МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего порядка

«Чувашский государственный университет имени И. Н. Ульянова»

Факультет информатики и вычислительной техники

Кафедра вычислительной техники

Дисциплина Параллельное программирование

**Лабораторная работа №4.**

**Параллельное программирование**

**Выполнил:** студ.ИВТ-42-18

Жижайкин К.В.

**Проверил:** доцент

Ковалев С. В.

Чебоксары, 2021 г.

22)

**Задание:**

Коммуникации «точка-точка»: схема «каждый каждому» Напишите MPI-программу, реализующую при помощи блокирующих функций посылки сообщений типа точка-точка схему коммуникации процессов «каждый каждому», в которой осуществляется пересылка сообщения от каждого процесса каждому. В качестве передаваемого сообщения используйте номер процесса. Каждый процесс должен вывести на экран все полученные сообщения.

**Код:**

for (auto dest = 0; dest < size; dest++)

{

buffer = rank;

if (rank != dest)

{

MPI\_Isend(&buffer, 1, MPI\_INT, dest, 0, MPI\_COMM\_WORLD, &request);

}

}

for (auto source = 0; source < size; source++)

{

if (rank != source)

{

MPI\_Irecv(&buffer, 1, MPI\_INT, source, 0, MPI\_COMM\_WORLD, &request);

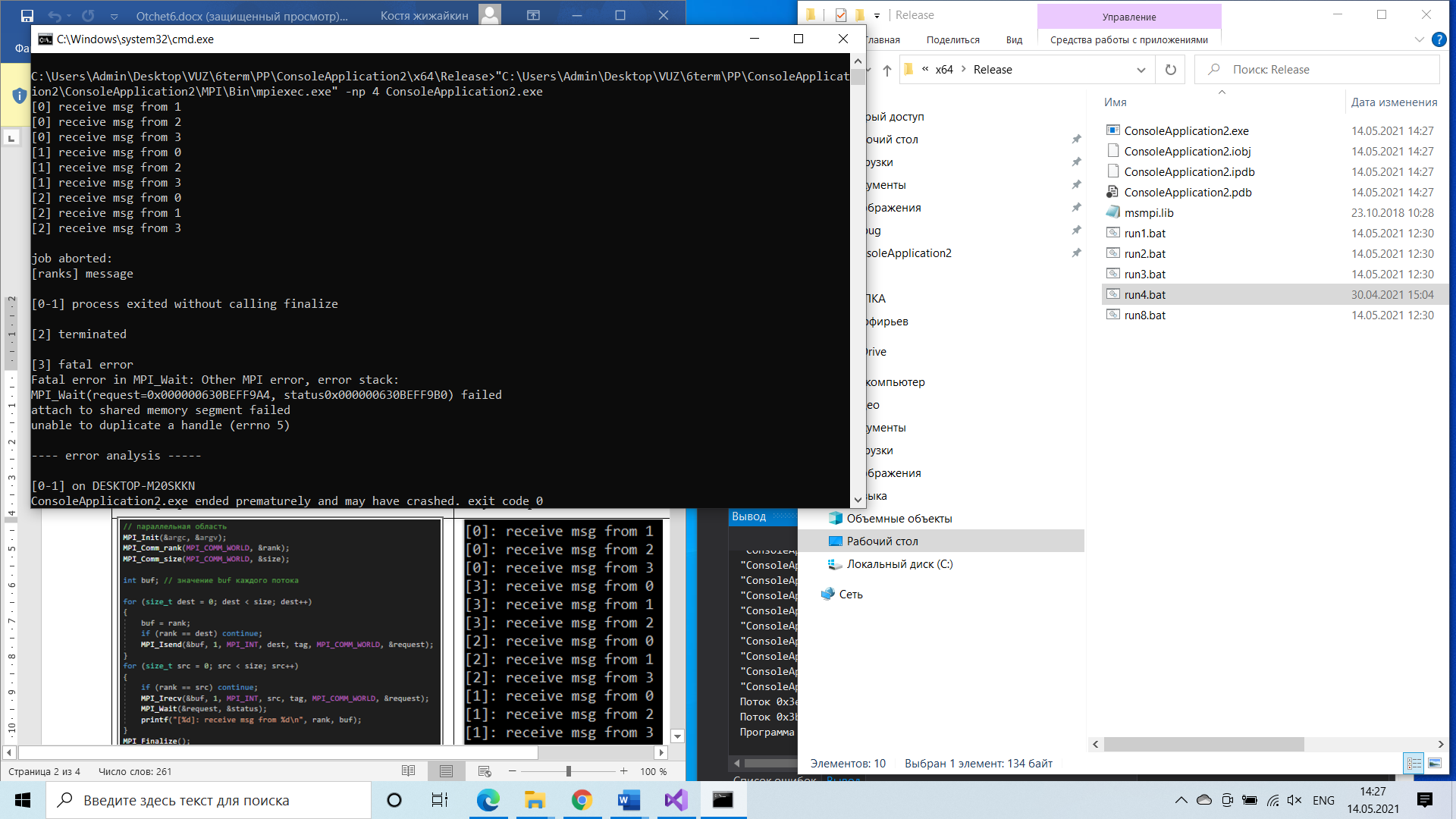
MPI\_Wait(&request, &status);

std::cout << "[" << rank << "] receive msg from " << buffer << "\n";

}

}

**Результат:**



23)

**Задание:**

Коллективные коммуникации: широковещательная рассылка данных 1. Изучите MPI-функцию широковещательной рассылки данных MPI\_Bcast. Напишите MPI-программу, которая в строке длины n определяет количество вхождений символов. Ввод данных должен осуществляться процессом с номером 0. Для рассылки строки поиска и ее длины по процессам используйте функцию MPI\_Bcast.

**Код:**

MPI\_Init(&argc, &argv);

{

MPI\_Status status;

MPI\_Request request;

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

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

if (rank == 0)

{

std::cout << "Введите длину строки(до 50)\n";

std::cin >> leng;

std::cout << "\n";

for (auto i = 0; i < leng; i++)

{

std::cin >> buffer[i];

}

}

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

MPI\_Bcast(&buffer, leng, MPI\_CHAR, 0, MPI\_COMM\_WORLD);

int count = 0;

for (int i = rank; i < 26; i = i + size)

{

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

{

if (buffer[j] == (char)((int)start + i))

{

count++;

}

}

std::cout << "[" << (char)(start+i) << "] counts " << count << "\n";

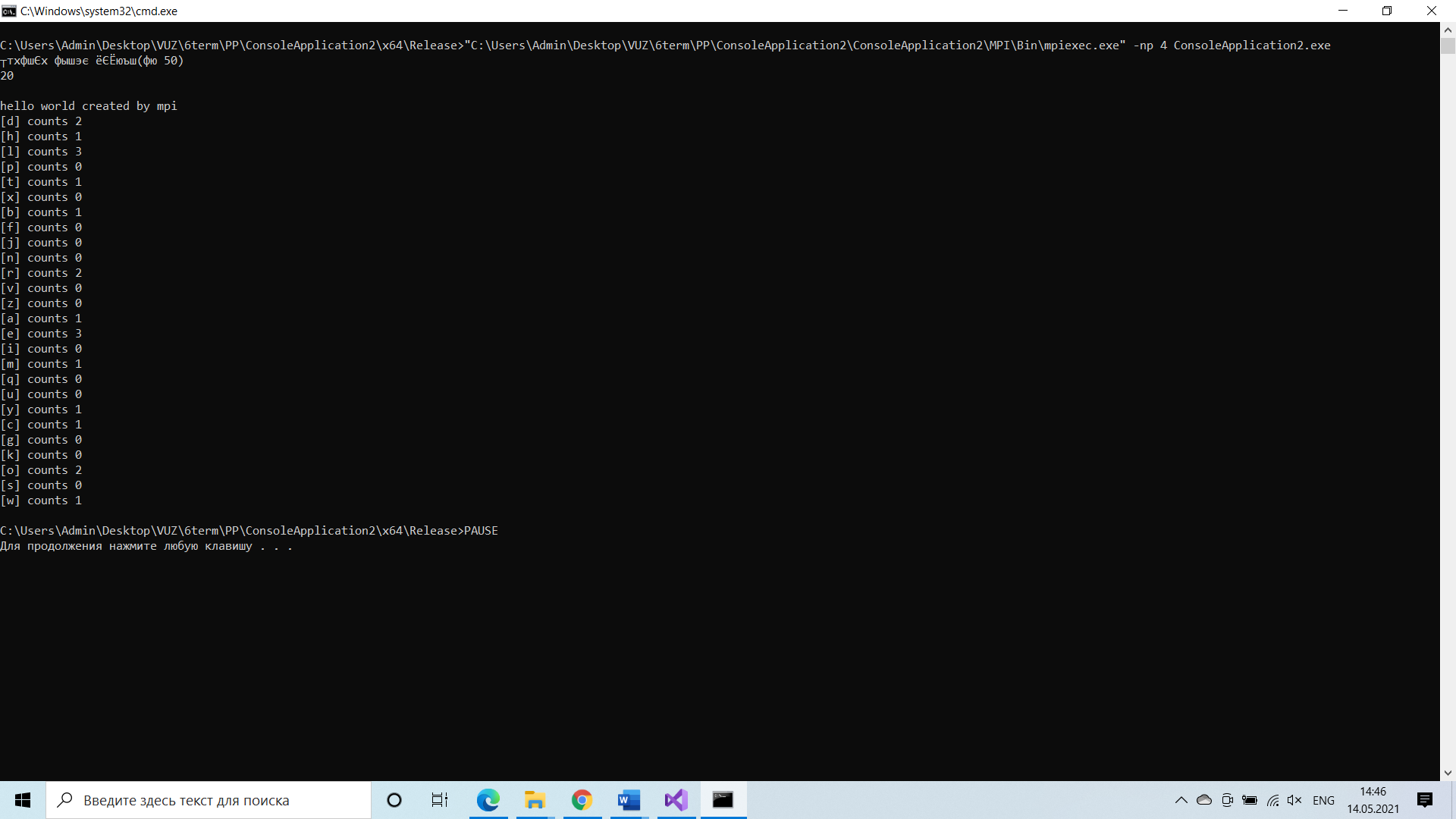
count = 0;

}

}

MPI\_Finalize();

**Результат:**



24)

**Задание:**

Коллективные коммуникации: операции редукции 1. Изучите MPI-функцию для выполнения операций редукции над данными, расположенными в адресных пространствах различных процессов, MPI\_Reduce. Реализуйте программу вычисления числа 𝜋, используйте функцию MPI\_Reduce для суммирования результатов, вычисленных каждым процессом

**Код:**

MPI\_Init(&argc, &argv);

{

MPI\_Status status;

MPI\_Request request;

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

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

for (auto i = rank; i < eps; i += size)

{

x = (i + 0.5) / eps;

sum += (4 / (1 + x \* x)) / eps;

}

float FinallySum = 0;

MPI\_Reduce(&sum, &FinallySum, 1, MPI\_FLOAT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0)

{

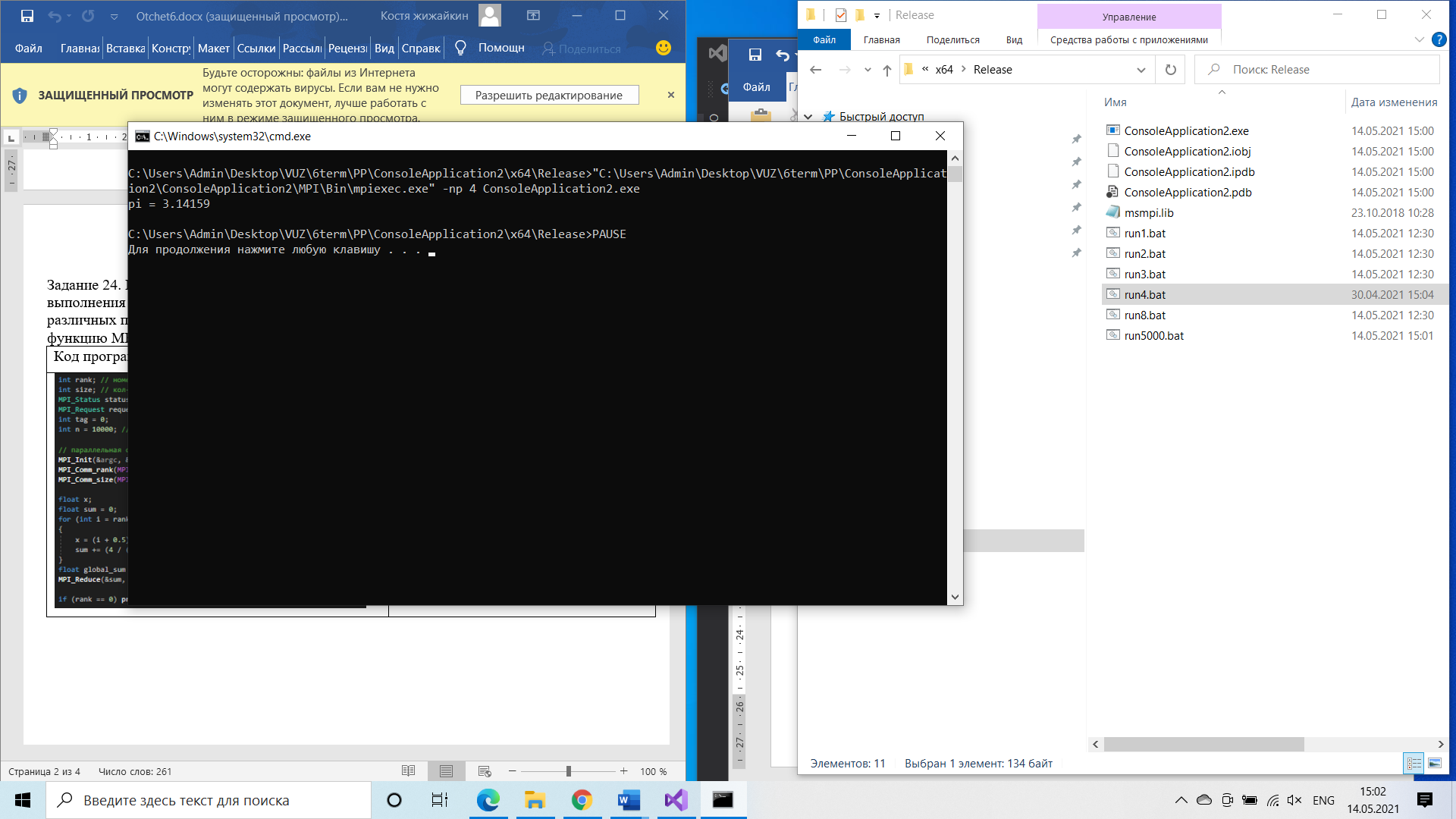
std::cout << "pi = " << FinallySum << "\n";

}

}

MPI\_Finalize();

**Результат:**



25)

**Задание:**

Коллективные коммуникации: функции распределения и сбора данных 1. Изучите MPI-функции распределения и сбора блоков данных по процессам MPI\_Scatter и MPI\_Gather. Напишите программу, которая вычисляет произведение двух квадратных матриц 𝐴 × 𝐵 = С размера 𝑛 × 𝑛. Используйте формулу, приведенную в задании 9. Ввод данных и вывод результата должны осуществляться процессом с номером 0. Для распределения матриц A и B и сбора матрицы С используйте функций MPI\_Scatter и MPI\_Gather

**Код:**

int rank; // номер процесса в приложении

int size; // кол-во процессов в приложении

int root = 0; // мастер поток

int tag = 0;

// матрицы

int\*\* matrixA = NULL; int\*\* matrixB = NULL; int\* arrA = NULL; int\* arrC = NULL;

// параллельная область   
MPI\_Init(&argc, &argv);

MPI\_Con«\_rank(MPI\_COMM\_WORLD, &rank);

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

OutMatrix(matrixB, n, rank);

// рассылка n всем потокам

MPI\_Bcast(&n, 1, MPI\_INT, root, MPI\_COMM\_WORLD);

// разложение строк по потокам int\* strA = new int[n];

MPI\_Scatter(arrA, n, MPI\_INT, strA, n, MPI\_INT, root, MPI\_COMM\_WO«LD) MPI\_Barrier(MPI\_COMM\_WORLD);

for (int i = 0; i < n; i++)

{

MPI\_Bcast(\*(matrixB + i), n, MPI\_INT, root, MPI\_COMM\_WO«LD);

}

// вычисление строки матрицы С int\* strC = new int[n] {0};

for (int i = 0; i < n; i++)

{

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

{

srrC[i]+=strA[j]\*matrix[j][i];

}

MPI\_Gather(strC, n, MPI\_INT, arrC, n, MPI\_INT, root, MPI\_COMM\_MORLO);

// вывод матрицы С if (rank == root) {

printf(“\nMatrixC: \n");

**Результат:**

