

## Projeto 2 – MPI

### Integral Paralela

Programa de Pós-Graduação em Ciência da Computação

Computação de Alto Desempenho

Inaê Soares de Figueiredo

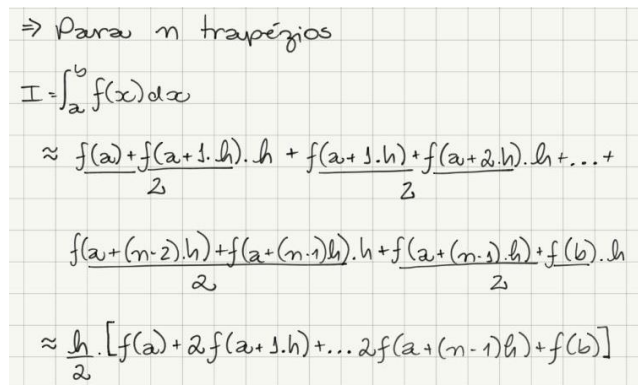
Escrever um programa para o cálculo da integral da função abaixo, usando a regra do Trapézio:

$$f(x) = \sqrt{100^2 - x^2}$$

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

- Número de processos 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



⇒ Para  $n$  trapézios

$$I = \int_a^b f(x) dx$$
$$\approx \frac{f(a) + f(a+1 \cdot h)}{2} \cdot h + \frac{f(a+1 \cdot h) + f(a+2 \cdot h)}{2} \cdot h + \dots +$$
$$\frac{f(a+(n-2) \cdot h) + f(a+(n-1) \cdot h)}{2} \cdot h + \frac{f(a+(n-1) \cdot h) + f(b)}{2} \cdot h$$
$$\approx \frac{h}{2} \cdot [f(a) + 2f(a+1 \cdot h) + \dots + 2f(a+(n-1) \cdot h) + f(b)]$$

Figura 1: Simplificação da Regra do Trapézio

Seguindo a regra do trapézio, a função a ser calculada foi simplificada de acordo com o que está disposto na Figura 1.

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

#### 2. Utilização da MPI

MPI (Message Passing Interface) é um padrão de comunicação entre ambientes que realizam troca de mensagens, como clusters e redes. Neste padrão, as tarefas são

divididas em partes e distribuídas entre os processadores de uma mesma máquina, ou de máquinas conectadas a uma mesma rede.

O MPI pode ser utilizado com diversas linguagens de programação (C, C++, Fortran, Python, entre outras) e para os propósitos deste trabalho foi aplicado em linguagem C.

Existem diversas implementações destes padrões de comunicação, como o MS-MPI, desenvolvido e mantido pela Microsoft para programação paralela em ambientes com sistema operacional Windows. MS-MPI é a implementação utilizada neste trabalho, em sua versão 10.1.3.

#### a. 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.

#### b. Paralelização

A paralelização foi feita utilizando 1, 2, 4 e 8 processos, e a junção dos resultados de cada processador foi realizada com a função a seguir:

```
MPI_Reduce(&integral_local, &resultado_final, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

A função `MPI_Reduce` define uma variável comum a todos os processos (`integral_local`) que deve ser somada em uma variável local ao processo 0 (`resultado_final`). O tipo da variável é definido como *double* (`MPI_DOUBLE`) e a função de junção dos valores de `integral_local` é a soma (`MPI_SUM`). A soma é realizada no processador raiz (0).

Como a definição do tempo de execução não pode ser feita de forma direta e linear na execução paralela do código, o tempo de execução foi estimado de acordo com o tempo do processo mais demorado do programa, considerado o processo limitante quanto à velocidade de execução. Este tempo foi utilizado para cálculo da métrica de *Speedup*, definida na Seção 3.

### 3. Resultados

Verificando o resultado esperado para a integral a ser calculada neste projeto em uma calculadora de integral, para referência, o resultado obtido foi de **7.854,000000**.

Os resultados obtidos com o programa desenvolvido foram: **7853.981630**, para precisão de 0,0001, e **7853.981634** para as precisões de 0,00001 e 0,000001. Entende-se que o tipo de dados (*double*) utilizado no programa não apresenta precisão suficiente para diferenciar os resultados com discretizações de 0,00001 e 0,000001. Tentativas de utilizar outros tipos de dados resultaram em erros de *overflow* e, como o valor obtido não afeta a análise de performance, manteve-se o código com tipos *double*.

Foram realizados testes com discretizações maiores (1, 25, 50...), para verificação do código, e observa-se que há maior imprecisão nos resultados com quantidades menores de trapézios.

Os resultados obtidos 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*.

$$\text{Definindo: } Speedup = \frac{\text{Tempo execução linear}}{\text{Tempo execução em } p \text{ processadores}}.$$

**a. 1 processo (execução linear)**

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

**b. 2 processos**

Intervalo	Tempo (s)	<i>Speedup</i>
0,0001	0,003869	2,9708
0,00001	0,041791	2,0134
0,000001	0,393156	2,0211

**c. 4 processos**

Intervalo	Tempo (s)	<i>Speedup</i>
0,0001	0,002969	3,8713
0,00001	0,025627	3,2834
0,000001	0,240904	3,2984

**d. 8 processos**

Intervalo	Tempo (s)	<i>Speedup</i>
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 do código desenvolvido, e é possível verificar que, de forma consistente, a velocidade aumenta de acordo com a quantidade de processos utilizados.

Analisando por quantidade de processos, observa-se também uma queda no *speedup* quando diminui-se o intervalo de discretização e, conseqüentemente, aumenta-se a quantidade de *loops* de cálculo da integral.

A utilização de 8 processos resultou nos menores tempos de execução e nos maiores *speedups*.

Uma comparação com os resultados obtidos na solução do mesmo problema de paralelização com a utilização de openMP (<https://github.com/InaeSoares/RegraTrapezio-OpenMP>) mostram que MPI resulta em tempos de execução consideravelmente menores, e em uma maior consistência nestas

melhorias (2 processos são mais rápidos que 1 processo, 4 são mais rápidos que 2 e 8 são mais rápidos que 8).

#### **4. Materiais de apoio**

Compile MPI and OpenMP Programs with VS Code in Windows. Disponível em <[https://www.youtube.com/watch?v=T\\_BVqSya1Is](https://www.youtube.com/watch?v=T_BVqSya1Is)>. Acesso em 09 julh. 2023.

How to setup VS Code for MPI based on C | Parallel Programming in VS Code | MS-MPI setup in Windows. Disponível em <<https://www.youtube.com/watch?v=bkfCrj-rBjU>>. Acesso em 09 julh. 2023.

Introdução ao MPI. Disponível em <<https://www.dcce.ibilce.unesp.br/~aleardo/cursos/hpc/mpi2023.pdf>>. Acesso em 10 julh. 2023.

Introdução ao MPI. Disponível em <<https://www.lncc.br/~borges/ist/SO2/trabalhos/Introducao%20ao%20MPI.pdf>>. Acesso em 12 julh. 2023.

Microsoft MPI. Disponível em <<https://learn.microsoft.com/pt-br/message-passing-interface/microsoft-mpi>>. Acesso em 10 julh. 2023.

OpenMPI v4.1.5 documentation. Disponível em <<https://www.open-mpi.org/doc/current/>>. Acesso em 14 julh. 2023.