Centro Universitário da FEI

Atividade 1 - Sistemas Distribuídos

Weverson da Silva Pereira - RA: 22.119.004-4
Gabriel Vila Real de Oliveira - RA: 22.119.077-0

São Bernardo do Campo
2022

1 Introdução

O objetivo deste relatório é discutir os resultados apresentados durante a experimentação do problema acerca da aplicação de estratégias multithreaded para solucionar um algoritmo de cálculo de números primos numa base de dados extensa.

Sabendo que o uso de computação paralela possui performance distinta entre as linguagens de programação, foram desenvolvidas duas aplicações, uma em Python[1] e outra em C#[2], com o intuito de averiguar essas diferenças e medir o tempo de execução e speedup em cada uma delas.

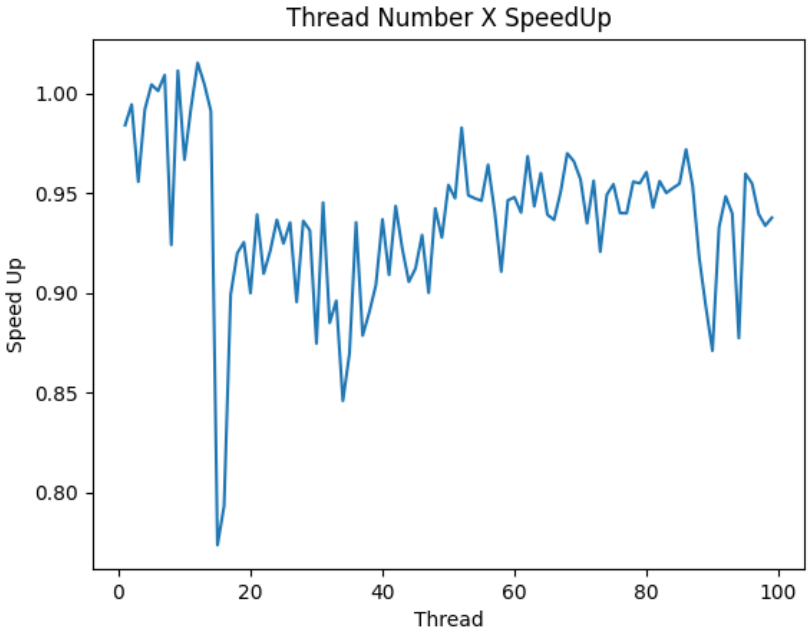
[1] https://github.com/WebisD/prime-with-threads
[2] https://github.com/GabrielBueno200/MultiThreadedPrimesSolver

2 Python

Devido ao GIL[3] o uso de Threads em Python não apresenta um resultado satisfatório.

> Since the GIL allows only one thread to execute at a time even in a multi-threaded architecture with more than one CPU core, the GIL has gained a reputation as an "infamous" feature of Python. [4]

A seguir, alguns gráficos mostrando a eficácia de diversas threads. Todas foram reproduzidas 50 vezes para assegurar a precisão



[3] https://wiki.python.org/moin/GlobalInterpreterLock
[4] https://realpython.com/python-gil/

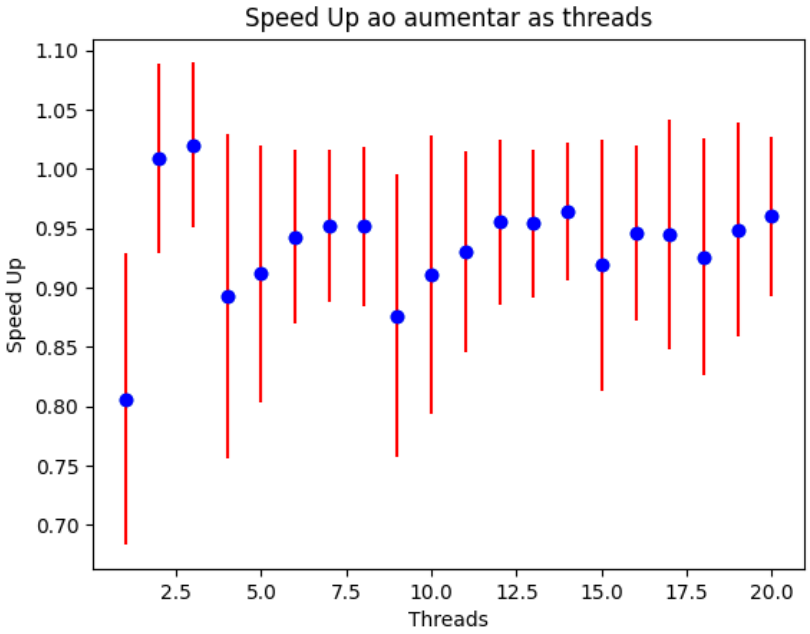


Figure 1: Média dos 50 testes com o cálculo do desvio padrão

3 C#

Conforme discutido no tópico anterior, sabe-se que o Python não apresenta alta eficiência em relação à multithreading.

Com isso, uma aplicação em C# foi desenvolvida, com o intuito de analisar empiricamente a abordagem multithreading com mais assertividade. Também foram incluídas algumas melhorias, como por exemplo, o retorno correto da quantidade total de números de primos, independentemente do número de threads aplicadas, o que não era garantido no Python, que por vezes apresentava uma certa variação neste número.

A seguir, temos alguns gráficos referentes à execução do código em C#.

3.1 Gráficos

Para os dois gráficos abaixo, os parâmetros analisados foram o número de threads e o speedup resultante de suas respectivas execuções.

Assim como nos gráficos anteriores, cada quantidade de thread (de 2 a 452 threads) foi executada por 50 vezes, calculando-se as medianas de cada valor (speedUp, e tempo de execução), assim como o desvio padrão dos speedups para todas as execuções.

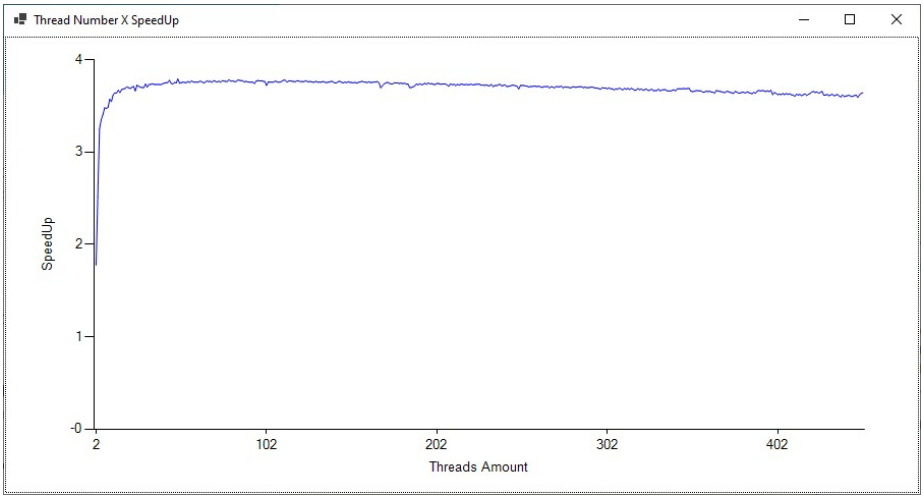


Figure 2: Mediana dos 50 testes com cada quantidade de threads

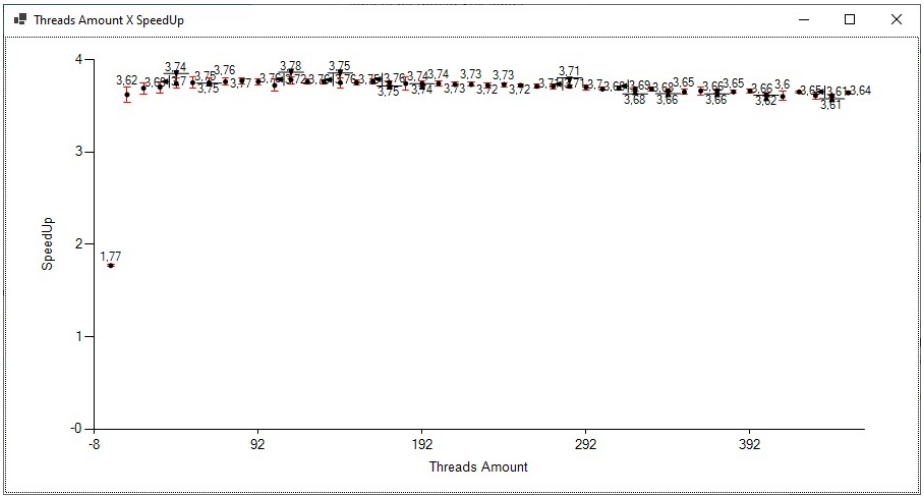


Figure 3: Mediana dos 50 testes com cada quantidade de threads aplicando-se o desvio padrão

Com base nos dois gráficos acima, observa-se uma diferença expressiva entre as duas aplicações, sendo o maior speedup encontrado próximo de 3.8 enquanto que no Python dificilmente supera 1.

O que podemos concluir do gráfico da Figura 2 é que o aumento de threads não necessariamente significa num aumento relevante de performance quando trabalhadas no C#; nota-se, inclusive, que mesmo muito estabilizado e quase constante, o C# apresenta um declínio mínimo de speedup conforme o aumento do número de threads. Além disso, observa-se um erro mínimo durante as execuções.

3.2 Fração serial

Com base nos mesmos testes feitos nos gráficos acima, calculamos o caso com o maior speedup entre os 452, extraindo o seu respectivo número de threads e a mediana do seu tempo de execução e do seu speedup, encontrando um número de threads igual a 50.

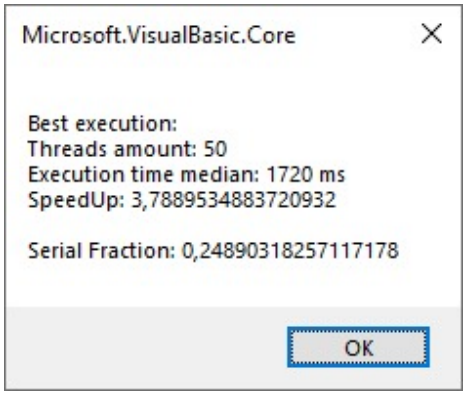


Figure 4: Melhor caso encontrado entre as 50 execuções no *range* de threads

$$S = \frac{f_s + (1 - f_s)}{f_s + \frac{1-f_s}{n}}$$

$$\Rightarrow f_s = \frac{n - S}{(n-1)\,S}$$

$$\Rightarrow f_s = \frac{n - S}{(n-1)\,S} \Rightarrow f_s = \frac{50 - 3.79}{(50-1)\,3.79} \Rightarrow \mathbf{f_s \approx 0.249}$$

Além desses dois gráficos, também foram desenvolvidos outros dois. O primeiro leva em conta Quantidade de Threads X Tempo de Execução, num range de 1 até uma quantidade de input digitada, sendo cada quantidade testada apenas uma única vez. Quanto ao segundo, a quantidade de testes feita é dada também por um input, sendo que o usuário tem a possibilidade de comparar uma quantidade pequena de threads (2 a 5) e uma quantidade alta (≥ 5). Ambos os gráficos levam em conta thread X tempo.

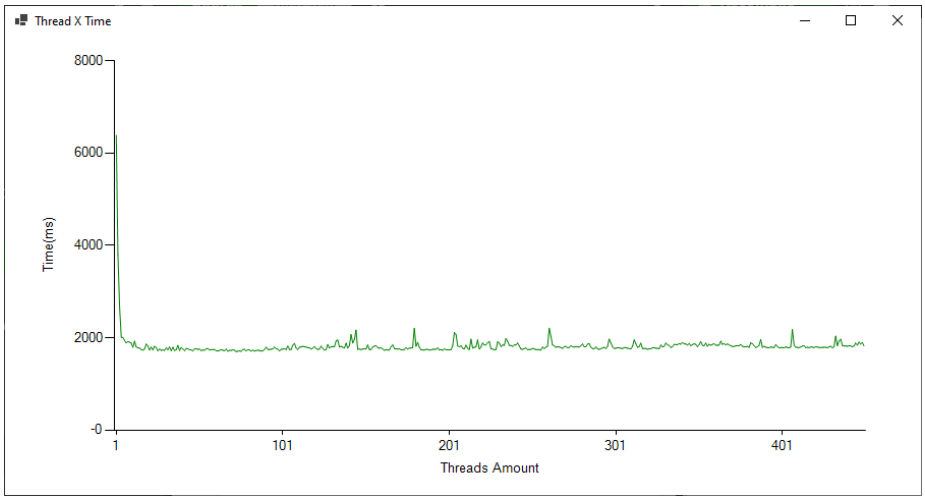


Figure 5: Tempo (ms) X Número de threads

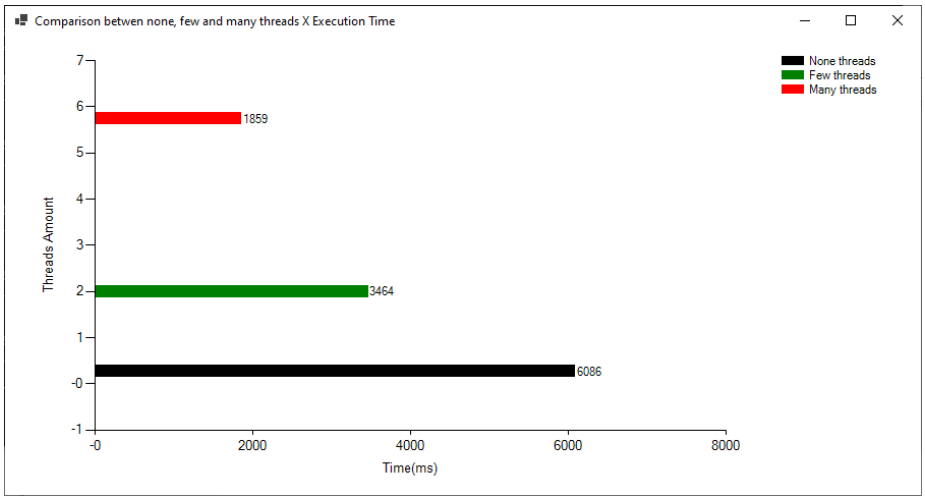


Figure 6: Nenhuma thread, poucas threads e muitas threads X Tempo (ms)

4 Conclusão

Podemos concluir com este trabalho que nem sempre paralelizar uma solução algorítmica resulta em ganho desempenho, isso também depende de certo fatores. Neste experimento, por exemplo, checamos que o fator linguagem pode ser um deles, em que o Python desempenha mal com multithreading, diferente do C#, que fica estável, mas para de ganhar performance quando atinge um certo valor de speedup (até mesmo perdendo, assim como o Python, mesmo que minimamente).

Outro fator é a alta troca de contexto. Quando elevamos o número de

threads, isso faz com que a comunicação entre as threads tenha alto custo de processamento em relação às próprias operações necessárias para a solução algorítmica, o que afeta diretamente o speedup resultante.