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

Балашова Вячеслава Вадимовича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
А. Ю. Власенко

Новосибирск 2023

Содержание

Цель

1. Изучение одного из основных методов параллельного программирования – программного интерфейса Message Parsing Interface (MPI)

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];
```

   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)

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

Ход работы:

1) На языке программирования Си были написаны три программы, выполняющие некоторую операцию с квадратичной сложностью над двумя векторами:

   a. Первая программа в файле «main_default.c» (см. Приложение 1) не содержит никаких распараллеливающих методов.

   b. Вторая программа в файле «main_dot.c» (См. Приложение 2) подразумевает разбиение на несколько процессов, взаимодействующих между собой по принципу связи «точка-точка» - одна ветвь вызывает функцию передачи данных, а другая – функцию приема данных.

   c. Третья программа в файле «main_collective.c» (См. Приложение 3) тоже подразумевает разбиение на несколько процессов, но взаимодействующих между собой при помощи функций коллективного обмена данными – на прием и/или передачу работают одновременно все задачи-абоненты указанного коммуникатора.

2) В каждой программе был произведен замер времени работы алгоритма (в случае параллельных программ вычислением времени занимался процесс с рангом = 0). Также для параллельных программ было проведено вычисление времени их работы на разном количестве процессов: 2, 4, 8, 16, 24. Каждая программа запускалась по 5 раз, и из этих 5 запусков бралось минимальное время работы (чтобы исключить погрешность).

3) Были построены графики зависимости времени работы программы, эффективности (Ep = Sp / p * 100%, где Sp – ускорение, p – количество процессов) и ускорения (Sp = T1 / Tp, где T1 – время работы последовательной программы, Tp – время работы программы на p процессах) от количества параллельных процессов.

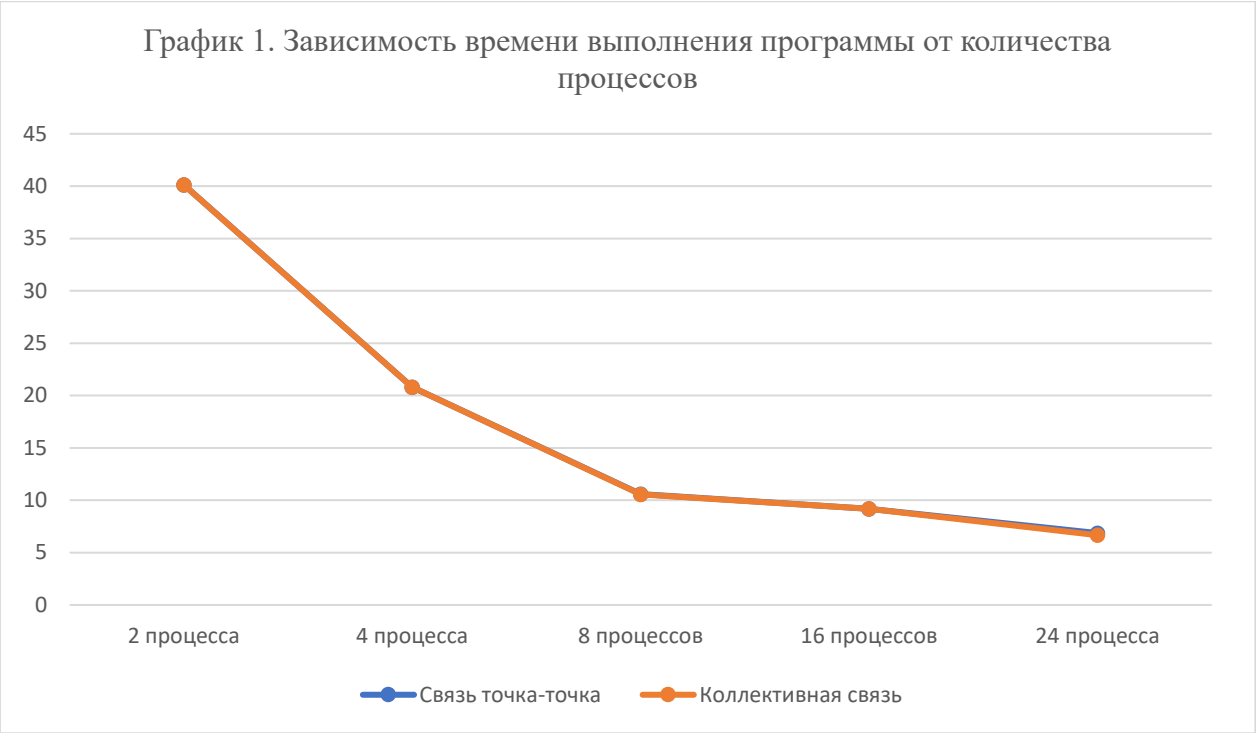
График 1. Зависимость времени выполнения программы от количества процессов
45
40
35
30
25
20
15
10
5
0
2 процесса
4 процесса
8 процессов
16 процессов
24 процесса
Связь точка-точка
Коллективная связь

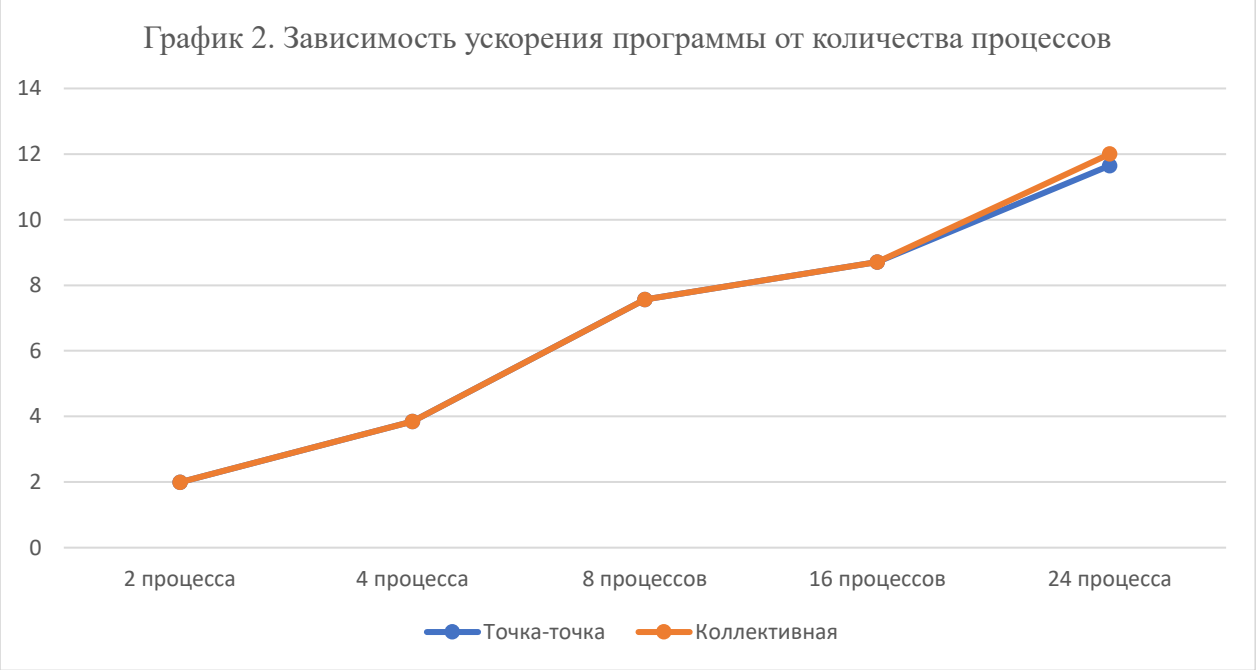
График 2. Зависимость ускорения программы от количества процессов
14
12
10
8
6
4
2
0
2 процесса
4 процесса
8 процессов
16 процессов
24 процесса
Точка-точка
Коллективная

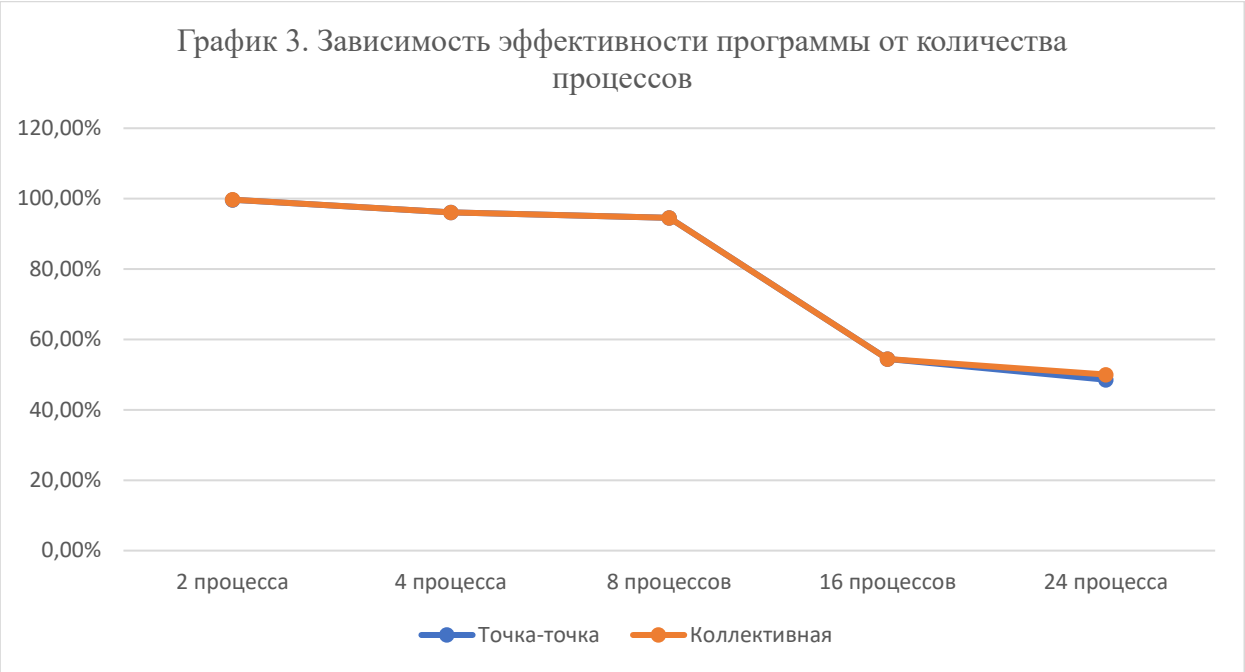

График 3. Зависимость эффективности программы от количества процессов

Заключение

В ходе выполнения практической работы были изучены базовые методы программного интерфейса параллельного программирования MPI.

Были изучены основные методы взаимодействия процессов в интерфейсе MPI: точка-точка и коллективное взаимодействия. Первый вид взаимодействия основан на том, что один процесс отправляет некоторые данные, а другой их принимает. Коллективное взаимодействие подразумевает взаимодействие между представителями одной группы на основе их общих внутренних данных. Например, разделить один большой массив из одного процесса на все процессы группы, отдав им последовательные кусочки, или собрать информацию со всех представителей какой-то группы, скопировать некоторое содержимое из одного процесса в другие и так далее.

Так же были выявлены некоторые особенности и проблемы данного интерфейса. Прежде всего интерфейс не подразумевает обмен указателями, из чего следует, что все данные передаются по значениям. Даже если производится работа над неизменяемым в процессе объектом, для работы с ним нужна его копия, что порождает затраты времени на копирование.

Приложение 1. Исходный код последовательной программы (main_default.c)

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

typedef unsigned long long ull;

ull countInVectors(int * a, int * b, int a_size, int b_size)
{
   ull forRet = 0;
   for (int i = 0; i < a_size; i++)
   {
      for (int j = 0; j < b_size; j++)
      {
         forRet += a[i] * b[j];
      }
   }
   return forRet;
}

int main(int argc, char * argv[])
{
   int vectorLength = 145000;
   if (argc > 1)
   {
      vectorLength = atoi(argv[1]);
   }

   int * a = (int *) malloc(vectorLength * sizeof(int));
   int * b = (int *) malloc(vectorLength * sizeof(int));
   ull s = 0;
   double startTime;
   double endTime;

   for (int i = 0; i < vectorLength; i++)
   {
      a[i] = (i * 7) % 17;
      b[i] = (vectorLength - i) * 11 % 13;
   }

   MPI_Init(&argc, &argv);
   startTime = MPI_Wtime();
   s = countInVectors(a, b, vectorLength, vectorLength);
   endTime = MPI_Wtime();

   printf("%llu, %lf\n", s, endTime - startTime);

   MPI_Finalize();
   return(0);
```

```
}
```

Приложение 2. Исходный код параллельной программы с типом связи «точка-точка» (main_dot.c)

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

#include "mpi.h"

typedef unsigned long long ull;

static const short FIRST_VECTOR_PART_MPI_ID = 0;
static const short SECOND_VECTOR_MPI_ID     = 1;
static const short ANSWER_MPI_ID            = 2;

ull countInVectors(int * a, int * b, int a_size, int b_size)
{
   ull forRet = 0;
   for (int i = 0; i < a_size; i++)
   {
      for (int j = 0; j < b_size; j++)
      {
         forRet += a[i] * b[j];
      }
   }
   return forRet;
}

int main(int argc, char * argv[])
{
   int vectorLength = argc > 1 ? atoi(argv[1]) : 145000;
   int vectorFictiousLength;
   int mpiSize;
   int mpiRank;
   int segmentLength;
   int * a;
   int * b;
   ull s = 0;

   MPI_Status status;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &mpiSize);
   MPI_Comm_rank(MPI_COMM_WORLD, &mpiRank);

   segmentLength = (vectorLength / mpiSize) + 1;
   vectorFictiousLength = segmentLength * mpiSize;

   if (mpiRank == 0)
   {
      a = (int *) malloc(vectorFictiousLength * sizeof(int));
```

```
        b = (int *) malloc(vectorLength * sizeof(int));
        for (int i = 0; i < vectorLength; i++)
        {
            a[i] = (i * 7) % 17;
            b[i] = (vectorLength - i) * 11 % 13;
        }
        for (int i = vectorLength; i < vectorFictiousLength; i++)
        {
            a[i] = 0;
        }

        double startTime = MPI_Wtime();

        for (int i = 1; i < mpiSize; i++)
        {
            MPI_Send(a + i * segmentLength, segmentLength, MPI_INT, i,
FIRST_VECTOR_PART_MPI_ID, MPI_COMM_WORLD);
            MPI_Send(b, vectorLength, MPI_INT, i, SECOND_VECTOR_MPI_ID,
MPI_COMM_WORLD);
        }

        s = countInVectors(a, b, segmentLength, vectorLength);

        ull temp;
        for (int i = 1; i < mpiSize; i++)
        {
            MPI_Recv(&temp, 1, MPI_UNSIGNED_LONG_LONG, i, ANSWER_MPI_ID,
MPI_COMM_WORLD, &status);
            s += temp;
        }

        printf("%llu %lf\n", s, MPI_Wtime() - startTime);
    }
    else
    {
        a = (int *) malloc(segmentLength * sizeof(int));
        b = (int *) malloc(vectorLength  * sizeof(int));

        MPI_Recv(a, segmentLength, MPI_INT, 0, FIRST_VECTOR_PART_MPI_ID,
MPI_COMM_WORLD, &status);
        MPI_Recv(b, vectorLength,  MPI_INT, 0, SECOND_VECTOR_MPI_ID,
MPI_COMM_WORLD, &status);

        s = countInVectors(a, b, segmentLength, vectorLength);
        MPI_Send(&s, 1, MPI_UNSIGNED_LONG_LONG, 0, ANSWER_MPI_ID,
MPI_COMM_WORLD);
    }
    free(a);
    free(b);
    MPI_Finalize();
    return 0;
}
```

Приложение 3. Исходный код программы с коллективной связью (main_collective.c)

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

#include "mpi.h"

typedef unsigned long long ull;

ull countInVectors(int * a, int * b, int a_size, int b_size)
{
   ull forRet = 0;
   for (int i = 0; i < a_size; i++)
   {
      for (int j = 0; j < b_size; j++)
      {
         forRet += a[i] * b[j];
      }
   }
   return forRet;
}


int main(int argc, char * argv[])
{
   int vectorLength = argc > 1 ? atoi(argv[1]) : 145000;
   int vectorFictiousLength;
   int mpiSize;
   int mpiRank;
   int segmentLength;
   int * a;
   int * b;
   int * aPartCopy;
   ull localAnswer = 0;
   ull answer = 0;

   MPI_Status status;

   double startTime;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &mpiSize);
   MPI_Comm_rank(MPI_COMM_WORLD, &mpiRank);

   segmentLength = (vectorLength / mpiSize) + 1;
   vectorFictiousLength = segmentLength * mpiSize;

   b = (int *) malloc(vectorLength  * sizeof(int));

   if (mpiRank == 0)
```

```
    {
        a = (int *) malloc(vectorFictiousLength * sizeof(int));
        for (int i = 0; i < vectorLength; i++)
        {
            a[i] = (i * 7) % 17;
            b[i] = ((vectorLength - i) * 11) % 13;
        }
        for (int i = vectorLength; i < vectorFictiousLength; i++)
        {
            a[i] = 0;
        }
        startTime = MPI_Wtime();
    }

    aPartCopy = (int *) malloc(segmentLength * sizeof(int));

    MPI_Scatter(a, segmentLength, MPI_INT, aPartCopy, segmentLength, MPI_INT, 0,
MPI_COMM_WORLD);
    MPI_Bcast  (b, vectorLength,  MPI_INT, 0, MPI_COMM_WORLD);

    localAnswer = countInVectors(aPartCopy, b, segmentLength, vectorLength);

    MPI_Reduce(&localAnswer, &answer, 1, MPI_UNSIGNED_LONG_LONG, MPI_SUM, 0,
MPI_COMM_WORLD);


    if (mpiRank == 0)
    {
        printf("%llu %lf\n", answer, MPI_Wtime() - startTime);
        free(a);
    }

    free(aPartCopy);
    free(b);
    MPI_Finalize();
    return 0;
}
```