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 | / Д.А. Савин / |
| Проверил: ст. преподаватель каф. ПМиИ | / В.А. Бызов / |

Киров 2021

Задание 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. Сравнить результаты. Сделать выводы.

Код:

|  |
| --- |
| #include <iostream>  #include <vector>  #include <ctime>  #include <execution>  #include <algorithm>  using namespace std;  const int dimm = 10000\*5;  vector <int> v1 = {};  vector <int> res = {};  chrono::time\_point<chrono::high\_resolution\_clock>start, ennd;  int getCountDev(int x)  {  int count = 1;  for (int i = 1; i <= x / 2; i++)  {  if (x % i == 0)  {  count++;  }  }  return count;  }  void display(vector <int> v)  {  for (int i = 0; i < v.size(); i++)  {  cout << v.at(i) << " ";  }  cout << endl;  }  void init()  {  srand(time(0));  for (int i = 0; i < dimm; i++)  {  v1.push\_back(rand() \* 5);  }  res.resize(v1.size());  }  int main()  {  init();  //display(v1);  start = chrono::high\_resolution\_clock::now();  transform(std::execution::par, v1.begin(), v1.end(), res.begin(), getCountDev);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time: " << diff.count() << endl;  //display(res);  } |

Таблица 1 – Время вычисления количества делителей

элементов целочисленного вектора, с

|  |  |  |  |
| --- | --- | --- | --- |
| Размер матрицы | Политика выполнения | | |
| seq | par | par\_unseq |
| 5\*105 | 7,72047 | 3,23782 | 3,30778 |
| 106 | 14,0302 | 7,01601 | 7,21814 |
| 2\*106 | 26,889 | 13,4826 | 13,8652 |

Вывод: Самое быстрое выполнение программы оказалось с политикой выполнения par.

Задание 2

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. Сравнить результаты. Сделать выводы.

Код:

|  |
| --- |
| #include <iostream>  #include <vector>  #include <ctime>  #include <execution>  #include <algorithm>  #include <chrono>  using namespace std;  const int dimm = 10\*100000000;  vector <long long> v1 = {};  //vector <int> res = {};  chrono::time\_point<chrono::high\_resolution\_clock>start, ennd;  void display(vector <long long> v)  {  for (long long i = 0; i < v.size(); i++)  {  cout << v.at(i) << " ";  }  cout << endl;  }  void init()  {  srand(time(0));  for (long long i = 0; i < dimm; i++)  {  v1.push\_back(rand() % 25);  }  //res.resize(v1.size());  }  int main()  {  init();  long long result = 0;  //display(v1);  start = chrono::high\_resolution\_clock::now();  reduce(std::execution::par, v1.cbegin(), v1.cend());  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  //cout << "result: " << result << endl;  cout << "time: " << diff.count() << endl;  //display(res);  } |

Таблица 2 – Время вычисления суммирования

элементов целочисленного вектора, с

|  |  |  |  |
| --- | --- | --- | --- |
| Размер матрицы | Политика выполнения | | |
| seq | par | par\_unseq |
| 5\*108 | 3,44933 | 0,607282 | 1.05086 |
| 109 | 40,042 | 15,7305 | 18,0193 |
| 2\*109 | Ошибка(не хватает памяти) | Ошибка(не хватает памяти) | Ошибка(не хватает памяти) |

Вывод: Самое быстрое выполнение программы оказалось с политикой выполнения par.

Задание 3

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. Сравнить результаты с результатами лабораторной работы 1. Сделать выводы.

Код:

|  |
| --- |
| asd |

Таблица 3 – Время выполнения классического алгоритма умножения матриц

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер матрицы | Порядок циклов | | | | | |
| ijk | ikj | jik | jki | kij | kji |
| 512 | 0,267317333 | 0,086795433 | 0,251307 | 0,697008667 | 0,1160648 | 0,740291 |
| 1024 | 17,6773 | 1,13226 | 13,15626667 | 33,02856667 | 1,422153333 | 34,10643333 |
| 2048 | 114,243 | 8,329586667 | 128,4096667 | 275,944 | 9,138473333 | 317,607 |

Вывод: Наилучшую асимптотику имеет порядок циклов ikj.

Задание 4

Выполнить задания 1-2 с использованием представления матриц в виде одномерных динамических массивов.

Код:

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <chrono>  using namespace std;  const int DIMM = 2048;  int\* A1 = new int[DIMM \* DIMM];  int\* B1 = new int[DIMM \* DIMM];  int\* C1 = new int[DIMM \* DIMM];  chrono::time\_point<chrono::high\_resolution\_clock>start, ennd;  void init2()  {  srand(time(0));  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  A1[DIMM \* i + j] = rand() % 25;  B1[DIMM \* i + j] = rand() % 25;  C1[DIMM \* i + j] = 0;  }  }  }  void display1(int\* A)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  cout << A[DIMM \* i + j] << " ";  }  cout << "" << endl;  }  cout << "" << endl;  }  void matr1(int\* A, int\* B)  {  for (int k = 0; k < DIMM; k++)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  C1[DIMM \* i + j] += A[DIMM \* i + k] \* B[DIMM \* k + j];  }  }  }  }  int main()  {  init2();  //display1(A1);  //display1(B1);  start = chrono::high\_resolution\_clock::now();  matr1(A1, B1);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time: " << diff.count() << endl;  //cout << " res = " << endl;  //display1(C1);  delete[]A1;  delete[]B1;  delete[]C1;  } |

Таблица 4 – Время выполнения алгоритма умножения матриц с одинарными указателями

|  |  |  |
| --- | --- | --- |
| Размер матрицы | Ключ оптимизации | |
| -O0 | -O2 |
| 512 | 0,711614 | 0,203333333 |
| 1024 | 4,142327 | 1,27 |
| 2048 | 30,67953 | 6,516666667 |

Вывод: Приложение с ключом оптимизации -О2 быстрее примерно в 3,83 раза.