# Regra do Trapézio Composta Concorrente

Programação Concorrente (ICP-361) - 2022/2

Wemerson Silva Caxias da Costa - 120063485

Novembro, 2022

# 1 Descrição do problema

No cálculo, a integral de uma função foi criada originalmente para determinar a área sob uma curva no plano cartesiano. Mas além disso, existem outras aplicações, que se estendem para os mais váriados campos do conhecimento, como por exemplo, a geometria, a física, engenharia, economia, biologia, probabilidade, e muitos outros. Na geometria, além do cálculo de áreas sob curvas como já citado, podemos utilizar integrais para calcular comprimentos de arcos, volumes, áreas de superfícies. Já na física, para calcular o trabalho realizado por uma força, momento, centros de massa e momento de inércia, e etc.

Mas embora integrais sejam muito úteis, integrais complexas são difíceis de serem calculadas, até mesmo para o computador. Métodos que facilitem o cálculo de integrais definidas possuem extrema importância em todas as áreas que utilizam o cálculo integral. A regra do trapézio composta é uma forma de estimar integrais, especialmente útil quando encontramos integrais que não sabemos como calcular.

## 1.1 Regra do Trapézio Composta

Nesse método de integração, a função é dividida em vários subintervalos e cada intervalo é aproximado por um trapézio de tamanho conhecido. Calculamos então a área de cada trapézios e somamos todas essas pequenas áreas para encontrarmos a integral. Quanto maior o número de trapézios usados, melhor a aproximação.

Como proposta de solução sequencial, o proposto foi criar uma estrutura de repetição que rode **n** vezes a expressão utilizada para calcular o valor de cada pequena parte, e que a cada iteração incremente uma variável **x** com o resultado de cada uma dessas somas. O valor da variável **x** ao final de todas essas iterações é o valor que desejamos.

Mas embora computacionalmente não seja tão custoso calcularmos mesmo um número consideravelmente grande de trapézios (um **n** grande), as vezes na nossa aplicação não queremos lidar com o número de sub-intervalos, e sim com um certo grau de precisão de casas decimais. É nesse aspecto que a concorrência se mostra importante, pois alcançar uma precisão de por exemplo, 8 casas decimais, é um processo longo, como veremos mais abaixo.

Neste relatório, abordaremos o uso de uma aplicação concorrente desse método, afim de podermos obter um maior grau de precisão sem sacrificar muito desempenho.

## 1.2 Entrada e Resultados Esperados

Como dados de entrada desse problema temos: a expressão da função que desejamos calcular a integral, os limites de integração **a** e **b**, e a nossa tolerância **tol**.

Como temos dificuldade na linguagem C para interpretarmos expressões matemáticas complexas digitadas pelo usuário, usaremos um pequeno conjunto de

funções onde serão varíados os dados citados logo acima.

Como nesse trabalho não há a dependência de uma estrutura de dados de entrada, não haverá influência do tempo de acesso a memoria, tirando uma possível vantagem da escolha de divisão de tarefa em bloco em relação a escolha de divisão alternada, onde nesse último caso acessamos muito mais a memória, causando problemas de overhead.

Serão utilizadas nessa análise de desempenho as funções:

- $\int x^2 \, dx$
- $\int \frac{5 \cdot x^3}{10^8} + -5x \, dx$
- $\int cos(x) \, dx$

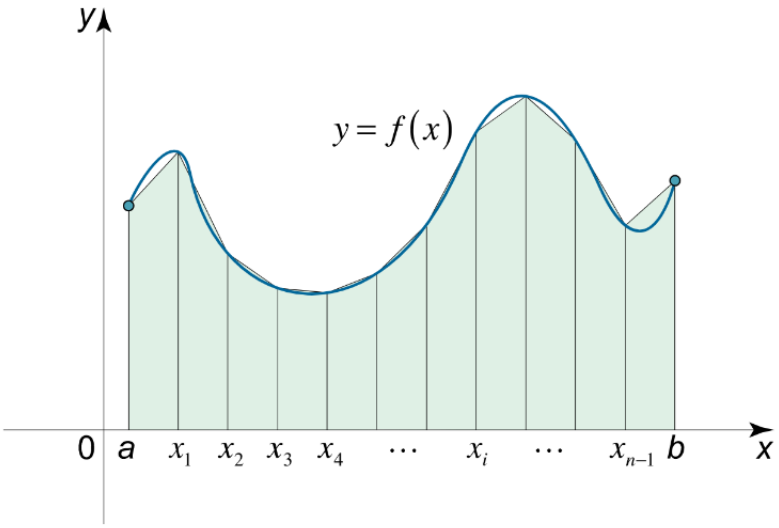
  Para análise de corretude, usaremos de resultados gerados a partir de ferramentas externas.

## 2 Projeto e implementacao da solucao concorrente

Para entendermos o projeto de solução concorrente, vejamos de forma mais extensiva o que eu descrevi um pouco mais acima: Seja $[a, b]$ um intervalo de integração, e $n$ o número de sub-intervalos que escolhemos, de forma que $a = x_0 < x_1 < ... < x_n = b$. Temos então:

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(a) + 2\sum_{j=1}^{n-1} f(a+jh) + f(b)\right]$$

A partir da expressão acima que usaremos para conseguir o que queremos, podemos concluir que de fato há um paralelismo nesse algoritmo, já que cada uma das partições que fazemos pode ser calculada independente de ordem, já que cada pedaço não depende do valor de nenhum dos outros pedaços, e sim dos valores da função nos pontos daquele subintervalo, valores esses que são totalmente independentes entre si (portanto não há condição de corrida). E portanto, uma solução concorrente permite que, ao utilizarmos fluxos de execução independentes para fazermos o nosso cálculo, reduzamos o tempo total de processamento requerido. Como a precisão da nossa solução depende de quão grande escolhemos o n, ao passarmos para a versão concorrente, podemos obter um grau maior de precisão sem tanta perda de desempenho como ocorre na solução sequencial.




Visualização gráfica da soma de Trapézios

# 3 Decisões da implementação

Inicialmente, como visto no Relatório Parcial desse trabalho, a estratégia inicial para resolver esse problema era usar a estratégia alternada, onde cada thread iria calculando de forma alternada cada iteração, nesse caso, o valor de um ponto na função. Mas após uma análise mais profunda, percebe-se que isso poderia gerar problemas no balanceamento de carga, pois a complexidade de cada ponto não necessariamente é igual, e sim pontos mais "difíceis"ou mais "fáceis"dependerão da função e de qual parte dela eu estou olhando.

Tendo isso em vista, a estratégia adotada foi a abordagem de divisão em blocos, onde cada thread calculará um número $\frac{n}{NTHREADS}$ de pontos da função, onde **n** é o número de trapézios que dividiremos o intervalo, e **NTHREADS** é o número total de threads que estamos usando naquela execução. Cada thread vai acumulando os valores relativos ao seu bloco e adicionam numa variável do tipo contador que vai atualizando, com auxílio da exclusão mútua, a nossa variável **integral**. Ao termos todos os pontos calculados, aplicamos sobre essa variável as outras duas manipulações da nossa fórmula: somamos $f(a)$ e $f(b)$, e, por fim, multiplicamos tudo por $h/2$, onde $h$ é a largura dos nossos trapézios.

O processo descrito acima diz respeito a uma iteração do programa, mas, como queremos atingir um certo grau de tolerância, precisamos fazer várias: se ainda não estamos dentro do limíte, pegamos o valor que acabamos de calcular, armazenamos numa variável auxiliar, e calculamos a integral de novo, mas agora com mais trapézios, isto é, com um **n** maior, gerando maior precisão. O processo para mesmo caso não seja atingido o limite caso chegue a 35 iterações.

Por não dependermos de uma estrutura de dados de entrada, não haverá influencia do tempo de acesso a memoria, tirando o que poderia ser uma vantagem nessa escolha de divisao de tarefa em bloco: acessar menos a memória do que acessariámos na abordagem alternada.

## 3.1 Corretude

A corretude do programa foi provada com auxílio de ferramentas externas como o Wolfram—Alpha e o Symbolab, e além disso, no código há uma implementação sequencial do programa que usaremos para comparar além do tempo, o erro relativo entre as duas abordagens:

$$\text{Erro} = \frac{|\text{Solução Concorrente - Solução Sequencial}|}{\text{Solução Concorrente}}$$

O motivo de eu colocar a solução concorrente foi como a mais correta e erro ser medido a partir dela, foi por observar em diversas comparações com as ferramentas externas que o resultado concorrente se aproximava mais do resultado esperado.

O programa considera que o resultado obtido passa no teste de corretude se Erro <= $10^-8$, i.e, se a diferença entre os dois resultados é no máximo 0.000001%.

# 4 Avaliação de desempenho

Os tempos de todos os casos de testes a seguir foram tomados a partir do tempo médio de 3 execuções. Para as duas primeiras integrais, os casos de teste estão definidos da seguinte forma:

$$\text{Caso } 1 : a = 0; b = 10^5$$

$$\text{Caso } 2 : a = 0; b = 10^6$$

$$\text{Caso } 3 : a = 0; b = 10^7$$

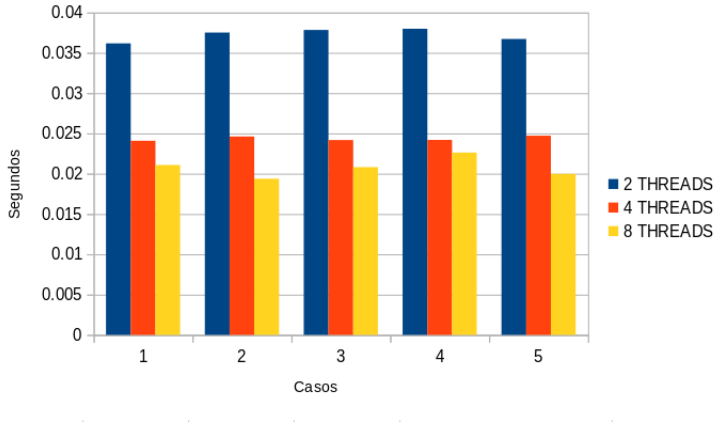
$$\text{Caso } 4 : a = 0; b = 10^8$$

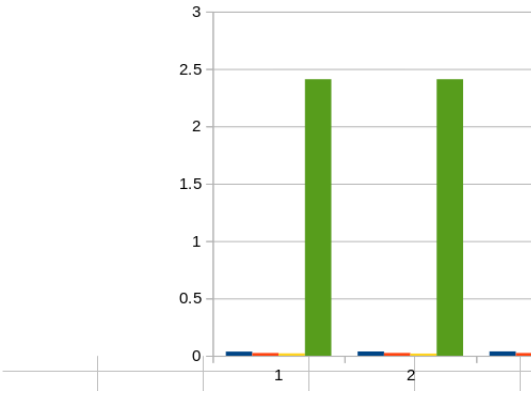
$$\text{Caso } 5 : a = 0; b = 10^9$$

Abaixo seguirão tabelas contendo os tempos de cada caso usando 2,4 ou 8 threads e na versão sequencial do programa. Além disso, para facilitar a visualização, fiz dois gráficos: no da esquerda, há a representação visual dos tempos concorrentes apenas, enquanto no da direita está também o tempo sequencial, para podermos ter uma ideia da discrepância .

$$\int x^2 \, dx$$

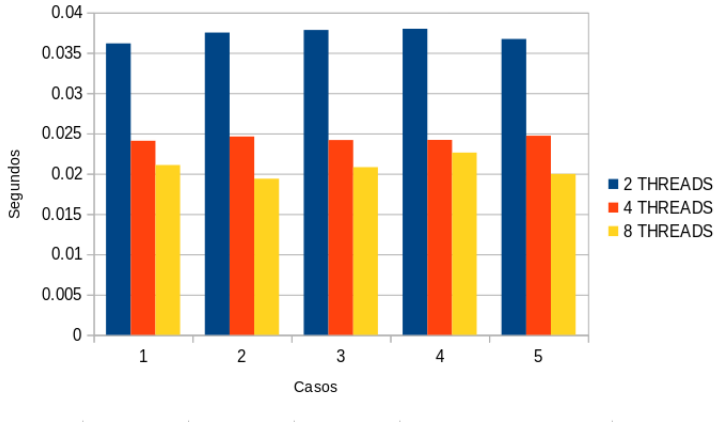
| | 2 THREADS | 4 THREADS | 8 THREADS | SEQUENCIAL |
|---|---|---|---|---|
| CASO 1 | 0.036168 | 0.024089 | 0.021078 | |
| CASO 2 | 0.037513 | 0.024619 | 0.019384 | |
| CASO 3 | 0.037839 | 0.02418 | 0.020816 | 2.41001 |
| CASO 4 | 0.037985 | 0.024197 | 0.022622 | |
| CASO 5 | 0.036717 | 0.024712 | 0.019977 | |

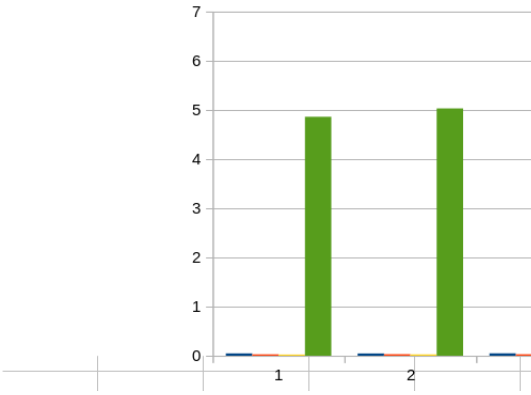


$$\int \frac{5 \cdot x^3}{10^8} + -5x \, dx$$

| | 2 THREADS | 4 THREADS | 8 THREADS | SEQUENCIAL |
|---|---|---|---|---|
| CASO 1 | 0.047809 | 0.028088 | 0.024715 | 4.860686 |
| CASO 2 | 0.045966 | 0.03088 | 0.026551 | 5.029989 |
| CASO 3 | 0.049085 | 0.029941 | 0.027491 | 5.75222 |
| CASO 4 | 0.051616 | 0.030629 | 0.027926 | 5.341694 |
| CASO 5 | 0.051268 | 0.030725 | 0.023978 | 5.38396 |



$$\int cos(x)\,dx$$

Para esse último exemplo, por ser uma integral mais difícil, tomarei valores menores:

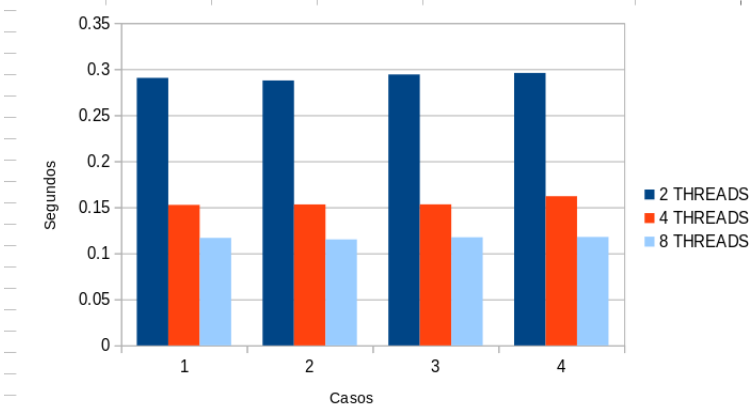
$$\text{Caso } 1 : a = 0; b = 10^3$$

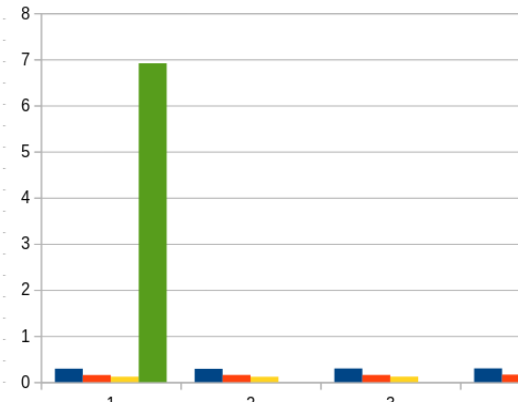
$$\text{Caso } 2 : a = 0; b = 10^4$$

$$\text{Caso } 3 : a = 0; b = 10^5$$

$$\text{Caso } 4 : a = 0; b = 10^6$$

| | 2 THREADS | 4 THREADS | 8 THREADS | SEQUENCIAL |
|---|---|---|---|---|
| CASO 1 | 0.290491 | 0.152554 | 0.116766 | 6.916 |
| CASO 2 | 0.287701 | 0.153101 | 0.115103 | |
| CASO 3 | 0.29421 | 0.153168 | 0.117362 | |
| CASO 4 | 0.295886 | 0.162021 | 0.117745 | |



## 4.1 Aceleração

Como sabemos, pela lei de Ahmdal:

$$s = \frac{T_{seq}}{T_{conc}}$$

Entao, voltando as tabelas que vimos acima , sendo $s_n$ a aceleração com **n** threads, e tomando sempre os tempos mais demorados dos resultados concorrentes (isto é, obtendo a aceleração mínima):

Para o primeiro exemplo $f(x) = x^2$:

$$s_2 \approx 6.34$$

$$s_4 \approx 99.59$$

$$s_6 \approx 106.53$$

Para o segundo exemplo $f(x) = \frac{5x^3}{10^8} - 5x$

$$s_2 \approx 94.809354763$$

$$s_4 \approx 158.199$$

$$s_6 \approx 202.71$$

Para o terceiro exemplo $f(x) = cos(x)$

$$s_2 \approx 23.80$$

$$s_4 \approx 45.33$$

$$s_6 \approx 58.73$$

## 4.2 Ambiente de Execução

Para a realização desse trabalho, foram utilizadas as seguintes configurações de máquina:

- Arquitetura: x86-64
- MOBO: Gigabyte Technology Co., Ltd. B450M DS3H V2
- Sistema Operacional: Pop!_OS 22.04 LTS
- CPU: Processador AMD Ryzen 5 3600, 3.6GHz(4.2GHz Max Turbo), Cache 32MB, 6 Núcleos, 12 Threads
- GPU: Zotac GeForce GTX 1050 Ti 4GB GDDR5 128-bit
- Memória RAM: 16GB DDR4 3000MHZ
- Armazenamento: SSD 480 GB Crucial BX500, SATA

# 5 Discussão

O ganho de desempenho definitivamente não foi esperado. Eu esperava uma aceleração no intervalo $2 < s < 10$, no máximo! Ter dado esses valores absurdos só me faz pensar que há algo de errado com minha implementação sequencial, que eu não devo ter feito da forma mais otimizada possível. Eu esperaria esses valores se eu tivesse tomado uma tolerância muito alta como $tol = 10^{-}15$, mas como nao foi o caso, não pode estar correto.
Embora a implementação concorrente tenha sido tranquila, como melhoria com certeza tem a versão sequencial do programa, mas também há a necessidade de criar uma abordagem especial para funções periódicas, pois na função $f(x) = cos(x)$, foi onde ambas as versões do programa mais engasgaram para responder e onde o resultado mais destoou do que eu chequei no Wolfram.Talvez esse último ponto que eu levantei talvez vá além da questão dos meus erros como programador, e seja até mesmo uma limitação do tema desse trabalho: funções periódicas/trigonométricas ficavam com a precisão muito reduzida quando executadas pelo programa (mas eu botei uma mesmo assim para os três exemplos não ficarem iguais). Além disso, como melhoria eu poderia criar uma biblioteca de funções em um arquivo separado para facilitar a usabilidade do programa: toda vez que eu queria trocar de função eu tinha que a sintaxe do código, ao invés de só chamar a função que eu quisesse da minha biblioteca. Uma maneira de automatizar os testes também seria muito bom num cenário com mais funções/maior número de casos de teste
A maior dificuldade que eu tive nesse trabalho foi de como fazer o que a professora indicou: rodar o programa até atingir uma determinada condição que é retornada pela concorrência! Foi diferente de tudo que vimos em aula (pelo menos na minha cabeça) e eu perdi horas tentando entender porque meu programa estava errado e o que estava acontecendo.

# 6 Referências Bibliográficas


https://www.integral-calculator.com/

https://www.wolframalpha.com

https://planetmath.org/compositetrapezoidalrule

http://homepage.math.uiowa.edu/~atkinson/ftp/ENA_Materials/Overheads/sec_5-2.pdf

https://www.whitman.edu/mathematics/calculus_online/chapter09.html

https://files.cercomp.ufg.br/weby/up/39/o/Cap%C3%ADtulo_11.pdf

https://www.matematica.pt/faq/calculo-integral-diferencial.php

https://www.colegioweb.com.br/matematica/o-que-sao-integral-e-derivada.html

https://www.desmos.com/calculator?lang=pt-BR