Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Высшая школа искусственного интеллекта**

**Лабораторная работа 1**

**Знакомство с многопоточной обработкой**

по дисциплине «Параллельное программирование»

Выполнил

студент гр.3530203/80102 Л.Д. Челищева

<*подпись*>

Руководитель К.А. Туральчук

<*подпись*>

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

# Параметры процессора



# Задания

## Реализуйте последовательную обработку элементов вектора, например умножение элементов вектора на число. Число элементов вектора задается параметром N.

Созданная программа создает массив из n случайных чисел от 200 до 500, а затем каждое из чисел сохраняется в результирующий массив с возведением в 1.789 степень.

Согласно проведенным тестам (в таблицы представлены средние значения для 25 запусков), при выполнении последовательной обработки с небольшой погрешностью время работы прямо пропорционально количеству элементов и представлено в таблице.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | 10 | 1000 | 100000 | 10000000 |
| t | 0,02011 | 0,04001 | 4,320946 | 364,2829 |

## Реализуйте многопоточную обработку элементов вектора, используя разделение вектора на равное число элементов. Число потоков задается параметром M.

Программа была модифицирована, в код добавлена возможность запустить m потоков, при чем массив делится на m элементов по порядку и их высчитывают потоки практически параллельно. Код обработки включает в себя функцию Run1 и ее вызов для массива потоков.

## Выполните анализ эффективности многопоточной обработки при разных параметрах N (10, 100, 1000, 100000) и M (2, 3, 4, 5, 10). Результаты представьте в табличной форме.

На основе кода Run1 был проведен ряд тестов для всех возможных пар значений n и m, при чем время высчитывалось как среднее значение за 25 запусков, исключая первый. Результаты тестирования представлены в таблице.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| m n | 10 | 1000 | 100000 | 10000000 |
| 2 | 25,99772 | 27,01856 | 53,39854 | 133,4136 |
| 3 | 43,2525 | 37,67835 | 68,90723 | 147,8707 |
| 4 | 44,81233 | 45,31622 | 88,00857 | 157,0795 |
| 5 | 56,95289 | 58,23729 | 101,3239 | 165,5955 |
| 10 | 92,57786 | 101,3257 | 174,8089 | 427,7896 |

Согласно полученным данным, распараллеливание до 5 потоков дает повышение производительности в два и более раза, но только на значительных показателях (последний столбец). 10 потоков, а также все случаи сравнительно небольших значений n не показали преимущества перед последовательной обработкой, более того, значительно ей уступают.

Несмотря на наличие 4х физических ядер в системе, наилучшим количеством потоков для распараллеливания оказались 2 потока. В дальнейших тестах это количество было зафиксировано.

## Выполните анализ эффективности при усложнении обработки каждого элемента вектора.

Для усложнения обработки был определен метод Run2, содержащий K итераций цикла для каждого элемента. Увеличение k с сохранением m=2 и n=100000 дало предсказуемое равномерное возрастание необходимого для расчета времени. Результаты представлены в таблице.

|  |  |  |  |
| --- | --- | --- | --- |
| k | 10 | 25 | 50 |
| t | 31,7187 | 57,53645 | 84,26377 |

## Исследуйте эффективность разделения по диапазону при неравномерной вычислительной сложности обработки элементов вектора.

Для неравномерной вычислительной сложности был создан метод Run3, в котом количество итераций, необходимых для подсчета каждого элемента зависит от номера этого элемента. При двух потоках можно наблюдать резкое возрастание времени выполнения при увеличении числа элементов. Можно отметить, что при запуске программы в отладчике отслеживалось, что большая часть времени отводится на выполнение второго потока, в то время как первый его ожидает. Результаты тестирования представлены в таблице

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | 10 | 100 | 1000 | 10000 |
| t | 11,93515 | 14,90903 | 26,89244 | 615,0826 |

## Исследуйте эффективность параллелизма при круговом разделении элементов вектора. Сравните с эффективностью разделения по диапазону.

Код Run3 был модифицирован в код Run4, позволяющий распределять элементы между потоками более равномерно, используя круговой метод разделения. Подобное решение показало наибольшую эффективность при достаточно больших значениях n – уже при 10000 элементов время сократилось почти на 1/3. При отладке было заметно, что потоки занимают практически одинаковое время и ожидание завершения одного потока другим минимально.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | 10 | 100 | 1000 | 10000 |
| t | 11,42012 | 14,09585 | 25,18043 | 433,6989 |

Тест проведен при оптимальном значении потоков (2), однако код адаптируется под любое число потоков.

# Полный код программы

using System;

using System.Threading;

using System.Diagnostics;

namespace lab1

{

class Program

{

static int N = 100000;

static int M = 2;

static int K = 10;

static int[] a = new int[N];

static double[] b = new double[N];

static Thread[] threads = new Thread[M];

static void Main(string[] args)

{

DateTime dt;

TimeSpan ts;

RndArr();

double sum = 0;

/\*//Последовательное выполнение

M = 1;

Run1(0);

for (int i = 0; i < 25; i++)

{

dt = DateTime.Now;

Run1(0);

ts = DateTime.Now - dt;

sum += (double)ts.TotalMilliseconds;

}

Console.WriteLine("Среднее время на 25 замеров: {0} мс", sum / 50);

sum = 0;\*/

//Лишний замер

for (int i = 0; i < M; i++)

threads[i] = new Thread(Run2);

for (int i = 0; i < M; i++)

threads[i].Start(i);

for (int i = 0; i < M; i++)

threads[i].Join();

//Замеры

for (int i = 0; i < 20; i++)

{

for (int j = 0; j < M; j++)

threads[j] = new Thread(Run2);

dt = DateTime.Now;

for (int j = 0; j < M; j++)

threads[j].Start(j);

for (int j = 0; j < M; j++)

threads[j].Join();

ts = DateTime.Now - dt;

sum += (double)ts.TotalMilliseconds;

}

Console.WriteLine("Среднее время на 20 замеров: {0} мс", sum / 50);

Console.ReadKey();

}

//Многопоточный

public static void Run1(object number)

{

int num = (int)number;

int fin = (num + 1) \* N / M > a.Length ? a.Length : (num + 1) \* N / M;

for (int i = num \* N / M; i < fin; i++)

b[i] = (int)Math.Pow(a[i], 1.789);

}

//Усложнение обработки

public static void Run2(object number)

{

int num = (int)number;

int fin = (num + 1) \* N / M > a.Length ? a.Length : (num + 1) \* N / M;

for (int i = num \* N / M; i < fin; i++)

{

for (int j = 0; j < K; j++)

b[i] += (int)Math.Pow(a[i], 1.789);

}

}

//Неравномерный

public static void Run3(object number)

{

int num = (int)number;

int fin = (num + 1) \* N / M > a.Length ? a.Length : (num + 1) \* N / M;

for (int i = num \* N / M; i < fin; i++)

{

for (int j = 0; j < i; j++)

b[i] += (int)Math.Pow(a[i], 1.789);

}

}

//Круговой

public static void Run4(object number)

{

int num = (int)number;

for (int i = num; i < a.Length; i+=M)

{

for (int j = 0; j < i; j++)

b[i] += (int)Math.Pow(a[i], 1.789);

}

}

static void RndArr()

{

Random rnd = new Random();

for (int i = 0; i < a.Length; i++)

{

a[i] = rnd.Next(200, 500);

}

}

}

}

# Выводы

В ходе лабораторной работы было исследование влияние различных параметров многопоточной обработки на ее эффективность. В итоге можно отметить следующие выводы:

Многопоточная обработка показывает свое преимущество перед последовательной при большом количестве элементов (до определённой границы время, необходимое на разделение потоков перекрывает выигрыш от параллельного выполнения).

В исследуемой системе наибольшую эффективность показали 2 потока, несмотря на заявленные 4 ядра.

При несбалансированном распределении сложности вычислений круговая декомпозиция показывает преимущество перед разделением по диапазонам, что становится заметно на достаточно большом количестве элементов. Однако это однозначно эффективно только в рассмотренном случае – при равномерном увеличении сложности с ростом номера элемента.