###### МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

**Кафедра параллельных вычислений**

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«Параллельная реализация метода Якоби в трехмерной области»

студента 2 курса, 23210 группы

**Лаухина Егора Денисовича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Артюхов А.А.

Новосибирск 2025

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_Toc101966097)

[ЗАДАНИЕ 3](#_Toc101966098)

ОПИСАНИЕ РАБОТЫ 3

ЗАКЛЮЧЕНИЕ 4

Приложение 1. Графики 4

Приложение 2. Код программы 5

[Приложение 3. Профилирование 8](#_Toc101966103)

# **ЦЕЛЬ**

Практическое освоение методов распараллеливания численных алгоритмов на регулярных сетках на примере реализации метода Якоби в трехмерной области

# **ЗАДАНИЕ**

1. **Реализация алгоритма:**
   * Разработать программу на MPI, решающую дифференциальное уравнение методом Якоби в трехмерной области в случае одномерной декомпозиции объекту
2. **Исследование производительности:**
   * Провести тесты для разных конфигураций решетки.
   * Измерить время выполнения и сравнить с последовательной версией.
   * Построить графики зависимости времени от размера матрицы и числа процессов.
3. **Профилирование программы:**
   * Выполнить профилирование программы

# **ОПИСАНИЕ РАБОТЫ**

1. **Реализация и проверка алгоритма**
   * Был успешно реализован **параллельный алгоритм решения дифференциального уравнения в трехмерной области** в MPI.
2. **Исследование производительности**
   * Проведены запуски программы с разным количеством ядер
   * Составлены **таблицы зависимости времени выполнения** от:
     + Количества используемых ядер
   * Построены **графики ускорения (Sp​)** и **эффективности (Ep​)** в зависимости от числа процессов
3. **Профилирование программы**
   * Выполнено **профилирование на 16 процессах**
   * Проанализированы:
     + Распределение нагрузки между процессами.
     + Время, затраченное на **коммуникации** (передачи данных) и **вычисления**.
     + Узкие места производительности
4. **Вывод**
   * На основе анализа полученных таблиц и графиков был сформулирован вывод.

Число узлов сетки по каждой координатной оси: 350

|  |  |  |  |
| --- | --- | --- | --- |
| ***Полученные результаты*** | | | |
| **Кол-во процессов** | **Время выполнения** | **Эффективность** | **Ускорение** |
| 1 | 27,9711 | 100% | 1 |
| 2 | 15,8943 | 88% | 1,75 |
| 4 | 12,9793 | 55% | 2,16 |
| 8 | 9,59185 | 37% | 2,9 |
| 12 | 7,39449 | 32% | 3,7 |
| 16 | 5,89235 | 29% | 4,7 |

# ЗАКЛЮЧЕНИЕ

Анализ таблицы позволяет сделать вывод, что использование 3-х мерной области в случае с одноименной декомпозиции области и асинхронной коммуникации между процессами в разы уменьшает время выполнения программы, но при этом видно, что увеличение числа процессов приводит к снижению эффективности.

# ГРАФИК

**Код Программы**

#include <mpi.h>  
#include <cmath>  
#include <iostream>  
#include <vector>  
#include <algorithm>  
  
using namespace std;  
  
#define IDX(i, j, k) ((i)\*(Ny+2)\*(Nz+2) + (j)\*(Nz+2) + (k))  
  
const double a = 1e5;  
const double epsilon = 1e-8;  
const int Nx = 350, Ny = 350, Nz = 350;  
const double X0 = -1.0, Y0 = -1.0, Z0 = -1.0;  
const double Dx = 2.0, Dy = 2.0, Dz = 2.0;  
const double hx = Dx / Nx;  
const double hy = Dy / Ny;  
const double hz = Dz / Nz;  
  
void initialize(vector<double> &phi, vector<double> &rho, int i\_start, int i\_end) {  
 for (int i = i\_start; i <= i\_end; ++i) {  
 double x = X0 + i \* hx;  
 for (int j = 0; j <= Ny+1; ++j) {  
 double y = Y0 + j \* hy;  
 for (int k = 0; k <= Nz+1; ++k) {  
 double z = Z0 + k \* hz;  
 int idx = IDX(i - i\_start + 1, j, k);  
 if (i == 0 || i == Nx || j == 0 || j == Ny || k == 0 || k == Nz) {  
 phi[idx] = x\*x + y\*y + z\*z;  
 } else {  
 phi[idx] = 0.0;  
 }  
 rho[idx] = 6.0 - a \* (x\*x + y\*y + z\*z);  
 }  
 }  
 }  
}  
  
int main(int argc, char\*\* argv) {  
 MPI\_Init(&argc, &argv);  
 auto startTime = MPI\_Wtime();  
 int rank, size;  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  
  
 int chunk = Nx / size;  
 int i\_start = rank \* chunk;  
 int i\_end = (rank == size - 1) ? Nx : i\_start + chunk;  
  
 int local\_Nx = i\_end - i\_start + 1;  
 vector<double> phi((local\_Nx+2)\*(Ny+2)\*(Nz+2), 0.0);  
 vector<double> phi\_new((local\_Nx+2)\*(Ny+2)\*(Nz+2), 0.0);  
 vector<double> rho((local\_Nx+2)\*(Ny+2)\*(Nz+2), 0.0);  
  
 initialize(phi, rho, i\_start, i\_end);  
  
 int iter = 0;  
 double max\_diff;  
 MPI\_Request reqs[4];  
  
 do {  
 if (rank > 0) {  
 MPI\_Isend(&phi[IDX(1, 0, 0)], (Ny+2)\*(Nz+2), MPI\_DOUBLE, rank - 1, 0, MPI\_COMM\_WORLD, &reqs[0]);  
 MPI\_Irecv(&phi[IDX(0, 0, 0)], (Ny+2)\*(Nz+2), MPI\_DOUBLE, rank - 1, 1, MPI\_COMM\_WORLD, &reqs[1]);  
 }  
 if (rank < size - 1) {  
 MPI\_Isend(&phi[IDX(local\_Nx, 0, 0)], (Ny+2)\*(Nz+2), MPI\_DOUBLE, rank + 1, 1, MPI\_COMM\_WORLD, &reqs[2]);  
 MPI\_Irecv(&phi[IDX(local\_Nx+1, 0, 0)], (Ny+2)\*(Nz+2), MPI\_DOUBLE, rank + 1, 0, MPI\_COMM\_WORLD, &reqs[3]);  
 }  
  
 for (int i = 2; i <= local\_Nx - 1; ++i)  
 for (int j = 1; j < Ny; ++j)  
 for (int k = 1; k < Nz; ++k) {  
 int idx = IDX(i, j, k);  
 phi\_new[idx] = (  
 phi[IDX(i+1,j,k)] + phi[IDX(i-1,j,k)] ) / (hx\*hx) +  
 (phi[IDX(i,j+1,k)] + phi[IDX(i,j-1,k)] ) / (hy\*hy) +  
 (phi[IDX(i,j,k+1)] + phi[IDX(i,j,k-1)] ) / (hz\*hz) -  
 rho[idx]  
 ;  
 phi\_new[idx] /= (2.0/hx/hx + 2.0/hy/hy + 2.0/hz/hz + a);  
 }  
  
 if (rank > 0) { MPI\_Wait(&reqs[1], MPI\_STATUS\_IGNORE); }  
 if (rank < size - 1) { MPI\_Wait(&reqs[3], MPI\_STATUS\_IGNORE); }  
  
 for (int i : **{**1, local\_Nx**}**)  
 for (int j = 1; j < Ny; ++j)  
 for (int k = 1; k < Nz; ++k) {  
 int idx = IDX(i, j, k);  
 phi\_new[idx] = (  
 phi[IDX(i+1,j,k)] + phi[IDX(i-1,j,k)] ) / (hx\*hx) +  
 (phi[IDX(i,j+1,k)] + phi[IDX(i,j-1,k)] ) / (hy\*hy) +  
 (phi[IDX(i,j,k+1)] + phi[IDX(i,j,k-1)] ) / (hz\*hz) -  
 rho[idx]  
 ;  
 phi\_new[idx] /= (2.0/hx/hx + 2.0/hy/hy + 2.0/hz/hz + a);  
 }  
  
 max\_diff = 0.0;  
 for (int i = 1; i <= local\_Nx; ++i)  
 for (int j = 1; j < Ny; ++j)  
 for (int k = 1; k < Nz; ++k) {  
 int idx = IDX(i,j,k);  
 max\_diff = max(max\_diff, fabs(phi\_new[idx] - phi[idx]));  
 phi[idx] = phi\_new[idx];  
 }  
  
 double global\_diff;  
 MPI\_Allreduce(&max\_diff, &global\_diff, 1, MPI\_DOUBLE, MPI\_MAX, MPI\_COMM\_WORLD);  
 max\_diff = global\_diff;  
 iter++;  
  
 } while (max\_diff > epsilon);  
  
 if (rank == 0) {  
// cout << "Converged in " << iter << " iterations." << endl;  
 cout << "time: " << MPI\_Wtime() - startTime << endl;  
 }  
 MPI\_Finalize();  
 return 0;  
}

**Профилирование**

