**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра «Систем автоматизированного проектирования (САПР)»**

отчет

**по лабораторной работе №1**

**по дисциплине «Архитектура параллельных вычислительных систем»**

Тема: «ОПЕРАЦИИ НАД ЭЛЕМЕНТАМИ ВЕКТОРОВ И МАТРИЦ НА СИСТЕМАХ С ОБЩЕЙ ПАМЯТЬЮ»

|  |  |  |
| --- | --- | --- |
| Студенты гр. 0302 |  | Бикеев М.Р. |
|  |  | Штейнберг Э.Э. |
|  |  | Камоликов В.А. |
|  |  | Билоблоцкий В.В. |
| Преподаватель |  | Костичев С.В. |

Санкт-Петербург

2024

**Задание на лабораторную работу.**

Вариант 4 – Умножение матрицы на матрицу.

Библиотека – MPI.

**Программное и аппаратное окружение при выполнении работы.**

Программное обеспечение:

* Windows 11 Pro 23H2 22631.4169
* Visual Studio 2022 (v143)
* Microsoft MPI v10.1.12498.52

Аппаратное окружение:

* Процессор с тактовой частотой 3.30 GHz (4.2 GHz)
* Количество ядер – 6, потоков – 12.

**Описание метода.**

Инициализация MPI:

* MPI\_Init инициализирует среду MPI, определяя количество процессов и их идентификаторы (rank).

Разделение работы между процессами:

* Процесс с rank 0 (главный) инициализирует матрицы A и B случайными числами.
* Матрица B рассылается всем процессам с помощью MPI\_Bcast.
* Матрица A разбивается на части, каждая часть (несколько строк) отправляется процессам с помощью MPI\_Scatter.

Параллельное умножение:

* Каждый процесс умножает свою часть матрицы A на общую матрицу B и записывает результат в локальную часть матрицы C.

Завершение:

* Локальные результаты умножения (части матрицы C) собираются в главный процесс с помощью MPI\_Gather, выводится время выполнения MPI\_Wtime, и программа завершает работу, вызывая MPI\_Finalize.

**Блок схема алгоритмов.**

На рисунке 1 показана блок схема выполнения приложения.

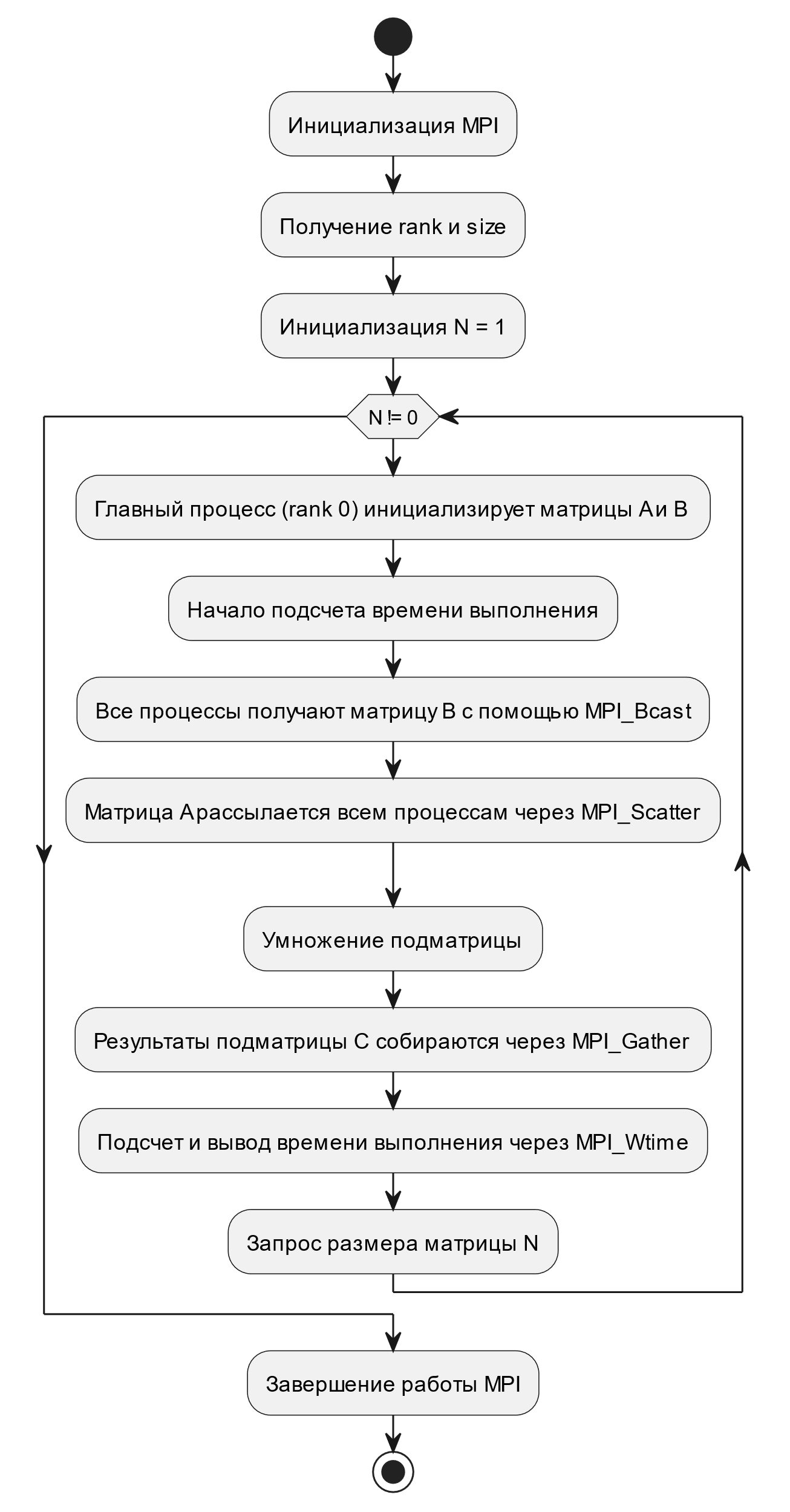
****

Рисунок 1 – Блок схема алгоритма программы

**Пример, подтверждающий работу программы.**

Для проверки работы программы решим задачу с матрицами A и B с использованием онлайн калькулятора и сравним с решением программы.

|  |  |
| --- | --- |
| Исходные матрицы: | |
|  | |
| Решения: | |
| Онлайн калькулятором ru.onlinemschool.com | Программы |
|  |  |

По итогу решение совпадают, что означает - программа правильно решает задачу.

**Тексты программы.**

Для данной лабораторной работы была разработана консольное приложение включающая в себя 1 файл программного кода.

Файл 1 - *1labOperationOnMatrix.cpp*

#include <mpi.h>

#include <iostream>

#include <vector>

#include <cstdlib>

void multiplyMatrices(const std::vector<int>& A, const std::vector<int>& B, std::vector<int>& C, int N, int rowsPerProcess) {

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

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

C[i \* N + j] = 0;

for (int k = 0; k < N; ++k) {

C[i \* N + j] += A[i \* N + k] \* B[k \* N + j];

}

}

}

}

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

int N = 1; // Размер матриц NxN

while (N != 0) {

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

std::vector<int> A(N \* N), B(N \* N), C(N \* N);

int rowsPerProcess = N / size; // количество строк на процесс

// Инициализация матриц в процессе 0

if (rank == 0) {

for (int i = 0; i < N \* N; ++i) {

A[i] = rand() % 10; // Заполнение случайными значениями

B[i] = rand() % 10;

}

}

// Начало замера общего времени с помощью MPI\_Wtime

double start\_time = MPI\_Wtime();

// Передаем всем процессам матрицу B

MPI\_Bcast(B.data(), N \* N, MPI\_INT, 0, MPI\_COMM\_WORLD);

// Выделяем память для подматрицы A и результата подматрицы C

std::vector<int> local\_A(rowsPerProcess \* N);

std::vector<int> local\_C(rowsPerProcess \* N);

// Рассылаем части матрицы A

MPI\_Scatter(A.data(), rowsPerProcess \* N, MPI\_INT, local\_A.data(), rowsPerProcess \* N, MPI\_INT, 0, MPI\_COMM\_WORLD);

// Умножаем подматрицу

multiplyMatrices(local\_A, B, local\_C, N, rowsPerProcess);

// Собираем результаты

MPI\_Gather(local\_C.data(), rowsPerProcess \* N, MPI\_INT, C.data(), rowsPerProcess \* N, MPI\_INT, 0, MPI\_COMM\_WORLD);

// Завершаем замер времени

double end\_time = MPI\_Wtime();

// Выводим общее время выполнения только на процесс 0

if (rank == 0) {

std::cout << "Total time taken: " << (end\_time - start\_time) << " seconds\n";

}

// Получаем новый размер матрицы от пользователя

if (rank == 0) {

std::cout << "\nEnter size matrix (0 to exit): ";

std::cin >> N;

}

// Передаем размер матрицы всем процессам

MPI\_Bcast(&N, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

return 0;

}

**Пример запуска программы.**

Для запуска приложения с различным количеством использованных процессов используется команда в терминале Windows: mpiexec -n COUNT\_PROCESS 1labOperationOnMatrix.exe. После ввода команды пользователь может вводить размер матрицы и получать время выполнения умножения как показано на рисунке 2.

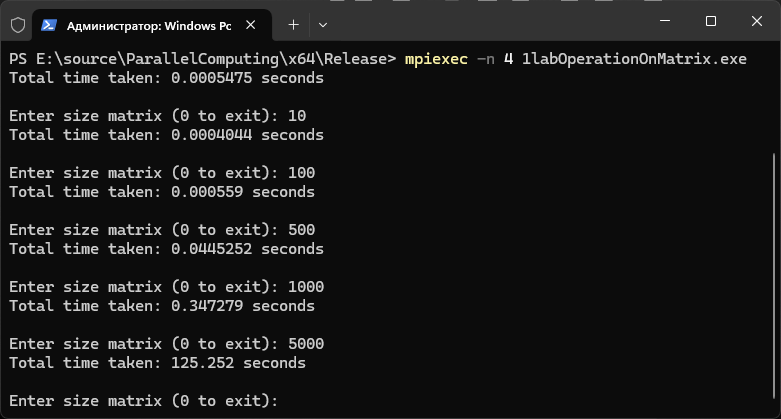
****

Рисунок 2 – Запуск и получение результатов на 4 процессах

**Сравнительная оценка эффективности программы.**

Рисунок 3 показывает результаты умножения матриц NxN размерности с разным количеством процессов. Программа становится эффективнее при увеличении числа процессов, особенно на больших матрицах, но эффективность падает на малых размерах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер матриц | 1 процесс | 2 процесса | 4 процесса | 12 процессов |
| 1 | 0,000019 | 0,000212 | 0,0005475 | 0,0005412 |
| 10 | 0,000068 | 0,000283 | 0,0004044 | 0,0000047 |
| 100 | 0,000673 | 0,000348 | 0,0005591 | 0,0008845 |
| 500 | 0,089245 | 0,075094 | 0,0445252 | 0,0391655 |
| 1000 | 0,850396 | 0,50668 | 0,347279 | 0,292338 |
| 5000 | 319,024 | 182,849 | 125,252 | 94,5969 |
| 10000 | 3828,28 | 2194,18 | 1509,57 | 1183,3 |

Рисунок 3 – Отношение выполняемого времени в зависимости от числа процессов и размеров умножаемых матриц

Ключевые моменты:

* Малые матрицы (1x1 - 100x100): Параллелизация малоэффективна, время на 1 процессе почти такое же, как на 4 и 12.
* Средние матрицы (500x500 - 1000x1000): Выгода от параллелизации становится заметной. Для 1000x1000 скорость на 4 процессах почти в 2.5 раза выше, чем на 1 процессе.
* Большие матрицы (5000x5000 и выше): Параллелизация жизненно необходима, но увеличение числа процессов выше 4 даёт относительно небольшой прирост. Например, на 10000x10000 переход с 4 на 12 процессов ускоряет выполнение только в 1.3 раза.

**Выводы.**

Программа эффективно масштабируется на больших матрицах, но сталкивается с ограниченной масштабируемостью из-за синхронизации процессов.