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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Параллельные Вычисления»

Тема: «Численные методы»

Студент гр. 1307 _____ Тростин М. Ю.

Преподаватель _____

Санкт-Петербург

2025

Цели и задачи

Цель: Приобрести навыки в распараллеливании программы.

Задание (по вариантам). Представить последовательный и параллельный вариант программы, реализующей метод приведения разреженной матрицы к треугольному виду.

Выполнение работы

Описание алгоритма

1. Генерация матрицы

В данном задании «главный процесс» (с rank = 0) генерирует квадратную матриц размер которой зависит от количества запущенных процессов: кол-во элементов матрицы равен квадрату кол-ву запущенных процессов. Сгенерированная матрица рассылается остальным процессам

2. Распараллеленный метод исключения Гаусса

Данный метод выбран так как:

- Каждый процесс работает независимо над своей строкой
- Синхронизация не требуется во время исключения

Описание алгоритма:

1. Каждый процесс берёт свою сводную строку для нормализации: каждый элемент строки делится на элемент, находящийся на диагонали
2. Каждый процесс исключает сводный столбец в своей строке
3. Вычисляется коэффициент масштабирования
4. Из изначальной строки вычитается строка умноженная на коэффициент, приводя её к виду в котором все элементы до диагонального элемента равны нулю

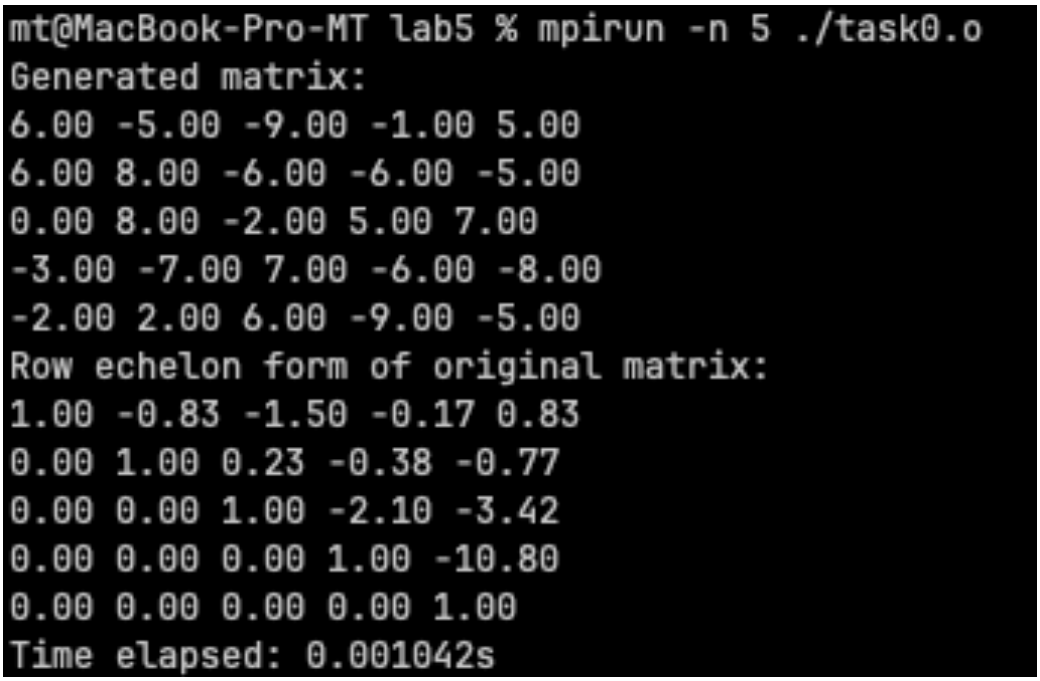
3. Сбор результата

Каждый процесс высылает полученную строку для сбора итоговой матрицы

В непараллельной программе временная сложность алгоритма составляет (в худшем случае) $O(n^3)$, однако при распараллеливании программы мы получаем сложность: $O(n^2 \log(n))$

Исходный код доступен в Приложении А

Скриншот выполнения



Результаты тестирования

Номер теста	**Кол-во элементов матрицы**	**Время выполнения (с)**
1	**16**	0.001109
2	**16**	0.000768
3	**36**	0.001021
4	**36**	0.001411
5	**64**	0.012008
6	**64**	0.009145
7	**256**	0.356410
8	**256**	0.341053
9	**1024**	1.400224
10	**1024**	1.550725
11	**1600**	1.619402
12	**1600**	1.473253
13	**2500**	3.973825
14	**2500**	4.136316
15	**3600**	5.707959
16	**3600**	6.560925

Выводы

В рамках выполнения данной работы были получены навыки в распараллеливании программы

ПРИЛОЖЕНИЕ А

Задание 1. Файл task1.cpp

```
#include <stdio.h>
#include <mpi.h>
#include <random>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>

#define element_t float
#define row_t     float *
#define matrix_t  float **

#define error_t unsigned char
#define ERR_VAL 1
#define ERR_CLK 2

#define BILLION  1000000000L
typedef struct SqareMatrix {
    matrix_t data;
    size_t size;
} SqaureMatrix;

void smPrint(SqareMatrix sm) {
    for (size_t i = 0; i < sm.size; i++) {
        for (size_t j = 0; j < sm.size; j++) {
            printf("%.2f ", sm.data[i][j]);
        }
        printf("\n");
    }
}

SqaureMatrix smEmpty(size_t size) {
    SqareMatrix sm;
    sm.size = size;
    sm.data = (matrix_t)malloc(sizeof(row_t) * sm.size);

    for (int i = 0; i < sm.size; i++) {
        sm.data[i] = (row_t)malloc(sizeof(element_t) * sm.size);
        for (int j = 0; j < sm.size; j++) {
            sm.data[i][j] = 0;
        }
    }

    return sm;
}

SqaureMatrix smGenerate(size_t size) {
    srand(time(NULL));
    SqareMatrix sm;
```

```
    sm.size = size;
    sm.data = (matrix_t)malloc(sizeof(row_t) * sm.size);

    for (int i = 0; i < sm.size; i++) {
        sm.data[i] = (row_t)malloc(sizeof(element_t) * sm.size);
        for (int j = 0; j < sm.size; j++) {
            sm.data[i][j] = (element_t)(rand() % 19 - 9);
        }
    }

    return sm;
}

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

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    struct timespec start, stop;
    if (rank == 0) {
        if (clock_gettime(CLOCK_REALTIME, &start) == -1) {
            printf("could not get clock time");
            return ERR_CLK;
        }
    }

    if (size < 2) {
        if (rank == 0) printf("Expected more than one process\n");
        MPI_Finalize();
        return ERR_VAL;
    }

    SqaureMatrix sm = rank == 0
        ? smGenerate(size)
        : smEmpty(size);

    if (rank == 0) {
        printf("Generated matrix:\n");
        smPrint(sm);
    }

    for (int i = 0; i < sm.size; i++) {
        MPI_Bcast(sm.data[i], sm.size, MPI_FLOAT, 0, MPI_COMM_WORLD);
    }

    for (int pivot = 0; pivot < sm.size; pivot++) {
        if (rank == pivot) {
            element_t pval = sm.data[pivot][pivot];
```

```
            for (int j = pivot; j < sm.size; j++) {
                sm.data[pivot][j] /= pval;
            }
        }

        MPI_Bcast(sm.data[pivot], sm.size, MPI_FLOAT, pivot,
MPI_COMM_WORLD);
        for (int i = pivot + 1; i < sm.size; i++) {
            if (rank == i) {
                element_t factor = sm.data[i][pivot];
                for (int j = pivot; j < sm.size; j++) {
                    sm.data[i][j] -= factor * sm.data[pivot][j];
                }
            }
        }
    }

    if (rank == 0) {
        if (clock_gettime(CLOCK_REALTIME, &stop) == -1) {
            printf("could not get stop clock time");
            return ERR_CLK;
        }

        printf("Row echelon form of original matrix:\n");
        smPrint(sm);

        double accum = ( stop.tv_sec - start.tv_sec )
             + (double)( stop.tv_nsec - start.tv_nsec )
               / (double)BILLION;
        printf( "Time elapsed: %lfs\n", accum);
    }

    MPI_Finalize();
    return 0;
}
```