**Projeto 1 – openMP**

**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 openMP 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**

Seguindo a regra do trapézio (Figura 1), foi estruturada uma solução inicial para o problema, em linguagem de programação C.

A solução inicial (Código 1) apresenta diversas etapas dentro do loop For, calculando os pontos xa e xb para cada iteração, a função nos pontos xa e xb, aplicando a regra do trapézio e somando o resultado ao valor final da integral. Nesta versão do código, o loop inicia em i=0, para que a função f(0) seja o ponto inicial de aplicação da regra do trapézio.

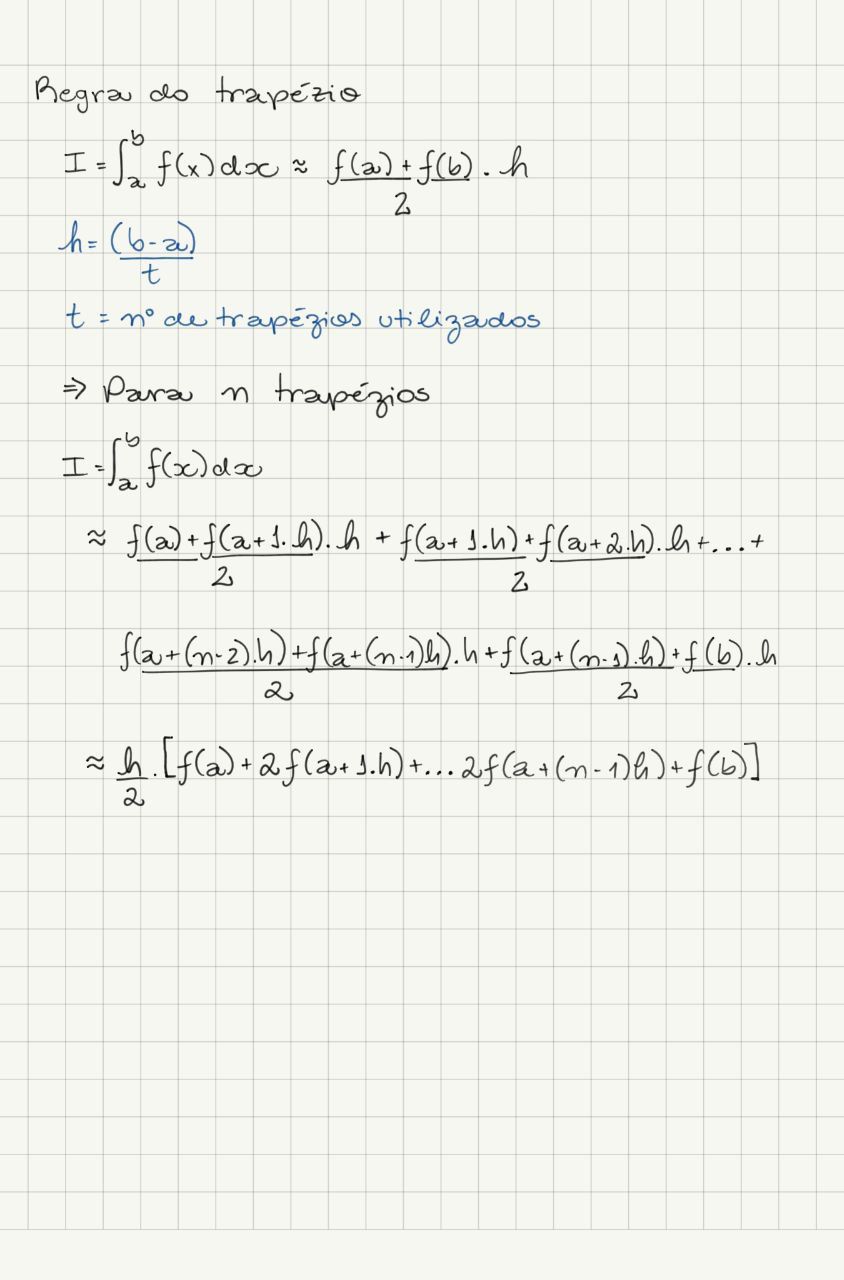


Figura 1: Regra do Trapézio

A screenshot of a computer program

Description automatically generated with medium confidenceNo entanto, pensando em otimizar o código e simplificar o conteúdo do loop, observa-se que a Regra do Trapézio pode ser simplificada da maneira disposta na Figura 2.

Código 1: Primeira implementação da regra do trapézio

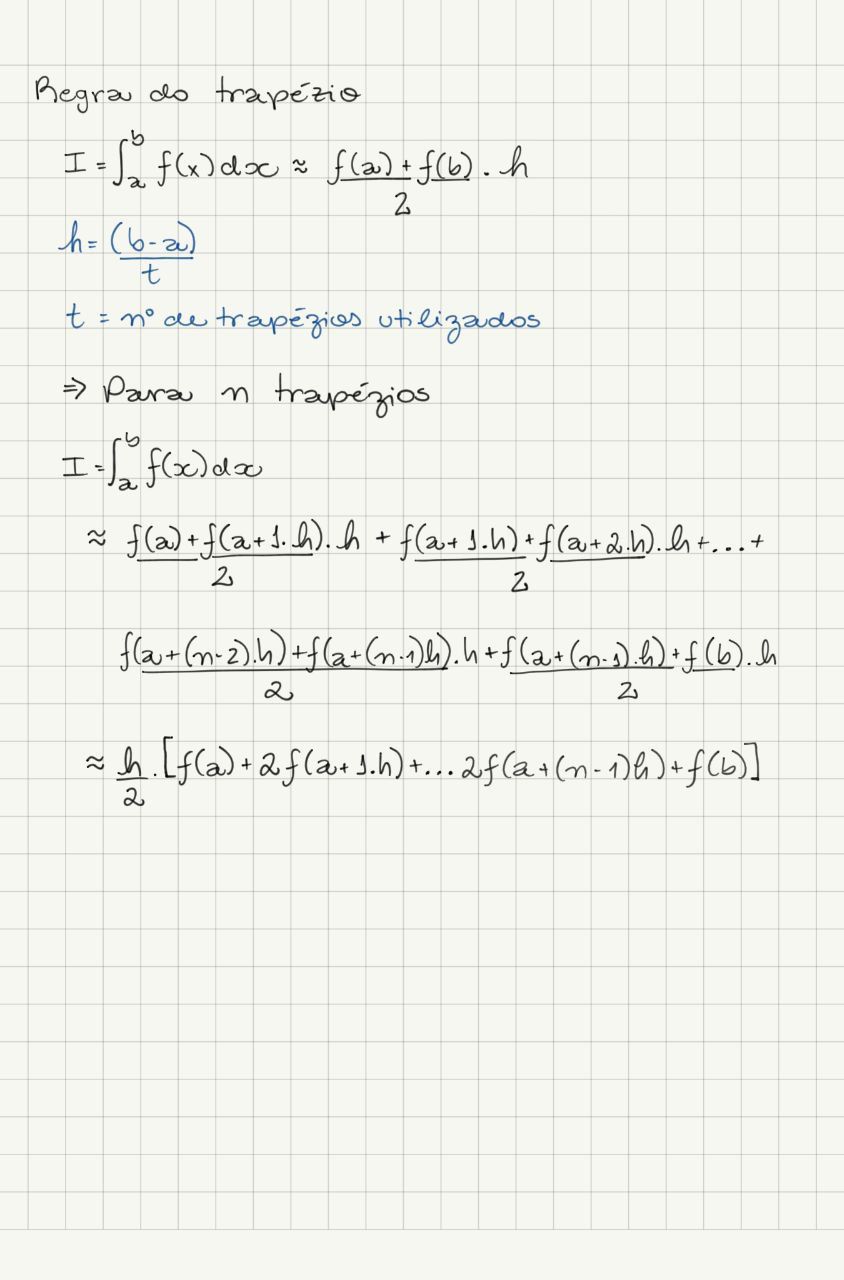
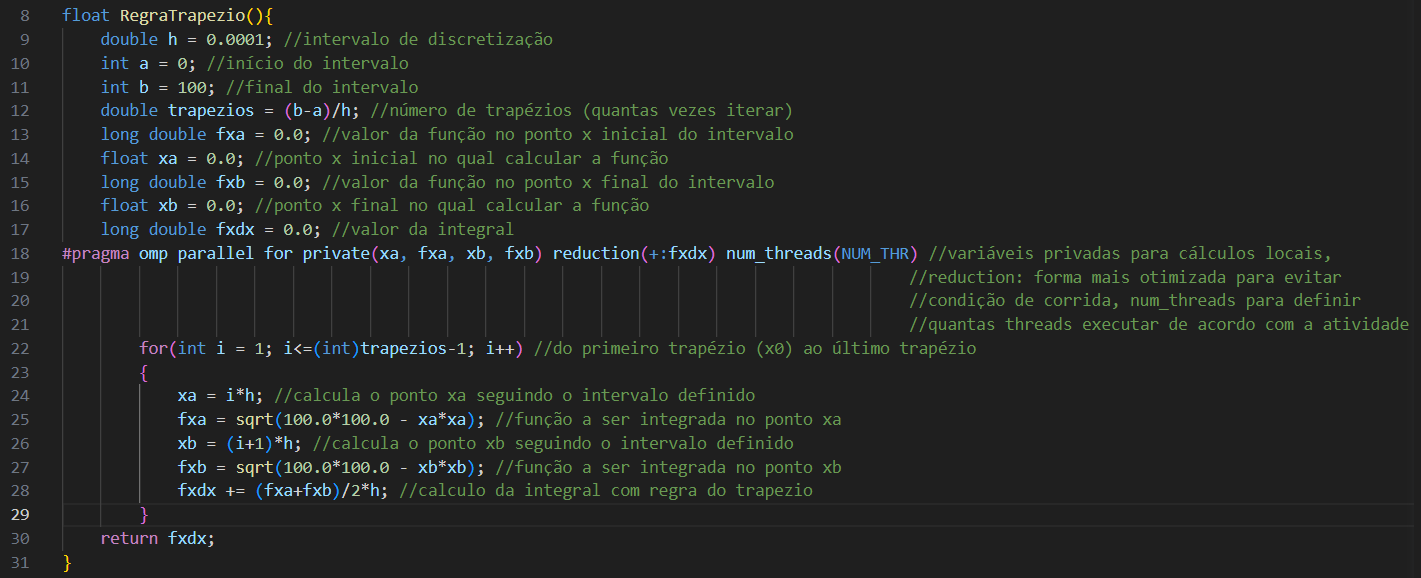
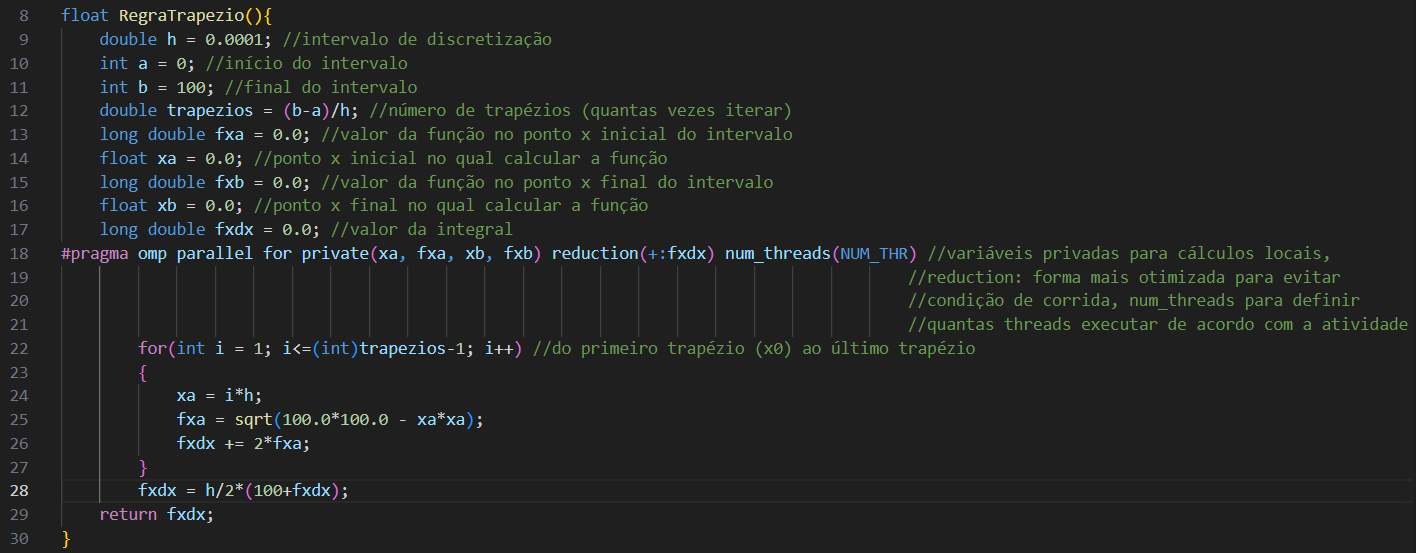


Figura 2: Simplificação da Regra do Trapézio

Seguindo esta simplificação e considerando: f(a) = 100 ( = = 100) e f(b) = 0 ( = = 0), foi desenvolvida uma nova versão da regra do trapézio (Código 2). Nesta versão do código, o loop é iniciado em i=1, já que f(0) deve ser somada apenas uma vez ao total das funções e isso é feito fora do loop.



Código 2: segunda implementação da regra do trapézio

Como disposto na Seção 3, houve ganho na velocidade de execução ao adotar o novo método, com os resultados dos cálculos permanecendo iguais.

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 OpenMP**

A OpenMP foi utilizada para realizar a paralelização do código, como se pode ver nas duas versões apresentadas anteriormente (Código 1 e Código 2). As configurações do hardware e das ferramentas utilizadas são explicadas a seguir.

* 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**

A paralelização do programa foi configurada de acordo com a linha de código a seguir:

#pragma omp parallel for private(xa, fxa, xb, fxb) reduction(+:fxdx) num\_threads(NUM\_THR),

Sendo que as definições seguem os seguintes propósitos:

* **private(xa, fxa, xb, fxb):** as variáveis são definidas como *private* para que, mesmo que todas as *threads* tenham que utilizá-las, cada uma tenha sua própria no momento de execução;
* **reduction(+:fxdx):** forma mais otimizada de realizar cálculos recorrentes sobre uma única variável, sem a necessidade de utilizar regiões críticas e evitando falso compartilhamento;
* **num\_threads(NUM\_THR):** definição de quantas threads devem executar a seção de código paralela.

1. **Resultados**

Verificando os resultados obtidos, observa-se claros impactos da paralelização no tempo de execução dos códigos, assim como diferenças entre Código 1 e Código 2.

Observa-se também que as especificações de intervalo de discretização não são suficientes para causar diferenças de cálculo com a precisão de casas decimais utilizada.

O valor obtido para a integral foi de **7.853,981445** em todos os testes executados, e verificando o resultado esperado em uma calculadora de integral, para referência, o resultado obtido foi de **7.854,000000**.

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 *thread* (execução linear)**

|  |  |  |
| --- | --- | --- |
| **Intervalo** | **Código 1 – Tempo (s)** | **Código 2 – Tempo (s)** |
| 0,0001 | 0,036 | 0,020 |
| 0,00001 | 0,330 | 0,197 |
| 0,000001 | 3,239 | 1,734 |

* 1. **2 *threads***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Intervalo** | **Código 1 – Tempo (s)** | ***SppedupC1*** | **Código 2 – Tempo (s)** | ***SpeedupC2*** |
| 0,0001 | 0,018 | 2 | 0,012 | 1,6667 |
| 0,00001 | 0,161 | 2,0497 | 0,098 | 2,0102 |
| 0,000001 | 1,576 | 2,0552 | 0,874 | 1,984 |

* 1. **4 *threads***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Intervalo** | **Código 1 – Tempo (s)** | ***SppedupC1*** | **Código 2 – Tempo (s)** | ***SpeedupC2*** |
| 0,0001 | 0,013 | 2,7692 | 0,004 | 5 |
| 0,00001 | 0,085 | 3,8824 | 0,051 | 3,8627 |
| 0,000001 | 0,813 | 3,9840 | 0,452 | 3,8363 |

* 1. **8 *threads***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Intervalo** | **Código 1 – Tempo (s)** | ***SppedupC1*** | **Código 2 – Tempo (s)** | ***SpeedupC2*** |
| 0,0001 | 0,007 | 5,1429 | 0,005 | 4 |
| 0,00001 | 0,051 | 6,4706 | 0,038 | 5,1842 |
| 0,000001 | 0,449 | 7,2138 | 0,287 | 6,0418 |

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. Em todos os outros caso, o *Speedup* foi maior que 2.

A utilização de 8 *threads* resultou nos melhores tempos de execução, com resultados piores do que os obtidos com 4 *threads* em apenas uma situação (Código 2, intervalo 0,0001).

O efeito do processamento em paralelo foi maior sobre o Código 1, como pode-se observar pelos maiores valores de *Speedup*, mas o Código 2 teve a execução mais rápida em todos os testes.

1. **Materiais de apoio**

Intel Modern Code Partner Openmp – Aula 01. Disponível em <<https://www.inf.ufrgs.br/gppd/intel-modern-code/slides/workshop-2/MCP_Pt2_Pratica.pdf>>.

OpenMP 3.1 API C/C++ Syntax Quick Reference Card. Disponível em < <https://www.openmp.org/wp-content/uploads/OpenMP3.1-CCard.pdf>>.

Using OpenMP With C. Disponível em < <https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html>>.

OpenMP Application Programming Interface. Disponível em <<https://www.openmp.org/spec-html/5.0/openmp.html>>.