|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 х 300] [530 х 530] [710 х 710]  [1030 х 1030] | 300  530  710  1030 |

**Результат выполнения работы**

**Задание 1**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 1** – Результат выполнения примера 1

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 2** – Результат выполнения примера 2

Изображение выглядит как снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 3** – Результат выполнения примера 3

Изображение выглядит как снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 3** – Результат выполнения примера 3

Изображение выглядит как снимок экрана, монохромный, искусство

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 4** – Результат выполнения примера 4

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 5** – Результат выполнения примера 5

**Задание 2**

Изображение выглядит как текст, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

**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**

Изображение выглядит как текст, линия, График, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

**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