Aplicando OpenCL na determinação da Dimensão Fractal

Raphael de Souza Rosa Gomes[1], Geison Jader Mello[2], Josiel Maimone Figueiredo[3], José de Souza Nogueira[4]

Universidade Federal de Mato Grosso, Programa de Pós Graduação em Física Ambiental Brasil

Resumo

A Teoria da Complexidade representa hoje uma tendência relativamente nova de pesquisa para representação de fenômenos observados pelo homem. Sua concepção envolve a aplicação conjunta de diversos conceitos e suas fórmulas. E um dos conceitos usados é a Dimensão Fractal, cuja propriedade revela o comportamento de dados e consequentemente do fenômeno estudado. Para realização desses cálculos foi utilizado GPU(Graphics Processing Unit), que tem uma de suas vertentes baseadas em bibliotecas do padrão OpenCL. Para a realização dos testes foi utilizado uma máquina com uma placa de vídeo NVIDIA GT 520m que continha 48 núcleos com velocidade de 1,5 Ghz. Para os testes foi desenvolvido dois softwares, um para OpenCL utilizando a placa de vídeo e outro com programação sequencial, ambos criados em Java. Os testes foram baseados nos 100.000 mil dados gerados da equação de Lorenz para o eixo X, sendo utilizado somente uma fração dos dados por testes (foram executados com 1.000, 2.000, 4.000, 8.000, 16.000 e 32.000 dados, tanto com OpenCL, quanto com a programação sequencial). Ao final dos testes foram confrontados os resultados para verificação dos cálculos e validação dos mesmos. O método aplicado mostrou redução de até 90% do tempo.

Palavras-chave: OpenCL, GPU, dimensão fractal

Contatos dos autores: [1]thesivis@gmail.com,
[2]geison@pgfma.ufmt.br,
[3]josiel@ic.ufmt.br,
[4]nogueira@cpd.ufmt.br.

1. Introdução

A Teoria da Complexidade representa hoje um ramo relativamente novo, e dentro dessa ciência existem inúmeros conceitos e fórmulas para tentar representar fenômenos observados pelo homem. Um desses conceitos é o da reconstrução do espaço de fase para identificação do atrator (Nussenzveig, 2000) utilizando defasagens temporais (Takens, 1981) em uma série de dados também temporal. Entende-se o espaço de fase como uma representação gráfica onde a variável tempo é oculta, e o atrator representa o conjunto para onde tendem as trajetórias. Essa técnica utiliza os dados $X_0(t)$, e reconstrói as outras variáveis $\{X_k(t)\}$ (sendo k=1, ..., n-1). Depois aplica-se uma defasagem fixa τ ($\tau = m\Delta t$, onde m é um número inteiro e Δt é um intervalo entre sucessivas amostras) para "n" pontos equidistantes do conjunto de dados. Isto é:

X_0:

$X_0(t_1), \ldots, X_0(t_n)$

X_1:

$X_0(t_1 + \tau), \ldots, X_0(t_n + \tau)$

:

:

:

X_{n-1}: $X_0[t_1 + (n-1)\tau], \ldots, X_0[t_n + (n-1)\tau]$

A escolha desse tempo de defasagem se deve a menor autocorrelação, onde pode-se obter o maior ganho de informação da série de dados. Por esse fator é importante escolher o tao (τ) adequado como citado por Baker e Gollub (1996).

A dimensão fractal é uma medida muito utilizada para medir a "estranheza" dos atratores, assim como o número de graus de liberdade e informações estatísticas sobre o sistema. Neste artigo aplicamos o algoritmo de Grassberger e Procaccia (1983) que é, dentre os diferentes procedimentos desenvolvidos para computar a dimensão fractal, o mais amplamente utilizado. A dimensão de correlação (D_c) também provê o número de variáveis independentes necessárias para descrever a evolução temporal da dinâmica do sistema com a dimensão de imersão *m* (Takens, 1981) que tem como limite superior $2D_c+1$ para modelar um sistema.

Em um espaço de fase m-dimensional, a função correlação integrante C(r) do atrator é dada por (Grassberger & Procaccia, 1983):

$$C(r) = \frac{1}{n^2} \sum_{\substack{i,j=1 \\ i \neq j}}^{n} \theta(r - |X_i - X_j|) \quad \text{(Eq. 1)}$$

em que θ é a função Heaviside, θ (x) = 0 se x < 0, e θ (x) = 1 se x > 0.

A partir de uma pequena correlação ε sonda-se a estrutura do atrator. Se esta estrutura é uma linha, o número de pontos dentro de uma sondagem a distância r de um ponto deve ser proporcional ao r/ε. Se for uma superfície, este número deve ser proporcional a $(r/\varepsilon)^2$ e, de forma geral, se for uma dimensão d deve ser proporcional a $(r/\varepsilon)^d$. Logo, para r relativamente pequeno, $C(r)$ deverá variar conforme:

$$C(r) \sim r^d \quad \text{(Eq. 2)}$$

Assim, como a dimensão de correlação (D_c) do atrator é aproximadamente igual à dimensão fractal d, ela é dada pela declinação $\ln C(r)$ por $\ln r$ para um valor de r infinitesimal crescente até a integração total do atrator. Assim, a dimensão fractal é obtida do prolongamento linear da figura "joelho" (Figura 1A e 1B) do qual são extraídos os coeficientes lineares (Figura 1C).

$$\ln C(r) = d \ln r \quad \text{(Eq. 3)}$$

Afere-se a dimensionalidade mínima, m, do espaço de fase dentro do qual o atrator mencionado está embutido. O parâmetro m define o número mínimo de variáveis que devem ser consideradas na descrição da dinâmica do sistema, ou seja, o número mínimo de equações diferenciais de primeira ordem que podem conter as características qualitativas do sistema dinâmico estudado (Grassberger & Procaccia, 1983). Importante ressaltar que D_c é necessariamente menor que m.

Ao analisar a composição desse cálculo pode-se verificar que dependendo da quantidade de dados, do número de dimensões e de raios escolhidos, o tempo de resposta do algoritmo poderá ser muito grande. Uma possibilidade para otimizar esse tempo é utilizando a programação para GPU, denominado GPGPU (General Purpose Computing on GPU), sendo utilizado para este estudo o OpenCL. A programação para GPGPU é utilizada desde 1978, sendo amplamente utilizada no mundo científico atualmente, como pode-se ver em Weigel, 2011; Yamanaha et al, 2011 entre outros.

2. Estrutura OpenCL

OpenCL é um padrão aberto para programação paralela de plataformas heterogêneas que fornece um acesso de alto e baixo nível para processos em dispositivos paralelos. Funciona tanto em CPU's, quanto em GPU's de vários fabricantes (Augusto & Barbosa, 2012, NVIDIA). OpenCL é também um framework que inclui uma linguagem, bibliotecas e uma API (Khronos).

A figura 2 mostra como o OpenCL realiza a paralelização, sendo que X e Y determinam a quantidade de Work-itens (processos ou threads) que irão existir na placa. Cada conjunto p de work-itens determina um work-group, que é executado em um determinado núcleo. Essa divisão pode ser realizada entre uma e três dimensões, sendo obrigatório que o resultado da multiplicação da quantidade de work-groups de cada dimensão não deve ultrapassar a quantidade de work-groups máxima da placa. Outra obrigatoriedade é que a quantidade de work-itens de cada dimensão deve ser múltipla da quantidade de work-group da dimensão em questão.

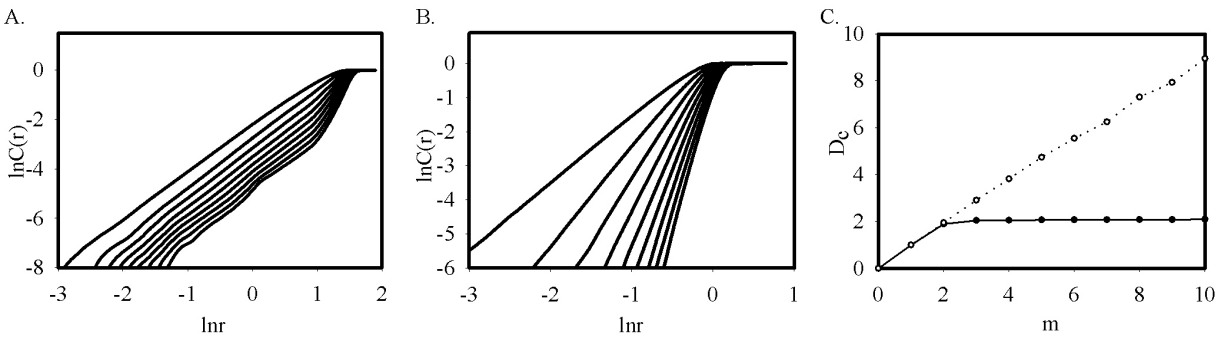

Figura 1:(a) ln C(r) vs ln (r) para valores crescentes de m para o atrator reconstruído a partir da série x de Lorenz e (b) lnC(r) vs ln (r) para uma série de dados aleatórios; e (c) saturação da dimensão de correlação versus dimensão de imersão m para o atrator reconstruído a partir da série x de Lorenz com Dc ≈ 2,05±0,01e m = 3 (●) e a instauração para a série aleatória com dimensão infinita (o).

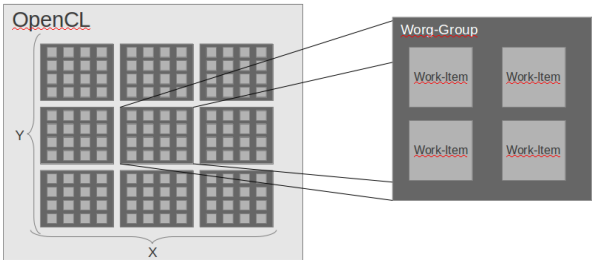

Figura 2: OpenCL

Essa estrutura permite paralelizar uma gama de problemas que necessitam de muito tempo de processamento. Com isso divide-se tarefas em tarefas menores, que são executadas simultaneamente, gerando ganho de desempenho para resolução do problema aplicado.

3. Testes

Para a realização dos testes foi utilizado uma máquina com uma placa de vídeo NVIDIA GT 520m que continha 48 núcleos com velocidade de 1,5 GHz. Essa máquina possuía um processador I5 com 4 núcleos de 2,30 GHz cada, e 6 GB de RAM, com o Ubuntu 11.10 64 bits.

Para os testes foram desenvolvidos dois softwares: um para OpenCL utilizando a placa de vídeo e outro com programação sequencial, ambos criados em Java. Os testes basearam-se nos 100.000 mil dados gerados da equação de Lorenz para o eixo X, sendo utilizado somente uma fração desses dados por teste, (executou-se os testes com 1.000, 2.000, 4.000, 8.000, 16.000 e 32.000 dados, tanto com OpenCL, quanto com a programação sequencial). Ao final dos testes foram confrontados os resultados para verificação dos cálculos e validação dos mesmos. Outra variação de teste, além da quantidade de dados, foi a variação na quantidade de raios e dimensões para o cálculo.

As tabelas a seguir mostram os algoritmos utilizados para os testes. Ao analisá-los percebe-se que os loops das linhas dois e três do algoritmo sequencial já não existem no algoritmo do OpenCL. Isso se deve a combinação dos valores desses dois loops para realizar a paralelização dentro da GPU através de um vetor resultante dos loops removidos. Essa combinação dos vetores e a leitura dos dados foram feitos em Java e passados como parâmetro para o OpenCL executar na placa gráfica.

Tabela 1 – Algoritmo Sequencial

```
1   inicializa dados;
2   for d ← 2 to maxDim do
3     for lr ← minR to maxR do
4       for i ← 0 to tamanho_dos_dados – 1 do
5         for j ← i + 1 to tamanho_dos_dados do
6           for n ← 0 to d do
7             soma as distâncias ao quadrado
8           endfor
9           executa a função Heaviside()
10        endfor
11      endfor
12    endfor
13  endfor
```

Tabela 2 – Algoritmo OpenCL

```
1   d = get_global_id(0)
2   for i ← 0 to tamanho_dos_dados – 1 do
3     for j ← i + 1 to tamanho_dos_dados do
4       for n ← 0 to d do
5         soma as distâncias ao quadrado
6       endfor
7       executa a função Heaviside()
8     endfor
9   endfor
```

O tempo gasto para a execução de cada algoritmo foi medido e plotado em gráficos para melhor análise. As figuras a seguir ilustram o comportamento dessa medida entre o algoritmo OpenCL e o sequencial:

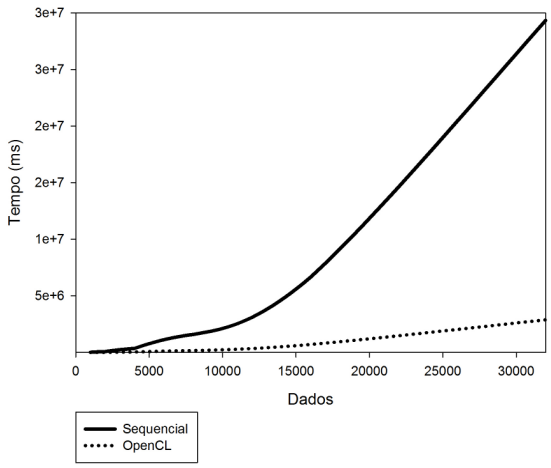

Figura 3: Resultado da variação da quantidade de dados, sendo o eixo X representado pela quantidade de dados e o eixo Y pelo tempo de resposta em milissegundos (ms). A linha contínua representa o desempenho do algoritmo sequencial e a linha pontilhada o desempenho do algoritmo OpenCL

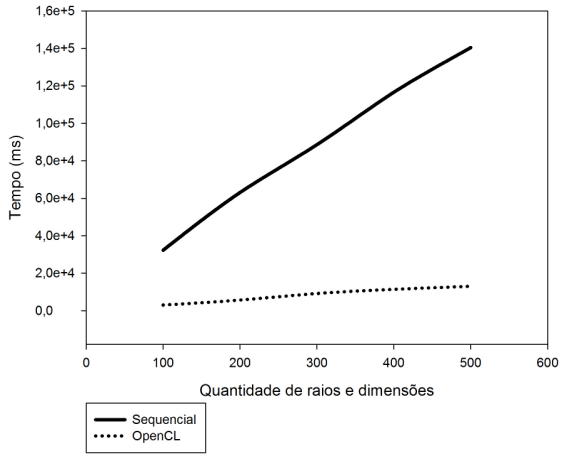

Figura 4: Resultado da variação de dimensões e raios, sendo o eixo X representado pela quantidade de raios e dimensões e o eixo Y pelo tempo de resposta em milissegundos (ms). A linha contínua representa o desempenho do algoritmo sequencial e a linha pontilhada o desempenho do algoritmo OpenCL

Ao observar as figuras 3 e 4 observa-se que a utilização do OpenCL para o cálculo da dimensão fractal mostrou-se muito eficiente. Isso se deve ao fato da fórmula permitir sua paralelização, pois pode-se calcular distâncias diferentes (R_i, …. R_n) junto com dimensões diferentes (D_i, … D_n) em paralelo dependendo da quantidade de work-groups permitida pela placa GPU. Assim, quanto mais núcleos a placa GPU tiver, mais work-groups ela poderá ter e consequentemente mais distâncias poderão ser paralelizadas.

A figura 3 nos permite visualizar que mesmo utilizando a mesma ideia para os algoritmos, o OpenCL tem um aproveitamento cada vez melhor, mesmo considerando que a velocidade da GPU (1,5 GHz) é menor que do CPU (2,3 Ghz). Isso talvez aconteça porque a troca entre as threads dentro de GPU é mais rápida pelo fato de serem lightweight threads, ou seja, utilizam recursos semelhantes.

3. Conclusão

Com a observação dos resultados pode-se concluir que a aplicação do OpenCL para o cálculo de Dimensão Fractal mostrou-se muito eficiente. O método aplicado mostrou redução de até 90% do tempo para a obtenção dos resultados. Assim, a compreensão e estudos de novos fenômenos através da Teoria da Complexidade poderá ser realizada de forma mais rápida, possibilitando respostas ágeis para o mundo científico.

Agradecimentos

Este trabalho tem apoio do Programa de Pós-Graduação em Física Ambiental na Universidade Federal de Mato Grosso junto com o CNPq.

Referências

Weigel, M., 2011. *Performance potential for simulating spin models on GPU*. Journal of Computational Physics 231, 3064.

Yamanaka, A., Aoki, T., Ogawa, S., Takaki, T., 2011. *GPU-accelerated phase-field simulation of dendritic solidification in a binary alloy*. Journal of Crystal Growth, Volume 318 (1), 40–45.

Augusto, D. A., Barbosa, H. J. C., 2012. *Accelerated parallel genetic programming tree evaluation with OpenCL*. Journal of Parallel and Distributed Computing.

NVIDIA, *NVIDIA OpenCL Jump Start Guide*. Disponível em: http://developer.download.nvidia.com/OpenCL/NVIDIA_OpenCL_JumpStart_Guide.pdf [Acessado em 23 abril 2012].

Khronos OpenCL Working Group, *The OpenCL specification, Version 1.1*. Disponível em: http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf [Acessado em 23 Abril 2012].

Nussenzveig, H. M., 2000. *Complexidade e Caos*. Editora URFJ/COPEA. Revista Brasileira de Ensino de Física, vol. 22, no. 2.

Takens, F., 1981. *Detecting Strange Attractors in Turbulence*. Lecture Notes in Mathematics, v.898: p.366-381, 1981.

Baker, G. L., Gollub, J. P., 1996. *Chaotic dynamics: an introduction*. Cambridge University Press: New York,. 256p.

Grassberger, P.; Procaccia, I., 1983. *Characterization of strange attractors*. Physical Review Letters, v.50, n.5, p.346-349.