МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №3

по дисциплине: Параллельное программирование
тема: «Гибридное параллельное программирование с использованием OpenMP + MPI»

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович

Проверили:
доц. Островский Алексей Мичеславович

Белгород 2025 г.

Лабораторная работа №3

Гибридное параллельное программирование с использованием OpenMP + MPI.

Цель работы: Изучить возможности гибридного подхода к параллельному программированию с использованием стандартов MPI и OpenMP, реализовать смешанную модель параллельных вычислений, оценить её эффективность по сравнению с отдельными технологиями, изучить механизмы межпроцессного взаимодействия (MPI) и многопоточного параллелизма (OpenMP)..

Цель работы обуславливает постановку и решение следующих задач:

1. Ознакомиться с основами MPI + OpenMP, включая компиляцию и запуск гибридных программ.
2. Изучить особенности модели MPI и многопоточности OpenMP.
3. Научиться программировать гибридное параллельное приложение:

3.1 Научиться декомпозировать вычислительную нагрузку между процессами (MPI)

3.2 Научиться выполнять параллельную обработку внутри процессов (OpenMP) в условиях гибридного приложения

4. Выполнить индивидуальное задание, связанное с гибридной обработкой данных: декомпозировать задачу на межпроцессные и внутрипроцессные фрагменты, обратить внимание на балансировку нагрузки и взаимодействие между уровнями.
5. Провести экспериментальную оценку масштабируемости, потенциала ускорения и накладных расходов для различных конфигураций. Сделать выводы о применимости гибридной модели.

Индивидуальное задание

Вариант 3

Гибридная реализация матричного умножения. Реализовать умножение двух квадратных матриц большого размера. Разделить строки матрицы A между MPI-процессами. $C = A \cdot B$. Внутри каждого процесса параллельно (OpenMP) выполнить умножение с матрицей B. Использовать MPI_Gather для сборки итоговой матрицы C на главном процессе. Провести сравнение с последовательной реализацией.

Ход выполнения лабораторной работы

Файл matrix_mul_combined.c

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

#define N 1024  // Размер квадратной матрицы

void fill_matrix(int* mat, int n, int value) {
    for (int i = 0; i < n * n; i++)
        mat[i] = value;
}

void matrix_multiply_seq(int* A, int* B, int* C, int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            int sum = 0;
            for (int k = 0; k < n; k++)
                sum += A[i * n + k] * B[k * n + j];
            C[i * n + j] = sum;
        }
}

void run_sequential() {
    int* A = (int*)malloc(N * N * sizeof(int));
    int* B = (int*)malloc(N * N * sizeof(int));
    int* C = (int*)malloc(N * N * sizeof(int));

    fill_matrix(A, N, 1);
    fill_matrix(B, N, 2);

    clock_t start = clock();

    matrix_multiply_seq(A, B, C, N);

    clock_t end = clock();
    double time_taken = (double)(end - start) / CLOCKS_PER_SEC;

    printf("Последовательное умножение завершено.\n");
    printf("Пример элемента C[0][0]: %d\n", C[0]);
    printf("Время выполнения: %.3f секунд\n", time_taken);

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

void run_hybrid(int argc, char* argv[]) {
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int rows_per_proc = N / size;
    int* A = NULL;
    int* B = (int*)malloc(N * N * sizeof(int));
    int* C_local = (int*)malloc(rows_per_proc * N * sizeof(int));
    int* A_local = (int*)malloc(rows_per_proc * N * sizeof(int));
    int* C = NULL;
```

```
    if (rank == 0) {
        A = (int*)malloc(N * N * sizeof(int));
        C = (int*)malloc(N * N * sizeof(int));
        fill_matrix(A, N, 1);
        fill_matrix(B, N, 2);
    }

    double t_start = MPI_Wtime();

    MPI_Scatter(A, rows_per_proc * N, MPI_INT, A_local, rows_per_proc * N, MPI_INT, 0,
MPI_COMM_WORLD);
    MPI_Bcast(B, N * N, MPI_INT, 0, MPI_COMM_WORLD);

    #pragma omp parallel for
    for (int i = 0; i < rows_per_proc; i++) {
        for (int j = 0; j < N; j++) {
            int sum = 0;
            for (int k = 0; k < N; k++)
                sum += A_local[i * N + k] * B[k * N + j];
            C_local[i * N + j] = sum;
        }
    }

    MPI_Gather(C_local, rows_per_proc * N, MPI_INT, C, rows_per_proc * N, MPI_INT, 0,
MPI_COMM_WORLD);

    double t_end = MPI_Wtime();

    if (rank == 0) {
        printf("Гибридное умножение завершено.\n");
        printf("Пример элемента C[0][0]: %d\n", C[0]);
        printf("Время выполнения: %.3f секунд\n", t_end - t_start);
        free(A);
        free(C);
    }

    free(B);
    free(A_local);
    free(C_local);

    MPI_Finalize();
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        if (argc == 1) {
            printf("Укажите режим запуска: seq (последовательно) или hybrid (MPI + OpenMP)\n");
            return 1;
        }
    }

    if (strcmp(argv[1], "seq") == 0) {
        run_sequential();
    } else if (strcmp(argv[1], "hybrid") == 0) {
        run_hybrid(argc, argv);
    } else {
        printf("Неизвестный режим: %s\n", argv[1]);
        printf("Доступные режимы: seq, hybrid\n");
        return 1;
    }

    return 0;
}
```

Компиляция:

```
mpicxx -fopenmp matrix_mul_combined.c -o matrix_mul
```

Запуск:

Последовательная версия:

```
./matrix_mul seq
```

Гибридная версия (4 процесса по 4 потока):

```
OMP_NUM_THREADS=4 mpirun -np 4 ./matrix_mul hybrid
```

Результаты:

Последовательная версия	4.899

Гибридная версия					
		Количество потоков			
		1	2	3	4
Количество процессов	1	4.527	2.020	1.952	1.945
	2	2.319	1.286	1.267	1.299
	3	1.707	1.300	1.077	1.142
	4	1.625	1.191	1.082	0.971

Работа программы:

```
user@Entik-Notebook:~$ mpicxx -fopenmp matrix_mul_combined.c -o matrix_mul
user@Entik-Notebook:~$ ./matrix_mul seq
Последовательное умножение завершено.
Пример элемента C[0][0]: 2048
Время выполнения: 4.899 секунд
user@Entik-Notebook:~$ OMP_NUM_THREADS=4 mpirun -np 4 ./matrix_mul hybrid
Гибридное умножение завершено.
Пример элемента C[0][0]: 2048
Время выполнения: 0.971 секунд
```

Во всех запусках пример элемента C[0][0] равен 2048, что подтверждает корректность вычислений как в последовательной, так и в гибридной реализации.

Последовательная реализация показала наихудшее время выполнения — 4.899 секунд, что ожидаемо, учитывая отсутствие какой-либо параллелизации.

Гибридная модель продемонстрировала значительное ускорение.

При использовании 1 процесса и 4 потоков время уменьшилось до 1.945 секунд, то есть ускорение более чем в 2.5 раза.

При увеличении числа MPI-процессов до 4 и использовании 4 OpenMP-потоков удалось достичь наилучшего времени выполнения — 0.971 секунд.

Увеличение потоков внутри одного процесса даёт заметное ускорение (с 4.527 до 1.945 при np=1), однако комбинирование MPI и OpenMP даёт лучшую масштабируемость.

Эффективность падает при несбалансированном числе потоков и процессов. Например, np=2 и OMP_NUM_THREADS=4 работает хуже, чем np=4 и OMP_NUM_THREADS=3, что связано с накладными расходами на синхронизацию и ограничениями планировщика.

Вывод: Гибридная модель параллельного программирования (MPI + OpenMP) обеспечивает значительное ускорение по сравнению с последовательной реализацией.

Оптимальная производительность достигается при сбалансированном распределении нагрузки между MPI-процессами и OpenMP-потоками.

Гибридный подход обладает лучшей масштабируемостью, что делает его предпочтительным для задач, требующих высокопроизводительных вычислений.