**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 ПРАКТИЧЕСКОЙ РАБОТЫ**

«ИГРА "ЖИЗНЬ" ДЖ. КОНВЕЯ С ИСПОЛЬЗОВАНИЕМ НЕБЛОКИРУЮЩИХ КОММУНИКАЦИЙ MPI»

студентки 2 курса, группы 22204

**Клочихиной Софьи Павловны**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А. Ю. Власенко

Новосибирск 2024

**СОДЕРЖАНИЕ**

ЦЕЛЬ 3

ЗАДАНИЕ 4

ОПИСАНИЕ РАБОТЫ 5

ЗАКЛЮЧЕНИЕ 9

ПРИЛОЖЕНИЕ.

Полный листинг параллельной программы на C++ 10

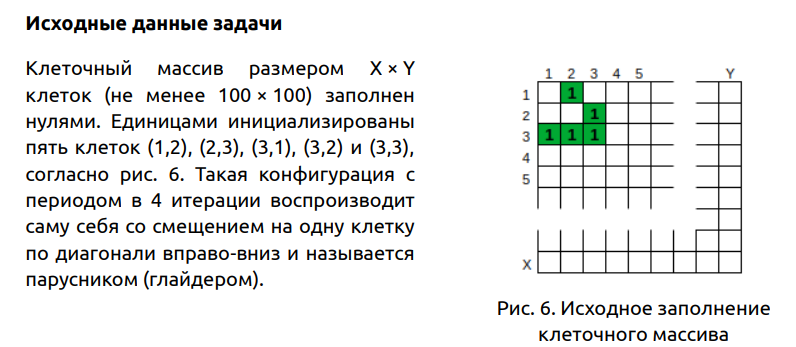
Полный листинг скрипта SLURM 13

**ЦЕЛЬ**

1. Практическое освоение методов реализации алгоритмов мелкозернистого параллелизма на крупноблочном параллельном вычислительном устройстве на примере реализации клеточного автомата «Игра "Жизнь" Дж. Конвея» с использованием неблокирующих коммуникаций библиотеки MPI.

**ЗАДАНИЕ**

1. Написать параллельную программу на языке C/C++ с использованием MPI, реализующую клеточный автомат игры "Жизнь" с завершением программы по повтору состояния клеточного массива в случае одномерной декомпозиции массива по строкам и с циклическими границами массива. Проверить корректность исполнения алгоритма на различном числе процессорных ядер и различных размерах клеточного массива, сравнив с результатами, полученными для исходных данных вручную.
2. Измерить время работы программы при использовании различного числа процессорных ядер: 1, 2, 4, 8, 16. Размеры клеточного массива X и Y подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд. Построить графики зависимости времени работы, ускорения и эффективности распараллеливания от числа используемых ядер.
3. Произвести профилирование программы и выполнить ее оптимизацию. Попытаться достичь 50-процентной эффективности параллельной реализации на 16 ядрах для выбранных X и Y.

****

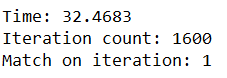
**ОПИСАНИЕ РАБОТЫ**

1. На языке программирования С++ была написана параллельная программа, в которой использовались неблокирующие коммуникации для реализации игры «Жизнь». Время выполнения было замерено с помощью функции MPI\_Wtime();
2. Был написан скрипт с использованием SLURM, с помощью которого программа компилировалась и запускалась:

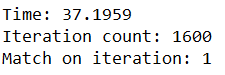


1. Минимальные выходные данные параллельной программы:

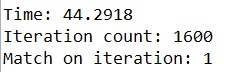
* На 1 процессе:



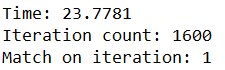
* На 2 процессах:



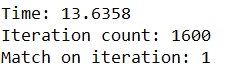
* На 4 процессах:



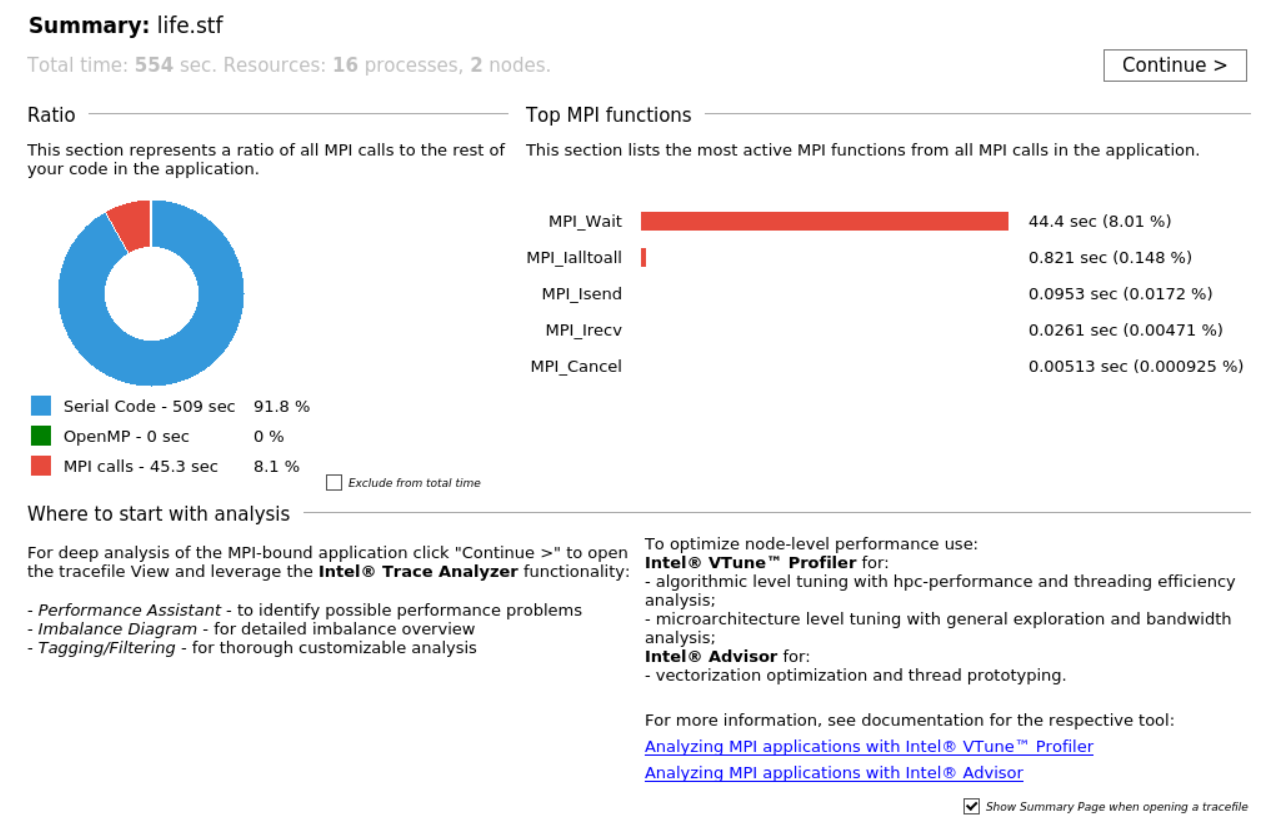
* На 8 процессах:

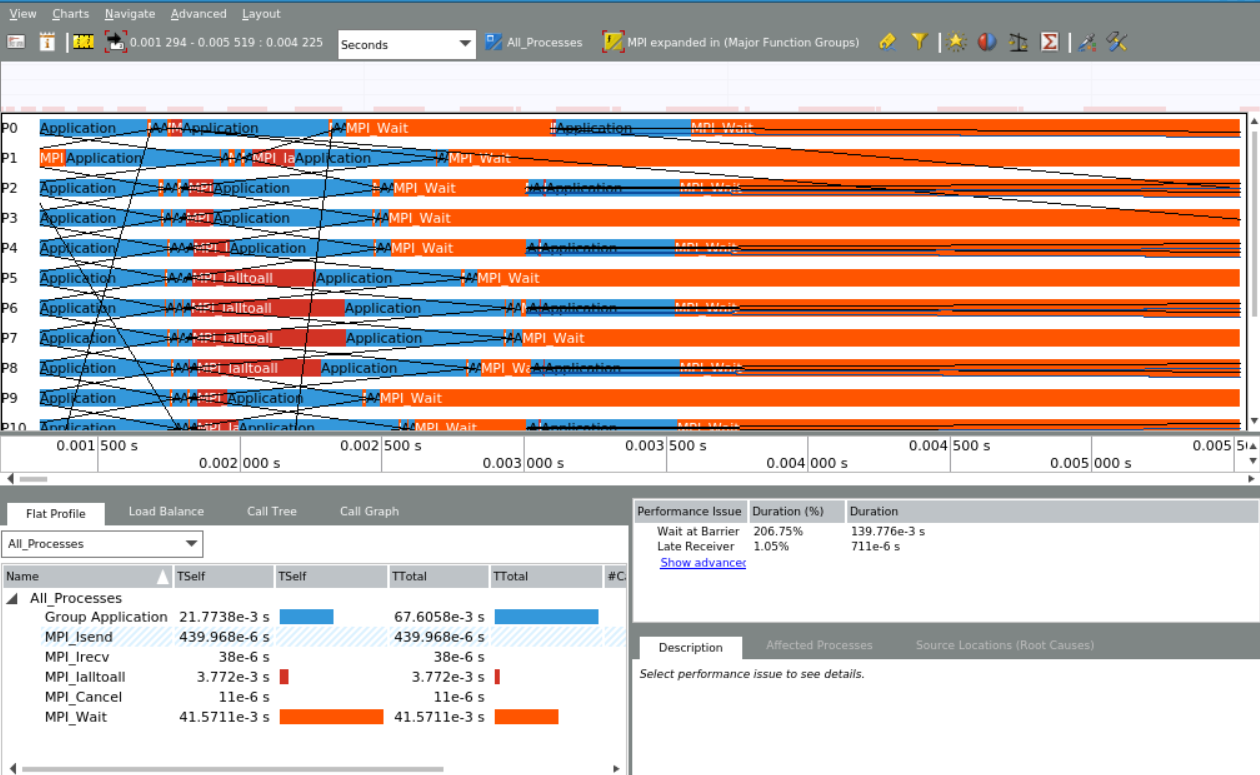


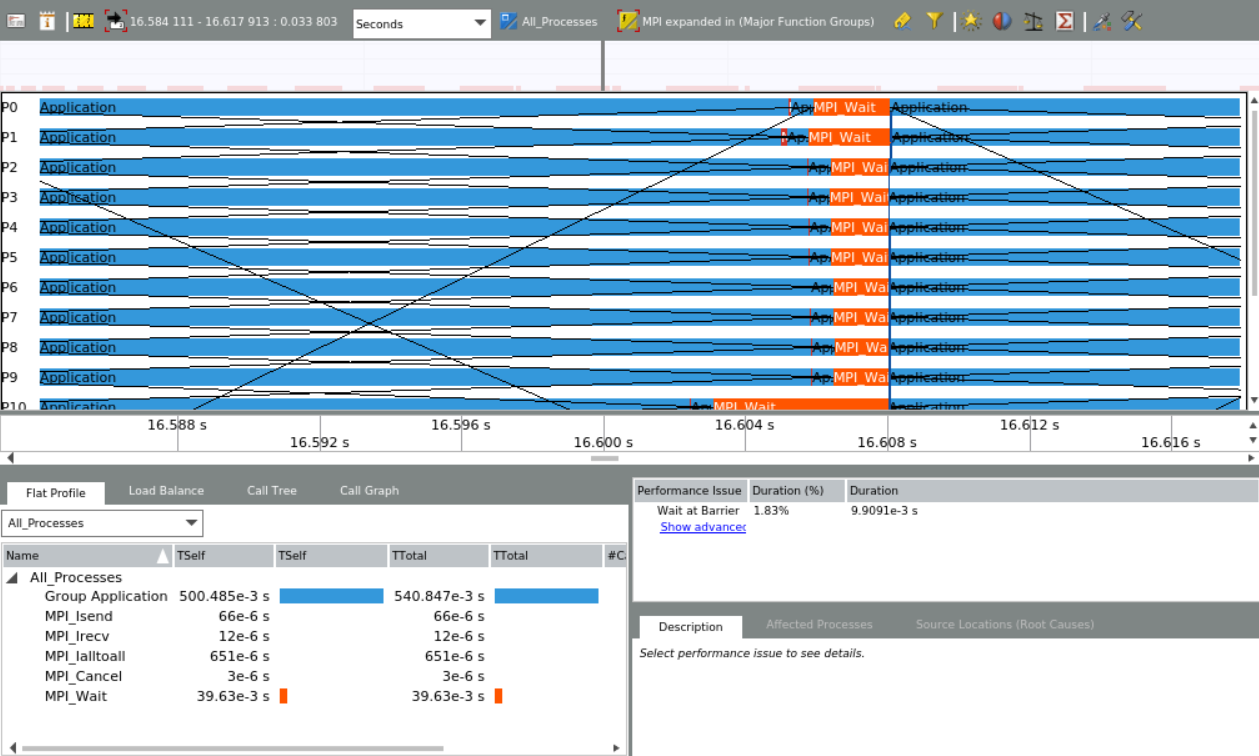
* На 16 процессах:



1. Было выполнено профилирование программы на 16 процессах:



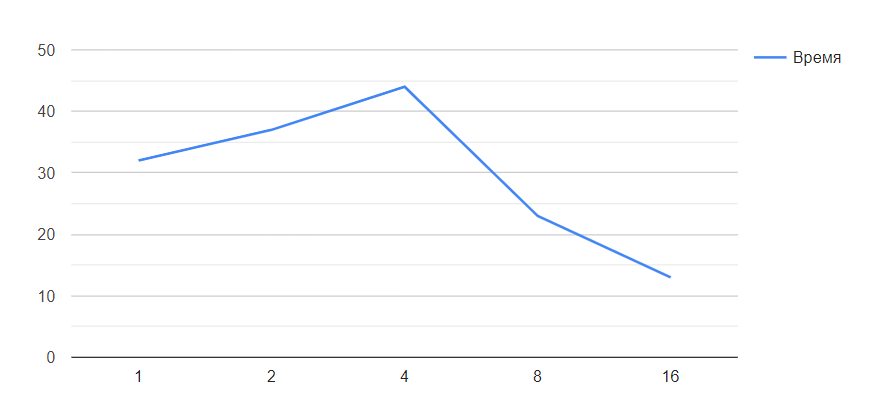


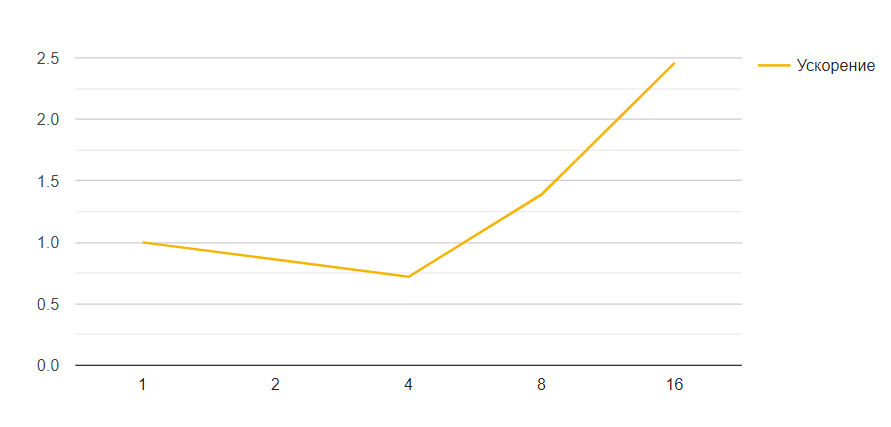


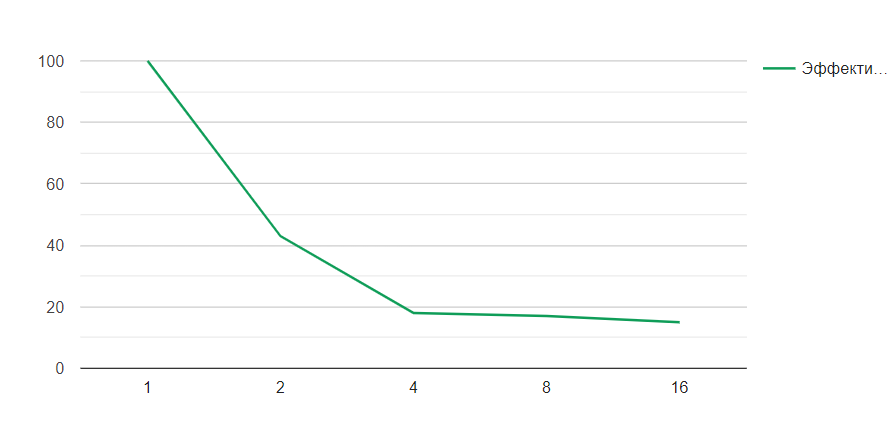
1. Было выполнено построение графиков времени, ускорения, эффективности и график параметров schedule, где:

**Ускорение**: Sp = T1 / Tp, где T1 - время работы ПОСЛЕДОВАТЕЛЬНОЙ программы. Tp - время работы параллельной программы на p процессах/потоках.

**Эффективность**: Ep = Sp / p \* 100%.







**ЗАКЛЮЧЕНИЕ**

Были освоены методы реализации алгоритмов параллелизма с использованием неблокирующих коммуникаций библиотеки MPI. По результатам профилирования можно сделать вывод, что, несмотря на использование неблокирующих коммуникаций, большая часть времени программы уходит на MPI\_Wait.

1. **Приложение 1.** Полный листинг параллельной программы на C++.

#**include** <iostream>

#**include** <mpi.h>

const int MAX\_ITERATIONS\_COUNT = 5000;

const int N = 400;

const bool ALIVE = true, DEAD = false;

void **initializeCells**(bool\* cells) {

cells[0 \* N + 1] = cells[1 \* N + 2] = cells[2 \* N + 0] = cells[2 \* N + 1] = cells[2 \* N + 2] = ALIVE;

/\*for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

cells[i \* N + j] = (rand() % 2) == 1;\*/

}

void **setMatrixParts**(int\* linesPerProc, int\* sendCounts, int\* displs, int\* offsets, int size){

int lines = N / size, rest = N % size, offset = 0;

**for** (int i = 0; i < size; i++) {

rest ? (linesPerProc[i] = lines + 1, rest--) : linesPerProc[i] = lines;

offsets[i] = offset;

displs[i] = offset \* N;

sendCounts[i] = linesPerProc[i] \* N;

offset += linesPerProc[i];

}

}

int **getIndex**(int index, int size) {

**if** (index < 0)

**return** size - 1;

**else** **if** (index >= size)

**return** 0;

**else**

**return** index;

}

int **getLiveNeighborsAmount**(const bool\* prev\_cells, int i, int j) {

int live\_neighbors = 0;

**for** (int y = -1; y <= 1; y++) {

int ni = getIndex(i + y, N + 2);

**for** (int x = -1; x <= 1; x++) {

**if** (x == 0 && y == 0)

**continue**;

int nj = getIndex(j + x, N);

**if** (prev\_cells[ni \* N + nj])

live\_neighbors++;

}

}

**return** live\_neighbors;

}

bool **updateCell**(const bool\* prev\_cells, int i, int j){

int live\_neighbors = getLiveNeighborsAmount(prev\_cells, i, j);

**if** (prev\_cells[i \* N + j])

**if** (live\_neighbors < 2 || live\_neighbors > 3)

**return** DEAD;

**else**

**return** ALIVE;

**else**

**if** (live\_neighbors == 3)

**return** ALIVE;

**else**

**return** DEAD;

}

void **updateInnerCells**(bool\* part, const bool\* prev\_cells, int linesProc) {

int end = linesProc + 2;

**for** (int i = 2; i < end; i++)

**for** (int j = 0; j < N; j++)

part[i \* N + j] = updateCell(prev\_cells, i, j);

}

void **updateUpOrDownCells**(bool\* part, const bool\* prev\_cells, int i) {

**for** (int j = 0; j < N; j++)

part[i \* N + j] = updateCell(prev\_cells, i, j);

}

bool **isEqual**(const bool\* part, const bool\* prev\_cells, int part\_size){

**for**(int i = N; i < part\_size - N; i++)

**if** (part[i] != prev\_cells[i])

**return** false;

**return** true;

}

void **getVectorOfFlags**(bool\*\* generations, const bool\* part, bool\* process\_flags, int iter, int num\_procs, int part\_size) {

**for** (int i = 0; i < iter; i++)

process\_flags[i] = isEqual(part, generations[i], part\_size);

**for** (int i = 1; i < num\_procs; i++)

**for** (int j = 0; j < iter; j++)

process\_flags[i \* iter + j] = process\_flags[0 \* iter + j];

}

int **checkFlags**(const bool\* flags, int sizeOfRow, int num\_procs) {

**if** (num\_procs == 1) {

**for** (int column = 0; column < sizeOfRow; column++)

**if** (flags[column])

**return** ++column;

}

**else**

**for**(int column = 0; column < sizeOfRow; column++)

**for** (int row = 1; row < num\_procs; row++) {

**if** (flags[column] != flags[row \* sizeOfRow + column])

**break**;

**else**

**if** (row == num\_procs - 1 && flags[column])

**return** ++column;

}

**return** -1;

}

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

srand(time(nullptr));

int rank, size;

double start, end;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

bool\* cells = nullptr;

bool\* part;

bool\*\* generations;

int\* linesPerProc;

int\* offsets;

int\* sendCounts;

int\* displs;

linesPerProc = **new** int[size];

offsets = **new** int[size];

sendCounts = **new** int[size];

displs = **new** int[size];

setMatrixParts(linesPerProc, sendCounts, displs, offsets, size);

int part\_size = sendCounts[rank] + 2 \* N;

part = **new** bool[part\_size];

**if** (!rank) {

cells = **new** bool[N \* N]();

initializeCells(cells);

}

MPI\_Scatterv(cells, sendCounts, displs, MPI\_C\_BOOL,

&part[N], sendCounts[rank], MPI\_C\_BOOL, 0, MPI\_COMM\_WORLD);

**if** (!rank) **delete**[] cells;

**delete**[] offsets;

**delete**[] sendCounts;

**delete**[] displs;

generations = **new** bool\*[MAX\_ITERATIONS\_COUNT]{nullptr};

int iter\_equal, iter\_count;

start = MPI\_Wtime();

**for** (iter\_count = 1; iter\_count < MAX\_ITERATIONS\_COUNT; iter\_count++) {

generations[iter\_count - 1] = part;

//if (!rank) printf("iter = %d\n", iter\_count);

MPI\_Request send\_request[3];

MPI\_Request recv\_request[2];

MPI\_Isend(&part[N], N, MPI\_C\_BOOL, (rank - 1 + size) % size, 0, MPI\_COMM\_WORLD, &send\_request[0]);

MPI\_Isend(&part[part\_size - 2 \* N], N, MPI\_C\_BOOL, (rank + 1) % size, 1, MPI\_COMM\_WORLD, &send\_request[1]);

part = **new** bool[part\_size];

MPI\_Irecv(&generations[iter\_count - 1][0], N, MPI\_C\_BOOL, (rank - 1 + size) % size, 1, MPI\_COMM\_WORLD, &recv\_request[0]);

MPI\_Irecv(&generations[iter\_count - 1][part\_size - N], N, MPI\_C\_BOOL, (rank + 1) % size, 0, MPI\_COMM\_WORLD, &recv\_request[1]);

bool process\_flags[size \* (iter\_count - 1)];

bool all\_flags[size \* (iter\_count - 1)];

**if** (iter\_count > 1) {

getVectorOfFlags(generations, generations[iter\_count - 1], process\_flags, iter\_count - 1, size, part\_size);

MPI\_Ialltoall(process\_flags, iter\_count - 1, MPI\_C\_BOOL,

all\_flags, iter\_count - 1, MPI\_C\_BOOL, MPI\_COMM\_WORLD, &send\_request[2]);

}

// inner lines

updateInnerCells(part, generations[iter\_count - 1], linesPerProc[rank] - 2);

// first line

MPI\_Wait(&recv\_request[0], MPI\_STATUS\_IGNORE);

updateUpOrDownCells(part, generations[iter\_count - 1], 1);

// last line

MPI\_Wait(&recv\_request[1], MPI\_STATUS\_IGNORE);

updateUpOrDownCells(part, generations[iter\_count - 1], part\_size / N - 2);

MPI\_Cancel(&send\_request[0]);

MPI\_Cancel(&send\_request[1]);

**if** (iter\_count > 1) {

MPI\_Wait(&send\_request[2], MPI\_STATUS\_IGNORE);

iter\_equal = checkFlags(all\_flags, iter\_count - 1, size);

**if** (iter\_equal != -1) {

iter\_count--;

**break**;

}

}

}

**if** (!rank) {

end = MPI\_Wtime();

std::cout << "Time: " << end - start << std::endl;

std::cout << "Iteration count: " << iter\_count << std::endl;

std::cout << "Match on iteration: " << iter\_equal << std::endl;

}

**delete**[] linesPerProc;

**delete**[] part;

**for** (int i = 0; generations[i] != nullptr; i++)

**delete**[] generations[i];

**delete**[] generations;

MPI\_Finalize();

**return** 0;

}

1. **Приложение 2.** Полный листинг скрипта SLURM.

#!/bin/bash

#SBATCH -J lab5 # Job name

#SBATCH -p compclass # Queue name (or "compclass\_unstable", or "gpuserv", or "a100serv")

#SBATCH -o lab5.%j.out # Name of stdout output file (%j expands to %jobId)

#SBATCH -N 2 # Total number of nodes requested

#SBATCH -n 16 # Total number of mpi tasks requested

#SBATCH -t 00:10:00 # Run time (hh:mm:ss)

module load mpi/mpich-x86\_64

**if** [ "$#" -eq 0 ]; **then**

echo "Usage: $0 <num\_processes>"

exit 1

**fi**

mpicxx life.cpp -o life

echo -e "SIZE = $1\n"

mpirun -np $1 ./life