МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №2
по курсу «Параллельная обработка данных»

Работа с матрицами. Метод Гаусса.

Выполнил: А.С. Федоров
Группа: 8О-407Б
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. Использование двухмерной сетки потоков. Исследование производительности программы с помощью утилиты nvprof.
Вариант задания. 6. Нахождение ранга матрицы.

Программное и аппаратное обеспечение

Графический процессор
Название: NVIDIA GeForce GTX 1050
Compute capability: 6.1
Объем графической памяти: 2147352576 байтов
Объем разделяемой памяти на блок: 49152 байтов
Объем регистров на блок: 65536
Размер варпа: 32
Максимальное количество потоков на блок: (1024, 1024, 64)
Максимальное число блоков: (2147483647, 65535, 65535)
Объем постоянной памяти: 65536 байтов
Число мультипроцессоров: 5

Процессор
Название: i5-8250U
Базовая тактовая частота процессора: 1,60 ГГц
Количество ядер: 4
Количество потоков: 8
Кеш L1: 256 Кб
Кеш L2: 1 Мб
Кеш L3: 6 Мб

Оперативная память
Тип: DDR4
Объем: 11.9 Гб
Частота: 2400 МГц

Програмное обеспечение
ОС: WSL2 (Windows 11)
IDE: Microsoft Visual Studio 2022 (аддон NVIDIA Nsight)
Компилятор: nvcc

Метод решения

Для определения ранга матрицы, выполняется ее приведение к ступенчатому виду, согласно методу Гаусса. После приведения, ранг матрицы можно определить по количеству ступенек получившейся матрицы. Процедура выполняется в два этапа: последовательное обнуление столбцов матрицы и подсчет ступенек.

Описание программы

Приведение матрицы к ступенчатому виду реализовано итеративно в цикле. На каждой итерации зануляется один столбец. Сам процесс зануления выполнен на GPU. Для оптимизации вычислений столбец не зануляется явно, достаточно вычислить новые значения элементов оставшейся подматрицы. Для оптимизации вычислений, в ячейки массива текущего столбца можно использовать для хранения коэффициентов и не вычислять их для каждого элемента отдельно. Таким образом можно сократить число применений операции деления к количеству строк в текущей подматрице. Для избежания потери точности до обнуления столбца выполняется поиск доминирующего элемента с помощью функции max_element библиотеки thrust. Для ее применения необходимо хранить столбцы матрицы непрерывно. Также, для объединения запросов в глобальную память, в ядре столбцы матицы расположены по координате x, а строки по координате y. Запросы объединяются так как расположение столбцов в памяти непрерывно.

Результат профилировки программы на тестовой матрице размером 256 на 256.
Количество запросов в глобальную память при обращении к элементам по строкам:

Invocations	Metric Name	Metric Description	Min	Max	Avg
Device "NVIDIA GeForce GTX 1050 (0)"					
Kernel: swap_kernel(double*, int, int, int, int, int)					
7	gld_transactions	Global Load Transactions	146	514	397
Kernel: kernel(double*, int, int, int, int)					
256	gld_transactions	Global Load Transactions	2	786434	408578

Количество запросов в глобальную память при обращении к элементам по столбцам:

Invocations	Metric Name	Metric Description	Min	Max	Avg
Device "NVIDIA GeForce GTX 1050 (0)"					
Kernel: swap_kernel(double*, int, int, int, int, int)					
247	gld_transactions	Global Load Transactions	18	514	274
Kernel: kernel(double*, int, int, int, int)					
256	gld_transactions	Global Load Transactions	2	195842	67247

Суда по метрикам, проход ядра по столбцам сокращает количество запросов в глобальную память в 4 раза, что должно положительно сказаться на производительности.
Время работы при обращении к элементам по строкам: 118.681 мс;
Время работы при обращении к элементам по столбцам: 74.667 мс.

Код ядра:

```
__global__ void swap_kernel(double *sub_matrix, int n, int m, int y, int x, int
max_row_index){
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    while (y+idx < m){
        double tmp = sub_matrix[(y+idx)*n+x];
        sub_matrix[(y+idx)*n+x] = sub_matrix[(y+idx)*n+max_row_index];
        sub_matrix[(y+idx)*n+max_row_index] = tmp;
        idx += gridDim.x * blockDim.x;
    }
}

__global__ void preparation_kernel(double* sub_matrix, int n, int m, int x, int y){
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    while (x+1+idx < n){
        sub_matrix[y*n+(x+1+idx)] /= sub_matrix[y*n+x];
        idx += gridDim.x * blockDim.x;
    }
}

__global__ void kernel(double* sub_matrix, int n, int m, int x, int y) {
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    int idy = blockDim.y * blockIdx.y + threadIdx.y;
    while (y+1+idy < m){
        idx = blockDim.x * blockIdx.x + threadIdx.x;
        while (x+1+idx < n){
            sub_matrix[(y+1+idy)*n+(x+1+idx)] -=
sub_matrix[(y+1+idy)*n+x]*sub_matrix[y*n+(x+1+idx)];
            idx += blockDim.x * gridDim.x;
        }
        idy += blockDim.y * gridDim.y;
    }
}
```

Результаты (в миллисекундах)

	Размер теста (по вертикали и горизонтали)				
	2^4	2^6	2^8	2^{10}	2^{12}
GPU(CUDA) < dim3(1, 1), dim3(1, 1)>	3.920	41.646	1232.401	76593.179	-
GPU(CUDA) < dim3(1, 1), dim3(32, 32)>	3.421	18.010	**70.175**	**877.434**	13908.099
GPU(CUDA) < dim3(32, 32), dim3(32, 32)>	15.831	23.156	81.813	1178.578	**7543.937**
GPU(CUDA) < dim3(64, 64), dim3(32, 32)>	9.419	23.030	116.529	944.624	7612.188
GPU(CUDA) < dim3(128, 128), dim3(32, 32)>	15.344	55.136	191.983	1036.041	8381.176
GPU(CUDA) < dim3(256, 256), dim3(32, 32)>	37.856	151.128	560.698	2245.834	13657.296
GPU(CUDA) < dim3(512, 512), dim3(32, 32)>	128.130	471.854	1695.805	7158.132	33293.070
CPU	**0.037**	**1.762**	101.111	6444.590	500052.510

Выводы

Алгоритм поиска ранга матрицы может быть широко применен для упрощения и определения совместности СЛАУ. Метод Гаусса, на принципе работы которого основан реализованный алгоритм определения ранга также имеет применение в конечно-разностных или конечно-элементных методах решений дифференциальных уравнений.
При программной реализации возникли трудности с откладкой. Для выявления ошибки была отдельно написана программа для автотестирования на случайно сгенерированных матрицах со случайным числом копий строк и столбцов, умноженных на случайные коэффициенты. Однако из-за того, что автотестер тестировал на матрицах малого размера выявить ошибку, возникающую в случае, когда на один поток приходится больше одного элемента матрицы. Отловить данную ошибку удалось только после анализа реализации прошлого года. В последствии, автотестер был скорректирован, чтобы покрывать все случаи и также показал наличие ошибки в изначальном варианте.
Исходя из результатов тестирования программы, сетка блоков 32 на 32 с 32 на 32 потоков в каждом достаточно даже для обработки самого большого теста, так как общее число потоков равно 2^{20}. Далее производительность падает, так как приходится обрабатывать большое число потоков, которые никак не используются для непосредственного вычисления результата.