Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Курсовая работа

по курсу «Математическое моделирование»

Студент: Куликов А.В.
Группа: М8О-408Б-17
Преподаватель: Тишкин В.Ф.
Оценка:

Москва, 2021

Вариант № 8:

ρ_1	ρ_2	ε_1	ε_2	u_1	u_2	γ
4.4	4	4	4	0	0	$\frac{7}{4}$

Условие задачи:

Требуется численно решить задачу для уравнений газовой динамики:

$$\begin{cases} \frac{\partial \rho}{\partial t}+\frac{\partial(\rho u)}{\partial x}=0, \\ \frac{\partial(\rho u)}{\partial t}+\frac{\partial(\rho u^2+p)}{\partial x}=0, \\ \frac{\partial(\rho E)}{\partial t}+\frac{\partial(\rho u E+p u)}{\partial x}=0, \\ p=(\gamma-1)\rho\varepsilon, \gamma>0, \\ E=\frac{u^2}{2}+\varepsilon, \\ -\infty\le x\le+\infty, t\ge 0, \end{cases}$$

т.е. необходимо найти пять неизвестных $\rho(x,t)$, $u(x,t)$, $\varepsilon(x,t)$, $p(x,t)$ и $E(x,t)$, построить соответствующие графики. Допускается использование любого языка программирования.

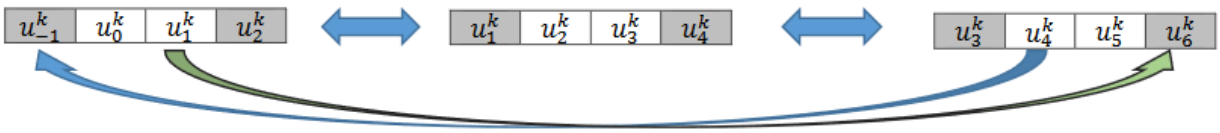
Дополнительное задание:

Распараллелить получившуюся программу с использованием технологии MPI либо другой подобной технологии, допускающей обработку данных на кластере машин. Например, CUDA (параллельные вычисления на видеокарте), OpenMP (потоки на одной машине), std::thread (потоки на одной машине)не годятся для этой цели. Схема распараллеливания для трёх процессов выглядит следующим образом.

Схема распараллеливания для трёх процессов выглядит следующим образом.

(N=5):

Каждый процесс отвечает за свой кусок сетки:



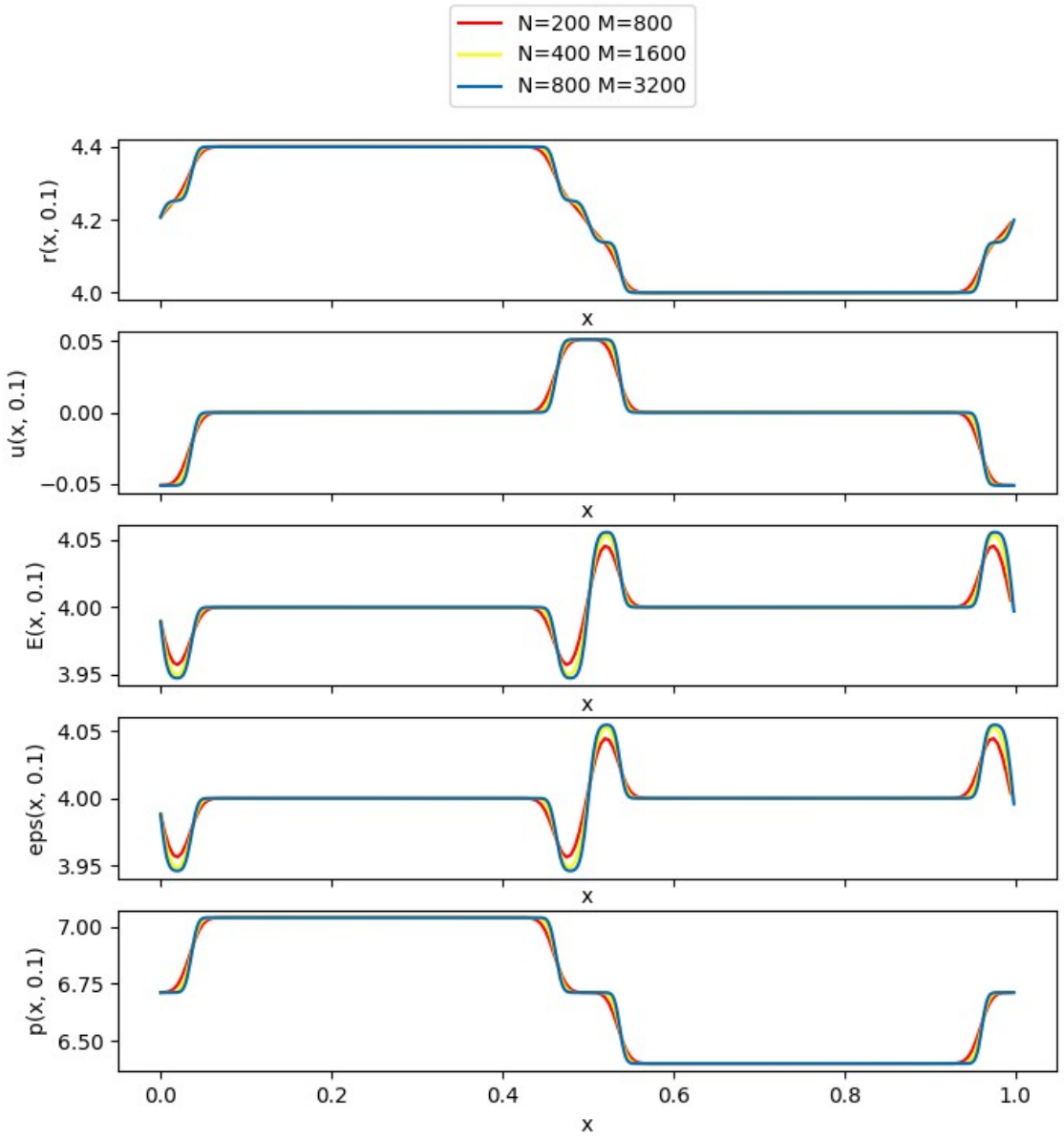
Серым цветом выделены виртуальные ячейки. После выполнения одной итерации алгоритма (вычисления одного временного слоя) необходимо выполнить обмен значениями граничных узлов с целью заполнения виртуальных ячеек.

Метод решения:

Для решения поставленной задачи реализована программа для решения задачи для уравнений газовой динамики. С помощью нее получено решение для задачи.

По полученному решению строятся графики при разных параметрах разбиения сетки. Ниже представлены примеры графиков решения задачи с параметрами, соответствующими варианту, в момент времени t = 0.1 при следующих разбиениях:

N (кол-во отсчетов по X)	M (кол-во отсчетов по T)
200	800
400	1600
800	3200



Вывод:

В ходе выполнения курсовой работы была реализована программа для численного решения задачи для уравнения газовой динамики с использованием MPI и OpenMP на языке C++.

В конечном итоге получено численное решение задачи и построены графики целевых функций: $\rho(x,t)$, $u(x,t)$, $E(x,t)$, $\varepsilon(x,t)$ и $p(x,t)$.

Листинг программного кода:

main.cpp

```
#include <cstdlib>
#include <string>
#include <iostream>
#include <fstream>
#include <cmath>
#include <iomanip>

#include <mpi.h>
#include <omp.h>

#define index(k, i) ((k) * (N + 2) + (i) + 1)

#define LEFT  0x10000000
#define RIGHT 0x01000000

#define R_TAG 0
#define U_TAG 1
#define E_TAG 2
#define EPS_TAG 3
#define P_TAG 4

int mod(int a, int b)
{
    int r = a % b;
    return r < 0 ? r + b : r;
}

struct Parameters{
    double alpha;
    double gamma;
    double dt;
    double dx;
    double r10, r20, eps1, eps2, u1, u2;
    double max_t, target_t;
    int N, M;
};

void compute(double *r, double *u, double* E, double* eps, double* p, int N,
int k, const Parameters& params){
    #pragma omp parallel for num_threads(omp_get_max_threads())
    for(int i = 0; i < N + 1; ++i){

             double cik = std::sqrt(params.gamma * (params.gamma - 1) *
eps[index(k - 1, i)]);

           double cikminus = std::sqrt(params.gamma * (params.gamma - 1) *
eps[index(k - 1, i - 1)]);
            double cikplus = std::sqrt(params.gamma * (params.gamma - 1) *
eps[index(k - 1, i + 1)]);
        double aiplus = params.alpha * std::max(std::fabs(u[index(k - 1, i)])
+ cik, std::fabs(u[index(k - 1, i + 1)]) + cikplus);
        double aiminus = params.alpha * std::max(std::fabs(u[index(k - 1, i -
1)]) + cikminus, std::fabs(u[index(k - 1, i)]) + cik);
           double ruplus = (((r[index(k - 1, i)] * u[index(k - 1, i)]) +
(r[index(k - 1, i + 1)] * u[index(k - 1, i + 1)])) / 2)
            - aiplus * ((r[index(k - 1, i + 1)] - r[index(k - 1, i)]) / 2);

        double ruminus = (((r[index(k - 1, i - 1)] * u[index(k - 1, i - 1)])
+ (r[index(k - 1, i)] * u[index(k - 1, i)])) / 2)
```

```
            - aiminus * ((r[index(k - 1, i)] - r[index(k - 1, i - 1)]) / 2);

          r[index(k, i)] = ((ruminus - ruplus) / params.dx) * (params.dt) +
r[index(k - 1, i)];

           double rup_plus = (((r[index(k - 1, i)] * u[index(k - 1, i)] *
u[index(k - 1, i)] + p[index(k - 1, i)])
            + (r[index(k - 1, i)] * u[index(k - 1, i + 1)] * u[index(k - 1, i
+ 1)] + p[index(k - 1, i + 1)])) / 2)
             - aiplus * ((r[index(k - 1, i + 1)] * u[index(k - 1, i + 1)] -
r[index(k - 1, i)] * u[index(k - 1, i)]) / 2);

        double rup_minus = (((r[index(k - 1, i - 1)] * u[index(k - 1, i - 1)]
* u[index(k - 1, i - 1)] + p[index(k - 1, i - 1)])
            + (r[index(k - 1, i)] * u[index(k - 1, i)] * u[index(k - 1, i)] +
p[index(k - 1, i)])) / 2) - aiminus * ((r[index(k - 1, i)]
             * u[index(k - 1, i)] - r[index(k - 1, i - 1)] * u[index(k - 1, i
- 1)]) / 2);

         u[index(k, i)] = (((rup_minus - rup_plus) / params.dx) * (params.dt)
+ r[index(k - 1, i)] * u[index(k - 1, i)]) / r[index(k, i)];

            double fplus = (((r[index(k - 1, i)] * u[index(k - 1, i)] *
eps[index(k - 1, i)] + p[index(k - 1, i)] * u[index(k - 1, i)])
             + (r[index(k - 1, i + 1)] * u[index(k - 1, i + 1)] * eps[index(k
- 1, i + 1)] + p[index(k - 1, i + 1)]
              * u[index(k - 1, i + 1)])) / 2) - aiplus * ((r[index(k - 1, i +
1)] * eps[index(k - 1, i + 1)] - r[index(k - 1, i)]
            * eps[index(k - 1, i)]) / 2);

         double fminus = (((r[index(k - 1, i - 1)] * u[index(k - 1, i - 1)] *
eps[index(k - 1, i - 1)] + p[index(k - 1, i - 1)]
             * u[index(k - 1, i - 1)]) + (r[index(k - 1, i)] * u[index(k - 1,
i)] * eps[index(k - 1, i)] + p[index(k - 1, i)]
              * u[index(k - 1, i)])) / 2) - aiminus * ((r[index(k - 1, i)] *
eps[index(k - 1, i)] - r[index(k - 1, i - 1)]
            * eps[index(k - 1, i - 1)]) / 2);

          E[index(k, i)] = (((fminus - fplus) / params.dx) * (params.dt) +
E[index(k - 1, i)] *r[index(k - 1, i)]) / r[index(k, i)];

        eps[index(k, i)] = E[index(k, i)] - std::pow(u[index(k, i)], 2) / 2;
         p[index(k, i)] = (params.gamma - 1) * r[index(k, i)] * eps[index(k,
i)];
    }
}

void Exchange(double* arr, int tag, int rank, int k, int N, int n_processes){
        MPI_Status status;

        int left_process_rank = mod(rank - 1, n_processes);
        int right_process_rank = mod(rank + 1, n_processes);

        if(n_processes == 1){
            arr[index(k, N)] = arr[index(k, 0)];
            arr[index(k, -1)] = arr[index(k, N-1)];
            return;
        }

        if(rank & 1){
            MPI_Sendrecv(&arr[index(k, 0)], 1, MPI_DOUBLE, left_process_rank,
LEFT | tag,
                         &arr[index(k, -1)], 1, MPI_DOUBLE, left_process_rank,
RIGHT | tag,
```

```
                    MPI_COMM_WORLD, &status);

                        MPI_Sendrecv(&arr[index(k,  N-1)],  1,  MPI_DOUBLE,
right_process_rank, RIGHT | tag,
                       &arr[index(k, N)], 1, MPI_DOUBLE, right_process_rank,
LEFT | tag,
                    MPI_COMM_WORLD, &status);
        }
        else{
                        MPI_Sendrecv(&arr[index(k,  N-1)],  1,  MPI_DOUBLE,
right_process_rank, RIGHT | tag,
                       &arr[index(k, N)], 1, MPI_DOUBLE, right_process_rank,
LEFT | tag,
                    MPI_COMM_WORLD, &status);

            MPI_Sendrecv(&arr[index(k, 0)], 1, MPI_DOUBLE, left_process_rank,
LEFT | tag,
                       &arr[index(k, -1)], 1, MPI_DOUBLE, left_process_rank,
RIGHT | tag,
                    MPI_COMM_WORLD, &status);
        }
}

void  RecieveAndSaveData(const  std::string&  path,  double*  arr,  int  tag,
double* buffer, int N, int M, int n_processes){
        std::ofstream ofs(path);
        if(!ofs){
            std::cerr << "Can't open" << path << std::endl;
            exit(0);
        }

        MPI_Status status;
        for(int k = 0; k < M+1; ++k){
            for(int i = 0; i < n_processes; ++i){
                if(i == 0){
                    for(int j = 0; j < N; ++j)
                                ofs << std::fixed << std::setprecision(10) <<
std::scientific << arr[index(k, j)] << ' ';
                }
                else{
                      MPI_Recv(buffer, N, MPI_DOUBLE, i, tag, MPI_COMM_WORLD,
&status);
                    for(int j = 0; j < N; ++j)
                                ofs << std::fixed << std::setprecision(10) <<
std::scientific << buffer[j] << ' ';
                }
            }
            ofs << std::endl;
        }
}

void SendData(double* arr, int tag, int N, int M){
    for(int k = 0; k < M+1; ++k){
        MPI_Send(&arr[index(k, 0)], N, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD);
    }
}

void  SetInitialConditions(double*  r,  double*  u,  double*  E,  double*  eps,
double* p, int N, int rank, const Parameters& params){
    for(int i = -1; i <= N; ++i){
        double x = (rank * N + i) * params.dx;
        double r0 = x < 0.5 ? params.r10 : params.r20;
        double eps0 = x < 0.5 ? params.eps1 : params.eps2;
        double u0 = x < 0.5 ? params.u1 : params.u2;
```

```
        r[index(0, i)] = r0;
        u[index(0, i)] = u0;
        eps[index(0, i)] = eps0;
        p[index(0, i)] = (params.gamma - 1.0) * r0 * eps0;
        E[index(0, i)] = u0 * u0 / 2.0 + eps0;
    }
}

int main(int argc, char** argv){

    bool verbose = false;
    if(argc > 1 && !strcmp(argv[1], "-v"))
        verbose = true;

    MPI_Init(&argc, &argv);

      int rank;
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
      int n_processes;
      MPI_Comm_size(MPI_COMM_WORLD, &n_processes);

    Parameters params;
    std::string output_file_path;

    if(rank == 0){
        std::cin >> params.r10 >> params.r20 >> params.eps1 >> params.eps2
            >> params.u1 >> params.u2 >> params.gamma >> params.alpha
            >> params.max_t >> params.target_t >> params.N >> params.M;
        std::cin >> output_file_path;

        if(verbose)
                  std::cout << "output_file_path: " << output_file_path <<
std::endl;
      }


    MPI_Status status;
    MPI_Bcast(&params, sizeof(Parameters), MPI_CHAR, 0, MPI_COMM_WORLD);

    int process_idx = rank;
    int n_points_per_process = params.N / n_processes;

    if(verbose && rank == 0){
        std::cout << "r10: " << params.r10 << std::endl;
        std::cout << "r20: " << params.r20 << std::endl;
        std::cout << "eps1: " << params.eps1 << std::endl;
        std::cout << "eps2: " << params.eps2 << std::endl;
        std::cout << "u1: " << params.u1 << std::endl;
        std::cout << "u2: " << params.u2 << std::endl;
        std::cout << "gamma: " << params.gamma << std::endl;
        std::cout << "alpha: " << params.alpha << std::endl;
        std::cout << "max_t: " << params.max_t << std::endl;
        std::cout << "target_t: " << params.target_t << std::endl;
        std::cout << "N: " << params.N << std::endl;
        std::cout << "M: " << params.M << std::endl;
           std::cout << "n_points_per_process: " << n_points_per_process <<
std::endl;
    }

    int size = (n_points_per_process + 2) * (params.M + 1);
    double* r = new double[size];
    double* u = new double[size];
    double* E = new double[size];
```

```
    double* eps = new double[size];
    double* p = new double[size];

    params.dx = 1.0 / params.N;
    params.dt = params.max_t / params.M;

       SetInitialConditions(r, u, E, eps, p, n_points_per_process, rank,
params);

    for (int k = 1; k < params.M + 1; ++k){
        double t_stop = k * params.dt;
        if(t_stop >= params.target_t + params.dt){
            break;
        }

        Exchange(r, R_TAG, rank, k-1, n_points_per_process, n_processes);
        Exchange(u, U_TAG, rank, k-1, n_points_per_process, n_processes);
        Exchange(E, E_TAG, rank, k-1, n_points_per_process, n_processes);
        Exchange(eps, EPS_TAG, rank, k-1, n_points_per_process, n_processes);
        Exchange(p, P_TAG, rank, k-1, n_points_per_process, n_processes);

        MPI_Barrier(MPI_COMM_WORLD);

        compute(r, u, E, eps, p, n_points_per_process, k, params);
    }

    MPI_Barrier(MPI_COMM_WORLD);

    double* buffer = new double[n_points_per_process];

    if(rank == 0){
           RecieveAndSaveData(output_file_path + "r.txt", r, R_TAG, buffer,
n_points_per_process, params.M, n_processes);
           RecieveAndSaveData(output_file_path + "u.txt", u, U_TAG, buffer,
n_points_per_process, params.M, n_processes);
           RecieveAndSaveData(output_file_path + "E.txt", E, E_TAG, buffer,
n_points_per_process, params.M, n_processes);
            RecieveAndSaveData(output_file_path + "eps.txt", eps, EPS_TAG,
buffer, n_points_per_process, params.M, n_processes);
           RecieveAndSaveData(output_file_path + "p.txt", p, P_TAG, buffer,
n_points_per_process, params.M, n_processes);

        std::string path = output_file_path + "x.txt";
        std::ofstream ofs(path);
        if(!ofs){
            std::cerr << "Can't open" << path << std::endl;
            exit(0);
        }
        for(int i = 0; i < params.N; ++i){
             ofs << std::fixed << std::setprecision(10) << std::scientific <<
i * params.dx << ' ';
        }
        ofs << std::endl;
    }
    else{
        SendData(r, R_TAG, n_points_per_process, params.M);
        SendData(u, U_TAG, n_points_per_process, params.M);
        SendData(E, E_TAG, n_points_per_process, params.M);
        SendData(eps, EPS_TAG, n_points_per_process, params.M);
        SendData(p, P_TAG, n_points_per_process, params.M);
    }

      MPI_Barrier(MPI_COMM_WORLD);
    delete[] buffer;
```

```
    delete[] r;
    delete[] u;
    delete[] E;
    delete[] eps;
    delete[] p;

    MPI_Finalize();
}
```