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: Введение в OpenMPI

Студент гр. 3381 _____ Иванов А.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2025

Цель работы.

Изучить основы написания программ с помощью OpenMPI на языке программирования C и написать простейший пример коммуникации между процессами.

Задание.

Задание 1

1. Все процессы, кроме процесса с рангом 0, передают значение своего ранга нулевому процессу. Процесс с рангом 0 сначала печатает значение своего ранга, а далее 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. Проанализировать порядок вывода сообщений на экран. Для этого запустить программу несколько раз при фиксированном числе процессов, например при восьми. Вывести правило, определяющее порядок вывода сообщений.

Задание 2

1. Модифицировать программу таким образом, чтобы порядок вывода сообщений на экран соответствовал номеру соответствующего процесса.
2. Сравнить результаты работы двух программ.

Выполнение работы.

Задание 1

1. Написать основную программу

   Для использования OpenMPI был подключён заголовок mpi.h (OpenMPI должен быть заранее установлен пользователем). Сначала OpenMPI инициализируется с помощью функции MPI_Init. Для получения ранга процесса используется функция MPI_Comm_rank. Далее представлена логика, различная для процессов с определёнными рангами.

   1. Для процесса с нулевым рангом запускается цикл for, который в своём теле ждёт получения сообщения от любого из остальных процессов и выводит строку о получении сообщения. Цикл выполняется n-1 раз, где n — количество процессов;
   2. Процессы с рангом, отличным от нулевого просто отправляют нулевому процессу свой ранг с помощью функции MPI_Send.

   После вызывается функция MPI_Finalize для завершения работы с MPI.

2. Проанализировать порядок вывода сообщений на экран.

   Сообщения выводятся только в процессе с рангом 0, так что что первой строкой обязательно будет «Hello from process 0», а далее в цикле for этот порядок будет случайным. В зависимости от того как быстро будет запущен каждый из процессов и как планировщик распределит время их выполнения.

Задание 2

1. Модифицировать программу таким образом, чтобы порядок вывода сообщений на экран соответствовал номеру соответствующего процесса.

   Для обеспечения вывода сообщений в порядке возрастания в упомянутом ранее цикле for необходимо значение ожидаемого источника заменить с MPI_ANY_SOURCE на индекс i, по которому

производятся итерации (начальное значение для i должно быть равно 1).

2. Сравнить результаты работы двух программ.

После указания определённого источника сообщения выводятся в порядке возрастания. Ранее порядок был случаен. Сравнение графиков времени показывает (см. рисунок 2), что при упорядочивании рангов время немного возрастает. Это кажется очевидным, так как нужное сообщение во втором случае приходит не сразу, следовательно есть какой-то промежуток времени ожидания каждого сообщения. Время по каждому значению количества процессов (2, 4, 8, 16, 32) усреднено на основе 50 измерений.

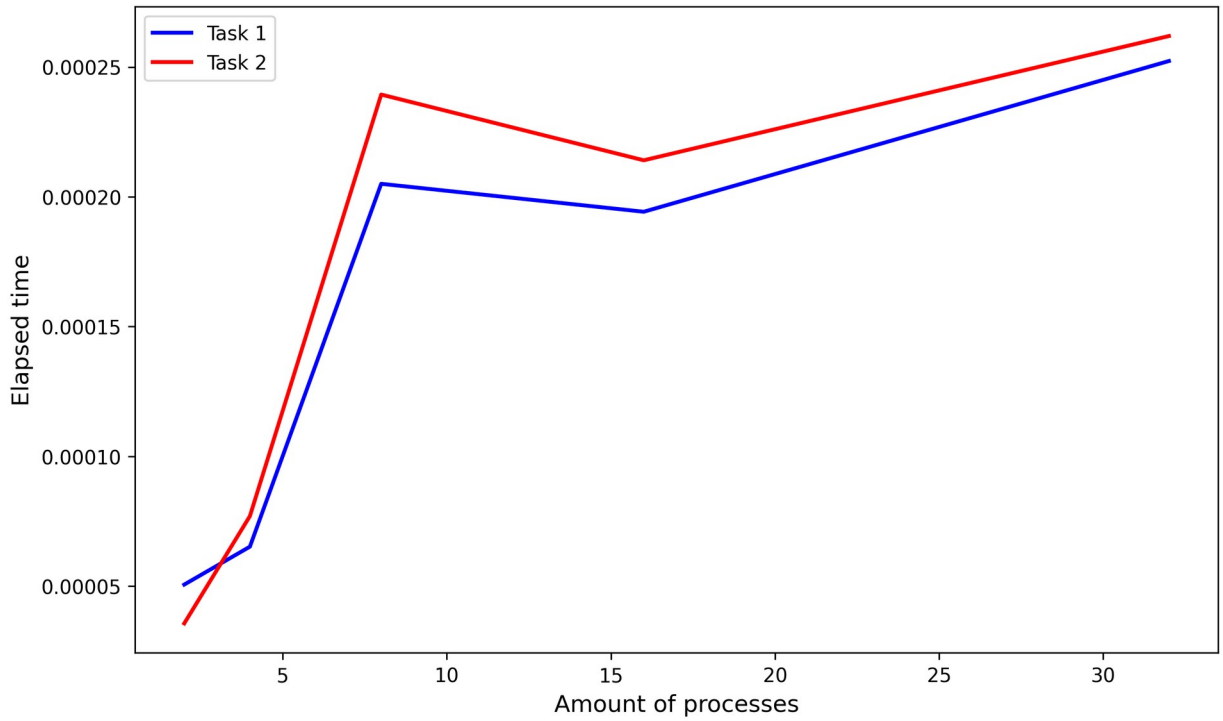

Рисунок 2.1 — Сравнение графиков времени

Также представлены графики времени и ускорения для каждой программы по отдельности: все замеры усреднены на основе 500 значений, отсеяны значения меньше 95-го квантиля.

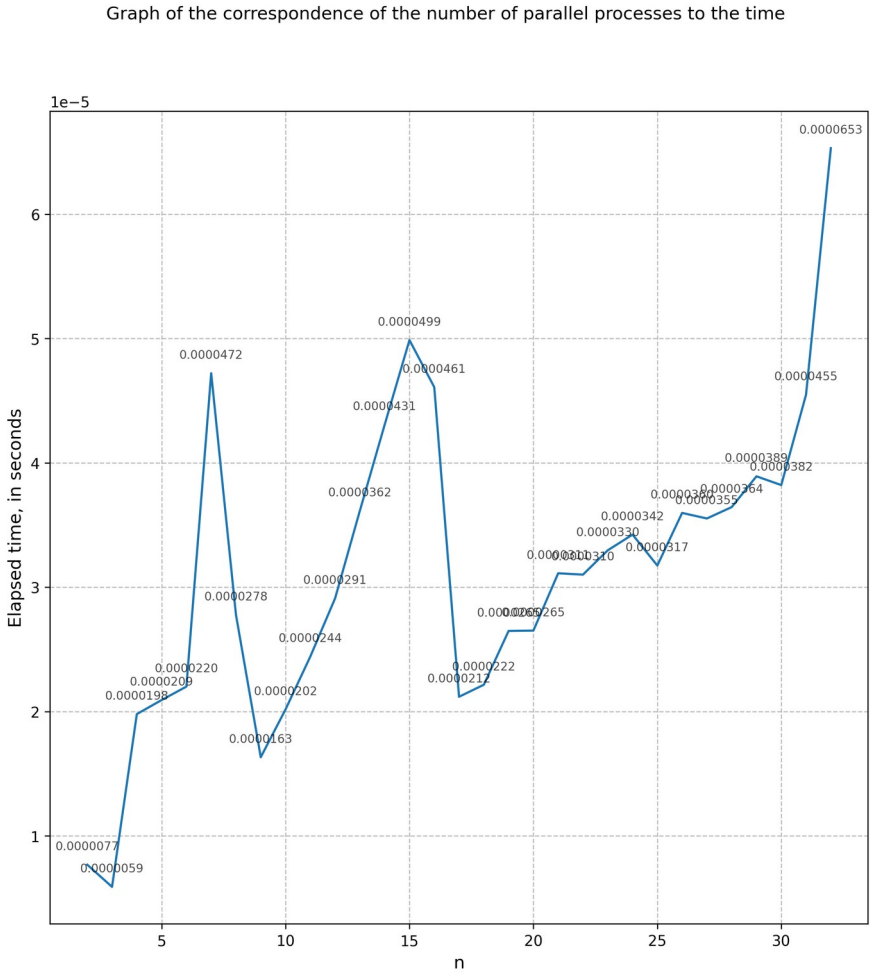

Рисунок 2.2 — График зависимости времени от количества процессов для первой программы

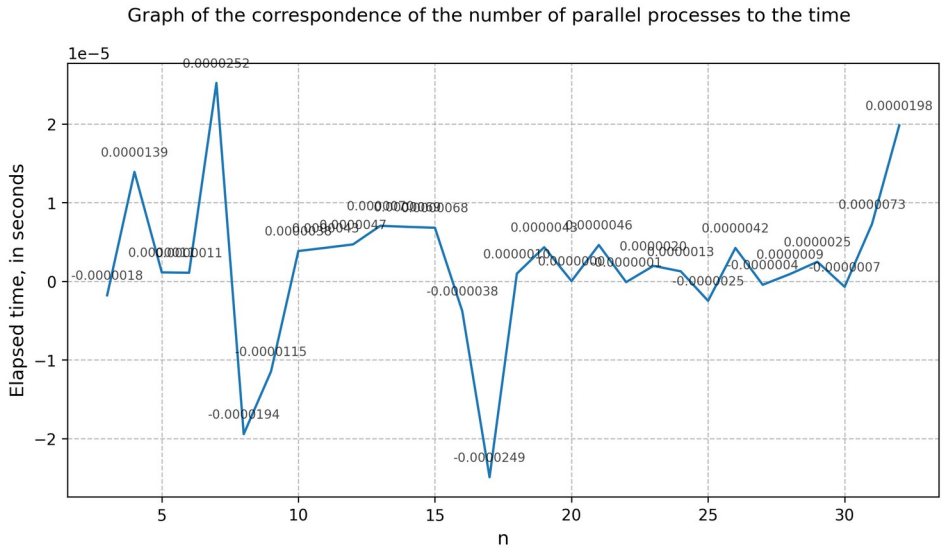

Рисунок 2.3 — График ускорения первой программы

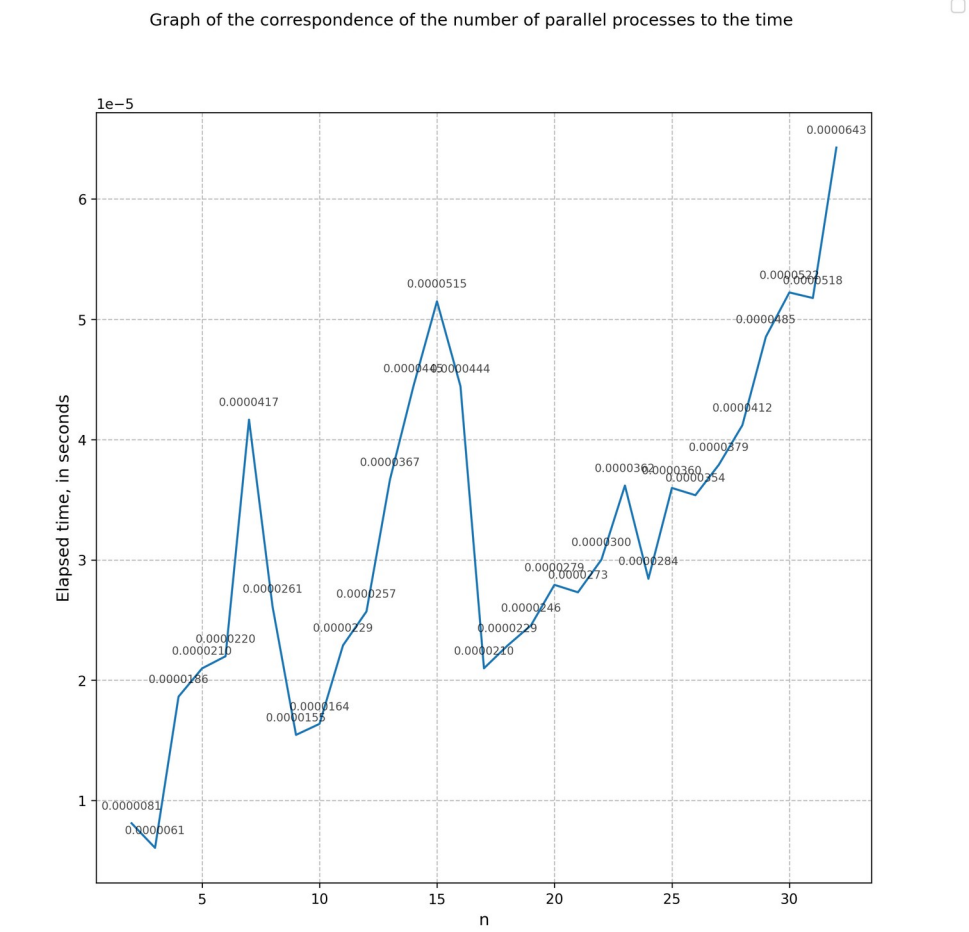

Рисунок 2.4 — График зависимости времени от количества процессов для второй программы

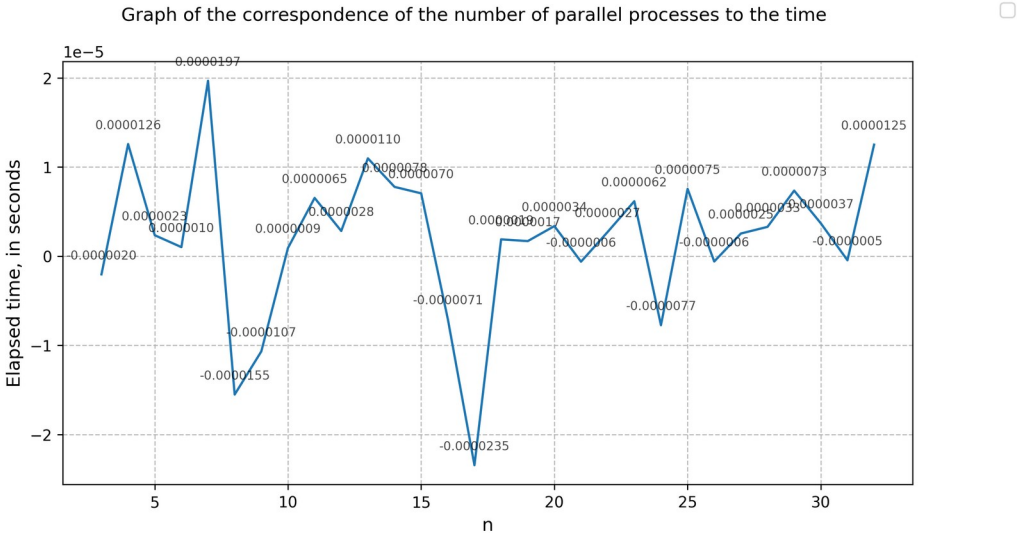

Рисунок 2.5 — График ускорения второй программы

Графики должны быть линейными или почти сверхлинейными, так как в задачи ничего не вычисляется параллельно, количество операций растёт линейно. Графики соответствуют этой гипотезе с небольшими выбросами.

Сети Петри для первого и второго задания приведены в приложении А (ограничился главным процессом и двумя вспомогательными, иначе кофмортный просмотр сети станет затруднителен).

Выводы.

В ходе лабораторной работы были изучены основы написания программ с помощью OpenMPI на языке программирования C и написаны простейшие примеры коммуникации между процессами.

Написана программа, в которой главный процесс печатает на экран свой ранг, далее получает от других процессов их ранги и также выводит их на экран. Программа представлена в двух вариациях: получение рангов от процессов в произвольном порядки и по увеличению значения ранга отправившего процесса.

ПРИЛОЖЕНИЕ А
СЕТИ ПЕТРИ

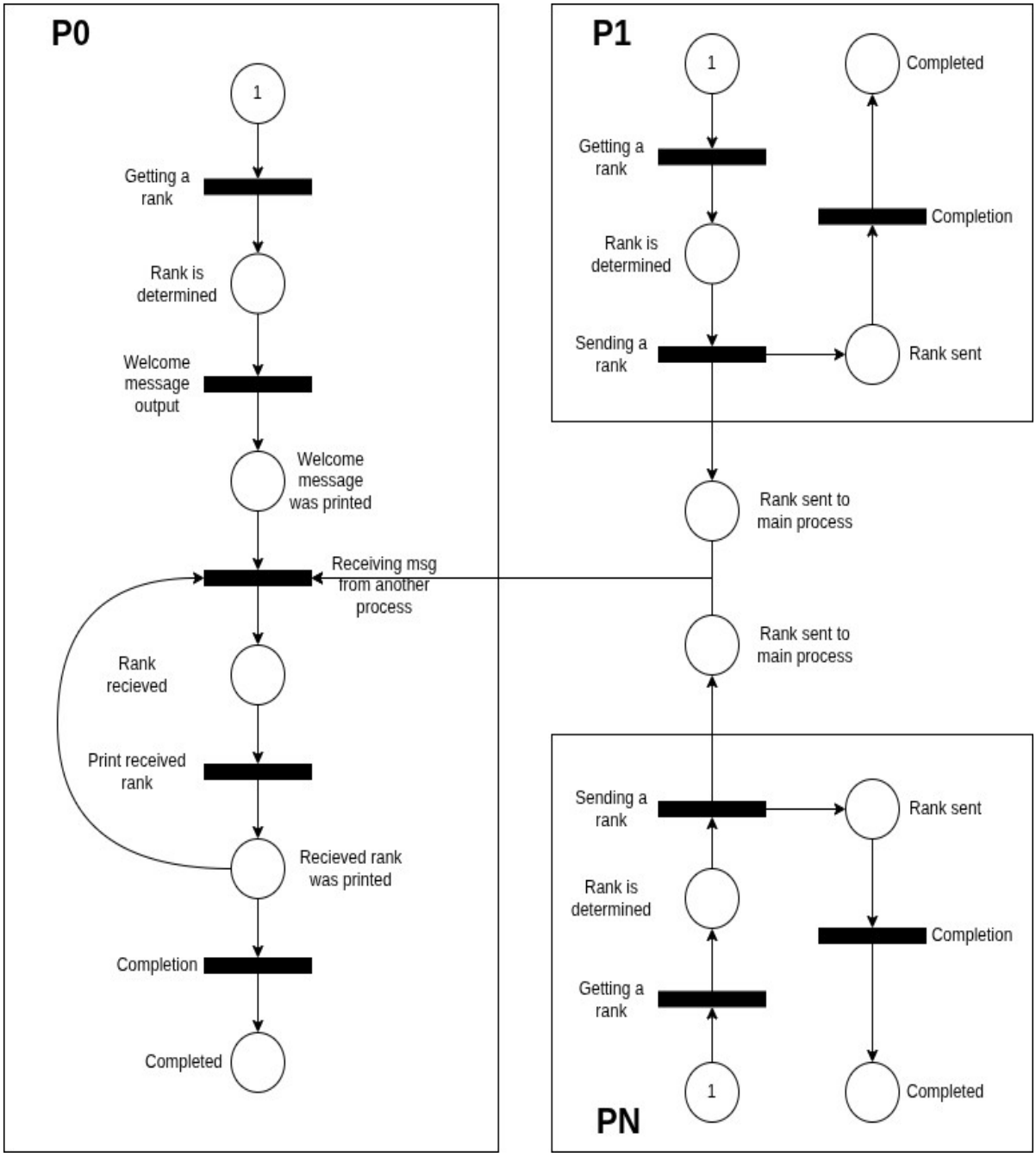

Рисунок А.1 — Сеть Петри для первого задания

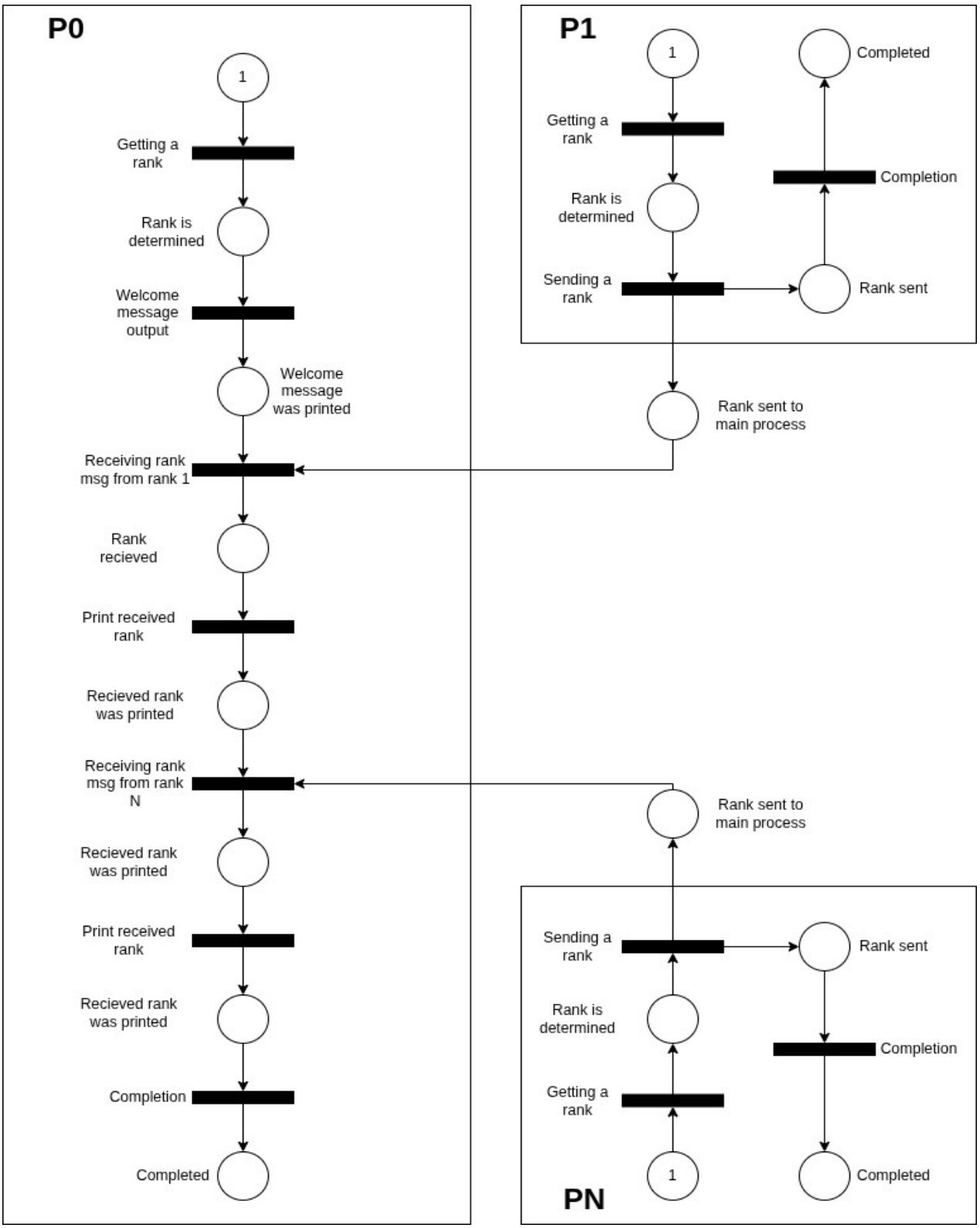

Рисунок А.2 — Сеть Петри для второго задания (отличие только в условии приёма по рангу)

ПРИЛОЖЕНИЕ Б
ЛИСТИНГ

Реализация задачи.

Название файла: lab1/src/task1.c

```
#include "../../task.h"

#include <mpi.h>
#include <stdio.h>

static inline void hello(int rank)
{
    printf("Hello from process %d\n", rank);
}

int task(int world_size, int world_rank, int n_args, char* args[])
{
    // Rank specific logic
    if (world_rank == 0) {
        #ifndef MEASURE_TIME
            hello(world_rank);
        #endif

        // Recieve rank of other processes
        int rank;
        // Loop is started from 1 just to abstractly say that we wait for
        // messages from processes with ranks from 1 to 7
        for (int i = 1; i < world_size; ++i) {
            MPI_Recv(&rank, 1, MPI_INT, MPI_ANY_SOURCE,
                     0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            #ifndef MEASURE_TIME
                hello(rank);
            #endif
        }
    } else {
        MPI_Send(&world_rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }

    return 0;
}
```

Название файла: lab1/src/task2.c

```
#include "../../task.h"

#include <mpi.h>
#include <stdio.h>

static inline void hello(int rank)
{
    printf("Hello from process %d\n", rank);
}

int task(int world_size, int world_rank, int n_args, char* args[])
{
    // Rank specific logic
    if (world_rank == 0) {
```

```
        #ifndef MEASURE_TIME
            hello(world_rank);
        #endif

        // Recieve rank of other processes
        int rank;
        // Loop is started from 1 just to abstractly say that we wait for
        // messages from processes with ranks from 1 to 7
        for (int i = 1; i < world_size; ++i) {
            MPI_Recv(&rank, 1, MPI_INT, i,
                     0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            #ifndef MEASURE_TIME
                hello(rank);
            #endif
        }
    } else {
        MPI_Send(&world_rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }

    return 0;
}
```

Построение графиков.

Название файла: chart_tools.py

```
import argparse
import subprocess
import statistics

import matplotlib.pyplot as plt

from functools import partial


def collect_time_stats(
    argv: str,
    avg_param: int,
    n_values: list = [1, 2, 4, 8, 16, 32, 64]) -> dict:

    time_stats = dict()

    for n in n_values:
        time_values = []
        for i in range(avg_param):
            print(f"{i+1:3}) n={n}")

            result = subprocess.run(
                f"mpirun --oversubscribe -n {n} {argv}".split(),
                encoding="utf-8",
                capture_output=True
            )
            if result.returncode != 0:
                    print(f"Error in subprocess run:\n\tstdout: {result.stdout}
\n\tstderr: {result.stderr}")
                exit(1)

            time_values.append(float(result.stdout))

        if len(time_values) == 0:
            print("Empty time values list, exiting...")
            exit(1)
```

```
        if len(time_values) > 1:
            # Calculate 95th quantile of measured time values
            time_values_95th_quantile = statistics.quantiles(
                time_values, n=100
            )[94]
                        time_values = [t for t in time_values if t <
time_values_95th_quantile]

        time_stats[n] = sum(time_values) / len(time_values)

    return (
        list(time_stats.keys()),
        list(time_stats.values())
    )

def setup_axes(
    fig,
    ax,
    suptitle="Graph of the correspondence of the number of parallel processes to
the time"
) -> None:
    fig.suptitle(suptitle)

    ax.grid(True, linestyle='--', alpha=0.8)

    # Setting up x axis
    ax.set_xlabel('n', fontsize=12)
    # Setting up y axis
    ax.set_ylabel('Elapsed time, in seconds', fontsize=12)

def calculate_acceleration(X: int, Y: float):
    # Use last coordinates to plot acceleration chart
    X_ac, Y_ac = X[1:], []
    for i in range(1, len(X)):
        Y_ac.append(Y[i] - Y[i-1])
    return (X_ac, Y_ac)

def plot_chart(ax, X: list, Y: list, label: str = None) -> None:
    ax.plot(X, Y, label=label)
    # Add this after the ax.plot(X, Y) line
    for i, (x, y) in enumerate(zip(X, Y)):
        ax.annotate(f'{y:.7f}',
                    (x, y),
                    textcoords="offset points",
                    xytext=(0,10),
                    ha='center',
                    fontsize=8,
                    alpha=0.7)

def int_limited(arg: str, lower: int = None, upper: int = None):
    try:
        limited_int_arg = int(arg)
        if lower is not None and limited_int_arg < lower:
            raise argparse.ArgumentTypeError(f"Int value must be greater than
{lower}")
        if upper is not None and limited_int_arg > upper:
            raise argparse.ArgumentTypeError(f"Int value must be lower than
{upper}")

        return limited_int_arg
    except ValueError:
        raise argparse.ArgumentTypeError(f"Invalid int value: {arg}")
```

```
int_is_positive = partial(int_limited, lower=1)

def parse_cli():
    parser = argparse.ArgumentParser(description="Plot chart for MPI program")
    parser.add_argument(
        "-p",
        "--averaging-parameter",
        type=int_is_positive,
        required=True,
        help="The number of values for which the averaging is performed"
    )
    parser.add_argument(
        "-a",
        "--argv",
        type=str,
        required=True,
        help="argv of program to run with mpirun"
    )

    return parser.parse_args()


if __name__ == "__main__":
    args = parse_cli()

    X, Y = collect_time_stats(args.executable, args.averaging_parameter)

    fig, ax = plt.subplots(1, figsize=(15, 15))

    setup_axes(fig, ax)
    plot_chart(ax, X, Y)

    fig.savefig("chart.png", dpi=300, bbox_inches="tight", facecolor="white")
```

Название файла: lab2/src/make_chart.py

```
import os
import sys
sys.path.append(os.path.join(os.path.dirname(__file__), '../..'))

import chart_tools
import matplotlib.pyplot as plt

if __name__ == "__main__":
    args = chart_tools.parse_cli()

    fig, ax = plt.subplots(1, figsize=(15, 15))
    chart_tools.setup_axes(fig, ax)
    X, Y = chart_tools.collect_time_stats(
        f"{args.argv}",
        args.averaging_parameter,
        n_values=(list(range(2, 32 + 1)))
    )
    chart_tools.plot_chart(ax, X, Y)
    fig.legend(fontsize=12)
    fig.savefig(args.output, dpi=300, bbox_inches="tight", facecolor="white")

    fig_ac, ax_ac = plt.subplots(1, figsize=(10, 5))
    chart_tools.setup_axes(fig_ac, ax_ac)
    X_ac, Y_ac = chart_tools.calculate_acceleration(X, Y)
    chart_tools.plot_chart(ax_ac, X_ac, Y_ac)
```

```
    fig_ac.legend(fontsize=12)
            fig_ac.savefig("acceleration.png",   dpi=300,   bbox_inches="tight",
facecolor="white")
```

Общая часть.

Название файла: common.h

```
#ifndef COMMON_H
#define COMMON_H

#include <stdio.h>
#include <stdlib.h>

void generate_array(int* buf, size_t bufsize, unsigned int seed);
void print_array(int* buf, size_t bufsize);

#endif
```

Название файла: common.c

```
#include "common.h"

void generate_array(int* buf, size_t bufsize, unsigned int seed)
{
    if (buf == NULL) return;

    srand(seed);

    const int upper_bound = 1000;
    for (size_t i = 0; i < bufsize; ++i) {
        buf[i] = rand() % upper_bound;
    }
}

void print_array(int* buf, size_t bufsize)
{
    if (buf == NULL || bufsize == 0) return;

    int *tmp = buf;
    for (size_t i = 0; i < bufsize; ++i) {
        printf("%d%c", tmp[i], ((i != bufsize - 1) ? (' ') : ('\n')));
        ++tmp;
    }
}
```

Название файла: task.h

```
#ifndef TASK_H
#define TASK_H

// Variadic arguments behavior defined by task itself
int task(int world_size, int world_rank, int n_args, char* args[]);

#endif
```

Название файла: common.c

```
#include <mpi.h>
#include <stdio.h>
#include <time.h>

#include "task.h"

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    // Get rank
    int world_rank = 0;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get amount of processes
    int world_size = 0;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int root_rank = 0;

    #ifdef MEASURE_TIME
    MPI_Barrier(MPI_COMM_WORLD);
    clock_t start_clock, end_clock;
    if (world_rank == root_rank) {
        start_clock = clock();
    }
    #endif

    if (task(world_size, world_rank, argc, argv)) {
        if (world_rank == 0) {
            fprintf(stderr, "Unable to recover from error in task\n");
        }
        goto finalize;
    }

    #ifdef MEASURE_TIME
    if (world_rank == root_rank) {
        end_clock = clock();
        printf("%lf\n", ((double)end_clock - start_clock) / CLOCKS_PER_SEC);
    }
    #endif

finalize:
    return MPI_Finalize();
}
```