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 OpenMP

Выполнил: И.Д. Черненко
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

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 задания: Вариант 1, Распараллеливание основных циклов через parallel for (+директива reduction для вычисления погрешности).
Входные данные: На первой строке заданы три числа: размер сетки процессов. Гарантируется, что при запуске программы количество процессов будет равно произведению этих трех чисел. На второй строке задается размер блока, который будет обрабатываться одним процессом: три числа. Далее задается путь к выходному файлу, в который необходимо записать конечный результат работы программы и точность ε. На последующих строках описывается задача: задаются
размеры области l_x, l_y и l_z, граничные условия: u_{down} , u_{up} , u_{left} , u_{right} , u_{front} ,u_{back} начальное значение u_0.

Выходные данные. В файл, определенный во входных данных, необходимо напечатать построчно значения ($u_{1,1,1}, u_{2,1,1}, ..., u_{1,2,1}, u_{2,2,1}, ..., u_{n_x-1,n_y,n_z}, u_{n_x,n_y,n_z}$) в ячейках сетки в формате с плавающей запятой с семью знаками мантиссы.

Программное и аппаратное обеспечение

Compute capability : 6.1
Name : GeForce GTX 1050
Total Global Memory : 2096103424
Shared memory per block : 49152
Registers per block : 65536
Max threads per block : (1024, 1024, 64)
Max block : (2147483647, 65535, 65535)
Total constant memory : 65536
Multiprocessors count : 5

OS:Linux
compiler:nvcc
IDE:VS Code
Editor:nano

Метод решения

За основу данной лабораторной была взята предыдущая, что существенно упростило задачу работы со своими типами, основные сложности возникли, как ни странно не во внедрении технологии OpenMP, а в реализации дополнительных типов для отправки

граничных слоев без дополнительных буферов, но это фактически не повлияло на реализацию, потому что всё распараллеливание в OpenMP, осуществляется с помощью простых директив. Иными словами в данной лабораторной мы распараллеливаем циклы в каждом конкретном процессе, в том числе расчет ошибки.

Описание программы

Вся программа была реализована в одном файле **main.cpp**.

parallel – создаём потоки

for – распределяем работу между потоками

Основной цикл:

```
#pragma omp parallel for private(i, j, k) shared(data, next_data, n_x, n_y, n_z, h_x, h_y, h_z) reduction(max:maximum_error)
    for (i = 0; i < n_x; i++) {
        for (j = 0; j < n_y; j++) {
            for (k = 0; k < n_z; k++) {

                double h_x_squared = h_x * h_x;
                double h_y_squared = h_y * h_y;
                double h_z_squared = h_z * h_z;

                double add1 = (data[i + 2 + (j + 1) * (n_x + 2) + (k + 1) * (n_x + 2) * (n_y + 2)] + data[i + (j + 1) * (n_x + 2) + (k + 1) * (n_x + 2) * (n_y + 2)]) / h_x_squared;
                double add2 = (data[i + 1 + (j + 2) * (n_x + 2) + (k + 1) * (n_x + 2) * (n_y + 2)] + data[i + 1 + j * (n_x + 2) + (k + 1) * (n_x + 2) * (n_y + 2)]) / h_y_squared;
                double add3 = (data[i + 1 + (j + 1) * (n_x + 2) + (k + 2) * (n_x + 2) * (n_y + 2)] + data[i + 1 + (j + 1) * (n_x + 2) + k * (n_x + 2) * (n_y + 2)]) / h_z_squared;
                double devider = 2 * (1.0 / h_x_squared + 1.0 / h_y_squared + 1.0 / h_z_squared);

                int tmp = i + 1 + (j + 1) * (n_x + 2) + (k + 1) * (n_x + 2) * (n_y + 2);
                next_data[tmp] = (add1 + add2 + add3) / devider;
                maximum_error = std::max(maximum_error, fabs(next_data[_i(i, j, k)] - data[_i(i, j, k)]));
            }
        }
    }
```

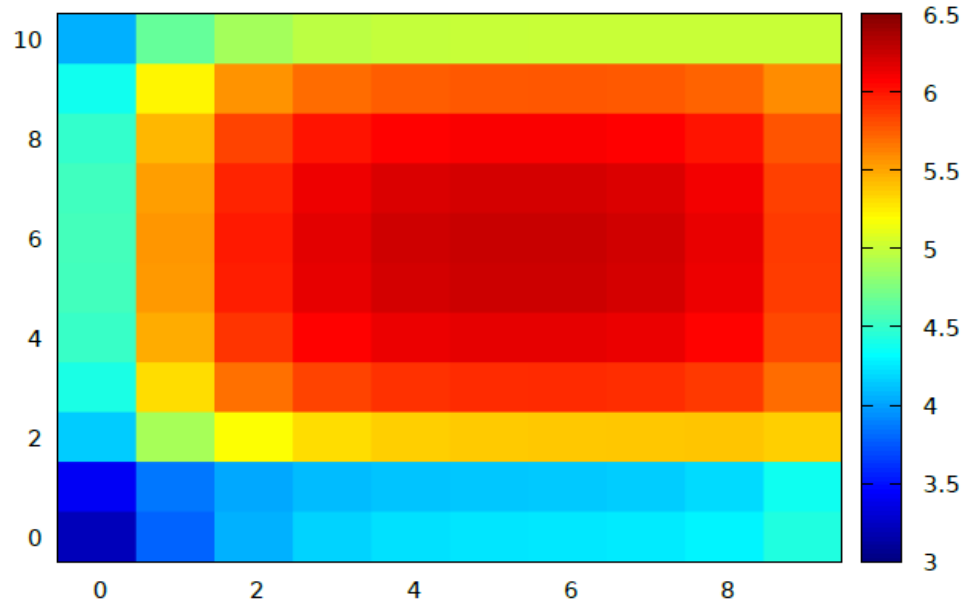
Результаты

net	block	mpi_cuda	mpi	cpu	mpi_open
1,1,1	10 10 10	166.618ms	43.646ms	16.698ms	41.32ms
1,1,1	20 20 20	779.714ms	498.166ms	248.949ms	345.65ms
2,2,2	10 10 10	627.18ms	164.939ms	254.609ms	145.23ms
2,2,2	20 20 20	2741ms	2361.61ms	7269.34ms	1634.12ms

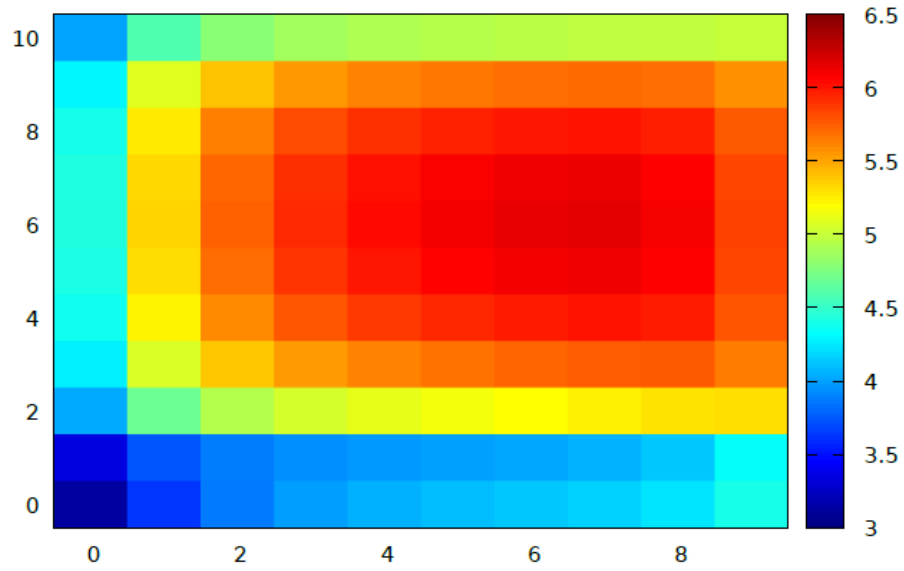
4,4,1	10 10 10	3224.13ms	904.25ms	1247.11ms	456.31ms
4,4,1	20 20 20	18634.7ms	14074ms	35821.9ms	10435ms

Полученные изображения температуры в заданной области:

epsilon = 1e-1



epsilon 1e-10



Из полученных данных можно сделать вывод, что с использованием OpenMP, можно получить значительное ускорение в производительности программы.

Выводы

Данную лабораторную всё также можно использовать для решения задач математической физики. При реализации были получены навыки работы с тандемом

OpenMP и MPI, а также с созданием производных типов данных в MPI и распараллеливанием программы с помощью простейших директив из OpenMP.