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 21207

Черновской Яны Тихоновны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю. Власенко

Новосибирск 2023

СОДЕРЖАНИЕ

ЦЕЛЬ

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

ЗАДАНИЕ

I Написать 3 программы, каждая их 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[1], MPI_Recv[2])

c) параллельная, использующая коллективные коммуникации (MPI_Scatter[3], MPI_Reduce[4], MPI_Bcast[5])

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.

---

[1] MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest int tag, MPI_Comm comm)

[2] MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

[3] MPI_Scatter (const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

[4] MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

[5] 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 34.181599
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 4 ./second.out
Total result = 24024609354830
Total time is 34.730797
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 8 ./second.out
Total result = 24023747236782
Total time is 35.683829
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 8 ./second.out
Total result = 24023747236782
Total time is 35.700942
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 16 ./second.out
Total result = 24021283342588
Total time is 41.513617
opp@comrade:~/207/Chernovskaya/lab0$ mpiexec -n 24 ./second.out
Total result = 24020866978758
Total time is 50.066146
```

- ☐ 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 13.567766
```

```
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 4.262663
```

```
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: Sp = T1 / Tp, где T1 - время работы последовательной программы, Tp - время работы параллельной программы на p процессах/потоках

б) Эффективность Ep = Sp / p * 100%.

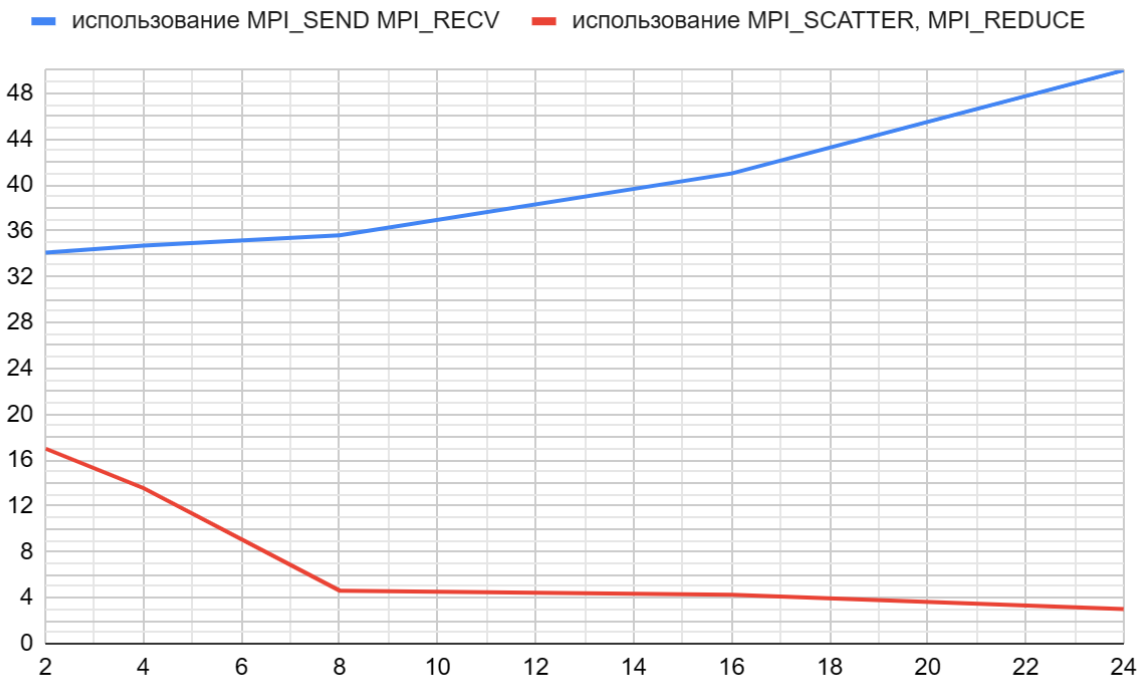

Рис.1 График времени

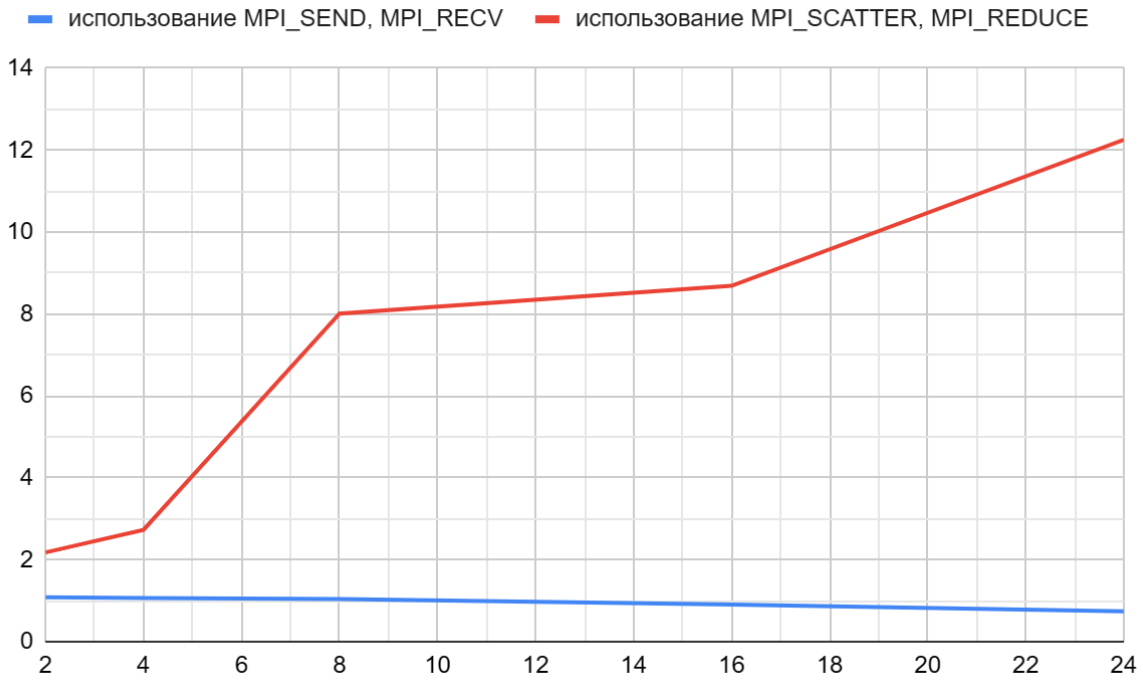

Рис 2. График ускорения

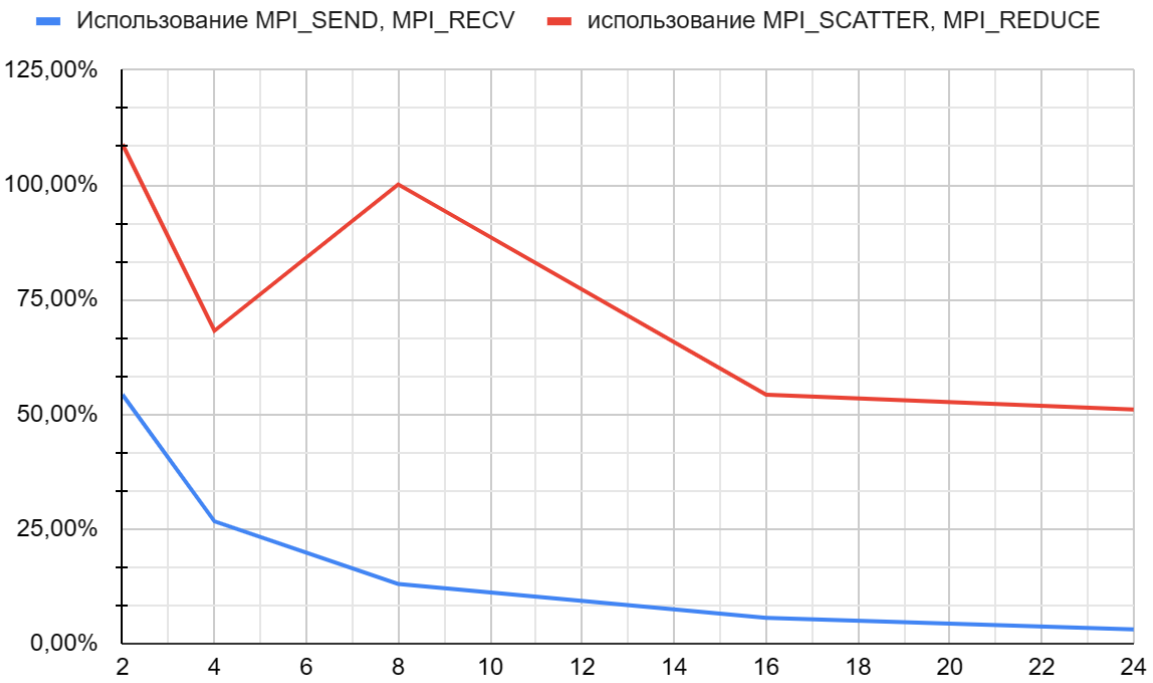

Рис 3. График эффективности

ЗАКЛЮЧЕНИЕ

Анализируя полученные графики, можно сделать вывод, что в программе наиболее эффективно использовать параллельную программу с функциями MPI_Scatter, MPI_Reduce, MPI_Bcast. Время при использовании коммуникации типа точка - точка возрастает при увеличении количества потоков.

ПРИЛОЖЕНИЕ

Приложение 1. Полный листинг последлвательной программы на C

```c
#include "stdio.h"
#include "stdlib.h"
#include "time.h"
#include "limits.h"

int SIZE = 99000;

void createVector(int* vector1, int* vector2, int size)
{
    for (long long int i = 0; i < size; i++) {
        vector1[i] = rand() % 100;
        vector2[i] = rand() % 100;
    }
}

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

int main(int argc, char *argv[])
{
    int* vector1 = malloc(SIZE * sizeof(int));
    int* vector2 = malloc(SIZE * sizeof(int));

    createVector(vector1, vector2, SIZE);

    time_t begin = time(NULL);
    printf("Result %lld\n", mult(vector1, vector2, SIZE));
    time_t end = time(NULL);
    printf("Total time is %ld seconds\n", (end - begin));

    free(vector1);
    free(vector2);

    return 0;
}
```

Приложение 2. Полный листинг паралелльной программы на C, использующей коммуникацию типа точка-точка

```
1.  #include "mpi.h"
2.  #include "stdio.h"
3.  #include "stdlib.h"
4.  #include "limits.h"ls
5.
6.  const 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 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.       {
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.    int *currentVector;
35.
36.    long long totalResult, currentResult;
37.    double startTime, endTime;
38.
39.    MPI_Init(&argc, &argv);
40.    MPI_Comm_rank( MPI_COMM_WORLD, &rank);
41.    MPI_Comm_size( MPI_COMM_WORLD, &numProc);
```

```
42.
43.    int currentSize = SIZE / (numProc - 1) + 1;
44.
45.    currentVector = malloc(currentSize * sizeof(int));
46.    vector1 = malloc(SIZE * sizeof(int));
47.    vector2 = malloc(SIZE * sizeof(int));
48.
49.    if (rank == 0)
50.    {
51.       totalResult = 0;
52.
53.       createVector(vector1, vector2, SIZE);
54.
55.       startTime = MPI_Wtime();
56.       for (int i = 1; i < numProc; i++)
57.       {
58.          for (int j = 0; j < currentSize - 1; j++)
59.          {
60.             currentVector[j] = vector1[(i - 1) * currentSize + j];
61.          }
62.
63.          if (i <=  SIZE % (numProc - 1))
64.          {
65.             currentVector[currentSize] = vector1[(numProc - 1) * (currentSize - 1) + (i -
    1)];
66.          }
67.
68.          else
69.          {
70.             currentVector[currentSize] = 0;
71.          }
72.
73.          MPI_Send(currentVector, currentSize, MPI_INT, i, 1,
    MPI_COMM_WORLD);
74.          MPI_Send(vector2, SIZE, MPI_INT, i, 1, MPI_COMM_WORLD);
75.
76.          MPI_Recv(&currentResult, 1, MPI_LONG_LONG,i, 1,
    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
77.
78.          totalResult += currentResult;
79.
80.       }
81.    }
82.
```

```
83.    else
84.    {
85.        currentResult = 0;
86.
87.        MPI_Recv(currentVector, currentSize, MPI_INT, 0, 1, MPI_COMM_WORLD,
    MPI_STATUS_IGNORE);
88.        MPI_Recv(vector2, SIZE, MPI_INT, 0, 1, MPI_COMM_WORLD,
    MPI_STATUS_IGNORE);
89.
90.        currentResult = mult(currentVector, vector2, currentSize);
91.
92.        MPI_Send(&currentResult, 1, MPI_LONG_LONG, 0, 1,
    MPI_COMM_WORLD);
93.
94.    }
95.
96.    if (rank == 0)
97.    {
98.        printf("Total result = %lld\n", totalResult);
99.        endTime = MPI_Wtime();
100.           printf("Total time is %f\n", endTime - startTime);
101.
102.           free(currentVector);
103.           free(vector1);
104.           free(vector2);
105.       }
106.
107.       MPI_Finalize();
108.
109.       return (0);
110.   }
```

Приложение 3. Полный листинг паралелльной 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. }
```