Параллельные и распределенные вычисления. Задание 3 “Вычисления на GPU”.

Старченко Владимир

Основная часть

Поставленная задача

Реализуйте на CUDA C программу, которая

- Генерирует случайным образом матрицы M и N с элементами типа double (только на CPU)
- Выполняет транспонирование матрицы M в TM (на CPU и GPU)
- Выполняет умножение матрицы TM на N (на CPU и GPU)
- Сравнивает результаты, полученные на CPU и GPU, выводит времена отдельных расчётов и ускорение относительно CPU.

Обоснование решения

Я реализовывал пункты из задания, идя по списку. Сначала на CPU, потом на GPU.

Описание реализации

В начале, я реализовал всё, используя двумерные массивы. Это привело к трудностям при выполнении задания на GPU. Поэтому всё было переписано на одномерных массивах в качестве матриц. В такой реализации строки матриц лежат последовательно в одном массиве.

Транспонирование выполняется не inplace. Это было сделано с целью упростить алгоритм. Более того, inplace транспонирование (по крайней мере то, которое мне известно) будет невозможно сделать на gpu. Функция транспонирования принимает размены матрицы, саму матрицу и массив для результата. Далее выполняет копирование элементов из изначальной матрицы в конечную, изменяя лишь индексы для записи.

В транспонировании всего 1 цикл. Каждая итерация цикла одинакова. Его и можно запустить на gpu, отдав каждому потоку по 1 итерации.

Умножение матриц. Используется простой алгоритм сложностью $O(N^3)$. Для каждого элемента мы считаем скалярное произведение соответствующих столбцов.

В транспонировании есть внешний цикл (который пробегает по всем элементам) и внутренний (вычисляющий скалярное произведение). Итерации внешнего цикла однотипные и не зависят друг от друга. Их можно запустить на gpu. При этом из одной первой матрицы элементы для разных потоков читают по строкам, а из второй по столбцам. При чтении по столбцам, читаемые элементы расположены подряд и кэширование работает хорошо. При чтении по строкам, элементы располагаются далеко друг от друга и кэш работает хуже. Чтобы этого избежать, мы один раз транспнируем первую матрицу. Строки станут столбцами и её можно будет обходить так же, как и вторую матрицу.

Результаты

https://everest.distcomp.org/jobs/5a02fe13320000bb3b8bb6a7

Анализ решения

Первое, что хочется отметить это результаты работы на маленьких матрицах. На маленьких матрицах время закрузки данных на и с gpu довольно большое. Если операции

довольно простые и данные не большие, использовать cpu выгоднее по времени. В том числе, я получал прирост производительности меньше 1 на матрицах размера порядка 10.

Второе, что бросается в глаза - это разница прироста, в зависимости от графической карты. Разница между домашней картой и эверестом получалась примерно 20-30 раз в пользу эвереста. При этом есть интересное наблюдение, загрузка данных на карту на локальной машине иногда производилась на порядок быстрее.

Итоги измерений:

Test name	Cpu time	Gpu time	Speedup
транспонирование	0.002250	0.000122	18.439581
умножение	0.925747	0.002105	439.876639

При разных входных параметрах получались разные значения. Если подводить итог, транспонирование на gpu давало прирост в 20-30 раз. Умножение – 150-450 раз.

Контрольные вопросы

Вопрос 1.

Какой вариант умножения матриц на GPU работает быстрее? Почему? Сколько чтений из глобальной памяти выполняет каждый из вариантов?

Второй вариант быстрее, потому что делается меньше обращений к глобальной памяти.

Первый вариант: N * M * K, в соответствии с алгоритмом.

Второй вариант: N * K так как к каждой ячейке суммарно обращаются один раз при копировании в $\_\_shared\_\_$ блок.

Если точнее, можно сказать, что запись тоже будет считаться и ответы были даны с точностью до константы 2

Вопрос 2.

Какие параметры грида являются оптимальными для блочной схемы умножения матриц размера 4096x4096?

Пусть x - максимально возможное количество блоков. Нам нужно, чтобы блоки покрывали матрицу целиком и не выезжали за края. У нас матрица 4096x4096 - степени двойки. Возьмем такой размер грида y * y, квадрат со стороной y - степень 2, чтобы оно было максимально возможное и помещалось в x.

Вопрос 3.

При блочном умножении матриц размера 1000x1000 с использованием блоков 32x32, в какой доле warp-ов (по всем блокам суммарно) будет иметь место расхождение потоков из-за проверки условий выхода за пределы матриц?

В матрице 1000x1000, которую мы покрываем блоками блоками 32x32, каждый $ceil(1000/32) = 32$ блок будет расходится, так как каждый 32 блок попадает на границу.