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

Regras dos Trapézios com Cuda

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 deste projeto foi utilizada a plataforma CUDA (*Compute Unified Device Architecture*) para realizar o devido processo de paralelização. A CUDA é uma plataforma de computação paralela e modelo de programação desenvolvida pela NVIDIA, que permite o aproveitamento da capacidade de processamento das GPUs para computação de propósito geral.

O funcionamento da CUDA é baseado no conceito de computação heterogênea, onde a CPU (host) e GPU (device) trabalham em conjunto. A parte sequencial do código é executada na CPU, enquanto as tarefas computacionalmente intensivas são processadas em paralelo pelos milhares de núcleos CUDA disponíveis na GPU.

Dessa forma, nesse projeto iremos realizar a execução do métodos dos trapézios para integração numérica utilizando a plataforma Cuda 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 Blocos:** 10, 100 e 1000 Blocos.
- **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 plataforma CUDA, aproveitando a capacidade massivamente paralela das GPUs NVIDIA. O modelo de programação CUDA nos permitiu dividir os cálculos do método entre milhares de threads, compostas pelos número de blocos especificados no proejto junto ao número fixo

de threads por bloco, 512 threads, que são executadas simultaneamente nos múltiplos núcleos da GPU.

No código fizemos uma função para o Kernel que realiza a integral dupla de forma massivamente paralela com Cuda, essa função divide os subproblemas para todas as Threads e realiza os cálculo do método dos trapézios, fora isso temos uma função main que realiza a toda a execução levando em conta todas as combinações de parâmetros e cálculo os tempos de execução. Tivemos apenas uma coisa no código que para conseguir executar o código produzindo o resultado correto quando intervalos(10^5,10^5), utilizamos o tipo long long para essa produção de resultado correto, ao invés do tipo int.

Com isso, cercamos a execução do kernel do método dos trapézios com a função cudaEventCreate() e cudaEventRecord() da CUDA, que são utilizadas para medir o tempo decorrido em programas paralelos na GPU que nos fornece um mecanismo preciso e específico de medição de tempo de apenas das execuções dos kernels.

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.

Ressaltando aqui as especificações da máquina de onde foram executados esses *Benchmarks*. Nesse caso todas as execuções foram feitas no Google Co-

Blocos	**x**	**y**	**tempo(ms)**	**Resultado**
10	1000	1000	2.62448	1.39956
10	1000	10000	20.96480	1.39956
10	1000	100000	93.95967	1.39956
10	10000	1000	9.00393	1.39956
10	10000	10000	89.79542	1.39956
10	10000	100000	901.17773	1.39956
10	100000	1000	90.21632	1.39956
10	100000	10000	897.69397	1.39956
10	100000	100000	9742.02246	1.39956
100	1000	1000	0.34175	1.39956
100	1000	10000	2.74953	1.39956
100	1000	100000	27.06881	1.39956
100	10000	1000	2.74580	1.39956
100	10000	10000	27.05013	1.39956
100	10000	100000	270.29279	1.39956
100	100000	1000	27.09256	1.39956
100	100000	10000	270.42319	1.39956
100	100000	100000	2867.47949	1.39956
1000	1000	1000	1.03929	1.39956
1000	1000	10000	2.45290	1.39956
1000	1000	100000	22.78800	1.39956
1000	10000	1000	2.46167	1.39956
1000	10000	10000	22.78840	1.39956
1000	10000	100000	226.04358	1.39956
1000	100000	1000	22.79438	1.39956
1000	100000	10000	226.10645	1.39956
1000	100000	100000	2386.50000	1.39956

Tabela 1: Tabela dos resultados para cada combinação

Blocos	**x**	**y**	**eficiência**
10	1000	1000	0.0008873%
10	1000	10000	0.00108253%
10	1000	100000	0.0023407%
10	10000	1000	0.00238978%
10	10000	10000	0.00262319%
10	10000	100000	0.00243412%
10	100000	1000	0.00238628%
10	100000	10000	0.00237122%
10	100000	100000	0.00226408%
100	1000	1000	0.00068141%
100	1000	10000	0.00082542%
100	1000	100000	0.00081249%
100	10000	1000	0.00078365%
100	10000	10000	0.00087079%
100	10000	100000	0.00081155%
100	100000	1000	0.00079461%
100	100000	10000	0.00078715%
100	100000	100000	0.0007692%
1000	1000	1000	9.25237690e-05%
1000	1000	10000	9.65117680e-05%
1000	1000	100000	8.74097926e-05%
1000	10000	1000	1.03364238e-04%
1000	10000	10000	9.70419382e-05%
1000	10000	100000	9.44449814e-05%
1000	100000	1000	9.41427356e-05%
1000	100000	10000	9.24229938e-05%
1000	100000	100000	9.51437256e-05%

Tabela 2: Tabela das eficiências por combinação

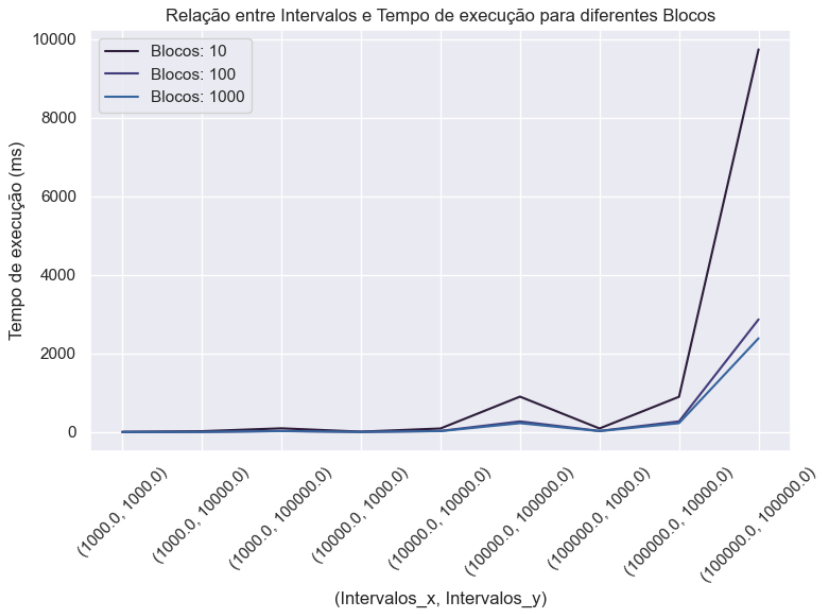

Figura 1: Gráfico do tempo de execução para diferentes combinações

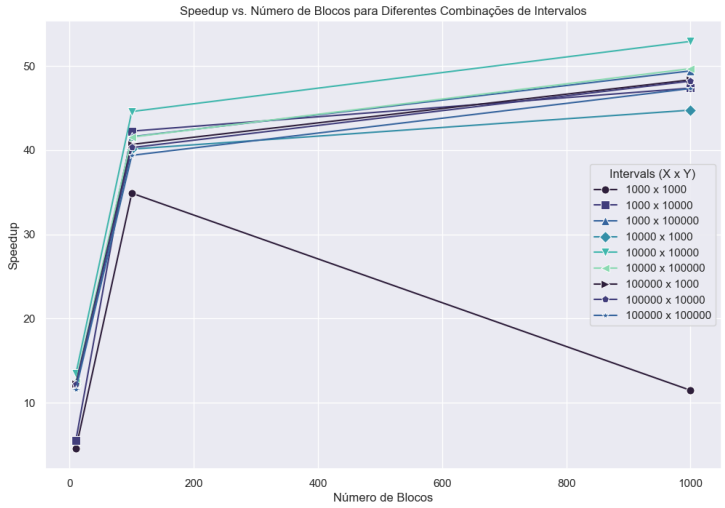

Figura 2: Gráfico de speedup para diferentes combinações

lab, utilizando o ambiente com uma GPU NVIDIA Tesla T4. Sendo utilizado também a arquitetura da GPU sm_75 para as execuções.

- Processador: Intel Xeon.

- GPU: NVIDIA Tesla T4 com Cuda cores 2,560.
- Memória RAM: 13 GB de memória GDDR6.

A fórmula utilizada para calcular o SpeedUp para esse projeto é mostrada abaixo:

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

onde T_{serial}, o tempo de execução medido rodando um programa para método dos trapézios de forma sequencial

Analisando o gráfico dos tempos de execução, podemos perceber que os tempos de execução para intervalos pequenos, mesmo que sejam para diferentes números se mantêm muito próximos mostrando que para esses problemas menores não tem muito efeito a aplicação de mais paralelismo. Porém, ao aumentar o tamanho do número de intervalos para o método dos trapézios nos mostra que o aumento de 10 blocos para 100 blocos temos um ganho grande no tempo de execução, mas de 100 blocos para 1000 blocos não temos um ganho expressivo de desempenho.

Ao olhar o gráfico 2, percebemos que os valores de speedup de 10 blocos para 100 blocos, também nos mostra que esse valor é muito melhor em questão de desempenho em paralelismo. Porém ao observarmos os speedups de 100 para 1000 blocos, vemos que não obtemos um ganho tão alto assim, ademais, na combinação de intervalos (1000x1000) temos um baixa no valor de speedup, o que pode ter causado isso é o fato de o problema nesse caso ser muito pequeno e número total de threads ser muito alto, causando provavelmente um overhead, consequentemente tenham threads ociosas e caindo o desempenho.

Por último, analisando a Tabela 2 das eficiências para cada combinação que pode ser descrita por:

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

nos mostra que a eficiência calculada pela fórmula, não foi tão satisfatória, porém isso provavelmente se da pelo fato que os valores de Threads totais são muito elevados comparados ao tamanho do problema inteiro. Sendo assim, se compararmos os valores de eficiência de Cuda com o MPI percebemos que são piores mas isso se da ao fato da explicação anterior. Quando vemos o valor Speedup para o Cuda é muito melhor o desempenho se comparado ao MPI. Sendo assim, para esse problema o Cuda se mostrou melhor para utilização em paralelização.

5 Conclusão

Portanto, a partir de toda as execuções com as diferentes combinações e os gráficos e tabelas extraídos, podemos concluir que a Cuda foi muito eficiente para a resolução da integral dupla, definida na especificação do projeto, pelo métodos dos trapézios. Podemos perceber, que ao paralelizar com Cuda se compararmos os tempos de execução com o projeto de MPI é muito mais eficiente e também com a execução em serial.

Porém, se utilizarmos a Cuda para resolver esse problema, nos atentamos que ao utilizar 1000 blocos não temos um ganho tão expressivo de desempenho, em alguns casos até perdas, se comparado com a execução para 100 blocos. Dessa forma, não seria muito interessante rodar esse código com 1000 blocos, tendo em vista que não nos leva a um ganho muito grande de desempenho e utilizaríamos de mais Threads, o que resultaria em mais overhead pelo fato do tamanho pequeno de um problema específico.

Dessa forma, percebemos que o ponto ótimo de paralelismo para esse problema utilizando Cuda seria rodando com 100 blocos cada com 512 Threads, assim extrairíamos o melhor desempenho para calcular o método dos trapézios de forma paralela com Cuda.