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

ОТЧЕТ

по практической работе №1
по дисциплине «Системы параллельной обработки данных»
Тема: Запуск параллельной программы на различном числе одновременно работающих процессов, упорядочение вывода результатов.

Студент гр. 0303 _____ Болкунов В.О.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2024

Цель работы.

Исследование и разработка параллельной программы на различном числе одновременно работающих процессов.

Постановка задачи.

1. Написать параллельную программу MPI, где каждый процесс определяет свой ранг, после чего действия в программе разделяются. Все процессы, кроме процесса с рангом 0 передают значение своего ранга нулевому процессу. Процесс с рангом 0 сначала печатает значение своего ранга, а далее последовательно принимает сообщения с рангами процессов и также печатает их значения. При этом важно отметить, что порядок приема сообщений заранее не определен и зависит от условий выполнения параллельной программы (более того, этот порядок может изменяться от запуска к запуску).
2. Запустить программу на 1,2 … N процессах несколько раз.
3. Проанализировать порядок вывода сообщений на экран. Вывести правило, определяющее порядок вывода сообщений.
4. Построить график времени работы программы в зависимости от числа запущенных процессов от 1 до 16. Размер шага – например, 4.
5. Построить график ускорения/замедления работы программы.
6. Модифицировать программу таким образом, чтобы порядок вывода сообщений на экран соответствовал номеру соответствующего процесса.
7. Нарисовать сеть Петри для двух вариантов MPI программы.

Выполнение работы.

Разработана параллельно работающая программа с использованием библиотеки MPI, где каждый процесс определяет свой ранг и передаёт сообщение главному процессу (с рангом 0), в котором указан номер процесса, отправившего сообщение. В главном процессе с рангом 0 определяется количество всех запущенных процессов N, и ожидается приём N - 1 сообщений

от любого процесса. Полученные сообщения записываются в подготовленный массив, после чего выводятся. Пример работы программы с 8 процессами представлен на рисунке 1. Исходный код программы представлен в приложении А.

Рисунок 1. Пример работы 1-го варианта программы

Независимо от количества процессов, при каждом запуске данной программы порядок принятия сообщений разный, так как главный процесс ожидает сообщение от любого процесса.

Программа была запущена на разном количестве процессоров: 1, и от 2 до 16 с шагом 2. Результаты времени работы программы представлены на рисунке 2. График ускорения (замедления) программы представлен на рисунке 3. Время работы программы измерялось на участке принятия сообщений главным процессом.

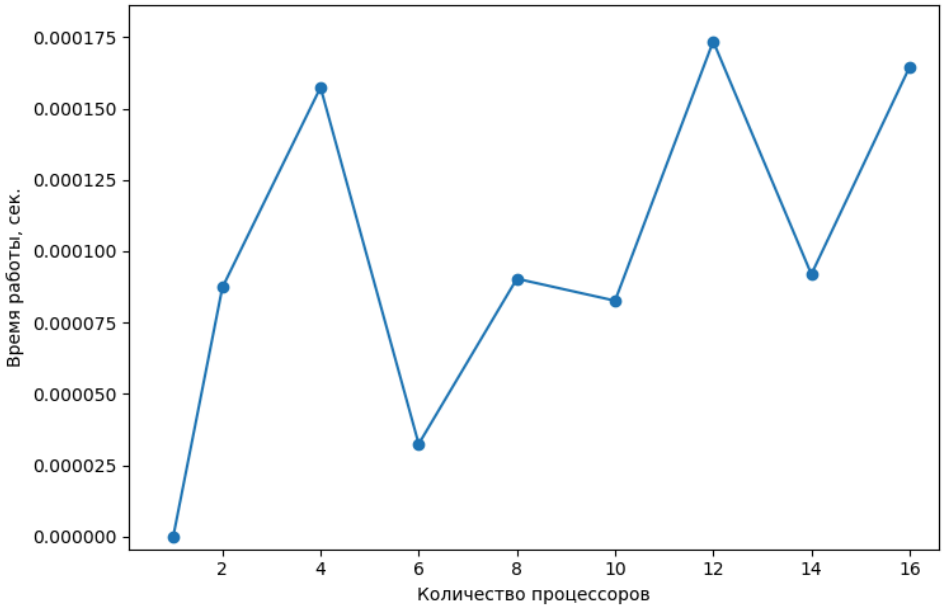

Рисунок 2. График времени работы программы

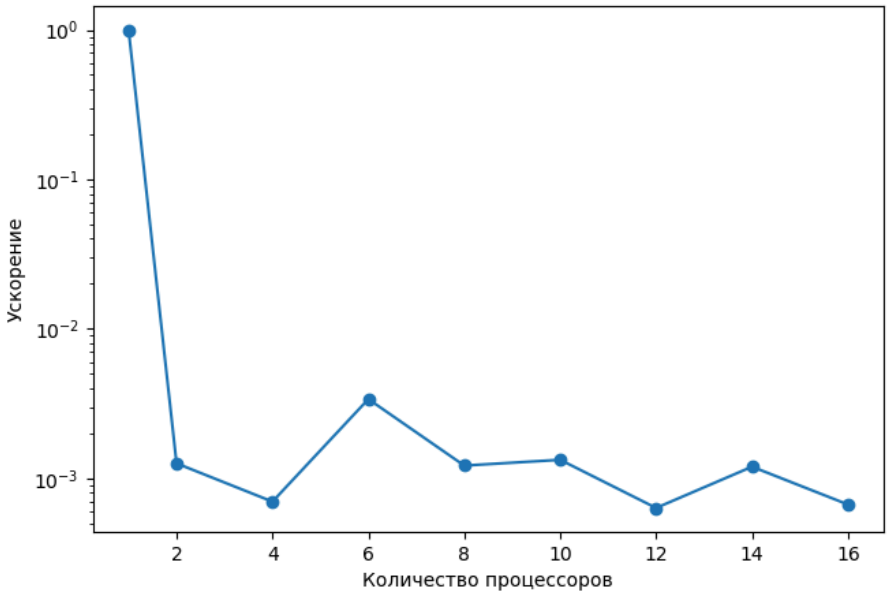

Рисунок 3. График замедления программы

По данным графикам видно, что при запуске программы на одном процессоре время работы минимальное, так как главный процесс не ожидает сообщений и сразу завершает свою работу. Дальнейшее увеличение количества процессоров замедляет работу программы.

Данная программа была модифицирована таким образом, чтобы главный процесс принимал сообщения по порядку номеров процессов-отправителей. Для этого был изменён фрагмент программы, в котором происходит принятие сообщений в главном процессе и в качестве источника указан порядковый номер процесса-отправителя. Применённое изменение представлено в листинге 1. Пример работы изменённой программы на 8 процессорах представлен на рисунке 4.

Листинг 1. Изменение фрагмента программы.

```
for (size_t i = 0; i < procCount - 1; i++) {
      comm.Recv(&messages[i], 1, MPI::INT, i + 1, MPI::ANY_TAG);
}
```

Рисунок 4. Пример работы 2-го варианта программы

Схема работы программы в виде сетей петри для трёх процессов представлены на рисунках 5-6 для 1го и 2го варианта программы соответственно.

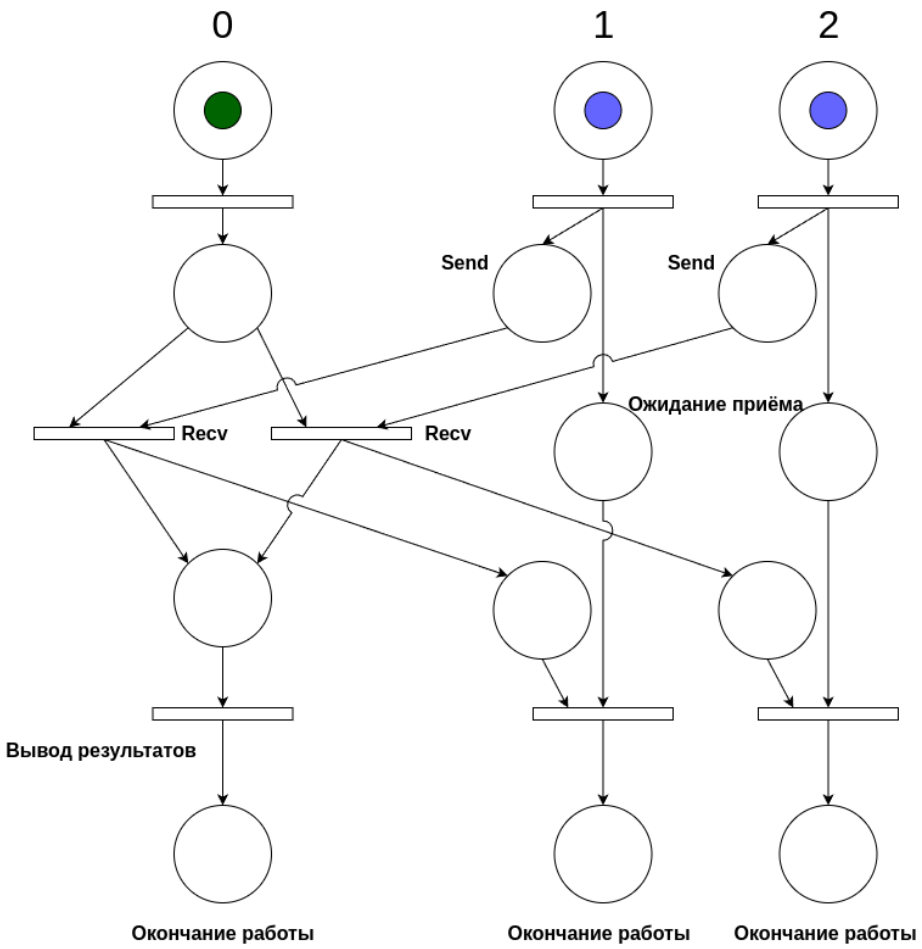

Рисунок 5. Сеть петри для первого варианта программы

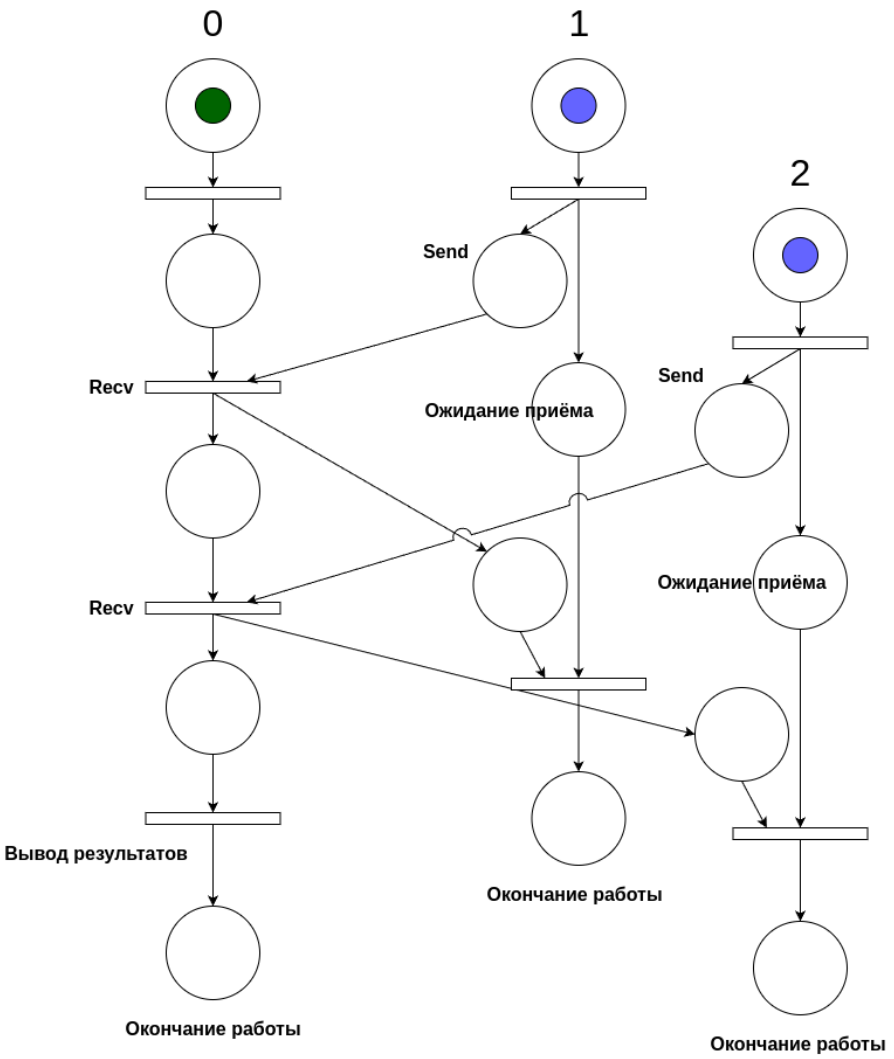

Рисунок 6. Сеть петри для второго варианта программы

Выводы.

В результате выполнения работы была разработана параллельная программа, использующая обмен сообщениями между процессами. Разработанная программа запущена на разном количестве процессоров. Построены графики времени работы программы и ускорения.

Данная программа была модифицирована таким образом, чтобы принимать сообщения в соответствии с порядком номеров процессов-отправителей. Для обоих вариантов программы построены сети петри.

При увеличении количества процессов время работы программы увеличивалось, так как главный процесс ожидает получения сообщений от всех остальных процессов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл main.cpp

```cpp
#include <mpi.h>
#include <iostream>
#include <vector>

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

 int procCount = comm.Get_size();
 int procRank = comm.Get_rank();

 if (procRank == 0) {
   std::cout << "Hello from process: " << procRank << std::endl;

   std::vector<int> messages(procCount - 1);

   double start = MPI::Wtime();

   for (size_t i = 0; i < procCount - 1; i++) {
     comm.Recv(&messages[i], 1, MPI::INT, MPI::ANY_SOURCE, MPI::ANY_TAG);
   }

   double end = MPI::Wtime();

   for (auto msg : messages) {
     std::cout << "\t Recieved hello from process: " << msg << std::endl;
   }

   std::cout << "Elapsed time: " << (end - start) << "s." << std::endl;
 } else {
   const int destProc = 0;
   comm.Send(&procRank, 1, MPI::INT, destProc, 0);
 }

 MPI::Finalize();
}
```

Файл build.sh

```
mpicxx.openmpi main.cpp -o lab1
```

Файл run.sh

```
mpiexec.openmpi -n $1 --oversubscribe ./lab1
```