МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа №3
по курсу «Параллельная обработка данных»
Технология MPI и технология CUDA. MPI-IO.

Выполнил: Д. А. Ваньков
Группа: 8О-407Б-17
Преподаватели: А.Ю. Морозов,
К.Г. Крашенинников

Москва, 2020

Условие

Цель работы. Совместное использование технологии MPI и технологии CUDA. Применение библиотеки алгоритмов для параллельных расчетов Thrust. Реализация метода Якоби. Решение задачи Дирихле для уравнения Лапласа в трехмерной области с граничными условиями первого рода. Использование механизмов MPI-IO и производных типов данных.

Требуется решить задачу, описанную в лабораторной работе No7, используя возможности графических ускорителей, установленных на машинах вычислительного кластера. Учесть возможность наличия нескольких GPU в рамках одной машины. На GPU необходимо реализовать основной расчет. Требуется использовать объединение запросов к глобальной памяти. На каждой итерации допустимо копировать только граничные элементы с GPU на CPU для последующей отправки их другим процессам. Библиотеку Thrust использовать только для вычисления погрешности в рамках одного Процесса.

Запись результатов в файл должна осуществляться параллельно всеми процессами. Необходимо создать производный тип данных, определяющий шаблон записи данных в файл.

Вариант 1. MPI_Type_create_subarray.

Программное и аппаратное обеспечение

Graphics card: GeForce 940M

Размер глобальной памяти: 4242604032

Размер константной памяти: 65536

Размер разделяемой памяти: 49152

Максимальное количество регистров на блок: 65536

Максимальное количество потоков на блок: 1024

Количество мультипроцессоров: 3

OS: Linux Ubuntu 18.04

Редактор: CLion, Atom

Машины в кластере:

1. Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 16 Gb, GeForce GTX 1050, 2 Gb

2. Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 16 Gb, GeForce GT 545, 3 Gb

3. Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 16 Gb, GeForce GTX 650, 2 Gb

4. Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 12 Gb, GeForce GT 530, 2 Gb

5. Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 8 Gb, GeForce GT 530, 2 Gb

Все машины соединены гигабатным ethernet и находятся в подсети 10.10.1.1/24.

Версии софта: mpirun 1.10.2, g++ 4.8.4, nvcc 7.0

Метод решения

Для решения задачи на сетке заданного размера я использовал сетку процессов, каждый из которых имел свой участок памяти для обработки блока. Каждый процесс имел 2 равных по величине блока для того, чтобы на основе «старых» значений вычислять «новые» по формуле:

$$u_{i,j,k}^{(n+1)} = \frac{\left(u_{i+1,j,k}^{(n)} + u_{i-1,j,k}^{(n)}\right)h_x^{-2} + \left(u_{i,j+1,k}^{(n)} + u_{i,j-1,k}^{(n)}\right)h_y^{-2} + \left(u_{i,j,k+1}^{(n)} + u_{i,j,k-1}^{(n)}\right)h_z^{-2}}{2\left(h_x^{-2} + h_y^{-2} + h_z^{-2}\right)},$$

Проблема заключается в том, что расчет граничных значений требует значения, рассчитанные в другом блоке, что является не тривиальной задачей, требующей реализации межпроцессорного взаимодействия между соседями.

Схема решения:

1. Передать данные другим процессам на границах.

2. Обновить данные во всех ячейках.

3. Вычисление локальной (в рамках процесса) погрешности и во всей области.

Описание программы

Все расчеты новых значений на каждой из итераций алгоритма в данной лабораторной работе выполнялись на графическом процессоре с помощью технологии Cuda. Для этого я выделил 5 блоков данных на графическом процессоре, логика обновлений значений в которых аналогична тому, как мы вычисляли значения в лабораторной работе *MPI+OMP*. При этом для обмена данными между процессами я использую блоки, память для которых выделена на хосте и на каждой итерации я делаю копирования данных между device и host памятью для переопределения отсылаемых/принимаемых значений. Описал несколько кернелов для каждого из буфферов, а также для вычисления значений для следующей итерации и вычисления ошибки.

Для передачи данных я использую отправку send в зависимости от границы, в таком случае будет 6 отправок данных. Затем принимаю их с помощью recv и в зависимости от условия либо обновляю значения на полученные данные, либо на граничные. Для ускорения, так как после отправки происходит блокировка до момента принятия, я решил отправлять и принимать по 3 блока.

Во время расчета новых значений я постоянно изменяю значение разности по блоку. Это необходимо для контроля границы, поскольку, зная максимальное значение по блоку мы можем узнать максимум по всей сетке с помощью функции Allreduce, которая вернет все значения максимума по каждому из процессу, что позволит рассчитать значение для контроля простым проходом по выходному массиву.

Результаты

Я сравнил время выполнения двух разных программ: написанных с помощью MPI, CPU и на MPI + CUDA.

Размер блоков \ расчета	**MPI**	**CPU**	**MPI + CUDA**
1 1 1 | 30 30 30	17861ms	9143.1ms	4900.2 ms
2 2 2 | 15 15 15	4623.51ms	9451.6ms	29042.51 ms
2 2 5 | 15 15 6	5931.25ms	9767.7ms	81391.57 ms

Можно заметить, что совместное использование MPI и CUDA работает гораздо быстрее на небольших данных, однако существенно медленнее на больших. Вероятно, это происходит из-за постоянно перекопированния данных с host-а на device и наоборот. Когда данных немного расходы на копирования покрываются скоростью вычисления на графическом процессоре, но с ростом данных расходы превосходят преимущество в вычислении.

Выводы

После выполнения данной лабораторной работы я убедился, что параллельная обработка данных с несколькими процессами происходит гораздо быстрее, чем на CPU с одним процессом, даже несмотря на пересылки данных.

Во время выполнения возникали проблемы с оптимизацией, в связи с чем возникал тайм лимит. Отправка 6 блоков подряд оказалась медленной и была заменена на отправку по 3 блока и последующий их прием.

Во время выполнения столкнулся с проблемами в аллоцировании памяти. Работая с GPU, нужно внимательно инициалировать значения на графическом процессоре, в ином случае могут происходит просто невероятные вещи. Также из-за того, что не внимательно написал цикл в одном из кернелов, опечатавшись и итеруюясь в цикле не с тем индексом потерял много часов на исправление. Однако познакомился с ошибкой an illegal memory access was encountered.

Эта лабораторная познакомила меня с тем, как использовать CUDA с MPI, что позволяет разбивать программу на процессы, каждый из которых будет иметь несколько потоков исполнения. Также я познакомился с мультипроцессорным выводом в файл.