Лабораторная работа №2

По Архитектурам вычислительных систем

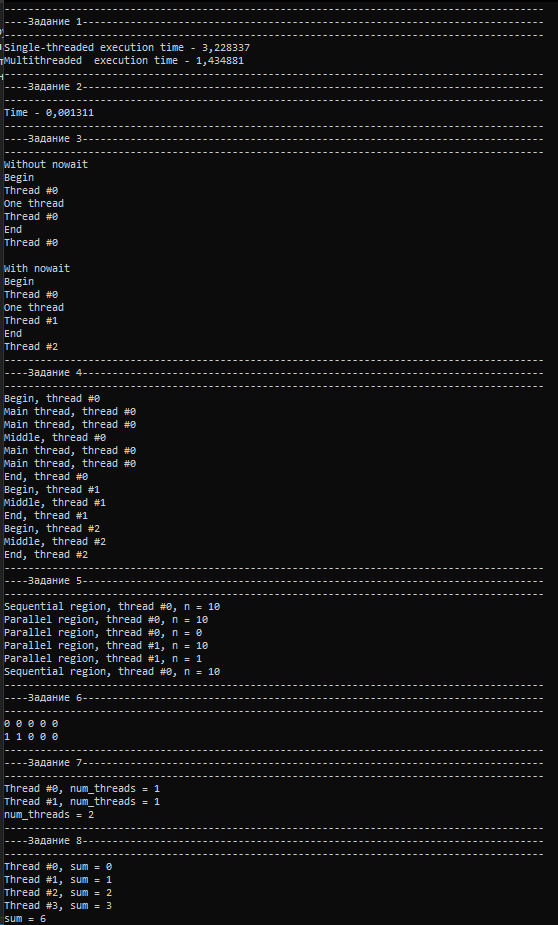
Выполнил: Лебедев В.В., 18-ПРИ

Задания:

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

Исходный код

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <time.h>  #include <math.h>  #include <string>  #include <stdio.h>  using namespace std;  #define COUNT\_MLTPLCTNS 10000000  int main()  {  setlocale(LC\_ALL, "Russian");  //------------------------------------------------------------------------------------------  //---------Задание 1------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 1-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  const int numThreads = 2;  omp\_set\_num\_threads(numThreads);  const int N = 100000000000;  double time = omp\_get\_wtime();  for (int i = 0; i < N; i++) {  double result = 10 \* 10;  }  printf("Single-threaded execution time - %lf\n", omp\_get\_wtime() - time);  time = omp\_get\_wtime();  #pragma omp parallel  {  for (int i = 0; i < N / numThreads; i++) {  double result = 10 \* 10;  }  }  printf("Multithreaded execution time - %lf\n", omp\_get\_wtime() - time);  //------------------------------------------------------------------------------------------  //---------Задание 2------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 2-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");    omp\_set\_dynamic(1);  double time1 = omp\_get\_wtime();  #pragma omp parallel  {  }  double time2 = omp\_get\_wtime();  printf("Time - %lf\n", time2 - time1);  //------------------------------------------------------------------------------------------  //---------Задание 3------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 3-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");    printf("Without nowait\n");  #pragma omp parallel num\_threads(3)  {  #pragma omp single  {  printf("Begin\nThread #%d\n", omp\_get\_thread\_num());  }  #pragma omp single  {  printf("One thread\nThread #%d\n", omp\_get\_thread\_num());  }  #pragma omp single  {  printf("End\nThread #%d\n\n", omp\_get\_thread\_num());  }  }  printf("With nowait\n");  #pragma omp parallel num\_threads(3)  {  #pragma omp single nowait  {  printf("Begin\nThread #%d\n", omp\_get\_thread\_num());  }  #pragma omp single nowait  {  printf("One thread\nThread #%d\n", omp\_get\_thread\_num());  }  #pragma omp single nowait  {  printf("End\nThread #%d\n", omp\_get\_thread\_num());  }  }  //------------------------------------------------------------------------------------------  //---------Задание 4------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 4-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  #pragma omp parallel num\_threads(3)  {  printf("Begin, thread #%d\n", omp\_get\_thread\_num());  #pragma omp master  {  printf("Main thread, thread #%d\n", omp\_get\_thread\_num());  printf("Main thread, thread #%d\n", omp\_get\_thread\_num());  }  printf("Middle, thread #%d\n", omp\_get\_thread\_num());  #pragma omp master  {  printf("Main thread, thread #%d\n", omp\_get\_thread\_num());  printf("Main thread, thread #%d\n", omp\_get\_thread\_num());  }  printf("End, thread #%d\n", omp\_get\_thread\_num());  }  //------------------------------------------------------------------------------------------  //---------Задание 5------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 5-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  int n = 10;  int ThreadsNum = omp\_get\_thread\_num();  printf("Sequential region, thread #%d, n = %d\n", omp\_get\_thread\_num(), n);  #pragma omp parallel num\_threads(2) private(n)  {  n = 10;  printf("Parallel region, thread #%d, n = %d\n", omp\_get\_thread\_num(), n);  n = omp\_get\_thread\_num();  printf("Parallel region, thread #%d, n = %d\n", omp\_get\_thread\_num(), n);  }  printf("Sequential region, thread #%d, n = %d\n", omp\_get\_thread\_num(), n);  //------------------------------------------------------------------------------------------  //---------Задание 6------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 6-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  int m[5] = { 0,0,0,0,0 };  for (int i = 0; i < 5; i++) {  printf("%d ", m[i]);  }  printf("\n");  #pragma omp parallel num\_threads(2) shared(m)  {  for (int i = 0; i < 5; i++) {  if (omp\_get\_thread\_num() == i) {  m[i] = 1;  }  }  }  for (int i = 0; i < 5; i++) {  printf("%d ", m[i]);  }  printf("\n");  //------------------------------------------------------------------------------------------  //---------Задание 7------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 7-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  int num\_threads = 0;  #pragma omp parallel reduction(+:num\_threads)  {  num\_threads++;  printf("Thread #%d, num\_threads = %d\n", omp\_get\_thread\_num(), num\_threads);  }  printf("num\_threads = %d\n", num\_threads);  //------------------------------------------------------------------------------------------  //---------Задание 8------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 8-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  int sum = 0;  #pragma omp parallel num\_threads(4) reduction(+:sum)  {  sum += omp\_get\_thread\_num();  printf("Thread #%d, sum = %d\n", omp\_get\_thread\_num(), sum);  }  printf("sum = %d\n", sum);  //------------------------------------------------------------------------------------------  } |

Пример выполнения программы

Ответы на 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?

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

double omp\_get\_wtime(void);

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

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

master – участок кода выполняется только нитью-мастером, single – участок кода выполняется первой свободной нитью

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

Да

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

Да

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

Да

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

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.

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

Нет

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

*Nowait* позволяет не ждать выполнения участка *single*

Барьерная синхронизация — метод синхронизации в распределённых вычислениях, при котором выполнение параллельного алгоритма или его части можно разделить на несколько этапов, разделённых барьерами.

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

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).

1. Каким образом избежать «гонки данных» при использовании общих массивов в параллельной области?

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

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

Кроме того, определяется список общих переменных, для каждой из которых создаются локальные копии в каждой нити, и в которых будут сохраняться соответствующие частичные значения

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

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.

1. Какой способ можно предложить для получения доступа к вычисленным частным значениям общей редуцируемой переменной после завершения параллельной области?

После завершения всех параллельных процессов частичные значения складываются (вычитаются, перемножаются и т. п.), и результат сохраняется в одноименной общей переменной.