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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Параллельные алгоритмы»

Тема: Виртуальные топологии

Студент гр. 3381 _____ Иванов А.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2025

Цель работы.

Изучить принципы создания виртуальной топологии параллельной программы на примере декартовой решётки. Реализовать программу на языке программирования C в соответствии с заданием.

Задание.

Вариант 3. Число процессов K является четным: $K = 2N,\ N > 1$. В процессах 0 и N дано по одному вещественному числу A. Определить для всех процессов декартову топологию в виде матрицы размера $2 \times N$, после чего, используя функцию MPI_Cart_sub, расщепить матрицу процессов на две одномерные строки (при этом процессы 0 и N будут главными процессами в полученных строках). Используя одну коллективную операцию пересылки данных, переслать число A из главного процесса каждой строки во все процессы этой же строки и вывести полученное число в каждом процессе (включая процессы 0 и N).

Выполнение работы.

1. Проверка количества процессов в глобальном коммуникаторе: если это количество нечётно, то выводится сообщение об ошибке и программа завершает свою работу;
2. Создание глобальной декартовой решётки: создаётся декартова решётся из процессов в глобальном коммуникаторе размером $(2, N)$. По столбцам и по строкам непериодическая, ранги процессов в решётке совпадают с рангами, образуемыми при построчной итерации по решётке;
3. Создание первой подрешётки: для первой строки полученной решётки создаётся отдельный коммуникатор;
4. Создание второй подрешётки: для второй строки полученной решётки создаётся отдельный коммуникатор;
5. Генерация A в процессах с рангами 0 и N;

6. Рассылка значения A в рамках соответствующей подрешётки: если процесс принадлежит к первой подрешётке, то выполняется MPI_Bcast по первой подрешётке, иначе по второй;
7. Вывод ранга и полученного вещественного числа A: если не определена директива MEASURE_TIME, то на экран выводится соответствующее сообщение.

По результатам многократного запуска полученной программы получены следующие графики:

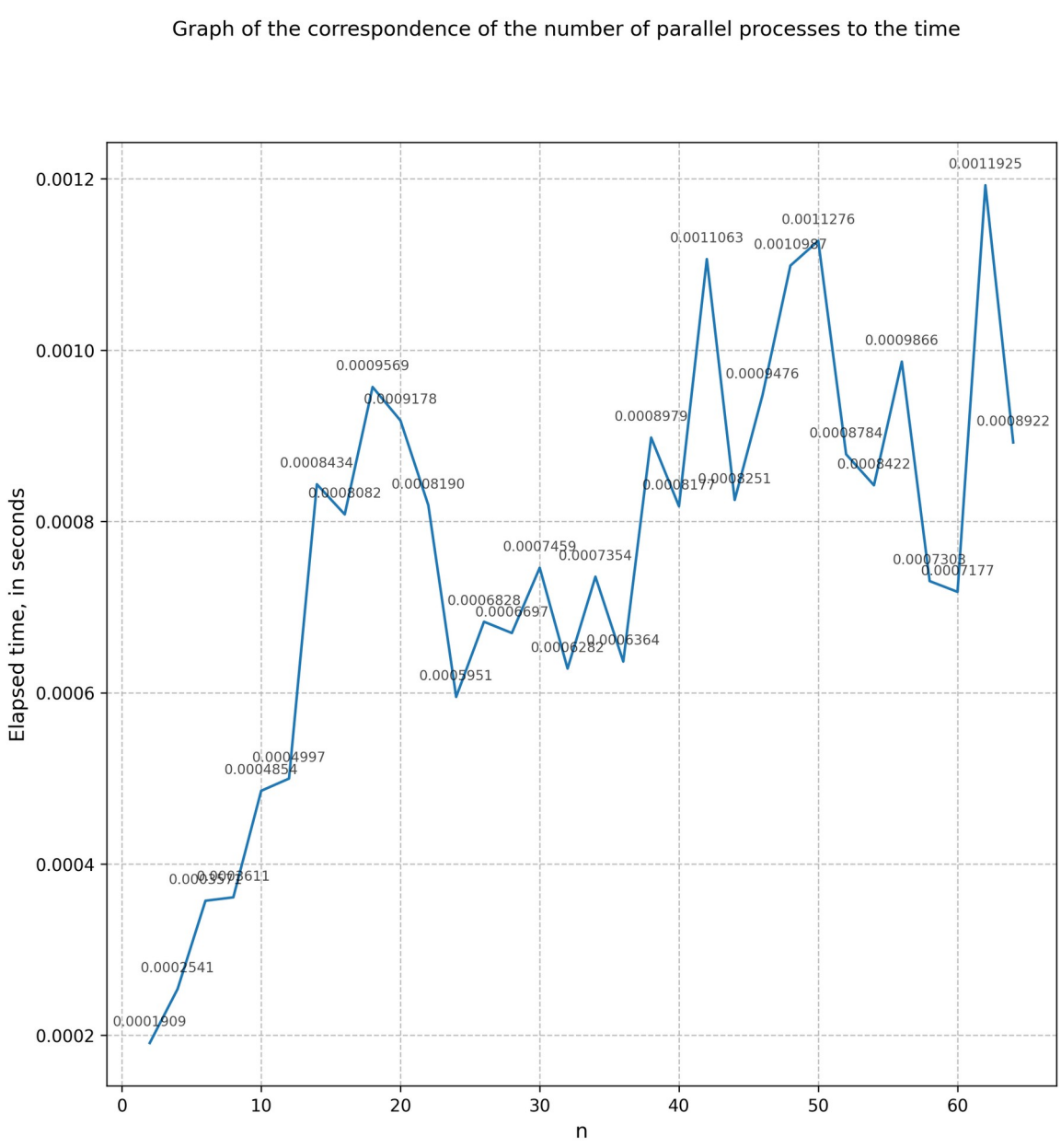

Рисунок 1 — Зависимость времени от количества процессов

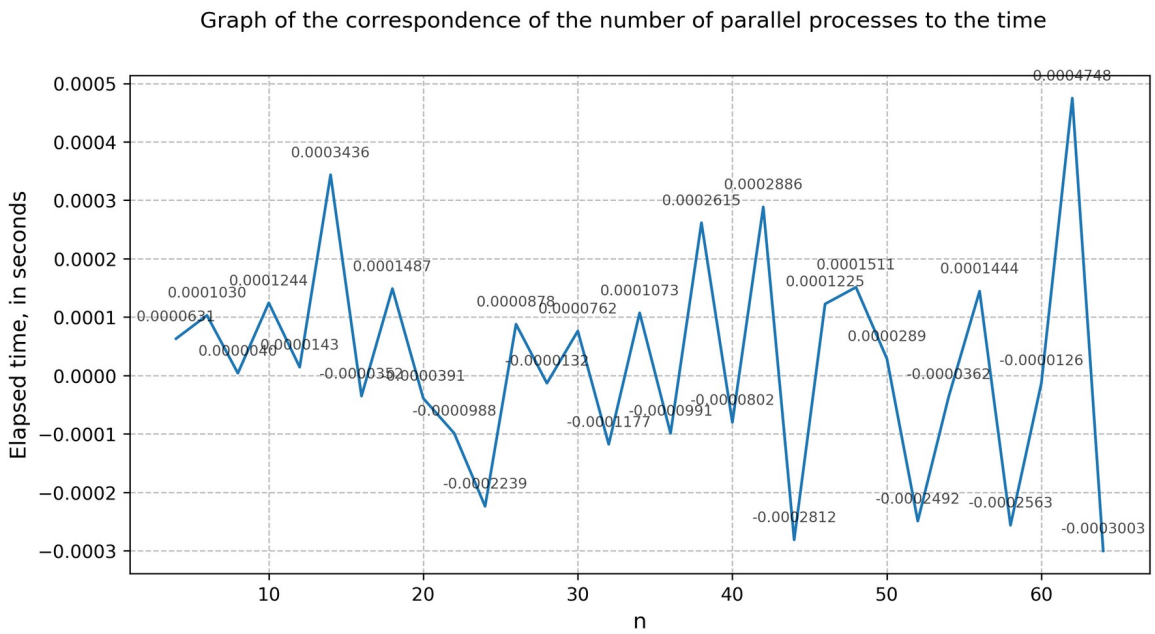

Рисунок 2 — Зависимость ускорения от количества процессов

Каждый замер произведён по 100 раз, из этих замеров отсеяны значения меньше 95-го квантиля. Если логически поразмыслить, форма графика должна быть примерно линейной (по крайней мере в первое время), так как здесь нет распараллеливания какой-то единой задачи. Эта оценка должна подтверждаться, но не подтверждается графиком: сложно понять причину такого поведения, скорее всего это связано с особенностями архитектуры процессора (Intel Core i5 12450H). Многократные повторы замеров показывают, что время выполнения растёт до 20 процессов, а далее падает.

Сеть Петри для реализованного алгоритма приведена в приложении А.

Пример работы написанной программы приведён в приложении В.

Листинг программ приведён в приложении С.

Выводы.

В ходе лабораторной работы изучены механизмы создания декартовых решёток и подрешёток с помощью библиотеки OpenMPI.

Написана программа, которая создаёт декартову решётку из всех доступных процессов, а далее делит эту решётку на две строки (количество процессов чётное, высота декартовой решётки равна 2). Произведены замеры времени и построена сеть Петри написанной программы.

ПРИЛОЖЕНИЕ А
СЕТЬ ПЕТРИ

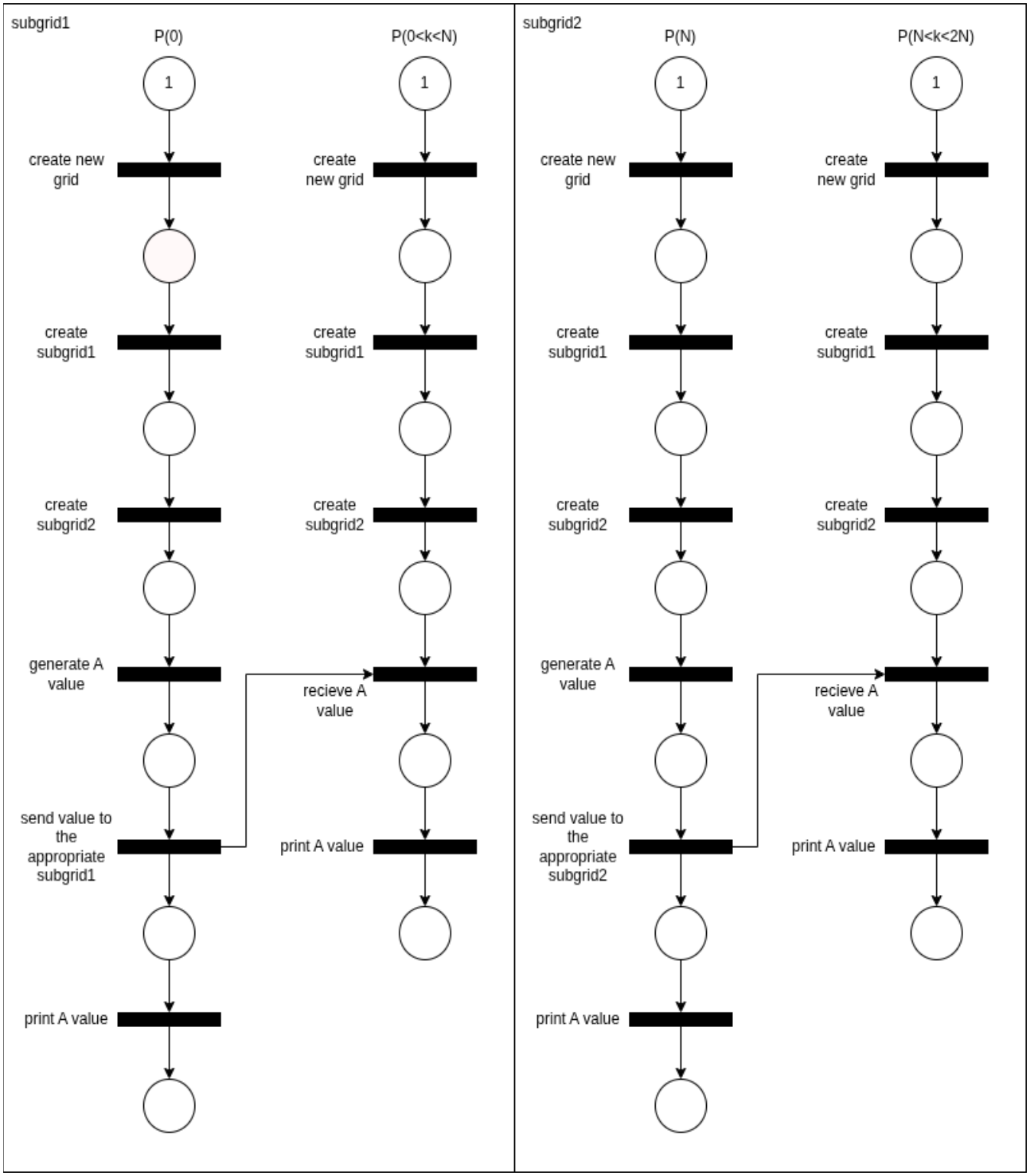

Рисунок А.1 — Сеть Петри

ПРИЛОЖЕНИЕ В
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Нечётное количество процессов на входе:

```
 $ mpirun -n 3 ./a.out
Process count must be even!
Unable to recover from error in task
```

Корректное выполнение программы:

```
 $ mpirun -n 8 ./a.out
rank=1: recieved A=4129.000000
rank=3: recieved A=4129.000000
rank=0: recieved A=4129.000000
rank=4: recieved A=422.000000
rank=7: recieved A=422.000000
rank=5: recieved A=422.000000
rank=2: recieved A=4129.000000
rank=6: recieved A=422.000000
```

При определённой директиве MEASURE_TIME:

```
 $ mpirun -n 8 ./a.out
0.000337
```

ПРИЛОЖЕНИЕ С
ЛИСТИНГ

Реализация задачи.

Название файла: lab5/src/task.c

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

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

int task(int world_size, int world_rank, int n_args, char* args[])
{
    int retcode = 0;

    if (world_size % 2 == 1) {
        retcode = 1;
        if (world_size == 0) {
            fprintf(stderr, "Process count must be even!\n");
        }
        return retcode;
    }

    int N = world_size / 2;

    // Create cartesian grid
    MPI_Comm grid_comm;
    int dims[2] = {2, N};
    int periods[2] = {0, 0};
    MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &grid_comm);

    // Create first subgrid
    MPI_Comm row1_comm;
    int remain_dims1[] = {1, 0};
    MPI_Cart_sub(grid_comm, remain_dims1, &row1_comm);

    // Create second subgrid
    MPI_Comm row2_comm;
    int remain_dims2[] = {0, 1};
    MPI_Cart_sub(grid_comm, remain_dims2, &row1_comm);

    // Generate A value
    float A = 0;
    if (world_rank == 0 || world_rank == N) {
        srand(time(NULL) + world_rank);
        A = (float)(rand() % 10000);
    }

    if (row1_comm != MPI_COMM_NULL) {
        MPI_Bcast(&A, 1, MPI_FLOAT, 0, row1_comm);
    } else {
        MPI_Bcast(&A, 1, MPI_FLOAT, N, row2_comm);
    }

    #ifndef MEASURE_TIME
    printf("rank=%d: recieved A=%f\n", world_rank, A);
    #endif

    return retcode;
}
```

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: lab5/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=(10, 10))
    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, 2)))
    )
    chart_tools.plot_chart(ax, X, Y)
    fig.legend(fontsize=12)
    fig.savefig("chart.png", 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: task.h

```
#ifndef TASK_H
#define TASK_H

#include <mpi.h>
#include <errno.h>

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

#endif
```

Название файла: timer.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();
}
```