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

Москва, 2021

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 задания: Вариант 1, обмен граничными слоями через send/receive, контроль сходимости allgather

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 ($u_{1,1,1}, u_{2,1,1}, \ldots, u_{1,2,1}, u_{2,2,1}, \ldots, u_{n_x-1,n_y,n_z}, u_{n_x,n_y,n_z}$) 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

Главный процесс считывает данные необходимые для задачи и отправляет их прочим процессам, после чего происходит сам итерационный процесс, при котором процессы обмениваются граничными значениями с помощью MPI_Send и MPI_Recv. После этого новые значения рассчитываются для каждой ячейки, исходя из предыдущих значений, за контроль сходимости отвечает функция MPI_Allgather, благодаря которой происходит вычисление максимума погрешности, исходя из информации для всех процессов. Далее данные выводятся в файл.

Описание программы

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

Часть программы для расчёта значений:

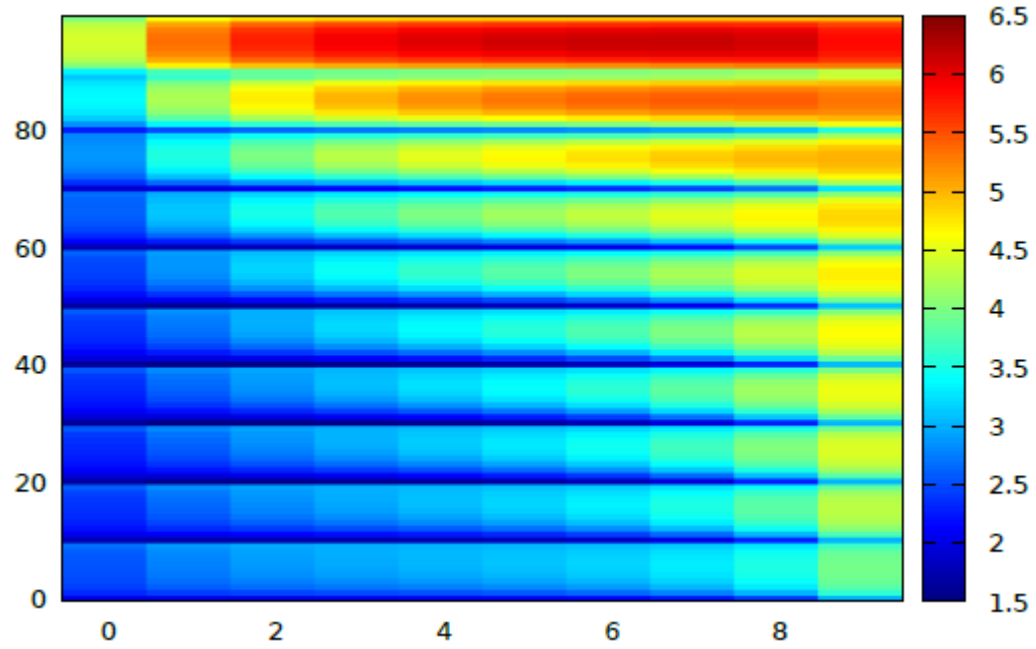
```
for (int i = 0; i < n_x; i++) {
    for (int j = 0; j < n_y; j++) {
        for (int k = 0; k < n_z; k++) {

            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;

            error = std::max(error, fabs(next_data[tmp] - data[tmp]));
        }
    }
}
```

Результаты

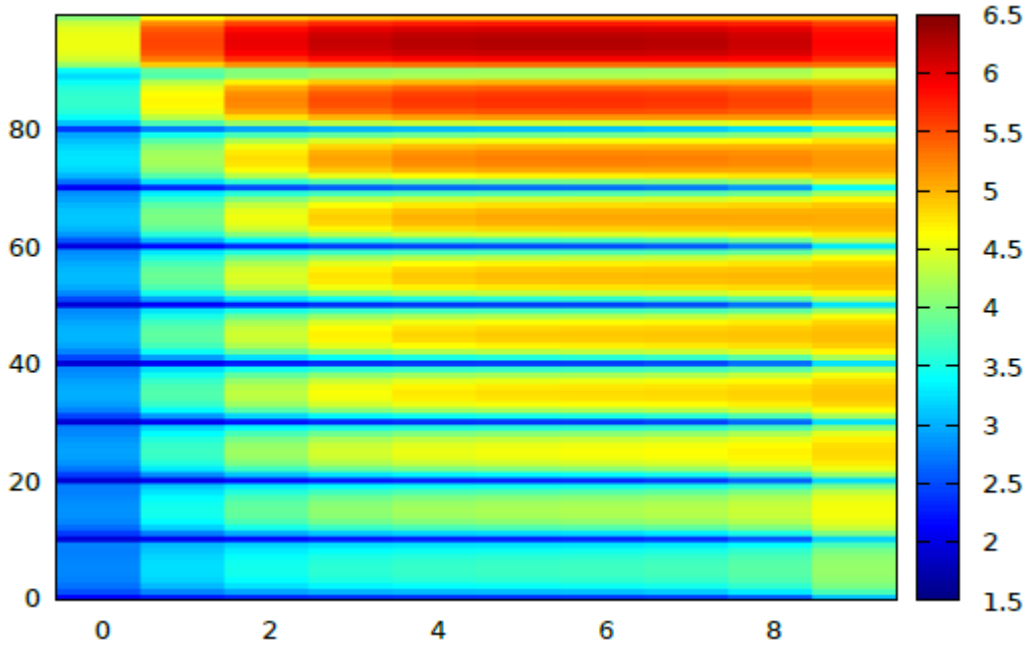
net	block	mpi	cpu
1,1,1	10 10 10	43.646ms	16.698ms
1,1,1	20 20 20	498.166ms	248.949ms
2,2,2	10 10 10	164.939ms	254.609ms
2,2,2	20 20 20	2361.61ms	7269.34ms

net	block	mpi	cpu
1,1,1	10 10 10	43.646ms	16.698ms
1,1,1	20 20 20	498.166ms	248.949ms
2,2,2	10 10 10	164.939ms	254.609ms
2,2,2	20 20 20	2361.61ms	7269.34ms
4,4,1	10 10 10	904.25ms	1247.11ms
4,4,1	20 20 20	14074ms	35821.9ms

Полученные изображения температуры в заданной области:
epsilon = 10e-7



Epsilon=10e-1



Из полученных данных можно сделать вывод, что при правильных настройках вычисления с помощью **MPI** превосходят **CPU** примерно в **2-3 раза**, что особенно заметно при грамотном распараллеливании и больших объемах блоков.

Выводы

Реализованный алгоритм вполне можно использовать для решения дифференциальных уравнений, что полезно, например, в физике.
Лабораторная работа оказалась вполне несложной благодаря примеру из лекции, по сути, граничные условия были взяты оттуда, формула для индексации была также сделана по аналогии с лекции, при этом изначально решение падало на тесте из-за неправильного распределения индексов при выводе и внутри основного цикла.
Но по итогу полученная реализация превзошла вычисления на CPU, что очевидно так как сама задача довольно неплохо распараллеливается с помощью MPI.