

Инженерная школа информационных технологий и робототехники (ИШИТР)

Направление подготовки Искусственный интеллект и машинное обучение

Отделение школы (НОЦ) 09.04.01 Информатика и вычислительная техника

**Лабораторная работа**

По дисциплине «Параллельные и высокопроизводительные вычисления»

|  |
| --- |
| **Тема работы** |
| Решение задач обработки данных с помощью вычислений на многопоточных CPU |

Студент

|  |  |  |  |
| --- | --- | --- | --- |
| **Группа** | **ФИО** | **Подпись** | **Дата** |
| 8ВМ13 | Е.М. Якунин |  | 03.03.2022 |

Принял

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Должность** | **ФИО** | **Ученая степень, звание** | **Подпись** | **Дата** |
| К.Т.Н. Доцент (ОИТ, ИШИТР) | С.В. Аксёнов |  |  |  |

Томск – 2022 г.

# Задание

1. Вычисления матрицы контраста для двух уровней (вариант 3).

Вход: Матрица М1 размерами 2D1×2S1, матрица M2 размерами 2D2×2S2. Где D1-G=D2 и S1-G=S2, где D1, D2, S1, S2, G-целые положительные числа.

Выход: Матрица M3 кроссмасштабного вычитания M1 и M2.

Для тестов требуется оценить работу программы нахождения кроссмасштабной разности для следующего случая:

1. D1=12, S1=12, D2=10, S2=10

Матрицы M1 и М2 заполнить случайными целыми числами из диапазона 0-100.

# Отчет

Лабораторная работа была реализована в программе «PyCharm» с использованием языка программирования «Python». Для выполнения данного задания были применены библиотеки *numpy*, *time* и *threading*. С помощью библиотеки «numpy» были созданы матрицы с заполнением случайными значениями от 0 до 100. При помощи библиотеки «time» мы засекали время выполнения задачи, а «threading» ― распараллелили вычисления.

Для начала проверим скорость выполнения поставленной задачи без применения распараллеливания.

Исполняемый код:

import numpy as np  
from time import time  
  
d1, s1, d2, s2 = 12, 12, 10, 10  
  
# Создаем матрицы  
M1 = np.random.randint(1, 100, (2\*\*d1, 2\*\*s1))  
M2 = np.random.randint(1, 100, (2\*\*d2, 2\*\*s2))  
M3 = np.zeros((2\*\*d1, 2\*\*s1))  
  
print("Matrix M1: \n", M1)  
print("Matrix M2: \n", M2)  
  
# Считаем отношение размера матриц  
dt = int(2\*\*d1/2\*\*d2)  
  
# количество шагов для второй матрицы  
attitude1 = int(2\*\*d1 / dt)  
  
t1 = time()  
  
for a in range(3):  
 for i in range(attitude1):  
 for j in range(attitude1):  
 M3[dt\*i:dt\*(i+1), dt\*j:dt\*(j+1)] = M1[dt\*i:dt\*(i+1), dt\*j:dt\*(j+1)] - M2[i, j]  
  
t2 = time()  
  
print("Matrix M3: \n", M3)  
print("Elapsed time: ", f'{t2 - t1}')

После выполнения данного кода получаем матрицы M1 и M2, а также посчитанную матрицу M3(рис. 1).

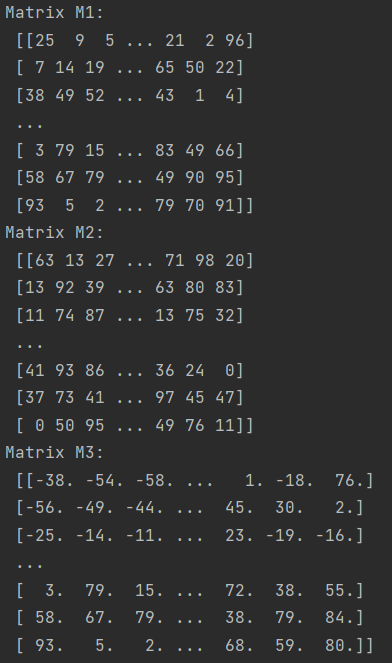


Рисунок 1 ‒ Результат выполненного кода

Проведем опыт 3 раза для фиксации времени выполнения (рис. 2).







Рисунок 2 ‒ Затраченное время на выполнение без использования многопоточности

Реализуем тот же самый опыт только с использованием многопоточности. Используем 2 потока. Проведем опыт 3 раза для фиксации времени исполнения.

Исполняемый код:

import threading  
import numpy as np  
from time import time

d1, s1, d2, s2 = 12, 12, 10, 10  
  
# Создаем матрицы  
M1 = np.random.randint(0, 100, (2\*\*d1, 2\*\*s1))  
M2 = np.random.randint(0, 100, (2\*\*d2, 2\*\*s2))  
M3 = np.zeros((2\*\*d1, 2\*\*s1))  
  
print("Matrix M1: \n", M1)  
print("Matrix M2: \n", M2)  
  
# Считаем отношение размера матриц  
dt = int(2\*\*d1/2\*\*d2)  
  
# количество шагов для второй матрицы  
attitude1 = int(2\*\*d1 / dt)  
  
# Создаем задачу для потоков  
def thread\_task(start1, finish1, start2, finish2, M3, M2, M1):  
 for i in range(start1, finish1):  
 for j in range(start2, finish2):  
 M3[dt\*i:dt\*(i+1), dt\*j:dt\*(j+1)] = M1[dt\*i:dt\*(i+1), dt\*j:dt\*(j+1)] - M2[i, j]  
  
# Исполняемая функция с потоками  
def main\_task():  
 # создаем потоки  
 t1 = threading.Thread(target=thread\_task, args=(0, attitude1/2, 0, attitude1, M3, M2, M1, ))  
 t2 = threading.Thread(target=thread\_task, args=(attitude1/2, attitude1, 0, attitude1, M3, M2, M1, ))  
  
 # начало работы потоков  
 t1.start()  
 t2.start()  
  
 # Ждем пока каждый поток завершит работу  
 t1.join()  
 t2.join()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start\_ap = time()  
  
 for i in range(3):  
 main\_task()  
  
 stop\_ap = time()  
  
 print("Matrix M3: \n", M3)  
 print("Elapsed time: ", f'{stop\_ap - start\_ap}')

Время на исполнение программы составило … (рис. 3).







Рисунок 3 ‒ Затраченное время на выполнение с использованием 2 потоков

Использование двух потоков для подсчета нашей задачи не принесло пользы. Расчет программы произошел медленнее на ~ 2-2.5с.

Проведем данный опыт с использованием 4, 6, 8, 10, 12, 14 и 16 потоками. Будем добавлять потоки с разбиением считаемых значений для каждого. Затраченное время (в секундах) на выполнение задачи занесем в таблицу 1.

Таблица 1. Затраченное время потоками на выполнение задачи

|  |  |  |  |
| --- | --- | --- | --- |
| Потоки | 1 замер | 2 замер | 3 замер |
| 1 | 13,159 | 13,149 | 13,192 |
| 2 | 15,764 | 15,620 | 15,165 |
| 4 | 16,726 | 16,298 | 16,459 |
| 6 | 16,333 | 16,503 | 16,452 |
| 8 | 16,891 | 16,830 | 16,695 |
| 10 | 16,888 | 16,887 | 16,516 |
| 12 | 17,019 | 16,631 | 16,835 |
| 14 | 16,813 | 16,522 | 16,344 |
| 16 | 16,558 | 16,561 | 16,719 |

# Результаты работы

Мы написали простую программу, которая находит кроссмаштабную разность двух матриц, и протестировали ее выполнение с распараллеливанием процесса. Один поток справился с задачей лучше всего. При использовании 2 потоков разница во времени составила ~ 2-2.5с, немного медленнее. Дальнейшее распараллеливание с 4 до 16 потоков справилось с задачей медленнее, примерно, на 1 секунду. То есть использование более 4 потоков не принесло никаких изменений.

# Выводы

В результате выполненной работы была написана небольшая программа для нахождения кроссмаштабной разности двух матриц. Также было выполнено распараллеливание процесса выполнения программы. Оценивание работы программы происходило замером времени выполнения расчетов. Были получены базовые знания работы с библиотекой «Threading» для распараллеливания процессов на языке Python. Познакомились с темой многопоточности на CPU.

По полученным итогам был сделан вывод, что распараллеливание процесса для выполнения несложных операций не имеет практической выгоды. Возможно многопоточность может помочь ускорить процессы, при выполнении больших и сложных операций.