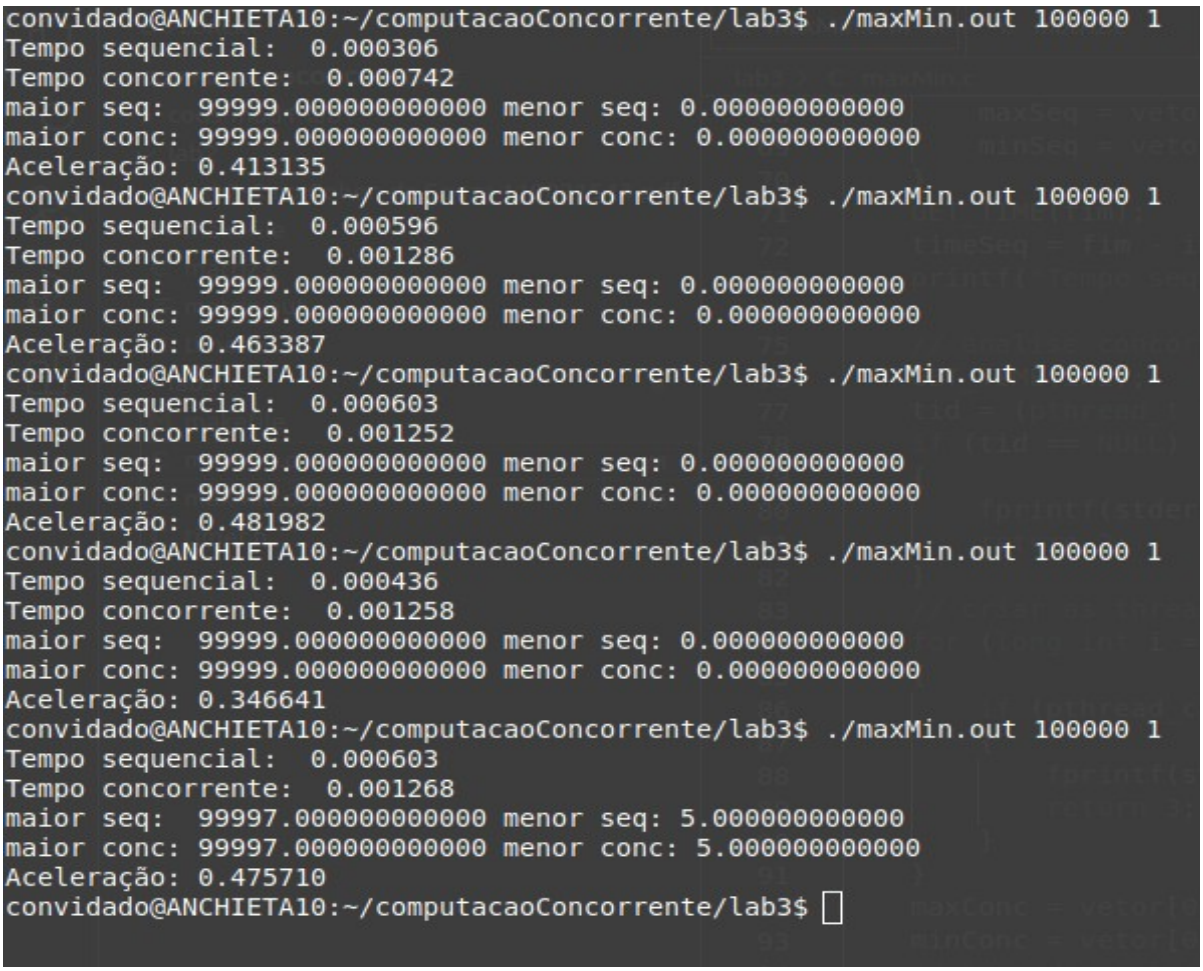
Análise Lab 3- Caio Cohen de Souza

No programa feito, ele encontra o maior e o menor valor de um array. Os elementos dessa array são aleatórios. O programa imprime o tempo de execução sequencial e concorrente, assim como os valores maximos e minimos encontrados em ambos(quem devem ser iguais).



Como pode-se observar, executar o algoritmo com apenas uma única thread acabou deixando o programa mais lento, no pior caso quase 3 vezes mais lento. Isso acontece por conta do tempo de criação da thread, como estamos operando com um array de dimensão única, o tempo levado para o programa ser executado é muito pequeno, então o tempo de criação das threads acaba tendo uma relevância muito grande.

```
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000 2
Tempo sequencial:  0.000305
Tempo concorrente:  0.000495
maior seq:  99999.000000000000 menor seq: 1.000000000000
maior conc: 99999.000000000000 menor conc: 1.000000000000
Aceleração: 0.615436
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000 2
Tempo sequencial:  0.000602
Tempo concorrente:  0.000825
maior seq:  99999.000000000000 menor seq: 0.000000000000
maior conc: 99999.000000000000 menor conc: 0.000000000000
Aceleração: 0.730654
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000 2
Tempo sequencial:  0.000593
Tempo concorrente:  0.000869
maior seq:  99999.000000000000 menor seq: 0.000000000000
maior conc: 99999.000000000000 menor conc: 0.000000000000
Aceleração: 0.682586
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000 2
Tempo sequencial:  0.000600
Tempo concorrente:  0.000831
maior seq:  99996.000000000000 menor seq: 0.000000000000
maior conc: 99996.000000000000 menor conc: 0.000000000000
Aceleração: 0.721552
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000 2
Tempo sequencial:  0.000601
Tempo concorrente:  0.000864
maior seq:  99996.000000000000 menor seq: 0.000000000000
maior conc: 99996.000000000000 menor conc: 0.000000000000
Aceleração: 0.695313
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$
```

Nesse caso, executando com duas threads, podemos percebemos que a aceleração geral, em comparação com o teste anterior melhorou, mas o tempo de criação da thread ainda está atrasando o programa.

```
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000 4
Tempo sequencial:  0.000300
Tempo concorrente:  0.000478
maior seq:  99999.000000000000 menor seq: 0.000000000000
maior conc: 99999.000000000000 menor conc: 0.000000000000
Aceleração: 0.628664
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000 4
Tempo sequencial:  0.000595
Tempo concorrente:  0.000761
maior seq:  99999.000000000000 menor seq: 0.000000000000
maior conc: 99999.000000000000 menor conc: 0.000000000000
Aceleração: 0.782292
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000 4
Tempo sequencial:  0.000601
Tempo concorrente:  0.000763
maior seq:  99999.000000000000 menor seq: 0.000000000000
maior conc: 99999.000000000000 menor conc: 0.000000000000
Aceleração: 0.787362
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000 4
Tempo sequencial:  0.000598
Tempo concorrente:  0.000775
maior seq:  99999.000000000000 menor seq: 0.000000000000
maior conc: 99999.000000000000 menor conc: 0.000000000000
Aceleração: 0.771541
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000 4
Tempo sequencial:  0.000593
Tempo concorrente:  0.000792
maior seq:  99999.000000000000 menor seq: 0.000000000000
maior conc: 99999.000000000000 menor conc: 0.000000000000
Aceleração: 0.748456
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$
```

Agora com 4 threads podemos perceber que ficou ligeiramente mais rápida que o teste anterior, mas ainda assim não está compensando, pelo fato de que o tempo de execução total já ser muito curta.

```
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 1
Tempo sequencial:  0.027507
Tempo concorrente:  0.048620
maior seq:  9999999.000000000000 menor seq: 0.000000000000
maior conc: 9999999.000000000000 menor conc: 0.000000000000
Aceleração: 0.565747
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 1
Tempo sequencial:  0.027234
^[[ATempo concorrente:  0.042448
maior seq:  9999999.000000000000 menor seq: 2.000000000000
maior conc: 9999999.000000000000 menor conc: 2.000000000000
Aceleração: 0.641594
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 1
Tempo sequencial:  0.027460
Tempo concorrente:  0.048547
maior seq:  9999998.000000000000 menor seq: 0.000000000000
maior conc: 9999998.000000000000 menor conc: 0.000000000000
Aceleração: 0.565638
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 1
Tempo sequencial:  0.027268
Tempo concorrente:  0.041875
maior seq:  9999998.000000000000 menor seq: 0.000000000000
maior conc: 9999998.000000000000 menor conc: 0.000000000000
Aceleração: 0.651184
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 1
Tempo sequencial:  0.027439
Tempo concorrente:  0.042126
maior seq:  9999997.000000000000 menor seq: 0.000000000000
maior conc: 9999997.000000000000 menor conc: 0.000000000000
Aceleração: 0.651355
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$
```

Agora testando com um array de maior dimensão. Dessa vez, o tempo total de processamento aumentou, diminuindo a relevância do tempo de criação de cada thread, por causa disso, a aceleração geral que executar com apenas 1 thread trouxe em uma array de 10^7 elementos está ligeiramente maior que com uma array de 10^5 elementos.

```
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 2
Tempo sequencial:  0.027250
Tempo concorrente:  0.028391
maior seq:  9999999.000000000000 menor seq: 0.000000000000
maior conc: 9999999.000000000000 menor conc: 0.000000000000
Aceleração: 0.959825
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 2
Tempo sequencial:  0.027297
Tempo concorrente:  0.028599
maior seq:  9999998.000000000000 menor seq: 2.000000000000
maior conc: 9999998.000000000000 menor conc: 2.000000000000
Aceleração: 0.954455
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 2
Tempo sequencial:  0.027277
Tempo concorrente:  0.028235
maior seq:  9999999.000000000000 menor seq: 0.000000000000
maior conc: 9999999.000000000000 menor conc: 0.000000000000
Aceleração: 0.966057
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 2
Tempo sequencial:  0.027232
Tempo concorrente:  0.028884
maior seq:  9999997.000000000000 menor seq: 1.000000000000
maior conc: 9999997.000000000000 menor conc: 1.000000000000
Aceleração: 0.942805
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 2
Tempo sequencial:  0.027323
Tempo concorrente:  0.025283
maior seq:  9999997.000000000000 menor seq: 1.000000000000
maior conc: 9999997.000000000000 menor conc: 1.000000000000
Aceleração: 1.080678
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$
```

Pode-se perceber que o tempo de execução geral ainda é muito pequeno, por isso, devido ao tempo de criação das threads, o tempo que a execução do algoritmo sequencial e concorrente com 2 threads acaba sendo quase igual.

```
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 4
Tempo sequencial:  0.027463
Tempo concorrente:  0.024686
maior seq:  9999999.000000000000 menor seq: 0.000000000000
maior conc: 9999999.000000000000 menor conc: 0.000000000000
Aceleração: 1.112479
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 4
Tempo sequencial:  0.027490
^[[ATempo concorrente:  0.019694
maior seq:  9999999.000000000000 menor seq: 0.000000000000
maior conc: 9999999.000000000000 menor conc: 0.000000000000
Aceleração: 1.395805
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 4
Tempo sequencial:  0.027272
Tempo concorrente:  0.024823
maior seq:  9999999.000000000000 menor seq: 0.000000000000
maior conc: 9999999.000000000000 menor conc: 0.000000000000
Aceleração: 1.098645
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 4
Tempo sequencial:  0.027253
Tempo concorrente:  0.014382
maior seq:  9999999.000000000000 menor seq: 2.000000000000
maior conc: 9999999.000000000000 menor conc: 2.000000000000
Aceleração: 1.894943
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 10000000 4
Tempo sequencial:  0.027245
Tempo concorrente:  0.012944
maior seq:  9999997.000000000000 menor seq: 0.000000000000
maior conc: 9999997.000000000000 menor conc: 0.000000000000
Aceleração: 2.104853
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$
```

Agora, com 4 threads, pode-se perceber uma aceleração geral do código, o que já diferencia do teste passado, com uma array de 10^5 elementos, uma vez que com uma array de 10^7 elementos o tempo geral aumenta e as threads se tornam mais relevantes.

```
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 1
Tempo sequencial:  0.283888
Tempo concorrente:  0.414413
maior seq:  99999999.000000000000 menor seq: 0.000000000000
maior conc: 99999999.000000000000 menor conc: 0.000000000000
Aceleração: 0.685036
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 1
Tempo sequencial:  0.272249
Tempo concorrente:  0.411318
maior seq:  99999999.000000000000 menor seq: 0.000000000000
maior conc: 99999999.000000000000 menor conc: 0.000000000000
Aceleração: 0.661894
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 1
Tempo sequencial:  0.272417
Tempo concorrente:  0.417807
maior seq:  99999996.000000000000 menor seq: 0.000000000000
maior conc: 99999996.000000000000 menor conc: 0.000000000000
Aceleração: 0.652015
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 1
Tempo sequencial:  0.272484
Tempo concorrente:  0.418187
maior seq:  99999999.000000000000 menor seq: 2.000000000000
maior conc: 99999999.000000000000 menor conc: 2.000000000000
Aceleração: 0.651585
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 1
Tempo sequencial:  0.274789
Tempo concorrente:  0.416498
maior seq:  99999998.000000000000 menor seq: 0.000000000000
maior conc: 99999998.000000000000 menor conc: 0.000000000000
Aceleração: 0.659760
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$
```

Como já era esperado, executar uma função concorrente com apenas uma thread sempre vai ser mais lento que o sequencial, uma vez que com apenas uma thread tem todo o trabalho de criar e remover as threads que o sequencial não tem.

```
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 2
Tempo sequencial:  0.272452
Tempo concorrente:  0.225371
maior seq:  99999999.000000000000 menor seq: 0.000000000000
maior conc: 99999999.000000000000 menor conc: 0.000000000000
Aceleração: 1.208907
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 2
Tempo sequencial:  0.272770
Tempo concorrente:  0.221477
maior seq:  99999999.000000000000 menor seq: 4.000000000000
maior conc: 99999999.000000000000 menor conc: 4.000000000000
Aceleração: 1.231599
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 2
Tempo sequencial:  0.273132
Tempo concorrente:  0.227223
maior seq:  99999999.000000000000 menor seq: 0.000000000000
maior conc: 99999999.000000000000 menor conc: 0.000000000000
Aceleração: 1.202045
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 2
Tempo sequencial:  0.275195
Tempo concorrente:  0.226618
maior seq:  99999999.000000000000 menor seq: 0.000000000000
maior conc: 99999999.000000000000 menor conc: 0.000000000000
Aceleração: 1.214353
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 2
Tempo sequencial:  0.272618
Tempo concorrente:  0.225819
maior seq:  99999999.000000000000 menor seq: 0.000000000000
maior conc: 99999999.000000000000 menor conc: 0.000000000000
Aceleração: 1.207238
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$
```

Agora nós obtivemos uma aceleração consistente ao aumentar o número de threads para 2, uma vez que o tempo total aumentou.

```
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 4
Tempo sequencial:  0.272702
Tempo concorrente:  0.134976
maior seq:  99999999.000000000000 menor seq: 0.000000000000
maior conc: 99999999.000000000000 menor conc: 0.000000000000
Aceleração: 2.020375
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 4
Tempo sequencial:  0.274924
Tempo concorrente:  0.147889
maior seq:  99999999.000000000000 menor seq: 2.000000000000
maior conc: 99999999.000000000000 menor conc: 2.000000000000
Aceleração: 1.858985
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 4
Tempo sequencial:  0.272430
Tempo concorrente:  0.181441
maior seq:  99999997.000000000000 menor seq: 1.000000000000
maior conc: 99999997.000000000000 menor conc: 1.000000000000
Aceleração: 1.501481
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 4
Tempo sequencial:  0.272486
Tempo concorrente:  0.154104
maior seq:  99999999.000000000000 menor seq: 0.000000000000
maior conc: 99999999.000000000000 menor conc: 0.000000000000
Aceleração: 1.768188
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$ ./maxMin.out 100000000 4
Tempo sequencial:  0.272434
Tempo concorrente:  0.147947
maior seq:  99999998.000000000000 menor seq: 0.000000000000
maior conc: 99999998.000000000000 menor conc: 0.000000000000
Aceleração: 1.841423
convidado@ANCHIETA10:~/computacaoConcorrente/lab3$
```

Agora sim, finalmente, tivemos uma aceleração considerável do tempo de processamento, uma vez que com 10^8 elementos no array, o tempo aumentou e ao usarmos 4 threads, obtivemos, no melhor caso dos teste, uma aceleração de até 2 vezes.