RELATÓRIO SOBRE O USO DA BIBLIOTECA CUDA PARA CÁLCULO DE INTEGRAL

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

1 Introdução

A biblioteca CUDA foi desenvolvida pela NVIDIA com o objetivo de utilizar os núcleos de processamento de GPUs para computação paralela. Propõe-se a separação de tarefas em que operações sequenciais ocorrem na CPU, enquanto operações simples são paralelizadas na GPU. À vista disso, o resultado de uma integral pode ser obtido paralelizando massivamente os cálculos de área dos trapézios enquanto observa-se o desempenho do paralelismo.

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 conjuntos de *threads*;
- Analisar o desempenho do paralelismo conforme o número de conjunto de *threads* aumenta.

2 Metodologia

Para a realização dos testes e cálculos, foi utilizado o Google Colab, que é uma plataforma de notebooks Jupyter com suporte a recursos de GPU. O ambiente de desenvolvimento foi configurado para usar o compilador nvcc (NVIDIA CUDA Compiler) e o processamento ocorreu por meio do acesso a Nvidia Tesla T4. O comando **time** foi utilizado para medir o tempo total decorrido desde o início até o final da execução da implementação. Os comandos de compilação e execução foram escrito nas células de código do Google Collab e o arquivo contendo o código em C foi carregado no ambiente.

3 Resultado

Esta seção é dedicada à análise das tabelas e gráficos de regressão linear como forma de avaliação do desempenho do número de blocos 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 - 3.

10^1 blocos				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.153	0.236	0.692
	10^4	0.225	0.611	3.630
	10^5	0.684	3.594	33.918

Tabela 1: Tempo de execução em segundos para 10 blocos

10^2 blocos				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.127	0.154	0.354
	10^4	0.208	0.371	1.288
	10^5	0.372	1.322	10.257

Tabela 2: Tempo de execução em segundos para 100 blocos

10^3 blocos				
	Intervalos no eixo y			
Intervalos no eixo x		10^3	10^4	10^5
	10^3	0.134	0.165	0.341
	10^4	0.213	0.347	1.148
	10^5	0.329	1.115	8.579

Tabela 3: Tempo de execução em segundos para 1000 blocos

A partir da tabela, nota-se que a implementação com 1000 blocos, cada um com 512 *threads*, obteve os menores tempos de execução, o que indica maior desempenho. Já a implementação com 10 blocos obteve os maiores valores de execução, apresentando o pior desempenho. Porém, é possível observar que os três valores com menores intervalos na implementação com 100 blocos foram menores que os seus correspondentes na de 1000 blocos. Isso ocorre, uma vez que há pouca carga de trabalho e o número de intervalos por *thread* é pequeno, não sendo necessário o uso de múltiplos blocos já que há um custo de comunicação e sincronização. 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. Assim, é possível ver no gráficos das Figuras 1 - 3 como os desempenhos são dispostos.

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. À vista que, o cálculo é proporcional ao número total de intervalos, que é a multiplicação X_intervalos x Y_intervalos,

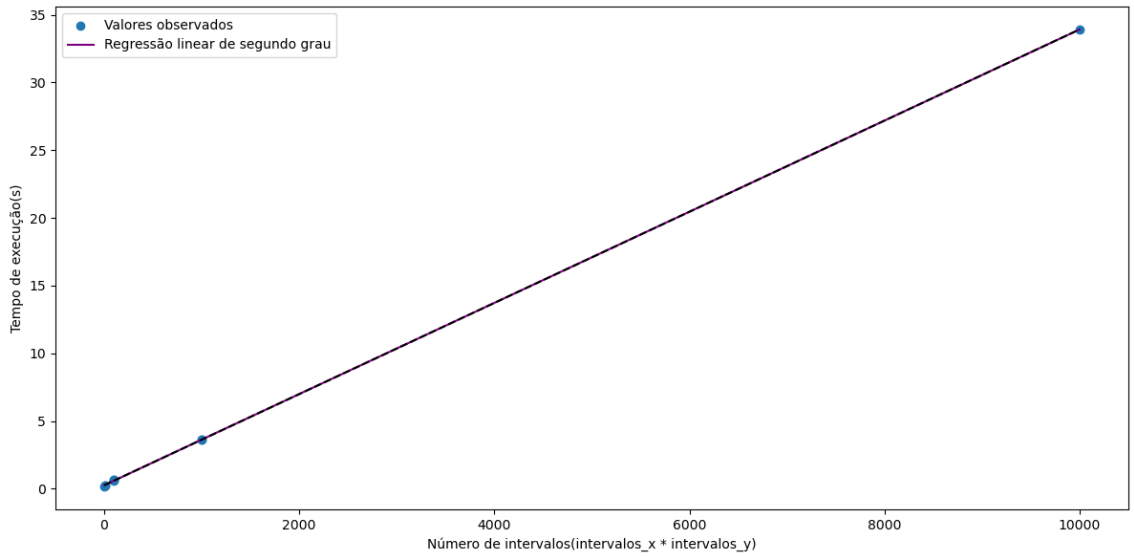

Figura 1: Redução Linear para 10 blocos

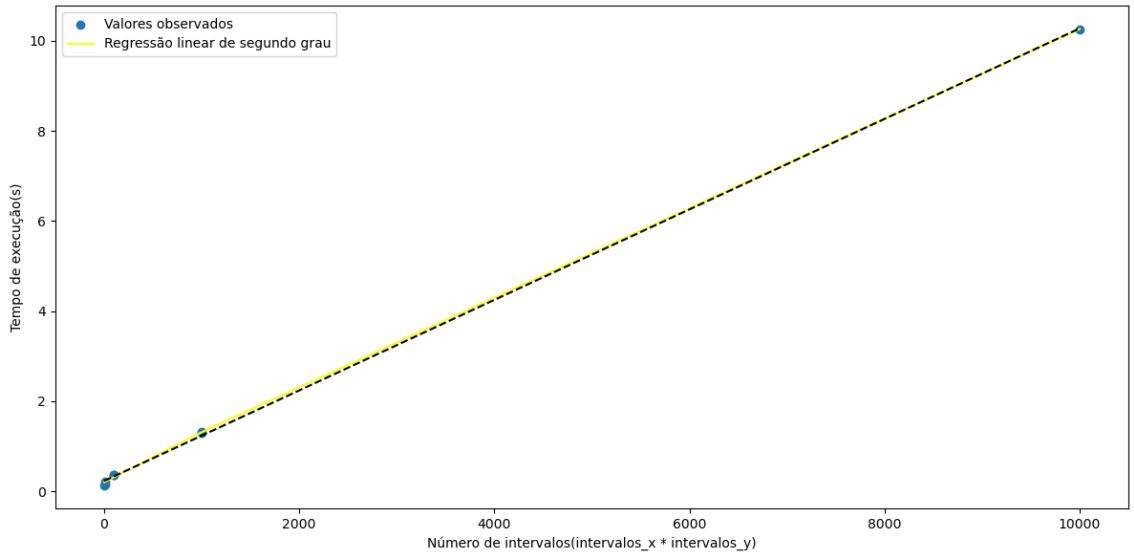

Figura 2: Redução Linear para 100 blocos

o tempo de execução cresce linearmente com o produto total dos intervalos. Entretanto, ao traçar a reta que interliga os dados e calcular a regressão linear, fica evidente que as duas variáveis apresentam comportamentos distintos. O crescimento quadrático do tempo pode ter muitos motivos, desde valores muito grandes de intervalos, cujo volume de dados pode saturar a largura de banda da GPU, à sobrecarga do paralelismo, em que a quantidade de trabalho por *thread* não compensa o custo de sincronização, criando uma curva com coeficiente de x^2.

4 Conclusão

De modo geral, como indicado pela Figura 4, mesmo com o modelo preditivo parabólico, a implementação com 1000 blocos demonstrou o melhor desempenho no cálculo da integral pelo método dos trapézios, apresentando quase todos os menores tempos de execução. Seguido

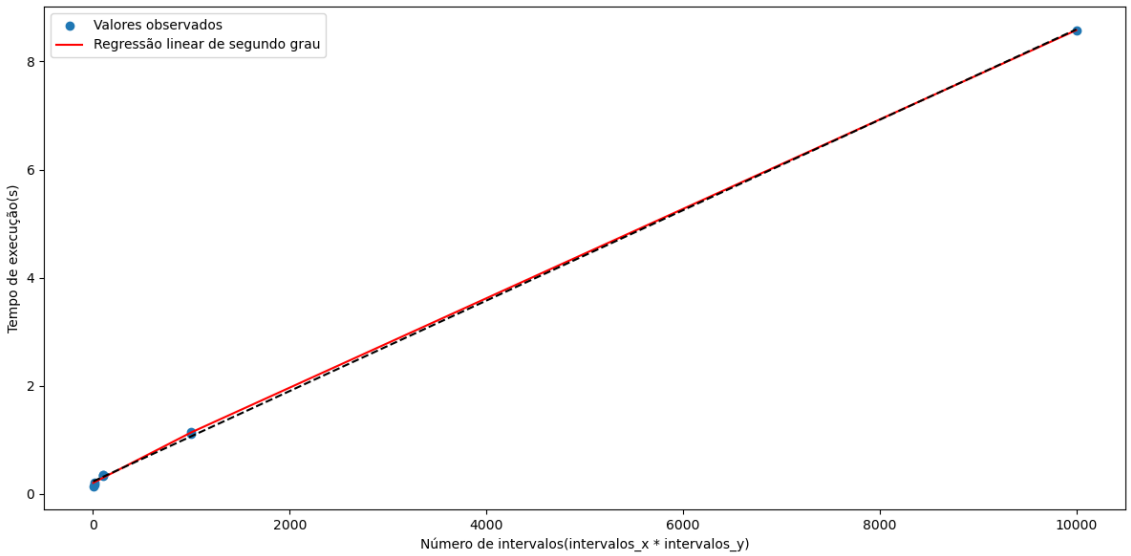

Figura 3: Redução Linear para 1000 blocos

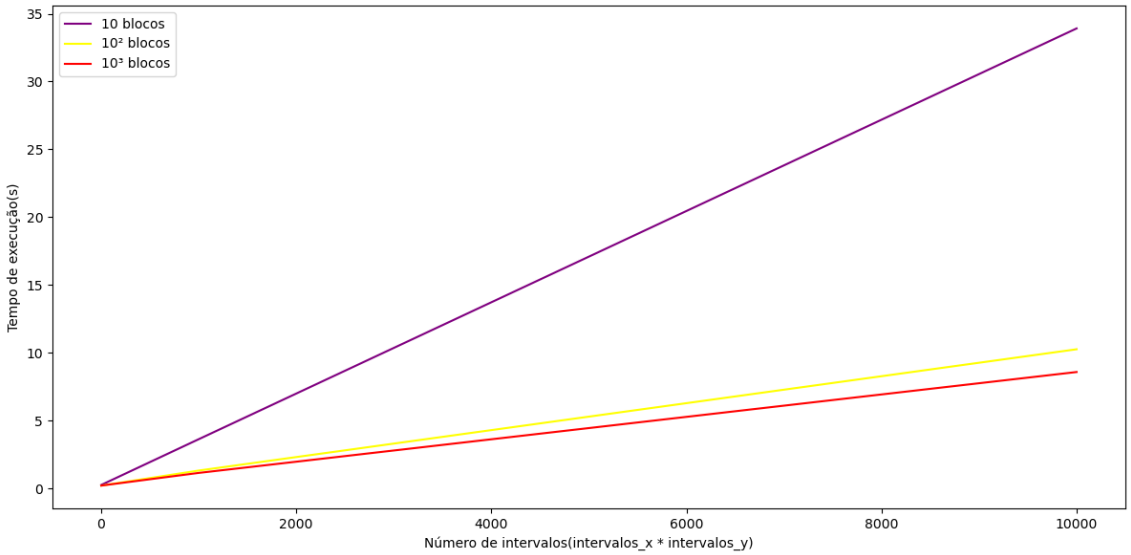

Figura 4: Comparação das Reduções Lineares

do desempenho da implementação com 100 blocos. E, por fim, o desempenho com 10 blocos, que não foi ruim, porém inferior aos outros, devido à maior eficiência proporcionada pelo uso de múltiplos conjuntos de *threads*, que permite a execução em menos tempo e melhora significativamente o desempenho do algoritmo.