

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 С 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.

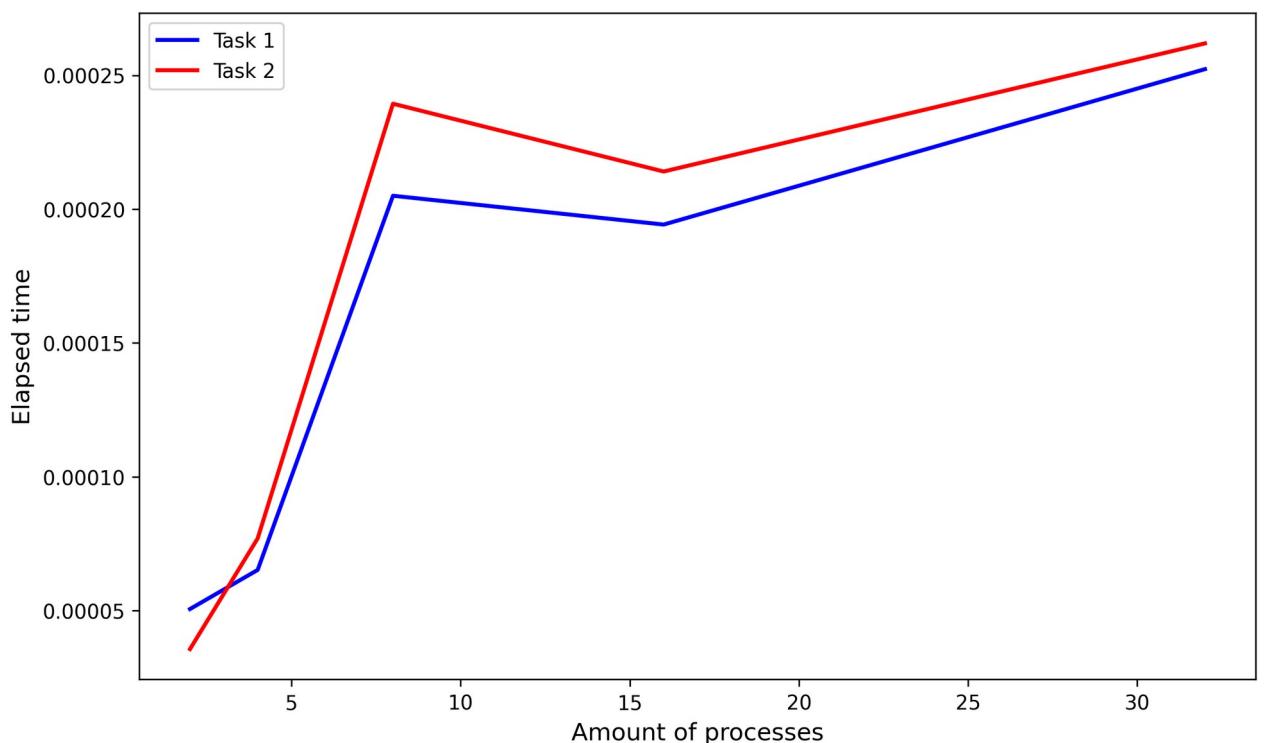


Рисунок 2 — Сравнение графиков времени

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

Выходы.

В ходе лабораторной работы были изучены основы написания программ с помощью OpenMPI на языке программирования С и написан простейший пример коммуникации между процессами. Произведен замер времени для разных вариаций программы.

ПРИЛОЖЕНИЕ А

СЕТИ ПЕТРИ

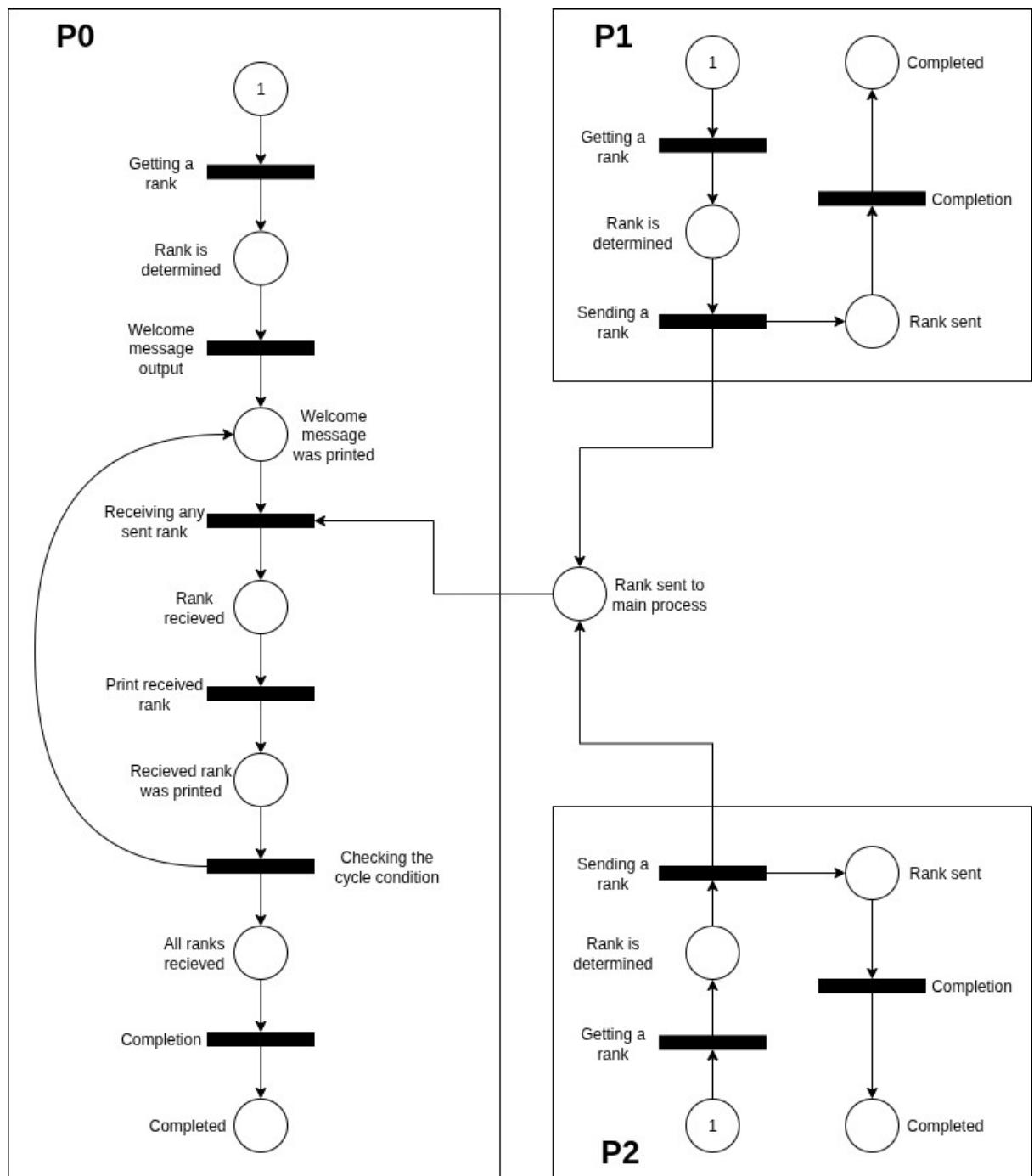


Рисунок А.1 — Сеть Петри для первого задания

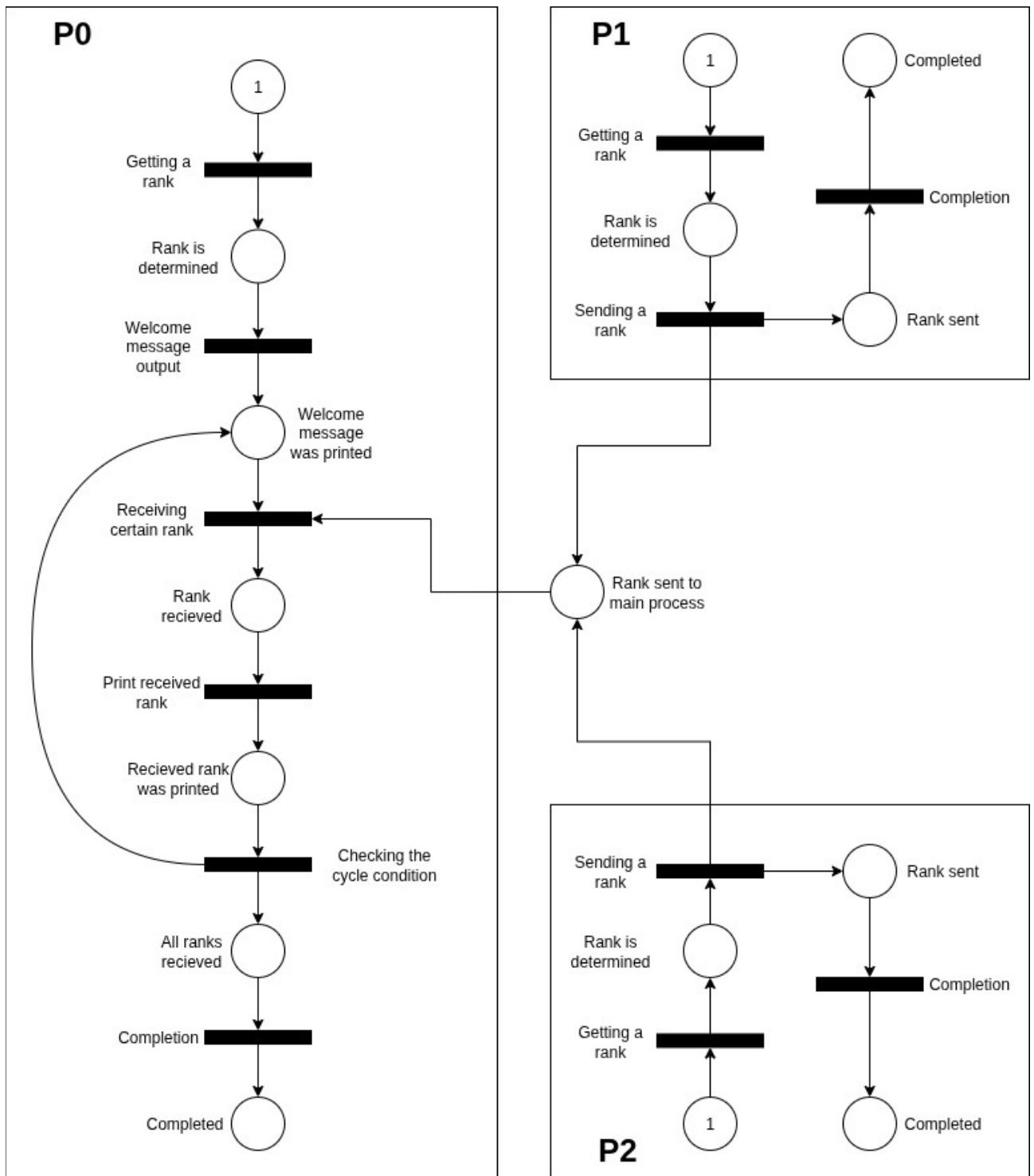


Рисунок А.2 — Сеть Петри для второго задания (отличие только в условиях приёма по рангу)

ПРИЛОЖЕНИЕ Б

ЛИСТИНГ

Название файла: src/task1.c

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

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

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

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Rank specific logic
    if (world_rank == 0) {
        #ifdef MEASURE_TIME
            double start = MPI_Wtime();
        #endif
        // Get the number of processes
        int world_size;
        MPI_Comm_size(MPI_COMM_WORLD, &world_size);
        #ifdef VERBOSE
            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);
            #ifdef VERBOSE
                hello(rank);
            #endif
        }
        #ifdef MEASURE_TIME
            printf("%lf\n", MPI_Wtime() - start);
        #endif
    } else {
        MPI_Send(&world_rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
}
```

```

    // Finalize the MPI environment.
    MPI_Finalize();

    return 0;
}

```

Название файла: src/task2.c

```

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

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

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

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Rank specific logic
    if (world_rank == 0) {
        #ifdef MEASURE_TIME
            double start = MPI_Wtime();
        #endif
        // Get the number of processes
        int world_size;
        MPI_Comm_size(MPI_COMM_WORLD, &world_size);
        #ifdef VERBOSE
            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);
            #ifdef VERBOSE
                hello(rank);
            #endif
        }
        #ifdef MEASURE_TIME
            printf("%lf\n", MPI_Wtime() - start);
        #endif
    }
}

```

```

    } else {
        MPI_Send(&world_rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }

    // Finalize the MPI environment.
    MPI_Finalize();

    return 0;
}

```

Название файла: Makefile

```

CC = mpicc

TASK1_SRC = src/task1.c
TASK2_SRC = src/task2.c

TASK1_TARGET=task1
TASK2_TARGET=task2

BUILD_DIR = build

.PHONY: all graph clean

all: ${TASK1_TARGET} ${TASK2_TARGET}

${TASK1_TARGET}: ${TASK1_SRC}
    @${CC} -DVERBOSE ${TASK1_SRC} -o ${TASK1_TARGET}

${TASK2_TARGET}: ${TASK2_SRC}
    @${CC} -DVERBOSE ${TASK2_SRC} -o ${TASK2_TARGET}

graph:
    if [ ! -d "venv" ]; then \
        python -m venv venv; \
        source venv/bin/activate; \
        pip install -r requirements.txt; \
    else \
        source venv/bin/activate; \
    fi; \
    TMP_DIR=$$(mktemp -d); \
    ${CC} -DMEASURE_TIME ${TASK1_SRC} \
        -o $$TMP_DIR/${TASK1_TARGET}; \
    ${CC} -DMEASURE_TIME ${TASK2_SRC} \
        -o $$TMP_DIR/${TASK2_TARGET}; \
    python3 src/make_graph.py graph.png \
        $$TMP_DIR/${TASK1_TARGET} $$TMP_DIR/${TASK2_TARGET}

clean:
    rm ${TASK1_TARGET} ${TASK2_TARGET}

```

Название файла: src/make_graph.py.

```
import sys
import subprocess

import matplotlib.pyplot as plt

def measure_time(program_name: str, iter: int = 100) -> dict:
    """Measure time for different amount of processes for given program

    :param program_name: name of program to execute
    :param iter: amount of iterations to get average time value
    :return: dictionary with schema process_count -> avg_time
    """

    result = dict()
    for n_proc in [2, 4, 8, 16, 32]:
        time_values = []
        for i in range(iter):
            output = subprocess.run(
                ["mpirun", "-n", str(n_proc),
                 "--oversubscribe", program_name],
                capture_output=True,
                encoding="utf-8"
            ).stdout
            print(f"n_proc={n_proc}: {i}/{iter}")
            time_values.append(float(output))

        result[n_proc] = sum(time_values) / len(time_values)

    return result

def plot_graph(output_path: str, X1: tuple, Y1: tuple, X2: tuple, Y2: tuple) -> None:
    """Plot graph to compare execution time of two programs.

    :param output_path: name for graph image file
    :param X1: first program processes counts
    :param Y1: elapsed times for first program
    :param X2: second program processes counts
    :param Y2: elapsed times for second program
    """

    plt.figure(figsize=(10, 6))
    plt.plot(X1, Y1, "b-", linewidth=2, label="Task 1")
    plt.plot(X2, Y2, "r-", linewidth=2, label="Task 2")
    plt.xlabel('Amount of processes', fontsize=12)
    plt.ylabel('Elapsed time', fontsize=12)
    plt.legend(fontsize=10)

    plt.savefig(
        output_path,
```

```
dpi=300,  
bbox_inches="tight", facecolor="white"  
)  
plt.close()  
  
if __name__ == "__main__":  
    if len(sys.argv) != 4:  
        print(f"Usage: {sys.argv[0]} <image> <prog1> <prog2>")  
        exit(0)  
  
X1, Y1 = zip(*measure_time(sys.argv[2]).items())  
X2, Y2 = zip(*measure_time(sys.argv[3]).items())  
plot_graph(sys.argv[1], X1, Y1, X2, Y2)
```