6ª FEIRA DE INICIAÇÃO CIENTÍFICA E EXTENSÃO
FICE
05 e 06 de setembro

OTIMIZAÇÃO DE DESEMPENHO NA GERAÇÃO DE MATRIZES DO FRACTAL DE JULIA UTILIZANDO PYTHON E A TECNOLOGIA CUDA

Iury Krieger[1] ; José Luiz Bermudez[2] ; Tiago Heineck[3]

INTRODUÇÃO

Com a rápida evolução e ubiquidade das GPUs, as unidades de processamento gráfico (GPU) antes utilizadas apenas como processadores gráficos, hoje são utilizadas como processadores paralelos de propósito geral, tornando-se a plataforma com o maior custo benefício de processamento. Estas unidades executam milhares de *threads* ao mesmo tempo, permitindo a viabilidade de aplicações antes consideradas com tempo de execução elevado, através de seu vasto poder de paralelismo (NICKOLLS; DALLY, 2010).

Para testar o desempenho desta arquitetura projetada para o processamento de grandes conjuntos de dados (FAN *et al.*, 2004), este trabalho implementou a geração de matrizes de densidade para um dos mais notáveis conjuntos de fractais, o conjunto fractal de Julia (MANDELBROT,1983). Uma vez que tais matrizes geram uma quantidade significativa de processamento, além de um volume denso de dados, o processo foi divido em duas partes: (1) A geração de matrizes de densidade representando o fractal de Julia, através de um algoritmo matemático processado pela tecnologia CUDA e (2) o armazenamento das matrizes geradas em forma de imagem, referenciando-as em um banco de dados não relacional. Dessa forma, foi possível medir o desempenho da GPU ao processar inúmeras matrizes, contrastando o tempo de execução com um mesmo algoritmo implementado utilizando a CPU para processar tais matrizes.

[1] Aluno do Instituto Federal Catarinense, Videira. Bacharelado em Ciência da Computação. E-mail: iurykrieger96@gmail.com
[2] Aluno do Instituto Federal Catarinense, Videira. Bacharelado em Ciência da Computação. E-mail: luiz.bermudez27@gmail.com
[3] Professor orientador do Instituto Federal Catarinense, Videira. Bacharelado em Ciência da Computação. E-mail: tiago.heineck@ifc-videira.edu.br

PROCEDIMENTOS METODOLÓGICOS

O conjunto de Julia refere-se a uma categoria de fractais, formas geométricas não euclidianas, uma vez que dimensões podem ser comprimidas de forma fracionada em uma mesma forma geométrica. Além disso, o padrão de criação de formas fractais é totalmente distinto do padrão euclidiano, constantemente utilizado para entender os padrões da natureza. Conforme Mandelbrot (1983) apresenta, o termo fractal refere-se a uma forma onde suas estruturas geométricas complexas se repetem em qualquer escala.

Conforme apresentado por Owens *et al.* (2008), a computação em GPU fornece bons ganhos de desempenho a aplicações que tenham um certo grupo de características: (1) o requerimento computacional grande, ou seja, a necessidade do processamento de grandes volumes de dados, (2) o paralelismo substancial, quando as operações executadas são naturalmente paralelas, não tendo influência direta de seus resultados sobre os próximos e (3) a vazão mais importante que a latência, onde o tempo de acesso ao dado não se torna mais o gargalo, mas sim o tempo de processamento. Todas essas características estão presentes na geração de matrizes de densidade do conjunto fractal de Julia, uma vez que o processamento dessas matrizes é substancial, possuindo um grande requerimento computacional para cada índice a ser processado na matriz.

Tendo em vista a dificuldade do processo de programação paralela, a tecnologia CUDA surge como forma de implementação mais abstrata da computação em GPU, fornecendo uma série de facilidades para dispositivos NVidia, tais como acesso a memória compartilhada, barramento sincronizado, grupos de *threads* hierárquicos, etc. (NICKOLLS; DALLY, 2008). Estas características fazem com que o modelo de programação CUDA permite aos desenvolvedores explorar o paralelismo de forma a escrever um código em C natural e conciso (LUEBKE, 2008).

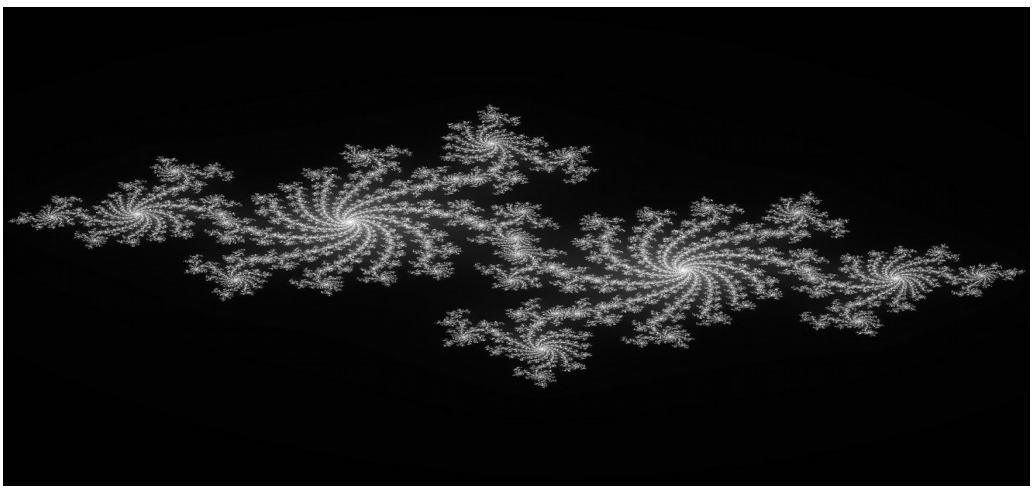
Para aumentar mais ainda o nível de abstração, levando em consideração a quantidade de operações a serem feitas utilizando matrizes e alocação dinâmica de memória, foi utilizada a biblioteca **CUDA** presente no pacote **NumbaPro** da linguagem Python. Tal biblioteca transcreve o código *python* em funções Kernel da extensão CUDA para a linguagem C, carregando o código C diretamente no

dispositivo de forma dinâmica, facilitando de forma significativa o desenvolvimento do algoritmo (ANALYTICS, 2014).

Sendo assim, o algoritmo desenvolvido para o processamento do conjunto fractal de Julia pode ser descrito da seguinte forma: primeiramente a CPU cria uma matriz C de tamanho *nm* preenchida com zeros, sendo *n* a largura da imagem e *m* sua altura. Tal matriz *C* é copiada para a memória da GPU, através de comandos da biblioteca CUDA, resultando em uma matriz *G* de iguais dimensões. A GPU, por sua vez, efetua 255 iterações da constante complexa que determina o fractal de Julia para cada índice $\{n_i , m_j\}$ presente na matriz *G*. Se a resultante *z* destas iterações conter algum valor maior que 50, assume-se que esta resultante está dentro do conjunto de Julia e que a mesma tende a infinito, preenchendo o índice $\{n_i , m_j\}$ com o número de iterações necessárias para a resultante *z* atingir o valor próximo de 50. Do contrário, o valor total de iterações é retornado (255). Após a GPU preencher a matriz *G* com os valores correspondentes as iterações, a mesma é copiada de volta à memória da CPU, apontando novamente para a matriz *C* anteriormente gerada.

Uma vez que a CPU obtém a matriz com seus valores já calculados, a matriz *C* é salva como bitmap em formato *.jpg*, gerando uma imagem do fractal em tons de cinza, variando de 0 a 255 para cada índice $\{n_i , m_j\}$, representado na imagem através de *pixels*. A imagem resultante da matriz calculada pela GPU, bem como suas representações de variações do conjunto em tons de cinza podem ser vistas na figura 1.

Figura 1 – Exemplo de imagem do fractal gerado.



Fonte: O autor.

Analisando a figura 1 pode-se perceber que a imagem gerada representa a função *f(c)*, *c* = -0.70176-0.3842*i*, sendo a parte real (-0.70176) fixa na geração do conjunto, enquanto o valor imaginário (-0.3842*i*) é atingido através da iteração sobre um intervalo pré definido no início do algoritmo. Dessa forma, cada matriz é gerada a partir do valor fixo real junto ao atual valor imaginário presente na iteração.

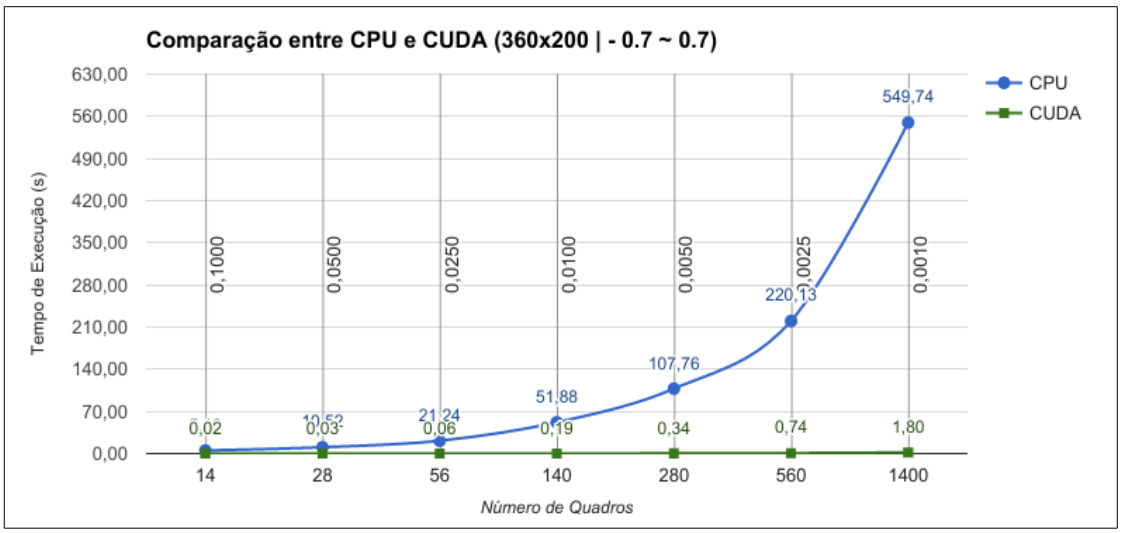
Sendo assim, o intervalo pré definido no início do algoritmo e a precisão de cada iteração são as métricas que definem a quantidade de iterações e, por sua vez, a quantidade de quadros gerados pelo algoritmo para representar tal variação. Por exemplo, utilizando um intervalo imaginário de -0.7 a 0.7 e uma precisão de 0.01, são obtidos 140 quadros, um para cada iteração do algoritmo.

Tais imagens geradas são referenciadas em um banco de dados relacional com um identificador único, possibilitando sua posterior recuperação e compilação em um arquivo *.gif*, possibilitando a representação de toda a variação do fractal em apenas alguns segundos.

RESULTADOS E DISCUSSÕES

Para verificar a melhora de desempenho obtida na geração das matrizes através do uso da tecnologia CUDA, foi desenvolvido um algoritmo com resultado computacional idêntico, porém executado apenas utilizando a CPU. Os resultados do tempo de execução do algoritmos, tanto da CPU quanto da GPU utilizando a tecnologia CUDA foram confrontados, utilizando como métrica a quantidade de quadros gerados pelo algoritmo. O gráfico comparativo do desempenho de cada algoritmo pela quantidade de quadros pode ser visto na figura 2.

Figura 2 – Tempo de processamento entre CPU e CUDA.



Fonte: O autor.

Levando em consideração um intervalo de -0.7 a 0.7 e imagens de tamanho 360x200, conforme apresentado na figura 2, pode-se notar a superioridade de desempenho da tecnologia CUDA em relação ao mesmo algoritmo executado pela CPU. Conforme a precisão aumenta gradativamente de 0.1 a 0.001 e, consequentemente, mais quadros são gerados, nota-se que a curva de tempo de execução pela CPU torna-se exponencial. Em contrapartida, a GPU mantém o tempo de execução do algoritmo abaixo dos 2 segundos, resultando em uma execução até **305 vezes** mais rápida nos testes efetuados, demonstrando uma superioridade de desempenho em relação a CPU.

Devido ao fato da tecnologia CUDA possuir um paralelismo natural em sua arquitetura, a mesma torna-se muito mais eficiente no manuseio de matrizes e aplicações que requerem um grande processamento de dados. Dessa forma torna-se notável o benefício da utilização da biblioteca, uma vez que apenas alguns ajustes no código Python original foram necessários, resultando em uma adaptação completa do cálculo utilizando a GPU.

CONSIDERAÇÕES FINAIS

Devido a facilidade trazida pela tecnologia CUDA no manuseio da programação paralela executada na GPU, além do benefício de desempenho obtido

em aplicações que possuam paralelismo substancial e grandes volumes de dados a serem processados, a utilização da GPU como um processador de propósito geral torna-se não só viável, mas superior à CPU em inúmeros casos.

Além disso, é importante ressaltar o ganho de desempenho obtido na geração das matrizes do fractal de Julia, verificando certa inviabilidade na execução do algoritmo pela CPU em casos com mais de 1400 quadros. Sendo assim, em casos onde o aumento de cálculos a serem feitos segue uma linha exponencial, ou onde tais dados não possuam influência sobre os próximos, a utilização da GPU torna-se não só uma opção de implementação mas também uma garantia de escalabilidade maior que a fornecida pela CPU, característica muitas vezes decisiva na viabilidade de uma aplicação.

REFERÊNCIAS

ANALYTICS, Continuum. **Numba**. (https://docs.continuum.io/numbapro/CUDAJit). Acesso : 14/05/2017.

FAN, Zhe et al. **GPU cluster for high performance computing**. In: Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference. IEEE, 2004. p. 47-47.

LUEBKE, David. **CUDA: Scalable parallel programming for high-performance scientific computing**. In: Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on. IEEE, 2008. p. 836-838.

MANDELBROT, Benoit B.; PIGNONI, Roberto. **The fractal geometry of nature.** New York: WH freeman, 1983.

NICKOLLS, John et al. **Scalable parallel programming with CUDA**. Queue, v. 6, n. 2, p. 40-53, 2008.

NICKOLLS, John; DALLY, William J. **The GPU computing era**. IEEE micro, v. 30, n. 2, 2010.

OWENS, John D. et al. GPU computing. **Proceedings of the IEEE**, v. 96, n. 5, p. 879-899, 2008.