**Avaliação de aplicação com multiprocessamento em sistema criado na linguagem Python**

**Aviso:** Esse relatório não é nada profissional, e pouquissimos testes foram realizados. É apenas um modelo simples de relatório que será aperfeiçoado em outros estudos.

**1. Introdução**

A utilização de computação nesse projeto será feita para realizar a soma total de um arquivo com vários números diferentes de threads. O objetivo é realizar uma comparação de funcionamento do programa e mostrar o desepenho que ocorreu em cada situação.

**2. Implementação**

Para adicionar timers ao código e registrar o tempo de início e término do processamento, foi utilizada a biblioteca `time` do Python. O que fiz aqui foi adicionar duas variáveis `start\_time` e `end\_time`, que armazenam o tempo de início e término do processamento, respectivamente. Entre elas, calculei o tempo total de processamento subtraindo o tempo de início do tempo de término. Depois, incluí a impressão desse tempo total ao final da execução do programa. Esses passos ajudam a monitorar o tempo de execução do programa.

A seguir, segue o código fonte:

import threading

import time

# Função para calcular a soma de uma parte do vetor

def calcular\_soma\_parte(vetor, inicio, fim, resultado):

soma = sum(vetor[inicio:fim])

resultado.append(soma)

# q: Para que serve o método append

# a: O método append() é utilizado para adicionar um elemento ao final de uma lista.

# Função principal

def main():

# Isso é uma compreensão de lista, o primeiro i é o valor que será adicionado À lista para cada interação do loop, é como se fosse um 'vetor\_grande[i]'

inicio\_execucao = time.time()

vetor\_grande = [i for i in range(1, 100\_000\_001)]

# Número de threads a serem utilizadas

num\_threads = 6

# Dividir o vetor em partes iguais para cada thread

# Lembra da iteração e índices? O primeiro indície é início e o segundo índice é fim.

tamanho\_parte = len(vetor\_grande) // num\_threads

partes = [(i \* tamanho\_parte, (i + 1) \* tamanho\_parte) for i in range(num\_threads)]

# Lista para armazenar os resultados parciais

resultados = []

# Criar e iniciar as threads

threads = []

for inicio, fim in partes:

thread = threading.Thread(target=calcular\_soma\_parte, args=(vetor\_grande, inicio, fim, resultados))

thread.start()

threads.append(thread)

# Aguardar todas as threads terminarem

for thread in threads:

thread.join()

fim\_execucao = time.time()

# Calcular a soma total a partir dos resultados parciais

soma\_total = sum(resultados)

print("Soma total:", soma\_total)

tempo\_execucao = fim\_execucao - inicio\_execucao

minutos = int (tempo\_execucao // 60)

segundos = format(tempo\_execucao % 60, ".2f")

print(f'Tempo de execução \n {minutos} minutos e {segundos} segundos')

if \_\_name\_\_ == "\_\_main\_\_":

main()

**3. Resultados**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Equipamento | Proc (MHz) | Núcleos | Ram | Ram(MHz) | Processador |
| I5 | 2270 | 2 | 6gb | 1067 | Intel(R) Core(TM) i5-M430 |

**4. Conclusão**

Ao utilizar os multiprossamento com threads no programa escrito com a linguagem de programação Python, foi observado que o tempo de execução possui diferentes níveis. Os testes foram realizado em somente um único processador de dois núcleos com 4 processadores lógicos, mas poderiam ser executados em uma ampla variedade de ambientes diferentes. A medida que a quantidade de Threads foi aumentando, de acordo com o número de processadores lógicos do equipamento utilizado (1 a 4), foi possível observar uma PIORA significava de desempenho do sistema, e ao aumentar ainda mais a quantidade de threads, elas começaram a aumentar o nível de alternância, fazendo com que qualquer procesamento com mais de quatro threads seja menos eficaz do que com apenas as 4 threads em si. Os testes deram uma visão mais ampla de como é o funcionamento de threads em python na prática, e a capacidade de desenvolver sistemas que na prática seriam mais eficientes (mas não foram, devido ao GIL do python), utilizando um algoritmo que consiga extrair a maior porcentagem possível de poder de processamento.