12000);

printf("A =\n");

print\_matrix(h\_A, 5);

printf("B =\n");

print\_matrix(h\_B, 5);

printf("C =\n");

print\_matrix(h\_C, 5);

cudaFree(d\_A);

cudaFree(d\_B);

cudaFree(d\_C);

free(h\_A);

free(h\_B);

free(h\_C);

return 0;

}  
/\*

#include <cstdlib>

#include <curand.h>

#include <cublas\_v2.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 256 // 256, 1024, 4096

void GPU\_fill\_rand(float\* A, int m\_length) {

curandGenerator\_t prng;

curandCreateGenerator(&prng, CURAND\_RNG\_PSEUDO\_DEFAULT);

curandSetPseudoRandomGeneratorSeed(prng, (unsigned long long) clock());

curandGenerateUniform(prng, A, m\_length \* m\_length);

}

// C(m,n) = A(m,k) \* B(k,n)

void gpu\_blas\_mmul(const float\* A, const float\* B, float\* C, const int m\_length) {

int lda = m\_length, ldb = m\_length, ldc = m\_length;

const float alf = 1;

const float bet = 0;

const float\* alpha = &alf;

const float\* beta = &bet;

// Create a handle for CUBLAS

cublasHandle\_t handle;

cublasCreate(&handle);

// Do the actual multiplication

cublasSgemm(handle, CUBLAS\_OP\_N, CUBLAS\_OP\_N, m\_length, m\_length, m\_length, alpha, A, lda, B, ldb, beta, C, ldc);

// Destroy the handle

cublasDestroy(handle);

}

void print\_matrix(float\* matrix, int m\_length) {

printf("\n");

for (int i = 0; i < m\_length; ++i) {

for (int j = 0; j < m\_length; ++j) {

printf("%f ", matrix[j \* m\_length + i]);

}

printf("\n");

}

printf("\n");

}

int main() {

int m\_length = SIZE, n2b = m\_length \* m\_length \* sizeof(float);

float\* h\_A = (float\*)malloc(n2b);

float\* h\_B = (float\*)malloc(n2b);

float\* h\_C = (float\*)malloc(n2b);

float\* d\_A, \* d\_B, \* d\_C;

cudaMalloc(&d\_A, n2b);

cudaMalloc(&d\_B, n2b);

cudaMalloc(&d\_C, n2b);

cudaEvent\_t start, stop;

cudaEventCreate(&start);

cudaEventCreate(&stop);

float gpuTime = 0.0f;

srand(time(0));

cudaEventRecord(start, 0);

for (int i = 0; i < 12; ++i) {

GPU\_fill\_rand(d\_A, m\_length);

GPU\_fill\_rand(d\_B, m\_length);

cudaMemcpy(h\_A, d\_A, n2b, cudaMemcpyDeviceToHost);

cudaMemcpy(h\_B, d\_B, n2b, cudaMemcpyDeviceToHost);

gpu\_blas\_mmul(d\_A, d\_B, d\_C, m\_length);

cudaMemcpy(h\_C, d\_C, n2b, cudaMemcpyDeviceToHost);

}

cudaEventRecord(stop, 0);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("GPU. SIZE: %d\n", SIZE);

printf("TIME OF WORK IS: %.6f\n\n", gpuTime /12000);

printf("A =");

print\_matrix(h\_A, 5);

printf("B =");

print\_matrix(h\_B, 5);

printf("C =");

print\_matrix(h\_C, 5);

cudaFree(d\_A);

cudaFree(d\_B);

cudaFree(d\_C);

free(h\_A);

free(h\_B);

free(h\_C);

return 0;

}\*/

ПРИЛОЖЕНИЕ Б

Код последовательной программы

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define IDX2C(i,j,ld) (((j)\*(ld))+(i))

#define SIZE 4096 //256, 1024, 4096

void cpu\_mmul(const float\* A, const float\* B, float\* C, int m\_lenght) {

for (int i = 0; i < m\_lenght; ++i) {

for (int j = 0; j < m\_lenght; ++j) {

C[IDX2C(i, j, m\_lenght)] = 0.0;

for (int r = 0; r < m\_lenght; ++r) {

C[IDX2C(i, j, m\_lenght)] += A[IDX2C(i, r, m\_lenght)] \* B[IDX2C(r, j, m\_lenght)];

}

}

}

}

void print\_matrix(float\* matrix, int m\_lenght) {

printf("\n");

for (int i = 0; i < m\_lenght; ++i) {

for (int j = 0; j < m\_lenght; ++j) {

printf("%f ", matrix[j \* m\_lenght + i]);

}

printf("\n");

}

printf("\n");

}

int main() {

int m\_lenght = SIZE, n2b = m\_lenght \* m\_lenght \* sizeof(float);

float\* h\_A = (float\*)malloc(n2b);

float\* h\_B = (float\*)malloc(n2b);

float\* h\_C = (float\*)malloc(n2b);

double start\_time, end\_time, timeofwork;

srand(time(0));

start\_time = clock();

for (int i = 0; i < 12; ++i) {

for (int j = 0; j < m\_lenght \* m\_lenght; j++) {

h\_A[j] = (float)rand() / RAND\_MAX;

h\_B[j] = (float)rand() / RAND\_MAX;

}

cpu\_mmul(h\_A, h\_B, h\_C, m\_lenght);

}

end\_time = clock();

timeofwork = end\_time - start\_time;

printf("Sequence. SIZE: %d\n", SIZE);

printf("TIME OF WORK IS: %.9f\n\n", timeofwork / CLOCKS\_PER\_SEC / 12);

printf("A =\n");

print\_matrix(h\_A, 5);

printf("B =\n");

print\_matrix(h\_B, 5);

printf("C =\n");

print\_matrix(h\_C, 5);

free(h\_A);

free(h\_B);

free(h\_C);

return 0;

}

ПРИЛОЖЕНИЕ В

Скрипт запуска CUDA программы

#!/bin/bash

#SBATCH --job-name=Mashtakoff

#SBATCH --nodes=1

#SBATCH --gres=gpu

#SBATCH --time=11:03:00

module load cuda

nvcc -g -G -O0 -lcublas -lcurand -o gpu.bin gpu.cu

./gpu.bin

ПРИЛОЖЕНИЕ Г

Скрипт запуска последовательной программы

#!/bin/bash

#SBATCH --job-name=Mashtakoff

#SBATCH --time=11:30:00

#SBATCH --nodes=1

#SBATCH --mem=2gb

./seq