Задание

Дана последовательность натуральных чисел {*a0…an-1*}. Создать многопоточное приложение для поиска суммы Σ*ai* , где *ai* – четные числа.

Теоретические положения

Каждый процесс может создавать несколько нитей. Каждая нить представляет собой независимо выполняющийся поток управления со своим счетчиком команд, регистровым контекстом и стеком.

Потоки представляют возможность проведения псевдопараллельных вычислений. Различные потоки имеют различный порядок выполнения, но при этом пользуются общей памятью. Действия потока задаются при его создании указанием его функции, созданный поток начинает выполнять команды этой функции и завершается, когда происходит возврат из функции.

***Параллельные вычисления*** – способ организации компьютерных вычислений, при котором программы разрабатываются, как набор взаимодействующих вычислительных процессов, работающих асинхронно и при этом одновременно.

***Параллельное программирование*** – это техника программирования, которая использует преимущества многоядерных или многопроцессорных компьютеров и является подмножеством более широкого понятия многопоточности (multithreading).

Многоядерные вычисления

Использование параллельного программирования становится наиболее необходимым, поскольку позволяет максимально эффективно использовать возможности многоядерных процессоров и многопроцессорных систем. По ряду причин, включая повышение потребления энергии и ограничения пропускной способности памяти, увеличивать тактовую частоту современных процессоров стало невозможно. Вместо этого производители процессоров стали увеличивать их производительность за счет размещения в одном чипе нескольких вычислительных ядер, не меняя или даже снижая тактовую частоту. Поэтому для увеличения скорости работы приложений теперь следует по-новому подходить к организации кода, а именно – оптимизировать программы под многоядерные системы.

Результат выполнения лабораторной работы

Для выполнения этой лабораторной работы использовался язык программирования Python. В ходе выполнения данной работы было определено, что при сложении 100 и 1000 элементов однопоточное сложение выгодно, т.к. время его выполнения меньше чем у четырех, шести и восьми потоков. Но как только кол-во элементов доходит до 10 000, то однопоточное сложение проигрывает по времени выполнения, а многопоточное выигрывает. Результаты работы представлены в таблице 1.

Результаты выполнения

Сравнительная таблица

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Количество потоков | | | |
| количество элементов | 1 | 4 | 8 | 16 |
| 100 | 0.012001752853393555 | 0.03365683555603027 | 0.05600309371948242 | \* |
| 1000 | 0.019999027252197266 | 0.023000717163085938 | 0.028001070022583008 | \* |
| 10000 | 0.03153848648071289 | 0.04685163497924805 | 0.05500483512878418 | \* |
| 100000 | 0.04427194595336914 | 0.05100202560424805 | 0.08500790596008301 | \* |
| 1000000 | 0.4056215286254883 | 0.44213151931762695 | 0.602036714553833 | \* |
| 10000000 | 3.507167339324951 | 3.9999234676361084 | 5.79999041557312 | \* |
| 100000000 | \* | \* | \* | \* |

Приложение

**from random import randint**

**from time import time**

**def calculate(sp, half):**

**sum = 0**

**global finish\_sum**

**finish\_sum = 0**

**if half == 0:**

**for i in range(round((len(sp)-1)/2)):**

**if i%2 == 1:**

**sum += sp[i]**

**else:**

**for i in range(round((len(sp)-1)/2),len(sp)):**

**if i%2 == 1:**

**sum += sp[i]**

**finish\_sum += sum**

**return sum**

**def make\_sp(n):**

**sp = []**

**for i in range(n):**

**sp.append(randint(0,100))**

**return sp**

**number\_elements = 100000000**

**sp = make\_sp(number\_elements)**

**start\_time = time()**

**calculate(sp, 0)**

**print("Sum = ", finish\_sum)**

**print("work time: ", time() - start\_time)**

**Многопоточное**

**from threading import Thread**

**from random import randint**

**from time import time**

**number\_elements = 100000**

**threads\_count = 8**

**finish\_sum = 0**

**def make\_start\_nums():**

**start\_numbers = [0]**

**for i in range(threads\_count - 1): start\_numbers.append(randint(start\_numbers[i],start\_numbers[i]+randint(round(number\_elements/(threads\_count\*2)),round(number\_elements/threads\_count))))**

**start\_numbers.append(number\_elements)**

**return start\_numbers**

**sn = make\_start\_nums()**

**def calculate(sp, part):**

**sum = 0**

**for i in range(sn[part], sn[part+1]):**

**sum += sp[i]**

**global finish\_sum**

**finish\_sum = 0**

**finish\_sum += sum**

**return sum**

**def make\_sp(n):**

**sp = []**

**for i in range(n):**

**sp.append(randint(0,100))**

**return sp**

**sp = make\_sp(number\_elements)**

**sp2 = sp**

**start\_time = time()**

**thread1 = Thread(target=calculate, args=(sp, 0))**

**thread2 = Thread(target=calculate, args=(sp, 1))**

**thread3 = Thread(target=calculate, args=(sp, 2))**

**thread4 = Thread(target=calculate, args=(sp, 3))**

**thread5 = Thread(target=calculate, args=(sp, 4))**

**thread6 = Thread(target=calculate, args=(sp, 5))**

**thread7 = Thread(target=calculate, args=(sp, 6))**

**thread8 = Thread(target=calculate, args=(sp, 7))**

**thread1.start()**

**thread2.start()**

**thread3.start()**

**thread4.start()**

**thread5.start()**

**thread6.start()**

**thread7.start()**

**thread8.start()**

**thread1.join()**

**thread2.join()**

**thread3.join()**

**thread4.join()**

**thread5.join()**

**thread6.join()**

**thread7.join()**

**thread8.join()**

**print("Sum of odd elements: ", finish\_sum)**

**print("Program work time: ", time() - start\_time)**