|  |  |
| --- | --- |
| 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 №2**

**«Параллельные программы умножения матрицы на вектор»**

**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 программирования параллельных процессов, исполняющихся на декартовой топологии связей и топологии "граф".

**Задачи**

1. Освоить функции задания декартовой топологии и топологии "граф".
2. Освоить методы распараллеливания алгоритмов решения задач, таких как умножение матрицы на вектор.

**Задание**

**Вариант 6**

Программа должна быть реализована на языке программирования С++ с использованием технологии MPI.

**Задание 1**

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

Внимательно изучить примеры 1, 2, 3. Примеры 2 и 3 (Умножение матрицы на вектор на топологиях "кольцо" и "полный граф") откомпилировать и запустить на 4-х компьютерах.

**Задание 2**

Параллельное умножение матрицы на вектор на топологии "кольцо".

Разработать алгоритм, написать и отладить параллельную программу умножения матрицы на вектор в топологии "кольцо": , при условии, что количество строк матрицы и элементов вектора нацело делится на количество компьютеров. Например, матрица размером [20х20] и вектор размером [20] на четырех компьютерах.

**Задание 3**

Параллельное умножение матрицы на матрицу на топологии "кольцо".

Разработать алгоритм написать и отладить параллельную программу умножения матрицы на матрицу в топологии "кольцо": , при условии, что количество строк матрицы и столбцов матрицы нацело не делится на количество компьютеров. Например, матрица размером [31х20] и матрица размером [20х31] на четырех компьютерах.

**Задание 4**

Сравнительные временные характеристики двух алгоритмов умножения матрицы на вектор на топологиях "кольцо" и "полный граф".

В качестве параллельных программ взять примеры 2 и 3 (Умножение матрицы на вектор на топологиях "кольцо" и "полный граф"). 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

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 1** – Результат выполнения примера 2

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.**Рисунок 2** – Результат выполнения примера 3

**Задание 2**

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 3** – Параллельное умножение матрицы на вектор на топологии "кольцо" А размером [20х20] и вектор В размером [20] на четырех компьютерах

**Листинг**

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define ROWS 20

#define COLS 20

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 (ROWS % size != 0) {

if (rank == 0)

fprintf(stderr, "Error: ROWS (%d) can't be divided by size (%d) returning whole number\n", ROWS, size);

MPI\_Finalize();

return 1;

}

const int local\_rows = ROWS / size;

const int block = COLS / size;

double \*A\_local = (double \*)malloc(local\_rows \* COLS \* sizeof(double));

double \*B\_block = (double \*)malloc(block \* sizeof(double));

double \*C\_local = (double \*)calloc(local\_rows, sizeof(double));

for (int i = 0; i < local\_rows; i++) {

for (int j = 0; j < COLS; j++) {

A\_local[i \* COLS + j] = 1.0;

}

}

for (int i = 0; i < local\_rows; i++) {

for (int j = 0; j < COLS; j++) {

C\_local[i] += 3.0;

}

}

for (int j = 0; j < block; j++) {

B\_block[j] = 2.0;

}

// Создание топологии "кольцо"

int dims[1] = {0};

int periods[1] = {1}; // периодическое кольцо

MPI\_Dims\_create(size, 1, dims);

MPI\_Comm ring\_comm;

MPI\_Cart\_create(MPI\_COMM\_WORLD, 1, dims, periods, 0, &ring\_comm);

int src, dst;

MPI\_Cart\_shift(ring\_comm, 0, 1, &src, &dst);

double start\_time = MPI\_Wtime();

for (int step = 0; step < size; step++) {

int col\_offset = ((rank - step + size) % size) \* block;

// Накопление: C\_local += A\_local[:, col\_offset : col\_offset+block] \* B\_block

for (int i = 0; i < local\_rows; i++) {

for (int j = 0; j < block; j++) {

C\_local[i] += A\_local[i \* COLS + col\_offset + j] \* B\_block[j];

}

}

// Циклический сдвиг блока вектора по кольцу

if (size > 1) {

MPI\_Sendrecv\_replace(B\_block, block, MPI\_DOUBLE,

dst, 0, src, 0, ring\_comm, MPI\_STATUS\_IGNORE);

}

}

double end\_time = MPI\_Wtime();

double elapsed = end\_time - start\_time;

// Сбор полного результата на процессе 0

double \*C\_full = NULL;

if (rank == 0) {

C\_full = (double \*)malloc(ROWS \* sizeof(double));

}

MPI\_Gather(C\_local, local\_rows, MPI\_DOUBLE,

C\_full, local\_rows, MPI\_DOUBLE,

0, MPI\_COMM\_WORLD);

printf("rank = %d: time = %.6f s\n", rank, elapsed);

if (rank == 0) {

printf("\nC = A \* B (first 10 elements):\n");

for (int i = 0; i < (ROWS < 10 ? ROWS : 10); i++) {

printf("C[%d] = %.2f\n", i, C\_full[i]);

}

free(C\_full);

}

free(A\_local);

free(B\_block);

free(C\_local);

MPI\_Comm\_free(&ring\_comm);

MPI\_Finalize();

return 0;

}

**Задание 3**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 4** – Параллельное умножение матрицы на матрицу на топологии "кольцо" А размером [31х20] и матрица В размером [20х31] на четырех компьютерах

**Листинг**

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define ROWS\_A 31

#define COLS\_A 20

#define COLS\_B 31

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);

// Создание топологии "кольцо" (обязательно по условию)

int dims[1] = {0};

int periods[1] = {1};

MPI\_Dims\_create(size, 1, dims);

MPI\_Comm ring\_comm;

MPI\_Cart\_create(MPI\_COMM\_WORLD, 1, dims, periods, 0, &ring\_comm);

// Неравномерное распределение строк A

int local\_rows = ROWS\_A / size;

int remainder = ROWS\_A % size;

if (rank < remainder) {

local\_rows++; // первые 'остаток' процессов получают +1 строку

}

// Смещение начала строк для текущего процесса

int offset = rank \* (ROWS\_A / size) + (rank < remainder ? rank : remainder);

double \*A\_local = (double \*)malloc(local\_rows \* COLS\_A \* sizeof(double));

double \*B = (double \*)malloc(COLS\_A \* COLS\_B \* sizeof(double));

double \*C\_local = (double \*)calloc(local\_rows \* COLS\_B, sizeof(double));

// Инициализация данных

for (int i = 0; i < local\_rows; i++) {

for (int j = 0; j < COLS\_A; j++) {

A\_local[i \* COLS\_A + j] = (double)(offset + i + j + 1);

}

}

for (int i = 0; i < COLS\_A; i++) {

for (int j = 0; j < COLS\_B; j++) {

B[i \* COLS\_B + j] = (double)(i + j + 2);

}

}

// Локальное умножение

double start\_time = MPI\_Wtime();

for (int i = 0; i < local\_rows; i++) {

for (int k = 0; k < COLS\_A; k++) {

double a\_ik = A\_local[i \* COLS\_A + k];

for (int j = 0; j < COLS\_B; j++) {

C\_local[i \* COLS\_B + j] += a\_ik \* B[k \* COLS\_B + j];

}

}

}

double end\_time = MPI\_Wtime();

double elapsed = end\_time - start\_time;

int \*recvcounts = NULL;

int \*displs = NULL;

double \*C\_full = NULL;

if (rank == 0) {

recvcounts = (int \*)malloc(size \* sizeof(int));

displs = (int \*)malloc(size \* sizeof(int));

C\_full = (double \*)malloc(ROWS\_A \* COLS\_B \* sizeof(double));

int total = 0;

for (int r = 0; r < size; r++) {

int lr = ROWS\_A / size + (r < (ROWS\_A % size) ? 1 : 0);

recvcounts[r] = lr \* COLS\_B;

displs[r] = total;

total += recvcounts[r];

}

}

int my\_count = local\_rows \* COLS\_B;

MPI\_Gatherv(C\_local, my\_count, MPI\_DOUBLE,

C\_full, recvcounts, displs, MPI\_DOUBLE,

0, MPI\_COMM\_WORLD);

printf("rank = %d: local\_rows = %d, time = %.6f s\n", rank, local\_rows, elapsed);

if (rank == 0) {

printf("\nC = A \* B (first 5x5 elements):\n");

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

printf("%8.1f ", C\_full[i \* COLS\_B + j]);

}

printf("\n");

}

free(recvcounts);

free(displs);

free(C\_full);

}

free(A\_local);

free(B);

free(C\_local);

MPI\_Comm\_free(&ring\_comm);

MPI\_Finalize();

return 0;

}

**Задание 4**

Изображение выглядит как текст, линия, График, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 5** – Сравнение топологий для умножения матрицы на вектор

**Листинг программы топологии «кольцо»**

#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 (argc != 2) {

if (rank == 0) fprintf(stderr, "Usage: %s <N>\n", argv[0]);

MPI\_Finalize();

return 1;

}

int N = atoi(argv[1]);

if (N % size != 0) {

if (rank == 0) fprintf(stderr, "N must be divisible by number of processes!\n");

MPI\_Finalize();

return 1;

}

const int local\_rows = N / size;

const int block = N / size;

double \*A\_local = (double\*)malloc(local\_rows \* N \* sizeof(double));

double \*B\_block = (double\*)malloc(block \* 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] = 3.0;

for (int j = 0; j < block; j++)

B\_block[j] = 2.0;

int dims[1] = {0};

int periods[1] = {1};

MPI\_Dims\_create(size, 1, dims);

MPI\_Comm ring;

MPI\_Cart\_create(MPI\_COMM\_WORLD, 1, dims, periods, 0, &ring);

int src, dst;

MPI\_Cart\_shift(ring, 0, 1, &src, &dst);

MPI\_Barrier(MPI\_COMM\_WORLD);

double t1 = MPI\_Wtime();

for (int step = 0; step < size; step++) {

int col\_offset = ((rank - step + size) % size) \* block;

for (int i = 0; i < local\_rows; i++) {

for (int j = 0; j < block; j++) {

C\_local[i] += A\_local[i \* N + col\_offset + j] \* B\_block[j];

}

}

if (size > 1) {

MPI\_Sendrecv\_replace(B\_block, block, MPI\_DOUBLE, dst, 0, src, 0, ring, MPI\_STATUS\_IGNORE);

}

}

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("ring %d %.6f\n", N, max\_time);

}

free(A\_local);

free(B\_block);

free(C\_local);

MPI\_Comm\_free(&ring);

MPI\_Finalize();

return 0;

}

**Листинг программы топологии «полный граф»**

#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 (argc != 2) {

if (rank == 0) fprintf(stderr, "Usage: %s <N>\n", argv[0]);

MPI\_Finalize();

return 1;

}

int N = atoi(argv[1]);

if (N % size != 0) {

if (rank == 0) fprintf(stderr, "N must be divisible by number of processes!\n");

MPI\_Finalize();

return 1;

}

const int local\_rows = 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;

int \*index = (int\*)malloc(size \* sizeof(int));

int \*edges = (int\*)malloc(size \* (size - 1) \* sizeof(int));

int pos = 0;

for (int i = 0; i < size; i++) {

index[i] = size - 1;

for (int j = 0; j < size; j++) {

if (i != j) edges[pos++] = j;

}

}

MPI\_Comm graph\_comm;

MPI\_Graph\_create(MPI\_COMM\_WORLD, size, index, edges, 0, &graph\_comm);

MPI\_Comm\_rank(graph\_comm, &rank);

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];

}

}

MPI\_Allgather(C\_local, local\_rows, MPI\_DOUBLE, B, local\_rows, MPI\_DOUBLE, graph\_comm);

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("graph %d %.6f\n", N, max\_time);

}

free(index);

free(edges);

free(A\_local);

free(B);

free(C\_local);

MPI\_Comm\_free(&graph\_comm);

MPI\_Finalize();

return 0;

}

**Вывод**: в ходе выполнения лабораторной работы были сформированы практические навыки программирования параллельных процессов, исполняющихся на декартовой топологии связей и топологии "граф".