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

ОТЧЕТ

по практической работе №3
по дисциплине «Системы параллельной обработки данных»
Тема: Использование аргументов-джокеров в библиотеке MPI.

Студент гр. 0303 _____ Болкунов В.О.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2024

Цель работы.

Исследование и разработка параллельной программы, использующей прямой обмен сообщениями между определёнными процессами с использованием джокеров.

Постановка задачи.

Вариант 5. Игра в снежки.

Процессы делятся на 2 группы (четные и нечетные номера). Каждый процесс нечетной группы случайным образом генерирует посылку четному процессу. Четный процесс, получив посылку, в свою очередь, отправляет случайно выбранному нечетному процессу следующую. Предусмотреть ситуацию получения одним процессом нескольких посылок.

Выполнение работы.

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

1. На вход программе подаётся общее требуемое количество сообщений которое будет отправлено из первой группы второй (нечётными чётным).
2. Нечётные процессы делят между собой требуемое количество сообщений для обмена по следующему принципу: общее количество сообщений делится на количество нечётных процессов и к этому числу прибавляется 0 или 1 в зависимости от индекса процесса в своей группе и остатка от деления исходного количества на количество нечётных процессов, итого: $n = \frac{N}{p} + (if\ [(N\ mod\ p)\ > i]\ then\ 1\ else\ 0)$, где N - общее количество сообщений, p - количество нечётных процессов и i - индекс нечётного процесса в своей группе (т.е. для процесса с рангом 1 - i = 0, с рангом 3 - i = 1, с рангом 5 - i = 2 и т.д.). Таким образом нечётные процессы равномерно распределяют между собой требуемое количество сообщений без обмена сообщениями между собой или ведущим процессом.
3. Нечётные процессы отправляют вычисленное количество сообщений случайным четным процессам (на каждое сообщение - новый случайный

получатель). Отправка сообщения в противоположную группу происходит с тегом 0.

4. Каждый чётный процесс ожидает принятия N сообщений (всего входного количества). Когда чётный процесс получает сообщение от нечётного (у сообщения будет тег 0), он отправляет всем остальным чётным процессам сообщение с тегом 1, для синхронизации получения. В случае же получения чётным процессом сообщения с тегом 1 (синхронизирующего сообщения), он просто уменьшает счётчик принятия сообщений. Таким образом чётный процесс может получить множество сообщений от нечётных или же не получить вовсе, но при этом приём сообщений корректно завершится в нужный момент.
5. Четный процесс после принятия очередного сообщения от нечётного (сообщение с тегом 0) создаёт ответ случайному процессу в нечётной группе, при этом отправляет сообщение тоже с тегом 0.
6. Нечётный процесс после отправки очередного сообщения ждёт принятие ответа от чётных процессов либо синхронизации от нечётных. И аналогично пунктам 4-5: если сообщение пришло от чётного процессов (с тегом 0), то процесс отправляет всем остальным нечётным процессам синхронизирующее сообщение (с тегом 1).
7. После отправки требуемого количества сообщений нечётный процесс продолжает принимать сообщения пока счётчик не достигнет N.

Сеть петри с 4 процессами для данного алгоритма представлена на рисунке 1.

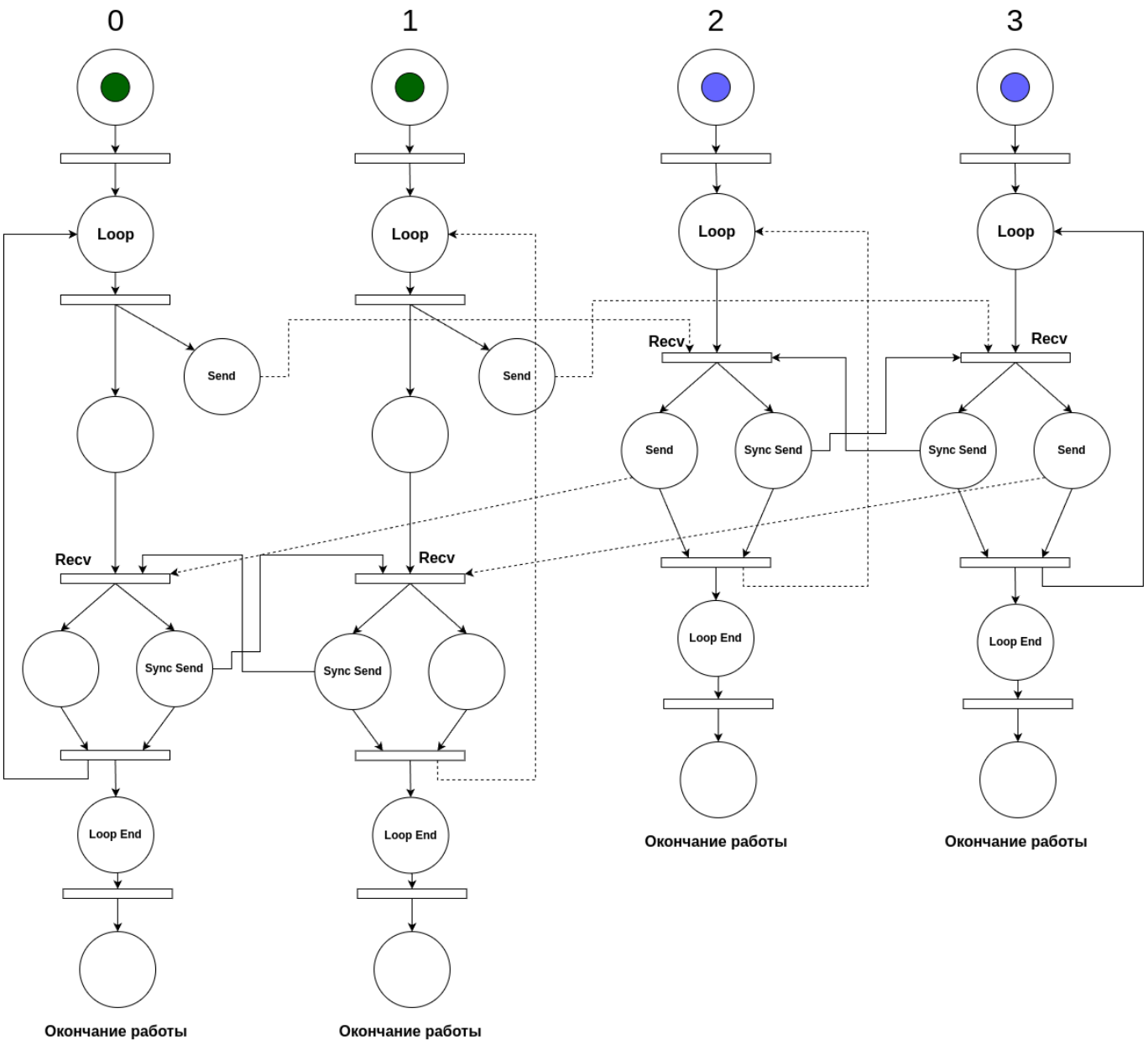

Рисунок 1. Сеть петри

Пример работы программы с 4 процессами и 2 сообщениями представлен на рисунке 2. Исходный код программы представлен в приложении А.

```
[2] time: 0.000151459s.
vlad@vlad-pc:~/projects/SPOD/lab3$ ./run.sh 4 2
[1] sent to 0
[3] sent to 0
[1] recieved from 0
[1] recieved from 0
[3] recieved sync from 1
[3] recieved sync from 1
[0] recieved from 3
[0] recieved from 1
[0] time: 1.8767e-05s.
[3] time: 0.000171066s.
[2] recieved sync from 0
[2] recieved sync from 0
[2] time: 0.000202336s.
[1] time: 0.000156579s.
vlad@vlad-pc:~/projects/SPOD/lab3$
```

Рисунок 2. Пример работы программы

На данном примере видно, что нечетные процессы (1, 3) отправили по одному сообщению процессу 0. Процесс 0 получил сообщения от них и на каждое отправил ответ процессу 1. Процессы 3 и 2 получили по два синхронизационных сообщения от процесса 1 и 0 соответственно.

Программа была запущена на разном количестве процессоров: от 2 до 16 с шагом 2 и на разном входном количестве требуемых сообщений: 10000, и 1000000. Время работы программы в данной задаче соответствует времени работы самого долгого процесса.

На рисунках 3-4 представлены графики зависимости времени работы программы от количества задействованных процессоров для 10000 и 1000000 входных сообщений соответственно. На рисунках 5-6 представлены графики ускорения в каждом из описанных случаев.

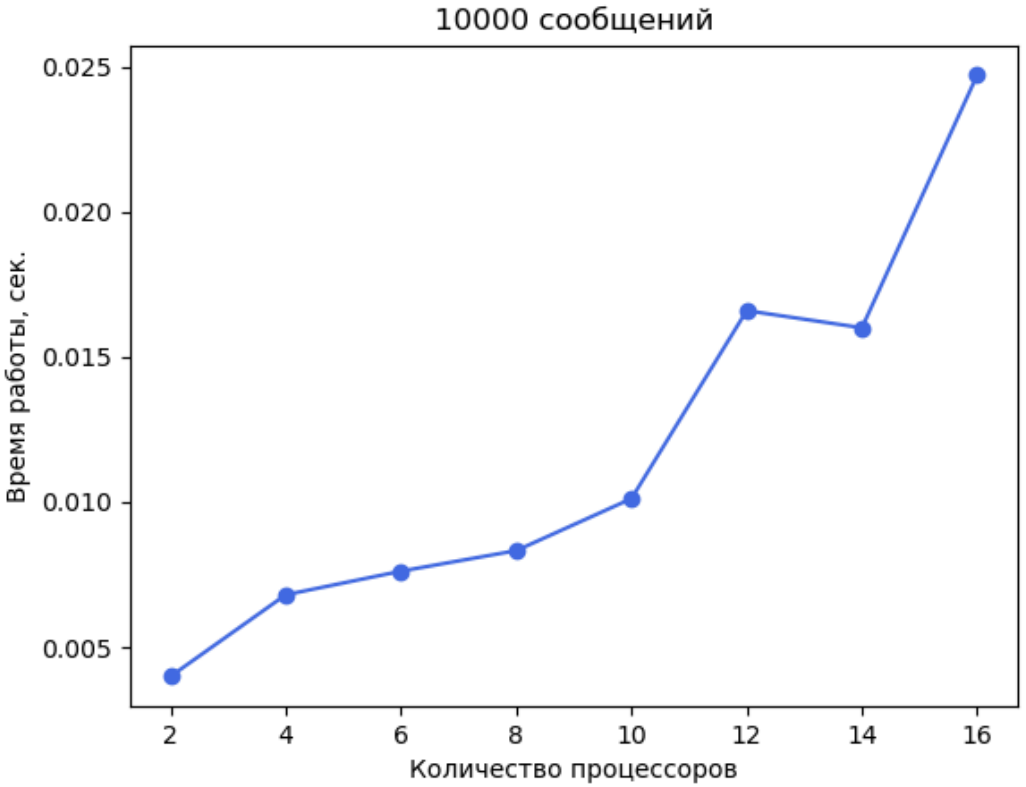

Рисунок 3. График времени работы программы для 10000 входных сообщений

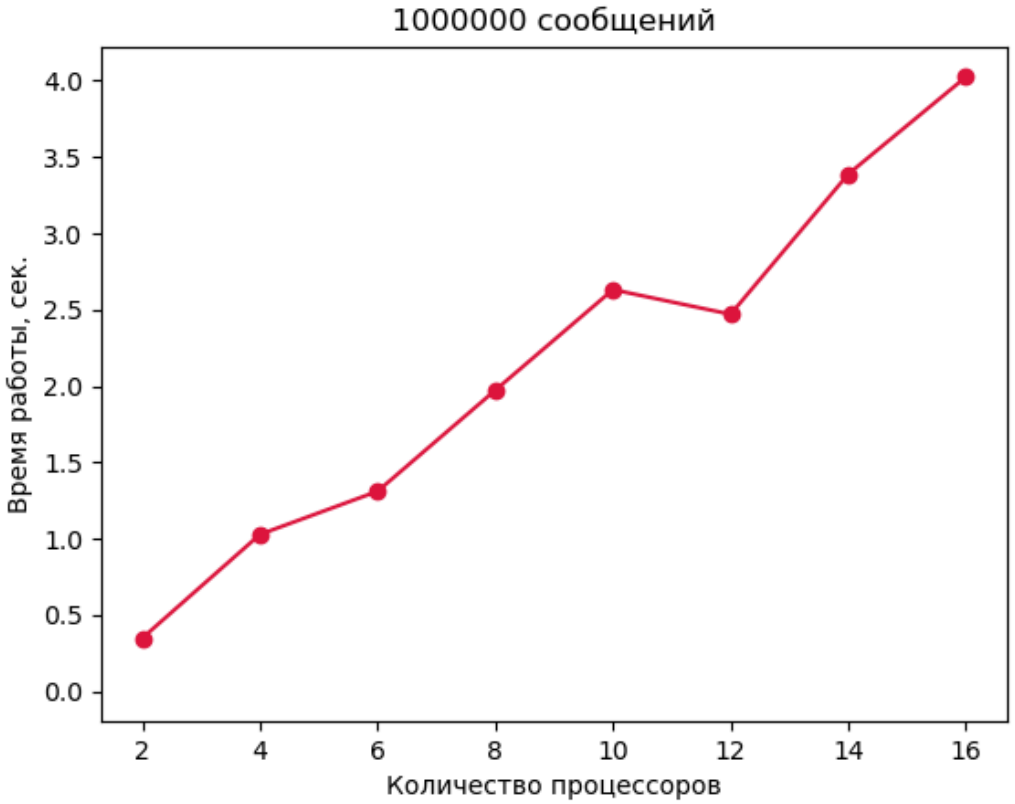

Рисунок 4. График времени работы программы для 1000000 входных сообщений

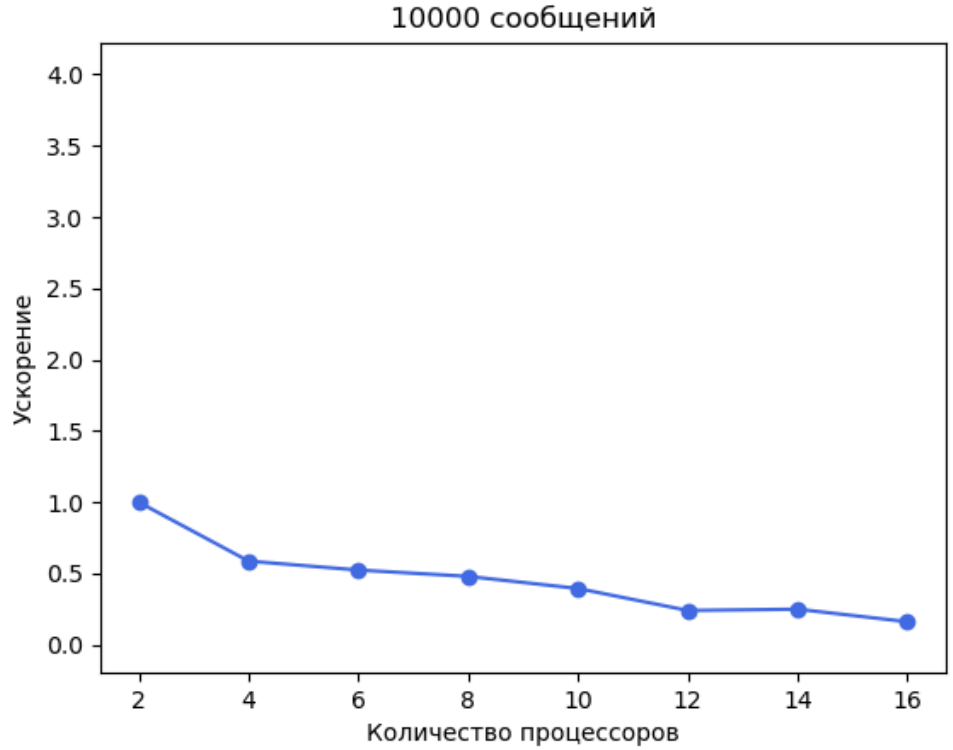

Рисунок 5. График ускорения работы программы для 10000 входных сообщений

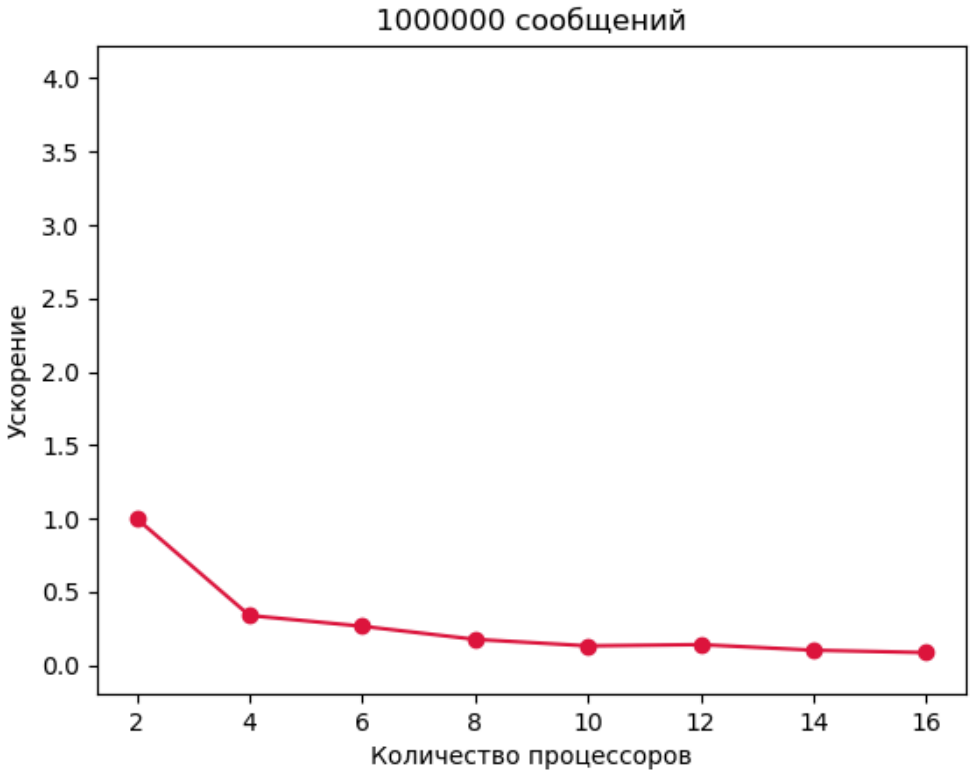

Рисунок 6. График ускорения работы программы для 1000000 входных сообщений

По данным графикам видно, что вне зависимости от объёма сообщений добавление дополнительных процессов только замедляет работу программы, так как при увеличении количества процессов, процессам требуется отправлять больше сообщений для синхронизации получения сообщений из противоположной группы, соответственно и возрастает время работы программы.

Выводы.

В результате выполнения работы была разработана параллельная программа, моделирующая обмен сообщениями между двумя группами процессов со случайным выбором адресата, для корректной работы циклов принятия сообщений в процессах, получение сообщений в группах было синхронизировано с помощью отправки дополнительных синхронизирующих сообщений. Разработанная программа запущена на разном количестве процессоров: от 2 до 16, и с разным объёмом входных данных. В результате тестирования программы построены графики времени работы программы и ускорения.

Увеличение количество процессов, вне зависимости от объёма данных, привело к замедлению работы программы, так как большему количеству процессов требуется отправка большего количества сообщений для синхронизации получения сообщений из другой группы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл main.cpp

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

// Тег сообщения - посылка или синхронизация
enum Tag { message, sync };

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

 // Общее количество посылок от первой группы
 const int msgCount = std::atoi(argv[1]);

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

 if (procCount < 2) {
   std::cerr << "Incorrect processes number" << std::endl;
   comm.Abort(0);
 }

 std::srand(std::time(0));

 // Данные о процессе и его группе
 const int group = procRank % 2;
 const int oddCount = procCount / 2;
 const int evenCount = procCount / 2 + procCount % 2;
 const int procIndexInGroup = procRank / 2;

 MPI::Status status;

 double time = 0, start, end;

 if (group == 1) {
   // Сколько сообщений должен отправить данный процесс
   int msgToSend = msgCount / oddCount +
                   (procIndexInGroup < (msgCount % oddCount) ? 1 : 0);

   for (size_t i = 0; i < msgCount; i++) {
     // Пока не отправлено нужное количество - отправляем
```

```
      if (i < msgToSend) {
        int dest = 2 * (std::rand() % evenCount);
        start = MPI::Wtime();
        comm.Send(nullptr, 0, MPI::INT, dest, Tag::message);
        end = MPI::Wtime();
        time += (end - start);
        std::cout << "[" << procRank << "] sent to " << dest << std::endl;
      }

      // Ждём ответ от чётного процесса или синхронизацию от нечётного

      start = MPI::Wtime();
      comm.Recv(nullptr, 0, MPI::INT, MPI::ANY_SOURCE, MPI::ANY_TAG, status);
      end = MPI::Wtime();
      time += (end - start);

      if (status.Get_tag() == Tag::sync) {
        std::cout << "[" << procRank << "] recieved sync from "
                  << status.Get_source() << std::endl;
      } else if (status.Get_tag() == Tag::message) {
        std::cout << "[" << procRank << "] recieved from "
                  << status.Get_source() << std::endl;

        // Если это ответ от чётного - отправляем посылку для синхронизации
        // другим нечётным процессам
        for (size_t i = 0; i < oddCount; i++) {
          int dest = (2 * i) + 1;
          if (dest == procRank) continue;
          start = MPI::Wtime();
          comm.Send(nullptr, 0, MPI::INT, dest, Tag::sync);
          end = MPI::Wtime();
          time += (end - start);
        }
      }
    }
  } else {
    for (size_t i = 0; i < msgCount; i++) {
      start = MPI::Wtime();
      comm.Recv(nullptr, 0, MPI::INT, MPI::ANY_SOURCE, MPI::ANY_TAG, status);
      end = MPI::Wtime();
      time += (end - start);

      if (status.Get_tag() == Tag::sync) {
        std::cout << "[" << procRank << "] recieved sync from "
                  << status.Get_source() << std::endl;
      } else if (status.Get_tag() == Tag::message) {
        std::cout << "[" << procRank << "] recieved from "
```

```
                << status.Get_source() << std::endl;
        // Если пришло сообщение от нечётного - отправляем посылку для
        // синхронизации другим чётным процессам
        for (size_t i = 0; i < evenCount; i++) {
          int dest = 2 * i;
          if (dest == procRank) continue;
          start = MPI::Wtime();
          comm.Send(nullptr, 0, MPI::INT, dest, Tag::sync);
          end = MPI::Wtime();
          time += (end - start);
        }

        // И ответ нечётному процессу
        int dest = 2 * (std::rand() % oddCount) + 1;
        start = MPI::Wtime();
        comm.Send(nullptr, 0, MPI::INT, dest, Tag::message);
        end = MPI::Wtime();
        time += (end - start);
      }
    }
  }

  std::cout << "[" << procRank << "]" << " time: " << time << "s." << std::endl;

  MPI::Finalize();
}
```

Файл build.sh

```
mpicxx.openmpi main.cpp -o lab3
```

Файл run.sh

```
mpiexec.openmpi -n $1 --oversubscribe ./lab3 $2
```