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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Параллельные алгоритмы».

Тема: ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ ОБМЕНА ДАННЫМИ «ТОЧКА-ТОЧКА» В БИБЛИОТЕКЕ MPI.

Студентка гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2022

Задание.

Вариант 9.

Написать программу обмена сообщениями «point to point» процессов.

Игра с ведущим (съедобное − несъедобное). Процесс 0 поочередно посылает остальным процессам сообщение. Получив сообщение, процесс-приемник информирует об этом процесс 0.

Листинг программы см. в Приложении А.

Выполнение работы.

В программе осуществляется обмен сообщениями между процессами со следующей логикой. Запускается проверка на номер процесса, и если результат нулевой, то запускает цикл, проходящийся по остальным процессам (i = 1 : size, где size выявляется с помощью MPI_Comm_size), каждому их которых на соответствующей итерации сначала отправляется сообщение (MPI_Isend from 0 to i), затем нулевой процесс переходит в режим приёма ответа (MPI_Recv from i to 0). Если номер процесса больше нуля, то осуществляется приём сообщения от нулевого процесса (MPI_Recv from 0 to rank, где rank инициализируется с помощью MPI_Comm_rank) и отправка того же сообщения назад (MPI_Isend from rank to 0).

Для отправки сообщения используется стандартная функция MPI_Isend, так как это неблокирующая отправка, позволяющая осуществить обмен сообщениями с условиями «съедобное − несъедобное», так как вызов функции возвращает управление к программе и можно задействовать также функцию MPI_Recv.

Сообщение формируются заранее, под них освобождается количество памяти, соизмеримое с заданной длиной *(int* message = (int*)(calloc(MAX_LENGTH, sizeof(int))),* где MAX_LENGTH – константа, которой предстоит изменение для составления графиков зависимости времени выполнения программы от числа процессов для разных длин пересылаемых сообщений.

Результаты работы программы на 1,2 …. N процессорах.

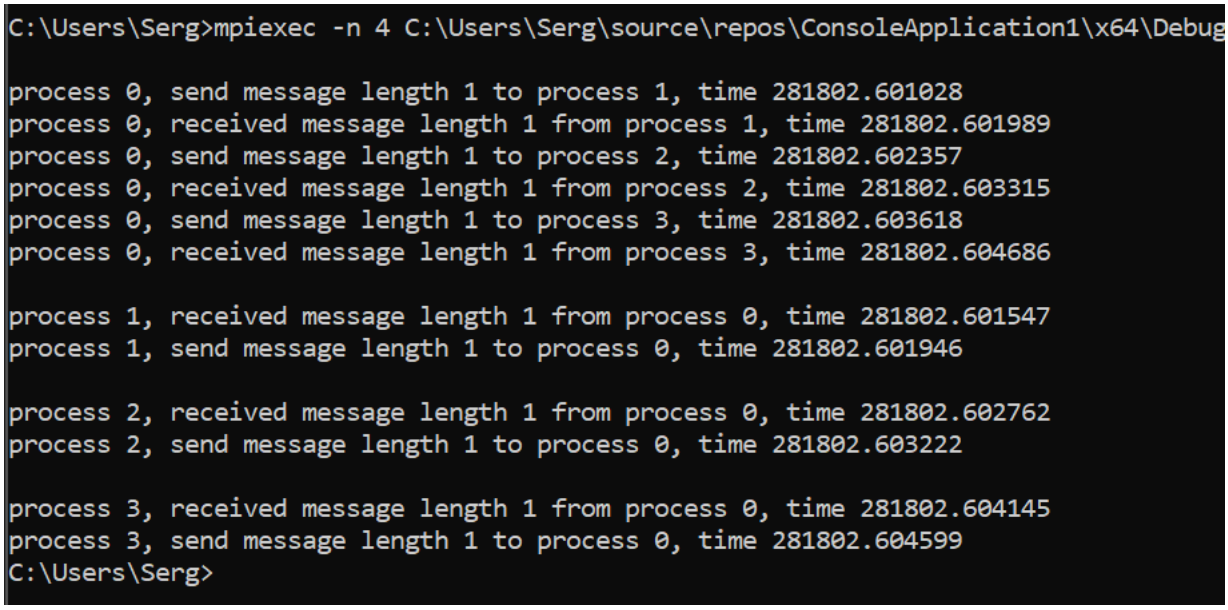

Рис. 1 - Результаты работы программы на 4 процессорах.

Восстановим порядок выполняемых действий с помощью полученного времени выполнения команды.

Номер выполнения команды.	Путь сообщения (Приём <-, отправка ->)	Текущее время команды.
1.	0 -> 1	281802.601028
2.	1 <- 0	281802.601547
3.	1 -> 0	281802.601946
4.	0 <- 1	281802.601989
5.	0 -> 2	281802.602357
6.	2 -< 0	281802.602762
7.	2 -> 0	281802.603222
8.	0 <- 2	281802.603315
9.	0 -> 3	281802.603618
10.	3 <- 0	281802.604145
11.	3 -> 0	281802.604599
12.	0 <- 3	281802.604686

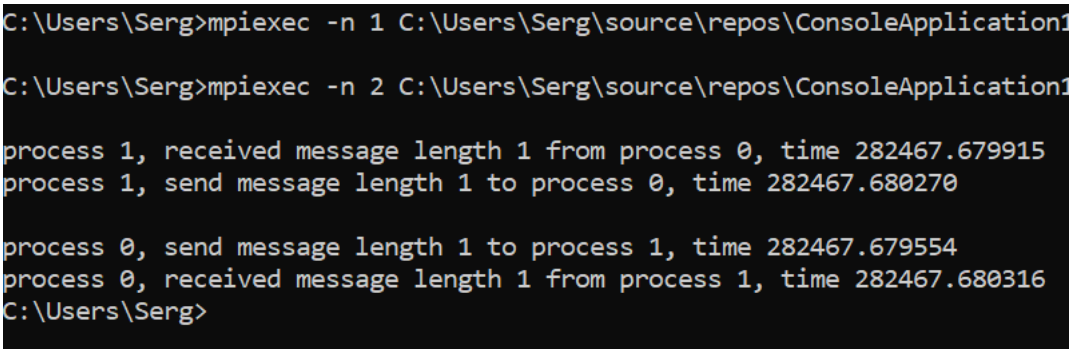

Рис. 2 - Результаты работы программы на 1,2 процессорах.

При запуске на одном процессоре не происходит никакой отправки, так как в программе стоит ограничение на количество участвующих в обмене процессов. При запуске на минимальном возможном (2) отправка и приём сообщения осуществляется корректно.

График зависимости времени выполнения программы от числа процессов для разных длин пересылаемых сообщений.

Для получения экспериментальных данных модифицируем программу так, чтобы выводились только нужные для построения графика данные: время работы программы на каждом процессоре и длина обмениваемого сообщения.

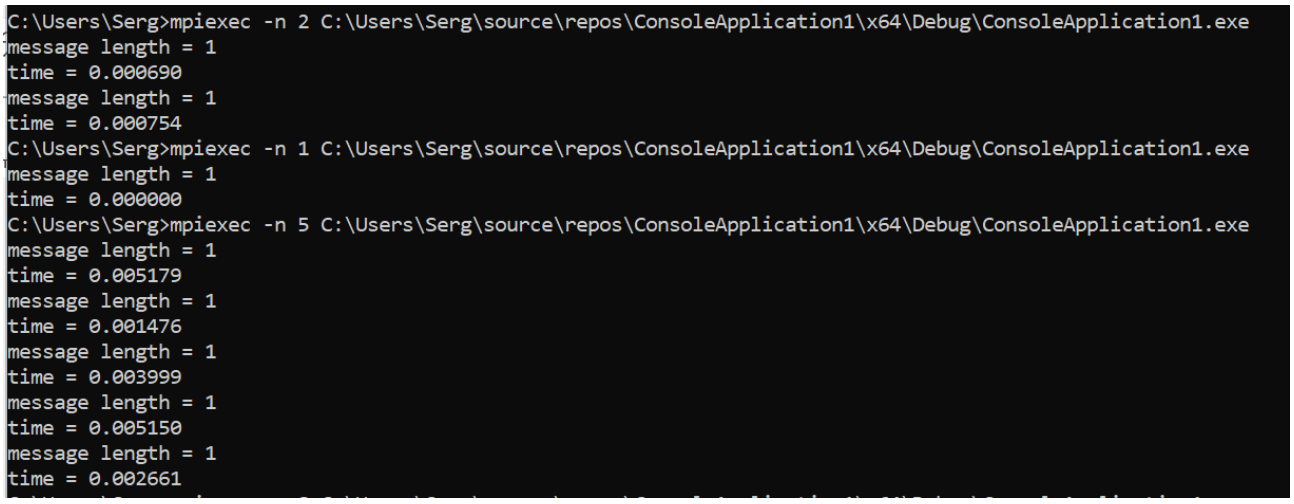

Рис. 3 – Пример выполнения модифицируемой программы на 1, 2, 5 процессах длины 1.

Для выявления зависимости получим средние значения времени каждого запуска программы с различной длиной сообщения и получим данные:

Номер процесса	Время выполнения программы, с	Длина сообщения		Номер процесса	Время выполнения программы, с	Длина сообщения		Номер процесса	Время выполнения программы, с	Длина сообщения
0	0	1		0	0	100		0	0	10000
Среднее	0	1		Среднее	0	100		Среднее	0	10000
0	0,00069	1		0	0,001001	100		0	0,001248	10000
1	0,000754	1		1	0,001066	100		1	0,001379	10000
Среднее	0,000722	1		Среднее	0,0010335	100		Среднее	0,0013135	10000
0	0,005179	1		0	0,002932	100		0	0,004087	10000
1	0,001476	1		1	0,00222	100		1	0,00415	10000
2	0,003999	1		2	0,003424	100		2	0,00143	10000
3	0,00515	1		3	0,003373	100		3	0,003127	10000
4	0,002661	1		4	0,001116	100		4	0,002418	10000
Среднее	0,003693	1		Среднее	0,002613	100		Среднее	0,0030424	10000
0	0,003528	1		0	0,004551	100		0	0,00547	10000
1	0,004771	1		1	0,002784	100		1	0,004694	10000
2	0,002431	1		2	0,000735	100		2	0,003534	10000
3	0,005573	1		3	0,002147	100		3	0,007922	10000
4	0,008289	1		4	0,001407	100		4	0,002203	10000
5	0,006673	1		5	0,003559	100		5	0,006558	10000
6	0,000982	1		6	0,00535	100		6	0,007381	10000
7	0,008495	1		7	0,006895	100		7	0,008516	10000
8	0,008058	1		8	0,005961	100		8	0,008506	10000
9	0,007206	1		9	0,006917	100		9	0,001065	10000
Среднее	0,0056006	1		Среднее	0,0040306	100		Среднее	0,0055849	10000

Рис.4 – Экспериментальные данные запусков программы на 1,2,5,10 процессах с различной длиной сообщения.

Число процессов	Длина сообщения		
	1	100	10000
1	0	0	0
2	0,00072	0,00103	0,00131
5	0,00369	0,00261	0,00304
10	0,0056	0,00403	0,00558

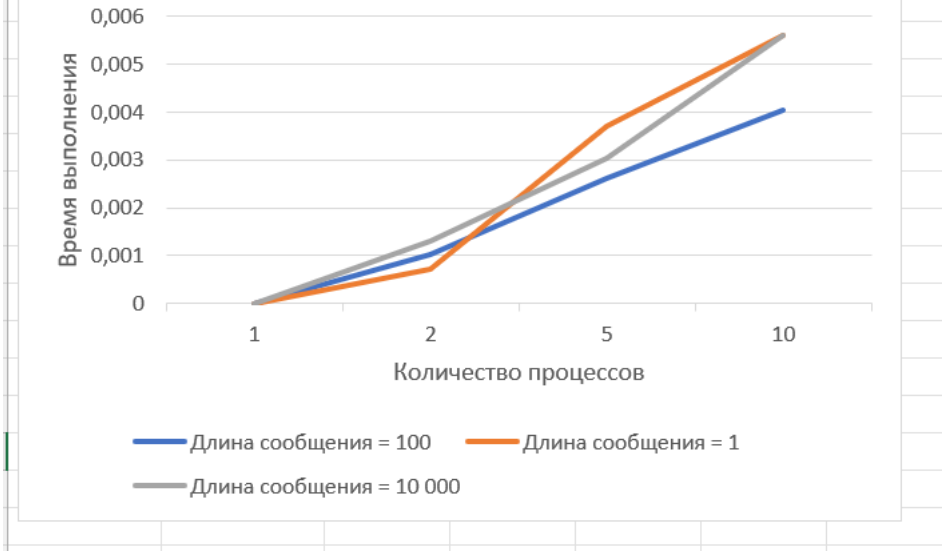

Рис.5 – Графики зависимости времени выполнения программы от числа процессов для разных длин пересылаемых сообщений (и данные для их построения).

Следовательно, выполняемое время напрямую зависит от количества зависимых процессов: чем больше процессов, тем дольше идёт работа с обменами сообщением. Это происходит, потому что чем больше процессов, тем

больше работы для программы: повышается количество выполняемых действий (отправок и приёмов сообщения).

Сети Петри.

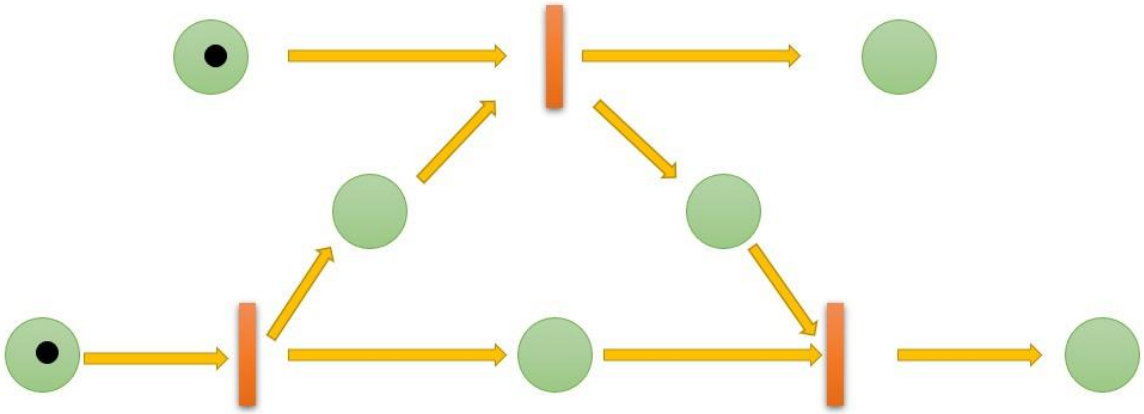

Рис.6 – Сети Петри.

Выводы.

Написана программа обмена сообщениями между процессами в режиме «точка-точка».

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <mpi.h>
#include <cstdlib>
#define MAX_LENGTH 1

int main(int argc, char* argv[])
{
  int size, rank;
  MPI_Status status;
  int* message = (int*)(calloc(MAX_LENGTH, sizeof(int)));
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Request req;
  if ( rank == 0 && rank < size - 1) {
    for (int i = 1; i < size; i++) {
      MPI_Isend(message, MAX_LENGTH, MPI_INT, i,
        0, MPI_COMM_WORLD, &req);
      printf("\nprocess %d, received message length %d from process %d,
time %f", rank, MAX_LENGTH, i, MPI_Wtime());

      MPI_Recv(message, MAX_LENGTH, MPI_INT, i, 0, MPI_COMM_WORLD,
        &status);
      printf("\nprocess %d, received message length %d from process %d,
time %f", rank, MAX_LENGTH, i, MPI_Wtime());

    }
  }
  else if (rank > 0) {
    MPI_Recv(message, MAX_LENGTH, MPI_INT, 0, 0, MPI_COMM_WORLD,
      &status);
    printf("\nprocess %d, received message length %d from process %d,
time %f", rank, MAX_LENGTH, 0, MPI_Wtime());
    MPI_Isend(message, MAX_LENGTH, MPI_INT, 0,
      0, MPI_COMM_WORLD, &req);
    printf("\nprocess %d, send message length %d to process %d,
time %f", rank, MAX_LENGTH, 0, MPI_Wtime());

  }

  MPI_Finalize();
```

```
    return 0;
}
```