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

ОТЧЕТ

по практической работе №4
по дисциплине «Системы параллельной обработки данных»
Тема: Коллективные операции.

Студент гр. 0303 _____ Болкунов В.О.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2024

Цель работы.

Исследование и разработка параллельной программы, использующей обмен сообщениями между процессами от дочерних к главному с помощью групповой функции Gather.

Постановка задачи.

Вариант 1.

В каждом процессе дано вещественное число. Используя функцию MPI_Gather, переслать эти числа в главный процесс и вывести их в порядке возрастания рангов переславших их процессов (первым вывести число, данное в главном процессе).

Выполнение работы.

Разработана параллельно работающая программа с использованием библиотеки MPI, работающая по следующему алгоритму:

1. Каждый процесс определяет своё вещественное число, вычисляемое по формуле $D_i = 3.0 + \frac{i}{10}$, где i - ранг процесса.
2. Главный процесс аллоцирует массив размера равным количеству процессов в коммуникаторе для принятия сообщений.
3. Все процессы запускают метод ***Gather*** коммуникатора ***World***, передавая своё число в качестве сообщения.
4. Главный процесс принимает сообщения с помощью идентичного метода и выводит полученный массив. Так как нулевой процесс взят в качестве главного, его сообщение всегда будет первым, так как функция ***Gather*** копирует сообщения в принимаемый буфер последовательно согласно рангу отправляющего процесса.

Сеть петри с 3 процессами для данного алгоритма представлена на рисунке 1.

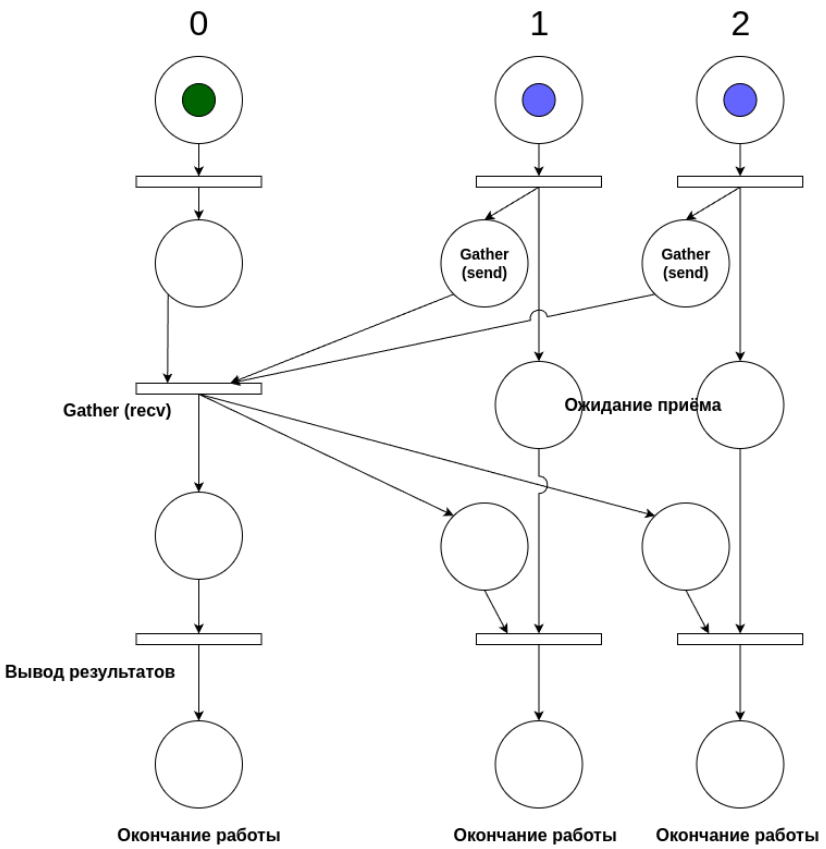

Рисунок 1. Сеть петри

Пример работы программы с 6 процессами представлен на рисунке 2. Исходный код программы представлен в приложении А.

```
vlad@vlad-pc:~/projects/SPOD/lab4$ ./run.sh 6
[0] -> 3
[1] -> 3.1
[2] -> 3.2
[3] -> 3.3
[4] -> 3.4
[5] -> 3.5
[0] time: 0.000103617s.
```

Рисунок 2. Пример работы программы

На данном примере видно, что сообщения принялись по порядку рангов процессов отправивших сообщение.

Программа была запущена на разном количестве процессоров: от 2 до 32 с шагом 2. Время работы программы в данной задаче соответствует времени работы главного процесса.

На рисунке 3 представлен график зависимости времени работы программы от количества задействованных процессоров. На рисунке 4 представлен график ускорения.

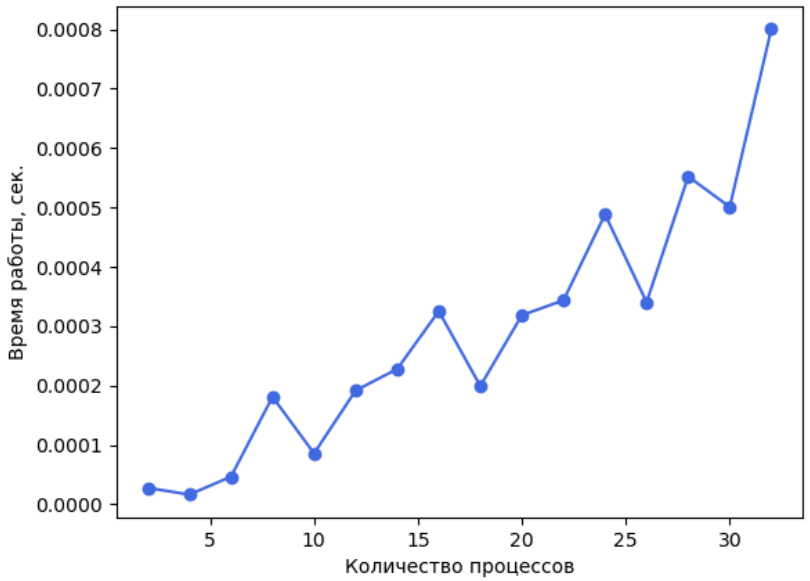

Рисунок 3. График времени работы программы

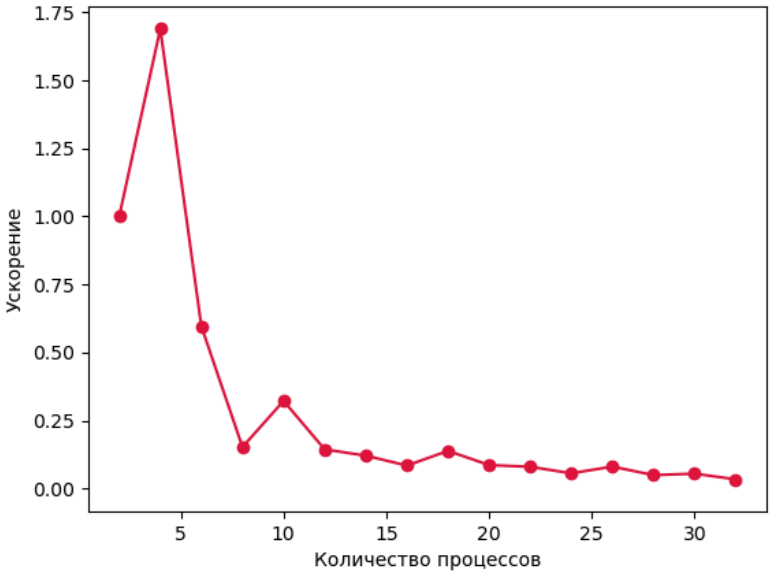

Рисунок 4. График ускорения программы

По данным графикам видно, что добавление дополнительных процессов замедляет работу программы, так в данной задаче крайне малая часть параллельно выполняемого кода и большая часть времени работы программы уходит на отправку сообщений процессами, которое также растёт при увеличении количества процессов.

Выводы.

В результате выполнения работы была разработана параллельная программа, использующая групповую функцию Gather для отправки сообщений главному процессу. Разработанная программа запущена на разном количестве процессоров: от 2 до 32. В результате тестирования программы построены графики времени работы программы и ускорения.

Увеличение количество процессов привело к замедлению работы программы, так как в решаемой задаче большую часть времени работы программы занимает само время отправки сообщения.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл main.cpp

```
#include <mpi.h>
#include <array>
#include <ctime>
#include <iostream>
#include <vector>

int main(int argc, char **argv) {
 MPI::Init(argc, argv);
 MPI::Comm &comm = MPI::COMM_WORLD;

 const int rootProc = 0;
 const int procRank = comm.Get_rank();
 const int procCount = comm.Get_size();

 const double procPrivateNumber = 3. + procRank / 10.;

 double time = 0, start, end;

 if (procRank == rootProc) {
   double recvBuffer[procCount];

   start = MPI::Wtime();
   comm.Gather(&procPrivateNumber, 1, MPI::DOUBLE, recvBuffer, 1, MPI::DOUBLE,
               rootProc);
   end = MPI::Wtime();
   time += end - start;

   for (size_t i = 0; i < procCount; i++) {
    std::cout << "[" << i << "] -> " << recvBuffer[i] << std::endl;
   }

   std::cout << "[" << procRank << "]" << " time: " << time << "s."
             << std::endl;
 } else {
   comm.Gather(&procPrivateNumber, 1, MPI::DOUBLE, nullptr, 0, MPI::DOUBLE,
             rootProc);
 }

 MPI::Finalize();
}
```

Файл build.sh

```
mpicxx.openmpi main.cpp -o lab4
```

Файл run.sh

```
mpiexec.openmpi -n $1 --oversubscribe ./lab4 $2
```