Adaptando Simulador Científico para CUDA

1º Gabriel da Cunha Borba
Instituto de computação
Universidade Federal Fluminense
Niterói, Brasil
gborba@id.uff.br

***Abstract*—Ao identificar áreas possíveis de serrem paralelizadas no código original de um solucionador de equações deferenciais (simulador) uma nova versão deste simulador foi desenvolvida pensando na paralelização das operações tirando proveito do poder de paralelização oferecido pelas GPUs.**

***Index Terms*—GPU, CUDA, simulação, programação científica, métodos numéricos.**

I. Introdução

GPUs são ótimos dispositivos para processar um alto número de tarefas iguais. A arquitetura da GPU faz da GPU um dispositivo SIMD, isto é dispositivo capaz de operar sobre vários dados de forma paralela usando apenas uma instrução e essa capacidade permite que as GPUs alcancem uma alta vazão na realização de tarefas.

O objetivo geral deste trabalho é resolver algum problema usando a GPU, então, o problema selecionado foi reescrever o simulador desenvolvido durante as aulas da disciplina de Programação Científica. Durante as aulas de programação científica foram escritos um conjunto de scripts para uma aplicação cientifica, esses scripts permitem a modelagem e a simulação da condutividade térmica de uma chapa através da solução de um problema de valor de contorno. Resolver a equação da condutividade térmica é importante em muitas áreas da física e engenharia por permitir entender a distribuição de temperatura sobre um corpo ao longo do tempo.

Com isso em mente, este trabalho cobre a reescrita do simulador das aulas de programação científica na linguagem C (e usando a extensão CUDA para a linguagem) de forma que a simulação utilize do paralelismo oferecido pela GPU e verificar a nova implementação melhora o desempenho do simulador quando comparado a implementação em CPU.

II. Referencial Teórico

Vários fenômenos físicos podem ser descritos como equações diferenciais que podem ser solucionadas de maneira analítica ou computacional, essas equações diferenciais permitem associar como a mudança de uma variável influencia mudanças em outra variável. É possível construir um simulador para esses fenômenos físicos ao solucionar essas equações utilizando um computador mas, a principio, um computador não consegue resolver diretamente essas equações diferenciais, é preciso reescrever as equações diferenciais em equações algébricas de forma que o computador seja capaz de resolver através de métodos numéricos.

Esses fenômenos físicos podem ser modelados como problemas de valor inicial, que são equações diferenciais onde a solução precisa satisfazer uma condição inicial dada em 1 local do domínio, ou como problemas de valor de contorno, que são equações diferenciais onde condições são impostas em diferentes pontos do domínio. Foram usados neste trabalho o método leapfrog para solucionar o PVI da 2 Lei de Newton e o método das diferenças finitas para solucionar o PVC da dispersão térmica.

A seguir foi feito um apanhado geral sobre problemas de valor incial e problemas de valor de contorno assim como métodos numéricos para soluciionar tas problemas, em [1] o esses assuntos são debatidos com mais profundidade.

A. Possibilidade de solução para PVIs

A equação diferencial precisa ser reescrita de uma maneira padronizada, a equação deve ser escrita na forma

$$\frac{dy}{dt} = f(t, y),\ a \leq t \leq b,\ y(a) = \alpha \tag{1}$$

caso a função ainda não esteja nesta forma, o termo de maior derivada deve ser colocado no lado esquerdo da equação e todos os outros termos devem ser movidos para a direita de maneira que, à direita, é obtida uma função f que é a derivada da função do problema que queremos solucionar. Quando o problema que queremos solucionar é de segunda ordem ou superior é preciso transformar a equação diferencial em um conjunto (ou sistema) de equações diferenciais de primeira ordem na forma padronizada.

Antes de tentar encontrar a solução para o problema de valor inicial usando o computador precisamos garantir que o problema é bem posto e que o problema tem solução é única.

1) problema bem posto: Dizer que um problema é bem posto é o mesmo que dizer que pequenas mudanças na definição do problema geram mudanças também pequenas na solução.

O teorema a seguir exibe condições que asseguram que um problema de valor inicial é bem posto.

Teorema:

Suponha $D = (t, y)|a \leq t \leq b, -\infty \leq y \leq \infty$. Se f é continua e satisfaz uma condição de Lipschitz na variável y no domínio D, então o problema de valor inicial (1) é bem posto.

2) Solução única: dado um problema de valor inicial, ele vai ter solução única se o domínio for convexo e o f satisfaz a condição de lipschitz no dominio

3) Conjunto Convexo: Todos os pontos que estejam entre qualquer outros dois pontos (t_1, y_1) e (t_2, y_2) no domínio $D \subset R^2$ da função, também devem estar em D.

Definição:

Um conjunto $D \subset R^2$ é dito convexo se sempre que (t_1, y_1) e $(t_2, y_2) \in D$, e $((1-\lambda)t_1 + \lambda t_2, (1-\lambda)y_1 + \lambda y_2) \in D \; \forall \lambda \in [0, 1]$

4) Condição de Lipschitz:

Definição:

Uma função é dita satisfazer uma condição de Lipschitz na variável y em um conjunto $D \subset R^2$, se existir uma constante $L > 0$ com

$$|f(t, y_1 - f(t, y-2)| \leq L|y_1 - y_2|$$

sempre que $(t, y_1) e (t, y_2) \in D$. A constante L é chamada de uma constante de Lipschitz para f.

O teorema a seguir é mais simples de utilizar para identificar se um problema de valor inicial satisfaz a condição de Lipschitz apesar de fornecer apenas uma condição suficiente para uma condição de Lipschitz.

Teorema:

Suponha que $f(t, y)$ é definida em um conjunto convexo $D \subset R^2$. Se uma constante $L > 0$ existir com

$$\left|\frac{df}{dy}\right| \leq L, \; \forall (t, y) \in D,$$

então f satisfaz uma condição de Lipschitz em D na variável y com constate de Lipschitz L.

B. Métodos numéricos para PVIs

Há diversos métodos numéricos para solucionar problemas de valor inicial, alguns desses métodos são: Euler, de Runge-Kutta, de Adams–Bashforth, de Nyston, Adams-Mouton, Milne-Simpson, leapfrog, Backward Differenciation Formula.

Ao recorrer à solução computacional de um problema de valor inicial o resultado obtida não será a solução analítica contínua para $y(t)$ mas sim aproximações de y que serão obtidos de forma iterativa para valores em uma malha de pontos criada sobre o intervalo $[a, b]$. Esses valores podem ser interpolados para encontrar aproximações para outros pontos que não estejam na malha de pontos do intervalo.

É assumido que os pontos da malha tem o mesmo tamanho de passo escolhendo um número inteiro positivo N de tal forma que os pontos na malha de pontos são $t_i = a + i \times h, para i = 0, 1, 2, ..., N$ e $h = (b-a)/N$

A maneira como o leapfrog faz para resolver equações diferenciais de segunda ordem é calcular o valor das variáveis do problema de valor inicial em posições alternadas e diferentes do dominio de maneira que os momentos de atualização das variáveis saltam entre si.

1) Classificação dos métodos numéricos: Há duas formas de classificar um método numérico para solucionar equações diferenciais, são elas o número de passos e a explicitude.

classificação quanto ao numero de passos: nos *métodos numéricos de passo simples* as aproximações para o ponto t_{i+1} na malha de pontos usa apenas 1 dos pontos anteriores, t_i. E nos *métodos numéricos de passos múltiplos* a aproximações usam mais que 1 ponto anterior da malha de pontos para determinar a aproximação para o ponto t_{i+1}.

Classificação quanto a explicitude: Os chamados *métodos numéricos explícitos* encontram uma aproximação w_{i+1} de maneira explicita com apenas os valores encontrados anteriormente, e os chamados *métodos numéricos implícitos* porque na equação de diferença usada para obter w_{i+1} o termo w_{i+1} aparece em ambos os lados dela.

os métodos numéricos para solucionar problemas de valor inicial são basicamente iguais, a aproximação inicial vai ser o valor dado como condição inicial e aproximações para passos subsequentes serão o valor da aproximação anterior mais o produto do tamanho do passo h e o valor da função f avaliada em valores específicos da variável independente e no valor da aproximação anterior.

Definição: Um método com m-passos para obter $w_i + 1 \approx y(t_{i+1})$ no ponto t_{i+1} da malha de pontos tem a equação de diferença da seguinte forma, onde m é um inteiro positivo maior que 0:

$$\begin{aligned} w_{i+1} = a_{m-1}w_i + a_{m-2}w_{i-1} + ... + a_0 w_{i+1-m} \\ + h[b_m f(t_{i+1}, w_{i+1}) + b_{m-1} f(t_i, w_i) \\ + ... + b_0 f(t_{i+1-m}, w_{i+1-m})] \end{aligned}$$

para $i = m-1, m, ..., N-1$, onde h é o tamanho do passo, $a_0, a_1, ..., a_{m-1}$ e $b_0, b_1, ..., b_m$ são constantes e os valores iniciais $w_0 = \alpha, w_1 = \alpha_1, w_2 = \alpha_2, ..., w_{m-1} = \alpha_{m-1}$ especificados.

C. Possibilidade de solução para PVCs

Assim como no caso dos problemas de valor inicial é preciso saber se o problema de valor de contorno tem solução, para isso há um teorema que garante que a solução é única e existe.

Teorema:

Suponha a função f no problema de valor de contorno

$$y'' = f(x, y, y'), \; fora \leq x \leq b, \; comy(a) = \alpha e y(b) = \beta \tag{2}$$

é contínua no conjunto

$$\begin{aligned} D = (x, y, y') | fora \leq x \leq b, \\ com -\infty < y < \infty e -\infty < y' < \infty \end{aligned}$$

, e que as derivadas parciais f_y e $f_{y'}$ também são contínuas em D. Se

- $f_y(x, y, y') > 0 \; \forall (x, y, y') \in D$, e
- a constante M existe, com

$$|f_{y'}(x, y, y')| < M, \; \forall (x, y, y') \in D$$

então o problema de valor de contorno tem solução única.

Em situações onde a equação (2) é linear, isto é, quando funções $p(x)$, $q(x)$ e $r(x)$ existem e

$$f(x, y, y') = p(x)y' + q(x)y + r(x)$$

, o teorema anterior pode ser simplificado de maneira que (2) satisfaz

- $p(x)$, $q(x)$ e $r(x)$ são contínuas em $[a, b]$, e
- $q(x) > 0$ em $[a, b]$.

então o problema de valor de contorno tem solução única.

D. Métodos numéricos para PVCs

Há diversos métodos numéricos para solucionar problemas de valor de contorno, alguns desses métodos são: do tiro, das diferenças finitas (MDF), dos elementos finitos (MEF), dos volumes finitos (MVF), dos elemntos de contorno (MEC), sem malha (MSM), de Lattice-Boltzmann e randon-walk.

O método das diferenças finitas consiste em discretizar o domínio em uma malha de N pontos e transformar o problema de valor de contorno em um sistema de equações algébricas. Assim como com os problemas de valor inicial, o resultado obtido não será a solução analítica contínua para $y(t)$ mas sim aproximações de y nos pontos da malha.

No MDF inicialmente é usada a expressão de diferenças finita em cada par de pontos da malha de pontos a fim de usar os valores (equações) obtidos como a aproximação da derivada entre eles, se a derivada for de ordem maior vai ser necessário usar a expressão de diferenças finitas novamente sobre os valores das derivadas anteriores para obter as aproximações das derivadas de ordem superior, os valores (as equações) encontradas são substituídas no problema de valor de contorno original, o pcv então é transformado em um conjunto equações algébricas onde as variáveis dessa equação são os valores aproximados para cada ponto da malha que foram usados para a aproximação, a partir daí é possível montar uma matriz usando o sistema de equações encontrado, solucionar a matriz permite determinar o valor aproximado de y em cada ponto da malha de pontos.

III. Metodologia

A. Aspectos gerais

Trabalho em caráter semi-experimental, quantitativo, preditivo, com finalidade de comparar tempos de execução entre a implementação original de simuladores (solucionadores de PVI e PVC) restritos a CPU que foram escritos na linguagem Julia e uma nova implementação escrita em C e CUDA dos mesmos simuladores agora utilizando a GPU e verificar o ganho de performance na execução dos simuladores.

Cada teste, isto é, a execução de 1 simulador com 1 arquivo de entrada, foi realizado 30 vezes a fim de gerar a média de tempo de execução usada para calcular o possível speedup obtido com a nova implementação. O tempo de execução aferido contempla todo o tempo de execução do programa desde a entrada de dados até a saída.

A versão do simulador reescrita em C e CUDA alterou o simulador original dividindo e paralelizando na GPU alguns laços de repetição executados durante a aproximação dos resultados, assim como também alterou a solução dos sistemas de equações para que sejam realizadas com o uso da GPU. Na nova implementação do simulador para PVI cada passo da simulação foi dividida em 3 kerneis que substituem os 2 loops aninhados do simulador original, o primeiro kernel atualiza a velocidade e o deslocamento para cada elemento da malha, o segundo kernel aplica as restrições de movimento nos elementos em que forem necessários e executa o algoritmo de contato somando os valores de forças sendo aplicadas em cada elemento da malha e o terceiro kernel atualiza a velocidade novamente e também a aceleração. Agora, na nova implementação do simulador para PVC dois kerneis foram desenvolvidos, o primeiro para inicializa o sistema de equações, o segundo que impõe as condições de contorno do problema, a solução do sistema de equações foi feita utilizando uma chamada a uma função cuda.

B. Amostra

- *q2_leapfrog.jl* - solucionador do PVI relacionado ao Problema da Segunda Lei de Newton escrito na linguagem Julia durante a disciplina Programação Científica, em relação ao original desenvolvido esse simulador foi alterado para reduzir ao mínimo o número de impressões;
- *q1_simulador.jl* - solucionador do PVC relacionado ao problema de Dispersão Térmica escrito na linguagem Julia durante a disciplina Programação Científica, em relação ao original desenvolvido esse simulador foi alterado para reduzir ao mínimo o número de impressões;
- *simulador.bin* - solucionador do PVI relacionado ao Problema da Segunda Lei de Newton e ao PVC do problema de Dispersão Térmica escritos na linguagem C e usando CUDA e compilado para linux.
- *q2in_1_10x10.json* - entrada para o simulador de PVI, uma chapa retangular discretizada em uma malha de 10x10, o domínio do tempo dividido 100 passos de tamanho 0.0005s, com movimento restringido nos eixos x e y na borda inferior da chapa.
- *q2in_1_50x50.json* - entrada para o simulador de PVI, uma chapa retangular discretizada em uma malha de 50x50, o domínio do tempo dividido 100 passos de tamanho 0.0005s, com movimento restringido nos eixos x e y na borda inferior da chapa.
- *q2in_3_10x10.json* - entrada para o simulador de PVI, uma chapa retangular discretizada em uma malha de 10x10, o domínio do tempo dividido 1000 passos de tamanho 0.00005s, com movimento restringido nos eixos x e y na borda inferior da chapa.
- *q2in_3_50x50.json* - entrada para o simulador de PVI, uma chapa retangular discretizada em uma malha de 50x50, o domínio do tempo dividido 100 passos de tamanho 0.00005s, com movimento restringido nos eixos x e y na borda inferior da chapa.
- *2022y039d22h27m02sIn.json* - entrada para o simulador de PVC, uma chapa retangular discretizada em uma malha 4x4 com toda a borda esquerda imposta uma temperatura no valor de 100, toda a borda direita imposta uma temperatura no valor de 0, a borda superior tem imposta uma temperatura no valor de 75 (exceto os pontos que interceptam as bordas esquerda e direita) e a borda inferior tem imposta uma temperatura no valor de

25 (exceto os pontos que interceptam as bordas esquerda e direita).

- *2022y040d19h03m39sIn.json* - entrada para o simulador de PVC, uma chapa retangular discretizada em uma malha 19x7 com toda a borda esquerda imposta uma temperatura no valor de 100, toda a borda direita imposta uma temperatura no valor de 0, a borda superior tem imposta uma temperatura no valor de 75 (exceto os pontos que interceptam as bordas esquerda e direita) e a borda inferior tem imposta uma temperatura no valor de 25 (exceto os pontos que interceptam as bordas esquerda e direita).
- *2022y040d19h22m41sIn.json* - entrada para o simulador de PVC, uma chapa retangular discretizada em uma malha 59x19 com a borda esquerda imposta uma temperatura no valor de 100 (exceto os pontos que interceptam as bordas superior e inferior), a borda direita imposta uma temperatura no valor de 100 (exceto os pontos que interceptam as bordas superior e inferior), toda a borda superior tem imposta uma temperatura no valor de 80 e a borda inferior tem imposta uma temperatura no valor de 80.

O arquivo *q2_leapfrog.jl* se encontra em ./aula_012_P2_GabrielDaCunhaBorba/ com relação ao repositório deste trabalho.

O arquivo *q1_simulador.jl* se encontra em ./aula_015_P4_GabrielDaCunhaBorba/ com relação ao repositório deste trabalho.

O arquivo *simulador.bin* se encontra em ./output/ com relação ao repositório deste trabalho.

Os arquivos *q2in_1_10x10.json*, *q2in_1_50x50.json*, *q2in_3_10x10.json* e *q2in_3_50x50.json* se encontram em ./aula_012_P2_GabrielDaCunhaBorba/output/ com relação ao repositório deste trabalho.

Os arquivos *2022y039d22h27m02sIn.json*, *2022y040d19h03m39sIn.json* e *2022y040d19h22m41sIn.json* se encontram em ./aula_015_P4_GabrielDaCunhaBorba/output/ com relação ao repositório deste trabalho.

C. Instrumentos

1) Ferrramentas de benchmark:

- "executarCronometrar.ps1" - Powershell script escrito para automatizar a execução, dos simuladores o cálculo de tempo médio e o desvio padrão amostral do tempo de execução dos simuladores.

2) Configurações do sistema: O computador usado para execução dos testes está equipado com:

- Processador AMD FX(tm)-6300 Six-Core a 3.50 GHz de velocidade de clock,
- Placa de vídeo NVIDIA GTX 1660 Super com 6GB de memória GDDR6,
- 8GB de memória ram DDR3 operando a 1600 Mhz single channel,
- Sistema operacional KDE Neon baseado na versão 22.04 do Ubuntu,
- Kernel Linux na versão 6.2.0-36-generic,
- Cuda Toolkit na versão 12.3.0,
- Driver cuda na versão 545.23.6,

3) Métricas de benchmark:

- Tempo de parede da execução;

$$\bar{t} = \frac{1}{n}\sum t_i$$

$$s = \sqrt{\frac{\sum (t_i - \bar{t})^2}{n-1}}$$

- Speedup.

$$S = \frac{t_{original}}{t_{C,CUDA}}$$

D. Sistemas de processamento de dados

1) Preparação da amostra: Todos os arquivos na amostra que foram usados para alimentar os simuladores foram desenvolvidos sob orientação do professor da disciplina Programação Cientifica durante o curso da mesma e não foram alterados desde então;

Os simuladores "q2_leapfrog.jl" e "q1_simulador.jl" foram alterados para reduzir ao mínimo o número de impressões na saída padrão;

2) Software para a análise de dados:

- "Google Spreadsheets" - usado para adequar os valores e visualizar de maneira gráfica os resultados obtidos da execução dos simuladores.

E. Repositório

o repositório deste tabalho se encontra em [2].

IV. Análises e Resultados

Nas tabelas I e II são exibidos os tempos obtidos na execução das duas versões dos simuladores nas duas implementações, respectivamente para o simulador que soluciona o PVI relacionado à Segunda Lei de Newton e para o simulador que soluciona o PVC relacionado à dispersão térmica.

Nas tabelas III e IV são exibidos os speedups obtidos pela execução do simulador na GPU usando C e CUDA.

Pode ser observada que a alteração do simulador para execução paralela (na GPU) das etapas do loop e calculo do sistema de matrizes também utilizando a gpu promoveu um ganho de desempenho.

V. Conclusão

Houve de fato a diminuição no tempo de execução do simulador com a nova implementação que promoveu um speedup de no mínimo 22 vezes dentro dos casos de teste avaliados e sob as condições descritas.

Apesar da speedup positivo há uma limitação da nova implementação em C e CUDA quando comparado a implementação original avaliada, essa limitação é o não uso de matrizes esparças como na implementação anterior, o uso de tais matrizes permitiriam solucionar problemas maiores, talvez poderia reduzir ainda mais o tempo de execução por causa do menor número de elementos sendo manipulados nas operações (mesmo considerando o aumento de tempo

devido a complexidade das operações), reduziria a quantidade de memória desperdiçada em elementos não importantes e também reduziria o tempo movendo dados ente as memórias do host e do device.

Fica para uma nova versão do simulador em C e CUDA o uso de matrizes esparças parar a intensidade do uso de memória.

REFERENCES

[1] R. L. Burden, *Numerical analysis*. Brooks/Cole Cengage Learning, 2011.

[2] G. Borba, "Trabalho de programação de gpus." https://github.com/gborba-uff-cc/trab_prog_gpus, 2023.

Table I
TEMPOS DO SIMULADOR PARA PVI EM CPU E GPU

Teste			Tempo de execução	
Vinculado à	Tamanho malha	Número de passos	Média	Desvio[a]
CPU	10x10	100	8,604837	0,112111
CPU	50x50	100	10,658500	0,127456
CPU	10x10	1000	9,247987	0,114854
CPU	50x50	1000	28,395515	0,410968
GPU	10x10	100	0,386811	0,497110
GPU	50x50	100	0,303948	0,013092
GPU	10x10	1000	0,320441	0,015367
GPU	50x50	1000	0,337585	0,014186

[a] desvio padrão amostral

Table II
TEMPOS DO SIMULADOR PARA PVC EM CPU E GPU

Teste		Tempo de execução	
Vinculado à	Tamanho malha	Média	Desvio[a]
CPU	4x4	11,163921	0,179996
CPU	19x7	11,122365	0,147854
CPU	59x19	11,194313	0,141202
GPU	4x4	0,362937	0,073378
GPU	19x7	0,350483	0,012745
GPU	59x19	0,394633	0,052627

[a] desvio padrão amostral

Table III
SPEEDUP DO SIMULADOR PARA PVI

Teste		
Tamanho malha	Número de passo	Speedup
10x10	100	22,24558974
50x50	100	35,06681136
10x10	1000	28,86022855
50x50	1000	84,11366899

Table IV
SPEEDUP DO SIMULADOR PARA PVC

Teste	Speedup
4x4	30,7599527
19x7	31,73437302
59x19	28,3664045