МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа №2
по курсу «Параллельная обработка данных»
Технология MPI и технология CUDA. MPI-IO.

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

Москва, 2021

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 №7, 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 3. MPI_Type_create_subarray

Программное и аппаратное обеспечение

GPU:

1. Compute capability: 7.5;
2. Графическая память: 4294967296;
3. Разделяемая память: 49152;
4. Константная память: 65536;
5. Количество регистров на блок: 65536;
6. Максимальное количество блоков: (2147483647, 65535, 65535);
7. Максимальное количество нитей: (1024, 1024, 64);
8. Количество мультипроцессоров: 6.

Сведения о системе:

1. Процессор: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
2. Память: 16,0 ГБ;
3. HDD: 237 ГБ.

Программное обеспечение:

1. OS: Windows 10;
2. IDE: Visual Studio 2019;
3. Компилятор: mpirun.

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 нетривиальной 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 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 буферов, а также для вычисления значений для следующей итерации и вычисления ошибки.

Для передачи и получения данных я использую sendrecv. Она комбинирует в одном обращении посылку сообщения одному получателю и прием сообщения от другого отправителя.

Затем вычисляю локальную (в рамках процесса) погрешность.

И вызываю Allgather.

Как работает allgather.

Он получает значения от одного процесса и отдаёт их всем остальным. В таком случае получается, что накапливаются значения во всех остальных процессах, включая тот самый. И в итоге мы получаем массив отсортированных по процессам значений, то есть первый процесс процесс - одно значение, второй процесс - второе и т.д. Таким образом мы можем накопить побочный максимум во всех остальных процессах и найти наибольший из них.

MPI_ALLGATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
sendbuf - указатель на буфер, sendcount - сколько слов отправляется, sendtype - тип отправленного (int, double), recvbuf - куда будем принимать, recvcount - сколько слов мы принимаем от одного другого процесса, включая этот, recvtype - тип полученного (int, double), comm - коммуникатор.

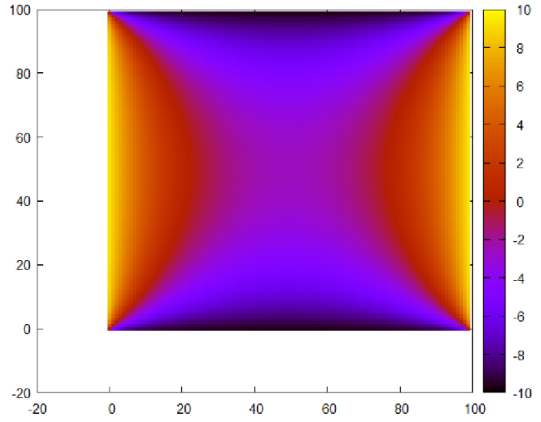
Затем сравниваю полученный максимум с eps, и если он меньше, прекращаю основной цикл.

Результаты

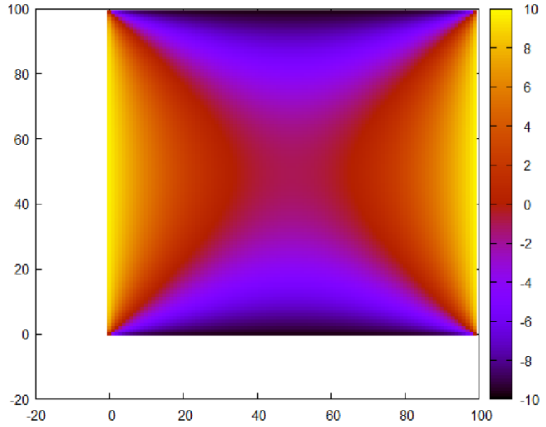
Размер блоков / расчета	**MPI**	**MPI + Cuda**
1 1 1 | 30 30 30	17861ms	4892ms
2 2 2 | 15 15 15	4621ms	29086ms
2 2 5 | 15 15 6	5938ms	81286ms

Можно заметить, что совместное использование MPI и CUDA работает гораздо быстрее на небольших данных, однако существенно медленнее на больших. Скорее всего, это происходит из-за постоянно копирования данных с host на device и наоборот. Когда данных немного, расходы на копирования покрываются скоростью вычисления на графическом процессоре.

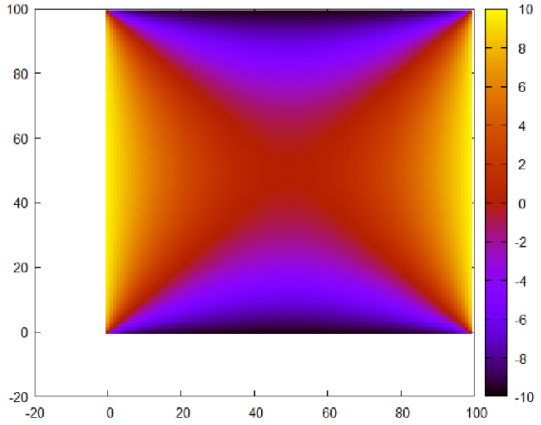
K=1


100
80
60
40
20
0
-20
-20
0
20
40
60
80
100
10
8
6
4
2
0
-2
-4
-6
-8
-10

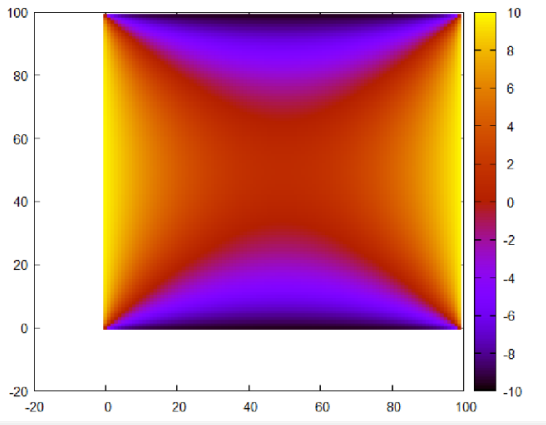
K=2


100
80
60
40
20
0
-20
-20
0
20
40
60
80
100
10
8
6
4
2
0
-2
-4
-6
-8
-10

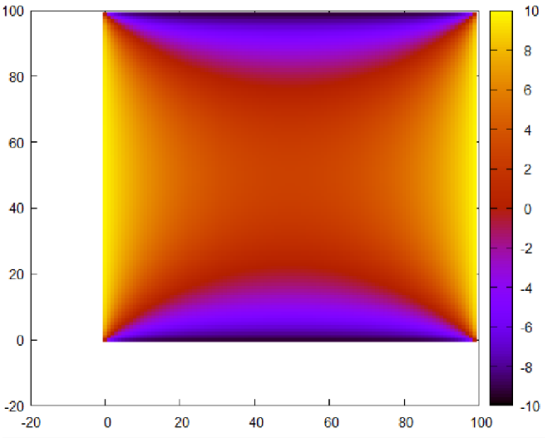
K = 3


100
80
60
40
20
0
-20
-20
0
20
40
60
80
100
10
8
6
4
2
0
-2
-4
-6
-8
-10

K = 4


100
80
60
40
20
0
-20
-20
0
20
40
60
80
100
10
8
6
4
2
0
-2
-4
-6
-8
-10

K = 5


100
80
60
40
20
0
-20
-20
0
20
40
60
80
100
10
8
6
4
2
0
-2
-4
-6
-8
-10

Выводы

Данный метод Дирихле являетя одним из конечно-разностных методов, которые широко используются для решения дифференциальных уравнений на заданной сетке с высокой степенью точности. Без этих методов сложно представить современную физику, которая использует эти методы для расчета уравнений теплопроводности при разных условиях среды.

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.

Во время выполнения столкнулся с проблемами в аллоцировании памяти.