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: М.А. Бронников
Группа: М8О-407Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

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 №4: Ообмен граничными слоями через isend/irecv, контроль сходимости allgather.

Программное и аппаратное обеспечение компьютера:

Device: GeForce GT 545
Размер глобальной памяти: 3150381056
Размер константной памяти : 65536
Размер разделяемой памяти: 49152
Регистров на блок: 32768
Максимум потоков на блок: 1024
Количество мультипроцессоров : 3
OS: Linux Mint 20 Cinnamon
Редактор: VSCode
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. В качестве операционной системы установлена Ubuntu 16.04.6 LTS. Версии софта:
mpirun 1.10.2, g++ 4.8.4, nvcc 7.0

Метод решения

Для решения задачи на сетке заднного 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:

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

Проблема такого подхода заключается в том, что расчет граничных значений требует знаний о значениях, рассчитанных в другом блоке, что является не тривиальной задачей, требующей реализации межпроцессорного взаимодействия между соседями.

Общая схема решения:
1. На первом этапе происходит обмен граничными слоями между процессами.
2. На втором этапе выполняется обновление значений во всех ячейках.
3. Третий этап заключается в вычислении погрешности: сначала локально в рамках каждого процесса а потом через обмены и во всей области.

Описание программы

В своей реализации я решил не использовать окружающие «основной» блок «виртуальные» блоки, руководствуясь тем, что это ведет к лишнему перерасходу памяти и временным накладкам, за счет копирования из обменных буфферов в мнимые. Однако это породило массу лишнего кода, что нельзя назвать иначе как «костыли».

Для обмена я в самом начале итерации начинаю ассинхронный обмен с помощью функций isend/irecv. После чего, чтобы не терять время на ожидании конца приема, я начинаю проход по середине блока, расчитывая новые значения за исключением границ. После чего я ожидаю конца обменов и расчитываю границы.

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

Результаты:

Сравним время выполнения двух разных программ: написанных для CPU и MPI. Будем делать честное сравнение, поэтому замерим полное время выполнения программы, а не только основной цикл. Будем подбирать такие значения, чтобы размер общей сетки был одинаковым при разных запусках:

Сетка \ Расчет	MPI	CPU
1 1 1 | 40 40 40	19861ms	9843.8ms
2 2 2 | 20 20 20	4943.57ms	9952.2ms
2 2 4 | 20 20 10	6224.85ms	9890.5ms

Как видим, использование нескольких процессов дает существенный прирост программе MPI, а также превосходит по времени программу, написанную на CPU, однако этот прирост едва ли можно назвать впечатляющим, поскольку плюсы MPI в лице распараллеливания кода нивелируются минусами, такими как необходимость постоянно делать обмен данными между процессами, что довольно сильно сказывается на производительности, особенно во время записи результата.

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 научила меня работе с технологией межпроцессорного взаимодействия на больших вычислительных кластерах — MPI и дала урок того, что следует лучше продумывать концепцию решения задания перед началом его решения, чтобы решить его с наибольшей элегантностью и эффективностью.

Прирост полученный мной по сравнению с программой на CPU меня не удовлетворил, однако я надеюсь на более впечатляющие результаты в последующих лабораторных, где я рассмотрю связки различных технологий и буду использовать несколько потоков в каждом из процессов MPI.