Отчет по третьему заданию MPI

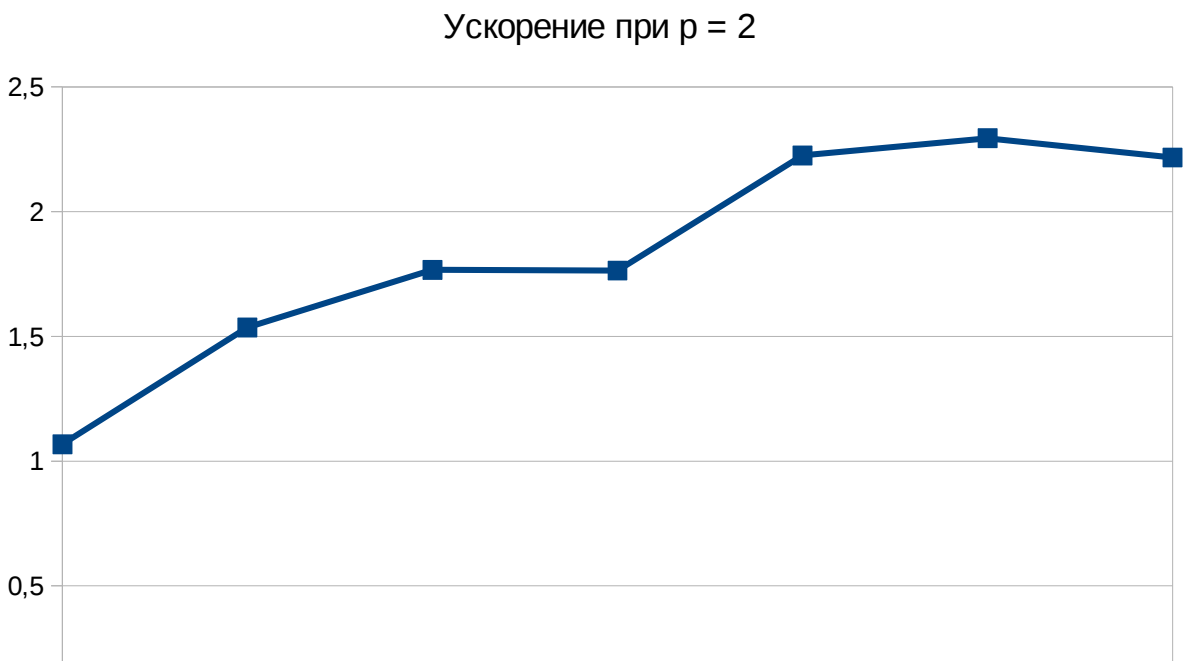
выполнил Женин Максим Николаевич, 523 группа
maximham@mail.ru

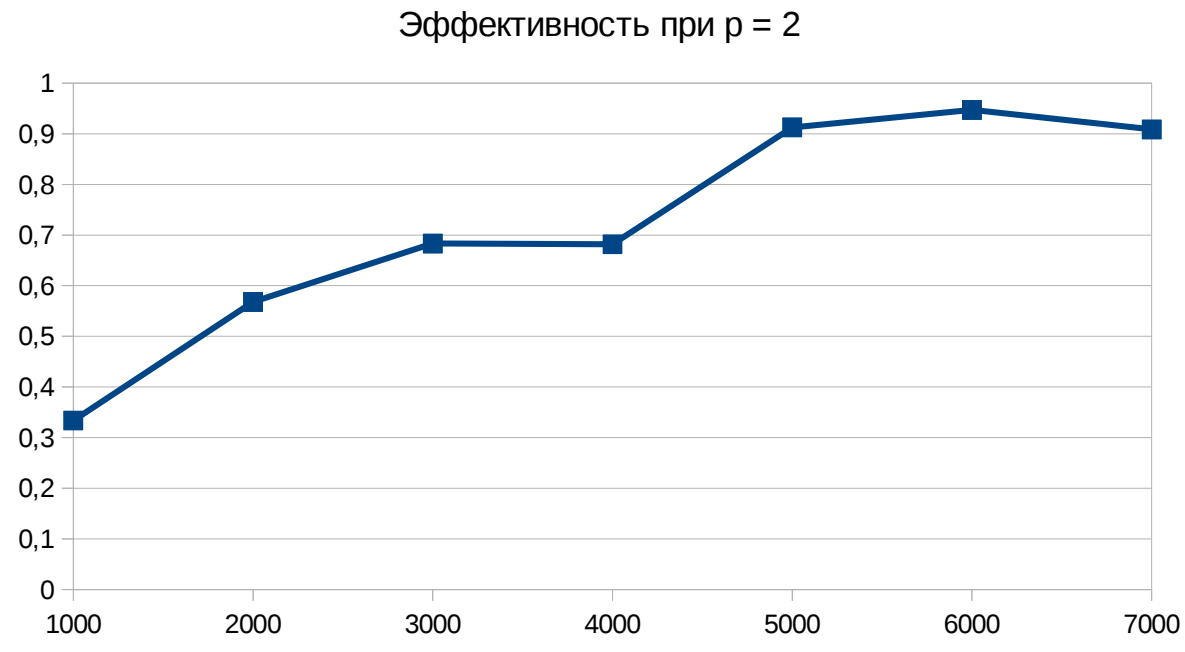
Конфигурация системы:
Своя система локальная 4 ядра

Выполнял программу с размером n = 1000 … 7000, при T = 100

Произвел замеры для 1 и 2 рабочих процессов(не включая мастер процесс), так как дальше на локальной машине графики замедлялись(не хватало ядер)


Время работы при p=2
6
5
4
3
2
1
0
1000
2000
3000
4000
5000
6000
7000


Ускорение при p = 2
2,5
2
1,5
1
0,5


Эффективность при p = 2
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1000
2000
3000
4000
5000
6000
7000

```
#include <fstream>
#include "mpi.h"
#include <cmath>
#include <iostream>
#include <time.h>

using namespace std;

int Nprocs = 1;
int Rank;

int f(int* data, int i, int j, int n)
{
int state = data[i*(n+2)+j];
int s = -state;
for( int ii = i - 1; ii <= i + 1; ii ++ )
for( int jj = j - 1; jj <= j + 1; jj ++ )
s += data[ii*(n+2)+jj];
if( state==0 && s==3 )
return 1;
if( state==1 && (s<2 || s>3) )
return 0;
return state;
}

void update_data(int n, int* data, int* temp)
{
//cout << "Update data" << endl;
for( int i=1; i<=n; i++ )
for( int j=1; j<=n; j++ )
temp[i*(n+2)+j] = f(data, i, j, n);
}

void init(int n, int* data, int* temp)
{
//cout << "Init" << endl;
for( int i=0; i<(n+2)*(n+2); i++ )
data[i] = temp[i] = 0;
int n0 = 1+n/2;
int m0 = 1+n/2;
data[(n0-1)*(n+2)+m0] = 1;
data[n0*(n+2)+m0+1] = 1;
for( int i=0; i<3; i++ )
data[(n0+1)*(n+2)+m0+i-1] = 1;
}

void setup_boundaries(int n, int* data)
{
//cout << "Setup Bounderies" << endl;
for( int i=0; i<n+2; i++ )
```

```
{
data[i*(n+2)+0] = data[i*(n+2)+n];
data[i*(n+2)+n+1] = data[i*(n+2)+1];
}
for( int j=0; j<n+2; j++ )
{
data[0*(n+2)+j] = data[n*(n+2)+j];
data[(n+1)*(n+2)+j] = data[1*(n+2)+j];
}
}

void setup_boundaries_mpi(int n, int * data, int p) {
//cout << "Setup Boundaries MPI" << endl;
MPI_Status status;
        int i = (Rank - 1) / p;
int j = (Rank - 1) % p;
int left = ((j == 0) ? Rank + p - 1: Rank - 1);
int right = ((j == p - 1) ? Rank - p + 1: Rank + 1);
int up = ((i == 0) ? p - 1: i - 1) * p + j + 1;
int down = ((i == p - 1) ? 0: i + 1) * p + j + 1;
MPI_Datatype datasend;
MPI_Type_vector(n + 2, 1, n + 2, MPI_INT, &datasend);
MPI_Type_commit(&datasend);
MPI_Sendrecv(&data[1], 1, datasend, left, 0, &data[n + 1], 1, datasend, right, 0,
MPI_COMM_WORLD, &status);
MPI_Sendrecv(&data[n], 1, datasend, right, 0, &data[0], 1, datasend, left, 0,
MPI_COMM_WORLD, &status);
MPI_Sendrecv(&data[n + 2], n + 2, MPI_INT, up, 0, &data[(n + 2) * (n + 1)], n + 2,
MPI_INT, down, 0, MPI_COMM_WORLD, &status);
MPI_Sendrecv(&data[(n + 2) * n], n + 2, MPI_INT, down, 0, &data[0], n + 2, MPI_INT, up, 0,
MPI_COMM_WORLD, &status);

}

void collectdata(int *data, int n, int p) {
//cout << "Collect Data" << endl;
MPI_Status status;
if (Rank == 0) {
MPI_Datatype datasend;
MPI_Type_vector(n, n, n * p + 2, MPI_INT, &datasend);
MPI_Type_commit(&datasend);
for(int i = 0 ; i < p; ++i) {
for(int j = 0 ; j < p; ++j) {
MPI_Recv(&(data[(i * p + j) / p * (n * p + 2) * n + (i * p + j) % p * n + (n * p) + 2 + 1]), 1,
datasend, i * p + j + 1, 0, MPI_COMM_WORLD, &status);
}
}
} else {
MPI_Datatype datasend;
MPI_Type_vector(n, n, n + 2, MPI_INT, &datasend);
MPI_Type_commit(&datasend);
```

```
MPI_Send(&data[n + 2 + 1], 1, datasend, 0, 0, MPI_COMM_WORLD);
}
}

void distribute_data(int *data, int n, int p, int Rank) {
//cout << "Distribute data" << endl;
//cout << "P " << p << endl;
MPI_Status status;
if (Rank == 0) {
MPI_Datatype block;
int N = n * p;
MPI_Type_vector(n + 2, n + 2, N + 2, MPI_INT, &block);
MPI_Type_commit(&block);
for(int i = 0 ; i < p; ++i) {
//cout << i << endl;
for(int j = 0 ; j < p; ++j) {
//cout << j << endl;
//cout << (i * p + j) / p * (N + 2) * n + (i * p + j) % p * n << endl;
//cout << i * p + j + 1 << endl;
MPI_Send(&(data[(i * p + j) / p * (N + 2) * n + (i * p + j) % p * n ]), 1, block, i * p + j + 1, 0,
MPI_COMM_WORLD);
}
}
} else {
MPI_Recv(data, (n + 2) * (n + 2), MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
}
}


void run_life_mpi(int n, int T)
{
//cout << "Run Life" << endl;
int p = (int)sqrt(Nprocs - 1);
int N = n * p;
int *data;
int *temp;
if (Rank == 0) {
data = new int[(N+2)*(N+2)];
temp = new int[(N+2)*(N+2)];
init(N, data, temp);
setup_boundaries(N, data);
} else {
data = new int[(n + 2) * (n + 2)];
temp = new int[(n + 2) * (n + 2)];
}
distribute_data(data, n, p, Rank);
double start_time, end_time;
start_time = MPI_Wtime();
for( int t = 0; t < T; t++ )
{
if (Rank != 0) {
```

```
update_data(n, data, temp);
setup_boundaries_mpi(n, temp, p);
swap(data, temp);
}
}
end_time = MPI_Wtime();
double time = end_time - start_time;
double maxtime = 0.0;
MPI_Reduce(&time,&maxtime,1,MPI_DOUBLE_PRECISION,MPI_MAX,0,MPI_COMM_WORLD);
collectdata(data, n, p);

if (Rank == 0) {
ofstream f("output.dat");
cout << maxtime << endl;
for( int i=1; i<=N; i++ )
{
for( int j=1; j<=N; j++ )
f << data[i*(N+2)+j];
f << endl;
}
f.close();
}
delete[] data;
delete[] temp;
}

int main(int argc, char** argv)
{
if (MPI_Init(&argc, &argv) != MPI_SUCCESS) {
fprintf(stderr, "failed to init MPI\n");
exit(1);
}
if (MPI_Comm_size(MPI_COMM_WORLD, &Nprocs) != MPI_SUCCESS ||
MPI_Comm_rank(MPI_COMM_WORLD, &Rank) != MPI_SUCCESS) {
fprintf(stderr, "failed to get communicator size or Rank\n");
exit(1);
}
int n = atoi(argv[1]);
int T = atoi(argv[2]);
// cout << Rank << endl;
run_life_mpi(n, T);

MPI_Finalize();

return 0;
}
```