Отчет по выполнению практического задания «Игра Жизнь» по курсу Естественные модели параллельных вычислений студента 523 группы

Ухина Сергея Алексеевича.
ser191097@gmail.com

Графики зависимостей времени работы, ускорения и эффективности для различных размеров задач и различного количества использованных процессов

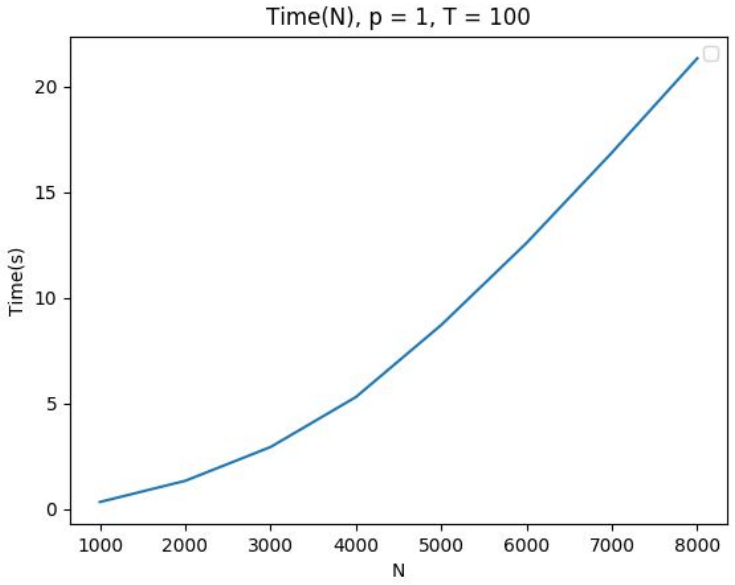

Рис. 1: Время работы при p = 1

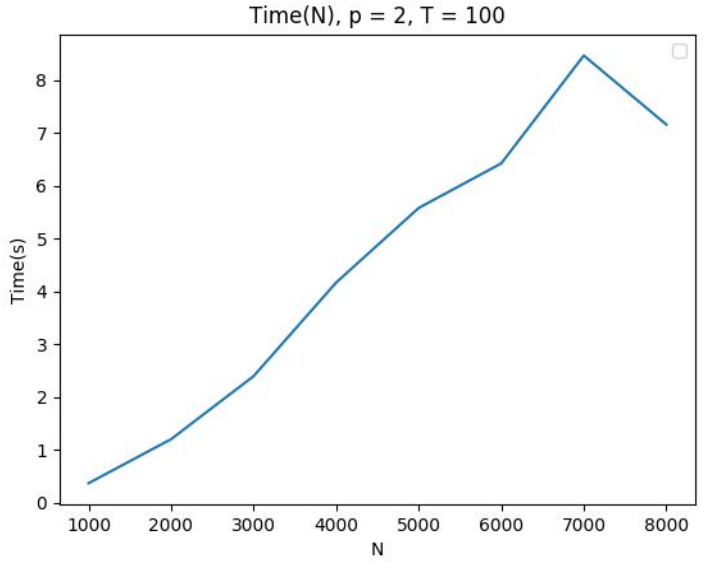

Рис. 2: Время работы при p = 2

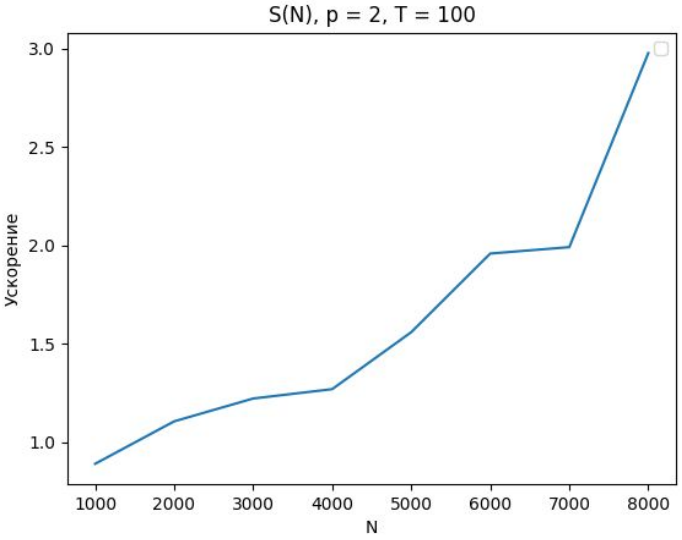

Рис. 3: Ускорение при p = 2

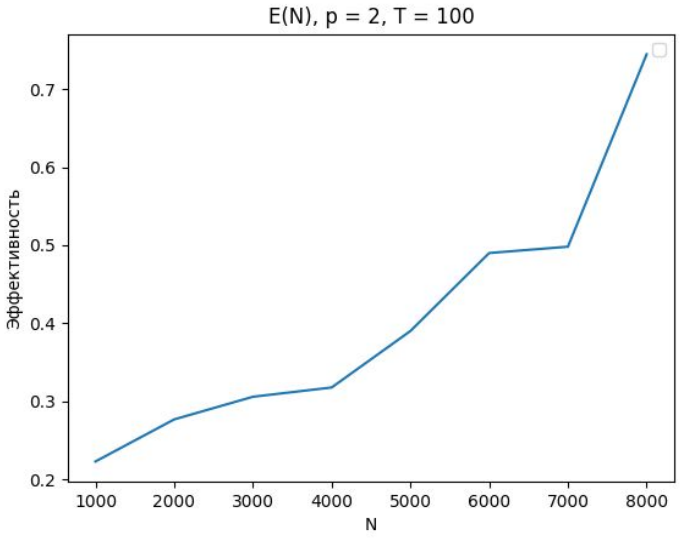

Рис. 4: Эффективность при p = 2

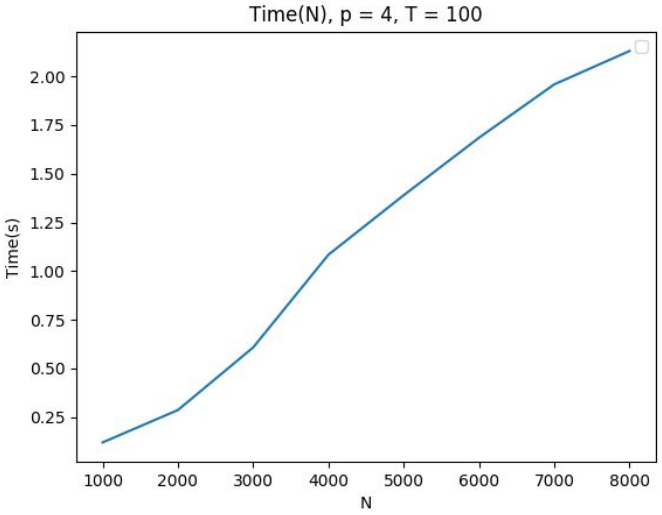

Рис. 5: Время работы при p = 4

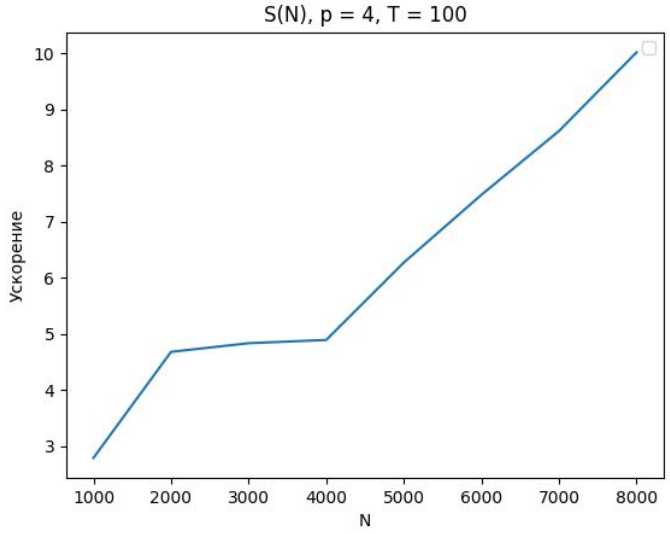

Рис. 6: Ускорение при p = 4

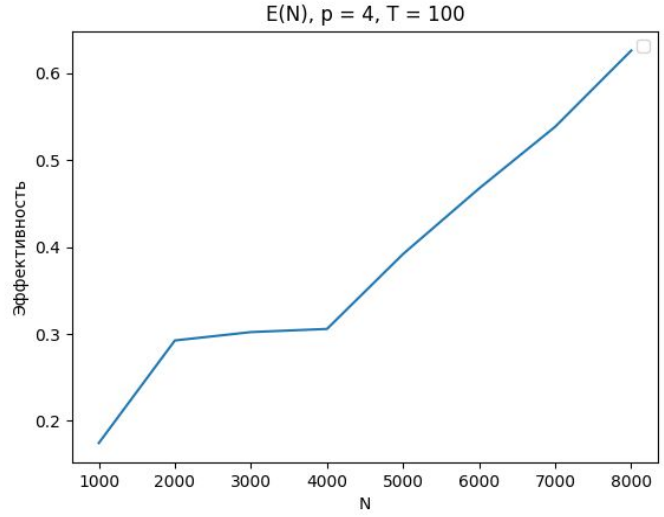

Рис. 7: Эффективность при p = 4

Листинг 1: Код программы

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

using namespace std;

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)
{
   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)
{
   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)
{
   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 rank, int p) {
    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 above = (i == 0) ? p - 1: i - 1;
    above = above * p + j + 1;
    int below = (i == p - 1) ? 0: i + 1;
    below = below * p +j + 1;
    MPI_Datatype column;
    MPI_Type_vector(n + 2, 1, n + 2, MPI_INT, &column);
    MPI_Type_commit(&column);
    MPI_Sendrecv(&data[1], 1, column, left, 0, &data[n + 1], 1, column, right, 0,
MPI_COMM_WORLD, 0);
    MPI_Sendrecv(&data[n], 1, column, right, 0, &data[0], 1, column, left, 0,
MPI_COMM_WORLD, 0);
    MPI_Sendrecv(&data[n + 2], n + 2, MPI_INT, above, 0, &data[(n + 2) * (n + 1)], n + 2,
MPI_INT, below, 0, MPI_COMM_WORLD, 0);
    MPI_Sendrecv(&data[(n + 2) * n], n + 2, MPI_INT, below, 0, &data[0], n + 2, MPI_INT,
above, 0, MPI_COMM_WORLD, 0);

}

void collectdata(int *data, int n, int p, int rank) {
    if (rank == 0) {
        MPI_Datatype blockrecv;
        int N = n * p;
        MPI_Type_vector(n, n, N + 2, MPI_INT, &blockrecv);
        MPI_Type_commit(&blockrecv);
        for(int i = 0 ; i < p; ++i) {
            for(int j = 0 ; j < p; ++j) {
                MPI_Recv(&(data[(i * p + j) / p * (N + 2) * n + (i * p + j) % p * n  + N + 2 + 1]), 1,
blockrecv, i * p + j + 1, 0, MPI_COMM_WORLD, 0);
            }
        }
    } else {
        MPI_Datatype blocksend;
        int N = n * p;
        MPI_Type_vector(n, n, n + 2, MPI_INT, &blocksend);
```

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

void distribute_data(int *data, int n, int p, int rank) {
    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) {
            for(int j = 0 ; j < p; ++j) {
                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, 0);
    }
}


void run_life(int n, int T, int rank, int size)
{
    int p = (int) floor(sqrt(size));
    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;
    MPI_Barrier(MPI_COMM_WORLD);
    start_time = MPI_Wtime();
    for( int t = 0; t < T; t++ )
    {
        if (rank != 0) {
            update_data(n, data, temp);
            setup_boundaries_mpi(n, temp, rank, p);
            swap(data, temp);
```

```
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
    end_time = MPI_Wtime();
    double time = end_time - start_time;
    collectdata(data, n, p, rank);

    if (rank == 0) {
        ofstream f("output.dat");
        ofstream stats("stat.txt", std::ofstream::out | std::ofstream::app);
        stats << "n = " << n << "; T = " << T << "; P = " << size << endl;
        stats << "time = " << time << endl;
        stats << "-----------------------------------------" << 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)
{

    MPI_Init(&argc, &argv);
    int rank;
    int size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank) ;
    MPI_Comm_size(MPI_COMM_WORLD, &size) ;
    int n = atoi(argv[1]);
    int T = atoi(argv[2]);

    run_life(n, T, rank, size);

    MPI_Finalize();

    return 0;
}
```