RELATÓRIO DA IMPLEMENTAÇÃO COM OPENMP PARA O CÁLCULO DE INTEGRAL

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

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 números de *threads*;
- Analisar o desempenho conforme o número de *threads* aumenta.

2 Metodologia

Usou-se o comando **time**, disponível em sistemas operacionais Unix, para medir o tempo de execução da implementação. Ele é utilizado antes do comando de execução e calcula o tempo que o processador opera para executar as instruções do programa sem o tempo de espera, o tempo para executar as instruções para realizar chamadas de sistema e, finalmente, 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 *threads* 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.

A partir da tabela, nota-se que a implementação com 8 *threads* obteve os menores tempos de execução, o que indica maior desempenho. Já a implementação com 1 *thread* 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. Porém, a quantidade maior de intervalos no eixo y deve ter impactado nos resultados, à vista que, as iterações do eixo x foram executadas em paralelo e o loop para o eixo y foi executado integralmente por cada *thread*. Assim, é possível ver no gráfico da Figura 4 como os

1 thread				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.052	0.515	4.442
	10^4	0.447	4.452	44.567
	10^5	4.673	45.419	458.088

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

2 threads				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.040	0.296	2.930
	10^4	0.278	2.479	26.734
	10^5	2.425	25.147	291.704

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

4 threads				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.049	0.291	2.457
	10^4	0.249	2.407	24.272
	10^5	2.365	24.307	260.502

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

8 threads				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.049	0.281	2.454
	10^4	0.248	2.312	24.321
	10^5	2.290	23.803	259.506

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

melhores resultados são os de 8 *threads*. Ademais, é evidente que o pior desempenho foi de 1 *thread*, que é visível no seu gráfico na Figura 1.

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 reta, embora útil, não captura adequadamente a tendência dos dados, que aumentam de forma quadrática. Por outro lado, a parábola, ao ajustar-se aos dados, possui uma curvatura que minimiza os erros e representa melhor a relação observada. Dessa forma, o modelo preditivo parabólico se revela mais eficaz do que o modelo

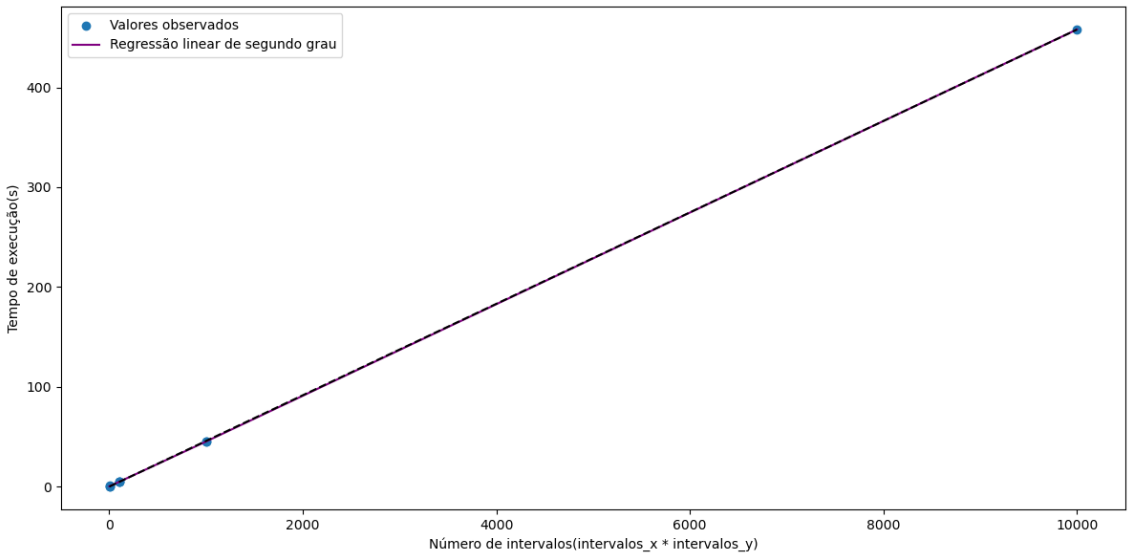

Figura 1: Redução Linear para 1 *thread*

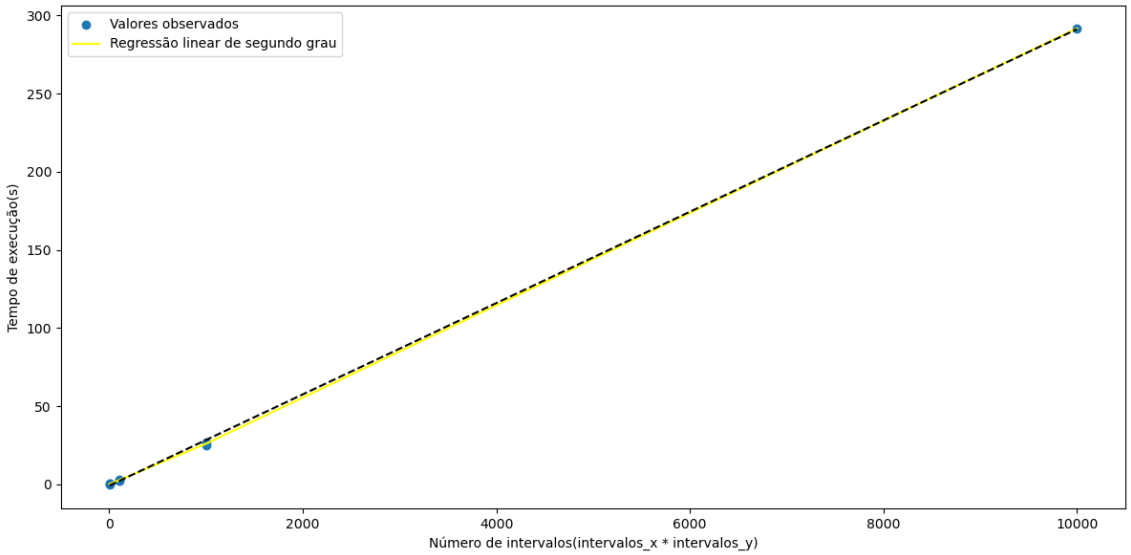

Figura 2: Redução Linear para 2 *threads*

linear para descrever o crescimento do tempo de execução.

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

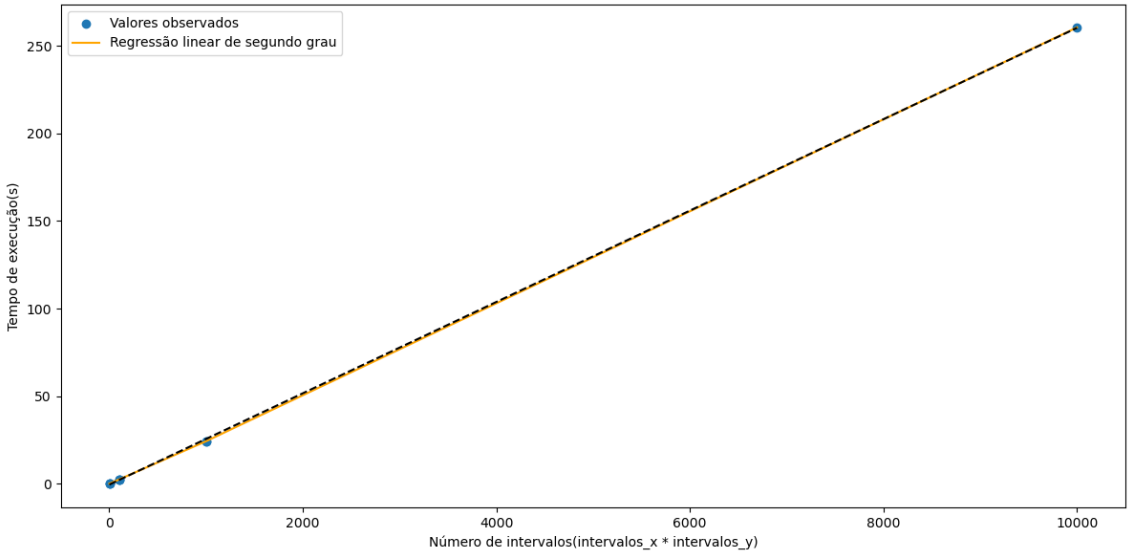

Figura 3: Redução Linear para 4 *threads*

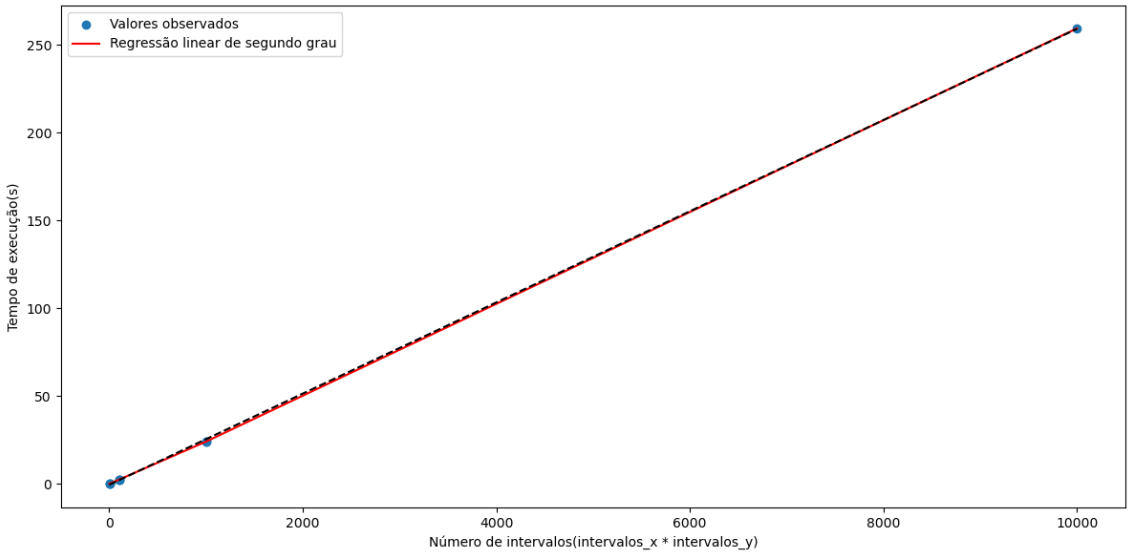

Figura 4: Redução Linear para 8 *threads*

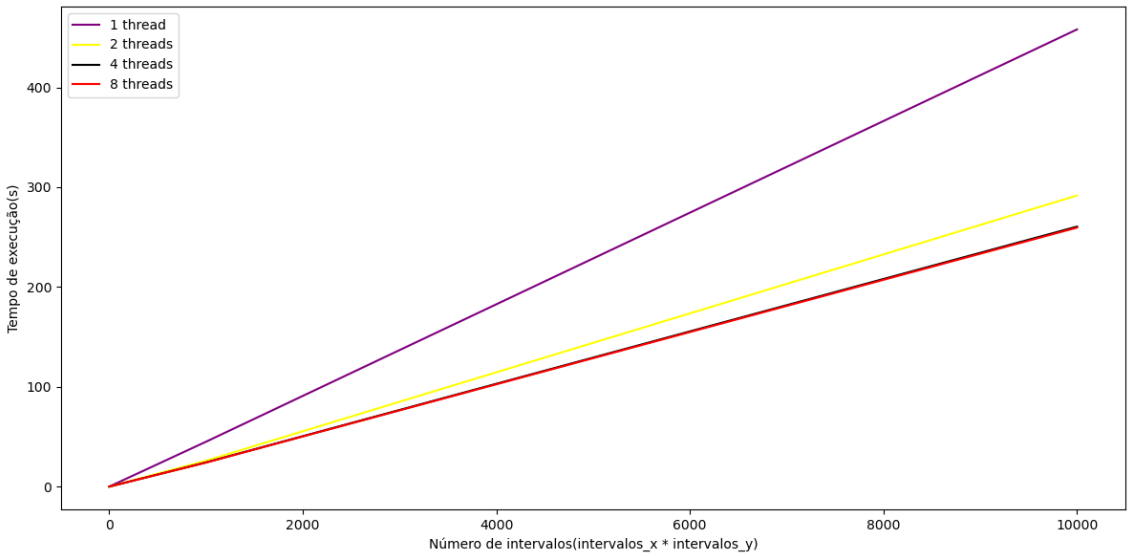

Figura 5: Comparação das Reduções Lineares