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

«Два вектора»

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

Сравнение времени работы последовательной и параллельной программы.

ЗАДАНИЕ

I Написать 3 программы, каждая из которых рассчитывает число s по двум данным векторам a и b равной длины N в соответствии со следующим двойным циклом:

```
for (i = 0; i < N; i++)
        for(j = 0; j < N; j++)
                s += a[i] * b[j];
```

a) последовательная программа

b) параллельная, использующая коммуникации типа точка-точка (MPI_Send, MPI_Recv)

c) параллельная, использующая коллективные коммуникации (MPI_Scatter, MPI_Reduce, MPI_Bcast)

II Замерить время работы последовательной программы и параллельных на 2, 4, 8, 16, 24 процессах. Рекомендуется провести несколько замеров для каждого варианта запуска и выбрать минимальное время.

III Построить графики времени, ускорения и эффективности.

IV Составить отчет, содержащий исходные коды разработанных программ и построенные графики.

ОПИСАНИЕ РАБОТЫ

1. Было написано три программы, вычисляющих число s, в соответствии с заданием.
2. Было замерено время работы последовательной программы. Длина векторов была выбрана таким образом, чтобы время работы последовательной программы было не менее 30 сек.

```
opp@comrade:~/206/Melnikov/lab1$ ./lab1_a
s = 18050000000
Time taken: 31.388852 sec.
```

3. Было замерено время работы параллельных программ на 2, 4, 8, 16, 24 процессах.
   - использующих коммуникации типа точка-точка

```
opp@comrade:~/206/Melnikov/lab1$ mpiexec -n 2 ./lab1_b
s = 18050000000
Time taken: 15.699460 sec.
```

```
opp@comrade:~/206/Melnikov/lab1$ mpiexec -n 4 ./lab1_b
s = 18050000000
Time taken: 8.185685 sec.
```

```
opp@comrade:~/206/Melnikov/lab1$ mpiexec -n 8 ./lab1_b
s = 18050000000
Time taken: 5.297014 sec.
```

```
opp@comrade:~/206/Melnikov/lab1$ mpiexec -n 16 ./lab1_b
s = 18050000000
Time taken: 3.629412 sec.
```

```
opp@comrade:~/206/Melnikov/lab1$ mpiexec -n 24 ./lab1_b
s = 18050000000
Time taken: 2.804205 sec.
```

   - использующих коллективные коммуникации

```
opp@comrade:~/206/Melnikov/lab1$ mpiexec -n 2 ./lab1_c
s = 18050000000
Time taken: 15.704814 sec.
```

```
opp@comrade:~/206/Melnikov/lab1$ mpiexec -n 4 ./lab1_c
s = 18050000000
Time taken: 8.183710 sec.
```

```
opp@comrade:~/206/Melnikov/lab1$ mpiexec -n 8 ./lab1_c
s = 18050000000
Time taken: 4.094704 sec.
```

```
opp@comrade:~/206/Melnikov/lab1$ mpiexec -n 16 ./lab1_c
s = 18050000000
Time taken: 3.629044 sec.
```

```
opp@comrade:~/206/Melnikov/lab1$ mpiexec -n 24 ./lab1_c
s = 18050000000
Time taken: 2.799820 sec.
```

4. 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%

График времени

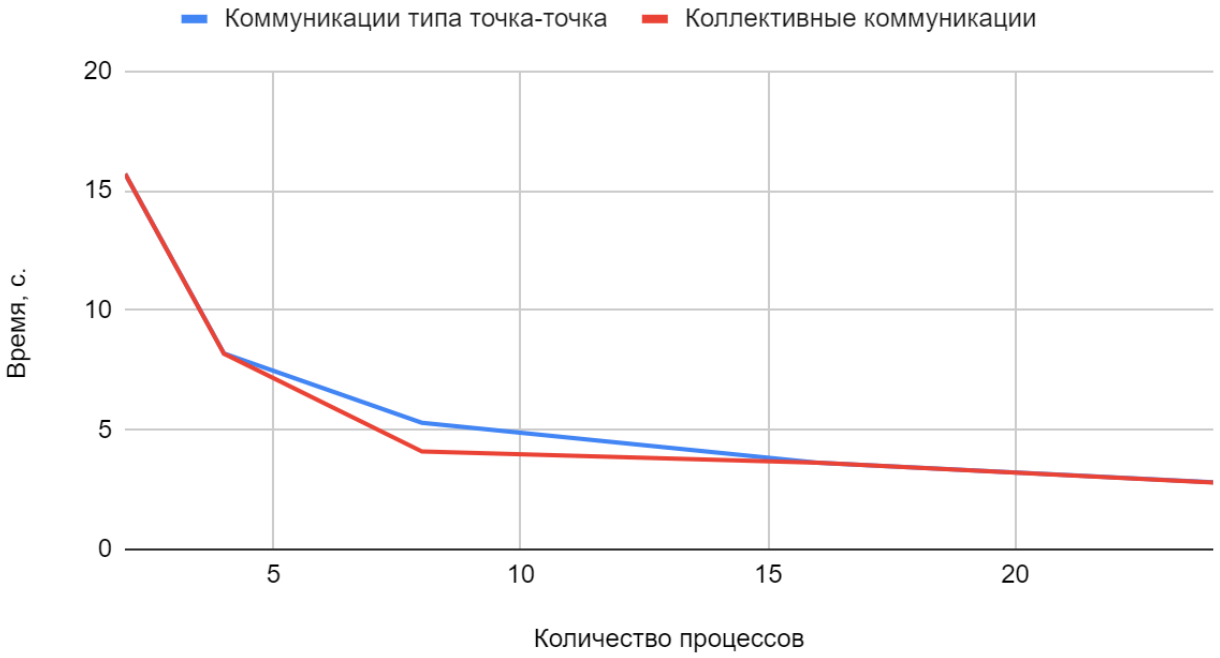


График ускорения

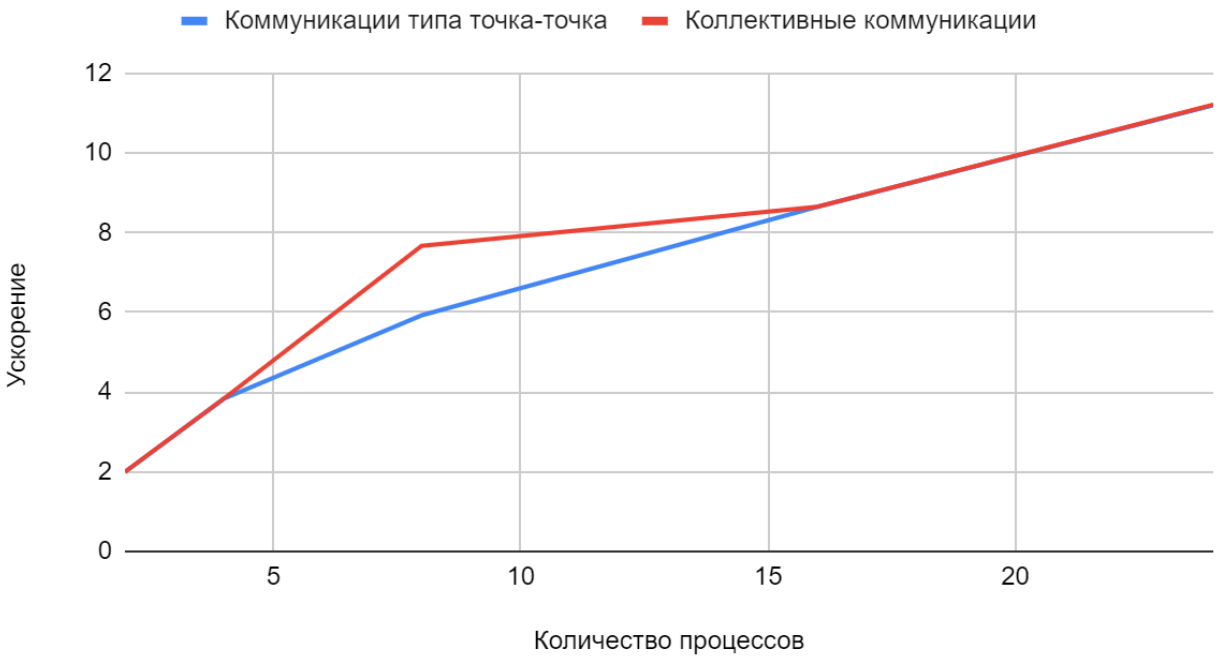
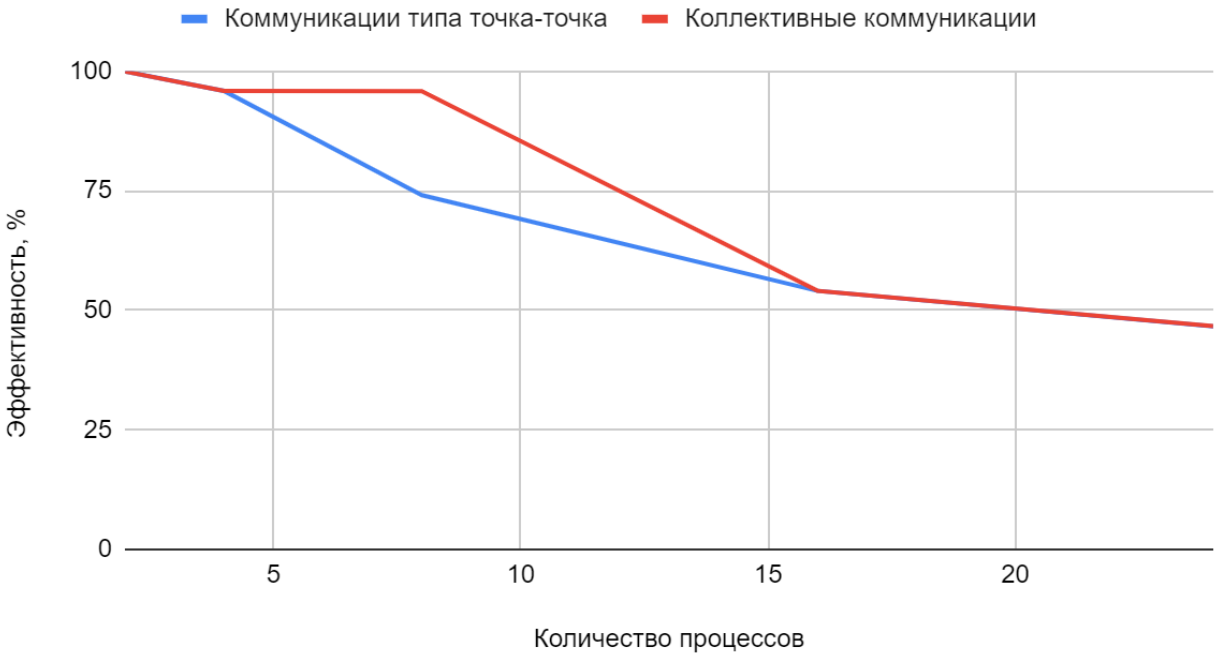


График эффективности



ЗАКЛЮЧЕНИЕ

В ходе практической работы удалось выявить существенное существенное повышение производительности при использовании параллельных программ по сравнению с последовательными. При увеличении количества процессов время работы программы уменьшается.

Приложение 1. *Листинг последовательной программы на языке Си*

Листинг 1 — Исходный код программы

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define VECTOR_SIZE           95000
#define NUMBER_OF_MEASURMENTS 5

void init_vectors(int *vector1, int *vector2) {
    for (size_t i = 0; i < VECTOR_SIZE; ++i) {
        vector1[i] = 1;
        vector2[i] = 2;
    }
}

double calc_time(int *vector1, int *vector2) {
    double time_start = 0;
    double time_end   = 0;
    double min_time   = ULLONG_MAX;
    long long int s   = 0;

    for (size_t k = 0; k < NUMBER_OF_MEASURMENTS; ++k) {
        s = 0;
        time_start = MPI_Wtime();
        for (size_t i = 0; i < VECTOR_SIZE; ++i) {
            for (size_t j = 0; j < VECTOR_SIZE; ++j)
                s += vector1[i] * vector2[j];
        }
        time_end   = MPI_Wtime();
             if (time_end - time_start < min_time) min_time = time_end -
time_start;
    }

    printf("s = %lld\n", s);
    return min_time;
}

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int *vector1 = malloc(VECTOR_SIZE * sizeof(int));
    int *vector2 = malloc(VECTOR_SIZE * sizeof(int));
```

```
    init_vectors(vector1, vector2);

    printf("Time taken: %f sec.\n", calc_time(vector1, vector2));

    free(vector1);
    free(vector2);
    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

Приложение 2. *Листинг параллельной программы, использующей коммуникации типа точка-точка на языке Си*

Листинг 2 — Исходный код программы

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define VECTOR_SIZE           95000
#define NUMBER_OF_MEASURMENTS 5
#define TAG                   123

void init_vectors(int *vector1, int *vector2) {
    for (size_t i = 0; i < VECTOR_SIZE; ++i) {
        vector1[i] = 1;
        vector2[i] = 2;
    }
}

long long mult(int *vector1, int *vector2, size_t size) {
    long long s = 0;
    for (size_t i = 0; i < size; ++i) {
        for (size_t j = 0; j < VECTOR_SIZE; ++j)
            s += vector1[i] * vector2[j];
    }
    return s;
}

int main(int argc, char *argv[]) {
    int rank = 0;
    int size = 0;
    long long int s = 0;

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

     size_t part_size = (VECTOR_SIZE % size == 0) ? VECTOR_SIZE / size :
VECTOR_SIZE / size + 1;

    int *vector1, *vector2;

    if (rank == 0) {
        double time_start = 0;
        double time_end   = 0;
```

```
        double min_time   = ULLONG_MAX;
        long long int tmp = 0;

        vector1 = malloc(VECTOR_SIZE * sizeof(int));
        vector2 = malloc(VECTOR_SIZE * sizeof(int));

        init_vectors(vector1, vector2);

        for (size_t k = 0; k < NUMBER_OF_MEASURMENTS; ++k) {
            s = 0;
            time_start = MPI_Wtime();
            for (int i = 1; i < size; ++i) {
                    MPI_Send(vector1 + i * part_size, part_size, MPI_INT, i,
TAG, MPI_COMM_WORLD);
                            MPI_Send(vector2, VECTOR_SIZE, MPI_INT, i, TAG,
MPI_COMM_WORLD);
            }
            s = mult(vector1, vector2, part_size);
            for (size_t l = 1; l < size; ++l) {
                 MPI_Recv(&tmp, 1, MPI_LONG_LONG_INT, l, TAG, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
                s += tmp;
            }
            time_end   = MPI_Wtime();
               if (time_end - time_start < min_time) min_time = time_end -
time_start;
        }
        printf("s = %lld\n", s);
        printf("Time taken: %f sec.\n", min_time);
    } else {
        vector1 = malloc(part_size * sizeof(int));
        vector2 = malloc(VECTOR_SIZE * sizeof(int));
        for (size_t k = 0; k < NUMBER_OF_MEASURMENTS; ++k) {
            s = 0;
               MPI_Recv(vector1, part_size, MPI_INT, 0, TAG, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
             MPI_Recv(vector2, VECTOR_SIZE, MPI_INT, 0, TAG, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            s += mult(vector1, vector2, part_size);
            MPI_Send(&s, 1, MPI_LONG_LONG_INT, 0, TAG, MPI_COMM_WORLD);
        }
    }

    free(vector1);
    free(vector2);
    MPI_Finalize();
```

```
    return EXIT_SUCCESS;
}
```

Приложение 3. *Листинг параллельной программы, использующей коллективные коммуникации на языке Си*

Листинг 3 — Исходный код программы

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define VECTOR_SIZE           95000
#define NUMBER_OF_MEASURMENTS 5

void init_vectors(int *vector1, int *vector2) {
    for (size_t i = 0; i < VECTOR_SIZE; ++i) {
        vector1[i] = 1;
        vector2[i] = 2;
    }
}

long long mult(int *vector1, int *vector2, size_t size) {
    long long s = 0;
    for (size_t i = 0; i < size; ++i) {
        for (size_t j = 0; j < VECTOR_SIZE; ++j)
            s += vector1[i] * vector2[j];
    }
    return s;
}

int main(int argc, char *argv[]) {
    int rank = 0;
    int size = 0;
    long long int s   = 0;
    long long int tmp = 0;
    double time_start = 0;
    double time_end   = 0;
    double min_time   = ULLONG_MAX;

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

     size_t part_size = (VECTOR_SIZE % size == 0) ? VECTOR_SIZE / size :
VECTOR_SIZE / size + 1;

    int *vector1;
    int *vector2 = malloc(VECTOR_SIZE * sizeof(int));
```

```
    if (rank == 0) {
        vector1 = malloc(VECTOR_SIZE * sizeof(int));

        init_vectors(vector1, vector2);
    }

    int *part_of_vector1 = malloc(part_size * sizeof(int));

    for (size_t k = 0; k < NUMBER_OF_MEASURMENTS; ++k) {
        tmp = 0;
        time_start = MPI_Wtime();
         MPI_Scatter(vector1, part_size, MPI_INT, part_of_vector1, part_size,
MPI_INT, 0, MPI_COMM_WORLD);
        MPI_Bcast(vector2, VECTOR_SIZE, MPI_INT, 0, MPI_COMM_WORLD);
        tmp = mult(part_of_vector1, vector2, part_size);
        MPI_Reduce(&tmp, &s, 1, MPI_LONG_LONG, MPI_SUM, 0, MPI_COMM_WORLD);
        time_end   = MPI_Wtime();
             if  (time_end  -  time_start  <  min_time)  min_time  =  time_end  -
time_start;
    }

    if (rank == 0) {
        printf("s = %lld\n", s);
        printf("Time taken: %f sec.\n", min_time);
        free(vector1);
    }

    free(part_of_vector1);
    free(vector2);
    MPI_Finalize();
    return EXIT_SUCCESS;
}
```