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, математики и электроники

Факультет информатики  
Кафедра суперкомпьютеров и общей информатики

**Отчет по лабораторной работе №1**

Дисциплина: «Облачные и высокопроизводительные вычисления»

Тема: **«Использование GPU и CPU при перемножении матриц»**

Выполнил: Печенин В.А.

Группа: 6133-010402D

Самара 2020

**ЗАДАНИЕ**

Задача: реализовать алгоритм перемножения матриц

Язык: C++ или Python

Входные данные: 2 матрицы размером от 100х100 до 2000х2000.

Выходные данные: проверка корректности перемножения + время вычисления

Реализация должна содержать 2 функции перемножения матриц: на CPU и на GPU с применением CUDA.

Отчет о проделанной лабораторной работе - это git-репозиторий с исходным кодом реализации + описание проделанной работы там же в readme.

Необходимо описать реализацию, объяснив, что конкретно было распараллелено и почему.

Провести эксперименты: перемножить матрицы разных размеров, посчитать ускорение. Результаты привести в виде таблицы.

**ХОД РАБОТЫ**

Компьютер, на котором выполнялась лабораторная работа, имеет: CPU: Intel(R) Core(TM) i5-3470 @3.2 GHz, 4 ядра; GPU: NVIDIA GeForce GTX 1050 Ti .

Реализация задачи выполнена на языке Python, с использованием библиотек **Numpy** и **CuPy**. **CuPy** - это библиотека, которая реализует массивы **Numpy** на графических процессорах Nvidia, используя библиотеку графических процессоров CUDA (на компьютере CUDA 10.1). За счет использования множества ядер CUDA может быть достигнуто превосходное распараллеливание вычисление и снижение времени расчетов. С учётом того, что в состав каждого потокового мультипроцессора (SM) входит по 128 потоковых ядер CUDA, их общее количество в GTX 1050 Ti составляет 768 (6 потоковых процессоров).

Посмотрим на примере довольно ёмкой задачи умножения матрицы А на матрицу B. Результат будет записан в матрицу M, который по размеру идентичен перемножаемым матрицам. Чтобы получить первое значение в первой строке матрицы М, нужно умножить первую строку матрицы А на первый столбец B, чтобы получить второе значение в первой строке M — первую строку A на второй столбец B, и так далее. Получается, каждый элемент матрицы M можно считать независимо от других, то есть параллельно. Размер каждой такой задачи один и тот же — все строки A и столбцы B имеют одинаковый размер.

Кроме библиотек **Numpy** и **CuPy** в программе были использованы библиотеки **time** (для подсчета времени вычисления), **matplotlib** (для визуализации расчетов), **openpyxl** и **datetime** (для записи результатов)

Был создан класс *ParallelDot*, входным параметром которого является вектор значений размеров матриц *size*, имя файла для сохранения таблицы результатов. С помощью метода класса **computations\_of\_experiment** в цикле размера вектора *size* создаются две матрицы *size × size*, заполненные единицами и рассчитывается результат матричного произведения (сначала с использованием библиотеки **Numpy**, потом **CuPy**. В цикле так же выполняется проверка точности результатов (сравнение с единичной матрицей и поиск наибольшего отклонения) и расчет ускорения расчетов на GPU относительно CPU (в % к времени расчета). Метод **save\_computations** используется для автоматизации записи расчетов вычислений в файл Excel, метод **plot\_results** для построения графика трудоемкости расчетов. В таблице 1 приведены результаты вычислений.

Таблица 1 – Результаты вычислений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер матрицы | Время CPU, с | Время GPU, с | Погрешность CPU | Погрешность GPU | Ускорение GPU относительно CPU, % |
| 100 | 0,01 | 0,24 | 0 | 0 | -2300 |
| 200 | 0 | 0 | 0 | 0 | 0 |
| 300 | 0 | 0 | 0 | 0 | 0 |
| 400 | 0,01 | 0 | 0 | 0 | 100 |
| 500 | 0 | 0 | 0 | 0 | 0 |
| 600 | 0 | 0 | 0 | 0 | 0 |
| 700 | 0 | 0 | 0 | 0 | 0 |
| 800 | 0,015001 | 0,0005 | 0 | 0 | 96,66704 |
| 900 | 0,011001 | 0,001 | 0 | 0 | 90,91086 |
| 1000 | 0,014002 | 0,0005 | 0 | 0 | 96,42936 |
| 1100 | 0,032504 | 0,001 | 0 | 0 | 96,92366 |
| 1200 | 0,022503 | 0,001 | 0 | 0 | 95,55539 |
| 1300 | 0,033004 | 0,001 | 0 | 0 | 96,97029 |
| 1400 | 0,041505 | 0,001001 | 0 | 0 | 97,58912 |
| 1500 | 0,043506 | 0,0015 | 0 | 0 | 96,55188 |
| 1600 | 0,058507 | 0,001 | 0 | 0 | 98,29053 |
| 1700 | 0,060508 | 0,0015 | 0 | 0 | 97,52076 |
| 1800 | 0,070509 | 0,0015 | 0 | 0 | 97,87209 |
| 1900 | 0,082011 | 0,0015 | 0 | 0 | 98,17081 |
| 2000 | 0,099012 | 0,0015 | 0 | 0 | 98,48467 |

**ЗАКЛЮЧЕНИЕ**

В результате лабораторный работы выполнены и обобщены в виде таблицы результаты перемножения матриц разных размеров с использованием CPU и GPU на языке Python. Можно отметить, что с некоторого размера матриц (*size*>200) GPU решает задачу примерно на 100% быстрее, чем CPU, что связано с большим количеством потоков, которые одновременно запускаются. При проверке корректности перемножения ошибок выявлено.