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 №2

по дисциплине «Параллельное программирование»

**Параллельные алгоритмы в языке C++**

Выполнил: студент группы ФИб-4302-51-00 / А.С. Глызин /

Проверил: ст. преподаватель каф. ПМиИ / В.А. Бызов /

Киров 2021

# *Задание на лабораторную работу*

1. Используя параллельные алгоритмы стандартной библиотеки языка C++, написать программу, вычисляющую для каждого элемента целочисленного вектора количество его делителей. Элементы вектора – случайные натуральные числа из диапазона [105, 106]. Результат должен быть записан в новый вектор.

Провести тестирование программы на векторах размера 5 ∙ 105, 106, 2 ∙ 106 с различными политиками выполнения. На каждом примере запустить не менее трех раз. В таблицу занести среднее время выполнения на одном примере в секундах.

Вывод: политика выполнения seq уступает другим политикам по эффективности более чем в пять раз при вычислении количества делителей.

*Таблица 1 – Время вычисления количества делителей элементов целочисленного вектора, с*

|  |  |  |  |
| --- | --- | --- | --- |
| **Размер вектора** | **Политика выполнения** | | |
| seq | par | par\_unseq |
| 𝟓 ∙ 𝟏𝟎𝟓 | 1.52263 | 0.282219 | 0.284206 |
| 𝟏𝟎𝟔 | 3.07808 | 0.57263 | 0.645494 |
| 𝟐 ∙ 𝟏𝟎𝟔 | 6.3319 | 1.16051 | 1.14213 |

1. Используя параллельные алгоритмы стандартной библиотеки языка C++, написать программу, суммирующую элементы вектора.

*Указание*. Использовать алгоритм std::reduce.

Провести тестирование программы на векторах размера 5 ∙ 108, 109, 2 ∙ 109 с различными политиками выполнения. На каждом примере запустить не менее трех раз. В таблицу занести среднее время выполнения на одном примере в секундах.

Вывод: для вычисления суммы элементов целочисленного вектора наиболее эффективной оказалась политика выполнения seq.

*Таблица 2 – Время вычисления сумм элементов целочисленного вектора, с*

|  |  |  |  |
| --- | --- | --- | --- |
| **Размер вектора** | **Политика выполнения** | | |
| seq | par | par\_unseq |
| 𝟓 ∙ 𝟏𝟎𝟓 | 0.948924 | 1.93935 | 3.86412 |
| 𝟏𝟎𝟔 | 0.182263 | 0.397372 | 30.4618 |
| 𝟐 ∙ 𝟏𝟎𝟔 | 0.214226 | 0.428125 | 4.27539 |

1. Используя параллельные алгоритмы стандартной библиотеки языка C++, реализовать классический алгоритм умножения матриц.

*Указание*. Алгоритмы использовать для вычисления скалярного произведения.

Провести тестирование программы на матрицах размера 512, 1024, 2048 с различными политиками выполнения. На каждом примере запустить не менее трех раз. В таблицу занести среднее время выполнения на одном примере в секундах.

Вывод: даже при использовании одинарных указателей время работы программы в разы больше без использования политики выполнения. При умножения матриц большую эффективность показала политика выполнения par\_unseq, практически наравне с par.

*Таблица 3 – Время вычисления произведений матриц, состоящих из целочисленных векторов, с*

|  |  |  |  |
| --- | --- | --- | --- |
| **Размер вектора** | **Политика выполнения** | | |
| seq | par | par\_unseq |
| 512 | 0.184178 | 0.0363166 | 0.0379735 |
| 1024 | 1.81756 | 0.348695 | 0.323909 |
| 2048 | 42.4008 | 6.95519 | 6.89509 |

*Таблица 4 – Время выполнения алгоритма умножения матриц*

*с одинарными указателями, с*

|  |  |  |  |
| --- | --- | --- | --- |
| **Размер вектора** | **Ключ оптимизации** | | |
| **-O0** | **-O2** |
| 512 | 0.475825 | 0.251344 |
| 1024 | 5.49238 | 2.06528 |
| 2048 | 162.479 | 73.3633 |

1. Используя параллельные алгоритмы стандартной библиотеки языка C++, написать программу, сортирующую массив действительных чисел.

Протестировать на массивах разной размерности (3-5 вариантов) со всеми политиками выполнения. Замерить время выполнения. Размеры массивов подобрать таким образом, чтобы нагрузить систему.

Вывод: для задачи сортировки массива, состоящего из действительных чисел, больше подходит политика выполнения par\_unseq.

*Таблица 5 – Время сортировки массивов действительных чисел, с*

|  |  |  |  |
| --- | --- | --- | --- |
| **Размер вектора** | **Политика выполнения** | | |
| seq | par | par\_unseq |
| 𝟏𝟎7 | 0.698154 | 0.217818 | 0.195941 |
| 𝟏𝟎8 | 6.75061 | 2.07004 | 1.92189 |
| 𝟐 ∙ 𝟏𝟎8 | 13.5871 | 4.0299 | 3.9573 |
| 𝟏𝟎9 | 69.1827 | 25.2823 | 23.6294 |

Вывод: выбор в пользу той или иной политики выполнения должен быть сделан исходя из поставленной задачи.

Исходный код программ

|  |
| --- |
| #include <iostream>  #include <vector>  #include <execution>  #define SIZE 5 \* pow(10, 6)  using namespace std;  int GetDividers(int number)  {  int cout = 0;  for (int i = 1; i < sqrt(number) + 1; i++)  {  if (number % i == 0)  {  cout++;  if (i != (number / i) && number % (number / i) == 0)  {  cout++;  }  }  }  return cout;  }  int main()  {  vector<int> in;  vector<int> out(SIZE);  const int MIN\_VALUE = pow(10, 5);  const int MAX\_VALUE = pow(10, 6);  for (int i = 0; i < SIZE; i++)  {  in.insert(  in.begin(),  (rand() % (MIN\_VALUE + rand() % MAX\_VALUE))  );  }  std::chrono::time\_point<std::chrono::high\_resolution\_clock> start,  end;  start = std::chrono::high\_resolution\_clock::now();  std::transform(std::execution::par\_unseq, // seq, par, par\_unseq  in.begin(), in.end(),  out.begin(),  GetDividers);  end = std::chrono::high\_resolution\_clock::now();  std::chrono::duration<double> diff = end - start;  cout << "time: " << diff.count() << endl;  } |

|  |
| --- |
| #include <iostream>  #include <vector>  #include <execution>  #define SIZE 2 \* pow(10, 9)  using namespace std;  int main()  {  vector<int> in;  const int MIN\_VALUE = pow(10, 5);  const int MAX\_VALUE = pow(10, 6);  for (int i = 0; i < SIZE; i++)  {  in.push\_back((rand() % (MIN\_VALUE + rand() % MAX\_VALUE)));  }  std::chrono::time\_point<std::chrono::high\_resolution\_clock> start,  end;  start = std::chrono::high\_resolution\_clock::now();  reduce(std::execution::par\_unseq, in.cbegin(), in.cend());  end = std::chrono::high\_resolution\_clock::now();  std::chrono::duration<double> diff = end - start;  cout << "time: " << diff.count() << endl;  } |

|  |
| --- |
| #include <iostream>  #include <vector>  #include <execution>  #define SIZE 2048  #define MAX\_VALUE 10  #define TEST\_COUNT 3  using namespace std;  int main()  {  double time = 0;  for (int i = 0; i < TEST\_COUNT; i++)  {  vector<vector<int>> matrix1(SIZE);  for (int i = 0; i < SIZE; i++)  {  matrix1[i] = vector<int>(SIZE);  for (int j = 0; j < SIZE; j++)  {  matrix1[i][j] = rand() % (MAX\_VALUE + 1);  }  }  vector<vector<int>> matrix2(SIZE);  for (int i = 0; i < SIZE; i++)  {  matrix2[i] = vector<int>(SIZE);  for (int j = 0; j < SIZE; j++)  {  matrix2[i][j] = rand() % (MAX\_VALUE + 1);  }  }  vector<vector<int>> out(SIZE);  std::chrono::time\_point<std::chrono::high\_resolution\_clock> start,  end;  start = std::chrono::high\_resolution\_clock::now();  std::transform(std::execution::par\_unseq, matrix1.begin(), matrix1.end(), out.begin(),  [matrix2](vector<int> row) -> vector<int> {  vector<int> result;  for (int i = 0; i < matrix2[0].size(); i++) {  int rowcolsum = 0;  for (int k = 0; k < row.size(); k++) {  rowcolsum += row[k] \* matrix2[k][i];  }  result.push\_back(rowcolsum);  }  return result;  });  end = std::chrono::high\_resolution\_clock::now();  std::chrono::duration<double> diff = end - start;  time += diff.count();  cout << "time: " << diff.count() << endl;  //for (int i = 0; i < SIZE; i++)  //{  // for (int j = 0; j < SIZE; j++)  // {  // cout << out[i][j] << " ";  // }  // cout << endl;  //}  }  cout << "avg time: " << time / TEST\_COUNT << endl;  } |

|  |
| --- |
| #include <iostream>  #include <vector>  #include <execution>  #define MAX\_VALUE 10  using namespace std;  int main()  {  vector<int> sizes { (int)pow(10,7), (int)pow(10,8), 2 \* (int)pow(10,8), (int)pow(10,9)};  for (int i = 0; i < sizes.size(); i++)  {  vector<double> vector;  for (int j = 0; j < sizes[i]; j++)  {  vector.push\_back(  0 + ((double)rand() / RAND\_MAX) \* (MAX\_VALUE - 0)  );  //vector[j] = rand() % (MAX\_VALUE + 1);  }  std::chrono::time\_point<std::chrono::high\_resolution\_clock> start, end;  start = std::chrono::high\_resolution\_clock::now();  sort(std::execution::par, vector.begin(), vector.end());  end = std::chrono::high\_resolution\_clock::now();  std::chrono::duration<double> diff = end - start;  cout << diff.count() << " " << endl;  }  } |