

# Computação Concorrente

# **Relatório**

---

Bruna Ribeiro | 118171816

Milton Quillinan | 118144649



## Introdução

- 1.** Descrição do Problema
- 2.** Projeto de Implementação da Solução Concorrente
- 3.** Casos de Teste
- 4.** Avaliação de Desempenho
- 5.** Discussão
- 6.** Referências Bibliográficas

# 1. Descrição do Problema

O problema escolhido por nosso grupo foi o de cálculo de integrais numéricas de forma retangular (também conhecido como método do ponto médio). Acreditamos que esse tema é bem interessante por ser uma das primeiras formas de calcular uma integral que aprendemos quando estudamos Cálculo I.

Primeiro, escolhemos um número de retângulos, calculamos suas áreas com a ajuda do ponto médio para definir a altura, e depois somamos os resultados.

Com o avanço do estudo, a gente percebe que, quanto maior o número de retângulos escolhidos, mais preciso é o resultado de nossa integral, e isso é essencial pro nosso problema.

## 1.1 Solução Sequencial

Para a solução sequencial, entendemos que, após o usuário digitar o número de retângulos e o intervalo que deseja, vamos fazer essa divisão e guardar cada um dos pontos que serão utilizados em um vetor. O retângulo irá de um desses pontos até o próximo, calcular área desse retângulo criado e salvar a resultante em uma variável, que será incrementada cada vez que o loop acontecer. No fim, teremos a soma completa, que é nossa integral aproximada.

Dessa forma, se nosso intervalo for  $[0,5]$ , e o número de retângulos escolhidos foram 10, quer dizer que os pontos serão  $P = \{0, 0.5, 1, \dots, 4.5, 5\}$ . Esse array será percorrido e calculado, de forma que o primeiro retângulo use  $P[1] = 0$  e  $P[i+1] = 0.5$ .



## 1.2 Dados de Entrada e Saída

Além de funções prontas para teste, para uma solução sequencial, os dados de entrada serão (1) o intervalo desejado e (2) o número de retângulos que deve ser usado no cálculo da integral.

A saída esperada será a soma das áreas desses retângulos gerados, que será a integral aproximada da função dada (lembrando que, quanto maior o número de retângulos usados, melhor será a precisão do resultado).

## 1.3 Como uma solução concorrente pode se beneficiar

Enquanto a solução sequencial fica presa pois pega os números em ordem do índice e os calcula, a concorrente pode se adiantar e fazer mais de um cálculo ao mesmo tempo, incrementando o resultado direto em uma variável compartilhada depois. Dessa forma, garantimos que o processo ocorra mais rápido e a aceleração ganha seja bem significativa quanto maior for o número de retângulos escolhidos.

## 2. Solução Concorrente

### 2.1 Estratégias

Fizemos alguns testes com duas estratégias. A primeira foi usar mutex para fazer uma divisão entre as threads. Cada uma olharia o índice compartilhado e pegaria ele para fazer os cálculos. A cada iteração do while, a thread olharia a soma global e incrementaria, usando mutex. A segunda se mostrou indiscutivelmente melhor nos testes: Ainda com a ajuda de mutex, optamos por fazer uma alternância entre threads por bloco, utilizando o compartilhamento de uma variável global soma para incrementar o cálculo de cada área, fazendo com que as threads sempre peguem o bloco, acessem a posição do vetor, e faça as contas com o retângulo a partir disso.

### 2.2 Decisões de Implementação

Teremos cinco funções que serão integradas. A escolha da função na hora dos testes será feita pelo usuário, em um menu que abrirá no início do programa. A implementação sequencial se manterá separada da concorrente, ambas sendo chamadas nos arquivos únicos de cada função, onde o cálculo do tempo e aceleração serão feitos. Teremos uma pasta separada para todos os includes necessários, abrigando as próprias funções que serão integradas, o timer, a função sequencial e a concorrente. O arquivo principal de integração irá chamar os cabeçalhos da pasta “includes” e printar, no fim, os resultados dos tempos tomados e a integração aproximada. Esse arquivo será mais amigável com o usuário, tendo uma interação melhor. Além deste, teremos outro arquivo de testes, que será usado para os testes de corretude e desempenho. Será basicamente igual ao arquivo principal, sendo sua única diferença que não terá conversa com o usuário e os argumentos dos testes serão passados em apenas uma linha. Neste arquivo, a diferença entre a corretude e o resultado da aceleração também serão printados.

### 3. Casos de Teste

Cinco funções foram usadas, então vamos alternar, em todas as 5:

- Número de threads (1, 2 e 4)
- Quantidade de retângulos ( $10^5$ ,  $10^7$  e  $10^8$ )
- Intervalos (usaremos 2 intervalos distintos)

Ou seja, teremos 90 testes distintos no total.

Para calcular a corretude, nós vamos calcular a integral no Wolfram e tomar esse resultado como base. Com isso, vamos comparar o resultado obtido com o do site e pontuar a diferença entre eles para a nossa solução Concorrente.

#### 3.1 Funções e intervalos

a.  $x^2 + x - 1$

- $[-1, 0], [5, 10]$

b.  $|3x^4|$

- $[-1, 0], [-100, 100]$

c.  $\sqrt{x^4 + 2}$

- $[-1, 50], [-99, -6]$

d.  $\sin(3x - ((6\pi + 11)/0.5))/2$

- $[-2, 2], [2.5, 3.5]$

e.  $\cos(e^{-x})(0.005x^3 + 1)$

- $[-6, -5], [-2, 5]$

## 3.2 Resultados

$$x^2 + x - 1$$

- Intervalo [-1, 0]

Resultado esperado: -1.16666...

Resultado obtido:

	1 thread	2 threads	4 threads
$10^5$	-1.16666666667500	-1.166666666667500	-1.166666666667500
$10^7$	-1.16666666666675	-1.166666666666675	-1.166666666666675
$10^8$	-1.16666666666667	-1.16666666666667	-1.16666666666667

Diferença máxima:  $\approx 0.000000000000833$

- Intervalo [5, 10]

Resultado esperado: 324.1666...

Resultado obtido:

	1 thread	2 threads	4 threads
$10^5$	324.166666665625	324.166666665625	324.166666665625
$10^7$	324.166666666666	324.166666666666	324.166666666666
$10^8$	324.166666666666	324.166666666666	324.166666666666

Diferença máxima:  $\approx 0.000000001041$

$$|3x^4|$$

- Intervalo [-1, 0]

Resultado esperado: 0.6

Resultado obtido:

	1 thread	2 threads	4 threads
$10^5$	0.5999999999500	0.5999999999500	0.5999999999500
$10^7$	0.5999999999999	0.5999999999999	0.5999999999999
$10^8$	0.6000000000000	0.6000000000000	0.6000000000000

Diferença máxima: 0.00000000005

- Intervalo [-100, 100]


Resultado esperado: 12000000000

Resultado obtido:

	1 thread	2 threads	4 threads
$10^5$	11999999600	11999999600	11999999600
$10^7$	11999999999.9600	11999999999.9600	11999999999.9600
$10^8$	11999999999.9996	11999999999.9996	11999999999.9996

Diferença máxima: 400





$$\sqrt{x^4 + 2}$$

- Intervalo [-1, 50]

Resultado esperado: 41670.20615

Resultado obtido:

	1 thread	2 threads	4 threads
$10^5$	41670.2061497586	41670.2061497586	41670.2061497586
$10^7$	41670.2061508548	41670.2061508548	41670.2061508548
$10^8$	41670.2061508549	41670.2061508549	41670.2061508549

Diferença máxima:  $\approx 0.0000002414$


- Intervalo [-99, -6]

Resultado esperado: 323361.15655

Resultado obtido:

	1 thread	2 threads	4 threads
$10^5$	323361.156546098	323361.156546098	323361.156546098
$10^7$	323361.156552801	323361.156552801	323361.156552801
$10^8$	323361.156552801	323361.156552801	323361.156552801

Diferença máxima:  $\approx 0.000003902$



$$\sin(3x - ((6\pi + 11)/0.5))/2$$

- Intervalo [-2, 2]

Resultado esperado: -0.000824398

Resultado obtido:

	1 thread	2 threads	4 threads
$10^5$	-0.0008243976655	-0.0008243976655	-0.0008243976655
$10^7$	-0.0008243976650	-0.0008243976650	-0.0008243976650
$10^8$	-0.0008243976650	-0.0008243976650	-0.0008243976650

Diferença máxima: -0.0000000003345

- Intervalo [2.5, 3.5]

Resultado esperado: -0.139705

Resultado obtido:

	1 thread	2 threads	4 threads
$10^5$	-0.1397048375955	-0.1397048375955	-0.1397048375955
$10^7$	-0.1397048375902	-0.1397048375902	-0.1397048375902
$10^8$	-0.1397048375902	-0.1397048375902	-0.1397048375902

Diferença máxima: -0.0000001624045



$$\cos(e^{-x})(0.005x^3 + 1)$$

- Intervalo [-6, -5]

Resultado esperado: 0,00152044

Resultado obtido:

	1 thread	2 threads	4 threads
$10^5$	0.001520441330107	0.001520441330107	0.001520441330107
$10^7$	0.00152044129866	0.00152044129866	0.00152044129866
$10^8$	0.00152044129866	0.00152044129866	0.00152044129866

Diferença máxima: -0.000000001330107

- Intervalo [-2, 5]

Resultado esperado: 5.30823

Resultado obtido:

	1 thread	2 threads	4 threads
$10^5$	5.3082293066427	5.3082293066427	5.3082293066427
$10^7$	5.3082293054194	5.3082293054194	5.3082293054194
$10^8$	5.3082293054192	5.3082293054192	5.3082293054192

Diferença máxima: 0.0000006933573

## 4. Avaliação de Desempenho

Para a avaliação de desempenho, vamos usar exatamente os mesmos parâmetros de teste. Ou seja, para todas as 5 funções, teremos os mesmos dois intervalos, quantidade de retângulos e threads especificados em (3).

Cada teste foi executado 5 vezes e o tempo escolhido foi o menor. A menor aceleração foi destacada em vermelho e a melhor em verde.

Para rodar o programa, foi utilizada uma máquina virtual do Linux. Suas especificações são:

**SO:** Ubuntu 64-bit

**Versão:** Workstation 16.2.x virtual machine

**RAM:** 4GB

**Processador:** AMD Ryzen 5 3600XT, Cache 35MB, 3.8GHz

$$x^2 + x - 1$$

- Intervalo [-1, 0]

Resultado obtido:

	Sequencial	1 thread	2 threads	4 threads
$10^5$	0.0060	0.0063	0.0035	0.0022
$10^7$	0.5704	0.5698	0.2870	0.1503
$10^8$	5.6998	5.7744	2.8686	1.4881

Aceleração:

	1 thread	2 threads	4 threads
$10^5$	<b>0.9523</b>	1.7142	2.7272
$10^7$	1.0010	1.9874	3.7951
$10^8$	0.9870	1.9870	<b>3.830</b>

- Intervalo [5, 10]

Resultado obtido:

	Sequencial	1 thread	2 threads	4 threads
$10^5$	0.0058	0.0066	0.0036	0.0022
$10^7$	0.5603	0.5663	0.2849	0.1475
$10^8$	5.6341	5.6254	2.8221	1.4775

Aceleração:

	1 thread	2 threads	4 threads
$10^5$	<b>0.8802</b>	1.6111	2.6363
$10^7$	0.9894	1.9666	3.7981
$10^8$	1.0016	1.9964	<b>3.8132</b>

$$|3x^4|$$

- Intervalo [-1, 0]

Resultado obtido:

	Sequencial	1 thread	2 threads	4 threads
$10^5$	0.0058	0.0062	0.0034	0.0023
$10^7$	0.5579	0.5563	0.2796	0.1487
$10^8$	5.5783	5.5611	2.7875	1.4744

Aceleração:

	1 thread	2 threads	4 threads
$10^5$	<b>0.9354</b>	1.7137	2.5217
$10^7$	1.0029	1.9953	3.7518
$10^8$	1.0030	2.0011	<b>3.7834</b>

- Intervalo [-100, 100]

Resultado obtido:

	Sequencial	1 thread	2 threads	4 threads
$10^5$	0.0057	0.0061	0.0035	0.0021
$10^7$	0.5506	0.5497	0.2830	0.1448
$10^8$	5.5387	5.4902	2.7916	1.4830

Aceleração:

	1 thread	2 threads	4 threads
$10^5$	<b>0.9286</b>	1.6285	2.7142
$10^7$	1.0015	1.9455	<b>3.8024</b>
$10^8$	1.0088	1.9840	3.7347

$$\sqrt{x^4 + 2}$$

- Intervalo [-1, 50]

Resultado obtido:

	Sequencial	1 thread	2 threads	4 threads
$10^5$	0.0061	0.0065	0.0037	0.0022
$10^7$	0.5875	0.5858	0.2951	0.1560
$10^8$	5.9813	5.8569	2.9403	1.5377

Aceleração:

	1 thread	2 threads	4 threads
$10^5$	<b>0.9285</b>	1.6410	2.7727
$10^7$	1.0029	1.9911	3.7655
$10^8$	1.0212	2.0342	<b>3.8899</b>

- Intervalo [-99, -6]

Resultado obtido:

	Sequencial	1 thread	2 threads	4 threads
$10^5$	0.0064	0.0066	0.0036	0.0022
$10^7$	0.6000	0.6003	0.3014	0.1586
$10^8$	6.0269	6.0217	3.0157	1.5859

Aceleração:

	1 thread	2 threads	4 threads
$10^5$	<b>0.9564</b>	1.8031	2.9090
$10^7$	0.9995	1.9908	3.7831
$10^8$	1.0008	1.9985	<b>3.8002</b>

$$\sin(3x - ((6\pi + 11)/0.5))/2$$

- Intervalo [-2, 2]

Resultado obtido:

	Sequencial	1 thread	2 threads	4 threads
$10^5$	0.0057	0.0065	0.0034	0.0021
$10^7$	0.5404	0.5374	0.2739	0.1405
$10^8$	5.4179	5.3859	2.7100	1.4448



Aceleração:

	1 thread	2 threads	4 threads
$10^5$	<b>0.8716</b>	1.6764	2.7142
$10^7$	1.0055	1.9731	<b>3.8441</b>
$10^8$	1.0035	1.9984	3.7498

- Intervalo [2.5, 3.5]

Resultado obtido:

	Sequencial	1 thread	2 threads	4 threads
$10^5$	0.0058	0.0061	0.0034	0.0022
$10^7$	0.5401	0.5369	0.2732	0.1404
$10^8$	5.4612	5.3703	2.7148	1.4133

Aceleração:

	1 thread	2 threads	4 threads
$10^5$	<b>0.9215</b>	1.7149	2.5354
$10^7$	1.0059	1.9769	3.8474
$10^8$	1.0169	2.0116	<b>3.8641</b>



$$\cos(e^{-x})(0.005x^3 + 1)$$

- Intervalo [-6, -5]

Resultado obtido:

	Sequencial	1 thread	2 threads	4 threads
$10^5$	0.0080	0.0083	0.0047	0.0028
$10^7$	0.7772	0.7864	0.3928	0.2106
$10^8$	8.0129	7.7559	3.9179	2.2915

Aceleração:

	1 thread	2 threads	4 threads
$10^5$	<b>0.9521</b>	1.6725	2.7931
$10^7$	0.9883	1.9786	<b>3.6904</b>
$10^8$	1.0331	2.0452	3.4968

- Intervalo [-2, 5]

Resultado obtido:

	Sequencial	1 thread	2 threads	4 threads
$10^5$	0.0073	0.0078	0.0041	0.0030
$10^7$	0.7133	0.7186	0.3713	0.1985
$10^8$	7.1784	7.1292	3.6812	2.0781



Aceleração:

	1 thread	2 threads	4 threads
$10^5$	<b>0.9373</b>	1.7651	2.4218
$10^7$	0.9925	1.9210	<b>3.5930</b>
$10^8$	1.0069	1.9500	3.4543


## 5. Discussão

Por definição, é comum o resultado da integral utilizando retângulos não ser 100% o esperado dependendo da função, exatamente por esse método ser uma resposta aproximada. Apesar disso, o resultado da corretude foi bastante satisfatório, e aconteceu o que já discutimos antes: quanto maior o número de retângulos, chegamos mais próximo do resultado, sendo os resultados mais discrepantes com o esperado exatamente os que usavam apenas  $10^5$  retângulos, e os mais próximos os que usavam  $10^8$ .

Depois de 90 entradas de testes diferentes, cada um sendo executado 5 vezes, podemos concluir que o ganho de desempenho também foi satisfatório. Apesar de mesmo com o cálculo sequencial usando  $10^8$  retângulos não passar nunca dos 10 segundos calculando, a versão concorrente se mostrou sempre melhor para 2 e 4 threads, enquanto o uso de apenas 1 thread perdia para a versão sequencial por bem pouco. Com 2 threads usadas, conseguimos, quase sempre, um pouco mais que a metade do tempo original da versão sequencial, enquanto as 4 threads eram ainda mais rápidas, fazendo em um pouco mais que  $\frac{1}{4}$  no tempo delas.

Por escolhermos intervalos relativamente aleatórios, foi comum vermos dízimas periódicas ou números grandes demais na referência de cálculo da variável. De certa forma isso é bom para testar a precisão do programa, mas por outro lado tivemos que, muitas vezes, arredondar esses resultados para uma melhor visualização de dados e cálculo da diferença máxima entre o referencial e o resultado da corretude.

A lógica do nosso programa mudou mais de uma vez. Nosso maior problema ao longo do trabalho foi achar uma boa forma da versão concorrente ser melhor que



a sequencial. Não conseguimos isso do primeiro jeito que fazíamos antes, que era deixando uma variável global de iteração disponível para cada thread pegar e usar, percorrendo o vetor de intervalos com ela. Acontece que era bem comum apenas uma thread fazer a maior parte do trabalho, não tendo uma divisão real de tarefas. Vimos muito a versão concorrente demorando mais que a sequencial, ou ver o uso de 4 threads demorando mais que 2. Quando essa lógica foi refatorada e mudamos a divisão para blocos invés de olhar uma variável global através de mutex, tudo ficou melhor, apesar de acharmos que podemos melhorar o programa ainda mais e retornar os resultados em ainda menos tempo ao mudar a versão de divisão por blocos.



## 6. Referências Bibliográficas

Wolfram:

<https://www.wolframalpha.com>

Integração Numérica:

[https://pt.wikipedia.org/wiki/Integração\\_numérica](https://pt.wikipedia.org/wiki/Integração_numérica)

Ponto médio:

[https://www.ufrgs.br/reatmat/CalculoNumerico/livro-sci/in-regras\\_de\\_newton-cotes.html](https://www.ufrgs.br/reatmat/CalculoNumerico/livro-sci/in-regras_de_newton-cotes.html)