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

ОТЧЕТ

по практической работе №5
по дисциплине «Системы параллельной обработки данных»
Тема: Группы процессов и коммуникаторы.

Студент гр. 0303 _____ Болкунов В.О.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2024

Цель работы.

Исследование и разработка параллельной программы, использующей функции построения виртуальной группы процессов и обмен сообщениями внутри построенной группы.

Постановка задачи.

Вариант 11.

В каждом процессе дано целое число N, которое может принимать два значения: 0 и 1 (имеется хотя бы один процесс с N = 1). Кроме того, в каждом процессе с N = 1 дано вещественное число A. Используя функцию MPI_Comm_split и одну коллективную операцию редукции, найти сумму всех исходных чисел A и вывести ее во всех процессах с N = 1.

Указание. При вызове функции MPI_Comm_split в процессах, которые не требуется включать в новый коммуникатор, в качестве параметра color следует указывать константу MPI_UNDEFINED.

Выполнение работы.

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

1. Каждый процесс определяет число N, которое вычисляется по формуле $N = (p + 1)\, mod\, 2$, где p - ранг процесса в глобальной группе. Таким образом всегда будет хотя бы 1 процесс с N = 1.
2. В каждом процессе вызывается функция **Split** у глобального коммуникатора, для процессов с N = 1 цвет выбран равным числу **1.** Для остальных процессов цвет равен MPI::UNDEFINED. Таким образом процессы с N = 1 объединяются в одну группу
3. Далее внутри построенной группы процессов формируется вещественное число $A = 3.0 + 0.1 * p_G$, где p_G - ранг процесса внутри группы.

4. Внутри группы вызывается функция **Allreduce,** с операцией сложения, таким образом каждый процесс в группе получит результат суммы всех вещественных чисел в каждом процессе.

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

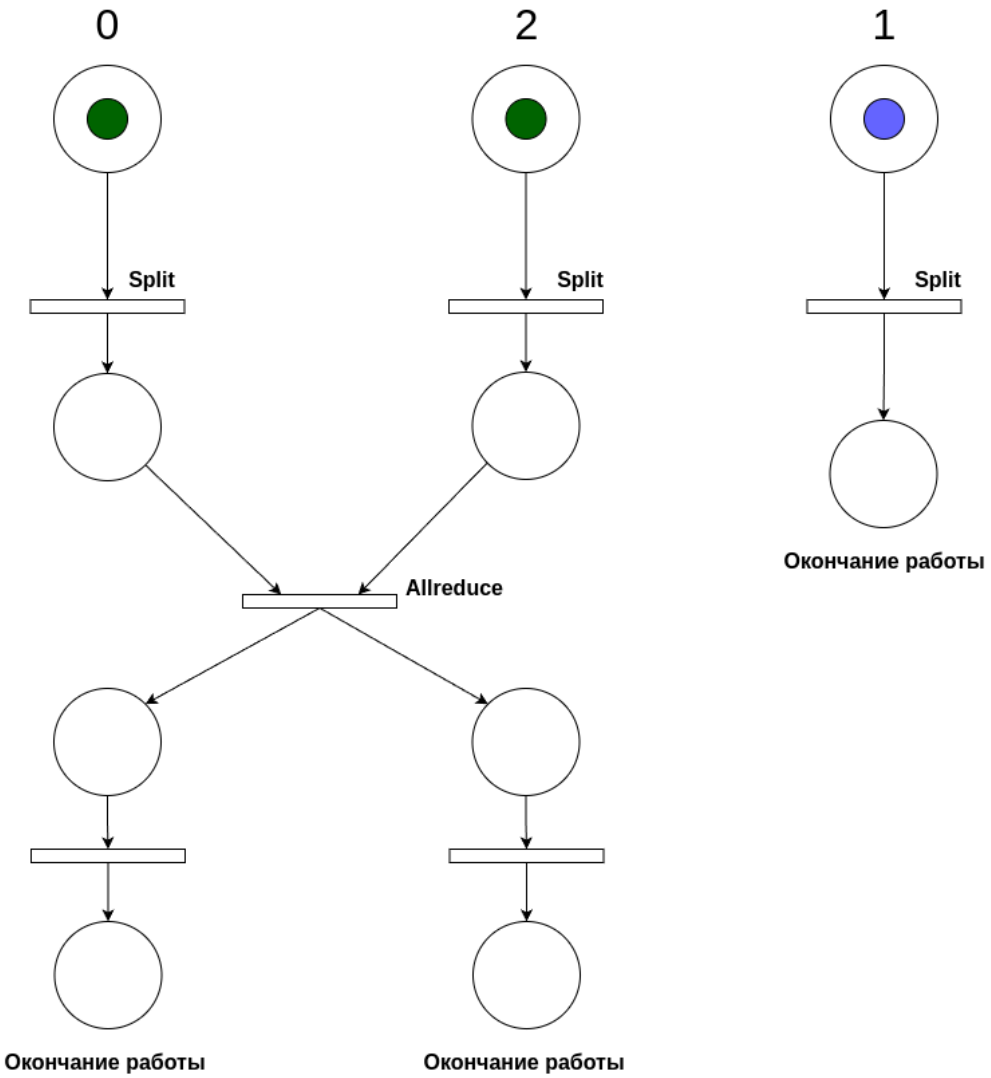

Рисунок 1. Сеть петри

Пример работы программы с 5 процессами представлен на рисунке 2. Исходный код программы представлен в приложении А.

```
vlad@vlad-pc:~/projects/SPOD/lab5$ ./run.sh 5
[4/2] A = 3.2
[4/2] Result = 9.3
[4/2] Time: 0.000145254s
[0/0] A = 3
[0/0] Result = 9.3
[0/0] Time: 0.000204672s
[2/1] A = 3.1
[2/1] Result = 9.3
[2/1] Time: 0.000184651s
```

Рисунок 2. Пример работы программы

В данном примере для каждого активного процесса показан ранг в глобальной группе и в созданной через черту. Видно, что каждый процесс определяет своё число и выводит результат суммирования.

Программа была запущена на разном количестве процессоров: от 1 до 21 с шагом 2. Время работы программы в данной задаче соответствует максимальному времени работы из всех процессов.

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

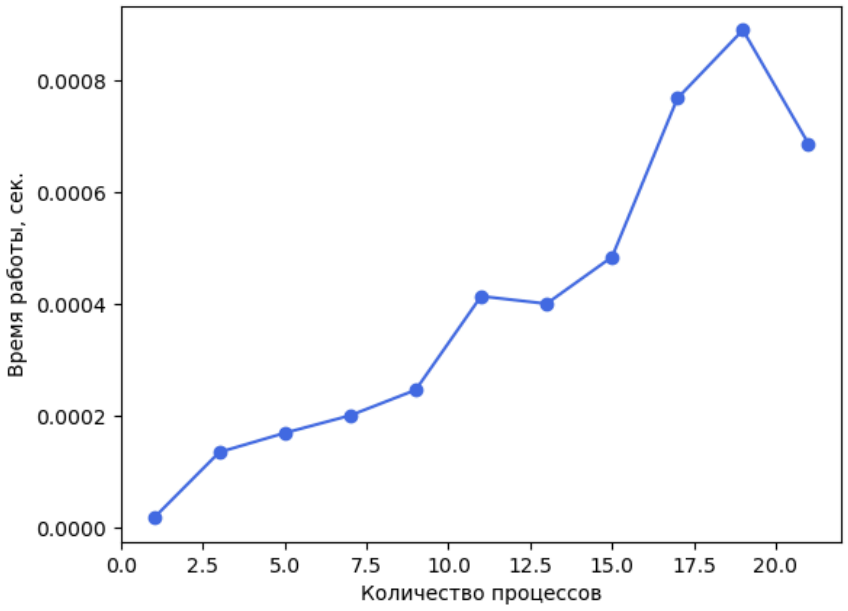

Рисунок 3. График времени работы программы

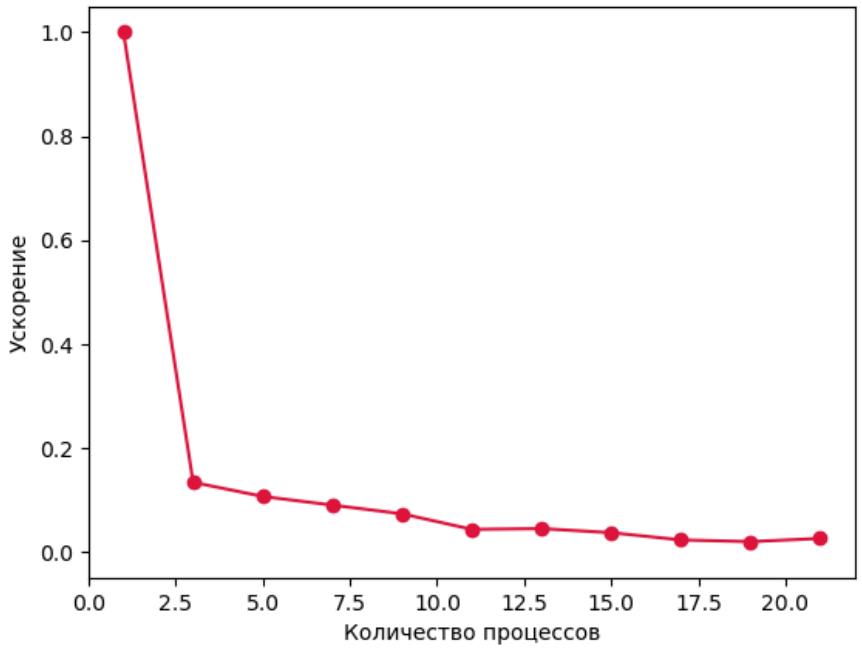

Рисунок 4. График ускорения программы

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 формирование группы процессов и применение групповой операции.

Выводы.

В результате выполнения работы была разработана параллельная программа, в которой создаётся новая группа процессов на основе глобальной группы, и применяется групповая операция редукции. Разработанная программа запущена на разном количестве процессоров: от 2 до 21. 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 время формирования группы процессов и время выполнения групповой операции.

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::Intracomm &worldComm = MPI::COMM_WORLD;

 double time = 0, start, end;

 const int procRank = worldComm.Get_rank();
 const int procCount = worldComm.Get_size();

 const int N = (procRank + 1) % 2;
 const int color = N == 1 ? 1 : MPI::UNDEFINED;

 start = MPI::Wtime();
 MPI::Intracomm groupComm = worldComm.Split(color, MPI::UNDEFINED);
 end = MPI::Wtime();
 time += end - start;

 if (N == 1) {
   const int groupRank = groupComm.Get_rank();
   const double A = 3.0 + 0.1 * groupRank;
   std::cout << "[" << procRank << "/" << groupRank << "] A = " << A
             << std::endl;
   double buf;

   start = MPI::Wtime();
   groupComm.Allreduce(&A, &buf, 1, MPI::DOUBLE, MPI::SUM);
   end = MPI::Wtime();
   time += end - start;

   std::cout << "[" << procRank << "/" << groupRank << "] Result = " << buf
             << std::endl;

   std::cout << "[" << procRank << "/" << groupRank << "] Time: " << time
             << "s" << std::endl;
 }

 MPI::Finalize();
}
```

Файл build.sh

```
mpicxx.openmpi main.cpp -o lab5
```

Файл run.sh

```
mpiexec.openmpi -n $1 --oversubscribe ./lab5
```