**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**Кафедра «Информатика и программное обеспечение»**

**ЛАБОРАТОРНАЯ РАБОТА № 1**

**ПО ДИСЦИПЛИНЕ**

**«Архитектура вычислительных систем»**

**Тема: РАЗРАБОТКА ПАРАЛЛЕЛЬНЫХ ПРИЛОЖЕНИЙ. ПОНЯТИЕ ПОТОКА И ПРОЦЕССА. ОСНОВЫ ТЕХНОЛОГИИ OPENMP: ДИРЕКТИВЫ, ОПЦИИ, ФУНКЦИИ, ПЕРЕМЕННЫЕ ОКРУЖЕНИЯ. ПАРАЛЛЕЛЬНЫЕ И ПОСЛЕДОВАТЕЛЬНЫЕ ОБЛАСТИ.**

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](#_Toc155655708)

[2. Теоретические сведения 4](#_Toc155655709)

[3. Задания 8](#_Toc155655710)

[3.1. Задание 1 8](#_Toc155655711)

[3.2. Задание 2 9](#_Toc155655712)

[3.3. Задание 3 10](#_Toc155655713)

[3.4. Задание 4 11](#_Toc155655714)

[3.5. Задание 5 12](#_Toc155655715)

[4. Контрольные вопросы 15](#_Toc155655716)

# Цель работы

1. Получение теоретических знаний об особенность выполнения параллельных программ в многозадачных операционных системах.
2. Изучение основ технологии OpenMP: настройка, инициализация, параллельные секции, функции определения параметров параллельной области.
3. Приобретение практических знаний и навыков в компиляции и запуске простейших OpenMP-программ.

# Теоретические сведения

Любая параллельная программа (*application*) в терминологии операционной системы обозначается понятием «*процесс*». Процессы статичны и состоят из набора потоков, исполняющих программный код, и области данных. Процесс может содержать много потоков, но обязательно содержит, по крайней мере, один. Когда системе предлагают программу «запустить на исполнение» она создает один процесс и один поток, в котором и запускает программу.

Построить многозадачную систему, эмулируя одновременную работу нескольких процессоров можно, переключая время от времени программы, исполняющиеся на процессоре. Если квант времени, в течение которого процессор занят одной программой, не очень велик, то будет создаваться иллюзия, что все программы исполняются системой одновременно. Однако переключение между процессами – довольно трудоемкая операция, требующая значительных процессорных ресурсов.

*Поток (нить, thread)* – динамическая сущность, последовательность исполнения команд внутри процесса, разделяющая вместе с процессом общее адресное пространство, код и глобальные данные. В одном процессе может быть несколько потоков (*многопоточность*). Процесс можно рассматривать как способ объединения родственных ресурсов в одну группу. Потоки создаются таким образом, чтобы работать совместно, разделяя ресурсы общего процесса и не мешая друг другу.

Многопоточность позволяет нескольким потокам выполняться в рамках одного процесса, совместно используя ресурсы процесса и при этом работая самостоятельно. Применяя многопоточную модель для разделения одного процесса на несколько потоков, становится возможным организовать его параллельное выполнение на многопроцессорной системе. Это преимущество многопоточной программы позволяет ей работать быстрее на компьютерных системах, которые имеют несколько процессоров или процессор с несколькими ядрами.

Технология OpenMP является на сегодняшний день наиболее популярным средством для программирования на масштабируемых SMP-системах (SSMP, ccNUMA и других) в модели общей памяти (*shared memory model*). Особенностью данной технологии является возможность естественного преобразования последовательной программы в параллельную с помощью библиотечных процедур, директив и переменных окружения.

OpenMP реализует параллельные вычисления с помощью многопоточности, в которой «главный» (*master*) поток создает набор «подчиненных» (*slave*) потоков, и задача распределяется между ними. Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами, причём количество процессоров не обязательно должно быть больше или равно количеству потоков.

Значительная часть функциональности **OpenMP** реализуется при помощи директив компилятору. Директивы **OpenMP** в программах оформляются указаниями препроцессору, начинающимися с *#pragma omp*. Ключевое слово *omp* используется для того, чтобы исключить случайные совпадения имён директив **OpenMP** с другими именами.

Объектом действия большинства директив является один оператор или блок, перед которым расположена директива в исходном тексте программы. В **OpenMP** такие операторы или блоки называются *ассоциированными* с директивой. Ассоциированный блок должен иметь одну точку входа в начале и одну точку выхода в конце. Порядок опций в описании директивы несущественен, в одной директиве большинство опций может встречаться несколько раз. После некоторых опций может следовать список переменных разделяемых запятыми.

Все директивы **OpenMP** можно разделить на 3 категории:

определение параллельной области;

распределение работы;

синхронизация.

Каждая директива может иметь несколько дополнительных атрибутов – опций (*clause*). Отдельно специфицируются опции для назначения классов переменных, которые могут быть атрибутами различных директив.

Чтобы задействовать функции библиотеки **OpenMP** периода выполнения (исполняющей среды), в программу нужно включить заголовочный файл **omp.h**. При использовании в приложении только **OpenMP**-директивы, включать этот файл не требуется.

Все функции, используемые в OpenMP, начинаются с префикса **omp\_**. Если пользователь не будет использовать в программе имён, начинающихся с такого префикса, то конфликтов с объектами OpenMP заведомо не будет. Названия функций OpenMP записываются строчными буквами.

Некоторые опции:

**if(условие)** – выполнение параллельной области по условию. Вхождение в параллельную область осуществляется только при выполнении некоторого условия. Если условие не выполнено, то директива не срабатывает и продолжается обработка программы в прежнем режиме;

**num\_threads(целочисленное\_выражение)** – явное задание количества нитей, которые будут выполнять параллельную область; по умолчанию выбирается последнее значение, установленное с помощью функции *omp\_set\_num\_threads()*, или значение переменной *OMP\_NUM\_THREADS*;

При входе в параллельную область порождаются новые *OMP\_NUM\_THREADS*-1 нитей, каждая нить получает свой уникальный номер, причём порождающая нить получает номер 0 и становится основной нитью группы («*мастером*»).

Остальные нити получают в качестве номера целые числа с 1 до *OMP\_NUM\_THREADS*-1. Количество нитей, выполняющих данную параллельную область, остаётся неизменным до момента выхода из области. При выходе из параллельной области производится неявная синхронизация и уничтожаются все нити, кроме породившей.

Все порождённые нити исполняют один и тот же код, соответствующий параллельной области. Предполагается, что в SMP-системе нити будут распределены по различным процессорам (однако это, как правило, находится в ведении операционной системы).

Использование функций *omp\_get\_thread\_num()* и *omp\_get\_num\_threads()* позволяет назначать каждой нити свой кусок кода для выполнения, и таким образом распределять работу между нитями. Однако использование этого стиля программирования в OpenMP далеко не всегда оправдано – программист в этом случае должен явно организовывать синхронизацию доступа к общим данным. Другие способы распределения работ в OpenMP обеспечивают значительную часть этой работы автоматически.

Количество нитей, с которым будет запущен параллельный регион, определяется переменной среды OMP\_NUM\_THREADS. Ее значение по умолчанию зависит от реализации. Более высокий приоритет над значением OMP\_NUM\_THREADS имеет опция *num\_threads* параллельного региона.

Задать значение переменной среды OMP\_NUM\_THREADS можно несколькими способами: из командной строки, в настройках проекта, вызовом функции.

# Задания

Для начала нужно подключить поддержку Open MP (см. рис. 1).

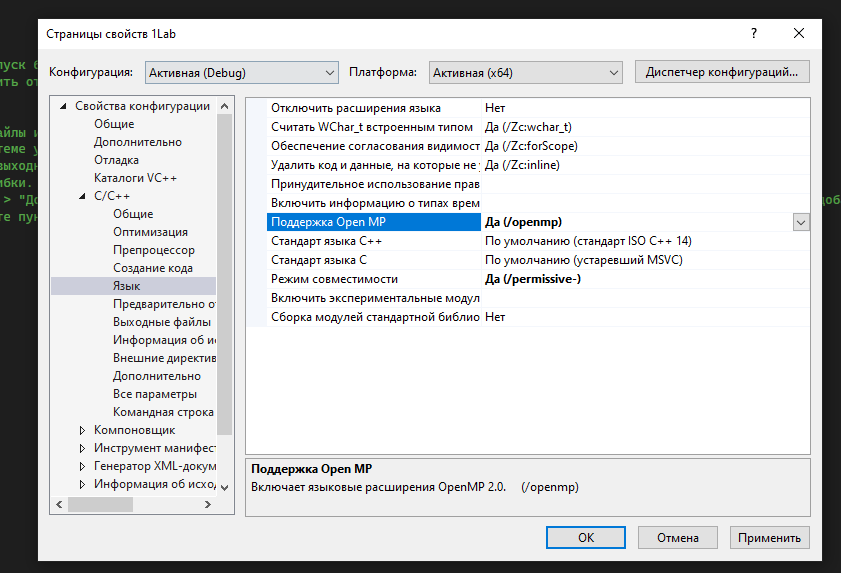


Рис. 1 . Поддержка Open MP

## Задание 1

Разработать программу, содержащую один параллельный регион. До входа в параллельную область на экран выводится текст «Последовательная область 1», в параллельной области каждая нить печатает текст «Параллельная область», после завершения параллельной области оставшаяся нить-мастер печатает текст «Последовательная область 2».

*Листинг 1*

#include <omp.h>

#include <stdio.h>

#include <iostream>

#include <clocale>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

*Продолжение Листинга 1*

int NumOfThreads, ThreadId;

omp\_set\_num\_threads(5);

cout << "Последовательная область 1" << endl;

#pragma omp parallel

{

ThreadId = omp\_get\_thread\_num();

NumOfThreads = omp\_get\_num\_threads();

cout << "Параллельная область" << endl;

}

cout << "Последовательная область 2" << endl;

return 0;

}

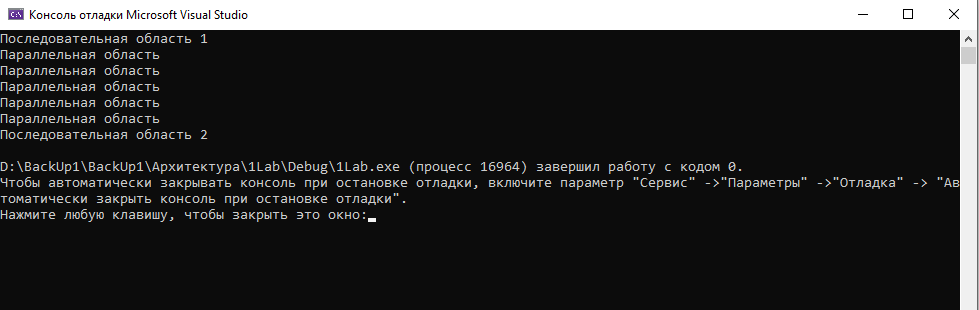


Рис. 2.Результат задания 1

## Задание 2

Разработать параллельную программу, нить-мастер которой печатает в консоли общее количество порождённых нитей, а остальные нити печатают свой порядковый номер.

*Листинг 2*

#include <omp.h>

#include <stdio.h>

#include <iostream>

#include <clocale>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int NumOfThreads, ThreadId;

omp\_set\_num\_threads(7);

int all = omp\_get\_num\_threads();

cout << "Общее число нитей: " << all << endl;

#pragma omp parallel

{

ThreadId = omp\_get\_thread\_num();

NumOfThreads = omp\_get\_num\_threads();

cout << " Поток " << ThreadId << " из " << NumOfThreads << endl;

}

return 0;

}

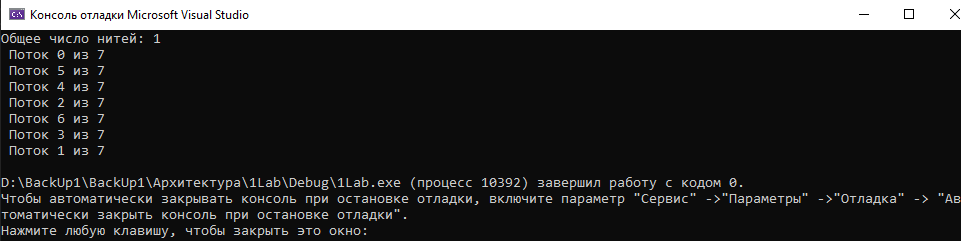


Рис. 3.Результат задания 2

## Задание 3

Разработать программу, состоящую из трех параллельных регионов. В каждом расположена функция вывода строки на консоль вида «Параллельный регион N», где N – номер региона. Для первого региона установить для исполнения две нити с помощью переменной окружения в настройках проекта, для второго – три нити с помощью опции *num\_threads*, для третьего – четыре нити с помощью вызова функции.

Необходимо установить количество нитей для первого региона через настройки проекта (см. рис. 4).

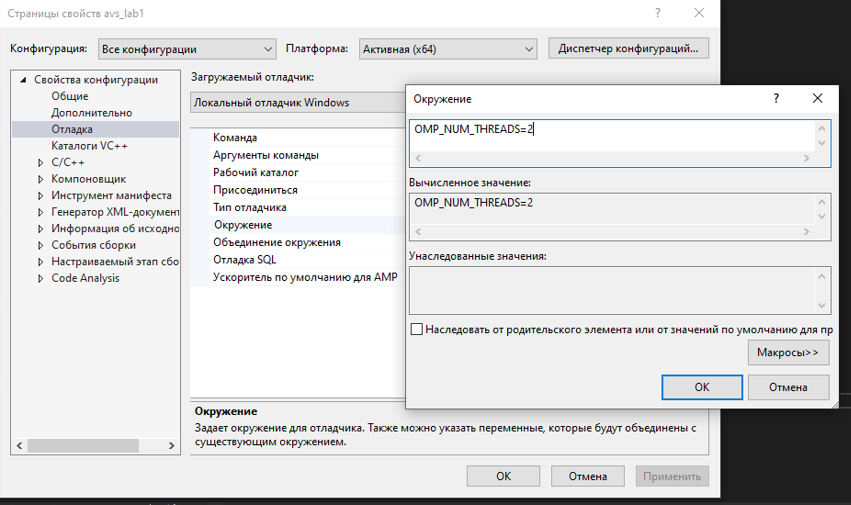


Рис. 4.Установка количества нитей

*Листинг 3*

#include <iostream>

#include <stdio.h>

#include <omp.h>

#include <clocale>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

#pragma omp parallel

{

cout << "Параллельный регион 1, номер потока: " << omp\_get\_thread\_num() <<endl;

}

cout << endl;

#pragma omp parallel num\_threads(3)

{

cout << "Параллельный регион 2, номер потока: " << omp\_get\_thread\_num() << endl;

}

cout << endl;

omp\_set\_num\_threads(4);

#pragma omp parallel

{

cout << "Параллельный регион 3, номер потока : " << omp\_get\_thread\_num() << endl;

}

return 0;

}

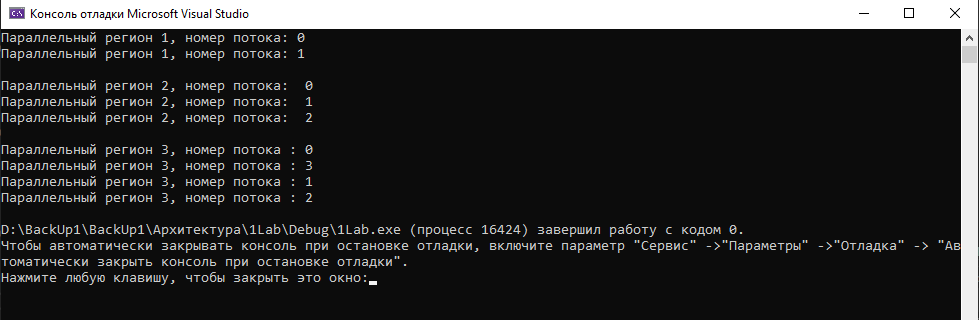


Рис. 5.Результат задания 3

## Задание 4

Разработать программу, включающую параллельный регион, выполняемый в 128 нитях. Указать вывод количества нитей в потоке-мастере. Проверить количество созданных для исполнения параллельного региона потоков при отключенном и включенном динамическом режиме. Использовать для задания динамического режима переменную среды или вызов функции.

*Листинг 4*

#include <omp.h>

#include <stdio.h>

#include <iostream>

#include <clocale>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int NumOfThreads, ThreadId;

omp\_set\_num\_threads(128);

omp\_set\_dynamic(0);

#pragma omp parallel

{

ThreadId = omp\_get\_thread\_num();

NumOfThreads = omp\_get\_num\_threads();

cout << ThreadId << " из " << NumOfThreads << endl;

}

return 0;

}



Рис. 6.Результат задания 4

## Задание 5

Разработать программу, которая принимает из командной строки параметр «parallel» или «serial» и определяет по его значению необходимость запуска параллельного региона или выполнения программы в обычном последовательном режиме. Использовать для реализации программы условную опцию параллельного региона.

*Листинг 5*

|  |
| --- |
| #include <omp.h>  #include <iostream>  #include <string>  #include <clocale>  using namespace std;  int main() {  setlocale(LC\_ALL, "ru");  int NumOfThreads;  int entrance;  cout << "Выберите режим\n1) parallel\n2) serial\n";  cin >> entrance;  if (entrance == 1) {  cout << "Параллельный режим" << endl;  #pragma omp parallel  {  int ThreadId;  ThreadId = omp\_get\_thread\_num();  NumOfThreads = omp\_get\_num\_threads();  printf("Поток номер %d из %d\n", ThreadId, NumOfThreads);  }  }  else if (entrance == 2) {  cout << "Последовательный режим" << endl;  }  else {  cout << "Неверное выражение" << endl;  return 1;  }  return 0;  } |

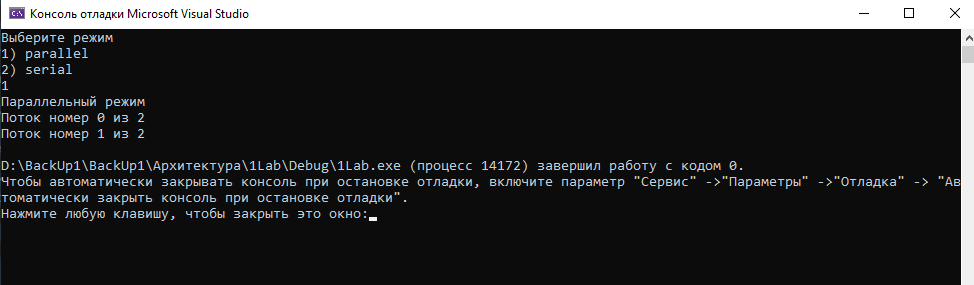


Рис. 7.Результат задания 5 в параллельном режиме

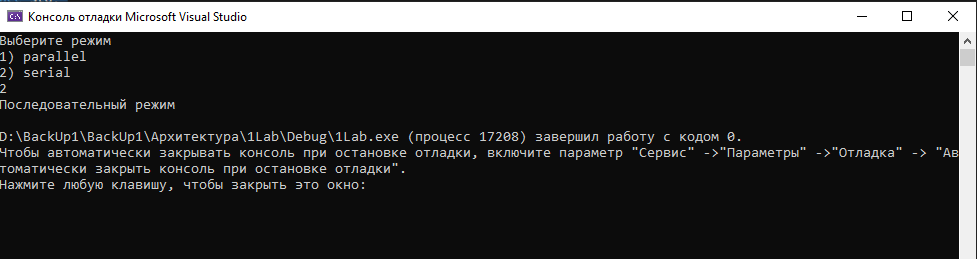


Рис. 8.Результат задания 5 в последовательном режиме

# Контрольные вопросы

1. **Каким образом связаны между собой процессы и потоки в многозадачной операционной системе?**

Любая параллельная программа обозначается понятием «процесс». Процессы статичны и состоят из набора потоков, исполняющих программный код, и области данных. Процесс может содержать много потоков, но обязательно содержит, по крайней мере, один.

1. **Каковы преимущества выполнения многопоточных приложений на многопроцессорных вычислительных системах?**

Многопоточные программы будут работать намного быстрее на компьютерах или компьютерных системах, в которых много ядер у процессора.

1. **Что такое технология OpenMP?**

Открытый стандарт для распараллеливания программ на языках Си, Си++ и Фортран.

1. **Опишите класс вычислительных систем, для которых предназначена OpenMP модель программирования.**

Технология OpenMP при верном написании многопоточной программы позволяет равномерно распределить нагрузку на процессор или вообще распределить нагрузку между процессорами, если таковых несколько.

1. **Что такое «главный» и «подчиненный» потоки?**

OpenMP реализует параллельные вычисления с помощью многопоточности, в которой «главный» поток создает набор «подчиненных» потоков, и задача распределяется между ними

1. **Каким образом осуществляется подключение OpenMP?**

В свойствах проекта C++ -> Язык -> Поддержка OpenMP.

1. **Как определить подключение библиотеки OpenMP?**

Достаточно написать директивы условной компиляции #ifdef или #ifndef.

1. **Опишите формат директив компилятору.**

#pragma omp parallel [опция[[,] опция]...]

1. **Опишите различие между последовательными и параллельными регионами?**

В последовательной области – работает один процесс (нить). В параллельной области порождается некоторое число процессов, между которыми в дальнейшем распределяются части кода.

1. **Каково назначение нити-мастера и порожденных нитей? Каково время их существования в системе?**

По завершении параллельной области все нити, кроме одной (нити-мастера), завершаются, и начинается последовательная область.

1. **Что такое блок, ассоциированный с оператором?**

Объектом действия большинства директив является один оператор или блок, перед которым расположена директива в исходном тексте программы. В OpenMP такие операторы или блоки называются ассоциированными с директивой.

1. **Каким образом описывается параллельная секция?**

#pragma omp parallel

1. **Каково назначение общих и частных переменных?**

Общая переменная доступна всем OpenMP-потокам. Частные переменные локализованы, и каждый процесс обладает собственной копией такой переменной.

1. **Каким образом определить номер потока, выполняющего программный код?**

omp\_get\_thread\_num()

1. **Опишите способы определения количества потоков для выполнения параллельного региона. Входит ли в это число поток-мастер?**

При помощи функции omp\_set\_num\_threads.

При помощи переменной окружения OMP\_NUM\_THREADS=N.

#pragma omp parallel num\_threads(N)