**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**Кафедра «Информатика и программное обеспечение»**

**ЛАБОРАТОРНАЯ РАБОТА № 3**

**ПО ДИСЦИПЛИНЕ**

**«Архитектура вычислительных систем»**

**Тема: ТЕХНОЛОГИЯ OPENmp: директива описания параллельного цикла, упорядочивание итераций**

Выполнил студент гр. О-21-ИВТ-1-по-Б

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шпаков В.В.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_ к.ф.-м.н., доц. Дмитроченко О.Н.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

**Брянск 2024**

**Содержание**

[1. Цель работы 3](#_Toc155667197)

[2. Теоретические сведения 4](#_Toc155667198)

[2.1. Распараллеливание циклических инструкций 4](#_Toc155667199)

[2.2. Директива упорядочивания итераций 5](#_Toc155667200)

[2.3. Альтернативный способ определения параллельного цикла 5](#_Toc155667201)

[3. Задания 7](#_Toc155667202)

[3.1. Задание 1 7](#_Toc155667203)

[3.2. Задание 2 8](#_Toc155667204)

[3.3. Задание 3 9](#_Toc155667205)

[3.4. Задание 4 11](#_Toc155667206)

[4. Контрольные вопросы 14](#_Toc155667207)

# Цель работы

Целью лабораторной работы является:

1. получение теоретических знаний об особенностях распределения итераций цикла между несколькими потоками с использованием возможностей библиотеки OpenMP;
2. изучение основных схем распределения данных между итерациями в параллельном цикле, применение общих массивов и операции редукции;
3. приобретение практических навыков синхронизации итераций параллельного цикла с использованием директивы упорядочивания.

# Теоретические сведения

## Распараллеливание циклических инструкций

Если в параллельной области встретился оператор цикла, то, согласно общему правилу, он будет выполнен всеми нитями текущей группы, т.е. каждая нить выполнит все итерации данного цикла. Для распределения итераций цикла между различными нитями можно использовать директиву *for*.

#pragma omp for [опция[[,] опция]...]

for(..;..;..)

{

//Тело распараллеливаемого цикла

}

Эта директива относится к идущему следом за данной директивой блоку.

Директива *for* предполагает использование опций:

*private(список)* – задаёт список переменных, для которых порождается локальная копия в каждой нити; начальное значение локальных копий переменных из списка не определено;

*reduction(оператор:список***)** – задаёт оператор и список общих переменных; для каждой переменной создаются локальные копии в каждой нити; над локальными копиями переменных после завершения всех итераций цикла выполняется заданный оператор;

*ordered* – опция, говорящая о том, что в цикле могут встречаться директивы *ordered*; в этом случае определяется блок внутри тела цикла, который должен выполняться в том порядке, в котором итерации идут в последовательном цикле.

На вид параллельных циклов накладываются достаточно жёсткие ограничения. В частности, предполагается, что корректная программа не должна зависеть от того, какая именно нить какую итерацию параллельного цикла выполнит. Нельзя использовать побочный выход из параллельного цикла.

Формат параллельных циклов упрощённо можно представить следующим образом:

for(int i = число;i {<,>,=,<=,>=} число;

i {+,-}= число)

Эти требования введены для того, чтобы OpenMP мог при входе в цикл точно определить число итераций.

Итеративная переменная распределяемого цикла по смыслу должна быть локальной, поэтому, если она специфицирована общей, то она неявно делается локальной при входе в цикл. После завершения цикла значение итеративной переменной цикла не определено.

При реализации алгоритма с помощью директивы *parallel for*алгоритма, обладающего свойствами параллелизма по данным, как правило, используют в вычислениях массивы данных. При этом каждый поток оперируется своим набором элементов массива. Тем самым обеспечивается возможность одновременного доступа к общему массиву множества потоков, гарантируется независимость результата от порядка выполнения итераций.

Примером такого алгоритма является реализация задачи нахождения квадрата первых шести натуральных чисел. Использование массива из шести элементов для хранения результатов и применение директивы *parallel for*позволяет повысить скорость вычисления результата за счет распределения итераций цикла по нескольким потокам.

## Директива упорядочивания итераций

Директива *ordered* определяет блок внутри тела цикла, который должен выполняться в том порядке, в котором итерации идут в последовательном цикле.

#pragma omp ordered

При этом в параллельном цикле должна быть задана опция *ordered*. Нить, выполняющая первую итерацию цикла, выполняет операции данного блока. Нить, выполняющая любую следующую итерацию, должна сначала дождаться выполнения всех операций блока всеми нитями, выполняющими предыдущие итерации. Может использоваться, например, для упорядочения вывода от параллельных нитей. Директива удобна в случае зависимости результатов вычислений в параллельном цикле от порядка выполнения итераций.

В качестве примера применения данной директивы можно привести программу, которая реализует многопоточную операцию вывода текста на экран. При этом каждая итерация цикла предполагает вывод двух строк: одной – без ожидания завершения предшествующих итераций, другой – с ожиданием завершения.

## Альтернативный способ определения параллельного цикла

Так как циклы являются самыми распространенными конструкциями, где выполнение кода можно распараллелить, OpenMP поддерживает сокращенный способ записи комбинации директив *#pragma omp parallel* и *#pragma omp for*:

#pragma omp parallel for [опция [[,]

опция]...]

for(..;..;..)

{

//Тело распараллеливаемого цикла

}

# Задания

## Задание 1

Разработать программу, инициализирующую два исходных массива *A* и *B* случайными числами и рассчитывающую в параллельной области в результирующий массив *C* сумму исходных массивов. На каждой итерации вывести номер нити и номер рассчитанного элемента массива.

*Листинг 1*

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <clocale>  #include <stdio.h>  #include <ctime>  using namespace std;  int main()  {  setlocale(LC\_ALL, "Russian");  srand(time(NULL));  const int n = 5;  int A[n] = { 0 }, B[n] = { 0 }, C[n] = { 0 };  int i = 0;  cout << "Матрица A:" << endl;  for (i = 0; i < n; i++)  {  A[i] = 1 + rand() % 100;  cout << "\t" << A[i] << endl;  }  cout << "Матрица B:" << endl;  for (i = 0; i < n; i++)  {  B[i] = 1 + rand() % 100;  cout << "\t" << B[i] << endl;  }  cout << endl;  #pragma omp parallel private(i)  {  #pragma omp for  for (i = 0; i < n; i++)  {  C[i] = A[i] + B[i];  cout << "Номер нити:" << omp\_get\_thread\_num() << ", номер элемента: " << i << endl;  }  }  cout << endl << "Матрица C:" << endl;  for (i = 0; i < n; i++)  cout << "\t" << C[i] << endl;  return 0;  } |

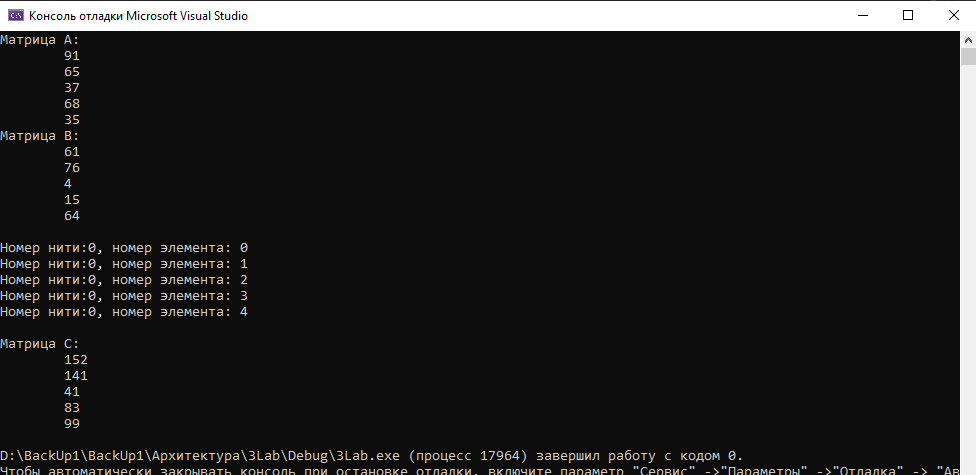


Рис. 1. Результат выполнения

## Задание 2

Разработать программу нахождения величины *sum* для двух исходных, заполненных случайными величинами векторов *A* и *B* длиной 100 элементов по формуле . Использовать вычисления в параллельном цикле с последующей операцией редукции.

*Листинг 2*

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <clocale>  #include <stdio.h>  #include <ctime>  using namespace std;  int main()  {  setlocale(LC\_ALL, "ru");  srand(time(NULL));  const int n = 100;  int A[n] = { 0 }, B[n] = { 0 };  int i = 0, sum = 0;  cout << "Матрица A:" << endl;  for (i = 0; i < n; i++)  {  A[i] = 1 + rand() % 100;  cout << "\t" << A[i];  }  cout << endl << endl << "Матрица B:" << endl;  for (i = 0; i < n; i++)  {  B[i] = 1 + rand() % 100;  cout << "\t" << B[i];  }  cout << endl;  // Используем reduction для переменной sum  #pragma omp parallel for reduction(+:sum)  for (i = 0; i < n; i++)  sum += A[i] \* B[i];  cout << endl << "Результат суммы произведений: " << sum << endl;  return 0;  } |

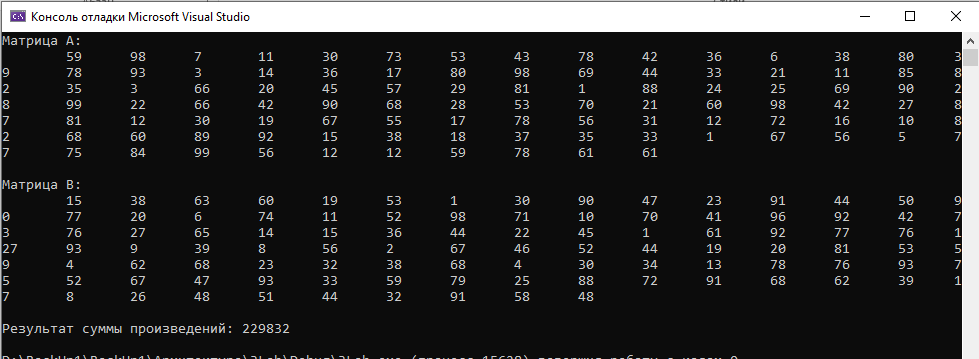


Рис. 2.Результат выполнения

## Задание 3

Разработать программу параллельного умножения квадратной матрицы на вектор, длина вектора и размер матрицы задаются константой. Распределить по нескольким потокам в параллельном цикле вычисление элементов результирующего вектора по формуле . На каждом шаге при вычислении очередного элемента результирующего вектора вывести на экран строку с указанием номера потока, индекса элемента и его значения. Вывести на экран исходные матрицу и вектор, результирующий вектор.

*Листинг 3*

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <clocale>  #include <stdio.h>  #include <ctime>  using namespace std;  int\*\* createMatrix(int size) {  int\*\* matrix = new int\* [size];  for (int i = 0; i < size; i++)  matrix[i] = new int[size];  for (int i = 0; i < size; i++)  for (int j = 0; j < size; j++)  matrix[i][j] = 1 + rand() % 1000;  return matrix;  }  void printMatrix(int\*\* matrix, int size) {  cout << " Матрица:" << endl;  for (int i = 0; i < size; i++) {  cout << "\t";  for (int j = 0; j < size; j++)  cout << matrix[i][j] << " ";  cout << endl;  }  }  void deleteMatrix(int\*\* matrix, int size) {  for (int i = 0; i < size; i++)  delete[] matrix[i];  delete[] matrix;  }  int main()  {  setlocale(LC\_ALL, "ru");  srand(time(nullptr));  const int matrixSize = 3;  int\*\* matrixA;  int vectorB[matrixSize] = { 0 }, resultVector[matrixSize] = { 0 };  matrixA = createMatrix(matrixSize);  for (int i = 0; i < matrixSize; i++)  vectorB[i] = 1 + rand() % 100;  printMatrix(matrixA, matrixSize);  cout << endl << " Вектор B:" << endl;  for (int i = 0; i < matrixSize; i++)  cout << "\tB[" << i << "] = " << vectorB[i] << endl;  cout << endl;  omp\_set\_nested(1);  #pragma omp parallel for schedule(static) collapse(2)  for (int i = 0; i < matrixSize; i++) {  for (int j = 0; j < matrixSize; j++) {  resultVector[i] += matrixA[i][j] \* vectorB[j];  cout << " Номер потока: " << omp\_get\_thread\_num() << ", индекс: " << i << ", значение: " << resultVector[i] << endl;  }  }  cout << endl << " Итоговый вектор:" << endl;  for (int i = 0; i < matrixSize; i++)  cout << "\tResult[" << i << "] = " << resultVector[i] << endl;  deleteMatrix(matrixA, matrixSize);  return 0; |

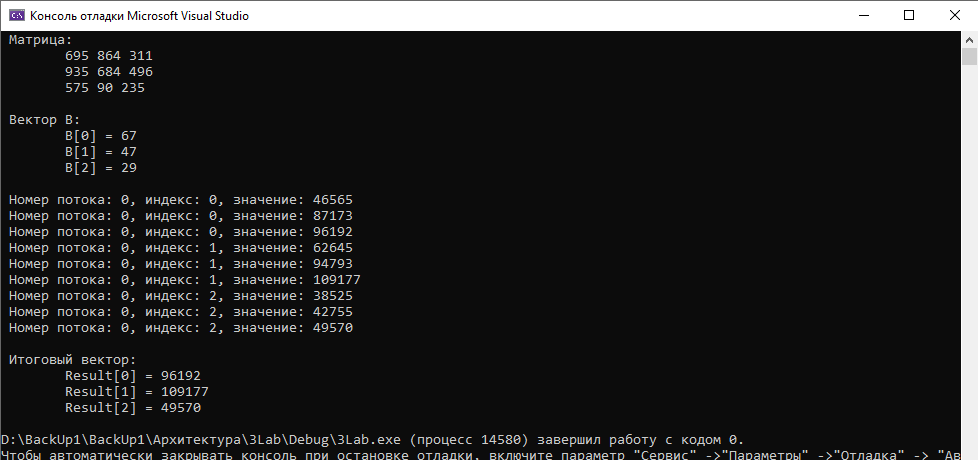


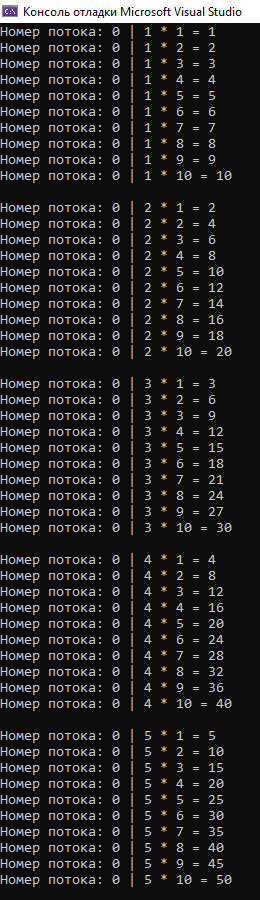
Рис. 3.Результат выполнения

## Задание 4

Разработать параллельную программу, выводящую таблицу умножения. Каждую строку таблицы умножения выводить в отдельном потоке, определив нужное количество потоков до начала исполнения цикла. Добиться упорядоченного вывода строк в цикле с использованием директивы *ordered*.

*Листинг 4*

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <clocale>  using namespace std;  int main() {  setlocale(LC\_ALL, "ru");  const int n = 11;  omp\_set\_nested(1);  #pragma omp parallel for ordered  for (int i = 1; i < n; i++) {  #pragma omp ordered  {  #pragma omp parallel for ordered num\_threads(10)  for (int j = 1; j < n; j++) {  #pragma omp ordered  cout << "Номер потока: " << omp\_get\_thread\_num() << " | " << i << " \* " << j << " = " << i \* j << endl;  }  cout << endl;  }  }  return 0;  } |

**

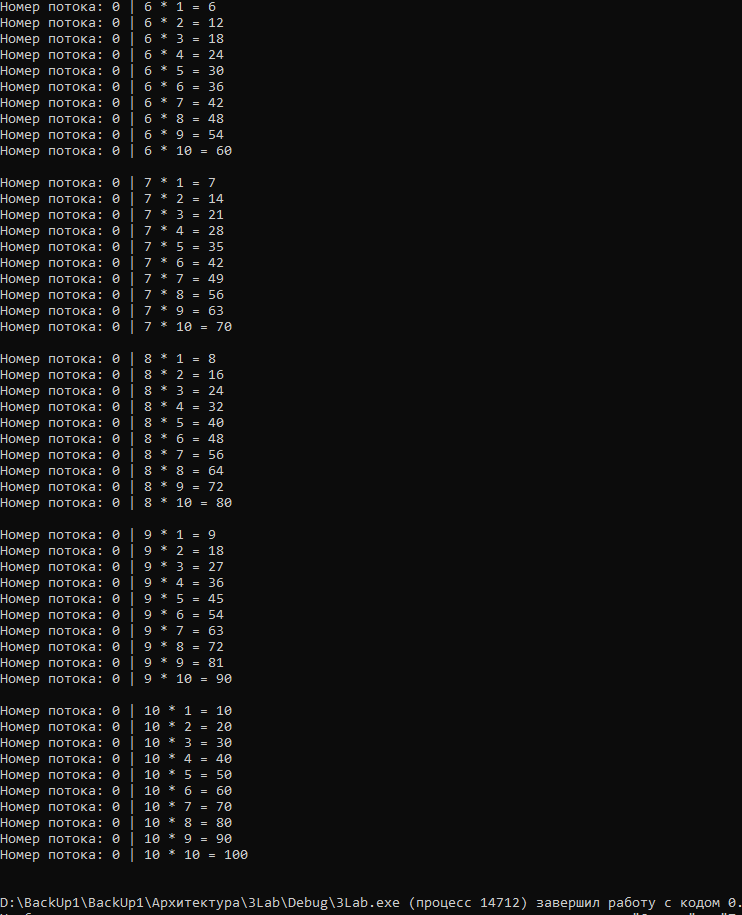


Рис. 4.Результат выполнения

# Контрольные вопросы

1. **Какой результат будет получен при использовании последовательного цикла *for* в параллельном регионе программы?**

Каждая нить выполнит все итерации данного цикла.

1. **Опишите формат директивы описания параллельного цикла, перечислите основные опции.**

Директива ordered в OpenMP применяется к блоку кода, следующему за ней, и указывает, что в этом блоке могут встречаться другие директивы ordered. Некоторые опции, которые могут использоваться с этой директивой:

private(список) – задает список переменных, для которых создаются локальные копии в каждой нити; начальное значение локальных копий переменных из списка не определено;

reduction(оператор:список) – определяет оператор и список общих переменных; для каждой переменной создаются локальные копии в каждой нити; после завершения всех итераций цикла выполняется заданный оператор над локальными копиями переменных;

ordered – опция, указывающая, что в цикле могут встречаться директивы ordered; в этом случае определяется блок внутри тела цикла, который должен выполняться в том порядке, в котором итерации выполняются в последовательном цикле.

Директива for используется для распараллеливания циклов, а дополнительные опции могут включать в себя параметры, такие как private и reduction.

1. **С чем связаны ограничения, накладываемые на реализуемый в теле параллельного цикла алгоритм?**

Корректная программа не должна зависеть от того, какая именно нить какую итерацию параллельного цикла выполнит. Нельзя использовать побочный выход из параллельного цикла. Формат параллельных циклов упрощённо можно представить следующим образом: for(int i = число; i {,=,<=,>=} число; i {+,-}= число) Эти требования введены для того, чтобы OpenMP мог при входе в цикл точно определить число итераций.

1. **Каким образом используя массивы данных в параллельном цикле можно избежать «эффекта гонок»?**

Каждый поток обрабатывает свой набор элементов массива, что обеспечивает возможность одновременного доступа нескольких потоков к общему массиву. Таким образом, гарантируется, что результат выполнения не зависит от порядка выполнения итераций, и обеспечивается независимость результатов работы различных потоков.

1. **Приведите пример алгоритма, поддающегося эффективному разложению на независимо исполняемые параллельные итерации.**

Задача нахождения квадрата первых шести натуральных чисел. Использование массива из шести элементов для хранения результатов и применение директивы parallel for позволяет повысить скорость вычисления результата за счет распределения итераций цикла по нескольким потокам.

1. **Можно ли заранее предположить порядок исполнения итераций в параллельном цикле?**

Только с использованием опции ordered.

1. **Какие опции могут использоваться для автоматической синхронизации порядка исполнения итераций в параллельном цикле?**

ordered.

1. **Каким образом влияет директива упорядочивания итераций цикла на время исполнения параллельной программы?**

Происходит замедление работы программы.

1. **Как сократить формат записи директив описания параллельного региона и определения параллельного цикла?**

#pragma omp parallel for{}

1. **Может ли счетчик параллельного цикла находиться в общей области памяти параллельных потоков?**

Нет, не может

1. **В чем особенности применения операции редукции к результатам выполнения итераций параллельных циклов?**

Над локальными копиями переменных после завершения всех итераций цикла выполняется заданный оператор.

1. **Каким образом определить количество потоков, по которым осуществляется распределение итераций параллельного цикла?**

C помощью опции редукции.