Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

Лабораторная работа №3 по курсу «Программирование графических процессоров» на тему «Работа с потоками CUDA»

Новосибирск

2020

Выполнение лабораторной работы поможет получить навыки требующиеся для выполнения третьего задания контрольной работы.

## Задание

1. Прочитайте главы теоретического материала под названиями "Pinned memory" и "Потоки (streams) в CUDA". Ответьте на контрольные вопросы в конце глав (ответы на контрольные вопросы не нужно включать в отчёт по лабораторной работе).
2. Примените потоки для алгоритмов реализованные в лабораторной работе №1.
3. Определите оптимальное количество потоков для матрицы размером 2500x2500 элементов и вектора размером 2500 элементов.

## Методические указания

Для выполнения лабораторной работы требуется модифицировать код, выполняемый на хосте таким образом, чтобы данные передавались на устройство частями асинхронно, после этого выполнялось функция-ядро над переданной частью, после завершения вычислений часть ответа асинхронно должна копироваться на хост.

На хосте создайте и проинициализируйте матрицу и вектор, которые будут умножаться. Затем выделите память на хосте под результирующий вектор и проинициализируйте его нулями. Выделите память под матрицу и вектора на устройстве, создайте нужное количество потоков (начните с двух потоков).

Теперь нужно определиться какие части данных передавать в каждый поток. Разделение нужно произвести таким образом, чтобы результатом работы функции-ядра была часть конечного результата, не требующая дальнейшей обработки. В таком случае имеет смысл в каждом потоке передавать на устройство часть строк матрицы, а вектор передать сразу полностью, потому что для вычислений, каждой нитью используется одна строка из матрицы и вектор. Результатом работы нити будет один элемент результирующего вектора.

Для того чтобы определить размер порции данных нужно количество строк матрицы поделить на количество потоков, и результат умножить на длину строки. После этого полностью скопируйте значения вектора на устройство. И для каждого потока асинхронно скопируйте часть данных на устройство, используя функцию cudaMemcpyAsync, запустите вычисления над порцией данных, асинхронно скопируйте результат на хост. В конце синхронизируйте все потоки вызовом функции cudaDeviceSynchronize() – эта функция будет ожидать завершения всех запущенных потоков.

Проведите исследование зависимости времени работы алгоритма от количества потоков. Начните с двух потоков и увеличивайте их количество до тех пор, пока время, затраченное на вычисления, не перестанет уменьшаться. Замеры времени следует проводить, включая асинхронные пересылки данных.

Псевдокод алгоритма выглядит следующим образом:

//Создание объектов потоков
//NUM\_STREAM - количество потоков
for(i = 0; i < NUM\_STREAM; ++i) {
 CreateStream(stream[i]);
}

//N - количество строк в матрице
//M - размер строки в матрице
//Строки матрицы делятся на части по количеству созданных потоков
//Размер каждой порции равен количеству строк в порции умноженное на размер строки
SIZE\_CHUNK = (N / NUM\_STREAM) \* M;
//Вектор копируется на устройство полностью, его разбивать на части не имеет смысла
Memcpy(devPtrVector, hostPtrVector, M, HostToDevice);
//Для каждого потока асинхронно копируется несколько строк матрицы на устройство
//Затем выполняется функция-ядро
//после этого результат асинхронно копируется на хост
//SIZE\_CHUNK - размер части данных с которым работает функция-ядро
//devPtr - адрес памяти, выделенной на устройстве под матрицу, куда копируются данные с хоста
//hostPtr - адрес памяти на хосте под матрицу откуда копируются данные
//---в этой точке нужно измерить время
for(i = 0; i < NUM\_STREAM; ++i) {
 MemcpyAsync(devPtr + i \* SIZE\_CHUNK, hostPtr + i \* SIZE\_CHUNK, SIZE\_CHUNK, HostToDevice, stream[i]);
 //Количество блоков теперь зависит не от общего размера данных
 //а от размера порции данных
 //devPtrResultVector - адрес результирующего вектора на устройстве
 MulMatrixVector<<<SIZE\_CHUNK / THREADS\_PER\_BLOCK + 1, THREADS\_PER\_BLOCK, stream[i]>>>(devPtr + i \* SIZE\_CHUNK, devPtrVector + i \* SIZE\_CHUNK, devPtrResultVector + i \* SIZE\_CHUNK);
 MemcpyAsync(hostPtrResultVector + i \* SIZE\_CHUNK, devPtrResultVector + i \* SIZE\_CHUNK, SIZE\_CHUNK, DeviceToHost, stream[i]);
}
DeviceSynchronize();
//---в этой точке нужно повторно измерить время
//разность между временем второй и первой точки измерения времени будет временем работы вычислений