МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №*4***

**по курсу «Программирование графических процессоров»**

***Работа с матрицами. Метод Гаусса.***

Выполнил: *А.Е. Аксенов*

Группа: *8О-408Б*

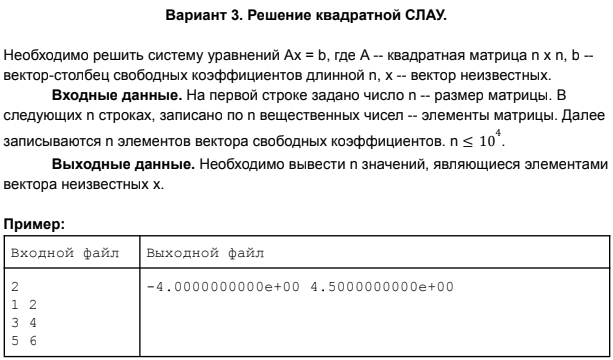
Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2023

**Условие**

Цель работы. Использование объединения запросов к глобальной памяти. Реализация метода Гаусса с выбором главного элемента по столбцу. Ознакомление с библиотекой алгоритмов для параллельных расчётов Thrust.

В качестве вещественного типа данных необходимо использовать тип данных double. Библиотеку Thrust использовать только для поиска максимального элемента на каждой итерации алгоритма. В вариантах (1, 5, 6,7), где необходимо сравнение по модулю с нулем, в качестве нулевого значения использовать . Все результаты выводить с относительной точностью .

**Программное и аппаратное обеспечение**

|  |  |
| --- | --- |
| Графический процессор | |
| GPU | NVIDIA RTX 2070 |
| Compute Capability | 7.5 |
| Объем видеопамяти | 8192 МБ |
| Тип видеопамяти | GDDR6 |
| Разделяемая память | 48 КБ |
| Константная память | 64 КБ |
| Число регистров в блоке | 65636 |
| Максимальное число блоков | 16 |
| Максимальное число нитей | 1024 |
| Количество мультипроцессоров | 34 |
| Центральный процессор | |
| СPU | Intel Core i5 9600K |
| Число физических ядер | 6 |
| Максимальное число потоков | 6 |
| Базовая частота | 3.6 GHz |
| Частота Turbo Boost | 4.5 GHz |
| Объем кэша L3 | 9 МБ |
| Оперативная память | |
| Тип оперативной памяти: | DDR4 |
| Частота памяти | 2666MHz |
| Объём | 32 ГБ |
| Постоянная память | |
| Тип постоянной памяти | SSD |
| Объём постоянной памяти | 1024ГБ |
| Программное обеспечение | |
| Операционная система | Microsoft Windows 11 |
| Версия операционной системы | 2004 |
| Среда разработки | Microsoft Visual Studio 2022 |
| Версия среды разработки | 16.7.3 |
| NVIDIA CUDA Toolkit | 12.2 |

**Метод решения**

1. Считываем размер входной матрицы.
2. Выделяем память под элементы.
3. Считываем элементы матрицы.
4. Транспонируем матрицу для более удобной обработки.
5. Выделяем память на графическом ускорителе.
6. Выполняем копирование транспонированной матрицы в оперативную память графического ускорителя.
7. С помощью библиотеки Thrust создаём указатель специального типа на скопированную в графическую память матрицу.
8. Запускаем основной цикл, выполняющий вычисления.
9. На каждой итерации выполняем поиск максимального по абсолютной величине элемента в *i*-строке матрицы, начиная со столбца *i*.
10. Запускаем ядро, выполняющее обмен строки, содержащей максимальный элемент с *i*-ой строкой.
11. Если максимальный по модулю элемент равен нулю, матрица является сингулярной и её определитель равен нулю.
12. При каждом обмене строк выполняем обмен столбца *b*.
13. Для каждого столбца, начиная с *i* + 1-го в *i*-ой строке матрицы заполняем коэффициенты для последующих вычислений.
14. Вычисленные коэффициенты применяем для вычитания из столбцов, начиная с *i* + 1-го столбца *i*.
15. После завершения основного этапа вычислений копируем матрицу .
16. Выполняем прямой проход метода прогонки.
17. Выполняем обратный проход метода прогонки.
18. Выводим ответ с заданной точностью.
19. Освобождаем выделенные ресурсы.

**Описание программы**

Весь цикл разработки и отладки выполнялся в среде Microsoft Visual Studio с установленными CUDA CDK. Программа написана на языке C++, используются библиотеки STL и Thrust. Для удобства поиска максимального по абсолютной величине элемента в столбце матрица в графическом ускорителе представлена в транспонированном виде. Эта схема хранения позволяет выделить линейный участок памяти, хранящий все элементы конкретного столбца исходной матрицы. Благодаря этому становится возможным применить функцию max\_element из библиотеки thrust. Для выполнения основных операций алгоритма Гаусса написано три ядра. Ядро swp\_kern выполняет обмен столбцов (строк исходной матрицы). Ядро div\_kern вычисляет коэффициенты, требуемые для вычитания. Основное ядро sub\_kern использует вычисленные коэффициенты и выполняет вычитание требуемых столбцов – строк исходной матрицы.

\_\_global\_\_ void sub\_kern(double\* m, int32\_t k, int32\_t n) {         // ядро вычитания строк под опорной строкой

    const int32\_t sy = gridDim.y \* blockDim.y;

    const int32\_t sx = gridDim.x \* blockDim.x;

    for (int i = blockIdx.y \* blockDim.y +

    threadIdx.y + k + 1;  i < n; i += sy) {

        for(int j = blockIdx.x \* blockDim.x +

        threadIdx.x + k + 1; j < n; j += sx) {

            m[i \* n + j] -= m[k \* n + j] \* m[i \* n + k];

        }

    }

}

\_\_global\_\_ void swp\_kern(double\* m, int32\_t i1, int32\_t i2, int32\_t n) {  // ядро перестановки (обмена) двух строк

    const int32\_t s = gridDim.x \* blockDim.x;

    for (int32\_t i = blockIdx.x \* blockDim.x

                + threadIdx.x; i < n; i += s) {

        double tmp = m[i \* n + i1];

        m[i \* n + i1] = m[i \* n + i2];

        m[i \* n + i2] = tmp;

    }

}

\_\_global\_\_ void div\_kern(double\* m, int32\_t pi, int32\_t n) {             // ядро записывающее коэффициенты матрицы L (LUP разложение)

    int32\_t s = gridDim.x \* blockDim.x;

    const double div = m[pi \* n + pi];

    for (int32\_t i = blockIdx.x \* blockDim.x

    + threadIdx.x + pi + 1; i < n; i += s) {

        m[pi \* n + i] /= div;

    }

}

**Результаты**

|  |  |  |
| --- | --- | --- |
| Размер матрицы | CPU (1 ядро) | GPU (16 16, 32 1) |
| 4000 | 25238.5 | 1559.2 |
| 6000 | 84632.3 | 5211.5 |
| 8000 | 199730.6 | 12136.9 |
| 10000 | 387873.3 | 24936.9 |

|  |  |  |
| --- | --- | --- |
| Размер матрицы | Параметры ядра | Время |
| 6000 | <(8, 32)(32, 1)> | 5062.42 |
| 6000 | <(16, 32)(32, 1)> | 4681.09 |
| 6000 | <(32, 32)(32, 1)> | 4033.34 |
| 6000 | <(8, 32)(32, 8)> | 4649.62 |
| 6000 | <(16, 32)(32, 8)> | 4000.09 |
| 6000 | <(32, 32)(32, 8)> | 3685.89 |
| 6000 | <(8, 32)(32, 16)> | 4167.35 |
| 6000 | <(16, 32)(32, 16)> | 3788.01 |
| 6000 | <(32, 32)(32, 16)> | 3568.35 |
| 6000 | <(8, 32)(32, 32)> | 4045.08 |
| 6000 | <(16, 32)(32, 32)> | 3833.16 |
| 6000 | <(32, 32)(32, 32)> | 3606.63 |
| 6000 | <(128, 128)(32, 32)> | 3588.42 |

Результат работы профайлера nvprof для матрицы 1000x1000. Приведены основные два ядра, участвующие в процессе вычислений. Видно, что запросов к глобальной памяти меньше, чем элементов матрицы (алгоритм производит обращения к каждому элементу матрицы). Данное явление являются результатом объединения запросов к глобальной памяти (англ. memory coalescing).

==32077== Profiling application: ./a.out

==32077== Profiling result:

==32077== Event result:

Invocations Event Name Min Device "GeForce RTX2070 (0)"

Kernel: sub\_kern (double\*, int, int)

999 divergent\_branch 0 30874 1038

999 global\_store\_transaction 3 135268 40643

999 l1\_shared bank conflict 0 0 0

999 l1\_local load hit 0 0 0

Kernel: swp\_kern(double\*, int, int, int)

990 divergent\_branch 1 1 1

990 global store transaction 2403 2403 2403

990 11\_shared bank conflict 0 0 0

990 11\_local load hit 0 0 0

**Выводы**

Алгоритм Гаусса лежит в основе LU и LUP разложений. Применяется при решении систем линейных алгебраических уравнений. С его помощью можно определить ранг и определитель матрицы. При многократном решении некоторой СЛАУ, в которой изменяется только лишь вектор свободных членов *b*, с вычислительной точки зрения, выгодно применить LUP разложение. Такая обработка позволяет производить последующие вычисления с алгоритмической сложностью *O*(*n*), где *n* – количество уравнений. Вычислительная сложность самого алгоритма разложения составляет *O*(*n3*). Из результатов тестирование видно – распараллеливание вычислений на графическом ускорителе позволило намного ускорить процесс.