Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного автономного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

ЛАБОРАТОРНАЯ РАБОТА №3

«Основные директивы OpenMP»

ДИСЦИПЛИНА: «Параллельные процессы в информационных системах»

Выполнил: студент гр. ИУК4-31М _____ (Сафронов Н.С.)
(Подпись) (Ф.И.О.)

Проверил: _____ (Корнюшин Ю.П.)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2025

Цель: формирование практических навыков построения простых параллельных программ на языке параллельного программирования OpenMP.

Задачи

1. Получить представление о построении простых параллельных программ на языке параллельного программирования OpenMP.
2. Закрепить практическое освоение директив языка реализовав программу с использованием директив OpenMP.

Задание

Вариант 6

Для реализации параллельной программы должен использоваться язык программирования OpenMP.

Задание 1

Проработка примеров из пункта "Примеры параллельных программ" этой же лабораторной.

Внимательно изучить примеры 1-5. Откомпилировать и запустить на 2-х процессах.

Задание 2

Разработать алгоритм написать и отладить параллельную программу умножения матрицы на матрицу с использованием директивы распараллеливания цикла по виткам.

Задание 3

Сравнительные временные характеристики двух алгоритмов умножения матрицы на вектор на языке MPI и языке OpenMP.

В качестве параллельных программ взять параллельные программы: «умножение матрицы на вектор в MPI на топологии "кольцо"» и «умножение матрицы на вектор в OpenMP (один из вариантов)». Для обеих программ построить небольшие графики зависимости времени решения задачи от размеров матрицы и вектора. Обе программы запустить последовательно для матриц и соответствующих этим матрицам векторов:

Вариант	Матрицы	Вектор
6	[300 x 300]	300
	[530 x 530]	530
	[710 x 710]	710
	[1030 x 1030]	1030

Результат выполнения работы

Задание 1

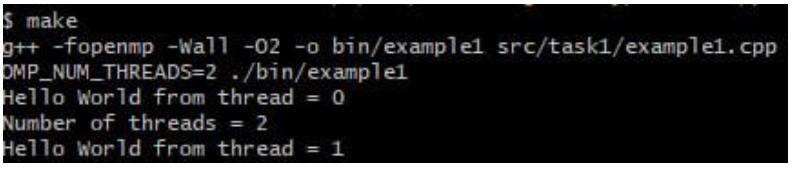

Рисунок 1 – Результат выполнения примера 1

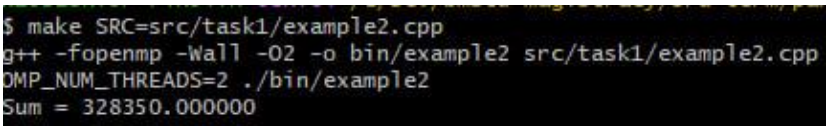

Рисунок 2 – Результат выполнения примера 2

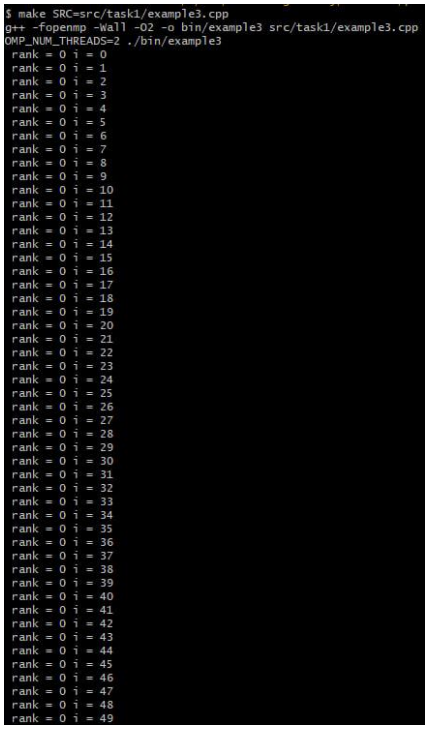

Рисунок 3 – Результат выполнения примера 3

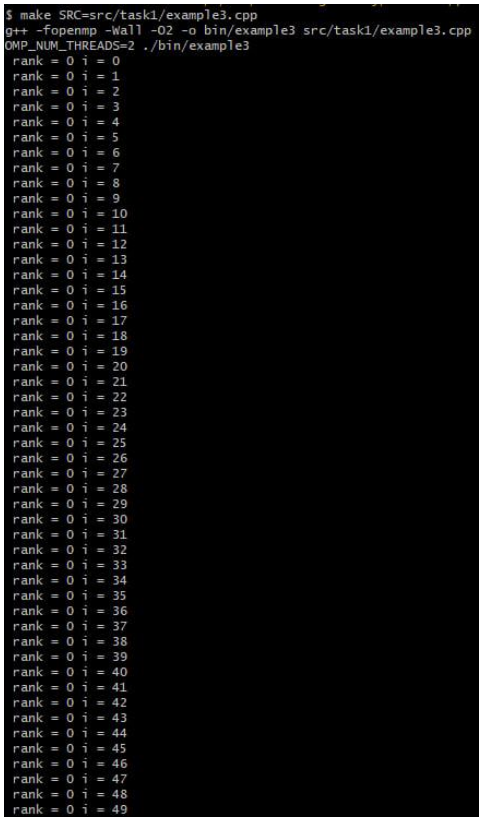

Рисунок 3 – Результат выполнения примера 3

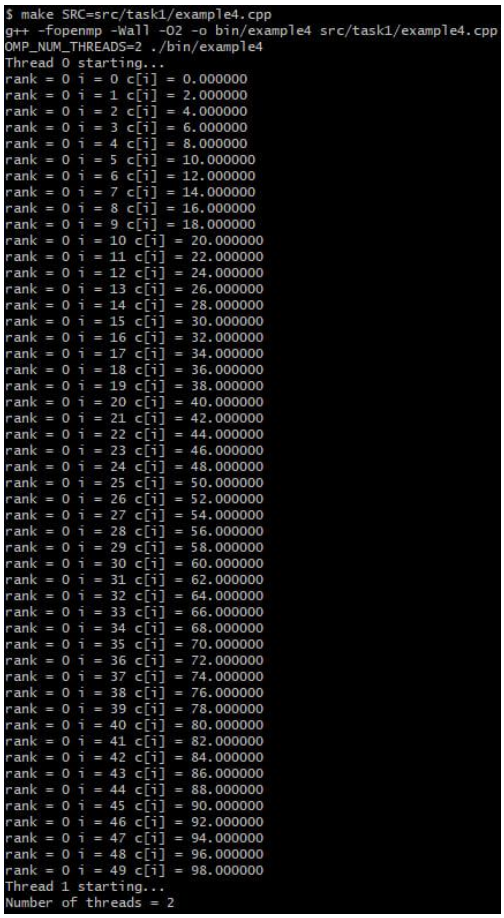

Рисунок 4 – Результат выполнения примера 4

```
$ make SRC=src/task1/example5.cpp
g++ -fopenmp -Wall -O2 -o bin/example5 src/task1/example5.cpp
OMP_NUM_THREADS=2 ./bin/example5

Matrix A and vector b:
  A[0] = 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0   b[0] = 1.0
  A[1] = 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0   b[1] = 2.0
  A[2] = 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0   b[2] = 3.0
  A[3] = 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0   b[3] = 4.0
  A[4] = 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0   b[4] = 5.0
  A[5] = 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0   b[5] = 6.0
  A[6] = 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0   b[6] = 7.0
  A[7] = 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0   b[7] = 8.0
  A[8] = 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0   b[8] = 9.0
  A[9] = 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0   b[9] = 10.0
 rank = 0 i = 0 c[0] = 385.00
 rank = 0 i = 1 c[1] = 385.00
 rank = 0 i = 2 c[2] = 385.00
 rank = 0 i = 3 c[3] = 385.00
 rank = 0 i = 4 c[4] = 385.00
 rank = 1 i = 5 c[5] = 385.00
 rank = 1 i = 6 c[6] = 385.00
 rank = 1 i = 7 c[7] = 385.00
 rank = 1 i = 8 c[8] = 385.00
 rank = 1 i = 9 c[9] = 385.00
```

Рисунок 5 – Результат выполнения примера 5

Задание 2

```
$ make SRC=src/task2.cpp
g++ -fopenmp -Wall -O2 -o bin/task2 src/task2.cpp
OMP_NUM_THREADS=2 ./bin/task2
Time elapsed - 0.0340 s
C[0][0] = 9090200.00
```

Рисунок 6 – Параллельное умножение матрицы на матрицу

Листинг

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

#define N 500
#define K 300
#define M 400

int main()
{
    double (*A)[K] = (double(*)[K])malloc(N * K * sizeof(double));
    double (*B)[M] = (double(*)[M])malloc(K * M * sizeof(double));
    double (*C)[M] = (double(*)[M])malloc(N * M * sizeof(double));

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < K; j++) {
            A[i][j] = i + j + 1;
        }
    }
    for (int i = 0; i < K; i++) {
        for (int j = 0; j < M; j++) {
            B[i][j] = i - j + 2;
        }
    }
```

```
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            C[i][j] = 0.0;
        }
    }

    double start = omp_get_wtime();

    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            for (int k = 0; k < K; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    double end = omp_get_wtime();

    printf("Time elapsed - %.4f s\n", end - start);
    printf("C[0][0] = %.2f\n", C[0][0]);

    free(A);
    free(B);
    free(C);

    return 0;
}
```

Задание 3

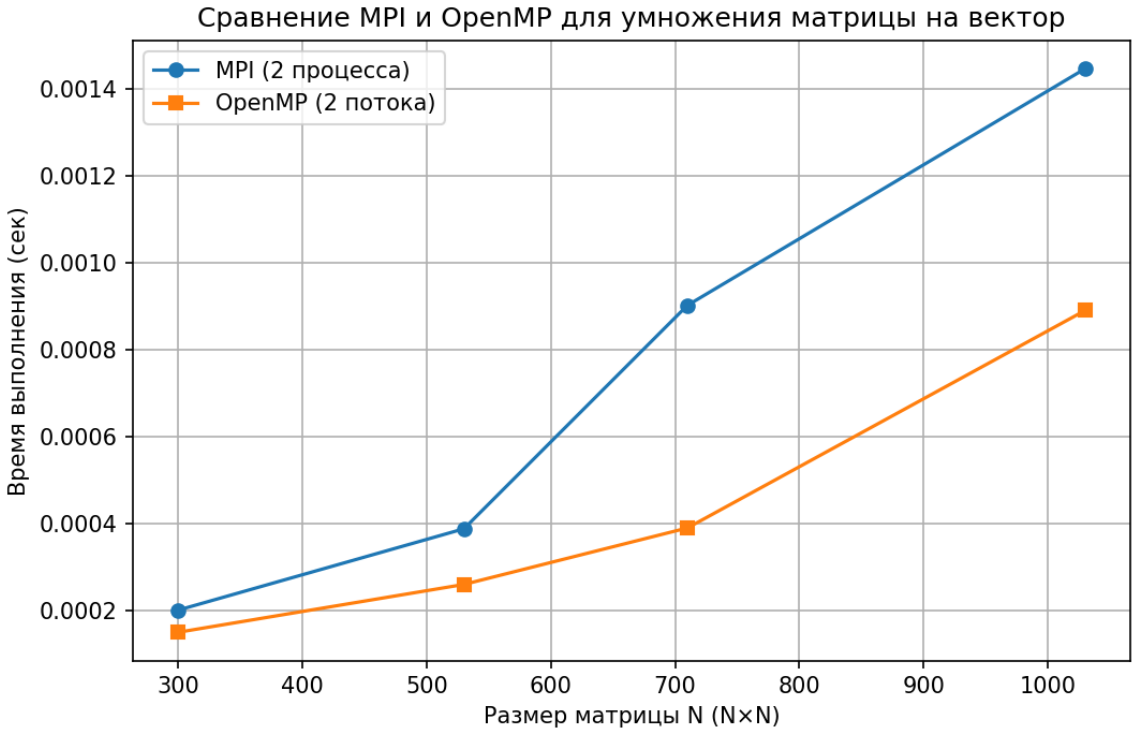

Рисунок 7 – Сравнение MPI и OpenMP для умножения матрицы на вектор

Листинг программы на MPI

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

int main(int argc, char **argv)
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (size != 2) {
        if (rank == 0) fprintf(stderr, "Execute with 2 processes!\n");
        MPI_Finalize();
        return 1;
    }

    if (argc != 2) {
        if (rank == 0) fprintf(stderr, "Usage: %s <N>\n", argv[0]);
        MPI_Finalize();
        return 1;
    }

    int N = atoi(argv[1]);
    int local_rows = N / size + (rank < N % size ? 1 : 0);
    int offset = rank * (N / size) + (rank < N % size ? rank : N % size);

    double *A_local = (double*)malloc(local_rows * N * sizeof(double));
    double *B = (double*)malloc(N * sizeof(double));
    double *C_local = (double*)calloc(local_rows, sizeof(double));

    for (int i = 0; i < local_rows; i++)
        for (int j = 0; j < N; j++)
            A_local[i * N + j] = 1.0;

    for (int j = 0; j < N; j++)
        B[j] = 2.0;

    MPI_Barrier(MPI_COMM_WORLD);
    double t1 = MPI_Wtime();

    for (int i = 0; i < local_rows; i++) {
        C_local[i] = 0.0;
        for (int j = 0; j < N; j++) {
            C_local[i] += A_local[i * N + j] * B[j];
        }
    }

    double t2 = MPI_Wtime();
    double local_time = t2 - t1;

    double max_time;
    MPI_Reduce(&local_time,    &max_time,    1,    MPI_DOUBLE,    MPI_MAX,    0,
MPI_COMM_WORLD);

    if (rank == 0) {
        printf("%d %.6f\n", N, max_time);
    }

    free(A_local);
    free(B);
    free(C_local);
    MPI_Finalize();
    return 0;
```

```
}
```

Листинг программы на OpenMP

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

int main(int argc, char **argv)
{
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <N>\n", argv[0]);
        return 1;
    }

    int N = atoi(argv[1]);
    const int REPS = 100;

    double *A = (double*)malloc(N * N * sizeof(double));
    double *B = (double*)malloc(N * sizeof(double));
    double *C = (double*)calloc(N, sizeof(double));

    srand(42);
    for (int i = 0; i < N * N; i++) {
        A[i] = (double)rand() / RAND_MAX;
    }
    for (int i = 0; i < N; i++) {
        B[i] = (double)rand() / RAND_MAX;
    }

    double t1 = omp_get_wtime();
    for (int rep = 0; rep < REPS; rep++) {
        #pragma omp parallel for
        for (int i = 0; i < N; i++) {
            C[i] = 0.0;
            for (int j = 0; j < N; j++) {
                C[i] += A[i * N + j] * B[j];
            }
        }
    }

    double t2 = omp_get_wtime();
    double avg_time = (t2 - t1) / REPS;

    // Используем результат
    double checksum = 0.0;
    for (int i = 0; i < N; i++) checksum += C[i];
    (void)checksum; // подавляем предупреждение

    printf("%d %.9f\n", N, avg_time);

    free(A);
    free(B);
    free(C);
    return 0;
}
```

Вывод: в ходе выполнения лабораторной работы были сформированы практические навыки построения простых параллельных программ на языке параллельного программирования OpenMP