Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе

По дисциплине «Параллельные вычисления»

«Рекурсивный фильтр для последовательности чисел с настраиваемой длиной импульсной реакции»

Работу выполнили студенты группы №13541/3 Карандашов С.М.\_\_\_\_\_\_\_\_\_

Работу принял преподаватель Стручков И. В.\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2018

Цель работы

Получить навык создания многопоточных программ.

Постановка задачи

Задача: построить рекурсивный фильтр для последовательности чисел с настраиваемой длиной импульсной реакции.

Решить следующую задачу тремя способами:

1. однопоточное приложение;
2. многопоточное приложение с использованием pthread;
3. с использованием библиотеки OpenMP.

Характеристики ПК

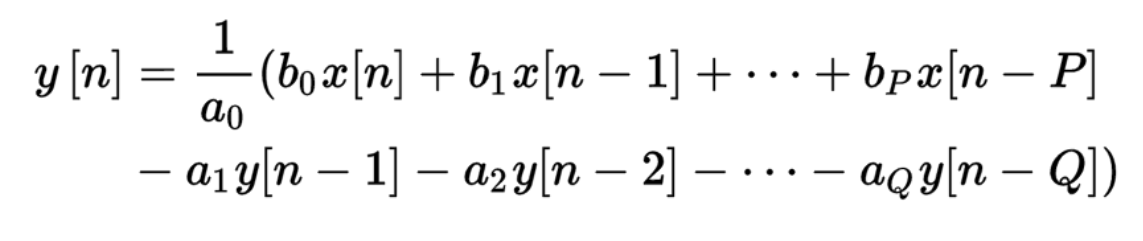
Процессор Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz, 2,5 ГГц, ядер: 4, логических процессоров: 8 (*hyper-threading*)

Обзор задачи

**Рекурсивный фильтр**

Алгоритм работы фильтра состоит из следующих этапов: фильтр получает значения, с помощью коэффициентов преобразует их и выдаёт выходную последовательность.

Формула, описывающая работу фильтра, выглядит следующим образом:



Коэффициенты рассчитываются с помощью функции баттерворта. Для рассчета коэффициентов был найден следующий матлаб скрипт:

|  |
| --- |
| fs = 48000; % sampling frequency (Hz) |
|  |

|  |
| --- |
| wp = 5000; % passband frequency |
|  |

|  |
| --- |
| ws = 7000; % stopband frequency |
|  |

|  |
| --- |
| n = 6; % filter order = 2\*n = 12 |
|  |

|  |
| --- |
| w1 = (2 \* wp)/fs; |
|  |

|  |
| --- |
| w2 = (2 \* ws)/fs; |
|  |

|  |
| --- |
| wn = [w1, w2]; |
|  |

[b, a] = butter(n, wn, 'stop');

Однопоточная задача

Написанная однопоточная программа представлена в листинге 1.

Листинг 1. Однопоточная программа

|  |
| --- |
| public static void main(String[] args) {  double[] in = generateIn();  double out[] = new double[size];  double x[] = new double[N]; // here x[i] represents input[n - i]  double y[] = new double[N]; // here y[i] represents output[n - i]  int a[] = new int[N]; // here x[i] represents input[n - i]  int b[] = new int[N]; // here y[i] represents output[n - i]  for (int i = 0; i < N; i++)  {  x[i] = 0;  y[i] = 0;  }    final Random random = new Random();  int q = 40;    for (int i = 0; i < N; i++)  {  a[i] = random.nextInt() \* q;  b[i] = random.nextInt() \* q;  }  //Фильтрация входных данных    long res = 0;// = System.nanoTime();  long start = System.nanoTime();  for (int i = 0; i < in.length; i++) {  out[i] = 0.;  out[i] += b[0] \* in[i];    for (int j = 1; j < N; j++)  {  out[i] += b[j] \* x[j];  out[i] -= a[j] \* y[j];  }    for (int j = N-1; j > 1; j--) { x[j] = x[j-1]; }  for (int j = N-1; j > 1; j--) { y[j] = y[j-1]; }  x[1] = in[i];  y[1] = out[i];  //System.out.println(out);  }  long end = System.nanoTime();  res = res + end-start;      System.out.println("time: " + res/1000000000.0);  } |

Многопоточная программы с использованием библиотеки pthread

Алгоритм распараллеливания реализован следующим образом: каждому потоку на вычисление отдается по возможности равное элементов массива.

Листинг 2. Многопоточная программа

|  |
| --- |
| public static void main(String[] args) throws ExecutionException, InterruptedException {  double[] in = generateIn();  double out[] = new double[size];    for (int i = 0; i < N; i++)  {  x[i] = 0;  y[i] = 0;  }    final Random random = new Random();  int q = 40;    for (int i = 0; i < N; i++)  {  a[i] = random.nextInt() \* q;  b[i] = random.nextInt() \* q;  }  //Фильтрация входных данных    List<Future<double[]>> workerFutures = new ArrayList<>();  ExecutorService executor = Executors.newFixedThreadPool(4);  long start = System.nanoTime();  int nn = size/4;  for (int j = 0; j < size; j = j + size / 4) {  double[] workerIn = new double[nn];  System.arraycopy(in, j, workerIn, 0, nn);  Worker worker = new Worker(workerIn);  Future<double[]> future = executor.submit(worker);  workerFutures.add(future);  }  for (Future<double[]> workerFuture : workerFutures) {  workerFuture.get();  }  System.out.println("time: " + ((System.nanoTime() - start))/1000000000.0);  } |

Многопоточная программа с использованием OpenMP

Код многопоточной программы с использованием OpenMP представлен в листинге 3.

Листинг 3. Многопоточная программы с использованием OpenMP.

|  |
| --- |
| #include <math.h>  #include <omp.h>  #include <ctime>  #include <iostream>  using namespace std;  const double M\_PI = 3.141592653589793238462643383279;  void main()  {  const int sizeIn = 80;  long double \*in;  long double \*out;  in = new long double[sizeIn];  out = new long double[sizeIn];  for (int i = 0; i < sizeIn; i++) {  in[i] = rand() % 10;  }  const int N = 1000000;  long int \*x;  long int \*y;  long int \*a;  long int \*b;  x = new long int[N]; // here x[i] represents input[n - i]  y = new long int[N]; // here y[i] represents output[n - i]  a = new long int[N]; // here x[i] represents input[n - i]  b = new long int[N]; // here y[i] represents output[n - i]  for (int i = 0; i < N; i++)  {  x[i] = 0;  y[i] = 0;  }      int q = 40;    for (int i = 0; i < N; i++)  {  a[i] = rand() \* q;  b[i] = rand() \* q;  }    omp\_set\_num\_threads(8);  int step = sizeIn / 8;    int i,j;  int res = 0;// = System.nanoTime();  unsigned int start = clock();  #pragma omp parallel for schedule(dynamic, step) private(j)  for (i = 0; i < sizeIn; i++) {  out[i] = 0.;  out[i] += b[0] \* in[i];    for (int j = 1; j < N; j++)  {  out[i] += b[j] \* x[j];  out[i] -= a[j] \* y[j];  }      for (int j = N-1; j > 1; j--) { x[j] = x[j-1]; }  for (int j = N-1; j > 1; j--) { y[j] = y[j-1]; }  x[1] = in[i];  y[1] = out[i];  //System.out.println(out);  }  unsigned int end = clock();  res = res + end-start;  cout << "time " << ((float)res) / CLOCKS\_PER\_SEC << "\n"; // искомое время  system("pause");  } |

Результаты вычислительных экспериментов

Сводная таблица результатов для всех программ при 80 элементах и 4 потоках:

Табл.1. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| Точность фильтра | Последовательный | Параллельный | |
| OpenMP | Pthreads |
| 100 | 5.66961E-4 | 0,001 | 0.00169 |
| 1000 | 0.0046 | 0,001 | 0.004899 |
| 10000 | 0.01685 | 0,006 | 0.02267 |
| 1000000 | 0.7888 | 0,423 | 0,458 |

Табл.2. Обработка результатов при точности фильтра 1000000, 80 элементах и переменном числе потоков:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во потоков | Последовательный | Параллельный | |
| OpenMP | Pthreads |
| 1 | 0.86862 | 1,227 | 0.82661 |
| 2 | 0.86862 | 0,699 | 0.56947 |
| 4 | 0.86862 | 0,421 | 0.46000 |
| 8 | 0.86862 | 0,343 | 0.47645 |

Табл.2. Обработка результатов при точности фильтра 100000, 4 потоках и зменяемым потоком данных:

|  |  |  |  |
| --- | --- | --- | --- |
| Данные | Последовательный | Параллельный | |
| OpenMP | Pthreads |
| 80 | 0.053652797 | 0,036 | 0.0685 |
| 1000 | 0.345282908 | 0,358 | 0.1584 |
| 10000 | 3.29441278 | 3,555 | 1.1994 |
| 100000 | 32.5140 | 37,264 | 11.503 |

Вывод

В данной работе были изучены основы создания многопоточных приложений. Бы­л изучен библиотека pthread и библиотека OpenMP. Созданные программы были протестированы на разных наборах данных и были оценены характеристики времени работы. На основе проведенных экспериментов можно сделать вывод, что наиболее эффектив­ным решением является параллельная обработка данных, но стоит заметить, что увеличение количества потоков не всегда ведет к возрастанию производительности.

Программу удалось реализовать полностью независимой по данным, поэтому в работе не возникло необходимости использования средств синхронизации.