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»

Технологии MPI и OpenMP

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 OpenMP. Реализация метода Якоби. Решение задачи Дирихле для уравнения Лапласа в трехмерной области с граничными условиями первого рода.

Вариант 1. Распараллеливание основных циклов через parallel for (+директива reduction для вычисления погрешности)

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 выполнения данной лабораторной работы необходима схема решения из лабораторной работы № 1. Однако второй этап можно распараллелить на каждом из

процессов (каждый цикл с принятем данных и перерасчетом) с помощью технологии OpenMP. Каждый процесс в потоке будет перерассчитывать значения для отдельного участка памяти в блоке.

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

В отличит от лабораторной № 1 добавились директивы препроцессора #pragma parallel, которые позволяют задавать участки, которые будут выполнятся в многопроцессорном режиме для каждого из потоков. Для того, чтобы каждый поток отвечал за отдельный участок, нужно было добавить эту директиву перед каждым циклом, за исключением вычисления погрешности.

#pragma omp parallel for private(i, j, k) shared(data, edge_yz)

Где edge_yz - буффер для конкретной области, a data - массив с данными.

#pragma omp parallel for private(i, j, k) shared(data, next) reduction(max: difference)

Директива reduction необходима для вычисления погрешности.

Для ускорения программы я использовал два потока.

omp_set_num_threads(2);

Результаты

Из прошлой лабораторной работы я взял сравнение времени выполнения двух разных программ: написанных с помощью MPI и на CPU и добавил MPI + OpenMP

Размер блоков \ расчета	**MPI + OpenMP**	**MPI**	**CPU**
1 1 1 | 30 30 30	7112.3ms	17861ms	9143.1ms

Как видно, результат превосходит всех остальных, несмотря даже на то, что эта программа внутри себя сталкивается со всеми издержками на пересылки и прием данных как MPI программа.

Также можно заметить, что использование нескольких потоков заметно ускоряет процесс вычисления.

Выводы

После выполнения данной лабораторной работы я убедился, что параллельная обработка данных с несколькими потоками, внутри которых находятся несколько процессов происходит гораздо быстрее.

Данная лабораторная работа познакомила меня с технологией OMP, позволяющей легко и быстро распараллеливать свой код с помощью легковесных потоков. Также я научился использовать ее вместе с MPI, что позволяет разбивать программу на процессы, каждый из которых будет иметь несколько потоков исполнения и получать существенный прирост в скорости вычисления.