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»

студента 2 курса, группы 21206

Мельникова Никиты Сергеевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
к.т.н., доцент
А.Ю. Власенко

Новосибирск 2023

СОДЕРЖАНИЕ

ЦЕЛЬ

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. Была написана (листинг 1) параллельная программа на языке 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. Была проверена корректность алгоритма на различном числе процессорных ядер и различных размерах клеточного массива. В качестве исходных данных на поле заданного размера помещается парусник (глайдер), который каждую четвертую итерацию приходит в исходную форму, смещенную на одну клетку по диагонали вправо-вниз.
2. Было измерено время работы программы на кластере НГУ при использовании различного числа процессорных ядер: 1, 2, 4, 8, 16, 24. Размеры клеточного массива подобраны таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд (X = 600, Y = 600). Были построены графики ускорения и эффективности, 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%

Количество процессов	Время, с.	Ускорение	Эффективность, %
1	73,71633	1	100
2	62,516091	1,179157699	58,95788494
4	38,488893	1,915262411	47,88156027
8	31,960742	2,306464912	28,83081141
16	16,596669	4,441634041	27,76021276
24	16,093481	4,580508717	19,08545299

График времени

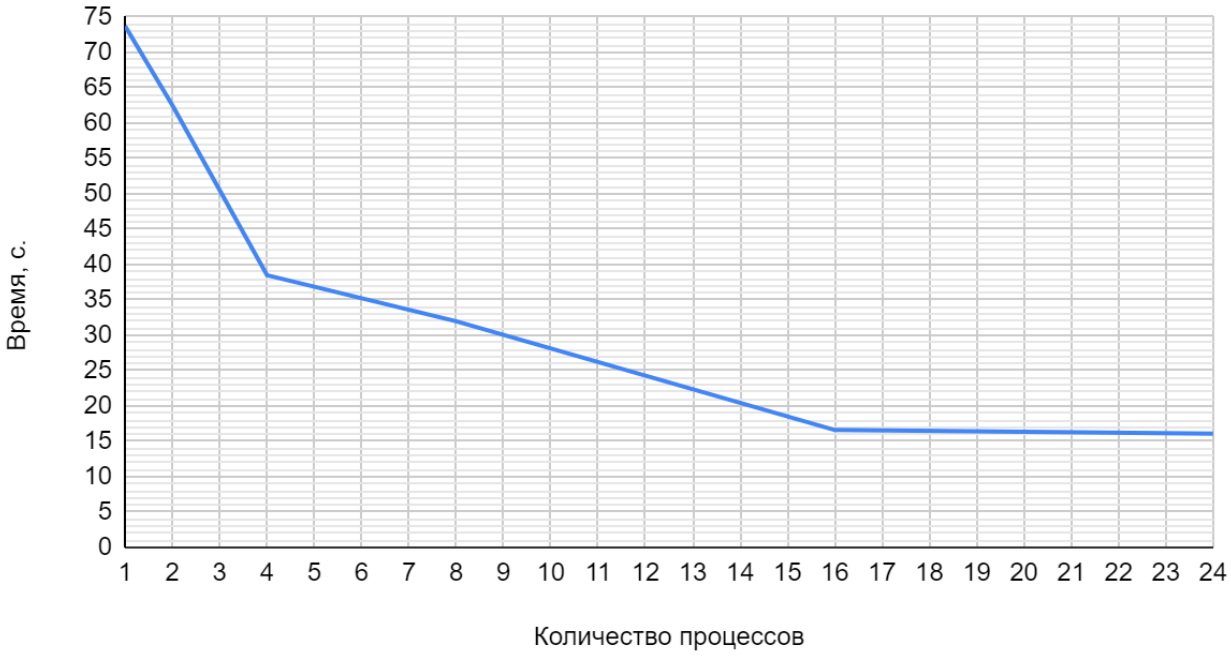
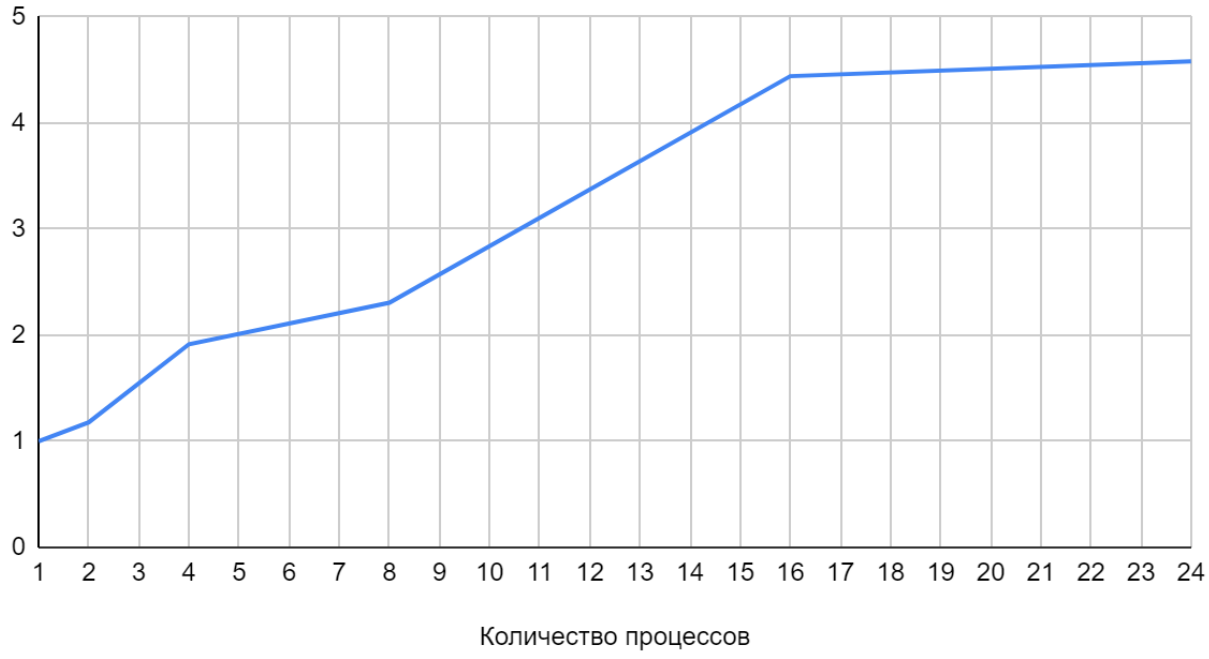
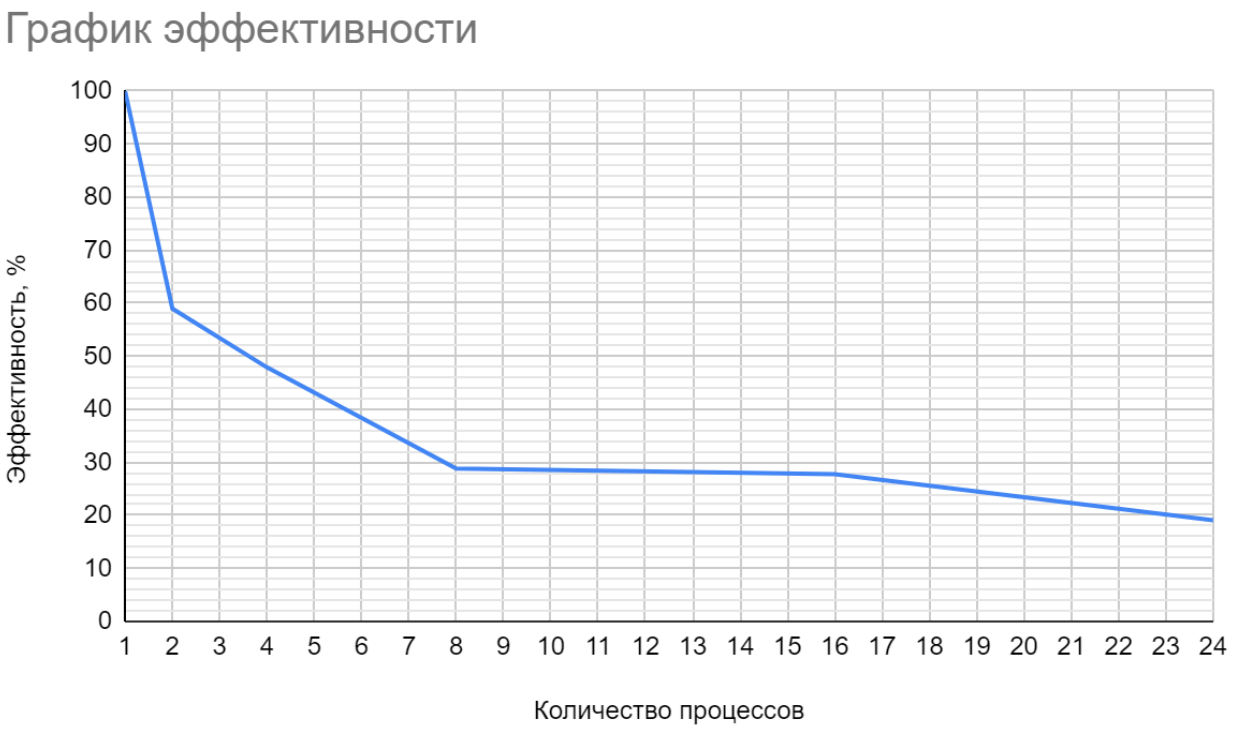


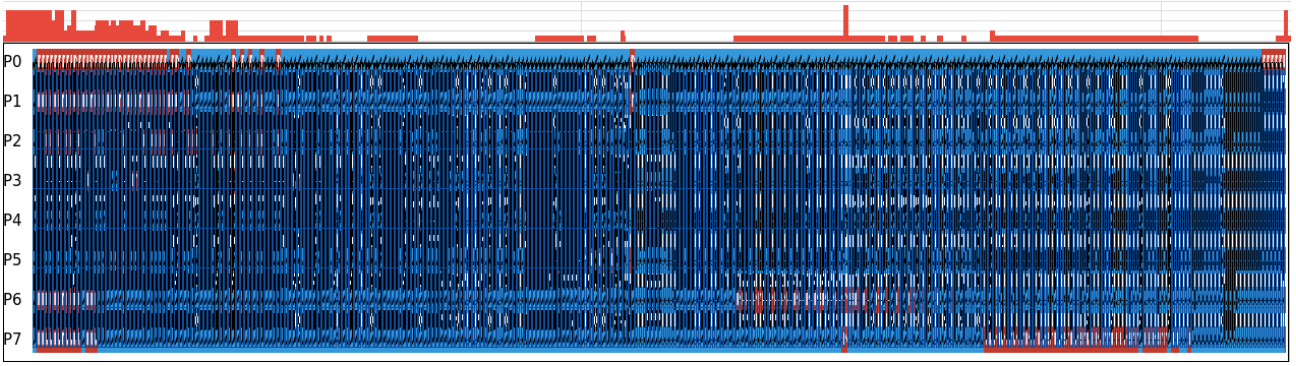
График ускорения

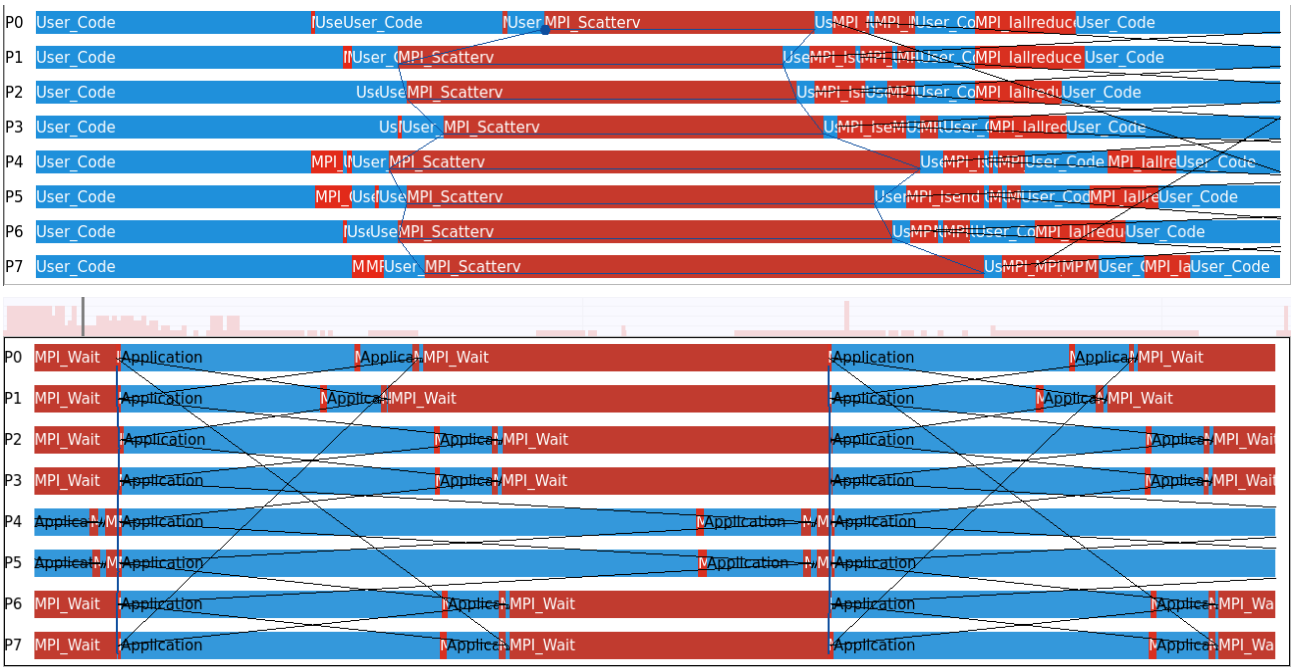




3. Было выполнено профилирование программы при использовании 8-и ядер средствами Intel Trace Analyzer and Collector (ITAC).

Name	TSelf	TSelf	TTotal	TTotal	#Calls
All_Processes					
User_Code	13.0291 s		17.9548 s		8
MPI_Comm_size	8e-6 s		8e-6 s		8
MPI_Comm_rank	21e-6 s		21e-6 s		8
MPI_Finalize	1.895e-3 s		1.895e-3 s		8
MPI_Iallreduce	154.241e-3 s		154.241e-3 s		9600
MPI_Isend	33.839e-3 s		33.839e-3 s		19200
MPI_Irecv	29.152e-3 s		29.152e-3 s		19200
MPI_Wtime	9e-6 s		9e-6 s		2
MPI_Scatterv	759e-6 s		759e-6 s		8
MPI_Wait	4.7058 s		4.7058 s		48000




P0
P1
P2
P3
P4
P5
P6
P7
User_Code
MPI_Scatterv
MPI_Iallreduce
MPI_Wait
Application

ЗАКЛЮЧЕНИЕ

В ходе практической работы удалось ознакомиться с концепцией неблокирующих коммуникаций библиотеки MPI. Были освоены методы реализации алгоритмов параллелизма с их использованием.

Приложение 1. *Листинг параллельной программы на языке Си*

Код компиляции: `mpicxx life.c -o life`

Листинг 1 — Исходный код программы

```
#include <mpi.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define X        600
#define Y        600
#define MAX_ITER 2400
#define TAG1     123
#define TAG2     321

bool *gen_field() {
    bool *field = (bool *) calloc(X * Y, sizeof(bool));

    field[0 * Y + 1] = true;
    field[1 * Y + 2] = true;
    for (int i = 0; i < 3; ++i)
        field[2 * Y + i] = true;

    return field;
}

void set_matrix_part(int *sendcounts, int *displs, int size) {
    int offset = 0;
    int linecounts[size];
    int offsets[size];
    for (int i = 0; i < size; ++i) {
        linecounts[i] = X / size;
        if (i < X % size)
            ++linecounts[i];

        offsets[i] = offset;
        offset += linecounts[i];
        sendcounts[i] = linecounts[i] * Y;
        displs[i] = offsets[i] * Y;
    }
}

void compare_with_prev(bool *field_part, bool **prev_field_parts, int len,
                       bool *stop_flags, size_t iter) {
```

```
    for (size_t i = 0; i < iter; ++i) {
        if (!memcmp(field_part, prev_field_parts[i], len)) {
            stop_flags[i] = true;
        } else {
            stop_flags[i] = false;
        }
    }
}

void update_field(bool *field, bool *next_gen_field, int len) {
    for (int i = 0; i < len / Y; ++i) {
        for (int j = 0; j < Y; ++j) {
            int num_of_neighbours = field[(i - 1) * Y + (Y + j - 1) % Y] +
                                    field[(i - 1) * Y + j] +
                                    field[(i - 1) * Y + (j + 1) % Y] +
                                    field[i * Y + (Y + j - 1) % Y] +
                                    field[i * Y + (j + 1) % Y] +
                                    field[(i + 1) * Y + (Y + j - 1) % Y] +
                                    field[(i + 1) * Y + j] +
                                    field[(i + 1) * Y + (j + 1) % Y];

            if (field[i * Y + j] == false) {
                next_gen_field[i * Y + j] =
                    (num_of_neighbours == 3) ? true: false;
            } else if (field[i * Y + j] == true) {
                next_gen_field[i * Y + j] =
                    (num_of_neighbours < 2 || num_of_neighbours > 3) ? false
                                                                     : true;
            }
        }
    }
}

bool check_flags(bool *stop_flags_reduced, size_t iter) {
    for (size_t i = 0; i < iter; ++i)
        if (stop_flags_reduced[i]) return true;
    return false;
}

int main(int argc, char *argv[]) {
    int rank    = 0;
    int size    = 0;
    size_t iter = 0;
    double time_start, time_end;
    bool *field, *field_part, *next_gen_part;
    bool **prev_field_parts;
```

```
int *sendcounts, *displs;
bool *stop_flags, *stop_flags_reduced;
int prev_proc_rank, next_proc_rank;
MPI_Request requests[5];

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

if (rank == 0) {
    field = gen_field();
    time_start = MPI_Wtime();
}

sendcounts = (int *) malloc(size * sizeof(int));
displs     = (int *) malloc(size * sizeof(int));
set_matrix_part(sendcounts, displs, size);

field_part    = (bool *) malloc(sendcounts[rank] + 2 * Y *
                                sizeof(bool));
next_gen_part = (bool *) malloc(sendcounts[rank] + 2 * Y *
                                sizeof(bool));
prev_field_parts = (bool **) malloc(MAX_ITER * sizeof(bool *));
stop_flags         = (bool *) malloc(MAX_ITER * sizeof(bool));
stop_flags_reduced = (bool *) malloc(MAX_ITER * sizeof(bool));

MPI_Scatterv(field, sendcounts, displs,
             MPI_C_BOOL, field_part + Y, sendcounts[rank],
             MPI_C_BOOL,
             0, MPI_COMM_WORLD);

prev_proc_rank = (size + (rank - 1)) % size;
next_proc_rank = (rank + 1) % size;

while (iter < MAX_ITER) {
    MPI_Isend(field_part + Y, Y, MPI_C_BOOL,
              prev_proc_rank, TAG1, MPI_COMM_WORLD, &requests[0]);
    MPI_Isend(field_part + sendcounts[rank], Y, MPI_C_BOOL,
              next_proc_rank, TAG2, MPI_COMM_WORLD, &requests[1]);
    MPI_Irecv(field_part, Y, MPI_C_BOOL,
              prev_proc_rank, TAG2, MPI_COMM_WORLD, &requests[2]);
    MPI_Irecv(field_part + sendcounts[rank] + Y, Y, MPI_C_BOOL,
              next_proc_rank, TAG1, MPI_COMM_WORLD, &requests[3]);

    prev_field_parts[iter] = (bool *) malloc(sendcounts[rank] *
                                             sizeof(bool));
```

```
            memcpy(prev_field_parts[iter], field_part + Y,
                   sendcounts[rank] * sizeof(bool));
            compare_with_prev(field_part + Y, prev_field_parts,
                              sendcounts[rank], stop_flags, iter);

            MPI_Iallreduce(stop_flags, stop_flags_reduced, iter + 1,
                           MPI_C_BOOL, MPI_LAND, MPI_COMM_WORLD, &requests[4]);

            update_field(field_part + 2 * Y, next_gen_part + 2 * Y,
                         sendcounts[rank] - 2 * Y);

            MPI_Wait(&requests[0], MPI_STATUS_IGNORE);
            MPI_Wait(&requests[2], MPI_STATUS_IGNORE);
            update_field(field_part + Y, next_gen_part + Y, Y);

            MPI_Wait(&requests[1], MPI_STATUS_IGNORE);
            MPI_Wait(&requests[3], MPI_STATUS_IGNORE);
            update_field(field_part + sendcounts[rank],
                         next_gen_part + sendcounts[rank], Y);

            MPI_Wait(&requests[4], MPI_STATUS_IGNORE);
            if (check_flags(stop_flags_reduced, iter)) break;

            bool *tmp = field_part;
            field_part = next_gen_part;
            next_gen_part = tmp;
            ++iter;
        }

        if (rank == 0) {
            time_end = MPI_Wtime();
            printf("Time taken: %f sec.\n", time_end - time_start);
            free(field);
        }
        for (size_t i = 0; i < iter; ++i)
            free(prev_field_parts[i]);
        free(prev_field_parts);
        free(sendcounts);
        free(displs);
        free(field_part);
        free(next_gen_part);
        free(stop_flags);
        free(stop_flags_reduced);
        MPI_Finalize();
        return EXIT_SUCCESS;
    }
```

Листинг 2 — Скрипт для запуска на кластере НГУ

```
#!/bin/bash

#PBS -l walltime=00:02:00
#PBS -l select=1:ncpus=8:mpiprocs=8:mem=8000m,place=free
#PBS -m n

cd $PBS_O_WORKDIR

MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')
echo "Number of MPI process: $MPI_NP"

mpirun -machinefile $PBS_NODEFILE -np $MPI_NP ./life
```