МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №1
по курсу «Параллельная обработка данных»

Message Passing Interface

Выполнил: М. А. Инютин
Группа: М8О-407Б-19
Преподаватель: А. Ю. Морозов

Москва, 2022

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 3. Обмен граничными слоями через `sendrecv`**, контроль сходимости** `allgather`**.**

Программное и аппаратное обеспечение

Характеристики системы:

- Процессор: «Intel i7-9750H (12) @ 4.500GHz»
- Память: два модуля «Kingston KHX2666C15S4/16G 16384 MB @ 2667MHz»
- SSD: «KINGSTON RBUSNS8154P3256GJ (E8FK11.C) 256 GB»

Программное обеспечение:

- ОС: «Ubuntu 20.04.5 LTS x86_64»
- Текстовый редактор: «Visual Studio Code 1.74.1»
- Компилятор: «mpicxx for MPICH version 3.3.2»

Метод решения

Над пространством строится регулярная сетка. С каждой ячейкой сопоставляется значение функции u в точке соответствующей центру ячейки. Граничные условия реализуются через виртуальные ячейки, которые окружают рассматриваемую область. Поиск решения сводится к итерационному процессу:

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

Процесс останавливается, как только

$$\max_{i,j,k} |u_{i,j,k}^{(n+1)} - u_{i,j,k}^{(n)}| < \varepsilon$$

Описание программы

Межпроцессное взаимодействие осуществляется с помощью вызовов `MPI_Sendrecv`. Рассмотрим обмен граничными значениями по координате z. Так как процесс сразу и отправляет, и читает, то другой процесс должен отправлять и читать в эту сторону, то есть один процесс вверх, а другой вниз, и наоборот. Реализую данный обмен в две фазы: сперва все процессы с чётным индексом по координате z обращаются к нижнему процессу, а все с нечётными к верхнему. На второй фазе чётные и нечётные меняются местами.

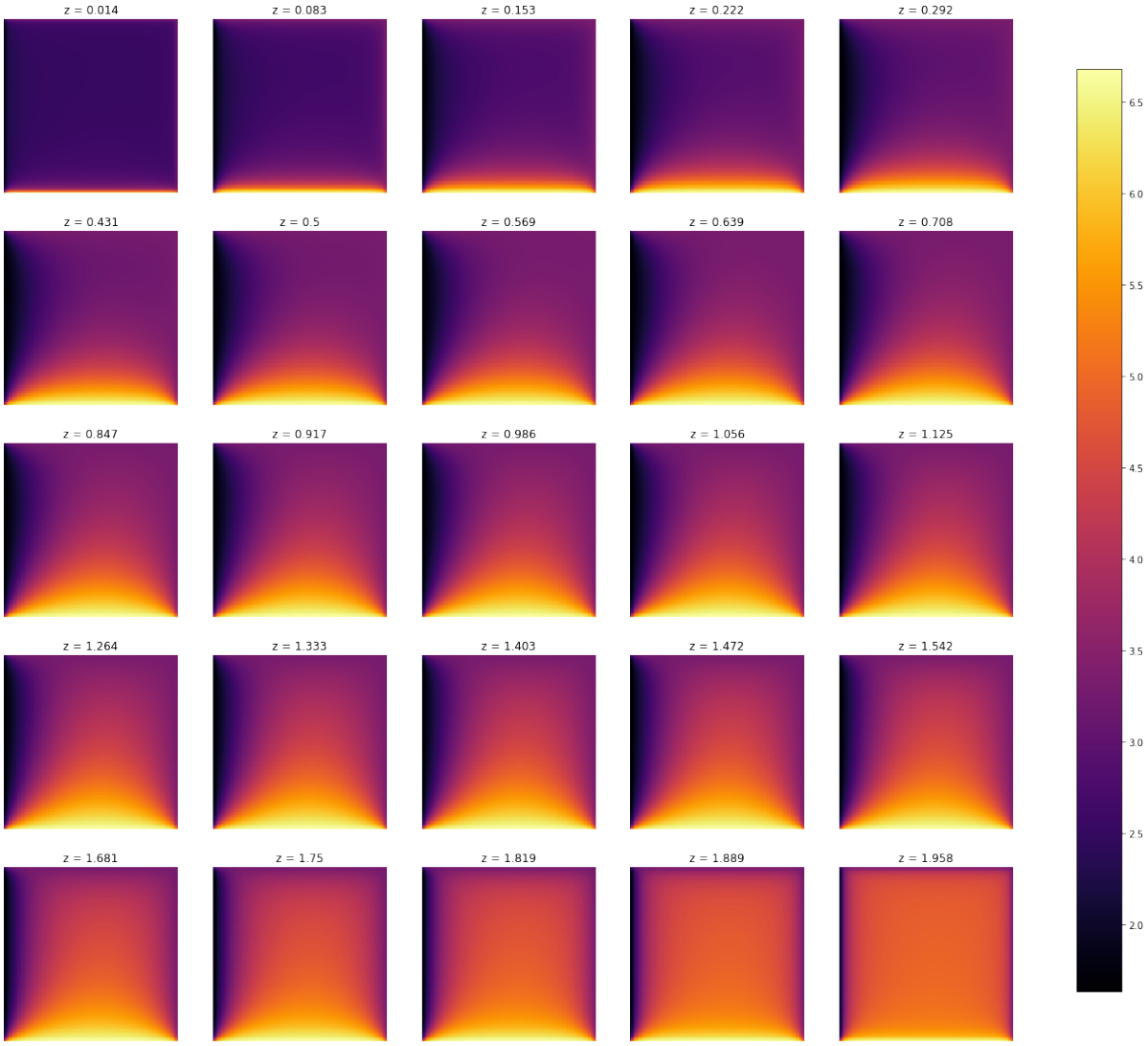
Аналогично происходит обмен по координатам x и y. После обмена выполняется шаг итерации на всех процессах независимо, вычисляется погрешность на шаге, происходит обмен вычисленными погрешностями через `MPI_Allgather`, вычисление максимальной погрешности по всем процессам и переход на следующую итерацию.

Результаты

Замеры времени работы программы с разным количеством процессов и разным размером сетки. Сетка имеет размер $n \times n \times n$.

Кол-во процессов	$n = 12$, мс	$n = 24$, мс	$n = 48$, мс	$n = 96$, мс	$n = 144$, мс
1, без MPI	0.911	24.122	639.690	17232.042	70445.164
2	0.927	14.587	369.080	9146.040	38472.441
3	1.203	11.756	201.487	6332.821	27606.756
4	1.125	11.698	270.489	4816.114	21962.017
6	2.112	13.897	204.099	4338.967	19534.214
8	2.643	13.243	252.020	4182.772	20196.098
12	5.036	15.314	183.897	3427.126	19790.645
16	22904.747	65687.236	> 120 сек	> 120 сек	> 120 сек

Визулизация результата



Размеры области:$l_x = 2, l_y = 2, l_z = 2$.

Начальные условия: $u_{down} = 3, u_{up} = 7, u_{left} = 2, u_{right} = 5, u_{front} = 1, u_{back} = 3$.

Выводы

Задача Дирихле вытекает из трёхмерного уравнения теплопроводности, которые необходимо решать при расчётах распределения температуры, например, корпуса космического корабля. Теория разностных схем широко применяется в решении дифференциальных уравнений в частных производных.

Метод Якоби позволяет найти решение задачи для разностной схемы с помощью итерационного процесса. Пусть итерационный процесс сходится за k шагов, тогда сложность алгоритма $O(k \cdot n_x \cdot n_y \cdot nz)$, где n_x, n_y, n_z — размеры сетки по соответствующим координатам. Если эти размеры равны, то сложность будет $O(k \cdot n^3)$. Метод Зейделя сходится гораздо быстрее метода Якоби, однако его труднее распараллелить.

Результаты показали, что при маленькой сетке большое количество процессов только замедляем программу из-за затрат на синхронизацию. На большой сетке видно уменьшение времени исполнения вплоть до 6 процессов, 6-12 процессов примерно одинаково. Так как мой процессор имеет 6 физических ядер и 12 поток, то при запуске на 16 процессах программа сильно замедляется, из-за того, что часть процессов находится в блокировке.