

**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**

«Вычисление числа двух векторов с использованием МР»

студентки 2 курса, группы 21207

**Черновской Яны Тихоновны**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю. Власенко

Новосибирск 2023

## СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
ЦЕЛЬ	3
ЗАДАНИЕ	4
ОПИСАНИЕ РАБОТЫ	5
ЗАКЛЮЧЕНИЕ	8
ПРИЛОЖЕНИЕ	9
Приложение 1. Полный листинг последовательной программы на С	9
Приложение 2. Полный листинг параллельной программы на С, использующей коммуникацию типа точка-точка	10
Приложение 3. Полный листинг параллельной программы на С, использующей коллективные коммуникации	13

## ЦЕЛЬ

Сравнение времени выполнения расчёта числа в двойном цикле путем последовательного и параллельного выполнения.

## ЗАДАНИЕ

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

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

a) последовательная программа

b) параллельная, использующая коммуникации типа точка-точка (MPI\_Send<sup>1</sup>, MPI\_Recv<sup>2</sup>)

c) параллельная, использующая коллективные коммуникации (MPI\_Scatter<sup>3</sup>, MPI\_Reduce<sup>4</sup>, MPI\_Bcast<sup>5</sup>)

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.

---

<sup>1</sup> MPI\_Send(const void \*buf, int count, MPI\_Datatype datatype, int dest, int tag, MPI\_Comm comm)

<sup>2</sup> MPI\_Recv(void \*buf, int count, MPI\_Datatype datatype, int source, int tag, MPI\_Comm comm, MPI\_Status \*status)

<sup>3</sup> MPI\_Scatter (const void \*sendbuf, int sendcount, MPI\_Datatype sendtype, void \*recvbuf, int recvcount, MPI\_Datatype recvtype, int root, MPI\_Comm comm)

<sup>4</sup> MPI\_Reduce(const void \*sendbuf, void \*recvbuf, int count, MPI\_Datatype datatype, MPI\_Op op, int root, MPI\_Comm comm)

<sup>5</sup> MPI\_Bcast(void \*buffer, int count, MPI\_Datatype datatype, int root, MPI\_Comm comm)

## ОПИСАНИЕ РАБОТЫ

1. Было написано три программы, каждая из которых вычисляет число  $s$  в двойном цикле.
2. Было замерено время работы (размер вектора определён экспериментально, чтобы первая программа выполняла свою работу за 30 секунд)

\*Управляющим процессом считается процесс с рангом 0.

- ☐ последовательной программы:

```
opp@comrade:~/207/Chernovskaya/lab0$ ./first.out
Result 24025055109048
Total time is 37 seconds
```

- ☐ параллельной программы, использующей коммуникацию типа точка-точка (MPI\_Send, MPI\_Recv)

```
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 2 ./second.out
Total result = 24025055109048
Total time is 17.518223
```

```
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 4 ./second.out
Total result = 24025055109048
Total time is 9.754369
```

```
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 8 ./second.out
Total result = 24025055109048
Total time is 4.653370
```

```
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 16 ./second.out
Total result = 24025055109048
Total time is 4.116824
```

```
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 24 ./second.out
Total result = 24025055109048
Total time is 2.889431
```

- ☐ параллельной программы, использующей коллективные коммуникации (MPI\_Scatter, MPI\_Reduce, MPI\_Bcast)

```
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 2 ./third.out
Total result = 24025055109048
Total time is 17.232051
```

```
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 4 ./third.out
Total result = 24025055109048
Total time is 9.046397
```

```
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 8 ./third.out
Total result = 24025055109048
Total time is 4.621271
```

```
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 16 ./third.out
Total result = 24023120241838
Total time is 4.093075
```

```
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 24 ./third.out
Total result = 24025055109048
Total time is 3.020290
```

3. Были построены графики времени, ускорения и эффективности, где

а) Ускорение:  $S_p = T_1 / T_p$ , где  $T_1$  - время работы последовательной программы,  $T_p$  - время работы параллельной программы на  $p$  процессах/потоках

б) Эффективность  $E_p = S_p / p * 100\%$ .

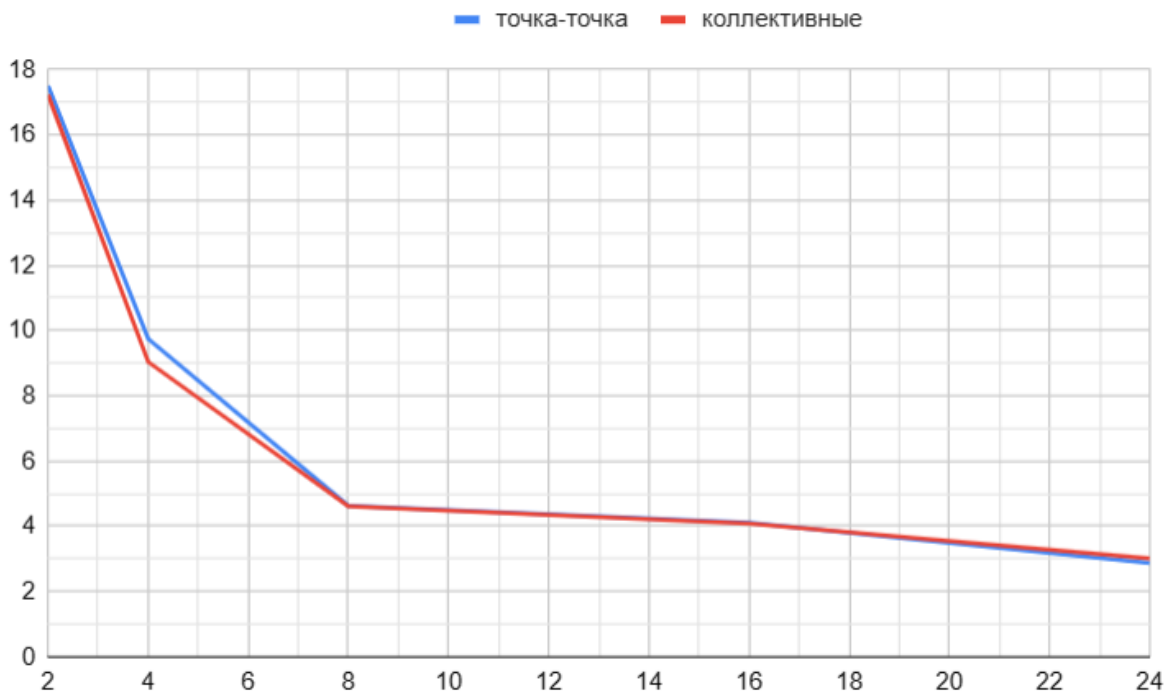


Рис.1 График времени

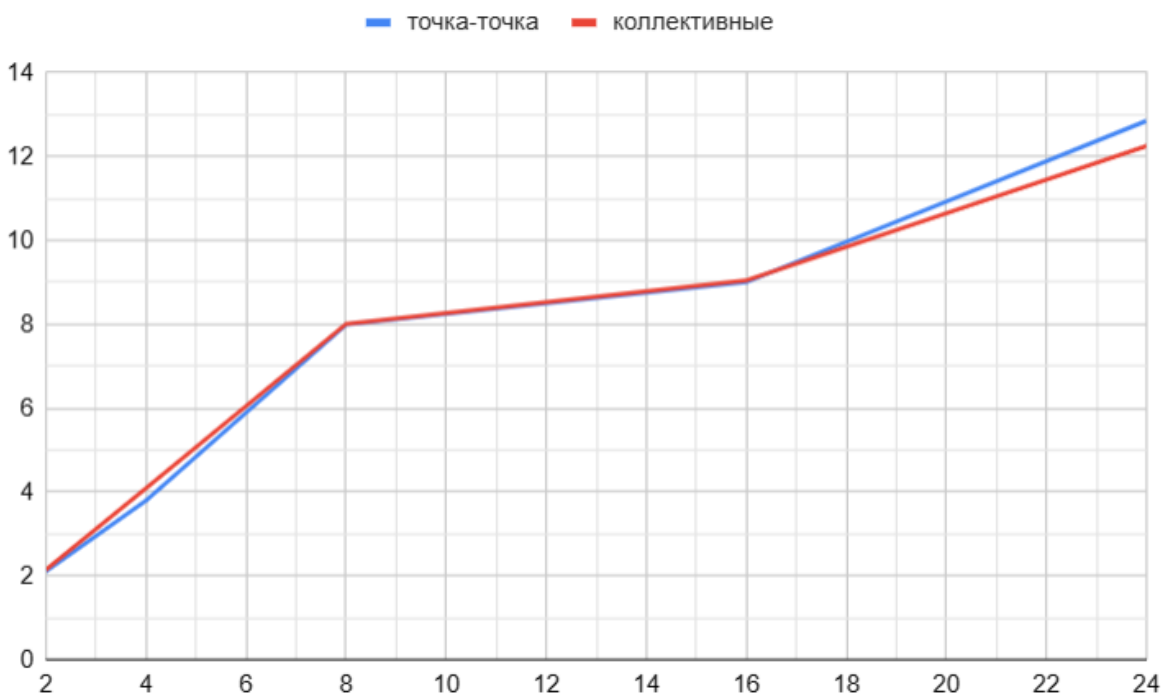


Рис 2. График ускорения

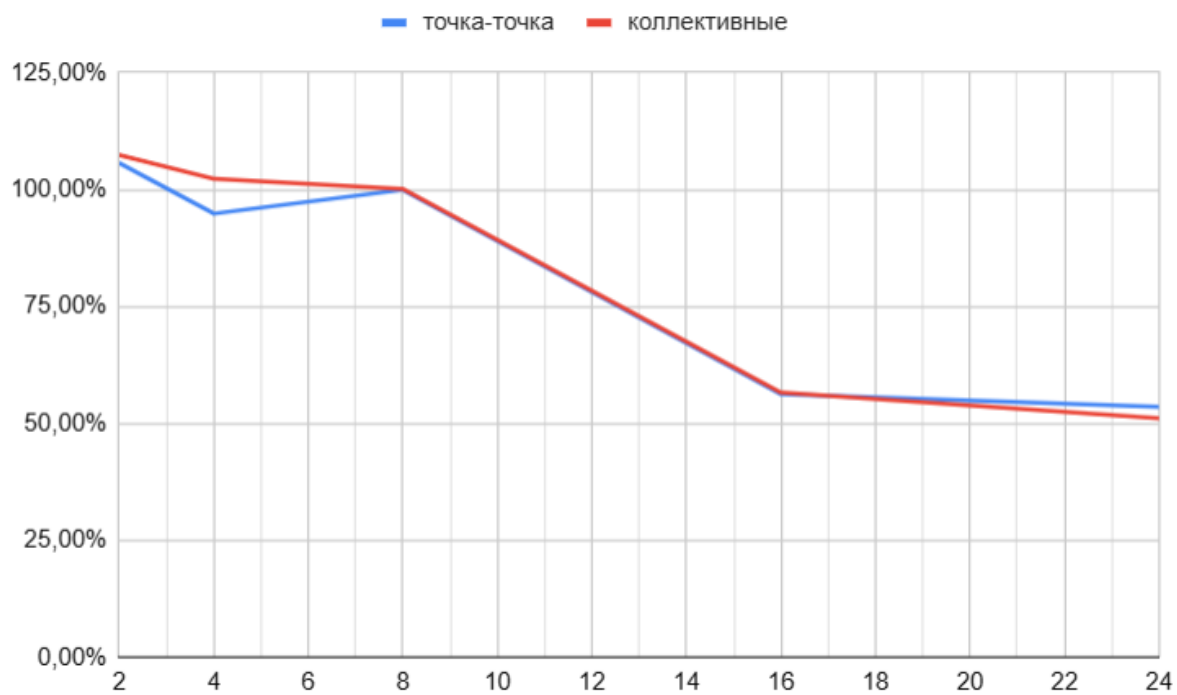


Рис 3. График эффективности



## ЗАКЛЮЧЕНИЕ

Параллельное программирование дает возможность уменьшить время выполнения программы в несколько раз

## ПРИЛОЖЕНИЕ

### Приложение 1. Полный листинг последовательной программы на C

```
1. #include "stdio.h"
2. #include "stdlib.h"
3. #include "time.h"
4. #include "limits.h"
5.
6. int SIZE = 99000;
7.
8. void createVector(int* vector1, int* vector2, int size)
9. {
10.     for (long long int i = 0; i < size; i++) {
11.         vector1[i] = rand() % 100;
12.         vector2[i] = rand() % 100;
13.     }
14. }
15.
16. long long int mult(int* vector1, int* vector2, int size)
17. {
18.     long long int result = 0;
19.     for (int i = 0; i < size; i++) {
20.         for (int j = 0; j < size; j++){
21.             result += vector1[i] * vector2[j];
22.         }
23.     }
24.     return result;
25. }
26.
27. int main(int argc, char *argv[])
28. {
29.     int* vector1 = malloc(SIZE * sizeof(int));
30.     int* vector2 = malloc(SIZE * sizeof(int));
31.
32.     createVector(vector1, vector2, SIZE);
33.
34.     time_t begin = time(NULL);
35.     printf("Result %lld\n", mult(vector1, vector2, SIZE));
36.     time_t end = time(NULL);
37.     printf("Total time is %ld seconds\n", (end - begin));
38.
39.     free(vector1);
40.     free(vector2);
41.
42.     return 0;
43. }
```

## Приложение 2. Полный листинг параллельной программы на С, использующей коммуникацию типа точка-точка

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. #include "mpi.h"
5. #define SIZE 99000
6.
7. int calcArraySize(int numProc) {
8.     if (SIZE % numProc == 0) {
9.         return SIZE;
10.    }
11.    else {
12.        return SIZE + numProc - (SIZE % numProc);
13.    }
14. }
15.
16. void createVector(int* vector1, int* vector2, int size)
17. {
18.     for (unsigned int i = 0; i < SIZE; i++) {
19.         vector1[i] = (rand() % 100);
20.         vector2[i] = (rand() % 100);
21.     }
22.
23.     for(unsigned int i = SIZE; i < size; i++)
24.     {
25.         vector1[i] = 0;
26.         vector1[i] = 0;
27.     }
28. }
29.
30. long long mult(int* vector1, int* vector2, int size) {
31.     long long result = 0;
32.     for (int i = 0; i < size; ++i) {
33.         for (int j = 0; j < SIZE; ++j) {
34.             result += vector1[i] * vector2[j];
35.         }
36.     }
37.     return result;
38. }
39.
40. int main(int argc, char** argv) {
41.
```

```

42.  int rank, numProc = 0;
43.
44.  long long totalResult, currentResult;
45.  double startTime, endTime;
46.
47.  MPI_Init(&argc, &argv);
48.  MPI_Comm_rank( MPI_COMM_WORLD, &rank);
49.  MPI_Comm_size( MPI_COMM_WORLD, &numProc);
50.
51.
52.  const int currentArraySize = calcArraySize(numProc);
53.
54.  int* vector1 = malloc(currentArraySize* sizeof(int));
55.  int* vector2 = malloc(currentArraySize * sizeof(int));
56.
57.  const int currentSize = currentArraySize / numProc;
58.
59.  if (rank == 0) {
60.      createVector(vector1, vector2, currentArraySize);
61.
62.      startTime = MPI_Wtime();
63.
64.      int shift = currentSize;
65.      for (int i = 1; i < numProc; ++i) {
66.          MPI_Send(vector1 + shift, currentSize, MPI_INT, i, 1,
MPI_COMM_WORLD);
67.          MPI_Send(vector2, SIZE, MPI_INT, i, 1, MPI_COMM_WORLD);
68.          shift += currentSize;
69.      }
70.
71.      totalResult = mult(vector1, vector2, currentSize);
72.      currentResult = 0;
73.
74.      for (int i = 1; i < numProc; ++i) {
75.          MPI_Recv(&currentResult, 1, MPI_LONG_LONG, MPI_ANY_SOURCE, 1,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
76.          totalResult += currentResult;
77.      }
78.
79.  }
80.  else {
81.      MPI_Recv(vector1, currentSize, MPI_INT, 0, 1, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

```

```
82.     MPI_Recv(vector2, SIZE, MPI_INT, 0, 1, MPI_COMM_WORLD,  
MPI_STATUS_IGNORE);  
83.     currentResult = mult(vector1, vector2, currentSize);  
84.     MPI_Send(&currentResult, 1, MPI_LONG_LONG, 0, 1,  
MPI_COMM_WORLD);  
85. }  
86.  
87. if (rank == 0)  
88. {  
89.     printf("Total result = %lld\n", totalResult);  
90.     endTime = MPI_Wtime();  
91.     printf("Total time is %f\n", endTime - startTime);  
92.  
93.     free(vector1);  
94.     free(vector2);  
95. }  
96.  
97. MPI_Finalize();  
98. return 0;  
99. }
```

**Приложение 3. Полный листинг параллельной программы на C, использующей коллективные коммуникации**

```
1. #include "mpi.h"
2. #include "stdio.h"
3. #include "stdlib.h"
4. #include "limits.h"
5.
6. const unsigned int SIZE = 99000;
7.
8. void createVector(int* vector1, int* vector2, int size)
9. {
10.     for (unsigned int i = 0; i < size; i++) {
11.         vector1[i] = (rand() % 100);
12.         vector2[i] = (rand() % 100);
13.     }
14. }
15.
16. long long mult(const int* vector1, const int* vector2, int size)
17. {
18.     long long result = 0;
19.     for (int i = 0; i < size; i++) {
20.         for (int j = 0; j < SIZE; j++)
21.         {
22.             result += vector1[i] * vector2[j];
23.         }
24.     }
25.     return result;
26. }
27.
28. int main(int argc, char *argv[])
29. {
30.     int rank, numProc = 0;
31.
32.     int* vector1;
33.     int* vector2;
34.
35.     int* currentVector;
36.
37.     long long totalResult, currentResult;
38.     double startTime, endTime;
39.
40.     MPI_Init(&argc, &argv);
41.     MPI_Comm_rank( MPI_COMM_WORLD, &rank);
42.     MPI_Comm_size( MPI_COMM_WORLD, &numProc);
43.
44.     int currentSize = SIZE / numProc;
45.     vector2 = malloc(SIZE * sizeof(int));
```

```
46.
47.  if (rank == 0)
48.  {
49.      totalResult = 0;
50.
51.      vector1 = malloc(SIZE * sizeof(int));
52.      createVector(vector1, vector2, SIZE);
53.
54.      startTime = MPI_Wtime();
55.  }
56.
57.  currentVector = malloc(currentSize * sizeof(int));
58.
59.  MPI_Scatter(vector1, currentSize, MPI_INT,
60.             currentVector, currentSize, MPI_INT, 0, MPI_COMM_WORLD);
61.  MPI_Bcast(vector2, SIZE, MPI_INT, 0, MPI_COMM_WORLD);
62.
63.  currentResult = mult(currentVector, vector2, currentSize);
64.
65.  MPI_Reduce(&currentResult, &totalResult, 1,
66.            MPI_LONG_LONG, MPI_SUM, 0, MPI_COMM_WORLD);
67.
68.  if (rank == 0)
69.  {
70.      printf("Total result = %lld\n", totalResult);
71.      endTime = MPI_Wtime();
72.      printf("Total time is %f\n", endTime - startTime);
73.  }
74.
75.  MPI_Finalize();
76.
77.  return (0);
78. }
```