



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

(Подпись)

(Сафронов Н.С.)
(Ф.И.О.)

Проверил:

(Подпись)

(Корнюшин Ю.П.)
(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

```
$ make
g++ -fopenmp -Wall -O2 -o bin/example1 src/task1/example1.cpp
OMP_NUM_THREADS=2 ./bin/example1
Hello World from thread = 0
Number of threads = 2
Hello World from thread = 1
```

Рисунок 1 – Результат выполнения примера 1

```
$ make SRC=src/task1/example2.cpp
g++ -fopenmp -Wall -O2 -o bin/example2 src/task1/example2.cpp
OMP_NUM_THREADS=2 ./bin/example2
Sum = 328350.000000
```

Рисунок 2 – Результат выполнения примера 2

```
$ make SRC=src/task1/example3.cpp
g++ -fopenmp -Wall -O2 -o bin/example3 src/task1/example3.cpp
OMP_NUM_THREADS=2 ./bin/example3
rank = 0 i = 0
rank = 0 i = 1
rank = 0 i = 2
rank = 0 i = 3
rank = 0 i = 4
rank = 0 i = 5
rank = 0 i = 6
rank = 0 i = 7
rank = 0 i = 8
rank = 0 i = 9
rank = 0 i = 10
rank = 0 i = 11
rank = 0 i = 12
rank = 0 i = 13
rank = 0 i = 14
rank = 0 i = 15
rank = 0 i = 16
rank = 0 i = 17
rank = 0 i = 18
rank = 0 i = 19
rank = 0 i = 20
rank = 0 i = 21
rank = 0 i = 22
rank = 0 i = 23
rank = 0 i = 24
rank = 0 i = 25
rank = 0 i = 26
rank = 0 i = 27
rank = 0 i = 28
rank = 0 i = 29
rank = 0 i = 30
rank = 0 i = 31
rank = 0 i = 32
rank = 0 i = 33
rank = 0 i = 34
rank = 0 i = 35
rank = 0 i = 36
rank = 0 i = 37
rank = 0 i = 38
rank = 0 i = 39
rank = 0 i = 40
rank = 0 i = 41
rank = 0 i = 42
rank = 0 i = 43
rank = 0 i = 44
rank = 0 i = 45
rank = 0 i = 46
rank = 0 i = 47
rank = 0 i = 48
rank = 0 i = 49
```

Рисунок 3 – Результат выполнения примера 3

```

$ make SRC=src/task1/example3.cpp
g++ -fopenmp -Wall -O2 -o bin/example3 src/task1/example3.cpp
OMP_NUM_THREADS=2 ./bin/example3
rank = 0 i = 0
rank = 0 i = 1
rank = 0 i = 2
rank = 0 i = 3
rank = 0 i = 4
rank = 0 i = 5
rank = 0 i = 6
rank = 0 i = 7
rank = 0 i = 8
rank = 0 i = 9
rank = 0 i = 10
rank = 0 i = 11
rank = 0 i = 12
rank = 0 i = 13
rank = 0 i = 14
rank = 0 i = 15
rank = 0 i = 16
rank = 0 i = 17
rank = 0 i = 18
rank = 0 i = 19
rank = 0 i = 20
rank = 0 i = 21
rank = 0 i = 22
rank = 0 i = 23
rank = 0 i = 24
rank = 0 i = 25
rank = 0 i = 26
rank = 0 i = 27
rank = 0 i = 28
rank = 0 i = 29
rank = 0 i = 30
rank = 0 i = 31
rank = 0 i = 32
rank = 0 i = 33
rank = 0 i = 34
rank = 0 i = 35
rank = 0 i = 36
rank = 0 i = 37
rank = 0 i = 38
rank = 0 i = 39
rank = 0 i = 40
rank = 0 i = 41
rank = 0 i = 42
rank = 0 i = 43
rank = 0 i = 44
rank = 0 i = 45
rank = 0 i = 46
rank = 0 i = 47
rank = 0 i = 48
rank = 0 i = 49

```

Рисунок 3 – Результат выполнения примера 3

```

$ make SRC=src/task1/example4.cpp
g++ -fopenmp -Wall -O2 -o bin/example4 src/task1/example4.cpp
OMP_NUM_THREADS=2 ./bin/example4
Thread 0 starting...
rank = 0 i = 0 c[i] = 0.000000
rank = 0 i = 1 c[i] = 2.000000
rank = 0 i = 2 c[i] = 4.000000
rank = 0 i = 3 c[i] = 6.000000
rank = 0 i = 4 c[i] = 8.000000
rank = 0 i = 5 c[i] = 10.000000
rank = 0 i = 6 c[i] = 12.000000
rank = 0 i = 7 c[i] = 14.000000
rank = 0 i = 8 c[i] = 16.000000
rank = 0 i = 9 c[i] = 18.000000
rank = 0 i = 10 c[i] = 20.000000
rank = 0 i = 11 c[i] = 22.000000
rank = 0 i = 12 c[i] = 24.000000
rank = 0 i = 13 c[i] = 26.000000
rank = 0 i = 14 c[i] = 28.000000
rank = 0 i = 15 c[i] = 30.000000
rank = 0 i = 16 c[i] = 32.000000
rank = 0 i = 17 c[i] = 34.000000
rank = 0 i = 18 c[i] = 36.000000
rank = 0 i = 19 c[i] = 38.000000
rank = 0 i = 20 c[i] = 40.000000
rank = 0 i = 21 c[i] = 42.000000
rank = 0 i = 22 c[i] = 44.000000
rank = 0 i = 23 c[i] = 46.000000
rank = 0 i = 24 c[i] = 48.000000
rank = 0 i = 25 c[i] = 50.000000
rank = 0 i = 26 c[i] = 52.000000
rank = 0 i = 27 c[i] = 54.000000
rank = 0 i = 28 c[i] = 56.000000
rank = 0 i = 29 c[i] = 58.000000
rank = 0 i = 30 c[i] = 60.000000
rank = 0 i = 31 c[i] = 62.000000
rank = 0 i = 32 c[i] = 64.000000
rank = 0 i = 33 c[i] = 66.000000
rank = 0 i = 34 c[i] = 68.000000
rank = 0 i = 35 c[i] = 70.000000
rank = 0 i = 36 c[i] = 72.000000
rank = 0 i = 37 c[i] = 74.000000
rank = 0 i = 38 c[i] = 76.000000
rank = 0 i = 39 c[i] = 78.000000
rank = 0 i = 40 c[i] = 80.000000
rank = 0 i = 41 c[i] = 82.000000
rank = 0 i = 42 c[i] = 84.000000
rank = 0 i = 43 c[i] = 86.000000
rank = 0 i = 44 c[i] = 88.000000
rank = 0 i = 45 c[i] = 90.000000
rank = 0 i = 46 c[i] = 92.000000
rank = 0 i = 47 c[i] = 94.000000
rank = 0 i = 48 c[i] = 96.000000
rank = 0 i = 49 c[i] = 98.000000
Thread 1 starting...
Number of threads = 2

```

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

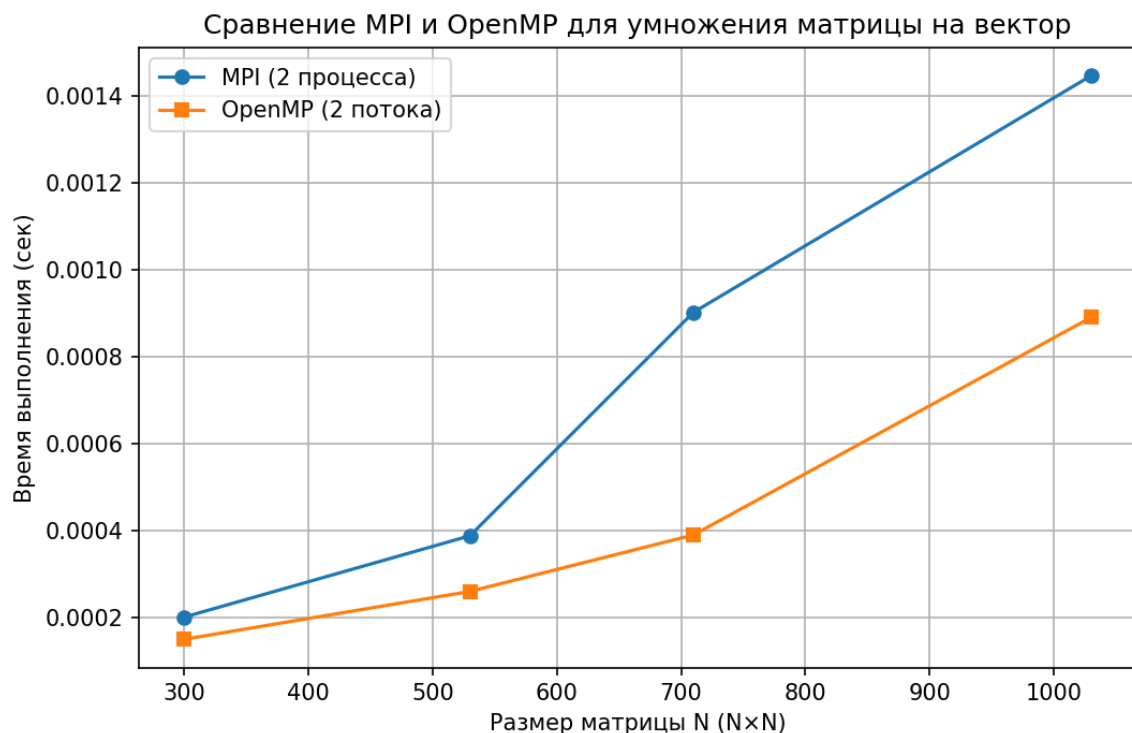


Рисунок 7 – Сравнение MPI и OpenMP для умножения матрицы на вектор

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