# **Практическая работа 6**

**Параллельное программирование с использованием расширенных средств технологии OpenMP (редукции, атомарных операций, критических секций, замков).**

# **Цель работы**

* Отработка принципов разработки интерфейса передачи сообщений для распределенных приложений;
* Реализация распределенной программы с использованием расширенных средств технологии MPI;
* Сравнительный анализ эффективности распределенных и параллельных программ

# **Постановка задачи**

Вариант 6.13

Составить программу последовательного и распределенного вариантов умножения двух матриц A[n,n] и В[n,n], используя блочный алгоритм Фокса. Провести тестирование программы в последовательном и распределенном исполнениях на матрицах размером n = 3, сформированных вводом с клавиатуры. Разработать форму представления результирующей матрицы на экране монитора.

Рабочие матрицы А и В сформировать, используя генератор псевдослучайных чисел. Провести контрольные прогоны программы для размеров n = 600, 700, 800, 900, 1000 в последовательном и распределенном исполнениях с количеством процессов p = 2, 4, 8, 16, 32. Полученные результаты свести в сводную таблицу. Построить графики изменения ускорения умножения в зависимости от размеров матриц. Построить графики изменения ускорения распределенных вычислений с разным количеством процессов. Вычислить показатели эффективности и стоимости распределенной реализации программы. Провести анализ полученных результатов. Сделать выводы по проделанной работе, основанные на полученных данных.

# **Теоретическое введение**

При построении параллельных способов выполнения матричного умножения наряду с рассмотрением матриц в виде наборов строк и столбцов широко используется блочное представление матриц.

При блочном разбиении данных для определения базовых подзадач естественным представляется взять за основу вычисления, выполняемые над матричными блоками. С учетом сказанного определим базовую подзадачу как процедуру вычисления всех элементов одного из блоков матрицы С.

В соответствии с алгоритмом Фокса в ходе вычислений на каждой базовой подзадаче (i,j) располагается четыре матричных блока:

* блок Cij матрицы C, вычисляемый подзадачей;
* блок Aij матрицы A, размещаемый в подзадаче перед началом вычислений;
* блоки A'ij, B'ij матриц A и B, получаемые подзадачей в ходе выполнения вычислений.

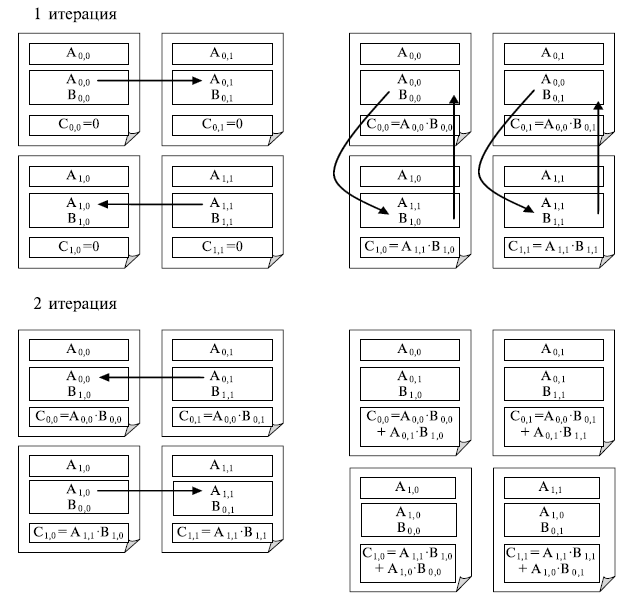
Выполнение параллельного метода включает:

* этап инициализации, на котором каждой подзадаче (i,j) передаются блоки Aij, Bij и обнуляются блоки Cij на всех подзадачах;
* этап вычислений, в рамках которого на каждой итерации l, 0<=l<q, осуществляются следующие операции:

для каждой строки i, 0<=i<q, блок Aij подзадачи (i,j) пересылается на все подзадачи той же строки i решетки; индекс j, определяющий положение подзадачи в строке, вычисляется в соответствии с выражением

полученные в результаты пересылок блоки A'ij, B'ij каждой подзадачи (i,j) перемножаются и прибавляются к блоку Cij

блоки B'ij каждой подзадачи (i,j) пересылаются подзадачам, являющимся соседями сверху в столбцах решетки подзадач (блоки подзадач из первой строки



*Рисунок.1. Наглядное представление работы блочного перемножения двух матриц при помощи алгоритма Фокса*

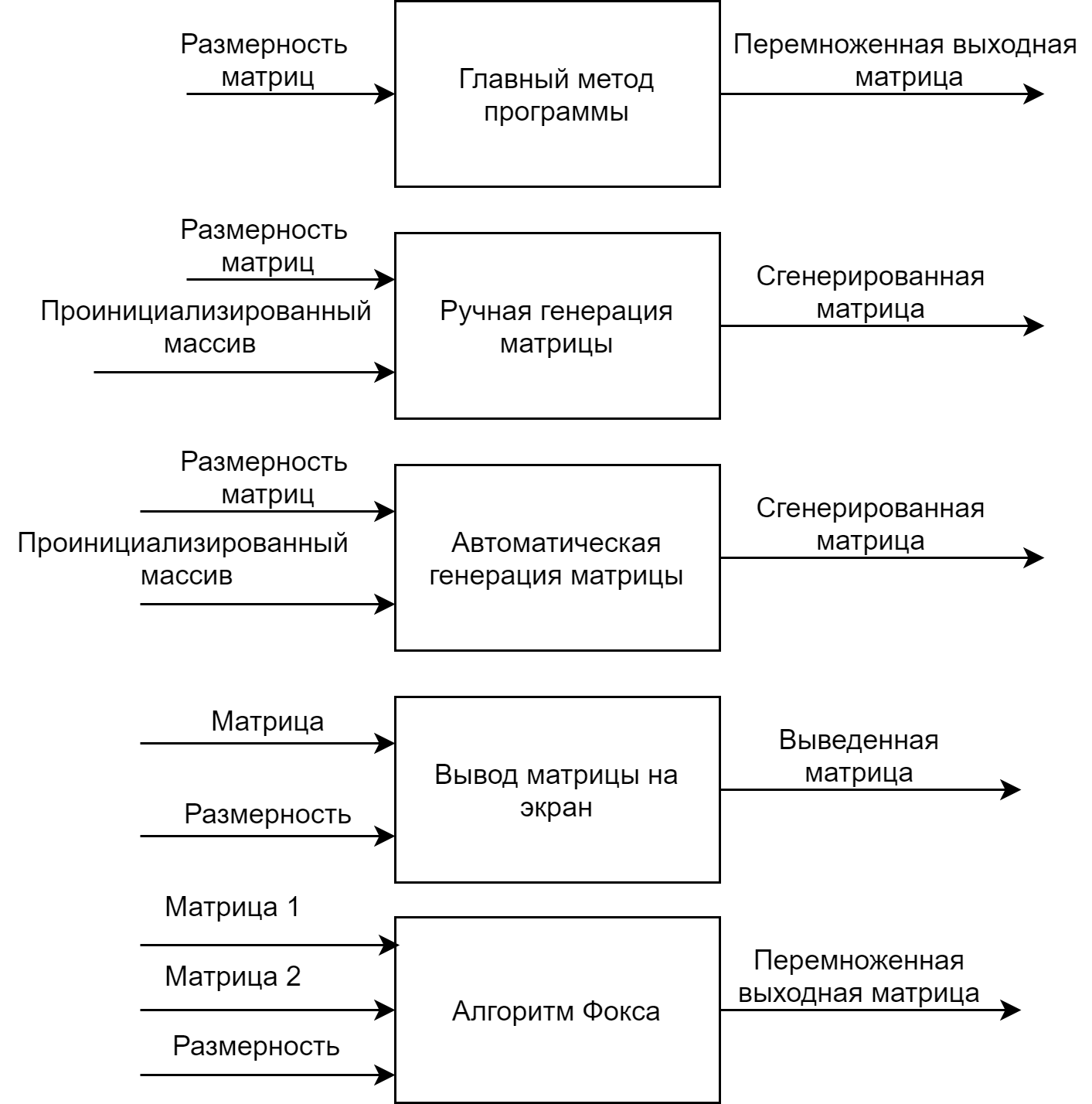
Ускорение параллельного алгоритма является его наиболее информативной характеристикой, которая показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение параллельного алгоритма определяется величиной SN=T1/TN, где T1 – время выполнения алгоритма на одном процессоре, TN – время выполнения алгоритма на N процессорах.

Эффективность (efficiency) использования параллельным алгоритмом процессоров при решении задачи определяется соотношением E = (n \* p) / (n + 2 \* p^2) (величина эффективности определяет среднюю долю времени выполнения алгоритма, в течение которой процессоры реально задействованы для решения задачи). Она вычисляется как Sn/nmax, где nmax – количество процессоров в системе.

Стоимость (cost) вычислений – метод, стоимость которого является пропорциональной времени выполнения наилучшего последовательного алгоритма. Вычисляется по формуле Cp = p\*Tp (число процессоров \* время выполнения задачи)

# **Описание алгоритмов, используемых для решения задачи**

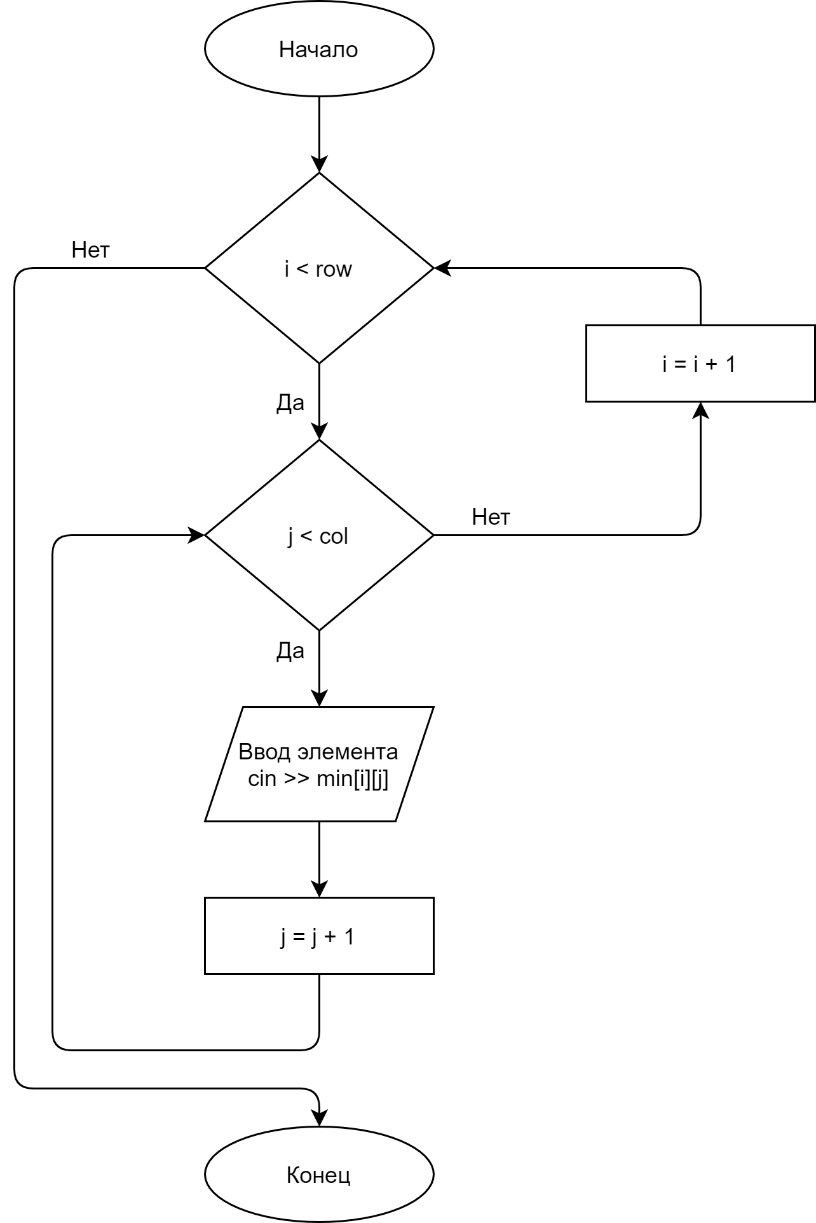
Для выполнения данной практической работы было решено провести декомпозицию общего процесса на более мелкие составляющие (рисунок 2).



*Рисунок. 2. Декомпозиция общей задачи на подзадачи*

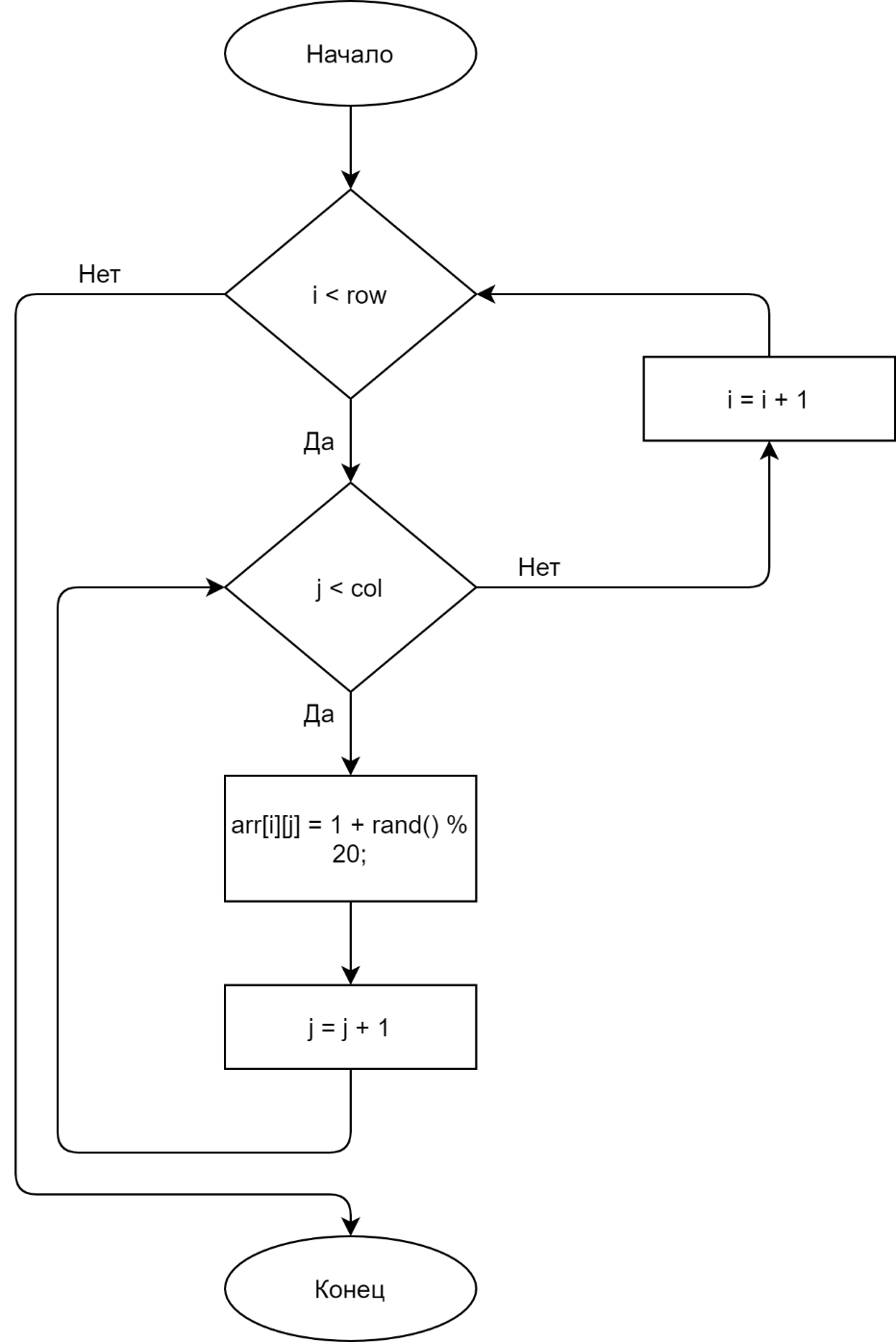
После проведения декомпозиции опишем использование всех алгоритмов, используемых в решении общей задачи.

Во-первых, ниже продемонстрирован рисунок алгоритма ручного заполнения матрицы (рисунок 3).



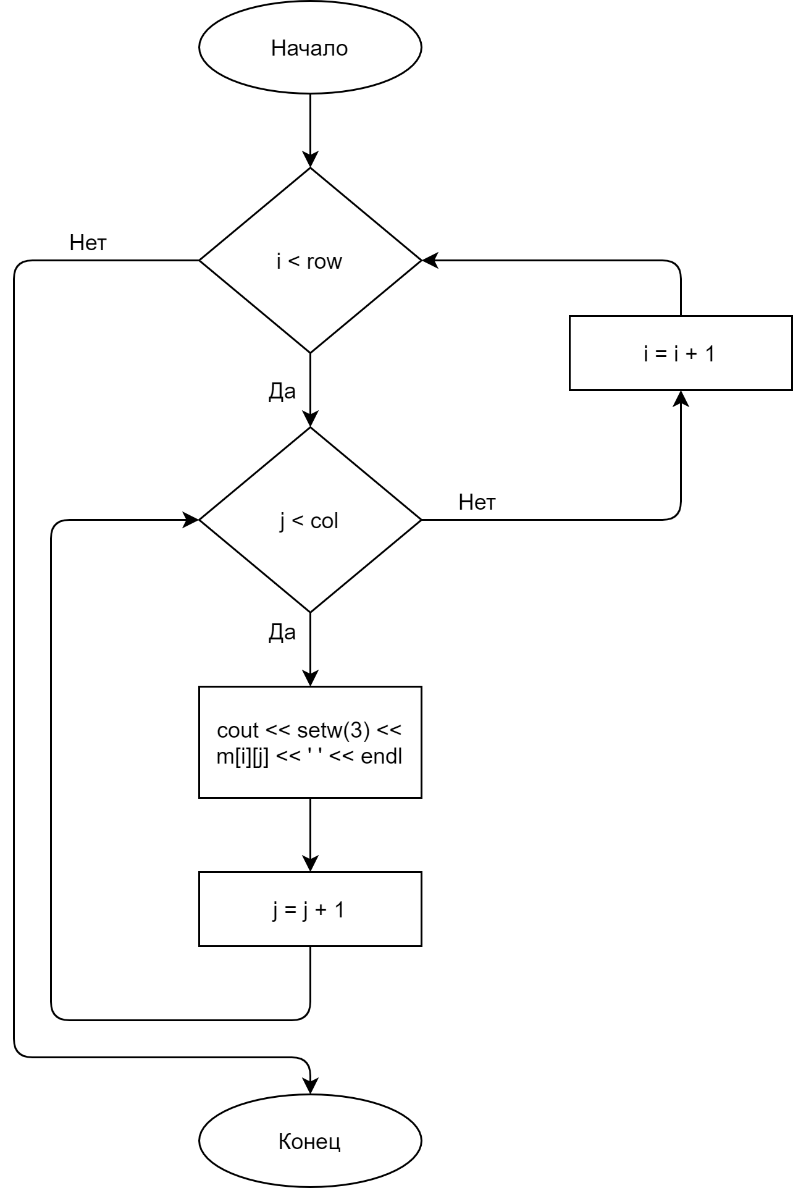
*Рисунок. 3. Схема алгоритма ручного заполнения матрицы*

Во-вторых, ниже продемонстрирован рисунок алгоритма автоматического заполнения матрицы (рисунок 4).



*Рисунок. 4. Схема алгоритма автоматического заполнения матрицы*

Во-третьих, ниже продемонстрирован рисунок алгоритма вывода матрицы на экран пользователя (рисунок 5).



*Рисунок. 5. Схема алгоритма вывода матрицы на экран*

Используется блочная схема разбиения матриц. При таком способе разделения данных исходные матрицы А, В и результирующая матрица С представляются в виде наборов блоков. Далее предполагается что все матрицы являются квадратными размера n×n, количество блоков по горизонтали и вертикали одинаково и равно q (т.е. размер всех блоков равен k×k, k=n/q).

В соответствии с алгоритмом Фокса в ходе вычислений на каждой базовой подзадаче (i,j) располагается четыре матричных блока:

* блок Cij матрицы C, вычисляемый подзадачей;
* блок Aij матрицы A, размещаемый в подзадаче перед началом вычислений;
* блоки A'ij, B'ij матриц A и B, получаемые подзадачей в ходе выполнения вычислений.

Выполнение параллельного метода включает:

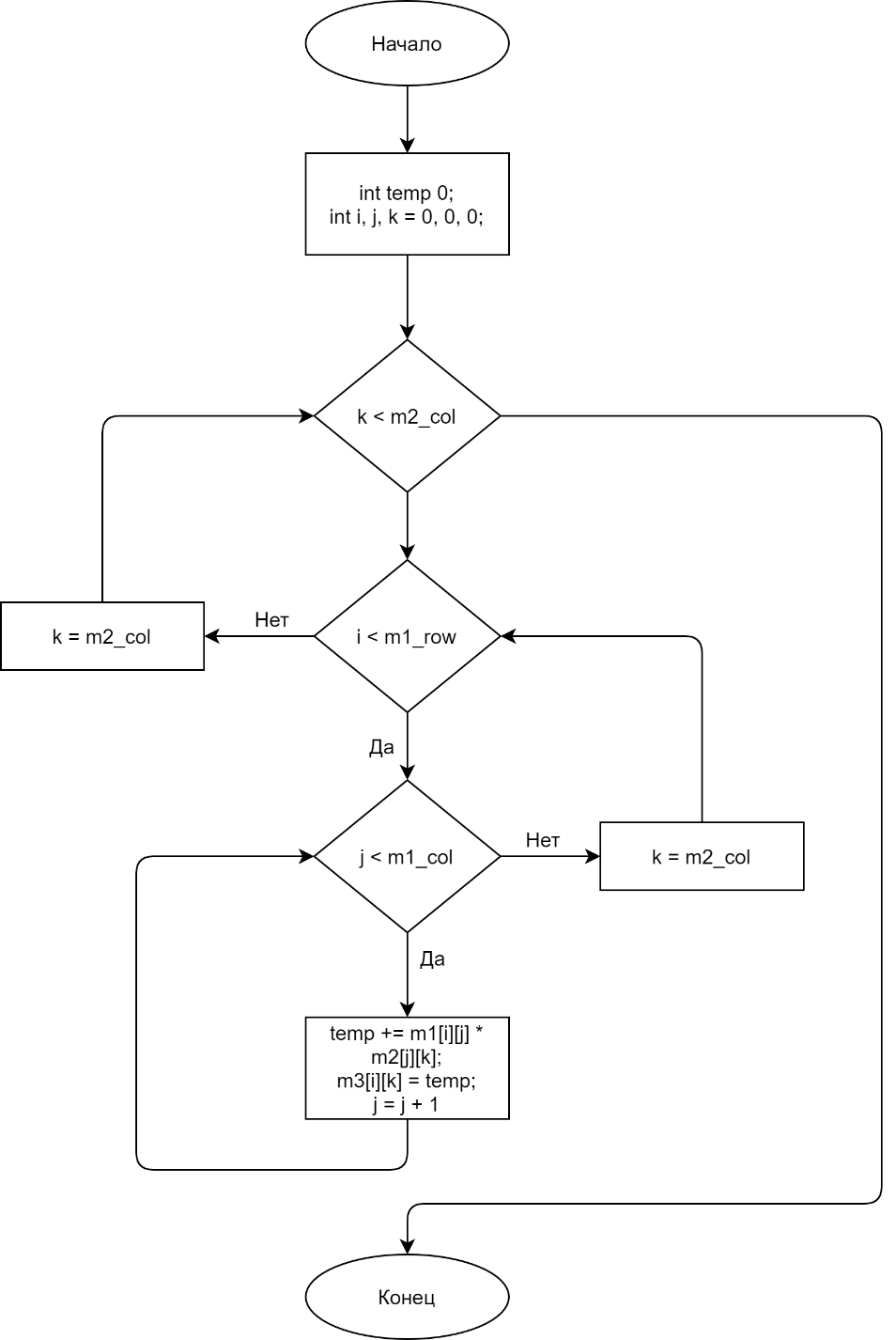
* этап инициализации, на котором каждой подзадаче (i,j) передаются блоки Aij, Bij и обнуляются блоки Cij на всех подзадачах;
* этап вычислений, в рамках которого на каждой итерации l, 0<=l<q, осуществляются следующие операции:

для каждой строки i, 0<=i<q, блок Aij подзадачи (i,j) пересылается на все подзадачи той же строки i решетки; индекс j, определяющий положение подзадачи в строке, вычисляется в соответствии с выражением

полученные в результаты пересылок блоки A'ij, B'ij каждой подзадачи (i,j) перемножаются и прибавляются к блоку Cij

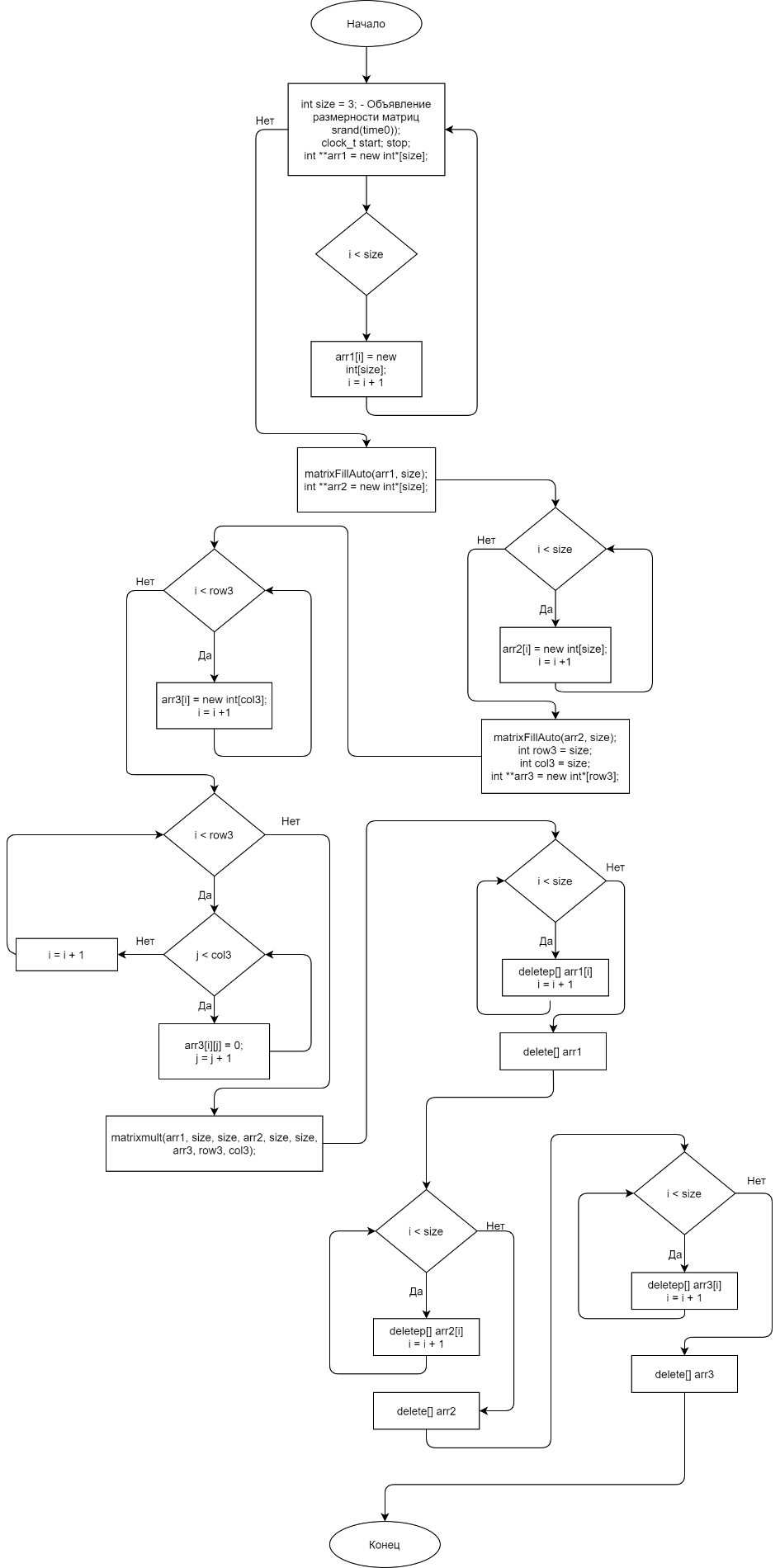
блоки B'ij каждой подзадачи (i,j) пересылаются подзадачам, являющимся соседями сверху в столбцах решетки подзадач (блоки подзадач из первой строки решетки пересылаются подзадачам последней строки решетки).

Ниже продемонстрирован рисунок алгоритма Фокса (рисунок 6).



*Рисунок. 6. Схема алгоритма Фокса перемножения двух матрицы*

Во-пятых, ниже продемонстрирован рисунок алгоритма выполнения главной функции программы main () (рисунок 7).



*Рисунок. 7. Схема главной функции main()*

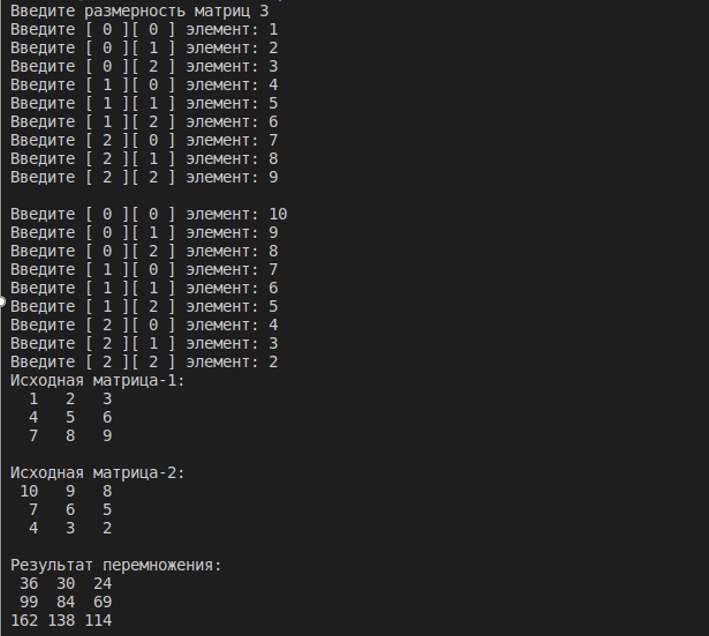
# **Текст исходного кода программы**

|  |
| --- |
| #include<stdlib.h>  #include <ctime>  #include<math.h>  #include"mpi.h"  #include <iostream>  using namespace std;  #define MATRIX\_SIZE 1000  int first\_matrix[MATRIX\_SIZE][MATRIX\_SIZE];  int second\_matrix[MATRIX\_SIZE][MATRIX\_SIZE];  typedef struct {  int proc\_count;  int dim;  int row;  int col;  int rank;  MPI\_Comm grid\_comm;  MPI\_Comm row\_comm;  MPI\_Comm col\_comm;  } GridStructure;  void GridSetup(GridStructure\* grid) {  int dimensions[2];  int wrap\_around[2];  int coordinates[2];  int free\_coords[2];  int world\_rank;  MPI\_Comm\_size(MPI\_COMM\_WORLD, &(grid->proc\_count));  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);  grid->dim = (int)sqrt((double)grid->proc\_count);  dimensions[0] = dimensions[1] = grid->dim;  wrap\_around[0] = 0;  wrap\_around[1] = 1;  MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, dimensions, wrap\_around, 1, &(grid->grid\_comm));  MPI\_Comm\_rank(grid->grid\_comm, &(grid->rank));  MPI\_Cart\_coords(grid->grid\_comm, grid->rank, 2, coordinates);  grid->row = coordinates[0];  grid->col = coordinates[1];  free\_coords[0] = 0;  free\_coords[1] = 1;  MPI\_Cart\_sub(grid->grid\_comm, free\_coords, &(grid->row\_comm));  free\_coords[0] = 1;  free\_coords[1] = 0;  MPI\_Cart\_sub(grid->grid\_comm, free\_coords, &(grid->col\_comm));  }  void MultiplyLocal(int\*\* a, int\*\* b, int\*\* c, int size) {  int temp = 0;  for (int i = 0; i < size; i++)  for (int j = 0; j < size; j++) {  temp = 0;  for (int k = 0; k < size; k++)  temp += (a[i][k] \* b[k][j]);  c[i][j] += temp;  }  }  void UnpackMatrix(int\* buff, int\*\* a, int size) {  int k = 0;  for (int i = 0; i < size; i++)  for (int j = 0; j < size; j++) {  a[i][j] = buff[k];  k++;  }  }  void PackMatrix(int\* buff, int\*\* a, int size) {  int k = 0;  for (int i = 0; i < size; i++)  for (int j = 0; j < size; j++) {  buff[k] = a[i][j];  k++;  }  }  void PrintMatrix(int\*\* matrix, int size) {  for (int i = 0; i < size; i++) {  std::cout << "|";  for (int j = 0; j < size; j++) {  int el = matrix[i][j];  if (el < 10)  std::cout << " ";  std::cout << el;  std::cout << "|";  }  std::cout << std::endl;  }  }  void PrintPackedMatrix(int\* matrix, int size) {  for (int i = 0; i < size; i++) {  std::cout << "|";  for (int j = 0; j < size; j++) {  int el = matrix[i \* size + j];  if (el < 10)  std::cout << " ";  std::cout << el;  std::cout << "|";  }  std::cout << std::endl;  }  }  void GenerateMatrices() {  for (int i = 0; i < MATRIX\_SIZE; i++)  for (int j = 0; j < MATRIX\_SIZE; j++) {  first\_matrix[i][j] = rand() % 100;//(i==j) ? 1 : 0;  second\_matrix[i][j] = rand() % 100;  }  }  void FoxMultiply(int n, GridStructure\* grid, int\*\* a, int\*\* b, int\*\* c) {  int\*\* temp\_a, \* buff, stage, root, submat\_dim, src, dst;  MPI\_Status status;  submat\_dim = n / grid->dim;  temp\_a = new int\* [submat\_dim];  for (int i = 0; i < submat\_dim; ++i)  temp\_a[i] = new int[submat\_dim];  for (int i = 0; i < submat\_dim; i++)  for (int j = 0; j < submat\_dim; j++)  temp\_a[i][j] = 0;  buff = new int[submat\_dim \* submat\_dim];  for (int i = 0; i < submat\_dim \* submat\_dim; i++)  buff[i] = 0;  src = (grid->row + 1) % grid->dim;  dst = (grid->row + grid->dim - 1) % grid->dim;  for (stage = 0; stage < grid->dim; stage++) {  root = (grid->row + stage) % grid->dim;  if (root == grid->col) {  PackMatrix(buff, a, submat\_dim);  MPI\_Bcast(buff, submat\_dim \* submat\_dim, MPI\_INT, root, grid->row\_comm);  UnpackMatrix(buff, a, submat\_dim);  MultiplyLocal(a, b, c, submat\_dim);  }  else {  PackMatrix(buff, temp\_a, submat\_dim);  MPI\_Bcast(buff, submat\_dim \* submat\_dim, MPI\_INT, root, grid->row\_comm);  UnpackMatrix(buff, temp\_a, submat\_dim);  MultiplyLocal(temp\_a, b, c, submat\_dim);  }  PackMatrix(buff, b, submat\_dim);  MPI\_Sendrecv\_replace(buff, submat\_dim \* submat\_dim, MPI\_INT, dst, 0, src, 0, grid->col\_comm, &status);  UnpackMatrix(buff, b, submat\_dim);  }  }  int main(int argc, char\* argv[]) {  int block\_size;  int\*\* local\_a, \*\* local\_b, \*\* local\_c;  clock\_t start, finish;  MPI\_Init(&argc, &argv);  GridStructure grid;  GridSetup(&grid);  GenerateMatrices();  block\_size = MATRIX\_SIZE / grid.dim;  int base\_row = grid.row \* block\_size;  int base\_col = grid.col \* block\_size;  local\_a = new int\* [MATRIX\_SIZE];  local\_b = new int\* [MATRIX\_SIZE];  local\_c = new int\* [MATRIX\_SIZE];  for (int i = 0; i < MATRIX\_SIZE; ++i)  {  local\_a[i] = new int[MATRIX\_SIZE];  local\_b[i] = new int[MATRIX\_SIZE];  local\_c[i] = new int[MATRIX\_SIZE];  }  for (int i = base\_row; i < base\_row + block\_size; i++)  for (int j = base\_col; j < base\_col + block\_size; j++) {  local\_a[i - (base\_row)][j - (base\_col)] = first\_matrix[i][j];  local\_b[i - (base\_row)][j - (base\_col)] = second\_matrix[i][j];  local\_c[i - (base\_row)][j - (base\_col)] = 0;  }  if (grid.rank == 0)  {  std::cout << "Ready..." << std::endl;  }  MPI\_Barrier(grid.grid\_comm);  if (grid.rank == 0)  {  start = clock();  }  FoxMultiply(MATRIX\_SIZE, &grid, local\_a, local\_b, local\_c);  MPI\_Barrier(grid.grid\_comm);  if (grid.rank == 0)  {  finish = clock();  clock\_t result\_time = finish - start;  std::cout << "Time: " << double(result\_time) / 1000. << std::endl;  }  int\* result\_buff = new int[MATRIX\_SIZE \* MATRIX\_SIZE];  int\* local\_buff = new int[block\_size \* block\_size];  PackMatrix(local\_buff, local\_c, block\_size);  MPI\_Gather(local\_buff, block\_size \* block\_size, MPI\_INT, result\_buff, block\_size \* block\_size, MPI\_INT, 0, grid.grid\_comm);  MPI\_Barrier(grid.grid\_comm);  if (grid.rank == 0) {  int\* data = new int[MATRIX\_SIZE \* MATRIX\_SIZE];  int k = 0;  for (int bi = 0; bi < grid.dim; bi++)  for (int bj = 0; bj < grid.dim; bj++)  for (int i = bi \* block\_size; i < bi \* block\_size + block\_size; i++)  for (int j = bj \* block\_size; j < bj \* block\_size + block\_size; j++) {  data[i \* MATRIX\_SIZE + j] = result\_buff[k];  k++;  }  }  MPI\_Finalize();  exit(0);  } |
|  |

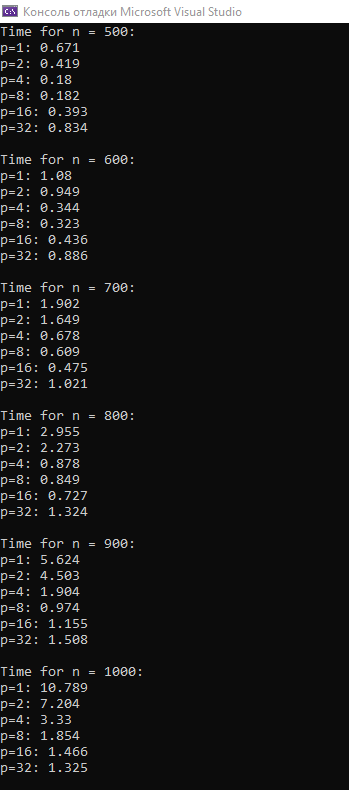
Листинг файла index.cpp. Исходный код программы.

# **Тестирование**

В соответствии с постановкой задачи требовалось произвести ручное тестирования для квадратных матриц размером n=3. На рисунке 7 представлен скриншот результата работы программы, соответствующей исходным требованиям.



*Рисунок.7. Скриншот выполнения работы программы*



*Рисунок.8. Скриншот выполнения работы программы (1)*

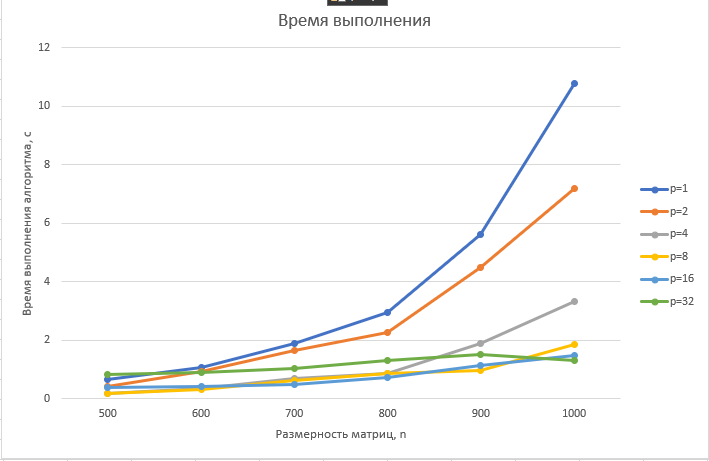
# **Контрольные прогоны**

На основе выполненных тестов были составлены сводные таблицы, на основе которых составлены графики, указанные в основной задаче.

*Таблица 1. Время работы алгоритма Фокса с параллельной и последовательной реализацией*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размерность, n | Время при последовательном перемножении | P = 2 | P=4 | P=8 | P=16 | P=32 |
| 500 | 0,671 | 0,419 | 0,18 | 0,182 | 0,393 | 0,834 |
| 600 | 1,08 | 0,949 | 0,344 | 0,323 | 0,436 | 0,886 |
| 700 | 1,902 | 1,649 | 0,678 | 0,609 | 0,475 | 1,021 |
| 800 | 2,955 | 2,273 | 0,878 | 0,849 | 0,727 | 1,324 |
| 900 | 5,624 | 4,503 | 1,904 | 0,974 | 1,155 | 1,508 |
| 1000 | 10,789 | 7,204 | 3,33 | 1,854 | 1,466 | 1,325 |

На основании этих данных был построен общий график зависимости время выполнения алгоритма от размера матриц (рисунок 8).

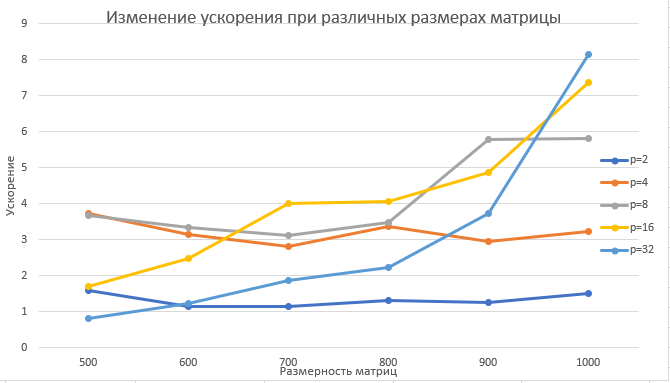


*Рисунок. 8. График зависимости времени выполнения программы от размерности матриц*

*Таблица 2. Изменение ускорения с параллельной и последовательной реализацией алгоритма Фокса в зависимости от размера матриц*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размерность, n | Ускорение Sn, при P = 2 | Ускорение Sn, при P=4 | Ускорение Sn, при P=8 | Ускорение Sn, при P=16 | Ускорение Sn, при P=32 |
| 500 | 1,603 | 3,721 | 3,685 | 1,702 | 0,802 |
| 600 | 1,135 | 3,136 | 3,345 | 2,471 | 1,211 |
| 700 | 1,151 | 2,808 | 3,12 | 4,008 | 1,863 |
| 800 | 1,3 | 3,369 | 3,42 | 4,069 | 2,233 |
| 900 | 1,249 | 2,951 | 5,773 | 4,869 | 3,729 |
| 1000 | 1,492 | 3,233 | 5,811 | 7,359 | 8,142 |

На основании этих данных был построен общий график зависимости ускорения от размера матриц (рисунок 9).

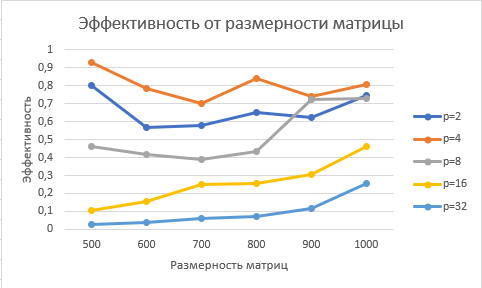


*Рисунок. 9. Зависимость ускорения от размерности матриц*

*Таблица 3. Изменение эффективность с параллельной и последовательной реализацией алгоритма Фокса в зависимости от размера матриц*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размерность, n | Эффективность En, при P = 2 | Эффективность En, при P=4 | Эффективность En, при P=8 | Эффективность En, при P=16 | Эффективность En, при P=32 |
| 500 | 0,8 | 0,931 | 0,41 | 0,102 | 0,025 |
| 600 | 0,569 | 0,784 | 0,415 | 0,151 | 0,038 |
| 700 | 0,576 | 0,701 | 0,33 | 0,26 | 0,058 |
| 800 | 0,65 | 0,841 | 0,435 | 0,255 | 0,069 |
| 900 | 0,624 | 0,738 | 0,726 | 0,303 | 0,116 |
| 1000 | 0,748 | 0,809 | 0,727 | 0,451 | 0,254 |

На основании этих данных был построен общий график зависимости эффективности от размера матриц (рисунок 10).

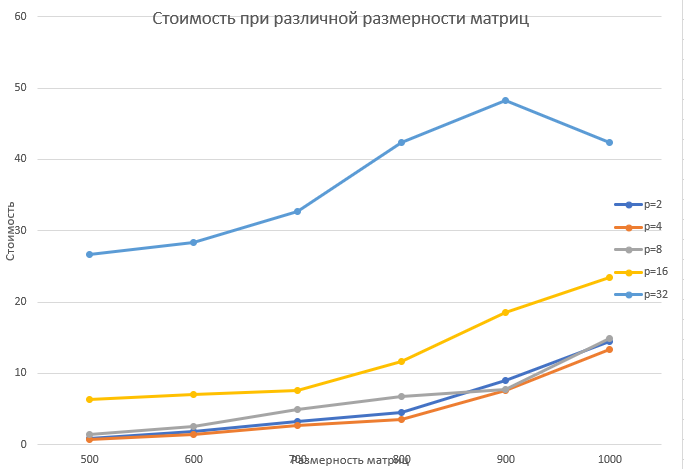


*Рисунок. 10. Зависимость эффективности от размерности матриц*

*Таблица 4. Изменение стоимости с параллельной и последовательной реализацией алгоритма Фокса в зависимости от размера матриц*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размерность, n | Стоимость Cn, при P = 2 | Стоимость Cn, при P=4 | Стоимость Cn, при P=8 | Стоимость Cn, при P=16 | Стоимость Cn, при P=32 |
| 500 | 0,838 | 0,72 | 1,456 | 6,288 | 26,688 |
| 600 | 1,898 | 1,376 | 2,584 | 6,976 | 28,352 |
| 700 | 3,298 | 2,712 | 4,872 | 7,6 | 32,672 |
| 800 | 4,546 | 3,512 | 6,792 | 11,632 | 42,368 |
| 900 | 9,006 | 7,616 | 7,792 | 18,48 | 48,256 |
| 1000 | 14,408 | 13,32 | 14,832 | 23,456 | 42,4 |

На основании этих данных был построен общий график зависимости стоимости от размера матриц (рисунок 11).



*Рисунок. 11. Зависимость стоимости от размерности матриц*

# **Выводы**

В ходе выполнения данной практической работы реализована программа по перемножению двух матриц при помощи блочного алгоритма Фокса при помощи библиотеки MPI для языка C++. Помимо этого, проведены тестирования программы, составлены сводные таблицы с результатами выполнения, графики.

1. Максимальное ускорение для n = 500 достигается при p = 4
2. Максимальное ускорение для n = 600 достигается при p = 8
3. Максимальное ускорение для n = 700 достигается при p = 16
4. Максимальное ускорение для n = 800 достигается при p = 16
5. Максимальное ускорение для n = 900 достигается при p = 8
6. Максимальное ускорение для n = 1000 достигается при p = 32
7. Для всех размерностей наибольшая эффективность достигается при p = 4;
8. Самый затратный алгоритм – с использованием количества процессов p = 32.
9. Менее затратный алгоритм – с использованием количества процессов p = 4.

# **Список используемых информационных источников**

1. Сыромятников В. П. Курс лекций по дисциплине «Параллельное программирование». – РТУ МИРЭА, 2020-2021 г.

2. Руководство по использованию пакета MPI [Электронный ресурс]. – Режим доступа: <http://appmath.mrsu.ru/assets/templates/appmath/pdf_docs/ParProg_MPI_OpenMP.pdf> (дата обращения 02.11.2020)

3. Официальные характеристики процессора Intel Core i5 4590 [Электронный ресурс]. – Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/80815/intel-core-i5-4590-processor-6m-cache-up-to-3-70-ghz.html> (дата обращения 02.11.2020)

4. Официальный сайт дистрибутива MVS Code [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com> (дата обращения 02.11.2020)

5. «Алгоритм Фокса умножения матриц при блочном разделении данных» [Электронный ресурс] – Режим доступа: <https://intuit.ru/studies/courses/1156/190/lecture/4954?page=3> (дата обращения 02.11.2020)