Ibilce - Instituto de Biociências, Letras e Ciências Exatas - Câmpus de São José do Rio Preto - Unesp

Regras dos Trapézios com MPI

Grupo:
- Leandro Aguiar Mota
- Mateus Rosolem Baroni
- Yhan de Brito Pena

Novembro
2024

Conteúdo

1 Introdução

A integração numérica nos permite calcular o valor aproximado de uma integral definida sem conhecer uma expressão analítica para a sua primitiva. Assim, integrar numericamente uma função $y = f(x)$ num intervalo [a, b] pode ser o mesmo que integrar um polinômio $P_n(x)$ aproxima $f(x)$ em um determinado intervalo.

A aproximação mais simples possível, que pode ser calculada sem dificuldade, seria a área do trapézio definido pelos pontos (a, $f(a)$) e (b, $f(b)$) , onde $f(a)$ e $f(b)$ são as bases e (b a) é a altura. Porém, o erro desta aproximação ainda é muito alto, dessa forma, uma estratégia para melhorar a qualidade da aproximação é dividir o intervalo de integração em diversos subintervalos menores, aproximando a integral em cada um desses subintervalos pela área dos respectivos trapézios.

Para implementação desse projeto foi utilizada a biblioteca MPI(Message Passing Interface) para realizar o devido processo de paralelização da execução do método dos trapézios. A MPI é um padrão para comunicação entre processos em sistemas paralelos e distribuídos, sendo amplamente usada em computação de alto desempenho (HPC). Seu funcionamento é baseado no conceito de passagem de mensagens, onde processos independentes trocam informações para realizar cálculos colaborativamente.

Dessa forma, nesse projeto iremos realizar a execução do métodos dos trapézios para integração numérica utilizando a biblioteca MPI para paralelização dos cálculos, bem como iremos fazer as medições de desempenho para diferentes combinações de parâmetros que seram analisados ao final do relatório.

2 Desenvolvimento

2.1 Problema

O problema deste projeto era basicamente realizar benchmarks do métodos dos trapézios para a seguinte integral dupla:

$$\int_0^{1.5} \int_0^{1.5} \sin(x^2 + y^2)$$

Assim, para realização dos benchmarks, foram considerados três parâmetros e seus respectivos valores:

- **Quantidade de MPI Cores:** 1, 2, 4 e 8 MPI Cores.
- **Quantidade de intervalos no eixo x:** 10^3, 10^4 e 10^5.
- **Quantidade de intervalos no eixo y:** 10^3, 10^4 e 10^5.

2.2 Solução

Na introdução foi feita uma breve apresentação da regra dos trapézios, nessa seção será feita um aprofundamento de como se realiza esse método para integração dupla.

Essa abordagem pode ser explicada para uma **integral dupla**, a ideia da regra do trapézio pode ser estendida para trabalhar com duas variáveis. No caso, consideramos a integral definida sobre uma região retangular no plano xy:

$$\iint_R f(x, y)\, dx\, dy,$$

essa regra permite calcular áreas ou volumes aproximados sobre domínios bidimensionais ao subdividir o domínio [a,b]×[c,d] em uma malha regular.

O domínio R é subdividido em $n \times m$ sub-retângulos menores, com dimensões $\Delta x = \frac{b-a}{n}$ e $\Delta y = \frac{d-c}{m}$. Cada sub-retângulo é definido por $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$, onde:

$$x_i = a + i \cdot \Delta x \quad \text{e} \quad y_j = c + j \cdot \Delta y.$$

Em vez de calcular a função $f(x, y)$ diretamente nos vértices do sub-retângulo, utilizamos o ponto médio do sub-retângulo (x_m, y_m), dado por:

$$x_m = \frac{x_i + x_{i+1}}{2}, \quad y_m = \frac{y_j + y_{j+1}}{2}.$$

A integral dupla é então aproximada pela soma ponderada dos valores de $f(x, y)$ calculados nos pontos dessa malha:

$$\iint_R f(x, y)\, dx\, dy \approx \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} \cdot f(x_i, y_j),$$

onde w_{ij} são os pesos que variam de acordo com a posição do ponto (x_i, y_j) na malha. Sendo assim, os pesos são definidos da seguinte forma:

- $w_{ij} = \frac{1}{2}$ para os pontos nas bordas;
- $w_{ij} = 1$ para os pontos internos;

Assim, a regra dos trapézios para integrais duplas é uma técnica eficiente e intuitiva para aproximação de áreas ou volumes, especialmente útil quando a função $f(x, y)$ é conhecida apenas em pontos discretos ou quando uma solução analítica para a integral não está disponível.

3 Descrição de atividades

Como já foi dito no início, nesse projeto utilizamos para fazer a paralelização dos cálculos a biblioteca MPI, sendo assim fizemos no código a divisão do número de trapézios que cada processo ficou responsável pelos cálculos e assim calculamos as integrais duplas locais e depois utilizamos da diretiva de

comunicação coletiva fornecida pela MPI, o MPI_Reduce, dessa forma realizamos a soma de todas os resultados locais das integrais duplas. O código pode ser ilustrado abaixo:

```
Função Trapezoidal_Rule(a, b, c, d, nx, ny, rank, size):
    hx ← (b - a) / nx
    hy ← (d - c) / ny
    integral ← 0.0
    local_int ← 0.0

    rows_per_work ← nx / size
    start ← rank × rows_per_work
    end ← Se rank = size - 1 então nx senão start + rows_per_work

    Para i de start até end faça:
        Para j de 0 até ny faça:
            xi ← a + i × hx
            yj ← c + j × hy
            se i == 0 ou j == 0 ou i == nx - 1 or j == ny - 1 então
                local_integral ← local_integral + f(xi, yj) x 0.5
            senão
                local_integral ← local_integral + f(xi, yj)
        Fim para
    Fim para

    MPI_Reduce(local_int, integral, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD)

    Retorne integral × hx × hy
Fim Função
```

Com isso, cercamos a execução do método dos trapézios com a função MPI_Wtime da MPI, que é utilizada para medir o tempo decorrido em programas paralelos MPI. Ela retorna o tempo atual em segundos desde um

ponto arbitrário no passado, que depende da implementação MPI, medindo assim o tempo de execução do método para cada combinação de parâmetros.

Dessa forma, para cada combinação de parâmetros que foi mostrada anteriormente, rodamos o código 10 vezes e fizemos a média dos tempos de execução extraídos, para assim, obter uma medida mais confiável de cada combinação para o método dos trapézios. Com isso, após fazer todas as medições, geramos uma tabela demonstrativa de cada combinação, como também gráficos para podermos fazer as análises dos resultados de forma eficiente.

4 Análise dos Resultados

Com as informações obtidas nessa tabelas, foi possível a produção de um gráfico calculando SpeedUp para cada combinação do projeto e também um gráfico analisando os tempos de execução das mesmas, além da disso conseguimos fazer uma tabelas mostrando as porcentagens de eficiências.

Ressaltando aqui as especificações da máquina de onde foram executados esses *Benchmarks*:

- **Processador:** Intel Core I5 com 8 Cores
- **Memória RAM:** 16 GB de RAM DDR4

Para extrair os dados de eficiência da Tabela 2, utilizamos a seguinte fórmula para cálculo da eficiência:

$$E = \frac{SpeedUp}{Threads}$$

Bem como o SpeedUp foi calculado utilizando a seguinte fórmula:

$$SpeedUp = \frac{T_{serial}}{T_{paralelo}}$$

Processos	**x**	**y**	**tempo(s)**	**Resultado**
1	1000	1000	0.01895	1.39892
1	1000	10000	0.09806	1.39921
1	1000	100000	0.86951	1.39924
1	10000	1000	0.09543	1.39921
1	10000	10000	0.83756	1.39950
1	10000	100000	8.56669	1.39952
1	100000	1000	0.86087	1.39924
1	100000	10000	8.44908	1.39952
1	100000	100000	84.32632	1.39955
2	1000	1000	0.00998	1.39892
2	1000	10000	0.05547	1.39921
2	1000	100000	0.44427	1.39924
2	10000	1000	0.05263	1.39921
2	10000	10000	0.44657	1.39950
2	10000	100000	4.55121	1.39952
2	100000	1000	0.44634	1.39924
2	100000	10000	4.55101	1.39952
2	100000	100000	45.07870	1.39960
4	1000	1000	0.00900	1.39890
4	1000	10000	0.03350	1.39920
4	1000	100000	0.24930	1.39920
4	10000	1000	0.03130	1.39920
4	10000	10000	0.24790	1.39950
4	10000	100000	2.47580	1.39950
4	100000	1000	0.24620	1.39920
4	100000	10000	2.43560	1.39950
4	100000	100000	25.56010	1.39960
8	1000	1000	0.00670	1.39890
8	1000	10000	0.03600	1.39920
8	1000	100000	0.21110	1.39920
8	10000	1000	0.03840	1.39920
8	10000	10000	0.20370	1.39950
8	10000	100000	1.76500	1.39950
8	100000	1000	0.24660	1.39920
8	100000	10000	1.98090	1.39950
8	100000	100000	21.18100	1.39960

Tabela 1: Tabela dos resultados para cada combinação

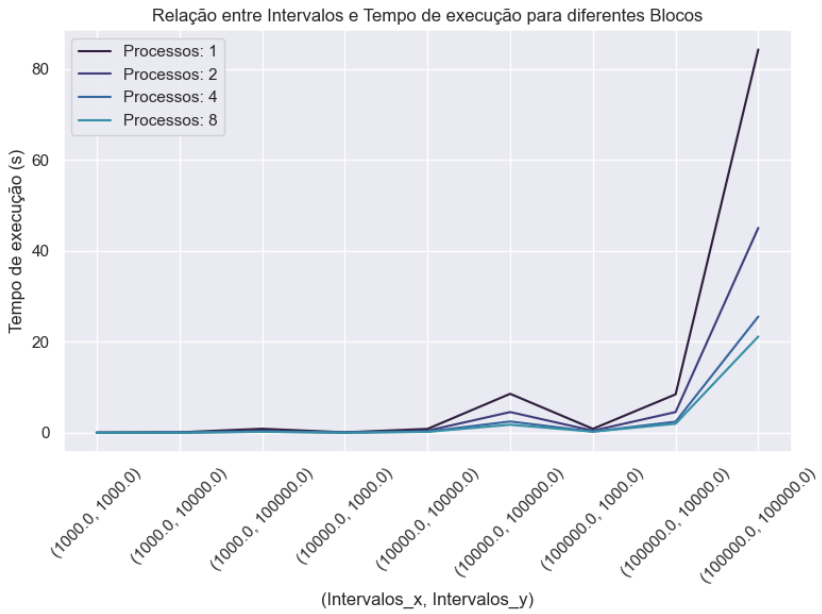

Figura 1: Gráfico do tempo de execução para diferentes combinações

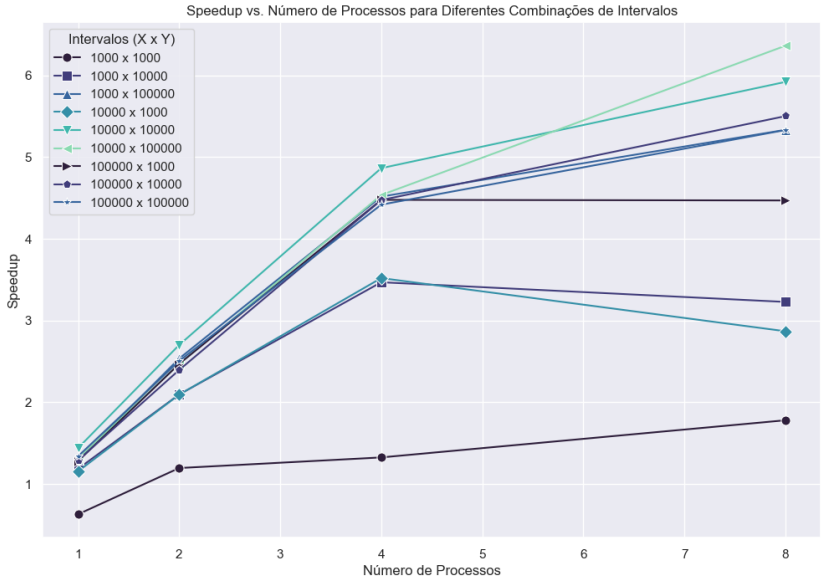

Figura 2: Gráfico de speedup para diferentes combinações

Após analisar os dados tanto das tabelas quanto dos gráficos, podemos perceber que ao observarmos a Figura 1, que explicita os tempos de execução ao longo dos intervalos, que para valores menores de intervalos a diferença

nos tempos são mínimas, o que nos diz que para esses intervalos não seria necessária a paralelização do método. Porém, ao aumentar o tamanho do problema o número de processos começa a fazer efeito e percebe-se facilmente que quanto mais aumentamos (até 8 processos) melhor fica o tempo de execução do programa, sendo 8 processos o melhor nesse quesito.

Ademais, olhando agora para a Figura 2, o gráfico que nos mostra o SpeedUp para diferentes tipos de combinações variando no eixo x pelo número de processos, conseguimos perceber que de 1 a 4 processos para a maioria dos problemas temos quase que um SpeedUp linear, mostrando que paralelismo está fazendo sentido, com um único "outlier"que é para o problema com 1000x1000 em que o SpeedUp se mantem praticamente constante conforme aumentou-se o número de processos, o que provavelmente ocorreu por algum tipo de overhead gerado pelo fato do problema ser muito pequeno nessa combinação.

Sendo assim, observando as figuras juntamente com a Tabela 2, podemos observar que apesar do método rodando com 8 processos estar nos dando um tempo de execução um pouco melhor que para 4 processos, seus valores de eficiência são muito ruins se comparados com os outros números de processos, o que nos fez pensar que esse não é o melhor ponto ótimo de paralelização do método.

Para complementar essa análise, e reforçar que para 4 processos o problema se situa com um paralelismo mais eficiente, na Figura 2, podemos ver que o SpeedUp de 4 para 8 processos uma perda relativamente grande na maioria dos casos, assim não parece ser o melhor número para extrair paralelismo para o problema proposto.

Processos	**x**	**y**	**Eficiência**
1	1000	1000	0.62918206%
1	1000	10000	1.18497858%
1	1000	100000	1.29503628%
1	10000	1000	1.15444829%
1	10000	10000	1.43991953%
1	10000	100000	1.31101767%
1	100000	1000	1.28038031%
1	100000	10000	1.28991263%
1	100000	100000	1.33920877%
2	1000	1000	0.59734469%
2	1000	10000	1.047404%
2	1000	100000	1.26730029%
2	10000	1000	1.0466369%
2	10000	10000	1.3503135%
2	10000	100000	1.23385671%
2	100000	1000	1.2347549%
2	100000	10000	1.19737981%
2	100000	100000	1.25259321%
4	1000	1000	0.33119444%
4	1000	10000	0.86715672%
4	1000	100000	1.12920878%
4	10000	1000	0.87994409%
4	10000	10000	1.21623538%
4	10000	100000	1.13408615%
4	100000	1000	1.11925366%
4	100000	10000	1.11867456%
4	100000	100000	1.10455893%
8	1000	1000	0.22244403%
8	1000	10000	0.40346875%
8	1000	100000	0.66677345%
8	10000	1000	0.35862305%
8	10000	10000	0.74007057%
8	10000	100000	0.79540241%
8	100000	1000	0.55871908%
8	100000	10000	0.68772875%
8	100000	100000	0.66646137%

Tabela 2: Tabela das eficiências por combinação

5 Conclusão

Após a análise detalhada dos resultados obtidos na implementação do método dos trapézios utilizando MPI, podemos concluir que a paralelização do método demonstrou diferentes níveis de eficácia dependendo das dimensões do problema e do número de processos utilizados. Para problemas de menor dimensão, especialmente na combinação 1000 x 1000, a paralelização não apresentou ganhos significativos, mantendo um SpeedUp praticamente constante devido ao overhead gerado pela comunicação entre processos.

A análise revelou que o ponto ótimo de paralelização foi alcançado com 4 processos, onde se observou um speedup quase linear na maioria dos casos, demonstrando um equilíbrio ideal entre ganho de desempenho e eficiência computacional. Embora a configuração com 8 processos tenha apresentado os menores tempos de execução em alguns casos, seus valores de eficiência foram significativamente inferiores quando comparados com as outras configurações, indicando um aproveitamento quase ótimo dos recursos.

Esta conclusão é reforçada pela análise do SpeedUp, que mostrou uma perda considerável de desempenho na transição de 4 para 8 processos na maioria dos casos analisados. O overhead de comunicação entre processos e a própria natureza do problema contribuíram para esta diminuição na eficiência. A implementação demonstrou ser uma solução eficiente para o cálculo de integrais duplas, especialmente quando aplicada a problemas de dimensões apropriadas com o número adequado de processos