МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №5 по дисциплине «Параллельное программирование»

**Технология OpenMP**

|  |  |
| --- | --- |
| Выполнил: студент группы ФИб-4302-51-00 | / Д.А. Савин / |
| Проверил: ст. преподаватель каф. ПМиИ | / В.А. Бызов / |

Киров 2021

Задание 0

Проверить, поддерживается ли OpenMP. Если нет, то включить поддержку в

параметрах проекта.

Код:

|  |
| --- |
| #include "pch.h"  #include <iostream>  int main()  {  #ifdef \_OPENMP  printf("OpenMP is supported! %d\n", \_OPENMP);  #else  printf("OpenMP is not supported!\n");  #endif  } |

Изображение выглядит как текст, оранжевый, снимок экрана

Автоматически созданное описание

Задание 1

Скопировать фрагмент кода в программу. Запустить на выполнение.

Ответить на вопросы:

а) Зачем нужна директива parallel?

Чтобы код параллельной области дублировался между параллельно исполняемыми потоками.

б) Сколько потоков было запущено? Почему?

12, потому что процессор имеет 6 ядер 12 потоков.

в) Сколько потоков одновременно работают с переменной tid? Почему?

12

г) Поток с каким tid останется после завершения параллельной области?

0

Запросить требуемое количество потоков у пользователя. Задать количество

потоков для параллельной области.

Код:

|  |
| --- |
| #include "pch.h"  #include <iostream>  #include <omp.h>  using namespace std;  int main() {  int nthreads, tid;  int num\_threads = 0;  int normal\_num\_threads = omp\_get\_num\_procs();  cout << "--> ";  cin >> num\_threads;  if (num\_threads > normal\_num\_threads)  {  cout << "Optimal number of threads: " << normal\_num\_threads << ", but u want: " << num\_threads << endl;  }  omp\_set\_num\_threads(num\_threads);  // Создание параллельной области  #pragma omp parallel private(tid)  {  // печать номера потока  tid = omp\_get\_thread\_num();  printf("Hello World from thread = %d\n", tid);  // Печать количества потоков – только master  if (tid == 0) {  nthreads = omp\_get\_num\_threads();  printf("Number of threads = %d\n", nthreads);  }    } // Завершение параллельной области  } |

Задание 2

Написать программу, задающую работу двух потоков. Первый поток в цикле

выводит последовательно числа от 1 до N, а второй – N раз выводит слово

«HELLO». Число N задаётся пользователем.

Код:

|  |
| --- |
| #include "pch.h"  #include <iostream>  #include <omp.h>  using namespace std;  int main()  {  int N = 0;  cout << "--> ";  cin >> N;  omp\_set\_num\_threads(2);  #pragma omp parallel  {  // Код внутри блока выполняется параллельно  #pragma omp sections  {  #pragma omp section  {  #pragma omp parallel num\_threads(1)  for (int i=0; i<N; i++)  cout << i << " ";  }  #pragma omp section  {  #pragma omp parallel num\_threads(1)  for (int i = 0; i < N; i++)  cout << "Hello\n";  }  }  }  } |

Задание 3

Написать параллельную программу, находящую поэлементное произведение

двух массивов размера N. Задать параметр schedule, попробовать разные

аргументы.

Код:

|  |
| --- |
| #include "pch.h"  #include <iostream>  #include <omp.h>  using namespace std;  int main()  {  double t1, t2, dt;      int size = 10000;  cout << "--> ";  cin >> size;  omp\_set\_num\_threads(2);  int \*arr\_1 = new int[size];  int \*arr\_2 = new int[size];  int \*arr\_3 = new int[size];  for (int i = 0; i < size; i++)  {  arr\_1[i] = rand() % 25;  arr\_2[i] = rand() % 25;  }  t1 = omp\_get\_wtime();  #pragma omp parallel  {  #pragma omp for schedule(dynamic, 1000)  for (int i = 0; i < size; ++i)  {  arr\_3[i] = arr\_1[i] \* arr\_2[i];  }    }  t2 = omp\_get\_wtime();  dt = t2 - t1;  cout << "time: " << dt << endl;  } |

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр | Размер массива | размер\_порции | Время |
| static | 100000000 | 1000 | 0.125646 |
| dynamic | 100000000 | 1000 | 0.107433 |
| guided | 100000000 | 1000 | 0.0687756 |
| runtime | 100000000 |  | 0.0694202 |
| auto | 100000000 |  | ошибка |

Задание 4

Написать параллельную программу, вычисляющую максимальное значение

среди элементов вектора

а) с использованием параметра reduction;

б) с использованием директивы critical;

в) с использованием атомарных операций.

В чем отличия между этими способами?

Код:

|  |
| --- |
| #include "pch.h"  #include <iostream>  #include <omp.h>  #include <vector>  #include <stdlib.h>  #include <cmath>  using namespace std;  int main()  {  int size = 1000; int maxx = -1; double t1, t2, dt;  vector <int> array\_;  for (int i = 0; i < size; i++)  {  array\_.push\_back(rand() % 25);  //cout << array\_.at(i) << " ";  }  t1 = omp\_get\_wtime();  #pragma omp parallel reduction(max: maxx)  {  for (int i = 0; i < size; i++)  {  //if (array\_.at(i) > maxx)  {  //maxx = array\_.at(i);  maxx = max(array\_.at(i), maxx);  }  }  }  t2 = omp\_get\_wtime();  dt = t2 - t1;  cout << "time reduction: " << dt << endl;  for (int i = 0; i < size; i++)  {  array\_.push\_back(rand() % 25);  //cout << array\_.at(i) << " ";  }  t1 = omp\_get\_wtime();  #pragma omp parallel  {  for (int i = 0; i < size; i++)  {  #pragma omp critical  {  maxx = max(array\_.at(i), maxx);  }  }  }  t2 = omp\_get\_wtime();  dt = t2 - t1;  cout << "time critical: " << dt << endl;  t1 = omp\_get\_wtime();  #pragma omp atomic  {  for (int i = 0; i < size; i++)  {  maxx = max(array\_.at(i), maxx);  }  }  t2 = omp\_get\_wtime();  dt = t2 - t1;  cout << "time atomic: " << dt << endl;  } |

|  |  |  |
| --- | --- | --- |
| Параметр | Размер массива | Время |
| reduction | 100000 | Не доступно в версии 2.0 |
| critical | 100000 | 0.820124 |
| atomic | 100000 | ошибка |

Задание 5

Написать программу, задающую работу M + K потоков. Первые M потоков вычисляют сумму от 1 до N, а оставшиеся K потоков вычисляют длину N-мерного вектора. Число N задаётся пользователем.

Указание. Использовать секции и вложенный параллелизм.

Код:

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <vector>  using namespace std;  int main()  {  vector <int> vector;  int M, K, N = 0;  cout << "Input num of threads (M) >> ";  while (!(cin >> M) || (cin.peek() != '\n'))  {  cin.clear();  while (cin.get() != '\n');  cout << "Input error! \nRetry input >> ";  }  cout << "Input num of threads (K) >> ";  while (!(cin >> K) || (cin.peek() != '\n'))  {  cin.clear();  while (cin.get() != '\n');  cout << "Input error! \nRetry input >> ";  }  cout << "Input (N) >> ";  while (!(cin >> N) || (cin.peek() != '\n'))  {  cin.clear();  while (cin.get() != '\n');  cout << "Input error! \nRetry input >> ";  }  for (int i = 0; i < N; i++)  {  vector.push\_back(rand() % 25);  }  int $ = M + K;  omp\_set\_nested(1);  int sum = 0;  double sum2 = 0;  double t1, t2, dt;  t1 = omp\_get\_wtime();  #pragma omp parallel num\_threads($)  {  #pragma omp sections  {  #pragma omp section  {  #pragma omp parallel for num\_threads(M)    //int sum = 0;  for (int i = 0; i < N; i++)  {  sum += i;  }    }  #pragma omp section  {  #pragma omp parallel for num\_threads(K)      for (int j = 0; j < N; j++)  {  sum2 += pow(vector.at(j), 2);  }  sum2 = sqrt(sum2);    }  }  }  t2 = omp\_get\_wtime();  dt = t2 - t1;  cout << "time: " << dt << endl;  //cout <<sum << " " <<sum2 << endl;  } |

Задание 6

Написать параллельную программу, которая каждый элемент вектора размера N заменяет на его наибольший простой делитель. Число N задается пользователем. Элементы вектора – случайные натуральные числа из диапазона [105, 106]. Замерить среднее время выполнения программы для N = 2·107, 5·107 и 108 на 1, 2, 4 и 8 потоках. Вычислить среднее ускорение для 2, 4 и 8 потоков. Построить диаграмму зависимости ускорения от числа потоков для каждого размера вектора (3 графика на одной диаграмме).

Код:

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <vector>  using namespace std;  int main()  {  vector <int> vector;  int M, K, N = 0;  cout << "Input num of threads (M) >> ";  while (!(cin >> M) || (cin.peek() != '\n'))  {  cin.clear();  while (cin.get() != '\n');  cout << "Input error! \nRetry input >> ";  }  cout << "Input num of threads (K) >> ";  while (!(cin >> K) || (cin.peek() != '\n'))  {  cin.clear();  while (cin.get() != '\n');  cout << "Input error! \nRetry input >> ";  }  cout << "Input (N) >> ";  while (!(cin >> N) || (cin.peek() != '\n'))  {  cin.clear();  while (cin.get() != '\n');  cout << "Input error! \nRetry input >> ";  }  for (int i = 0; i < N; i++)  {  vector.push\_back(rand() % 25);  }  int $ = M + K;  omp\_set\_nested(1);  int sum = 0;  double sum2 = 0;  double t1, t2, dt;  t1 = omp\_get\_wtime();  #pragma omp parallel num\_threads($)  {  #pragma omp sections  {  #pragma omp section  {  #pragma omp parallel for num\_threads(M)    //int sum = 0;  for (int i = 0; i < N; i++)  {  sum += i;  }    }  #pragma omp section  {  #pragma omp parallel for num\_threads(K)      for (int j = 0; j < N; j++)  {  sum2 += pow(vector.at(j), 2);  }  sum2 = sqrt(sum2);    }  }  }  t2 = omp\_get\_wtime();  dt = t2 - t1;  cout << "time: " << dt << endl;  //cout <<sum << " " <<sum2 << endl;  } |

Таблица Время выполнения:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Количество потоков | | |
| Размер | 2 | 4 | 8 |
| 2\*10^4 | 2,18108 | 1,33058 | 0,835142 |
| 2\*10^5 | 21,4476 | 11,7245 | 8,03467 |
| 2\*10^6 | 212,426 | 116,735 | 76,6997 |

Таблица Ускорение:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 2 | 4 | 8 |
| 2\*10^7 | 1,918018 | 3,144005 | 5,009148 |
| 5\*10^7 | 1,947738 | 3,562992 | 5,199255 |
| 10^8 | 1,967608 | 3,580511 | 5,449448 |

Задание 7

Написать параллельную программу, выполняющую умножение двух матриц размера N \* N. Разработать программы с использованием распараллеливания циклов разного уровня вложенности. Замерить среднее время выполнения программ для N = 500, 1000, 2000 на 1, 2, 4 и 8 потоках. Сравнить полученные результаты. Оцените величину накладных расходов на создание и завершение потоков. Для оптимального варианта вычислить среднее ускорение на 2, 4 и 8 потоках. Построить диаграмму зависимости ускорения от числа потоков для каждого размера матриц (3 графика на одной диаграмме).

Код:

|  |
| --- |
| // 7.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.  //  #include <iostream>  #include <omp.h>  #define SIZE 500  #define NUM\_THREAD 1  using namespace std;  int\*\* matrix\_1 = new int\*[SIZE];  int\*\* matrix\_2 = new int\*[SIZE];  int\*\* result\_matrix = new int\* [SIZE];  void fillMatrixes()  {  cout << "FILLING.." << endl;  int max\_threads = omp\_get\_max\_threads();  cout << "FILLING USE " << max\_threads << " THREADS." << endl;  #pragma omp parallel for num\_threads(max\_threads)  for (int i = 0; i < SIZE; i++)  {  matrix\_1[i] = new int[SIZE];  matrix\_2[i] = new int[SIZE];  result\_matrix[i] = new int[SIZE];  for (int j = 0; j < SIZE; j++)  {  matrix\_1[i][j] = rand() % 25;  matrix\_2[i][j] = rand() % 25;  result\_matrix[i][j] = 0;  }  }    cout << "FILLING ENDED.." << endl << endl;  }  int main()  {  #ifdef \_OPENMP  printf("OpenMP is supported! %d\n", \_OPENMP);  #else  printf("OpenMP is not supported!\n");  #endif  fillMatrixes();  double t1 = omp\_get\_wtime();  #pragma omp parallel for num\_threads(NUM\_THREAD)  for (int i = 0; i < SIZE; i++)  {  for (int j = 0; j < SIZE; j++)  {  for (int k = 0; k < SIZE; k++)  {  result\_matrix[i][j] += matrix\_1[i][k] \* matrix\_2[k][j];  }  }  }  double t2 = omp\_get\_wtime();  double dt = t2 - t1;  cout << "time: " << dt << endl;  for (int i = 0; i < SIZE; i++)  {  delete[] matrix\_1[i];  delete[] matrix\_2[i];  delete[] result\_matrix[i];  }  delete[] matrix\_1;  delete[] matrix\_2;  delete[] result\_matrix;  } |

Таблица Время выполнения:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Количество потоков | | |
| Размер | 2 | 4 | 8 |
| 500 | 0,101789 | 0,0561813 | 0,0340044 |
| 1000 | 0,813979 | 0,494299 | 0,37222 |
| 2000 | 13,2118 | 7,44981 | 5,20365 |

Таблица Ускорение:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 2 | 4 | 8 |
| 500 | 1,9632 | 3,557 | 5,8768 |
| 1000 | 1,9873 | 3,2725 | 4,3458 |
| 2000 | 1,8558 | 3,2911 | 4,7118 |

Задание 8

Написать параллельную программу, выполняющую поиск максимального значения среди минимальных элементов строк матрицы размера N \* N. Обосновать выбор средств и методов для распараллеливания. Замерить среднее время выполнения программ для N = 500, 1000, 2000 на 1, 2, 4 и 8 потоках. Вычислить среднее ускорение на 2, 4 и 8 потоках.

Код:

|  |
| --- |
| #include <iostream>  #include <omp.h>  #define SIZE 1000  #define NUM\_THREAD 2  using namespace std;  int\*\* matrix\_1 = new int\* [SIZE];  int\* result\_matrix = new int[SIZE];  void fillMatrixes()  {  cout << "FILLING.." << endl;  int max\_threads = omp\_get\_max\_threads();  cout << "FILLING USE " << max\_threads << " THREADS." << endl;  #pragma omp parallel for num\_threads(max\_threads)  for (int i = 0; i < SIZE; i++)  {  matrix\_1[i] = new int[SIZE];  // matrix\_2[i] = new int[SIZE];  // result\_matrix[i] = new int[SIZE];  for (int j = 0; j < SIZE; j++)  {  matrix\_1[i][j] = rand() % 25;  // matrix\_2[i][j] = rand() % 25;  // result\_matrix[i][j] = 0;  }  }  cout << "FILLING ENDED.." << endl << endl;  }  void findMin()  {  int max = -1;  #pragma omp parallel for num\_threads(NUM\_THREAD)  for (int i = 0; i < SIZE; i++)  {  for (int j = 0; j < SIZE; j++)  {  //#pragma omp critical  if (matrix\_1[i][j] > max)  {  max = matrix\_1[i][j];  }  }  result\_matrix[i] = max;  max = -1;  }    max = -1;  #pragma omp parallel for num\_threads(NUM\_THREAD)  for (int i = 0; i < SIZE; i++)  {  //#pragma omp critical  if (result\_matrix[i] > max)  {  max = result\_matrix[i];  }  }    }  int main()  {  #ifdef \_OPENMP  printf("OpenMP is supported! %d\n", \_OPENMP);  #else  printf("OpenMP is not supported!\n");  #endif  fillMatrixes();  double t1 = omp\_get\_wtime();  findMin();  double t2 = omp\_get\_wtime();  double dt = t2 - t1;  cout << "time: " << dt << endl;  for (int i = 0; i < SIZE; i++)  {  delete[] matrix\_1[i];    }  delete[] matrix\_1;  delete[] result\_matrix;  } |

Таблица Время выполнения:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Количество потоков | | |
| Размер | 2 | 4 | 8 |
| 500 | 0,0004837 | 0,0004076 | 0,0005253 |
| 1000 | 0,000657 | 0,0005049 | 0,0006629 |
| 2000 | 0,0018729 | 0,0014173 | 0,0011422 |

Таблица Ускорение:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 2 | 4 | 8 |
| 500 | 5,209427 | 6,182041 | 4,796878 |
| 1000 | 14,95084 | 19,45474 | 14,81777 |
| 2000 | 1,692509 | 2,236577 | 2,775258 |

Вывод

При выполнении лабораторной работы были получены навыки разработки с использованием OpenMP. Распараллеливание с использованием данной технологии позволяет получить в некоторых задачах ускорение в 4-6 раз. Кроме того, использование reduction может сильно упростить процесс разработки и получить аналогичную производительность.