Relatório sobre a utilização de MPI para integração dupla

Otávio Augusto Teixeira
Luan Marques Batista

IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO

Introdução

O método dos trapézios é uma técnica amplamente utilizada para a aproximação numérica de integrais, permitindo a resolução de integrais simples e duplas de forma eficiente. Neste relatório, aplicaremos o método dos trapézios para calcular uma integral dupla da função $f(x, y) = sin(x^2 + y^2)$, definida sobre a região $[0, 1.5] \times [0, 1.5]$, utilizando a biblioteca MPI para paralelização.

O MPI (Message Passing Interface) é um padrão amplamente utilizado para comunicação entre processos em sistemas de computação paralela. Ele permite que programas distribuam tarefas entre múltiplos núcleos ou máquinas, maximizando o desempenho em aplicações que requerem alto poder computacional. Por meio de uma interface padronizada, MPI fornece funcionalidades para troca de mensagens entre processos, sincronização, e controle de execução paralela, garantindo que os dados sejam compartilhados de forma eficiente mesmo em sistemas distribuídos.

Neste estudo, serão realizados testes com as seguintes configurações:

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

Ao final, o relatório apresentará o tempo de execução para cada cenário, utilizando tabelas e gráficos para visualizar o impacto do paralelismo e da granularidade dos intervalos. O código fonte da aplicação será entregue junto ao relatório, garantindo que os experimentos possam ser replicados.

Resultados

O nosso código utiliza o MPI para distribuir os cálculos do método dos trapézios entre múltiplos processos, permitindo uma execução paralela eficiente. Ele aceita como entrada o número de intervalos de discretização em x e y, além do número de núcleos MPI (especificado via linha de comando com mpiexec). A divisão do trabalho é feita atribuindo diferentes subconjuntos das linhas de discretização em x a cada processo, que calcula localmente sua contribuição para a integral. Posteriormente, os resultados locais são combinados usando uma operação de redução coletiva para obter o valor total da integral.

Código usado para buildar:

mpicc -o trabalho3 Trabalho3.c -lm

Código usado para executar:

mpiexec -np <num_de_cores> ./trabalho3 <num_eixo_x> <num_eixo_y>

Tabela de tempos obtida após a execução do código para cada caso acima:
Obs: Cada caso foi testado 5 vezes e, após isso, peguei a mediana para melhor representação dos dados

	Quantidades de Cores			
Quantidade de intervalos eixo x e y	1 MPI Core	2 MPI Core	4 MPI Core	8 MPI Core
$x = 10^3$ $y = 10^3$	0,117s	0,106s	0,105s	0,114s
$x = 10^3$ $y = 10^4$	0,445s	0,272s	0,203s	0,190s
$x = 10^3$ $y = 10^5$	3,819s	2,022s	1,107s	0,675s
$x = 10^4$ $y = 10^3$	0,452s	0,278s	0,213s	0,196s
$x = 10^4$ $y = 10^4$	3,788s	2,008s	1,085s	0,701s
$x = 10^4$ $y = 10^5$	37,120s	19,347s	9,949s	5,709s
$x = 10^5$ $y = 10^3$	3,779s	2,010s	1,084s	0,684s
$x = 10^5$ $y = 10^4$	37,031s	19,326s	9,949s	5,741s
$x = 10^5$ $y = 10^5$	6m 10,292s	3m 12,715s	1m 38,438s	55,789s

Tabela dos Speedups comparando a quantidade de MPI cores com os intervalos, a base do speedup foi 1 MPI core:

Speedups				
Quantidade de intervalos eixo x e y	1 MPI Core Base	2 MPI Core	4 MPI Core	8 MPI Core
$x = 10^3$ $y = 10^3$	1,0	1,10	1,11	1,02
$x = 10^3$ $y = 10^4$	1,0	1,63	2,19	2,34
$x = 10^3$ $y = 10^5$	1,0	1,88	3,44	5,65
$x = 10^4$ $y = 10^3$	1,0	1,62	2,12	2,30
$x = 10^4$ $y = 10^4$	1,0	1,88	3,49	5,40
$x = 10^4$ $y = 10^5$	1,0	1,91	3,73	6,50
$x = 10^5$ $y = 10^3$	1,0	1,88	3,48	5,52
$x = 10^5$ $y = 10^4$	1,0	1,91	3,72	6,45
$x = 10^5$ $y = 10^5$	1,0	1,92	3,76	6,63

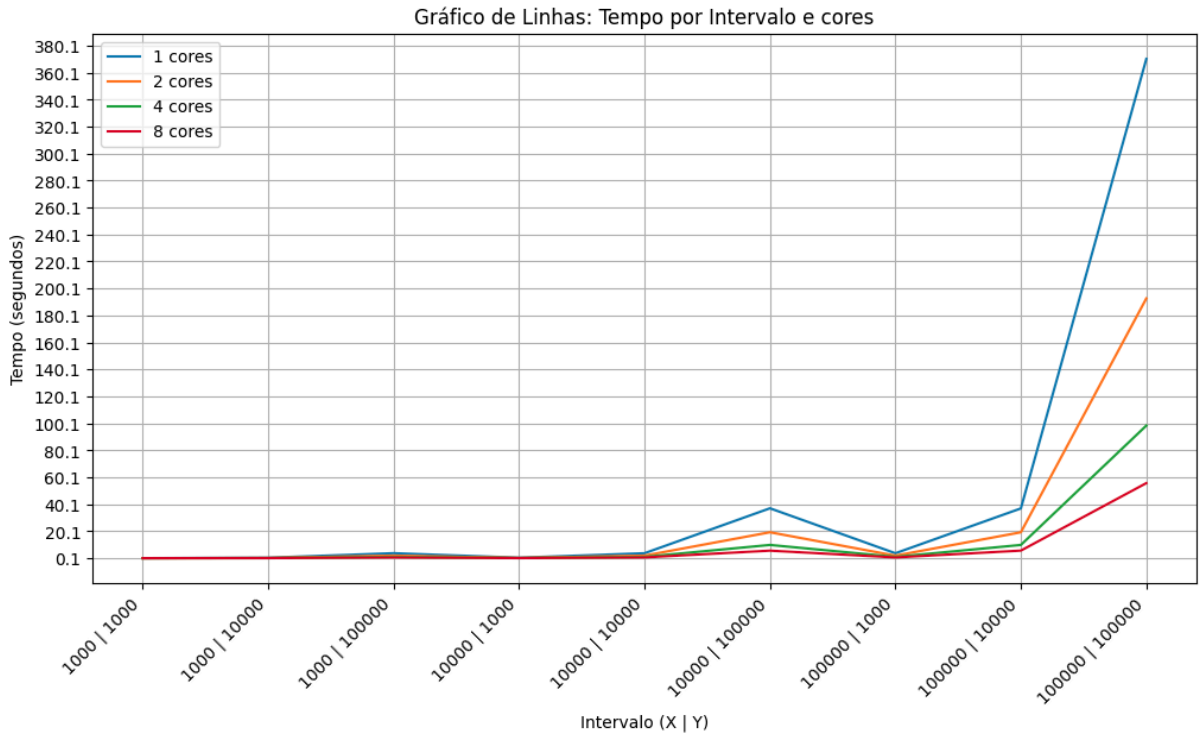
Análise das tabelas

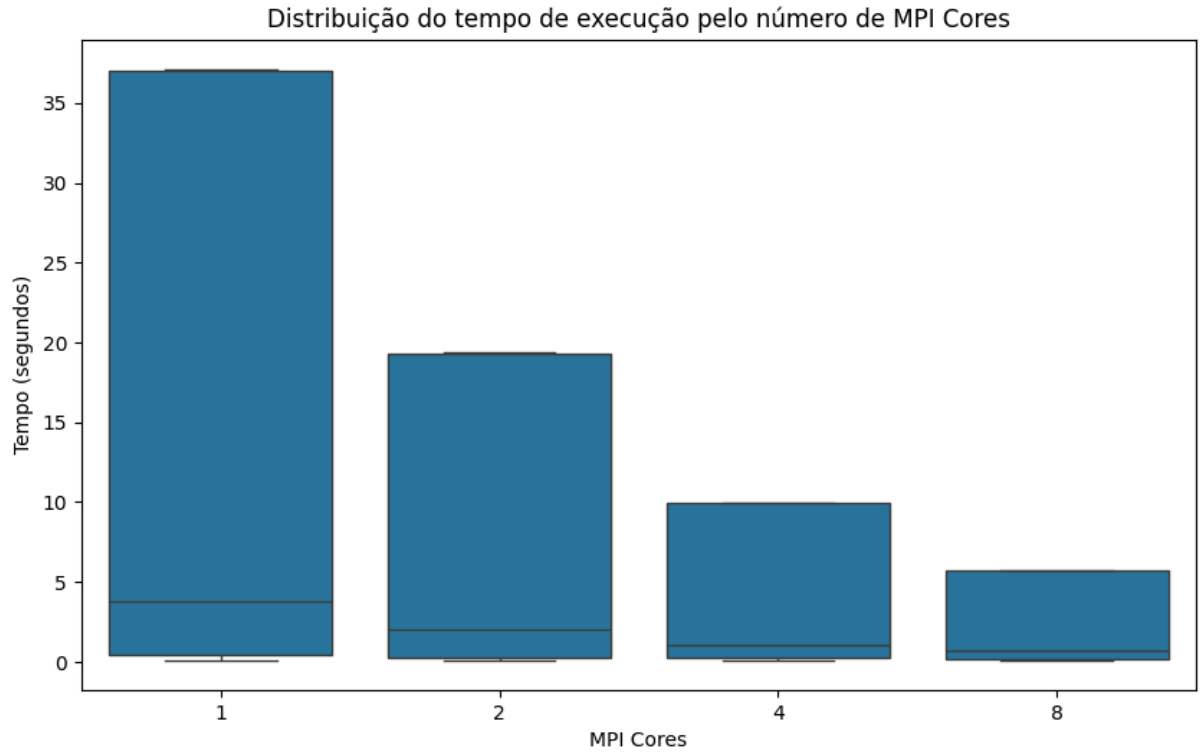
As tabelas fornecem uma visão clara sobre o impacto da paralelização utilizando o MPI para diferentes tamanhos de problema e números de MPI cores. Em problema pequenos (10^3), observamos uma pequena melhora de desempenho entre 1, 2 e 4 cores, porém um tempo quase igual é observado para 8 cores, o que indica um possível overhead, já que o problema é muito pequeno. Com o aumento da granularidade do trabalho, a divisão entre os núcleos começa a mostrar ganhos substanciais. A redução do tempo entre 1 e 8 núcleos é significativa, especialmente em problemas como x=10^3, y=10^4. Nos problemas grandes com 10^5, os tempos de execução para 1 núcleo chegam a vários minutos, enquanto a divisão com 8 núcleos reduz esse tempo drasticamente para segundos em alguns casos. Isso reflete a eficiência da distribuição do cálculo, embora ainda seja limitado por fatores como o balanceamento de carga e o custo de agregação dos resultados.

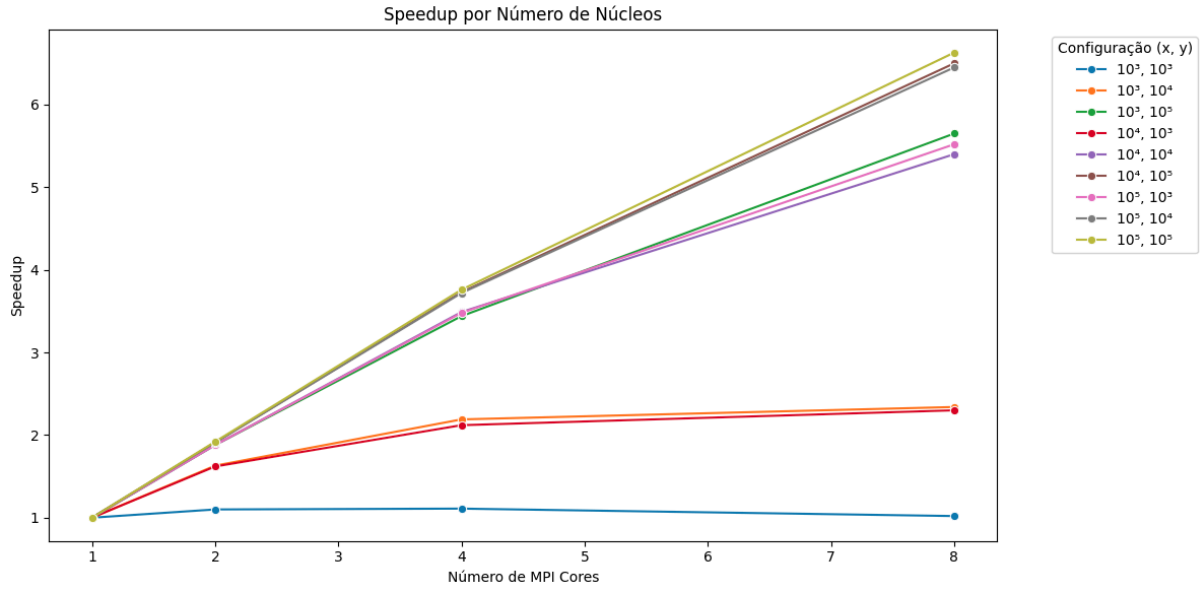
A tabela de Speedup reflete de maneira mais palpável a vantagem de se usar mais núcleos MPI para os problemas médio e grandes, com 8 núcleos MPI chegamos a um pico de 6,63 em relação a 1 núcleo.

Gráficos

Os gráficos abaixo foram plotados usando seaborn, matplotlib, pandas e numpy. As imagens do gráfico estão junto com o código fonte e os resultados do benchmark







No primeiro gráfico, observamos que, para problemas de menor escala, as diferenças nos tempos de execução entre diferentes números de MPI cores são insignificantes. Contudo, à medida que o tamanho dos problemas aumenta, a vantagem do uso de um maior número de núcleos MPI torna-se evidente. Essa tendência é reforçada pelo segundo gráfico, um boxplot dos tempos de execução, que demonstra claramente que a utilização de 8 núcleos é significativamente mais eficiente para problemas de maior complexidade.

No terceiro gráfico, que ilustra os speedups para diferentes intervalos de tempo, observamos a formação de quatro grupos distintos. Cada grupo representa um aumento no tamanho do problema, evidenciado pela elevação do speedup à medida que o número de núcleos MPI aumenta. Isso reflete a eficiência crescente proporcionada pelo uso de mais núcleos, especialmente em problemas de maior complexidade.

Conclusão

O projeto demonstrou com sucesso a eficácia da paralelização com MPI para o cálculo da integral dupla da função $\sin(x^2 + y^2)$ usando o método dos trapézios. A análise dos resultados, tanto nas tabelas quanto nos gráficos, confirma que o aumento do número de núcleos MPI reduz significativamente o tempo de execução, especialmente para problemas de maior granularidade (intervalos maiores em x e y). Em suma, o projeto demonstra a viabilidade e os benefícios da paralelização com MPI, fornecendo uma base sólida para futuras explorações e otimizações.