**Projeto 2 – MPI**

**Integral Paralela**

Programa de Pós-Graduação em Ciência da Computação

Computação de Alto Desempenho

Inaê Soares de Figueiredo

Escrever um programa para o cálculo da integral da função abaixo, usando a regra do Trapézio:

A square root of a mathematical equation

Description automatically generated with medium confidence

O programa deve calcular a integral no intervalo [0,100], usando a biblioteca MPI e ser executado para as seguintes condições:

* Número de trabalhadores variando entre 1, 2, 4 e 8 threads
* Intervalos de discretização variando entre 0.000001, 0.00001 e 0.0001

1. **Solução para a integral**

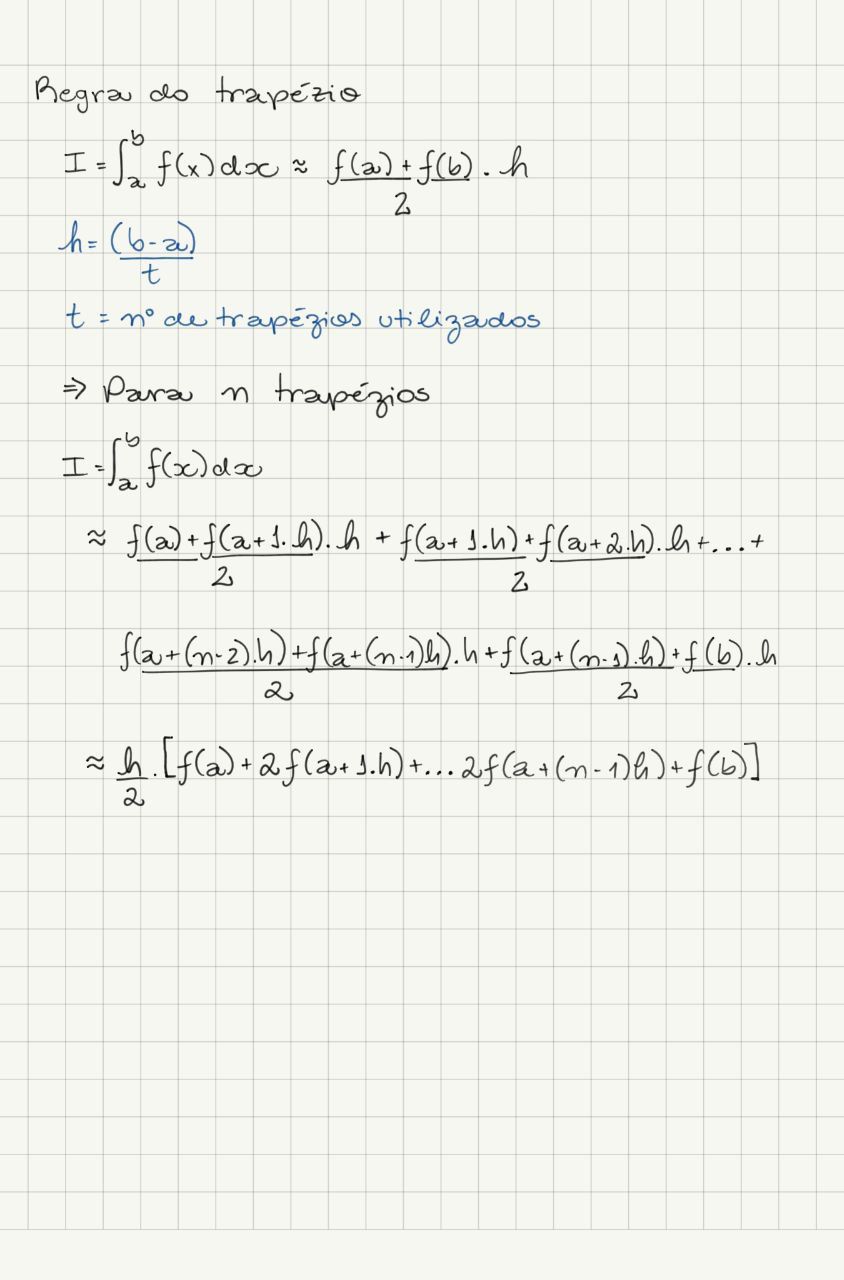


Figura 1: Simplificação da Regra do Trapézio

Seguindo esta simplificação, foi

O código completo foi enviado juntamente com este relatório e um executável do programa, mas também pode ser encontrado no GitHub através do link <https://github.com/InaeSoares/RegraTrapezio-OpenMP.git>.

1. **Utilização da MPI**

Versão

Instalação (MSMPI)

* 1. **Configurações**

Códigos desenvolvidos em linguagem C, compilador gcc versão 8.1.0, por meio do Microsoft Visual Studio Code sem otimizações, SO Windows 11 Home, processador Intel® Core TM i7 1.80GHz e 16,0 GB de memória DDR4 2.667 Ghz.

* 1. **Paralelização**

1. **Resultados**

Verificando o resultado esperado em uma calculadora de integral, para referência, o resultado obtido foi de **7.854,000000**.

7853.981630 – 0,0001, 8 proc, 4 proc, 2 proc, 1 proc

7853.981634 – 0,00001, 8 proc, 4 proc, 2 proc, 1 proc

7853.981634 – 0,000001, 8 proc, 4 proc, 2 proc, 1 proc

Foram realizados testes com intervalos diferentes e observa-se que há maior imprecisão com quantidade menor de trapézios.

Os resultados foram comparados em termos de velocidade de execução. A partir dos valores obtidos foram também calculados os valores de *Speedup* obtidos com as quantidades diferentes de *threads*.

Definindo: .

* 1. **1 processo (execução linear)**

|  |  |
| --- | --- |
| **Intervalo** | **Tempo (s)** |
| 0,0001 | 0,011494 |
| 0,00001 | 0,084144 |
| 0,000001 | 0, 794602 |

* 1. **2 processos**

|  |  |  |
| --- | --- | --- |
| **Intervalo** | **Tempo (s)** | ***Sppedup*** |
| 0,0001 | 0,003869 | 2,9708 |
| 0,00001 | 0,041791 | 2,0134 |
| 0,000001 | 0,393156 | 2,0211 |

* 1. **4 processos**

|  |  |  |
| --- | --- | --- |
| **Intervalo** | **Tempo (s)** | ***Sppedup*** |
| 0,0001 | 0,002969 | 3,8713 |
| 0,00001 | 0,025627 | 3,2834 |
| 0,000001 | 0,240904 | 3,2984 |

* 1. **8 processos**

|  |  |  |
| --- | --- | --- |
| **Intervalo** | **Tempo (s)** | ***Sppedup*** |
| 0,0001 | 0,001461 | 7,8672 |
| 0,00001 | 0,016069 | 5,2364 |
| 0,000001 | 0,154740 | 5,1351 |

Observa-se pelos resultados apresentados que há um grande impacto da paralelização no tempo de execução dos dois códigos apresentados, com apenas duas instâncias em que o tempo de execução diminuiu em menos do que a metade.

A utilização de 8 *threads* resultou nos melhores tempos de execução.

1. **Materiais de apoio**