МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа №1
по курсу «Параллельная обработка данных»
Message passing interface (MPI).

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

Москва, 2020

Условие

Цель работы: Знакомство с технологией MPI. Реализация метода Якоби.

Решение задачи Дирихле для уравнения Лапласа в трехмерной области с граничными условиями первого рода.

Вариант 2. Обмен граничными слоями через send/recv, контроль сходимости allreduce.

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

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 и на CPU.

Размер блоков \ расчета	MPI	CPU
1 1 1 | 30 30 30	17861ms	9143.1ms
2 2 2 | 15 15 15	4623.51ms	9451.6ms
2 2 5 | 15 15 6	5931.25ms	9767.7ms

Можно заметить, что использование нескольких процессов заметно ускоряет процесс вычисления.

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.