МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №01-02

по дисциплине «Параллельные алгоритмы»

Тема: Использование функций обмена данными «точка-точка» в библиотеке MPI

Студент гр. 8303 _____ Почаев Н.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2020

Задание.

Л.Р. 00 Запуск параллельной программы на различном числе одновременно работающих процессов, упорядочение вывода результатов.

Задание: запустить программу на $1, 2, \dots, N$ процессах несколько раз. Проанализировать порядок вывода сообщений на экран. Вывести правило, определяющее порядок вывода сообщений. Модифицировать программу таким образом, чтобы порядок вывода сообщений на экран соответствовал номеру соответствующего процесса.

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

int main(int argc, char* argv[]){
     int ProcNum, ProcRank, RecvRank;
     MPI_Status Status;
     MPI_Init(&argc, &argv);

     MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
     MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

     if ( ProcRank == 0 ){
          //Действия, выполняемые только процессом с рангом 0
          printf ("\n Hello from process %3d", ProcRank);
          for ( int i=1; i<ProcNum; i++ ) {
          MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
          MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
          printf("\n Hello from process %3d", RecvRank); }
     } else // Сообщение, отправляемое всеми процессами,
     // кроме процесса с рангом 0
     MPI_Send(&ProcRank,1,MPI_INT,0,0, MPI_COMM_WORLD);
     MPI_Finalize();
     return 0;
}
```

Л.Р. 01 Обмен сообщениями чётных и нечётных процессов.

Напишите программу обмена сообщениями чётных и нечётных процессов. Замерьте время на одну итерацию обмена и определите зависимость времени обмена от длины сообщения.

Описание решения.

Л.Р. 01

Порядок вывода в изначальной программе: первым всегда идёт ответ от 0 процесса, а затем в случайном порядке от всех остальных. Связано это с тем, что

всегда мы сначала попадём в ветку `if (ProcRank == 0)` и выведем приветственное сообщение от него. Далее, нулевой процесс будет обрабатывать все остальные процессы MPI_ANY_SOURCE в ходе их завершения.

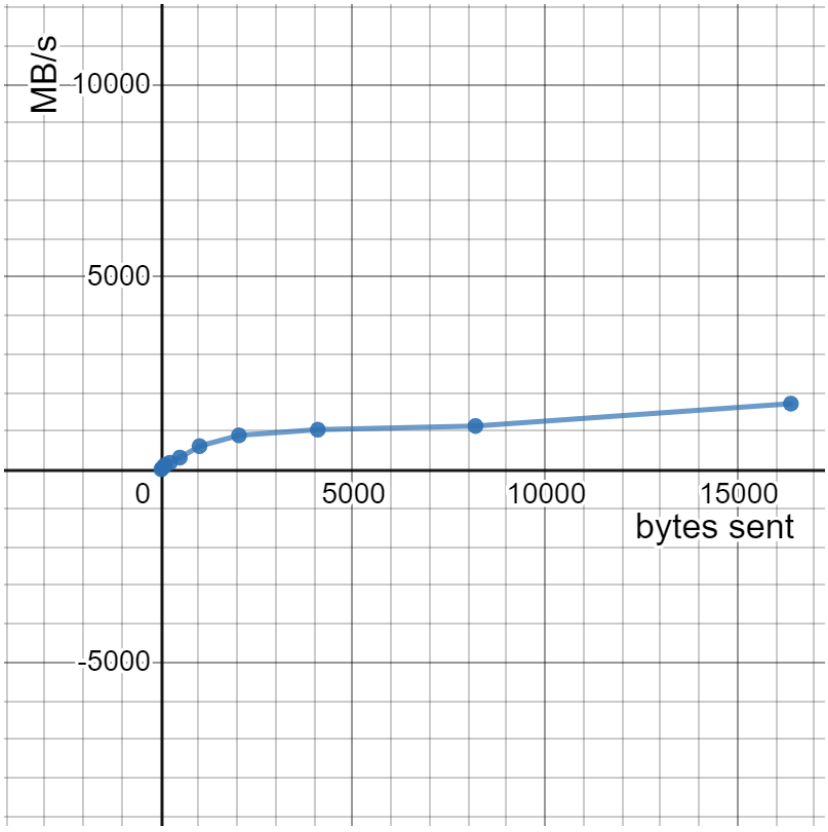
Модификация программы для вывода сообщений в соответствии с номером процесса: `MPI_Recv(&RecvRank, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &Status);` - на каждой итерации цикла обрабатываем соответствующий процесс.

Л.Р. 02

Прикладываю две программы: even-odd.c - обмен сообщениями между чётными и нечётными процессами. measure.c - программа, моделирующая последовательный обмен сообщениями между двумя процессами с замером времени. В данном случае определяется зависимость времени обмена от длины передаваемых данных.

График и ссылку на его интерактивную версию с таблицей, содержащую таблицу с полученными данными, прилагаю.

Из полученных результатов можно сделать вывод, что есть прямая зависимость времени от длины сообщения, коррелирующая, однако, с ограниченным ростом по закону Амдала.



Выводы.

В ходе выполнения лабораторной работы лабораторной работы были изучены основы работы с API MPI и выполнены замеры производительности для созданной программы обмена сообщениями между чётными и нечётными сообщениями.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ EVEN-ODD.

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

int main(int argc, char **argv) {
   int rank, size, prev, next;
   int buf[2];
   MPI_Request reqs[2];
   MPI_Status stats[2];

   MPI_Init(&argc,&argv);
   double start = MPI_Wtime();

   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   prev = rank - 1;
   next = rank + 1;

   if(rank == 0) prev = size - 1;
   if(rank == size - 1) next = 0;

   // sendbuf, sendcount, sendtype, dest, sendtag, recvbuf,
recvcount, recvtype, source, recvtag, comm, status
   MPI_Sendrecv(&rank, 1, MPI_INT, next, 5, &buf[0], 1, MPI_INT,
prev, 5, MPI_COMM_WORLD, &stats[0]);
   printf("process %d got '%d', send '%d' to %d\n", rank, buf[0],
rank, next);

   double end = MPI_Wtime();
   MPI_Finalize();

   printf("The process took %f seconds to run.\n", end - start);

   return 0;
}
```

ПРИЛОЖЕНИЕ Б.

ИСХОДНЫЙ КОД ПРОГРАММЫ MEASURE.

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

int main(int argc,char * argv[]) {
  int ierr, rank, size, i, n, nmax, lmax, NTIMES = 10;
  //const int NMAX = 1000000;
  const int NMAX = 1000;
  double time_start,time,bandwidth,max;
  float a[NMAX*8]; // equal to real*8  a[NTIMES] at fortran
  struct MPI_Status status;
  ierr = MPI_Init(&argc,&argv);
  // ierr=MPI_Comm_size(MPI_COMM_WORLD, &size);
  ierr = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  // time_start=MPI_Wtime();
  n = 0;
  nmax = lmax = 0;
  while (n <= NMAX) {
       time_start=MPI_Wtime();
       for (i=0; i < NTIMES; ++i) {
            if (rank == 0) {
                 ierr   =   MPI_Send(&a,8*n,MPI_FLOAT,   1,   1,
MPI_COMM_WORLD) ;
                 ierr  =  MPI_Recv(&a,  8*n,  MPI_FLOAT,  1,  1,
MPI_COMM_WORLD, &status) ;
            } else if (rank == 1) {
                 ierr=MPI_Recv(&a,   8*n,   MPI_FLOAT,   0,   1,
MPI_COMM_WORLD, &status) ;
                 ierr=MPI_Send(&a,   8*n,   MPI_FLOAT,   0,   1,
MPI_COMM_WORLD);
            }
        }

       time = (MPI_Wtime() - time_start) / (2 * NTIMES); // this
is time for one way transac-tion
       bandwidth = ((double)8 * n * sizeof(float))/ (1024 * 1024)
/ time;
       if (max < bandwidth) {
            max=bandwidth;
            lmax = 8 * n * sizeof(float);
        }
       if (rank == 0) {
            if (!n) {
                printf("Latency = %10.4f seconds\n", time);
            } else {
                    printf("%li bytes sent, bandwidth = %10.4f
MB/s\n", 8 * n * sizeof(float), bandwidth);
                  //   printf("(%li;   %10.4f)\n",   8*n*sizeof
(float), bandwidth);
```

```
                    }
                }
            if (n == 0) {
                n = 1;
            } else {
                n *= 2;
            }
        }
        // Finally print maximum bandwidth
        if (rank==0) {
            printf("Max bandwidth = %10.4f MB/s, length = %i bytes\n",
    max, lmax);
        }
        ierr = MPI_Finalize();

        return 0;
}
```