Mandelbrot *Streaming* para Sistemas Multi-core com GPUs

Charles M. Stein[1], **João V. Stein**[1], **Leonardo B. Caitano**[1],
Dinei A. Rockenbach[1,2], **Dalvan Griebler**[1,2]

[1]Laboratório de Pesquisas Avançadas para Computação em Nuvem (LARCC)
Faculdade Três de Maio (SETREM) – Três de Maio – RS – Brasil

[2]Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

{charles.mst,joaovstein,leonardoboz,dineiar}@gmail.com

Resumo. *Este trabalho visa explorar o paralelismo na aplicação Mandelbrot* Streaming *para arquiteturas multi-core com GPUs, usando as bibliotecas FastFlow, TBB e SPar com CUDA. A implementação do paralelismo foi baseada no padrão farm, alcançando speedup de 16x no sistema multi-core e de 77x em um ambiente multi-core com duas GPUs. Os resultados evidenciam um melhor desempenho no uso de GPUs embora tenham sido identificadas futuras melhorias.*

1. Introdução

Com a limitação do aumento de frequência dos núcleos do processador desde o início do século, o uso do paralelismo se tornou o principal meio para aumentar o desempenho em aplicações que exigem altas demandas computacionais. Porém, as dificuldades da programação de software paralelo limitaram o uso deste paralelismo disponível no hardware. Esta limitação se mostrou ainda mais crítica com o surgimento de novas arquiteturas altamente paralelas (*many-thread*), como as GPUs, que têm liderado a busca por desempenho bruto em quantidade de operações por segundo. A integração entre código sequencial com paralelo em sistemas *multi-core* e GPUs apresenta vários desafios, principalmente devido às diferenças entre essas arquiteturas. Buscando facilitar a integração entre estes paradigmas e permitir o desenvolvimento de código paralelo por desenvolvedores de aplicação sem conhecimento detalhado do funcionamento do hardware, surgiram ferramentas como CUDA, FastFlow [Aldinucci et al. 2014], TBB [Reinders 2007] e SPar [Griebler et al. 2017]. O desafio é paralelizar com essas interfaces de programação paralela a aplicação Mandelbrot *Streaming*, a qual é baseada em uma equação matemática recursiva, ideal para a implementação do paralelismo devido à alta taxa de computação em relação ao tráfego de dados [Brodtkorb and Hagen 2010].

Dentre os trabalhos relacionados, o estudo de [Huseinović and Ribić 2015] apresenta a implementação do Mandelbrot com o uso de GPU e a biblioteca OpenCL. O artigo não apresenta os dados necessários para calcular o *speedup*. [Brodtkorb and Hagen 2010] implementaram o Mandelbrot em GPU e CPU, evidenciando os benefícios e dificuldades de cada. Em seu estudo foi obtido o speedup de 40x em GPU com CUDA e 1,81x em CPU com OpenMP. O presente trabalho se diferencia de [Huseinović and Ribić 2015] e [Brodtkorb and Hagen 2010] ao realizar as implementações utilizando processamento de *stream* em CPU com GPUs, utilizando as bibliotecas SPar, TBB, FastFlow e CUDA. [Löff et al. 2017] explora a versatilidade e desempenho do padrão paralelo *farm* em um

sistema *multi-core* através da biblioteca FastFlow com os algoritmos Mandelbrot e K-Means. [Griebler et al. 2017] introduz a biblioteca SPar, implementa e analisa os resultados em um algoritmo renderizador do conjunto de Mandelbrot. Ainda que [Griebler et al. 2017] e [Löff et al. 2017] utilizem processamento de *stream*, os mesmos não fazem o uso de GPU. Neste trabalho explora-se o paralelismo de *stream* em CPU e de dados em GPU utilizando CUDA no algoritmo Mandelbrot.

O trabalho tem como principal objetivo fazer a análise do desempenho da paralelização da aplicação Mandelbrot *Streaming* para arquiteturas *multi-core* com GPUs. Não foram utilizadas outras aplicações do tipo Mandelbrot com a finalidade de comparação, uma vez que existem diferenças no formato de execução, pois estas não levam em consideração o tempo de renderização da imagem e não permitem visualizar a imagem em tempo de execução, uma característica trazida pelo processamento via *stream*, processamento que tem um fluxo contínuo e independente. Assim, o artigo apresentada a implementação do paralelismo na Seção 2 e os resultados de desempenho da aplicação são analisados na Seção 3.

2. Exploração do Paralelismo para Sistemas *Multi-Core* com GPUs

De acordo com [McCool et al. 2012], o conjunto de Mandelbrot é um conjunto de todos os pontos c, em um plano complexo, que não vão ao infinito quando a função $z = z^2 + c$ está em iteração. Devido à sua função ser independente, o cálculo do Mandelbrot pode ser efetuado em paralelo. Inicialmente implementou-se o cálculo do conjunto de Mandelbrot em CPU fazendo o processamento dos números em *stream* através do padrão paralelo *farm*. Então, o trecho de código que realiza o cálculo para cada ponto da linha foi passado para a GPU a fim de aproveitar o paralelismo massivo da arquitetura. A Figura ao lado do Algoritmo 1 ilustra o fluxo de computação e indica como o código foi anotado usando a SPar. Apesar das diferenças de programação, o código foi estruturado da seguinte forma:

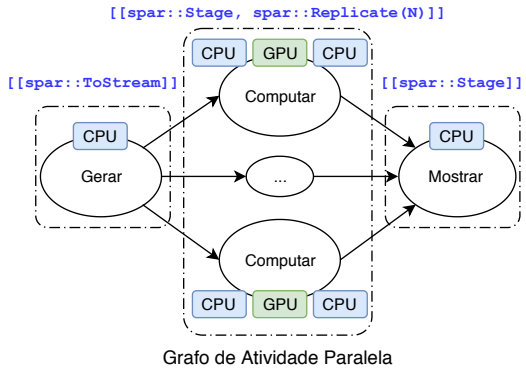
- **Gerar**: Executa um laço de repetição para cada linha x da imagem. A linha x é enviada para o segundo estágio.
- **Computar**: Recebe a linha x e executa o cálculo de Mandelbrot para cada ponto y da linha, armazenando os resultados em um vetor. O vetor resultante é enviado para o próximo estágio.
- **Mostrar**: Recebe o vetor de resultados e salva em uma imagem que está sendo mostrada interativamente.

Na implementação do algoritmo para GPU, foram realizados experimentos utilizando a mesma estratégia da CPU, modificando apenas o estágio **Computar** para que fizesse o *offload* do cálculo para a GPU. Nesse formato a carga de trabalho ficou muito baixa para cada *kernel* invocado, portanto seria necessário que fossem passados mais elementos para cada invocação do *kernel* da GPU. Desta forma, o estágio 1 do algoritmo foi modificado para que cada iteração processasse 1024 elementos x por vez, aumentando assim a carga de trabalho da GPU. O fluxo do Mandelbrot em CPU e GPU está representado na Figura ao lado do Algoritmo 1. Os estágios 1 e 3 permaneceram idênticos nas duas implementações, porém o segundo estágio realiza as operações com o uso de GPU. Além disso, o segundo estágio suporta multi-GPU através da replicação do estágio conforme o número de GPUs disponíveis, onde cada réplica fica responsável por coordenar as tarefas de uma GPU. A distribuição do trabalho para as GPUs é implementada através

de um escalonamento *round-robin*. A réplica que está rodando na CPU é responsável por encaminhar o item do *stream* para a GPU definida previamente no estágio que gera a computação. O *kernel* utilizado para realizar a computação em GPU no segundo estágio pode ser vista no Algoritmo 1. Neste algoritmo, nota-se que os principais delimitadores de carga computacional da aplicação são o número de iterações ($NITER$) e a dimensão da imagem resultante (DIM). Estas variáveis aumentam a complexidade computacional do *loop* da linha 10 e na quantidade de invocações do *kernel*, respectivamente.

Algoritmo 1 Kernel GPU do Mandelbrot

```
1:  DIM ← 1024, NITER ← 200000
2:  procedure M(init_a, init_b, range, step, image)
3:      current ← getGpuGlobalIndex()
4:      j ← mod(current, DIM)
5:      i ← div(current, DIM)
6:      im ← init_b + step * i
7:      a ← init_a + step * j
8:      b ← im
9:      k ← 0
10:     for k, NITER do
11:         a2 ← a * a
12:         b2 ← b * b
13:         if (a2 + b2) > 4.0 then break
14:         b ← 2 * a * b + im
15:         a ← a2 − b2 + cr
16:     image[j] ← 255 − ((k * 255/niter))
```



Grafo de Atividade Paralela

3. Análise dos resultados

Para avaliar o desempenho das implementações foi executado um *benchmark* com as mesmas. Os testes foram executados em um ambiente com duas GPUs Titan XP (cada uma com 12GB de memória RAM), um processador Intel(R) Core(TM) I9-7900X CPU (20 *threads* considerando o *hyper-threading*), 32GB de RAM e 2TB de disco. O sistema operacional utilizado foi o Ubuntu Server 18.04 (Kernel 4.15.0-38-generic). Foram utilizados os compiladores g++ 7.3.0 e nvcc 10 com a opção $-O3$. Os testes que visavam somente a CPU foram executados utilizando os estágios replicados com valores entre 2 e 20. Nas versões com GPU, os testes foram executados com uma e duas GPUs. Para todas as aplicações, foram utilizadas como parâmetros para o Mandelbrot as dimensões 1024x1024, com 200000 iterações. Os testes foram repetidos 5 vezes para cada aplicação para então calcular a média aritmética do tempo das execuções.

Na Figura 1 são apresentados os *speedups* das versões em CPU com 20 *threads*, pois trouxe os melhores resultados, e as versões em GPU com o uso de uma e duas GPUs, tendo como base o tempo sequencial de 104 segundos. Pode-se observar que foi possível aproveitar o paralelismo massivo da GPU, obtendo um *speedup* de 44x com 1 GPU e 77x com 2 GPUs. Nota-se que o aumento de GPUs não representa uma escalabilidade linear. As análises permitiram constatar que isso ocorre devido ao escalonamento *round-robin* utilizado para escolher a GPU da tarefa, uma vez que foi descoberto que os itens do *stream* possuem custo computacional diferente. Isso também afeta o uso da GPU, como pode ser visto na Figura 2, que apresenta o gráfico de uso gerado pelo NVIDIA Visual Profiler (`nvvp`). Para resolver este problema, identificou-se que seria necessário implementar um escalonamento sob demanda para a distribuição dos itens do *stream* entre as GPU.

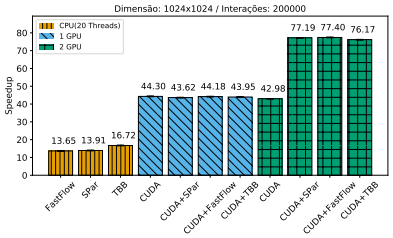


Figura 1. Resultados.

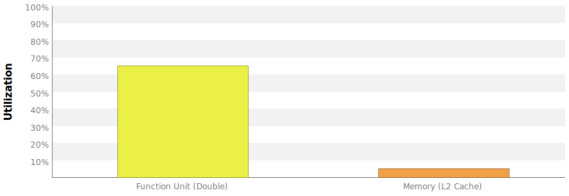


Figura 2. Análise do uso de GPU.

Em relação às bibliotecas utilizadas para o paralelismo de CPU, é possível observar que SPar e FastFlow apresentaram resultados semelhantes, uma vez que a SPar é uma abstração do FastFlow. O TBB conseguiu um melhor desempenho na CPU, enquanto que na GPU ele acabou sendo pior que as outras bibliotecas. A causa desta perda pode ter sido em função da política de distribuição ser *round-robin* para GPU, que não se adequou ao escalonamento padrão do TBB.

4. Conclusões

Este artigo apresentou a implementação do paralelismo na aplicação Mandelbrot *Streaming* para arquiteturas multi-core com GPUs usando diferentes interfaces de programação paralela. Devido ao alto grau de paralelismo que esta aplicação apresenta, foi possível alcançar um *speedup* de 77x com duas GPUs, 44x com uma GPU e 16x com apenas o uso de CPU. Neste sentido, identificou-se que o uso de GPU representou um melhor desempenho que a CPU nesta aplicação, porém, ainda são possíveis novas otimizações no escalonamento e balanceamento de carga da aplicação. Como trabalhos futuros, busca-se implementar outras aplicações de *stream* para estudar o uso das GPUs. Além disso, a geração de código CUDA automaticamente a partir dos atributos da SPar pode ser implementada, visando abstrair e facilitar a programação para GPU para os programadores.

Referências

Aldinucci, M., Danelutto, M., Kilpatrick, P., and Torquati, M. (2014). *Fastflow: High Level and Efficient Streaming on Multicore*, chapter 13, pages 261–280. Wiley-Blackwell.

Brodtkorb, A. and Hagen, T. (2010). A comparison of three commodity-level parallel architectures: Multi-core CPU, cell BE and GPU. In *Mathematical Methods for Curves and Surfaces: 7th International Conference, MMCS 2008, Tønsberg, Norway, June 26-July 1, 2008, Revised Selected Papers*, pages 70–80.

Griebler, D., Danelutto, M., Torquati, M., and Fernandes, L. G. (2017). SPar: A DSL for High-Level and Productive Stream Parallelism. *Parallel Processing Letters*, 27(01):1740005.

Huseinović, A. and Ribić, S. (2015). Benchmark comparison of computing the mandelbrot set in opencl. In *2015 23rd Telecommunications Forum Telfor (TELFOR)*, pages 994–997.

Löff, J., Griebler, D., Ledur, C., and Fernandes, L. G. (2017). Explorando a Flexibilidade e o Desempenho da Biblioteca FastFlow com o Padrão Paralelo Farm. In *17th Escola Regional de Alto Desempenho do Rio Grande do Sul (ERAD/RS)*, page 4.

McCool, M., Reinders, J., and Robison, A. (2012). *Structured Parallel Programming: Patterns for Efficient Computation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

Reinders, J. (2007). *Intel Threading Building Blocks*. O'Reilly, Sebastopol, CA, USA.