Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, v. 11, n. 1, 2025.

Trabalho apresentado no XLIII CNMAC, Centro de Convenções do Armação Resort - Porto de Galinhas - PE, 2024

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Eficiência de Programação Concorrente em Métodos para Obtenção de Zeros de Funções

Anderson C. da S. Morais[1], João V. da S. Cruz[2], Matheus da S. Menezes [3]
DCME/UFERSA, Mossoró, RN

Embora os métodos numéricos sejam frequentemente eficazes, em certos casos, encontrar uma raiz de função, ou seja, um valor x tal que $f(x) = 0$ não é uma tarefa trivial [1], o que pode tornar o processamento computacional lento, influenciado pela precisão e pelas condições de convergência desejadas.

Computacionalmente, cada método de refinamento de raízes de funções utiliza uma quantidade de recursos computacionais e possui um tempo de processamento próprio. Uma forma de minimizar esses gastos é utilizar apenas do método que consiga convergir primeiro, assim, os demais métodos não precisam estar em execução. Com base nessa situação, temos uma abordagem do problema para um algoritmo sequencial e para um algoritmo em paralelo.

A programação em paralelo ou concorrente é uma técnica utilizada para melhorar o desempenho computacional em determinadas aplicações. A técnica consiste em selecionar uma tarefa complexa e subdividir em segmentos menores que sejam resolvidos simultaneamente, diminuindo o tempo de conclusão total do processo [2]. Uma das formas de implementação é por meio do *multithreading*, ou paralelismo de *thread*, em que cada processo será executado simultaneamente em *threads* distintos, reduzindo o tempo final de execução [3].

Com base nos conceitos de programação concorrente e sequencial, foram criado três algoritmos. O primeiro sendo um algoritmo sequencial que executa todos os métodos númericos de zeros de funções e retorna um conjunto de soluções, o segundo que executa cada método em paralelo e apresenta apenas o resultado do método numérico com menor tempo de processamento, encerrando a execução dos demais, e o terceiro que executa os métodos em paralelo sem encerrar a execução do cálculo dos outros métodos, mesmo após a convergência de um dos métodos.

Com o objetivo de atestar a eficiência em tempo da programação concorrente por meio do *multithreading*, realizamos um experimento computacional considerando os métodos da bissecção, falsa posição, Newton-Raphson e secante, utilizando as seguintes funções com seus valores de iniciação para cada método:

- $f_1(x) = x^5 - 2x^4 - 9x^3 + 22x^2 + 4x - 24$, no intervalo $[0, 5]$, $x_0 = 3$ e $x_1 = 5$.
- $f_2(x) = e^x - x - 1$, no intervalo $[-1, 2]$, $x_0 = 2$ e $x_1 = 1$.
- $f_3(x) = (x - 3)^5 \log_e(x)$, no intervalo $[1.5, 5]$, $x_0 = 5$ e $x_1 = 4$.
- $f_4(x) = x^3 - 9x + 5$, no intervalo $[2, 3]$, $x_0 = 1$ e $x_1 = 1.5$.

Os cálculos foram realizados por meio da linguagem de programação Python3, considerando as funções listadas acima, com a determinação da derivada calculada numericamente com $dx =$

[1]anderson.morais92395@alunos.ufersa.edu.br
[2]joao.cruz@alunos.ufersa.edu.br
[3]matheus@ufersa.edu.br

0.0000000001 utilizado no método de Newton-Raphson. Como critérios de parada foram usados $|f(x)| < 10^{-11}$ ou um limite de 1000 iterações. Os experimentos foram realizados em um computador do tipo *notebook* com sistema operacional Ubuntu 22.04 LTS, com processador AMD ryzen 5 5500U, memória RAM de 12GB ddr4 3200mhz e um SSD NVME de 256GB de armazenamento.

Tabela 1: Tempo de processamento para algoritmos sequencial, paralelo com e sem exclusão.

Função	Método que convergiu	Sequencial (ns)	Paralelismo e exclusão (ns)	Paralelismo (ns)
f_1	Secante	115299	6538964	92673
f_2	Newton-Raphson	116766	4097284	74724
f_3	Bissecção	34848	2099833	60129
f_4	Newton-Raphson	5866	1775304	6216

A Tabela 1 apresenta os resultados encontrados para cada função, onde a primeira coluna indica a função considerada, a segunda coluna nos diz qual foi o método que obteve a convergência mais rápida para o problema, e da terceira até a quinta coluna temos o tempo de processamento por estratégia de implementação. A implementação utilizando *multithreading* para encontrar os zeros de funções não foi satisfatória, uma vez que esse método foi menos eficiente quanto ao tempo de processamento. Com isso, nota-se que em casos de problemas que já são eficientes, o uso do *multithread* pode não ser a melhor solução para otimizá-los, devido principalmente a sobrecarga das *threads* que supera os benefícios do paralelismo como também o tempo de leitura das bibliotecas necessárias para a implementação do *multithread*.

Referências

[1] M. A. G. Ruggiero e V. L. R. Lopes. **Cálculo Numérico: Aspectos Teóricos e Computacionais.** 2a Ed. São Paulo: Makron Books, 1997.

[2] G. R. Andrews e F. B. Schneideil. "Concepts and Notations for Concurrent Programming." Em: **ACMComputing surveys** 15.1 (1983).

[3] D. E. Culler, J. P. Singh e A. Gupta. **Parallel Computer Architecture: A Hardware/Software Approach.** 1a Ed. Morgan Kaufmann, 1998.