

**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

Студентки 2 курса, группы 21205

Евдокимовой Дари Евгеньевны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук, доцент
А.Ю. Власенко

Новосибирск 2022

СОДЕРЖАНИЕ

ЦЕЛЬ.....	3
ЗАДАНИЕ.....	3
ОПИСАНИЕ РАБОТЫ.....	4
ЗАКЛЮЧЕНИЕ.....	6
Приложение 1. Листинг последовательной программы.....	7
Приложение 2. Листинг программы с использованием коммуникации «точка-точка».....	9
Приложение 3. Листинг программы при использовании коллективных коммуникаций.....	12
Приложение 4. Проверка корректности работы программы.....	15
Приложение 5. Графики времени, ускорения и эффективности.....	19

ЦЕЛЬ

1. Ознакомление со стандартом MPI.
2. Ознакомление с коммуникацией типа точка-точка.
3. Ознакомление с коллективными коммуникациями.
4. Сравнение времени работы программы при коммуникации типа «точка-точка» с коллективной коммуникацией.

ЗАДАНИЕ

1. 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];
```

2. 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.
3. Построить графики времени, ускорения и эффективности.
4. Составить отчет, содержащий исходные коды разработанных программ и построенные графики.

ОПИСАНИЕ РАБОТЫ

1. Был создан файл *sequential.c*, в котором была реализован последовательная программа и добавлено время измерения программы с помощью функции `clock_gettime()`. Полный компилируемый листинг программы см. Приложение 1. Команда для компиляции:

```
gcc -o sequential sequential.c  
./sequential
```

Размер векторов, при котором время программы составляет не менее 30сек: 122880 ($=512 * 24 * 10$). Число подобрано так, чтобы оно делилось нацело на 2, 4, 8, 16, 24.

2. Был создан файл *point-to-point.c*, в котором реализован типа коммуникации «точка-точка» при помощи методов `MPI_Send` и `MPI_Recv`. И также добавлено время измерения программы с помощью функции `MPI_Wtime()`.

Листинг программы см. в Приложении 2. Команда для компиляции:

```
mpicc -o point-to-point point-to-point.c  
mpiexec -n <n> ./point-to-point
```

3. Был создан файл *collective_commun.c*, в котором были реализованы коллективные коммуникации при помощи методов `MPI_Scatter`, `MPI_Bcast` и `MPI_Reduce`. И также добавлено время измерения программы с помощью функции `MPI_Wtime()`.

Полный компилируемый листинг программы см. Приложение 3. Команда для компиляции:

```
mpicc -o collect collective_commun.c  
mpiexec -n <n> ./collect
```

4. Была проведена проверка корректности вычислений. Размер векторов = $32 * 32$. Программа работает корректно, т.к. на всех типах коммуникаций при разном количестве процессов результат такой же, как при работе последовательной программы. Скриншоты экспериментов см. в Приложении 4.

5. Были проведены замеры для вектора размером 122880. Полученные результаты см. в таблице 1.

Таблица 1. Результаты измерений времени

Количество процессов	Тип работы программы		
	Время работы последовательной программы, с	Время работы программы с коммуникацией «точка-точка», с	Время работы программы с коллективными коммуникациями, с
2	58,102796	29,072807	29,07144
4	58,102796	15,041252	15,039296
8	58,102796	9,501523	7,692684
16	58,102796	6,824421	6,642048
24	58,102796	5,167288	5,065426

6. На основании полученных данных рассчитаем эффективность и ускорение. Полученные результаты представлены в Приложении 5.

ЗАКЛЮЧЕНИЕ

В ходе работы мы ознакомились с базовыми функциями библиотеки MPI, изучили 2 способа передачи данных в параллельных программах: «точка-точка» и использование коллективных коммуникаций.

По расчётам эффективности и ускорения можно сделать вывод о том, что метод коллективных коммуникаций работает быстрее «точки-точки», потому что в нем задействованы все процессы одного коммуникатора. Во взаимодействии же «точка-точка» обмен происходит только между двумя процессами одного коммуникатора. Также хочется отметить, что функции MPI_Send и MPI_Recv - блокирующие, т.е. они запускают операцию и возвращают управление процессу только после ее завершения, а это может снизить быстродействие программы.

Приложение 1. Листинг последовательной программы

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void fillVector(unsigned long long *vector1,
               unsigned long long *vector2, size_t sizeVect) {
    // srand(time(NULL));
    // for (size_t i = 0; i < sizeVect; i++) {
    //     vector1[i] = rand() % 100;
    //     vector2[i] = rand() % 100;
    // }

    /* for checking the correct computations*/
    for (size_t i = 0; i < sizeVect; i++) {
        vector1[i] = 1;
        vector2[i] = 2;
    }
}

// void printVector(unsigned long long *vector, size_t sizeVect) {
//     for (size_t i = 0; i < sizeVect; i++) {
//         printf("%lld ", vector[i]);
//     }
//     printf("\n");
// }

unsigned long long countScalarMult(unsigned long long *vector1,
                                   unsigned long long *vector2,
                                   size_t sizeVect1,
                                   size_t sizeVect2) {
    unsigned long long sum = 0;
    for (size_t i = 0; i < sizeVect1; i++) {
        for (size_t j = 0; j < sizeVect2; j++) {
            sum += vector1[i] * vector2[j];
        }
    }
    return sum;
}

int main() {
    // size_t sizeVect = 32 * 32; //for checking the correct
    computations
    size_t sizeVect = 32 * 1024 * 4;
    printf("Size of vector is: %ld\n", sizeVect);

    unsigned long long *vector1 = (unsigned long long *)calloc(
        sizeVect, sizeof(unsigned long long ));
}
```

```

unsigned long long *vector2 = (unsigned long long *)calloc(
    sizeVect, sizeof(unsigned long long ));

struct timespec start, end;
clock_gettime(CLOCK_MONOTONIC_RAW, &start);
fillVector(vector1, vector2, sizeVect);

// printVector(vector1, sizeVect);
// printVector(vector2, sizeVect);

unsigned long long result =
    countScalarMult(vector1, vector2, sizeVect, sizeVect);
clock_gettime(CLOCK_MONOTONIC_RAW, &end);

printf("Result is: %llu\n", result);
printf("Time taken: %lf sec\n",
    end.tv_sec - start.tv_sec +
    0.000000001 * (end.tv_nsec - start.tv_nsec));
free(vector1);
free(vector2);
}

```


Приложение 2. Листинг программы с использованием коммуникации «Точка-точка»

```
#include <float.h> //for DBL_MAX
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void fillVector(unsigned long long *vector1,
               unsigned long long *vector2, size_t sizeVect) {
    // srand(time(NULL));
    // for (size_t i = 0; i < sizeVect; i++) {
    //     vector1[i] = rand() % 100;
    //     vector2[i] = rand() % 100;
    // }

    /* for checking the correct computations*/
    for (size_t i = 0; i < sizeVect; i++) {
        vector1[i] = 1;
        vector2[i] = 2;
    }
}

// void printVector(unsigned long long *vector, size_t sizeVect) {
//     for (size_t i = 0; i < sizeVect; i++) {
//         printf("%lld ", vector[i]);
//     }
//     printf("\n");
// }

unsigned long long countScalarMult(unsigned long long *vector1,
                                   unsigned long long *vector2,
                                   size_t sizeVect1,
                                   size_t sizeVect2) {
    unsigned long long sum = 0;
    for (size_t i = 0; i < sizeVect1; i++) {
        for (size_t j = 0; j < sizeVect2; j++) {
            sum += vector1[i] * vector2[j];
        }
    }
    return sum;
}

int main(int argc, char *argv[]) {
    // size_t sizeVect = 32 * 32; //for checking the correct
    // computations
    size_t sizeVect = 24 * 1024 * 4;
    printf("Size of vector is: %ld\n", sizeVect);
```

```

MPI_Init(&argc, &argv);

int amountOfAvailableProc;
MPI_Comm_size(MPI_COMM_WORLD, &amountOfAvailableProc);
printf("Amount of processes: %d\n", amountOfAvailableProc);

int rankOfCurrentProc;
MPI_Comm_rank(MPI_COMM_WORLD, &rankOfCurrentProc);

const size_t lengthOfPortion = sizeVect / amountOfAvailableProc;

double minTime = DBL_MAX;

if (rankOfCurrentProc == 0) {
    unsigned long long *vector1 = (unsigned long long *)calloc(
        sizeVect, sizeof(unsigned long long));
    unsigned long long *vector2 = (unsigned long long *)calloc(
        sizeVect, sizeof(unsigned long long));

    double startTime = MPI_Wtime();
    fillVector(vector1, vector2, sizeVect);

    // printVector(vector1, sizeVect);
    // printVector(vector2, sizeVect);

    for (size_t i = 1; i < amountOfAvailableProc; i++) {
        int dest = i;
        int msgTag1 = 1;
        int msgTag2 = 2;

        //      sendBuff,      countOfElemsInBuff,      datatype,
destNumberOfReciever,
        // tagMsg, comm
        MPI_Send(&vector1[i * lengthOfPortion], lengthOfPortion,
            MPI_UNSIGNED_LONG_LONG, dest, msgTag1,
MPI_COMM_WORLD);
        MPI_Send(vector2, sizeVect, MPI_UNSIGNED_LONG_LONG, dest,
            msgTag2, MPI_COMM_WORLD);
    }

    unsigned long long result =
        countScalarMult(vector1, vector2, lengthOfPortion, sizeVect);

    for (size_t i = 1; i < amountOfAvailableProc; i++) {
        unsigned long long bufferOfPartialResults = 0;
        // data will save here
        int source = i;
        int tag = i;

        // recieveBuff, countOfRecvBuffer, datatype, sourceNumofSender,
        // tagMsg, comm, status

```

```

        MPI_Recv(&bufferOfPartialResults, 1, MPI_UNSIGNED_LONG_LONG,
                source, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        result += bufferOfPartialResults;
    }

    double endTime = MPI_Wtime();
    double timeForThisRepeat = endTime - startTime;
    if (timeForThisRepeat < minTime) {
        minTime = timeForThisRepeat;
    }

    printf("Result is: %llu\n", result);
    printf("Time taken: %f sec\n", minTime);

    free(vector1);
    free(vector2);
}

else {
    unsigned long long *vector1 = (unsigned long long *)calloc(
        sizeVect, sizeof(unsigned long long));
    unsigned long long *vector2 = (unsigned long long *)calloc(
        sizeVect, sizeof(unsigned long long));

    fillVector(vector1, vector2, sizeVect);

    // recieveBuff, countOfRecvBuffer, datatype, sourceNumofSender,
    // tagMsg, comm, status
    MPI_Recv(vector1, lengthOfPortion, MPI_UNSIGNED_LONG_LONG, 0, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    MPI_Recv(vector2, sizeVect, MPI_UNSIGNED_LONG_LONG, 0, 2,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    unsigned long long result =
        countScalarMult(vector1, vector2, lengthOfPortion, sizeVect);

    MPI_Send(&result, 1, MPI_UNSIGNED_LONG_LONG, 0,
rankOfCurrentProc,
             MPI_COMM_WORLD);

    free(vector1);
    free(vector2);
}

MPI_Finalize();
}

```

Приложение 3. Листинг программы при использовании коллективных коммуникаций

```
#include <float.h> //for DBL_MAX
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

const int root = 0;

void fillVector(unsigned long long *vector1, unsigned long long *vector2,
               size_t sizeVect) {
    // srand(time(NULL));
    // for (size_t i = 0; i < sizeVect; i++) {
    //     vector1[i] = rand() % 100;
    //     vector2[i] = rand() % 100;
    // }

    /* for checking the correct computations*/
    for (size_t i = 0; i < sizeVect; i++) {
        vector1[i] = 1;
        vector2[i] = 2;
    }
}

// void printVector(unsigned long long *vector, size_t sizeVect) {
//     for (size_t i = 0; i < sizeVect; i++) {
//         printf("%lld ", vector[i]);
//     }
//     printf("\n");
// }

unsigned long long countScalarMult(unsigned long long *vector1, unsigned
long long *vector2,
                                size_t sizeVect1, size_t sizeVect2) {
    unsigned long long sum = 0;
    for (size_t i = 0; i < sizeVect1; i++) {
        for (size_t j = 0; j < sizeVect2; j++) {
            sum += vector1[i] * vector2[j];
        }
    }
    return sum;
}

int main(int argc, char *argv[]) {
    // size_t sizeVect = 32 * 32; // for checking the correct computations
    size_t sizeVect = 24 * 1024 * 4;
```

```

printf("Size of vector is: %ld\n", sizeVect);

MPI_Init(&argc, &argv);

int amountOfAvailableProc;
MPI_Comm_size(MPI_COMM_WORLD, &amountOfAvailableProc);
printf("Amount of processes: %d\n", amountOfAvailableProc);

int rankOfCurrentProc;
MPI_Comm_rank(MPI_COMM_WORLD, &rankOfCurrentProc);

unsigned long long *baseVector1 = NULL;
unsigned long long *baseVector2 = NULL;

const size_t lengthOfPortion = sizeVect / amountOfAvailableProc;
unsigned long long *bufferedVector1 =
    (unsigned long long *)calloc(lengthOfPortion, sizeof(unsigned long
long ));
unsigned long long *bufferedVector2 =
    (unsigned long long *)calloc(sizeVect, sizeof(unsigned long
long ));

double startTime = 0;

if (rankOfCurrentProc == 0) {
    baseVector1 = (unsigned long long *)calloc(sizeVect, sizeof(unsigned
long long ));
    baseVector2 = (unsigned long long *)calloc(sizeVect, sizeof(unsigned
long long ));
    fillVector(baseVector1, baseVector2, sizeVect);

    // printVector(baseVector1, sizeVect);
    // printVector(baseVector2, sizeVect);

    memcpy(bufferedVector2, baseVector2, sizeVect * sizeof(unsigned long
long ));
    startTime = MPI_Wtime();
}

// 1st vector uses Scatter
// sendbuf, sendcount, sendtype, recievebuf, recievetype, root, comm
MPI_Scatter(baseVector1, lengthOfPortion, MPI_UNSIGNED_LONG_LONG,
            bufferedVector1, lengthOfPortion, MPI_UNSIGNED_LONG_LONG,
root,
            MPI_COMM_WORLD);

// 2nd vector uses Broadcast
// inout buffer, countOfSendingElems, datatype, root, comm
MPI_Bcast(bufferedVector2, sizeVect, MPI_UNSIGNED_LONG_LONG, root,
            MPI_COMM_WORLD);

```

```

    unsigned long long tmpRes = countScalarMult(bufferedVector1, buffered-
Vector2,
                                                lengthOfPortion, sizeVect);

    unsigned long long finalRes = 0;

    // Broadcast + Reduce for the 2nd vector

    // sendbuff, recieverbuff, countOfElems in sendBuff, datatype,
    // operation, root, comm
    MPI_Reduce(&tmpRes, &finalRes, 1, MPI_UNSIGNED_LONG_LONG, MPI_SUM,
root,
              MPI_COMM_WORLD);

    double minTime = DBL_MAX;
    if (rankOfCurrentProc == 0) {
        double endTime = MPI_Wtime();
        double timeForThisRepeat = endTime - startTime;
        if (timeForThisRepeat < minTime) {
            minTime = timeForThisRepeat;
        }
        printf("Result is: %llu\n", finalRes);
        printf("Time taken: %f sec\n", minTime);
    }

    MPI_Finalize();

    free(baseVector1);
    free(baseVector2);
    free(bufferedVector1);
    free(bufferedVector2);
}

```

Приложение 4. Проверка корректности работы программы

```
opp@comrade:~/205/Evdokimova/laba1$ gcc -o sequential sequential.c
opp@comrade:~/205/Evdokimova/laba1$ ./sequential
Size of vector is: 1024
Result is: 2097152
Time taken: 0.007935 sec
```

Рис. 1. Результат для последовательной программы

```
opp@comrade:~/205/Evdokimova/laba1$ mpicc -o point-to-point point-to-point.c
opp@comrade:~/205/Evdokimova/laba1$ mpiexec -n 2 ./point-to-point
Size of vector is: 1024
Size of vector is: 1024
Amount of processes: 2

Amount of processes: 2

Result is: 2097152
Time taken: 0.002324 sec
opp@comrade:~/205/Evdokimova/laba1$ mpiexec -n 4 ./point-to-point
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Amount of processes: 4

Amount of processes: 4

Amount of processes: 4

Amount of processes: 4

Result is: 2097152
Time taken: 0.001692 sec
```

Рис. 2. Результат для коммуникации «точка-точка» для 2х и 3х процессов

```
opp@comrade:~/205/Evdokimova/lab1$ mpiexec -n 8 ./point-to-point
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Result is: 2097152
Time taken: 0.001178 sec
```

Рис. 3. Результат для коммуникации «точка-точка» для 8и процессов

```
Amount of processes: 16

Result is: 2097152
Time taken: 0.001507 sec
```

Рис. 4. Результат для коммуникации «точка-точка» для 16и процессов


```

opp@comrade:~/205/Evdokimova/lab1$ mpicc -o collective_commun collective_commun.c
opp@comrade:~/205/Evdokimova/lab1$
opp@comrade:~/205/Evdokimova/lab1$ mpicc -o collective_commun collective_commun.c
opp@comrade:~/205/Evdokimova/lab1$ mpiexec -n 2 ./collective_commun
Size of vector is: 1024
Size of vector is: 1024
Amount of processes: 2

Amount of processes: 2

Result is: 2097152
Time taken: 0.002618 sec
opp@comrade:~/205/Evdokimova/lab1$ mpiexec -n 4 ./collective_commun
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Amount of processes: 4

Amount of processes: 4

Amount of processes: 4

Amount of processes: 4

Result is: 2097152
Time taken: 0.001608 sec

```

Рис. 5. Результат для коллективной коммуникации для 2х и 4х процессов

```

opp@comrade:~/205/Evdokimova/lab1$ mpiexec -n 8 ./collective_commun
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Size of vector is: 1024
Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Amount of processes: 8

Result is: 2097152
Time taken: 0.000887 sec

```

Рис. 6. Результат для коллективной коммуникации для 8и процессов

```
Amount of processes: 16  
Result is: 2097152  
Time taken: 0.000921 sec
```

Рис. 7. Результат для коллективной коммуникации для 16и процессов

Приложение 5. Графики времени, ускорения и эффективности

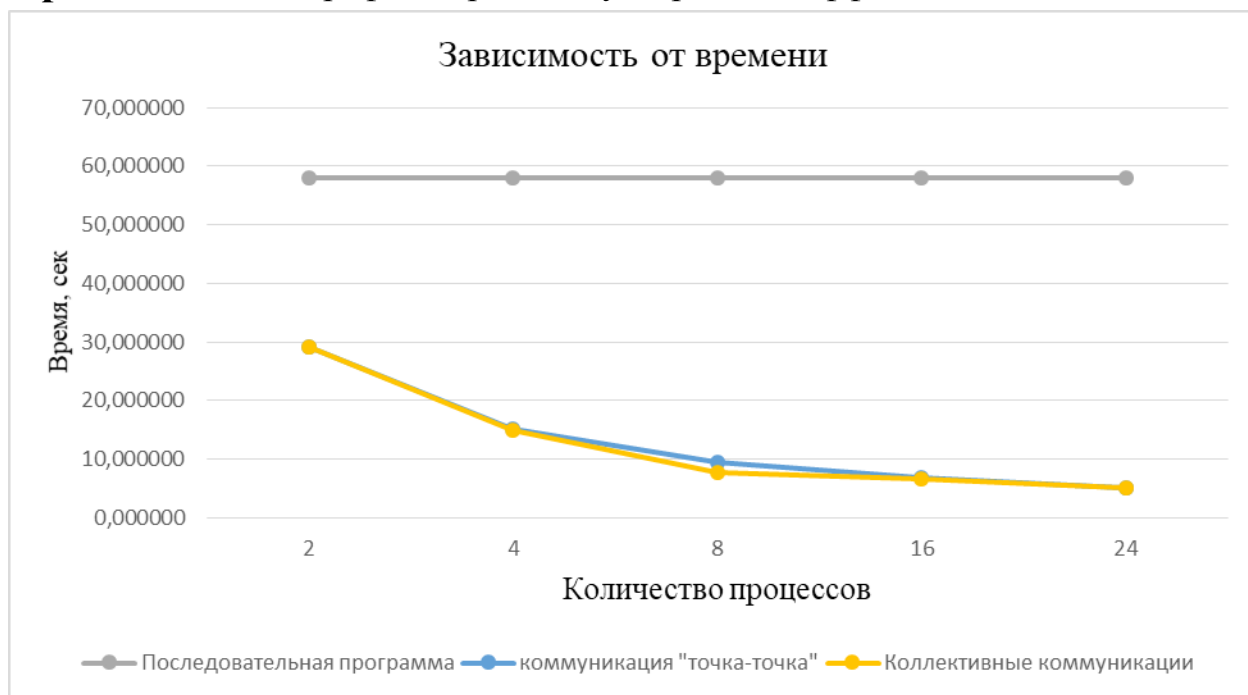


Рис. 1. График зависимости от времени

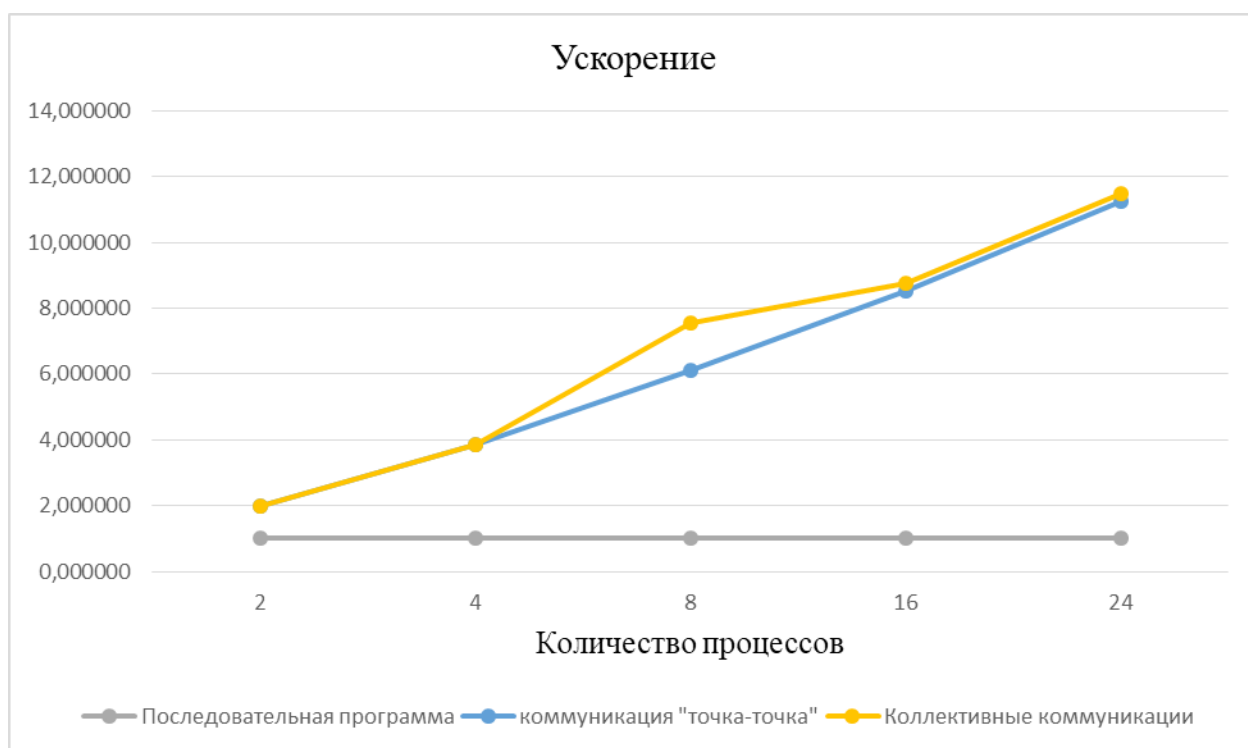


Рис. 2. График ускорения

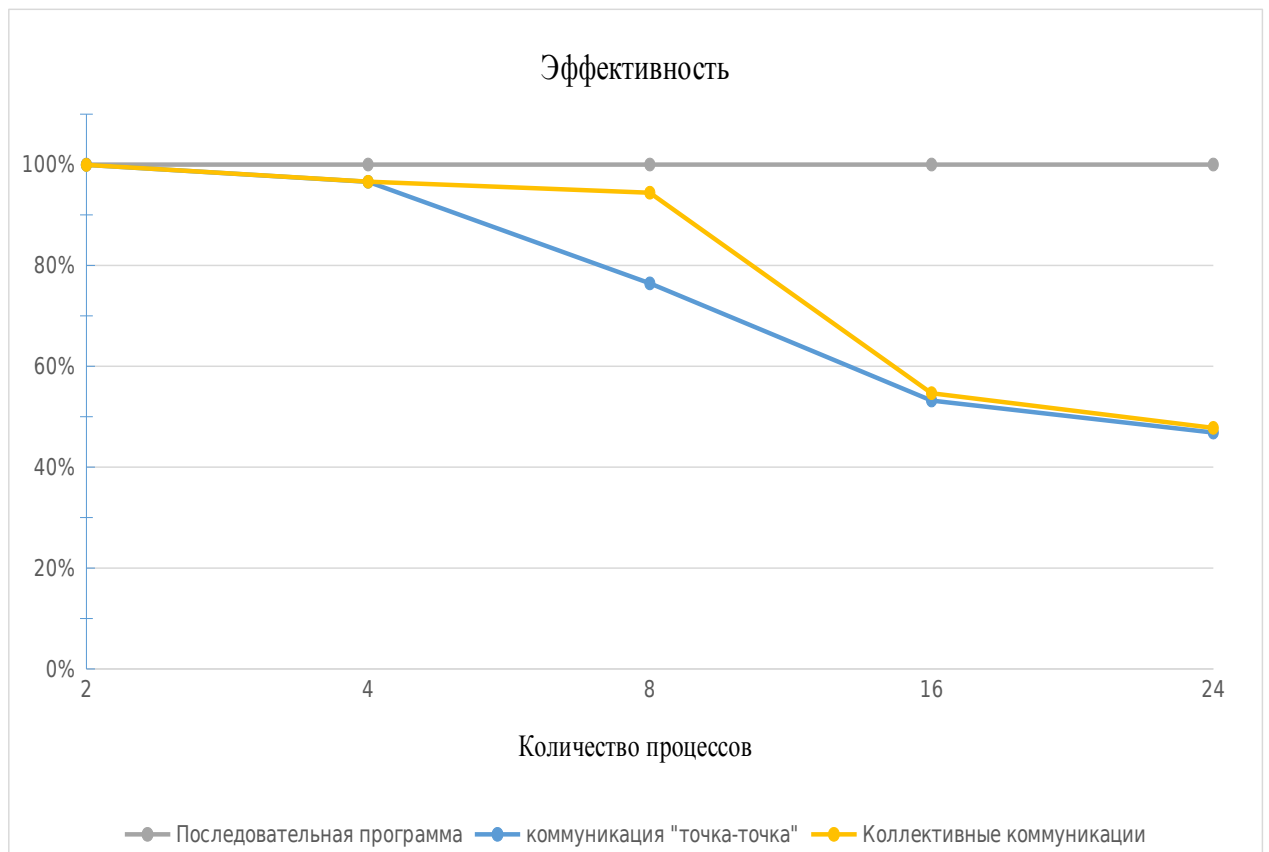


Рис. 3. График эффективности