RELATÓRIO SOBRE O USO DA BIBLIOTECA MPI PARA O CÁLCULO DE INTEGRAL

Gabryella Mika Tanigawa
gabryella.mika@unesp.br
Júlia Mendes
julia.mendes23@unesp.br

1 Introdução

MPI (Message Passing Interface) é um protocolo de comunicação restrito ao modelo SPMD (Single Program Multiple Data) usado para programação paralela em redes, clusters ou máquinas massivamente paralelas. Foi criado em 1991, a partir de fóruns de pesquisadores, com o auxílio de mais de 40 empresas da área de sistemas distribuídos.

As funções da biblioteca MPI delimitam o escopo do programa que será paralelizado e executado por cada um dos processos. E por meio da troca de mensagens entre os mesmos, é possível obter o resultado da integral dupla, somando as áreas dos trapézios.

1.1 Objetivos

O presente relatório tem como objetivo central:

- Descrever o tempo de execução que resulta da combinação dos critérios determinados pela quantidade de intervalos no eixo x, pela quantidade de intervalos no eixo y e pelo número de *MPI Cores*;
- Realizar uma sucessão de *benchmarks* conforme o número de *MPI Cores* varia.

2 Metodologia

Para a realização dos testes, foi utilizada uma máquina com 8 núcleos físicos, com o objetivo de obter o poder de processamento necessário. Foi utilizada a implementação de código aberto, **Open MPI**, em Linux, para compilar e executar o programa MPI em C. O número de *Cores* foi especificado na linha do comando de execução com **mpiexec**. E o comando **time** foi utilizado para medir o tempo total decorrido desde o início até o final da execução. Utilizou-se o tempo total para a análise do desempenho.

3 Resultado

Esta seção é dedicada à análise das tabelas e gráficos de regressão linear como forma de avaliação do desempenho do número de *Cores* utilizados. As tabelas fornecem os resultados

exatos, enquanto os gráficos oferecem uma visão dos padrões e tendências dos dados. Os primeiros resultados (tempo de execução em segundos) estão nas Tabelas 1 - 4.

1 MPI Core				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.400	1.158	9.620
	10^4	1.469	9.665	92.514
	10^5	9.613	91.846	908.149

Tabela 1: Tempo de execução em segundos para 1 *Core*

2 MPI Cores				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.444	0.744	4.043
	10^4	0.748	3.882	35.721
	10^5	4.139	36.197	377.116

Tabela 2: Tempo de execução em segundos para 2 *Cores*

4 MPI Cores				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.433	0.593	2.363
	10^4	0.610	2.467	21.298
	10^5	2.376	20.516	236.607

Tabela 3: Tempo de execução em segundos para 4 *Cores*

8 MPI Cores				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.509	0.618	1.711
	10^4	0.564	1.680	14.135
	10^5	1.658	13.532	193.374

Tabela 4: Tempo de execução em segundos para 8 *Cores*

A partir da tabela, nota-se que a implementação com 8 *Cores* obteve quase todos os menores tempos de execução, o que indica maior desempenho. Já a implementação com 1 *Core* obteve os maiores valores de execução, apresentando o pior desempenho. Sobre os valores em relação aos outros critérios, uma maior quantidade de intervalos em x e y demanda maior tempo para executar, enquanto as implementações com menores valores no x e y executaram em menos

tempo. Observa-se que os valores dos dois menores intervalos foram menores na implementação com 4 *Cores*, uma vez que, o custo de comunicação aumenta com a quantidade de *Cores* e o número de intervalos por processo é pequeno. Assim, é possível ver nos gráficos das Figura 1 - 4, o desempenho das implementações variando o número de intervalos e de *Cores*.

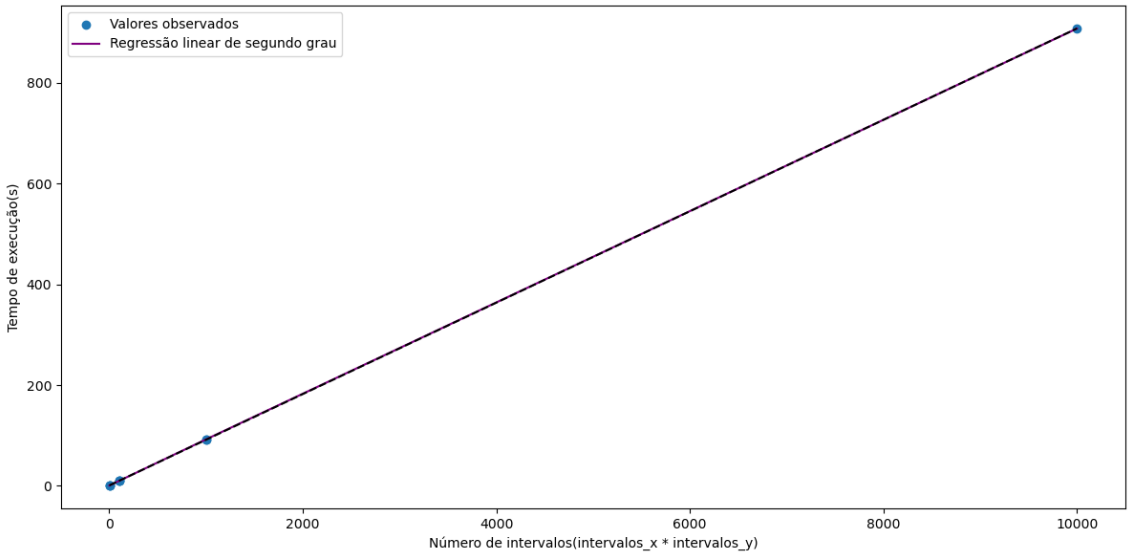

Figura 1: Redução Linear para 1 *Core*

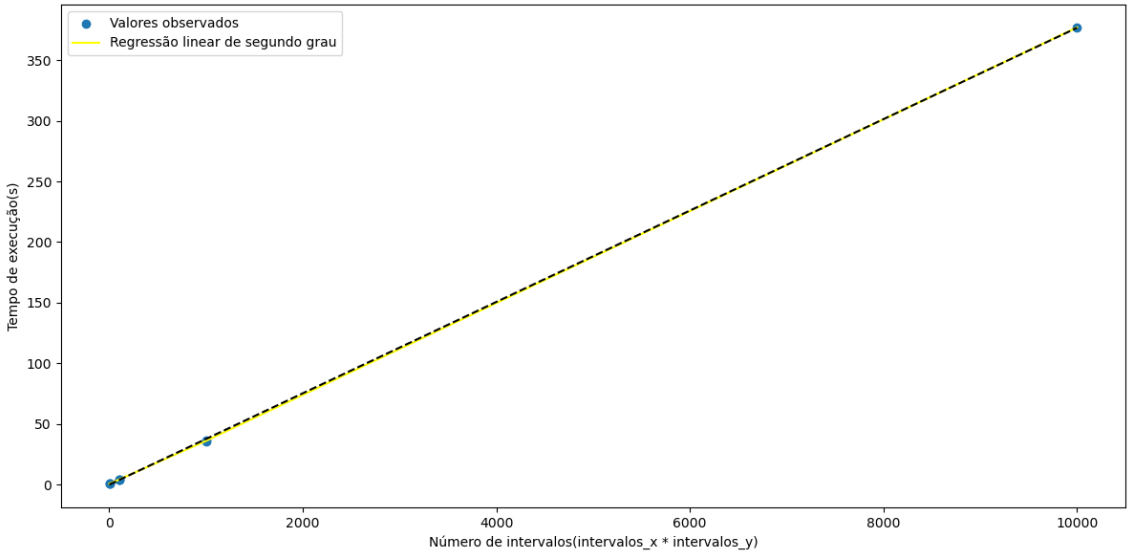

Figura 2: Redução Linear para 2 *Cores*

Analisando os gráficos, é possível ter uma percepção errônea de que o tempo de execução aumenta de forma constante com o número de intervalos em x e y. Entretanto, ao traçar a reta que interliga os dados e calcular a regressão linear, fica evidente que as duas variáveis apresentam comportamentos distintos. A curva da parábola abre para cima conforme o número de *Cores* aumenta, exceto pela implementação com 1 *Core*, que possui um coeficiente negativo, apontando que, após um certo número de intervalos, poderá ter um *overhead*.

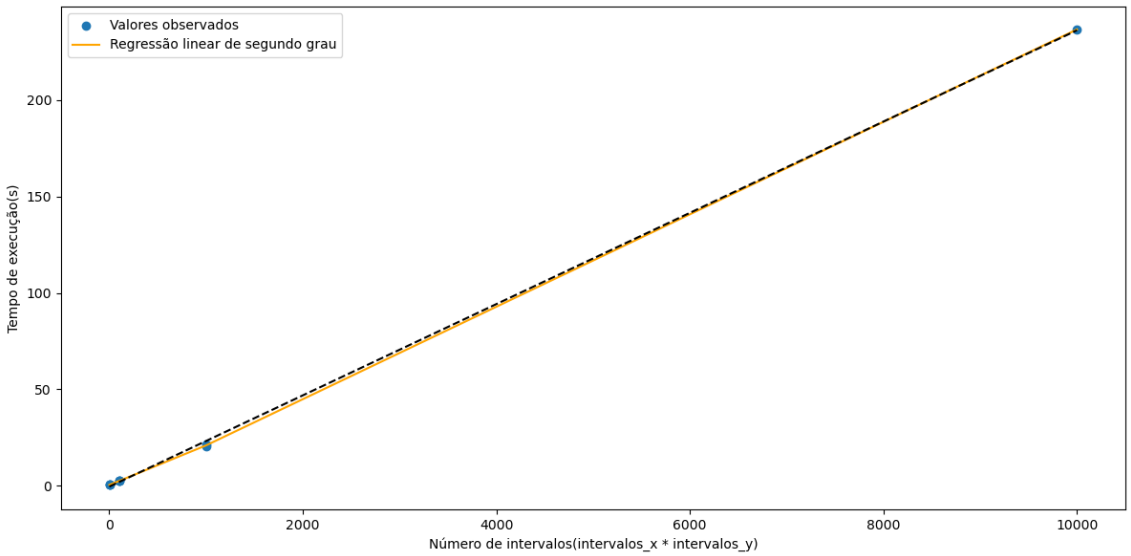

Figura 3: Redução Linear para 4 *Cores*

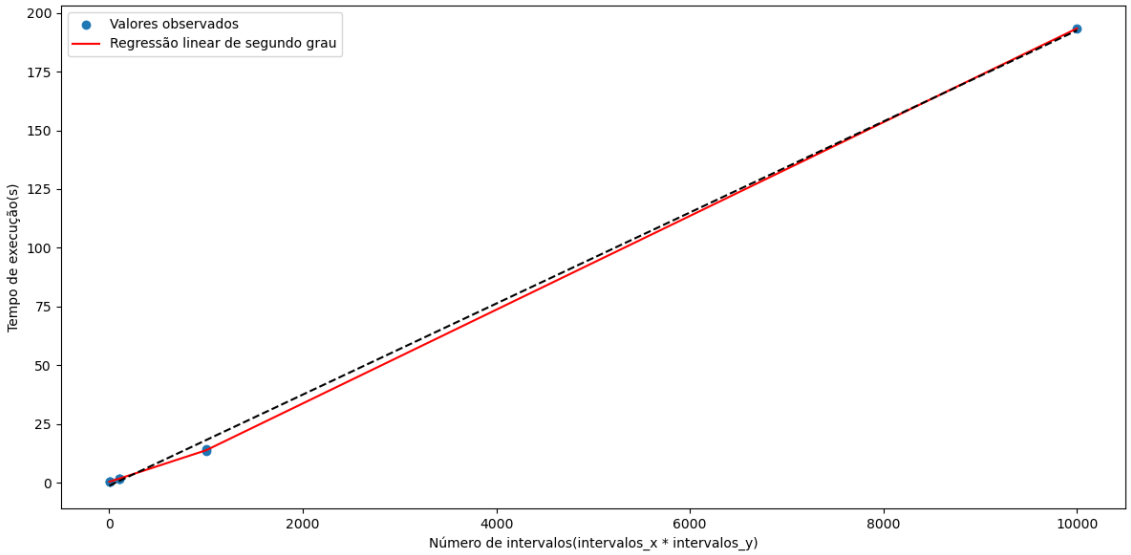

Figura 4: Redução Linear para 8 *Cores*

4 Conclusão

Por fim, como indicado pela Figura 5, a implementação com 8 *Cores* demonstrou o melhor desempenho no cálculo da integral pelo método dos trapézios, apresentando os menores tempos de execução. A implementação com 4 *Cores* teve um desempenho ligeiramente abaixo. E, finalmente, o desempenho com 1 *Core* foi ruim, como esperado, devido à maior eficiência proporcionada pelo uso de múltiplas *MPI Cores*, que permite a execução em menos tempo e melhora significativamente o desempenho do algoritmo.

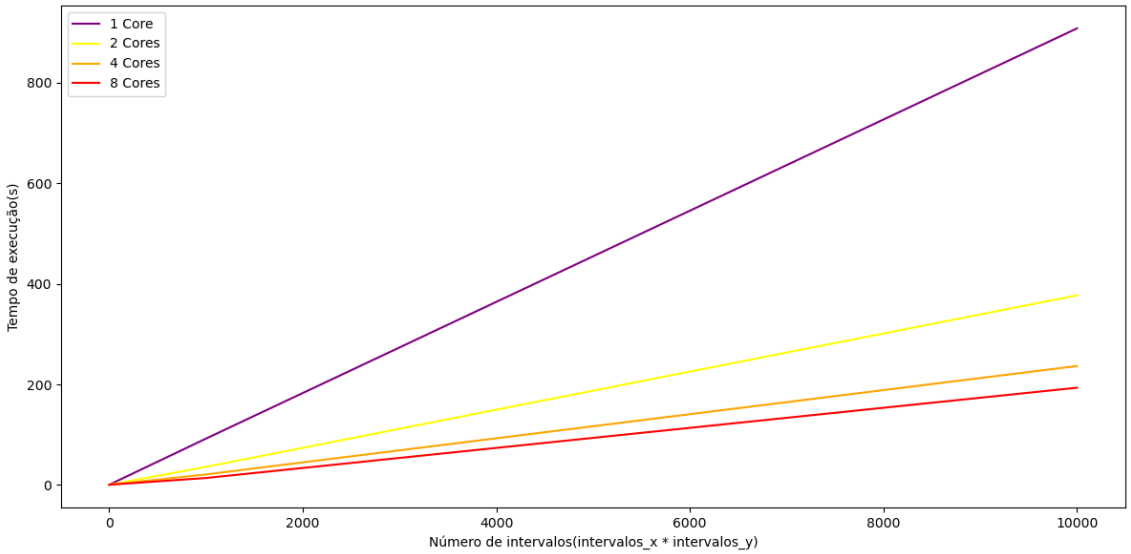

Figura 5: Comparação das Reduções Lineares