МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт «Информационные технологии и прикладная математика»
Кафедра «Вычислительная математика и программирование»

Лабораторная работа №4
по курсу «Параллельная обработка данных»

Работа с матрицам. Метод Гаусса.

Выполнил: А.В. Скворцов
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2020

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.
Вариант 3: Решение квадратной СЛАУ. Необходимо решить систему уравнений Ax=b.

Программное и аппаратное обеспечение

- GPU: Geforce 940MX
  - Compute capability : 5.0
  - Total Global Memory : 2147483648
  - Shared memory per block : 49152
  - Registers per block : 65536
  - Max threads per block : (1024, 1024, 64)
  - Max block : (2147483647, 65535, 65535)
  - Total constant memory : 65536
  - Multiprocessors count : 3
- CPU: Intel Core i5-6200U 2.30GHz
- RAM: 4GB
- Software: Windows 10, Visual Studio Code, nvcc

Метод решения

Глобальная память не кэшируется и поэтому одно обращение к ней может занимать до 600 тактов. Именно оптимизация работы с глобальной памятью может дать наибольший прирост производительности (в десятки раз). Достичь этого можно с помощью объединения нескольких запросов в глобальную память (coalescing global memory accesses) в одну транзакцию. При этом должно выполняться несколько условий, одним из которых является обращение всех нитей полуварпа к памяти в пределах одного блока.
Эти особенности были учтены при выполнении лабораторной работы. Алгоритм Гаусса состоит из нескольких последовательных частей (прямой и обратный ход), при этом каждый этап в последовательной части может быть распараллелен (вычитание i-ой строки из i+1, … строк). Для удобства хранения и применения библиотеки thrust (поиск максимального элемента) было решено хранить матрицу по столбцам. Тогда при вычитании i-ой строки из всех нижележащих один варп будет обрабатывать один столбец, а все нити полуварпа будут одновременно обращаться к одному элементу i-ой строки и к последовательным элементам столбца.

Описание программы

- int main() - ввод/вывод данных, запуск ядра.
- void solve(double *A, double *b, double *x, int n) – подготовка памяти на видеокарте, запуск ядер прямого и обратных ходов.
  - double *A_dev – указатель на исходную матрицу на видеокарте.

    - size_t pitch — фактический размер столбца после выравнивания.
- __global__ void swap(double *A, int n, size_t pitch, int i, int j) — ядро для параллельного обмена строк матрицы (транспонированной).
- __device__ int align_row(int i) — выравнивает сетку нитей на начало блока.
- __global__ void rows_elimination(double *A, int n, size_t pitch, int left_upper_corner) — ядро для одной итерации прямого хода метода Гаусса.
- __global__ void backward_elimination(double *A, int n, size_t pitch, int i) — ядро для одной итерации обратного хода метода гаусса.
- __global__ void normalize(double *A, int n, size_t pitch) — после выполнения прямого и оратного хода вышеописанными ядрами, на главной диагонали могут остаться неединичные элементы, это ядро производит их нормализацию.

Результаты

Посмотрим, на производительность программы с различными конфигурациями количества блоков, при фиксированном количестве потоков (32, 32) в одном блоке:

Размерность матрицы(снизу), размерность блока(справа)	(2, 2)	(4, 4)	(8, 8)	(16, 16)
(1000, 1000)	1662	1367	1373	1513
(2000, 2000)	5716	5297	5236	5256
(4000, 4000)	30760	29136	28808	28831

Сравним производительность программы на cpu и gpu:

Размерность матрицы	(500, 500)	(1000, 1000)	(2000, 2000)	(4000, 4000)
gpu	573	1941	5584	30901
cpu	270	3117	51628	439436

Выводы

Алгоритм Гаусса — один из базовых алгоритмов численных методов, на нем основано множество других алгоритмов, поэтому скорость его выполнения критична и чрезвычайно важна. Несмотря на то, что выполняется он последовательно, каждый его шаг можно распараллелить, причем сделать это, по идее, достаточно легко. Однако, чтобы выжить максимум производительности, приходится учитывать некоторые особенности cuda, которые с непривычки, доставляют некоторые трудности.
Что касается производительности, из приведенных выше таблиц видно, что на больших матрицах cuda с отрывом выигрывает у cpu.