Лабораторная работа №3

По Архитектурам вычислительных систем

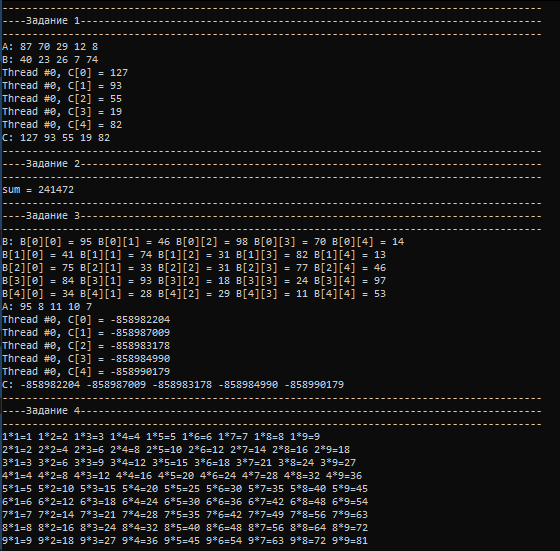
Выполнил: Лебедев В.В., 18-ПРИ

Задания:

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.
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.
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.
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*.

Исходный код

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <time.h>  #include <math.h>  #include <string>  #include <stdio.h>  using namespace std;  void print\_array(int arr[], int n) {  for (int i = 0; i < n; i++) {  printf("%d ", arr[i]);  }  printf("\n");  }  int main()  {  setlocale(LC\_ALL, "Russian");  srand(time(NULL));  //------------------------------------------------------------------------------------------  //---------Задание 1------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 1-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  const int N = 5;  int A[N];  int B[N];  int C[N];  for (int i = 0; i < N; i++) {  A[i] = rand() % 100;  B[i] = rand() % 100;  }  printf("A: ");  print\_array(A, N);  printf("B: ");  print\_array(B, N);  int i;  #pragma omp parallel for shared(C) private(i)  for (i = 0; i < N; i++) {  C[i] = A[i] + B[i];  printf("Thread #%d, C[%d] = %d\n", omp\_get\_thread\_num(), i, C[i]);  }  printf("C: ");  print\_array(C, N);  //------------------------------------------------------------------------------------------  //---------Задание 2------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 2-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  const int N1 = 100;  int A1[N1];  int B1[N1];  int sum = 0;  for (int i = 0; i < N1; i++) {  A1[i] = rand() % 100;  B1[i] = rand() % 100;  }  #pragma omp parallel for private(i) reduction(+:sum)  for (int i = 0; i < N1; i++) {  sum += A1[i] \* B1[i];  }  printf("sum = %d\n", sum);  //------------------------------------------------------------------------------------------  //---------Задание 3------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 3-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  int B3[N][N];  int A3[N];  int C3[N];  for (int i = 0; i < N; i++) {  A3[i] = rand() % 100;  for (int j = 0; j < N; j++) {  B3[i][j] = rand() % 100;  }  }  printf("B: ");  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++) {  printf("B[%d][%d] = %d ", i, j, B3[i][j]);  }  printf("\n");  }  printf("A: ");  print\_array(A3, N);  int i1;  int j;  #pragma omp parallel for shared(C) private(i1,j)  for (j = 0; j < N; j++) {  for (i1 = 0; i1 < N; i1++) {  C3[j] += A3[i1] \* B3[i1][j];  }  printf("Thread #%d, C[%d] = %d\n", omp\_get\_thread\_num(), j, C3[j]);  }  printf("C: ");  print\_array(C3, N);  //------------------------------------------------------------------------------------------  //---------Задание 4------------------------------------------------------------------------  //------------------------------------------------------------------------------------------  printf("------------------------------------------------------------------------------------------\n");  printf("----Задание 4-----------------------------------------------------------------------------\n");  printf("------------------------------------------------------------------------------------------\n");  int i3;  #pragma omp num\_theads(9) parallel for ordered private(i3)  for (i3 = 1; i3 <= 9; i3++) {  for (int j = 1; j <= 9; j++) {  printf("%d\*%d=%d ", i3, j, i3 \* j);  }  printf("\n");  }  //------------------------------------------------------------------------------------------  } |

Пример выполнения программы 

Ответы на вопросы

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

Если область определена как parallel for, тогда итерации распределятся между нитями а если просто parallel, то каждый поток пройдет все итерации.

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

#pragma omp for [опция[[,] опция]...]

for(..;..;..)

{

//Тело распараллеливаемого цикла

}

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

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

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

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

Каждый поток должен обращаться только к своей части массива.

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

#include <omp.h>

#include <stdio.h>

#include <conio.h>

int main(int argc, char \*argv[])

{

int A[6], i;

#pragma omp parallel shared(A) private(i)

{

#pragma omp for

for (i=1; i<=6; i++) {

A[i-1]=i\*i;

}

}

for(i=1;i<=6;i++) {

printf("%d^2=%d\n", i, A[i-1]);

}

getch();

return 0;

}

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

Да, при использовании директивы ordered

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

#pragma omp ordered

#pragma omp single

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

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.

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

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

for(..;..;..)

{

//Тело распараллеливаемого цикла

}

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

Нет, иначе цикл выполнится некорректно из-за эффекта гонок

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

Для каждой переменной создаются локальные копии в каждой нити; над локальными копиями переменных после завершения всех итераций цикла выполняется заданный оператор;

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

Если задавать динамически: количество потоков = количество итераций цикла или количество потоков = максимальное количество ядер процессора.

Если задавать вручную: количество потоков = количество итераций цикла, при условии, что заданное число потоков = количество итераций.

Если заданное число потоков < количество итераций, то какие-то потоки выполнят несколько итераций.

Если заданное число потоков > количество итераций, то какие-то потоки будут простаивать.