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 6408

Студент Н.В. Носов

(*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 | Complex |
| N | 1250, 2500, 5000 |
| Параметры транспонирования | без Т, с Т |

ВВЕДЕНИЕ

В современном мире есть большое количество задач, для решения которых необходимо использовать огромные вычислительные мощности. Также постоянно возрастают требования к точности решения и скорости выполнения вычислений. Один из основных способов решения этих проблем – создание параллельных вычислительных систем.

Не теряют актуальности основные операции линейной алгебры, которые широко используются, например, в графике. При обработке изображений одной из основных операций является умножение матриц. В данной лабораторной работе необходимо познакомиться с одной из таких библиотек – CUBLAS, на примере реализации умножения двух квадратных матриц.

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.

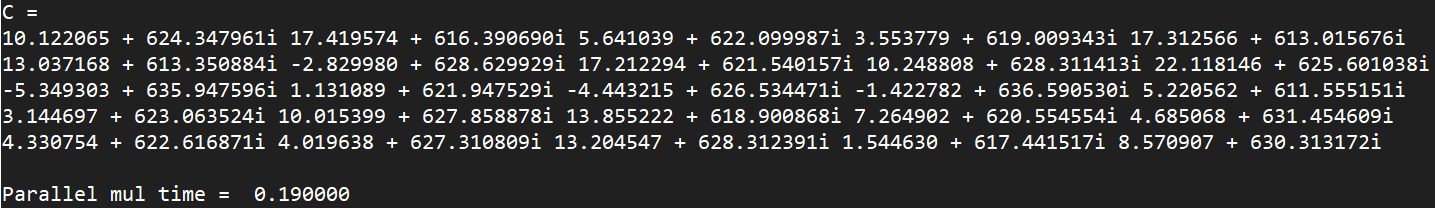


Рисунок 1 – Пример работы программы для N = 1250 на GPU.

Последовательная программа представляла собой алгоритм для отыскания произведения двух матриц с предварительными генерацией и транспонированием.

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 |
| 1250 | 71,853627 | 0,191333 | 0,397395 |
| 2500 | 573,088226 | 0,552850 | 1,391057 |
| 5000 | 5221,293629 | 3,193855 | 6,942593 |

Таблица 4 – Ускорение параллельных программ

|  |  |  |
| --- | --- | --- |
| N | Ускорение параллельной программы | |
| Генерация на GPU | Генерация CPU |
| 1250 | 371,3462 | 180,8116 |
| 2500 | 1036,6070 | 411,9804 |
| 5000 | 1634,7941 | 752,0668 |

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. Генерация на GPU быстрее для всех N, так как данные генерируются с помощью специальной функции, которая работает быстрее, чем наивная реализация. Это можно объяснить тем, что в моем варианте N является достаточно большим, из-за чего рациональнее использовать GPU.
2. Максимальное ускорение равно 1634,7941. Данное ускорение получено при максимальной размерности матриц с генерацией на GPU, так как генерация на GPU при больших размерностях быстрее, чем генерация на CPU и отношение количества операций распараллеливания к вычислительным операциям минимально при наибольшей размерности. Минимальное ускорение равно 180,8116. Данное ускорение получено при минимальной размерности с генерацией на CPU.
3. С увеличением размерности увеличиваются время и ускорение. Увеличение происходит некратно. Ускорение на GPU растет быстрее, чем на CPU. Это связано с отсутствием пересылки сгенерированных матриц на GPU.

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 с помощью CUBLAS было получено ускорение от 371,3462 до 1634,7941. Данное ускорение является очень значительным.

В ходе выполнения лабораторной работы я изучил основы использования библиотеки CUBLAS, приобрел навыки по написанию параллельных программ с ее использованием. Наиболее сложной частью выполнения лабораторной работы было освоение новой библиотеки CUBLAS. Интерес вызвали полученные результаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Козлова, Е.С. Лабораторные работы по курсу «Параллельное программирование»: Методические указания [Текст] / Сост. Е.С. Козлова, А.С. Широканев − Самара, 2019. – 61 с.
2. оеводин, В. В. Параллельные вычисления [Текст] / В. В. Воеводин, Вл. В. Воеводин. — СПб.: БХВ-Петербург, 2002. — 608 с.
3. Богачёв К.Ю. Основы параллельного программирования: учебное пособие, 2-е изд. [Текст] / К. Ю. Богачёв ‑ М. : БИНОМ. Лаборатория знаний, 2013. ‑ 344 с.
4. Гергель, В. П. Теория и практика параллельных вычислений, 2-е изд. [Текст] / B. II. Гергель. — М.: Интуит. 2016. - 500 с.
5. 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.
6. 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 <stdlib.h>

#include <stdio.h>

#include <curand.h>

#include <iostream>

#include <ctime>

#include "cublas\_v2.h"

using namespace std;

// Fill the array A(nr\_rows\_A, nr\_cols\_A) with random numbers on GPU

void GPU\_fill\_rand(cuDoubleComplex\* A, int nr\_rows\_A, int nr\_cols\_A) {

    // Create a pseudo-random number generator

    curandGenerator\_t prng;

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

    // Set the seed for the random number generator using the system clock

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

    // Fill the array with random numbers on the device

    curandGenerateUniformDouble(prng, (double\*)A, nr\_rows\_A \* nr\_cols\_A \* 2);

}

/// Multiply the arrays A and B on GPU and save the result in C

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

void gpu\_blas\_mmul(const cuDoubleComplex\* A, const cuDoubleComplex\* B, cuDoubleComplex\* C, const int m, const int k, const int n) {

    int lda = m, ldb = k, ldc = m;

    const cuDoubleComplex alf = make\_cuDoubleComplex(1.0, 0.0);

    const cuDoubleComplex bet = make\_cuDoubleComplex(0.0, 0.0);

    const cuDoubleComplex\* alpha = &alf;

    const cuDoubleComplex\* beta = &bet;

    // CUBLAS is starting here

    cublasHandle\_t handle;

    cublasStatus\_t status = cublasCreate(&handle);

    // Do the actual multiplication

    cublasZgemm(handle,

                CUBLAS\_OP\_N,

                CUBLAS\_OP\_T,

                m,

                n,

                k,

                alpha,

                A,

                lda,

                B,

                ldb,

                beta,

                C,

                ldc);

    // Destroy the handle

    cublasDestroy(handle);

}

void print\_matrix(cuDoubleComplex\* matrix, int rows, int cols) {

    for (int i = 0; i < rows; ++i) {

        for (int j = 0; j < cols; ++j) {

            printf("%f + %fi ", matrix[i + j \* rows].x, matrix[i + j \* rows].y);

        }

        printf("\n");

    }

    printf("\n");

}

void fill\_matrix(cuDoubleComplex\* matrix, int rows, int cols) {

    for (int i = 0; i < rows; ++i) {

        for (int j = 0; j < cols; ++j) {

            matrix[i + j \* rows] = make\_cuDoubleComplex((rand() + 0.0) / RAND\_MAX, (rand() + 0.0) / RAND\_MAX);

        }

    }

}

int main() {

    int nr\_rows\_A, nr\_cols\_A, nr\_rows\_B, nr\_cols\_B, nr\_rows\_C, nr\_cols\_C;

    nr\_rows\_A = nr\_cols\_A = nr\_rows\_B = nr\_cols\_B = nr\_rows\_C = nr\_cols\_C = 1250; // 1250, 2500, 5000

    cuDoubleComplex\* h\_A = (cuDoubleComplex\*)malloc(nr\_rows\_A \* nr\_cols\_A \* sizeof(cuDoubleComplex));

    cuDoubleComplex\* h\_B = (cuDoubleComplex\*)malloc(nr\_rows\_B \* nr\_cols\_B \* sizeof(cuDoubleComplex));

    cuDoubleComplex\* h\_C = (cuDoubleComplex\*)malloc(nr\_rows\_C \* nr\_cols\_C \* sizeof(cuDoubleComplex));

    // create pointers into the GPU

    cuDoubleComplex\* d\_A;

    cuDoubleComplex\* d\_B;

    cuDoubleComplex\* d\_C;

    // allocate memory in the GPU

    cudaMalloc(&d\_A, nr\_rows\_A \* nr\_cols\_A \* sizeof(cuDoubleComplex));

    cudaMalloc(&d\_B, nr\_rows\_B \* nr\_cols\_B \* sizeof(cuDoubleComplex));

    cudaMalloc(&d\_C, nr\_rows\_C \* nr\_cols\_C \* sizeof(cuDoubleComplex));

    double start\_ = clock();

    // GPU\_fill\_rand(d\_A, nr\_rows\_A, nr\_cols\_A);

    // GPU\_fill\_rand(d\_B, nr\_rows\_B, nr\_cols\_B);

    fill\_matrix(h\_A, nr\_rows\_A, nr\_cols\_A);

    fill\_matrix(h\_B, nr\_rows\_B, nr\_cols\_B);

    // copy the vectors into the GPU

    cudaMemcpy(d\_A, h\_A, nr\_rows\_A \* nr\_cols\_A \* sizeof(cuDoubleComplex), cudaMemcpyHostToDevice);

    cudaMemcpy(d\_B, h\_B, nr\_rows\_B \* nr\_cols\_B \* sizeof(cuDoubleComplex), cudaMemcpyHostToDevice);

    gpu\_blas\_mmul(d\_A, d\_B, d\_C, nr\_rows\_A, nr\_cols\_A, nr\_cols\_B);

    // cudaMemcpy(h\_A, d\_A, nr\_rows\_A \* nr\_cols\_A \* sizeof(cuDoubleComplex), cudaMemcpyDeviceToHost);

    // cudaMemcpy(h\_B, d\_B, nr\_rows\_B \* nr\_cols\_B \* sizeof(cuDoubleComplex), cudaMemcpyDeviceToHost);

    cudaMemcpy(h\_C, d\_C, nr\_rows\_C \* nr\_cols\_C \* sizeof(cuDoubleComplex), cudaMemcpyDeviceToHost);

    double end\_ = clock();

    printf("C = \n");

    print\_matrix(h\_C, 5, 5);

    printf("Parallel mul time =  %.6f ", (end\_ - start\_) / CLOCKS\_PER\_SEC);

    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>

#include <complex>

#define NMAX 1250 // 1250, 2500, 5000

void printMatrix(std::complex<double>\* matrix, int rows, int cols) {

    for (int i = 0; i < rows; ++i) {

        for (int j = 0; j < cols; ++j) {

            printf("%f + %fi ", matrix[j + i \* rows].real(), matrix[j + i \* rows].imag());

        }

        printf("\n");

    }

    printf("\n");

}

void transpose(std::complex<double>\* matrix, int nrCols) {

    std::complex<double> t;

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

        for (int j = i; j < nrCols; ++j) {

            t = matrix[j + i \* nrCols];

            matrix[j + i \* nrCols] = matrix[i + j \* nrCols];

            matrix[i + j \* nrCols] = t;

        }

    }

}

void multiply(std::complex<double>\* A, std::complex<double>\* B, std::complex<double>\* C, int N)

{

    int i, j, k;

    for (i = 0; i < N; i++) {

        for (j = 0; j < N; j++) {

            C[i + j \* N] = 0;

            for (k = 0; k < N; k++)

                C[i + j \* N] += A[i + k \* N] \* B[k + j \* N];

        }

    }

}

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

    int s = NMAX \* NMAX;

    std::complex<double>\* a = new std::complex<double>[s], \* c = new std::complex<double>[s], \* b = new std::complex<double>[s];

    double st\_time, end\_time = 0;

    srand(clock());

    st\_time = clock();

    for (int j = 0; j < s; ++j) {

        a[j] = std::complex<double>((rand() + 0.0) / RAND\_MAX, (rand() + 0.0) / RAND\_MAX);

        b[j] = std::complex<double>((rand() + 0.0) / RAND\_MAX, (rand() + 0.0) / RAND\_MAX);

    }

    transpose(b, NMAX);

    multiply(a, b, c, NMAX);

    end\_time = clock() - st\_time;

    printf("\nTIME OF WORK IS: %f ", end\_time / CLOCKS\_PER\_SEC);

    printf("\nC = \n");

    printMatrix(c, 5, 5);

    delete[] a, b, c;

    return 0;

}

ПРИЛОЖЕНИЕ В

Скрипт запуска параллельной программы

#!/bin/bash

#SBATCH --job-name=myjob

#SBATCH --nodes=1

#SBATCH --gres=gpu

#SBATCH --time=00:05:00

module load cuda/8.0

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

./cuda.bin

ПРИЛОЖЕНИЕ Г

Скрипт запуска последовательной программы

#!/bin/bash

#SBATCH --job-name=myjob

#SBATCH --time=03:00:00

#SBATCH --nodes=1

#SBATCH --mem=1gb

./simple