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 В.С. Лаптев

(*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 | double |
| N | 300, 1500, 7500 |
| Параметры транспонирования | с Т, без 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.

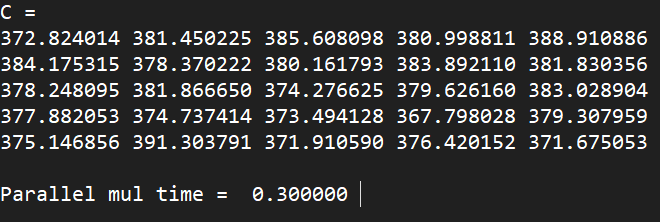


Рисунок 1 – Пример работы программы для N = 1500.

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 |
| 300 | 0,231667 | 0,146662 | 0,118333 |
| 1500 | 26,984523 | 0,178548 | 0,294367 |
| 7500 | 4169,586671 | 3,246567 | 8,096431 |

Таблица 4 – Ускорение параллельных программ

|  |  |  |
| --- | --- | --- |
| N | Ускорение параллельной программы | |
| Генерация на GPU | Генерация CPU |
| 300 | 1,580 | 1,958 |
| 1500 | 151,133 | 91,670 |
| 7500 | 1284,306 | 514,991 |

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. В общем случае генерация на GPU быстрее, так как данные генерируются с помощью специальной функции, которая работает быстрее, чем наивная реализация. Кроме всего, такая генерация не требует отправки матриц с хоста на девайс. Но стоит отметить, что при N=300 генерация на CPU оказывается быстрее. Это можно объяснить тем, что для малых N не выгодно создавать генератор, так как на это тоже уходит некоторое время. В данном случае наивная генерация и отсылка данных оказываются быстрее, чем генерация на GPU.
2. Максимальное ускорение равно 1284. 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 1,58. Данное ускорение получено при минимальной размерности с генерацией на GPU. На данной размерности GPU показывает себя хуже, чем CPU. При малой размерности операции распараллеливания занимают более значимое время.
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 1,5 до 1284. 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 <cublas\_v2.h>

#include <iostream>

#include <ctime>

using namespace std;

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

void GPU\_fill\_rand(double\* 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, A, nr\_rows\_A \* nr\_cols\_A);

}

/// 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 double\* A, const double\* B, double\* C, const int m, const int k, const int n) {

int lda = m, ldb = k, ldc = m;

const double alf = 1;

const double bet = 0;

const double\* alpha = &alf;

const double\* beta = &bet;

// Create a handle for CUBLAS

cublasHandle\_t handle;

cublasCreate(&handle);

// Do the actual multiplication

cublasDgemm(handle, CUBLAS\_OP\_T, CUBLAS\_OP\_N, m, n, k, alpha, A, lda, B, ldb, beta, C, ldc);

// Destroy the handle

cublasDestroy(handle);

}

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

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

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

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

}

printf("\n");

}

printf("\n");

}

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

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

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

matrix[i + j \* rows] = (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 = 7500;

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

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

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

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

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

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

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

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);

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

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

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

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

double end\_ = clock();

printf("C = \n");

print\_matrix(h\_C, 5, 5);

printf("Parallel mul time = %f ", (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>

#define NMAX 7500

void printMatrix(double\* matrix, int rows, int cols) {

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

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

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

}

printf("\n");

}

printf("\n");

}

void transpose(double\* matrix, int nrCols) {

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(double\* A, double\* B, 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;

double\* a = new double[s], \* c = new double[s], \* b = new double[s];

double st\_time, end\_time = 0;

srand(clock());

st\_time = clock();

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

a[j] = (rand() + 0.0) / RAND\_MAX;

b[j] = (rand() + 0.0) / RAND\_MAX;

}

transpose(a, 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;

}