**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**Кафедра «Информатика и программное обеспечение»**

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**ПО ДИСЦИПЛИНЕ**

**«Архитектура вычислительных систем»**

**Тема: ТЕХНОЛОГИЯ OPENmp: ФУНКЦИИ РАБОТЫ СО ВРЕМЕНЕМ, модели памяти, операция редукции**

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**

# Цель работы

Целями данной работы являются:

1. получение теоретических знаний об измерении временных интервалов выполнения участков кода в параллельной программе;
2. изучение классов памяти технологии OpenMP, особенностей работы с переменными, находящимися в общей памяти потоков, методов определения локализованных приватных переменных;
3. приобретение практических навыков использования операции редукции для нахождения общей суммы локальных частных значений.

# Теоретические сведения

## Функции для работы с системным таймером

В стандарте OpenMP предусмотрены функции для работы с системным таймером. Функция

double omp\_get\_wtime(void);

возвращает в вызвавшей нити астрономическое время в секундах (вещественное число двойной точности), прошедшее с некоторого момента в прошлом.

Если некоторый участок программы окружить вызовами данной функции, то разность возвращаемых значений покажет время работы данного участка.

Гарантируется, что момент времени, используемый в качестве точки отсчета, не будет изменён за время существования процесса. Таймеры разных нитей могут быть не синхронизированы и выдавать различные значения.

## Директивы распределения работы между потоками

Если в параллельной области какой-либо участок кода должен быть выполнен только один раз, то его необходимо выделить директивой:

#pragma omp single [nowait]

После выполнения отмеченного данной директивой участка программы происходит неявная барьерная синхронизация параллельно работающих нитей. Их дальнейшее выполнение происходит только тогда, когда все они достигнут данной точки. Если в подобной задержке нет необходимости, опция *nowait* позволяет нитям, уже дошедшим до конца участка, продолжить выполнение без синхронизации с остальными.

Какая именно нить будет выполнять выделенный участок программы, не специфицируется. Одна нить будет выполнять данный фрагмент, а все остальные нити будут ожидать завершения её работы, если только не указана опция *nowait*. Необходимость использования директивы *single* часто возникает при работе с общими переменными.

Директива *master* выделяет участок кода, который будет выполнен только нитью-мастером:

#pragma omp master

Остальные нити просто пропускают данный участок и продолжают работу с оператора, расположенного следом за ним. Неявной синхронизации данная директива не предполагает.

## Модель данных

Модель данных в OpenMP предполагает наличие как общей для всех нитей области памяти, так и локальной области памяти для каждой нити.

В OpenMP переменные в параллельных областях программы разделяются на два основных класса:

* *shared* (общие; все нити видят одну и ту же переменную);
* *private* (локальные, приватные; каждая нить видит свой экземпляр данной переменной).

Общая переменная всегда существует лишь в одном экземпляре для всей области действия и доступна всем нитям под одним и тем же именем. Объявление локальной переменной вызывает порождение своего экземпляра данной переменной (того же типа и размера) для каждой нити. Изменение нитью значения своей локальной переменной никак не влияет на изменение значения этой же локальной переменной в других нитях.

Если несколько переменных одновременно записывают значение общей переменной без выполнения *синхронизации* или если как минимум одна нить читает значение общей переменной и как минимум одна нить записывает значение этой переменной без выполнения *синхронизации*, то возникает ситуация так называемой «*гонки данных*» (*datarace*), при которой результат выполнения программы непредсказуем.

По умолчанию все переменные, порождённые вне параллельной области, при входе в эту область остаются общими (*shared*). Переменные, объявленные внутри параллельной области, по умолчанию являются локальными (*private*).

Не рекомендуется постоянно полагаться на правила по умолчанию, для большей надёжности лучше всегда явно описывать классы используемых переменных, указывая в директивах OpenMP опции *private* и *shared*.

Следующее описание параллельной области:

#pragma omp parallel private(n)

объявляет переменную *n* как локальную. Это значит, что каждая нить будет работать со своей копией переменной *n*, при этом в начале параллельной области на каждой нити переменная *n* не будет инициализирована.

Общая переменная используются в параллельном регионе в том случае, если не предполагается изменение ее значения. Часто общими объявляют массивы, например:

int m[10];

#pragma omp parallel shared(m)

При этом потоки должны осуществлять изменение отличающихся элементов массива, не допуская одновременную запись значений в один и тот же элемент из различных потоков.

## Операция редукции

Особым видом модели памяти, совмещающим в себе возможности локального и общего классов, является опция редукции. Эта опция имеет следующий синтаксис:

#pragma parallel reduction(operator: var1   
[, var2,…, varN])

Поле *operator* содержит одну из арифметических операций (+, -, \*, /), для которой будут вычисляться частичные значения в параллельных потоках. Кроме того, определяется список общих переменных *var1, var2, …, varN*, для каждой из которых создаются локальные копии в каждой нити, и в которых будут сохраняться соответствующие частичные значения. Локальные копии инициализируются соответственно типу оператора (для аддитивных операций – 0, для мультипликативных операций – 1). После завершения всех параллельных процессов частичные значения складываются (вычитаются, перемножаются и т. п.), и результат сохраняется в одноименной общей переменной.

# Задания

## Задание 1

Разработать программу, определяющую время многократного выполнения операции умножения двух вещественных чисел, введенных пользователем (количество итераций задавать константой). Вывести время решения задачи при однопоточных циклических многократных вычислениях и при распределении итераций по двум нитям параллельной области.

*Листинг 1*

|  |
| --- |
| int main(int argc, char\* argv[]) {  setlocale(LC\_ALL, "ru");  double ch1, ch2;  cout << "Введите 2 числа: ";  cin >> ch1 >> ch2;  const int kolvo = 1000000000;  double start = omp\_get\_wtime();  #pragma omp parallel for  for (int i = 0; i < kolvo; i++) {  double ch = ch1 \* ch2; // избыточное умножение  }  double end = omp\_get\_wtime();  printf("Затраченное время: %lf\n", end - start);  return 0;  } |

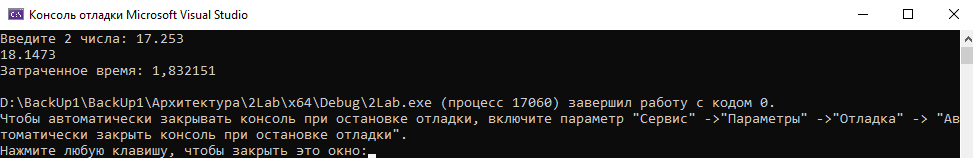


Рис. 1. Результат выполнения

*Листинг 2*

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <clocale>  using namespace std;  int main() {  setlocale(LC\_ALL, "ru");  double num1, num2, result;  cout << "Введите 2 числа: ";  cin >> num1 >> num2;  const int iterations = 1000000000;  double start, end;  #pragma omp parallel num\_threads(6)  {  double localNum1 = num1;  double localNum2 = num2;  #pragma omp master  start = omp\_get\_wtime();  #pragma omp for  for (int i = 0; i < iterations; i++)  result = localNum1 \* localNum2;  #pragma omp master  end = omp\_get\_wtime();  }  printf("Затраченное время: %lf\n", end - start);  return 0;  } |

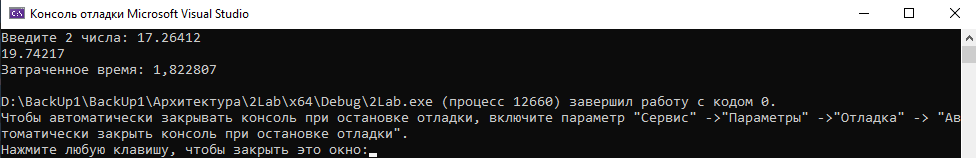


Рис. 2.Результат выполнения

## Задание 2

Разработать программу, определяющую время выполнения служебных операций по организации параллельной области, не содержащей операций.

*Листинг 3*

|  |
| --- |
| #include <iostream>  #include <omp.h>  int main() {  setlocale(LC\_ALL, "ru");  double startTime = omp\_get\_wtime();  #pragma omp parallel  {    }  double endTime = omp\_get\_wtime();  printf("Время выполнения пустой параллельной области: %f секунд\n", endTime - startTime);  return 0;  } |

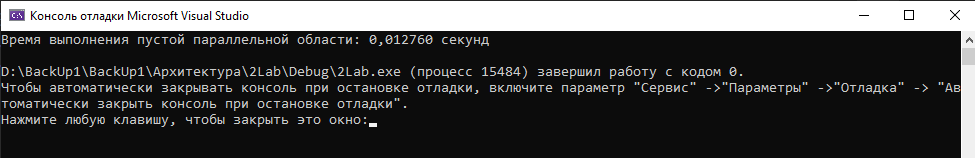


Рис. 3.Результат выполнения

## Задание 3

Организовать трехпоточное исполнение параллельной области. Вывести в параллельной области текстовую строку «Начало» и номер выполняемой нити, затем один из потоков выводит на экран строку «Одна нить» с указанием своего номера, завершается параллельная область выводом строки «Окончание» с указанием номера нити. Проанализировать порядок вывода сообщений без и с использованием опции nowait.

*Листинг 4*

|  |
| --- |
| #include <iostream>  #include <omp.h>  int main() {  setlocale(LC\_ALL, "ru");  std::cout << "Основная область: Начало" << std::endl;  #pragma omp parallel num\_threads(3)  {  int threadNum = omp\_get\_thread\_num();  std::cout << "Параллельная область: Начало нити " << threadNum << std::endl;  #pragma omp critical  {    if (threadNum == 1) {  std::cout << "Одна нить: Нить " << threadNum << std::endl;  }  }  #pragma omp barrier  #pragma omp critical  {  std::cout << "Параллельная область: Окончание нити " << threadNum << std::endl;  }  }  std::cout << "Основная область: Окончание" << std::endl;  return 0;  } |

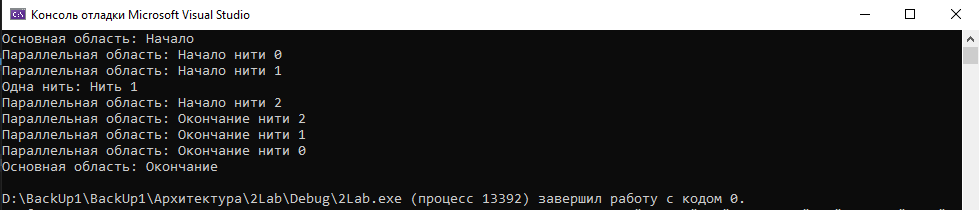


Рис. 4.Результат выполнения

## Задание 4

Разработать программу, содержащую параллельную область с тремя нитями выполнения. Каждый поток в параллельной области выводит на экран сообщение «Начало», «Середина» и «Конец». Между выводами сообщений главный поток должен дополнительно вывести на экран текстовую строку «Главный поток». Каждый вывод текстовой строки в параллельной области сопроводить выводом номера потока.

*Листинг 5*

#include <omp.h>

#include <iostream>

#include <clocale>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int ThreadId;

omp\_set\_num\_threads(3);

#pragma omp parallel private(ThreadId)

{

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

cout << "Начало №" << ThreadId + 1 << endl;

#pragma omp barrier

#pragma omp master

{

cout << "Главный поток №" << ThreadId + 1 << endl;

cout << "Главный поток №" << ThreadId + 1 << endl;

}

#pragma omp barrier

cout << "Середина №" << ThreadId + 1 << endl;

#pragma omp barrier

#pragma omp master

{

cout << "Главный поток №" << ThreadId + 1 << endl;

cout << "Главный поток №" << ThreadId + 1 << endl;

}

#pragma omp for

for (int i = 0; i < 3; i++)

cout << "Конец №" << ThreadId + 1 << endl;

}

return 0;

}

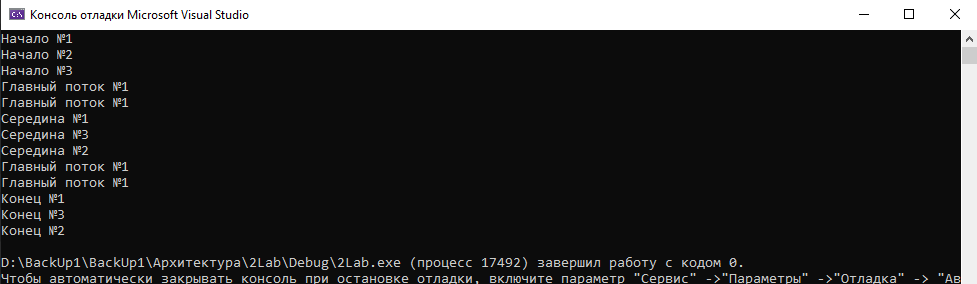


Рис. 5.Результат выполнения

## Задание 5

Разработать программу, содержащую двухпоточную параллельную область и демонстрирующую особенности работы с приватными переменными. В последовательной области объявить, инициализировать значением 10 и вывести на экран значение переменной *n*. В начале параллельной области все нити должны вывести на экран значение своей копии переменной *n*, которая должна быть объявлена приватной (возможно возникновение предупреждения при запуске программы). Затем необходимо присвоить переменной порядковый номер нити и вновь вывести значение переменной на экран. После завершения параллельной области значение переменной *n* еще раз выводится на экран. Каждый вывод значения переменной сопроводить текстом «Последовательная область» или «Параллельная область» и указанием номера потока.

*Листинг 6*

#include <omp.h>

#include <clocale>

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int ThreadId, n = 10;

#pragma omp parallel private(ThreadId,n) num\_threads(2)

{

n = 10;

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

cout << "Параллельная область №" << ThreadId + 1 << " n=" << n << endl;

#pragma omp barrier

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

cout << "Параллельная область №" << ThreadId + 1 << " n=" << n << endl;

#pragma omp barrier

}

cout << "Последовательная область n=" << n << endl;

return 0;

}

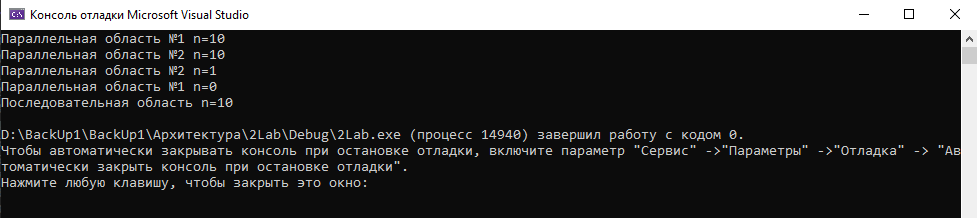


Рис. 6.Результат выполнения

## Задание 6

Разработать программу, содержащую объявление массива *m* в пять элементов и параллельной области, для которой массив указан общим. Параллельная область исполняется в два потока. Массив *m* инициализируется нулями и выводится на печать. В параллельной области каждая нить находит элемент, номер которого совпадает с порядковым номером нити в общем массиве, и присваивает этому элементу значение 1. Далее, в последовательной области печатается изменённый массив *m*.

*Листинг 7*

|  |
| --- |
| #include <iostream>  #include <omp.h>  int main() {  setlocale(LC\_ALL, "ru");  const int size = 5;  int m[size] = { 0 };  std::cout << "Последовательная область: Исходный массив m: ";  for (int i = 0; i < size; ++i) {  std::cout << m[i] << " ";  }  std::cout << std::endl;  #pragma omp parallel num\_threads(2) shared(m)  {  int threadNum = omp\_get\_thread\_num();    m[threadNum] = 1;  #pragma omp barrier  #pragma omp master  {  std::cout << "Параллельная область: Измененный массив m: ";  for (int i = 0; i < size; ++i) {  std::cout << m[i] << " ";  }  std::cout << std::endl;  }  } // Конец параллельной области  return 0;  } |

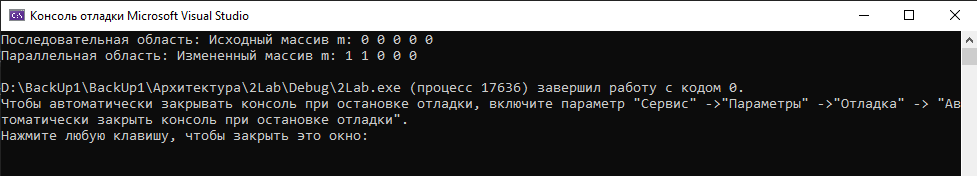


Рис. 7.Результат выполнения

## Задание 7

Разработать программу, подсчитывающую с использованием операции редукции общее количество нитей в параллельной области. В параллельной области задать редуцируемой переменной значение 1 и вывести текущее значение переменной и номера потока. Вывести значение редуцируемой переменной в последовательной области перед завершением программы.

*Листинг 8*

|  |
| --- |
| #include <iostream>  #include <omp.h>  int main() {  setlocale(LC\_ALL, "ru");  int totalThreads = 0;  #pragma omp parallel reduction(+:totalThreads)  {  int threadNum = omp\_get\_thread\_num();  // Редуцируемая переменная увеличивается на 1 для каждой нити  totalThreads += 1;  std::cout << "Номер потока: " << threadNum << std::endl;  } // Конец параллельной области  // После завершения параллельной области выводим общее количество нитей  std::cout << "Количество потоков: " << totalThreads << std::endl;  return 0;  } |

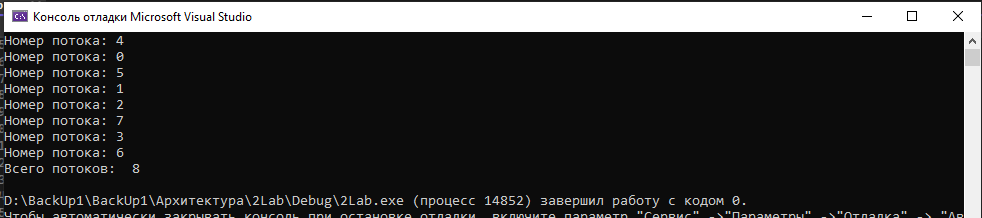


Рис. 8.Результат выполнения

## Задание 8

Используя операцию редукции в параллельной области найти сумму номеров четырех потоков. Каждый поток осуществляет вывод на экран своего номера и значения редуцируемой переменной. Вывести значение редуцируемой переменной перед завершением программы.

*Листинг 9*

|  |
| --- |
| #include <iostream>  #include <omp.h>  int main() {  setlocale(LC\_ALL, "ru");  int sumOfThreadNums = 0;  #pragma omp parallel reduction(+:sumOfThreadNums) num\_threads(4)  {  int threadNum = omp\_get\_thread\_num();  // Редуцируемая переменная увеличивается на номер текущей нити  sumOfThreadNums += threadNum;  std::cout << "Параллельная область: Нить " << threadNum << ", сумма номеров = " << sumOfThreadNums << std::endl;  } // Конец параллельной области  std::cout << "Последовательная область: Сумма номеров четырех потоков = " << sumOfThreadNums << std::endl;  return 0;  } |

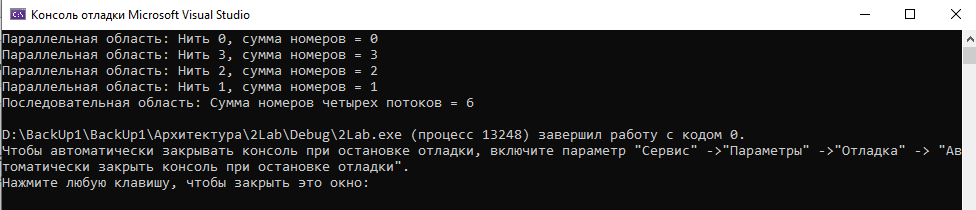


Рис. 9.Результат выполнения

# Контрольные вопросы

1. **Какая функция библиотеки OpenMP предназначена для получения текущего момента времени?**

double omp\_get\_wtime(void);

1. **Каким образом используя функции работы с системным таймером можно определить время выполнения некоторого участка в программе?**

Перед началом выполнения определенного участка кода сохраните текущее время в переменной типа double с помощью функции omp\_get\_wtime(). После выполнения участка кода снова используйте функцию omp\_get\_wtime() и сохраните текущее время в другой переменной double. Затем вычислите разность между двумя сохраненными значениями, чтобы получить

1. **Чем отличаются директивы распределения работы между потоками single и master?**

single используется, когда необходимо, чтобы в параллельном регионе какой-то код выполнился 1 раз ЛЮБЫМ потоком, также она имеет неявную синхронизацию потоков: остальные потоки ждут, но можно и отключить данную синхронизацию, указав опцию nowait после директивы single.

master используется тогда, когда необходимо, чтобы какой-то участок кода выполнил главный поток остальные потоки просто пропускают эту директиву и приступают выполнять идущий за ней код.

1. **Может ли нить-мастер выполнить область, ассоциированную с директивой single?**

Да, может.

1. **Может ли нить с номером 1 выполнить область, ассоциированную с директивой single?**

Да, может.

1. **Может ли одна и та же переменная выступать в одной части программы как общая, а в другой – как локальная?**

Да, может.

1. **Что произойдет, если несколько нитей одновременно обратятся к общей переменной?**

Возникает ситуация, известная как "гонка данных" (data race), где результат выполнения программы становится непредсказуемым из-за конфликта при доступе к данным, когда несколько потоков пытаются изменить общие данные одновременно.

1. **Может ли произойти конфликт, если несколько нитей одновременно обратятся к одной и той же локальной переменной?**

Нет, т.к. у каждой нити свой экземпляр локальной переменной и нить не может обратиться не к своей локальной переменной.

1. **Какую роль играет опция nowait в директиве single? Поясните смысл барьерной синхронизации.**

nowait указывается если нужно, чтобы остальные потоки не ждали тот поток, который выполняет регион single.

1. **Каким образом узнать к какой модели памяти относится объявленная переменная, если это не специфицировано явно в программе при объявлении параллельного региона?**

Если переменная определена вне параллельной области, то она автоматически считается общей (shared). В случае, когда переменная объявлена внутри параллельной области, её рассматривают как локальную (private)..

1. **Каким образом избежать «гонки данных» при использовании общих массивов в параллельном регионе?**

Нужно разработать параллельный участок кода так, чтобы не произошла ситуация, при которой несколько потоков могут записывать и читать элемент с одним и тем же индексом.

1. **Какой тип модели памяти используется при объявлении переменной в списке опции редукции?**

private

1. **Каким образом библиотека OpenMP организует работу с редуцируемыми переменными?**

Переменные в списке редукции становятся локальными, после каждая нить записывает в такую переменную что, после выполнения параллельного региона эти локальные переменные «собираются » в одну одноименную переменную путём \* / + - этих локальных переменных.

1. **Какой способ можно предложить для получения доступа к вычисленным частным значениям общей редуцируемой переменной после завершения параллельного региона?**

Создать переменную перед параллельным блоком с опцией редукции, затем после operator: указать её. После выполнения параллельного региона все частные вычисления будут соединены в одну путём их \* / + -