МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №4
по курсу «Программирование графических процессоров»

Работа с матрицами. Метод Гаусса.

Выполнил: А. О. Тояков
Группа: М8О-407Б-18
Преподаватели: К. Г. Крашенинников,
А. Ю. Морозов

Москва, 2022

УСЛОВИЕ

Цель работы: Использование объединения запросов к глобальной памяти. Реализация метода Гаусса с выбором главного элемента по столбцу. Ознакомление с библиотекой алгоритмов для параллельных расчетов 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 10^{-7}. Все результаты выводить с относительной точностью 10^{-10}.

Вариант 7. Решение матричного уравнения.

Необходимо найти *любое* решение матричного уравнения AX = B, где A -- матрица n x m, X -- неизвестная матрица m x k, B -- матрица n x k.

Входные данные. На первой строке заданы числа n, m и k -- размеры матриц. В следующих n строках, записано по m вещественных чисел -- элементы матрицы A. Далее записываются n строк, по k чисел -- элементы матрицы B.
$n * m + m * k + n * k \leq 1.2 * 10^8$.

Выходные данные. Необходимо вывести на m строках, по k чисел -- элементы неизвестной матрицы X.

Пример:

Входной файл	Выходной файл
1 2 1 1 2 5	5.0000000000e+00 0.0000000000e+00
2 2 2 1 2 3 4 1 0 0 1	-2.0000000000e+00 1.0000000000e+00 1.5000000000e+00 -5.0000000000e-01
2 3 2 1 2 3 4 8 6 3 4 7 1	5.0000000000e-01 -3.5000000000e+00 0.0000000000e+00 0.0000000000e+00 8.3333333333e-01 2.5000000000e+00

ПРОГРАММНОЕ И АППАРАТНОЕ ОБЕСПЕЧЕНИЕ

Device: GeForce MX250

Размер глобальной памяти: 3150381056

Размер константной памяти : 65536

Размер разделяемой памяти: 49152

Регистров на блок: 32768

Максимум потоков на блок: 1024

Количество мультипроцессоров : 3

OS: Linux Ubuntu 18.04

Редактор: VSCode

Компилятор: nvcc версии 11.4 (g++ версии 7.5.0)

МЕТОД РЕШЕНИЯ

Для реализации метода Гаусса решения матричного уравнения необходимо найти максимальный элемент в столбце (это реализовано с помощью функции библиотеки thrust и написанного компаратора). Затем нужно строку с этим элементом переместить наверх.

После чего мы делаем параллельный проход методом Гаусса вниз, обнуляя соответствующие элементы под главным, и запоминаем индекс ступеньки. После приведения матрицы к треугольному виду необходимо совершить проход вверх, чтобы матрица A стала единичной, а в матрице B столбцы станут решением. Затем копируем данные с GPU на CPU и выводим результат таким образом, что на месте ступенек в искомой матрице X появятся столбцы матрицы B, а строки, индексы которых не соответствуют индексам ступенек занулятся.

ОПИСАНИЕ ПРОГРАММЫ

Макрос **CSC** отвечает за отслеживание ошибок в функциях cuda, поэтому все cuda-вызовы оборачиваются в него и при cudaError_t != cudaSuccess выводится сообщение об ошибке.

struct comparator – перегрузка оператора () для вычисления максимального элемента в столбце с помощью библиотеки thrust.

fill_AB – отвечает за заполнение матрицы AB элементами по столбцам.

swap – функция на GPU, в которой строки меняются местами.

down_pass – функция на GPU, которая совершает один проход методом Гаусса вниз.

up_pass – функция на GPU, которая совершает один проход методом Гаусса вверх.

int main() – отвечает за ввод, и перенос данных на GPU и вывод.

ТЕСТЫ ПРОИЗВОДИТЕЛЬНОСТИ

В обоих программах числа для массивов генерировались рандомно в промежутке [-1000, 1000].

Работа на GPU:

Тест: (размеры матрицы)	Результат:			
	down_pass<<<dim3(8,16), dim3(8,16)>>> up_pass<<<dim3(8,16), dim3(8,16)>>>	down_pass<<<dim3(8,32), dim3(8,32)>>> up_pass<<<dim3(8,32), dim3(8,32)>>>	down_pass<<<dim3(16,32), dim3(16,32)>>> up_pass<<<dim3(16,32), dim3(16,32)>>>	down_pass<<<dim3(32,32), dim3(32,32)>>> up_pass<<<dim3(32,32), dim3(32,32)>>>
10 * 10 * 10	time = 0.542720	time = 0.648192	time = 0.916480	time = 2.220032
100 * 100 * 100	time = 5.253792	time = 5.637184	time = 9.588672	time = 22.995457
1000 * 1000 * 1000	time = 851.362183	time = 859.228455	time = 950.938354	time = 1378.070435

Работа на CPU:

Тест:	Результат:
10 * 10 * 10	time = 9.234577
100 * 100 * 100	time = 215.25452
1000 * 1000 * 1000	time = 12235.547828

ВЫВОДЫ

После выполнения лабораторной работы № 4 я узнал для себя несколько новых аспектов параллельного программирования.

Во-первых, метод Гаусса имеет сложность o(n^3), и как известно, чем сложнее метод, тем проще его распараллелить, поэтому я считаю это преимуществом этого алгоритма.

Во-вторых, я познакомился с библиотекой thrust, которая позволяет экономить время и не писать фундаментальные алгоритмы (например поиск максимального элемента или scan).

В-третьих, я познакомился с одним из способов оптимизации при работе с глобальной памятью, а именно с объединением запросов к глобальной памяти. Это значит, что GPU умеет объединять ряд запросов к глобальной памяти в транзакцию одного сегмента, причем длина сегмента должна быть 32/64/128 байт и сегмент должен быть выровнен по своему размеру. Таким образом мы получим увеличение скорости работы с памятью на порядок.