Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №4 по дисциплине "Анализ алгоритмов"

Тема Многопоточность на CUDA

Студент Костев Д.И.

Группа ИУ7-51Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Многопоточность — способность центрального процессора (CPU) или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой.

Этот подход отличается от многопроцессорности, так как многопоточность процессов и потоков совместно использует ресурсы одного или нескольких ядер: вычислительных блоков, кэш-памяти ЦПУ или буфера перевода с преобразованием (TLB).

В тех случаях, когда многопроцессорные системы включают в себя несколько полных блоков обработки, многопоточность направлена на максимизацию использования ресурсов одного ядра, используя параллелизм на уровне потоков, а также на уровне инструкций.

Поскольку эти два метода являются взаимодополняющими, их иногда объединяют в системах с несколькими многопоточными ЦП и в ЦП с несколькими многопоточными ядрами.

Многопоточная парадигма стала более популярной с конца 1990-х годов, поскольку усилия по дальнейшему использованию параллелизма на уровне инструкций застопорились.

Смысл многопоточности — квазимногозадачность на уровне одного исполняемого процесса.

Значит, все потоки процесса помимо общего адресного пространства имеют и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

Достоинства:

- облегчение программы посредством использования общего адресного пространства;
- меньшие затраты на создание потока в сравнении с процессами;
- повышение производительности процесса за счёт распараллеливания процессорных вычислений;
- если поток часто теряет кэш, другие потоки могут продолжать использовать неиспользованные вычислительные ресурсы.

Недостатки:

- несколько потоков могут вмешиваться друг в друга при совместном использовании аппаратных ресурсов;
- с программной точки зрения аппаратная поддержка многопоточности более трудоемка для программного обеспечения;

- проблема планирования потоков;
- специфика использования. Вручную настроенные программы на ассемблере, использующие расширения MMX или AltiVec и выполняющие предварительные выборки данных, не страдают от потерь кэша или неиспользуемых вычислительных ресурсов. Таким образом, такие программы не выигрывают от аппаратной многопоточности и действительно могут видеть ухудшенную производительность из-за конкуренции за общие ресурсы.

Однако несмотря на количество недостатков, перечисленных выше, многопоточная парадигма имеет большой потенциал на сегодняшний день и при должном написании кода позволяет значительно ускорить однопоточные алгоритмы.

Цель лабораторной работы

Целью данной лабораторной работы является изучение и реализация параллельных вычислений.

Задачи лабораторной работы

В рамках выполнения работы необходимо решить следующие задачи:

- изучить понятие параллельных вычислений;
- реализовать последовательный и параллельный реализации алгоритма перемножения матриц(на GPU);
- сравнить временные характеристики реализованных алгоритмов экспериментально.

1 Аналитическая часть

1.1 Описание задачи

Пусть даны две прямоугольные матрицы

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

тогда матрица C

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1,l}; j = \overline{1,n}) \tag{1.3}$$

будет называться произведением матриц A и B.

В данной лабораторной работе стоит задача распараллеливания алгоритма Винограда по 2 схемам. Так как каждый элемент матрицы C вычисляется независимо от других и матрицы A и B не изменяются, то для параллельного вычисления произведения, достаточно просто равным образом распределить элементы матрицы C между потоками.

1.2 Вывод

Обычный алгоритм перемножения матриц независимо вычисляет элементы матрицы-результата, что дает большое количество возможностей для реализации параллельного варианта алгоритма.

2 Конструкторская часть

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема обычного алгоритма перемножения матриц (без распараллеливания). На рисунке 2.2 представлена схема распараллеливания алгоритма умножения матриц.

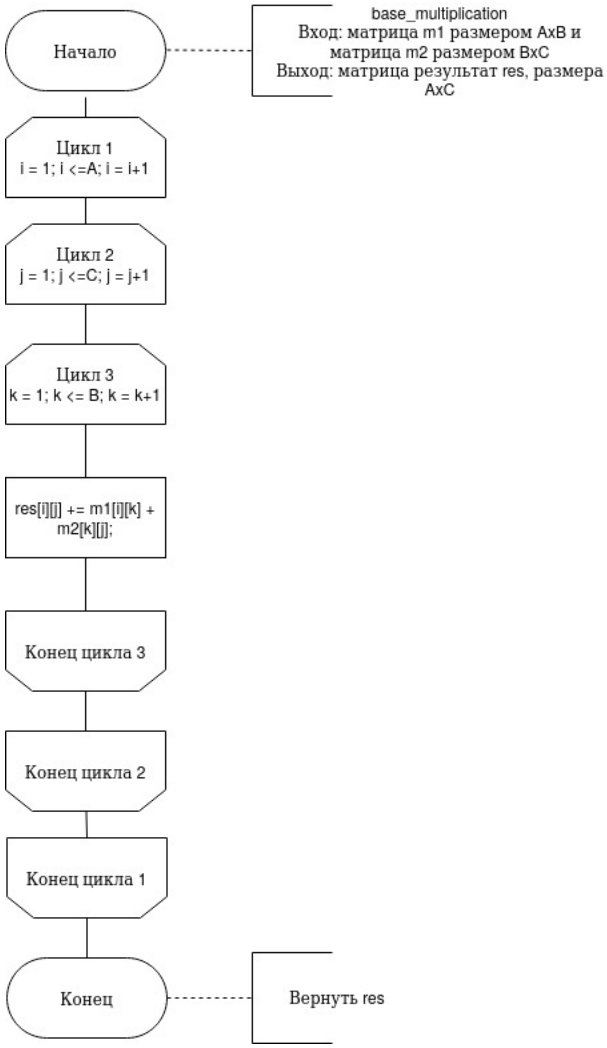

Рис. 2.1: Схема стандартного алгоритма умножения матриц.

На рисунке 2.3 представленна схема с параллельным выполнением цикла (по строкам матрицы).

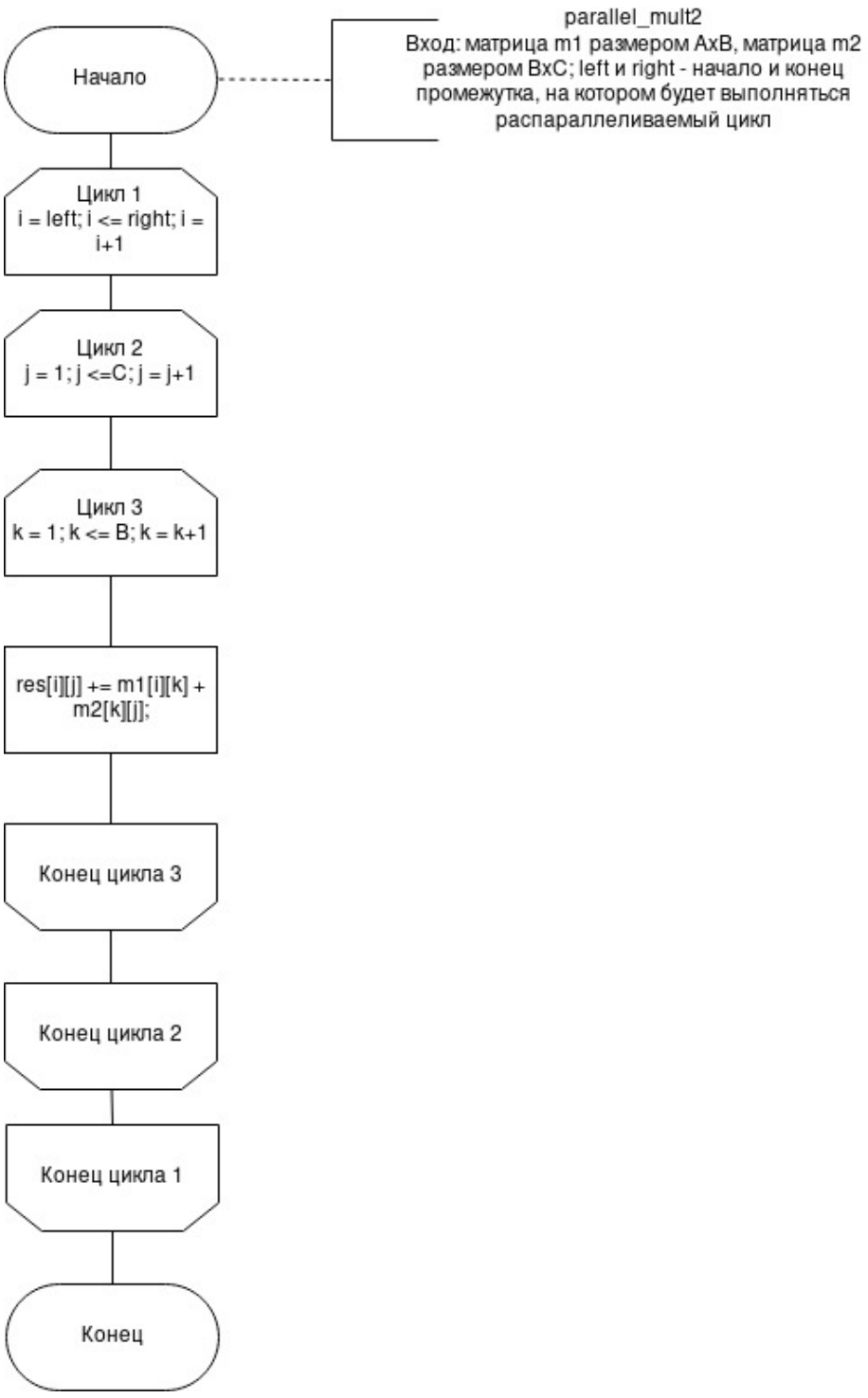

Рис. 2.2: Схема распараллеленного алгоритма умножения матриц, способ №1.

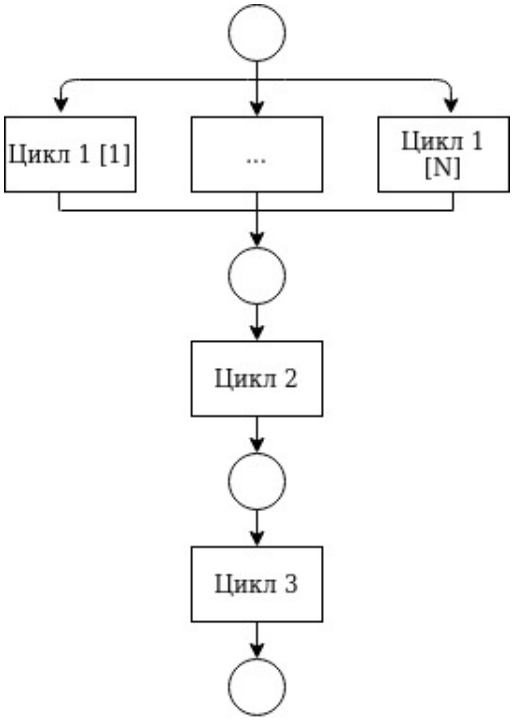

Рис. 2.3: Схема с параллельным выполнением первого цикла

2.2 Вывод

На основе теоретических данных, полученных из аналитического раздела, была построена схема стандартного алгоритма умножения матриц, а так же после разделения алгоритма на этапы были предложены 2 схемы параллельного выполнения данных этапов.

3 | Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

- на вход подаются размеры 2 матриц, а также их элементы;
- на выходе — матрица, которая является результатом умножения входных матриц.

3.2 Средства реализации

Для реализации ПО я выбрал язык программирования Cuda. Данный выбор обусловлен высокой скоростью работы языка, а так же наличия инструментов для создания и эффективной работы с потоками.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.2 приведена реализация расмотренных ранее алгоритмов перемножения матриц.

Листинг 3.1: Функция умножения матриц обычным способом

```
__global__ void matmulsimp(int* M, int* N, int* P, int width) {
  for (int row = 0; row < width; row++) {
    for (int col = 0; col < width; col++) {
      // Multiply the row of A by the column of B to get the row, column of
         product.
      for (int inner = 0; inner < width; inner++)
        P[row * width + col] += M[row * width + inner] * N[inner * width +
           col];
    }
  }
}
```

Листинг 3.2: Функция умножения матриц параллельно

```
__global__ void MatMul(int* M, int* N, int* P, int width)
{
  int x = threadIdx.x;
  int y = threadIdx.y;

  float Pervalue = 0;

  float elem1 = 0.0, elem2 = 0.0, value = 0.0;
  for (int i = 0; i < width; i++)
  {
    elem1 = M[y * width + i]; //
                          M
    elem2 = N[i * width + x]; //                                        N


    value += elem1 * elem2; //
  }

  P[y * width + x] = value;
}
```

3.4 Тестовые данные

В таблице 3.1 приведены тесты для функций, реализующих параллельное и обычное умножение матриц. Все тесты пройдены успешно.

Таблица 3.1: Тестирование функций

Первая матрица	Вторая матрица	Ожидаемый результат
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 6 & 12 & 18 \\ 6 & 12 & 18 \\ 6 & 12 & 18 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 2 \\ 1 & 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 5 & 10 \\ 5 & 10 \end{pmatrix}$
(2)	(2)	(4)
$\begin{pmatrix} 1 & -2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 6 \\ 4 & 12 & 18 \\ 4 & 12 & 18 \end{pmatrix}$

3.5 Вывод

В данном разделе были разработаны исходные коды алгоритмов: обычный способ умножения матриц и два способа параллельного перемножения матриц.

4 | Исследовательская часть

4.1 Технические характеристики

Ниже приведеные технические характеристики устройства, на котором было проведенно тестирование ПО.

- Операционная система: Windows 11 64-bit.
- Оперативная память: 8 ГБ.
- Процессор: Intel(R) Core(TM) i5-8250 CPU @ 1.60ГГц.

4.2 Время выполнения алгоритмов

В таблице 4.1 представлены замеры времени работы для каждого из алгоритмов на разных размерах матриц.

Таблица 4.1: Таблица времени выполнения алгоритмов(в милисекундах)

Размер матрицы	Cuda	Стандарт
3	1.157120	0.141376
10	0.527360	1.644640
20	0.097280	13.453536
30	0.116768	39.8552320
40	1.094656	93.969246
50	0.226304	188.345596
100	0.013312	1408.762817
200	0.021056	9884.333984

4.3 Вывод

При малых размерах матриц различия времени - минимальны, из-за того что часть времени, в которой выигрывает параллельная реализация, тратится на распределение потоков. Но при больших размерах становится видно, что скорость параллельной реализации на Cuda куда выше. Так при 200 элементах скорость параллельногой реализации в 470000 раз выше.

Заключение

В рамках данной лабораторной работы была достигнута её цель: изучены паралелльные вычисления. Также выполнены следующие задачи:

- было изучено понятие параллельных вычислений;
- были реализованы обычный и параллельный реализаций алгоритма перемножения матриц;
- было произведено сравнение временных характеристик реализованных алгоритмов экспериментально.

Параллельные реализации алгоритмов выигрывают по скорости у обычной (однопоточной) реализации перемножения двух матриц. Наиболее эффективны данные алгоритмы при количестве потоков, совпадающем с количеством логических ядер компьютера. Потому что задействован каждый поток.

Литература

[1] Кормен Т. Алгоритмы: построение и анализ [Текст] / Корм ен Т. - Вильямс, 2014. - 198 с. - 219 с.

[2] GetProcessTimes function. URL: https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-getprocesstimes, 01.10.2021

[3] Алгоритм Копперсмита — Винограда. URL: https://ru.wikipedia.org/wiki/Алгоритм-Копперсмита—Винограда