Resultados do Laboratorio 3

Leonardo Santiago (120036072)

1 Métodos

O computador utilizado para cálcular tem as seguintes especificações:

```
Model name:           Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
    CPU family:         6
    Model:              158
    Thread(s) per core: 2
    Core(s) per socket: 4
    Socket(s):          1
    Stepping:           10
    CPU max MHz:        4100,0000
    CPU min MHz:        800,0000
```

O programa em C `lab3.c` foi compilado utilizando

```
gcc lab3.c -o lab3 -Wall -lpthread -O3
```

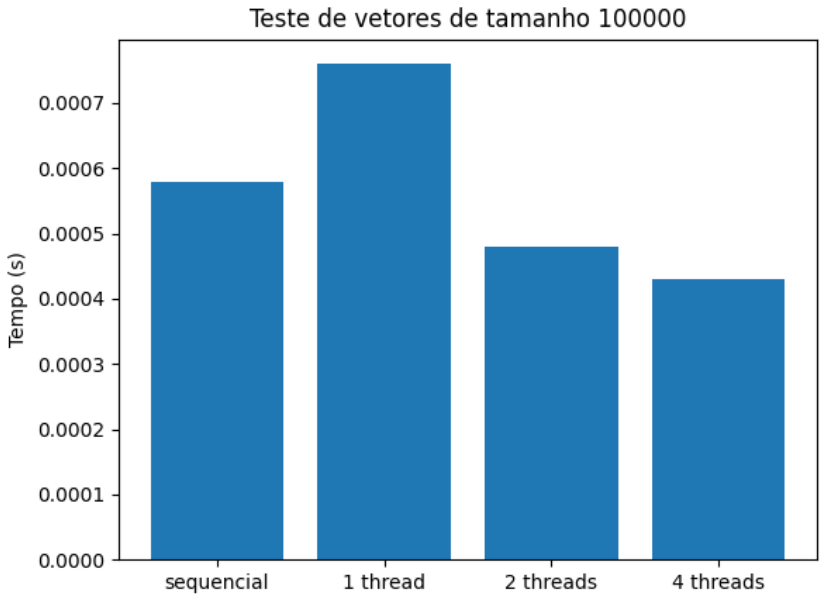
Utilizei o seguinte script em python para medir e plotar os gráficos:

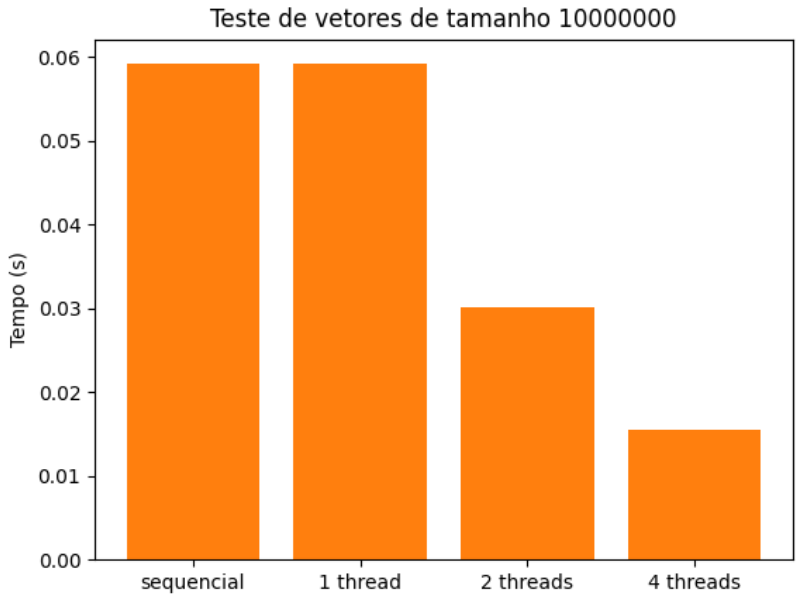
```python
from matplotlib import pyplot as plt
from matplotlib import cm
from subprocess import Popen, PIPE, STDOUT
import re
import numpy as np

labels = ["sequencial", "1 thread", "2 threads", "4 threads"]
x_pos = np.arange(len(labels))

tamanhos = [10**5, 10**7, 10**8]
nthreads = [1, 2, 4]

melhorias = []
```

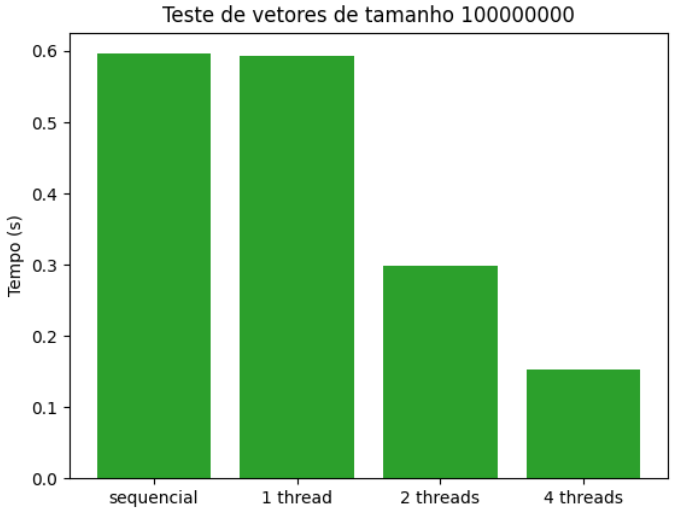
```
for tam in tamanhos:
    melhor_s = 1000
    tempos = []
    for thread in nthreads:
        melhor_c = 1000
        for i in range(5):
            p = Popen(['./lab3', str(tam), str(thread)], stdout=PIPE,
             ↪  stdin=PIPE)
            output, errors = p.communicate(input=str.encode("0\n123"))
            conc = re.search("CONCORRENTE: (\d+\.\d+)s",
             ↪  output.decode()).group(1)
            seq = re.search("SEQUENCIAL: (\d+\.\d+)s",
             ↪  output.decode()).group(1)
            if float(conc) < melhor_c:
                melhor_c = float(conc)
            if float(seq) < melhor_s:
                melhor_s = float(seq)
        tempos.append(melhor_c)
    [melhorias.append((thread, melhor_s/tempo)) for thread, tempo in
     ↪  zip(nthreads, tempos)]
    tempos.insert(0, melhor_s)
    plt.bar(x_pos, tempos)
    plt.xticks(x_pos, labels)
    plt.ylabel("Tempo (s)")
    plt.title(f"Teste de vetores de tamanho {tam}")
    plt.savefig(f"tam{tam}thread{thread}.png")

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)

x = np.linspace(1, 5, 1000)
ys = [(slope * x, nthread) for nthread, slope in melhorias]

for y, nthread in ys:
    ax.plot(x, y, c=cm.Dark2(nthread/4))

l, = ax.plot(x, x, "r--", label="sequencial")
ax.legend([f'{n} threads' for n in nthreads], loc="upper left")

fig.savefig("performance.png", dpi=fig.dpi)
```

2 Resultados

Novamente, vemos que a solução concorrente foi consistentemente mais rápida do que a solução sequencial, chegando a ser até 4 vezes mais rápida. Podemos ver isso nas imagens 1, 2 e 3, que demonstram claramente a relação:







Se plotarmos a aceleração de cada uma das execuções ($T_{sequencial}/T_{concorrente}$), vemos que o número de threads afetou diretamente na velocidade de execução dos arquivos. Vemos também que uma das retas de 4 threads está com um coeficiente abaixo das demais, mas isso provavelmente se deve ao fato de que o programa com 10^5 threads roda tão rápido (nos décimos de milésimos de segundos) que a quantidade de threads não é um fator tão grande.

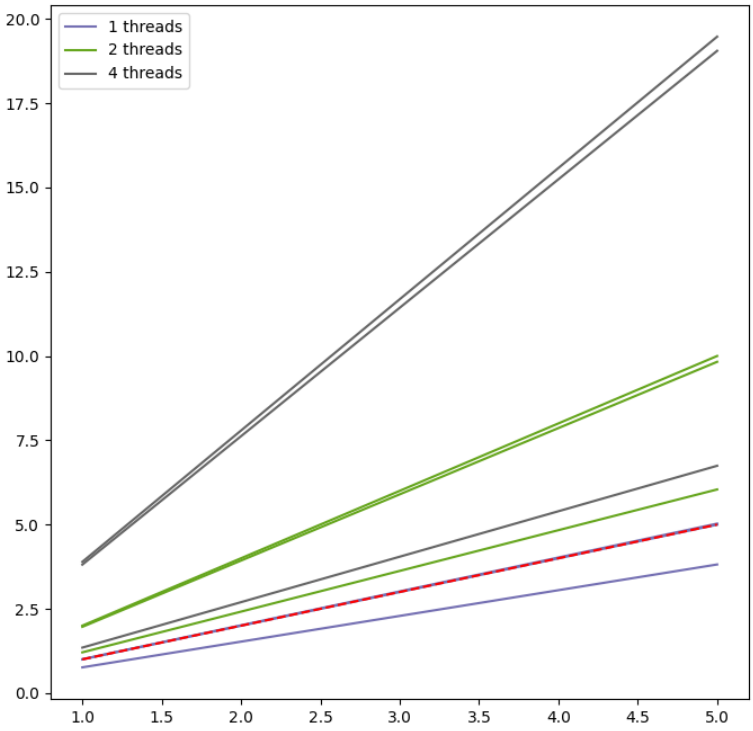


Figure 1: A linha vermelha pontilhada representa a performance do algoritmo sequencial.