Implementação de uma Aplicação de Simulação Geofísica em OpenCL

Arthur Mittmann Krause[1], **Matheus da Silva Serpa**[1], **Philippe Olivier Alexandre Navaux**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{amkrause,msserpa,navaux}@inf.ufrgs.br

Resumo. *A indústria energética recorre a aplicações de simulação que são normalmente implementadas em CUDA ou OpenMP, o que reduz a portabilidade do programa e prende a empresa a um fabricante específico de hardware. Neste trabalho é apresentada uma implementação no padrão aberto OpenCL de um software de simulação geofísica que apresenta desempenho comparável à versões equivalentes em CUDA e OpenMP.*

1. Introdução

Há quase duas décadas, as GPUs são cada vez mais usadas para aceleração de aplicações de propósito geral massivamente paralelas (GPGPU), graças ao grande número de unidades de processamento e largura de banda da memória desses dispositivos [Nickolls and Dally 2010]. Um grande facilitador da popularização de GPGPU são as APIs que permitem aos programadores não precisarem abstrair seus problemas em operações gráficas para que executem em GPUs, entre elas a mais conhecida é CUDA.

Uma outra alternativa é o OpenCL, um *framework* para desenvolvimento de programas não apenas para CPUs e GPUs, mas também para DSPs, FPGAs e demais dispositivos [Stone et al. 2010]. OpenCL especifica uma linguagem baseada em C, um modelo de hierarquia de memória e APIs para controlar esses dispositivos, e principalmente uma interface de programação paralela. A principal vantagem do OpenCL é que o código é portável, permitindo que um mesmo código possa ser executado tanto em uma GPU quanto em uma CPU sem qualquer alteração. Outra vantagem é que ele é um padrão aberto, ou seja, um software feito em OpenCL não estará atrelado a um determinado fabricante de hardware como no caso do CUDA.

Uma classe de algoritmos muito apropriada para paralelização em GPUs são os códigos de estêncil. Nesse tipo de código, uma operação é realizada em cada um dos elementos de um arranjo de forma independente, podendo ser realizadas em paralelo, obtendo assim grande eficiência quando executados em GPUs.

Neste trabalho, uma paralelização em OpenCL de uma aplicação de modelagem geofísica baseada em estêncil é apresentada, assim como o seu desempenho em comparação a versões paralelizadas em CUDA para GPUs e em OpenMP para CPUs.

2. Aplicação e Paralelização com OpenCL

Enquanto os combustíveis fósseis ainda são necessários para atender a demanda energética da sociedade, as empresas de energia precisam executar escavações com um custo que chega a centenas de milhões de dólares, e possuem uma precisão menor do que

50%. Essas empresas recorrem a softwares de simulação baseados em computação de alto desempenho como uma forma de reduzir custos e riscos.

$$\frac{1}{V^2}.\frac{\partial^2 p}{\partial t^2} = \nabla^2 p \tag{1}$$

Foi fornecida pela Petrobras uma aplicação geofísica que simula a propagação de uma *wavelet* ao longo do tempo através da resolução da equação isotrópica de propagação da onda (Equação 1). O laço principal do programa executa por um número de passos de tempo parametrizável, e tem duas partes: a primeira é a inserção da *wavelet* fonte através de um ponto no espaço definido como parâmetro, e a segunda é a propagação das ondas no meio. Através de uma implementação em OpenMP já existente, foi construída uma versão em OpenCL capaz de executar tanto em GPUs quanto em CPUs. A segunda parte do laço principal é um código estêncil, e foi escrita como um kernel OpenCL. A primeira parte consiste em breves operações não paralelizáveis, e foram estudadas três maneiras de implementá-la, cada uma com suas vantagens e desvantagens:

Como um kernel específico
: Com um kernel apenas para inserir a onda no meio evita-se a transferência de dados do host para o dispositivo, mas cria-se um overhead de compilação e chamada desse kernel.

No kernel da segunda parte
: Desta forma, continua-se com um único kernel e evita-se a transferência de dados do host para o dispositivo, mas no início do kernel é necessário verificar se é o *work-item* responsável por introduzir a onda fonte, o que adiciona um overhead.

Como parte do código do host
: Insere-se a onda fonte no código do host antes de chamar o kernel para a segunda parte. Isso adiciona a latência de leitura e escrita de uma posição de memória que reside no dispositivo, mas simplifica o código do kernel.

O método escolhido foi o terceiro, pois apresentou o melhor desempenho na maioria dos casos de teste para ambas as plataformas. Desta forma, o único kernel da aplicação executa o código estêncil de propagação da onda, calculando todas as posições de z para uma determinada posição no plano xy, para um intervalo de tempo. O cabeçalho do kernel ficou da seguinte forma:

```
__kernel void kernel_cte(global float *U0, global float *U1,
global float *VP0, uint stride, uint nnoi, constant float *g_W,
uint k0, uint k1, float FATMDFX, float FATMDFY, float FATMDFZ)
```

Os dados que precisam ser enviados ao dispositivo podem ser vistos pelo cabeçalho. A maior parte dos dados pertence aos três vetores de float que representam o espaço tridimensional. Dois deles (U0 e U1) representam a amplitude da onda em cada ponto para passos de tempo subsequentes, e o outro (VP0), contém a velocidade de propagação da onda no ponto, totalizando três floats para cada ponto do espaço. Foi utilizada apenas a memória global do dispositivo. Após o término da execução, apenas um desses vetores precisa ser lido de volta pelo host.

3. Desempenho

O desempenho da aplicação foi testado tanto com GPU quanto com CPU como plataforma de execução. Os resultados foram comparados à implementações equivalentes em CUDA e OpenMP. Como demonstrado em Serpa et al. [Serpa et al. 2017], o desempenho desta aplicação com OpenMP depende fortemente da ordem dos laços no laço principal, por alterar o padrão de acesso à memória e consequentemente a eficiência da memória cache. Portanto, o desempenho foi comparado tanto com uma versão simples, com a ordem dos laços inalterada, quanto com a versão demonstrada como mais eficiente.

3.1. Metodologia

O ambiente de execução consistiu em uma máquina que possui uma GPU NVidia Tesla P100 e dois processadores Intel Xeon E5-2699 v4, cada um com 22 núcleos operando em 2.2GHz e com *Hyper-threading* habilitado. A máquina possui o sistema operacional Ubuntu com o kernel do Linux 4.4.0-104 instalado. A versão do OpenCL utilizada foi a 1.2, a do CUDA foi a 9.0 e a do GCC foi a 5.4.0. A metodologia utilizada nesse trabalho se baseou na definição de um Projeto Experimental (*Design of Experiments*), tendo como fatores a API e a dimensão do espaço tridimensional de simulação. O número de passos de tempo foi fixado em 500. O experimento consiste em um projeto de fatorial completo, onde todas as combinações de fatores são executadas 15 vezes de forma aleatória, fazendo com que anomalias afetem estatisticamente de forma homogênea as diferentes combinações.

3.2. Resultados

A Figura 1 detalha os tempos de execução da implementação em OpenCL executada na GPU comparada à versão em CUDA, para os quatro tamanhos de entrada. A Figura 2 é semelhante, mas compara o desempenho na CPU com as versões em OpenMP.

A implementação em OpenCL apresentou um desempenho levemente inferior à versão em CUDA. Para um tamanho pequeno de entrada, o tempo de execução com OpenCL foi 80% maior, mas com a dimensão x aumentada em quatro vezes, o tempo com OpenCL chega a ser menor do que o com CUDA. Para tamanhos maiores, o desempenho com CUDA é superior, e torna-se cada vez melhor conforme aumenta-se o tamanho da simulação. Era esperado que o desempenho com CUDA fosse superior, pois diversos trabalhos como Fang et al. [Fang et al. 2011] mostram que a performance do CUDA pode ser até 30% melhor do que o OpenCL para implementações equivalentes da mesma aplicação.

Utilizando as CPUs como dispositivo de execução, o desempenho depende fortemente das dimensões da simulação. Para um tamanho pequeno, o tempo de execução é semelhante à melhor versão com OpenMP e 75% melhor que a versão padrão. Com os tamanhos médios, o desempenho é levemente pior do que a versão com OpenMP otimizada, mas aumenta a diferença para a versão padrão. Com uma entrada muito grande, a implementação em OpenCL apresenta um tempo de execução muito semelhante à OpenMP padrão, enquanto a OpenMP otimizada é 8.3x melhor que ambas. Este comportamento é esperado porque o desempenho dessa aplicação em CPUs é diretamente relacionado com o aproveitamento da memória cache, e é sabido que a versão OpenMP padrão faz um uso ineficiente da mesma, enquanto a OpenMP otimizada utiliza-a da melhor maneira possível. A implementação em OpenCL, como foi realizada de forma genérica tanto

para GPUs como CPUs, não toma nenhuma providência para melhor utilizar a cache, e acaba por depender totalmente do escalonamento dos *work-items* feito pelo *runtime* do OpenCL.

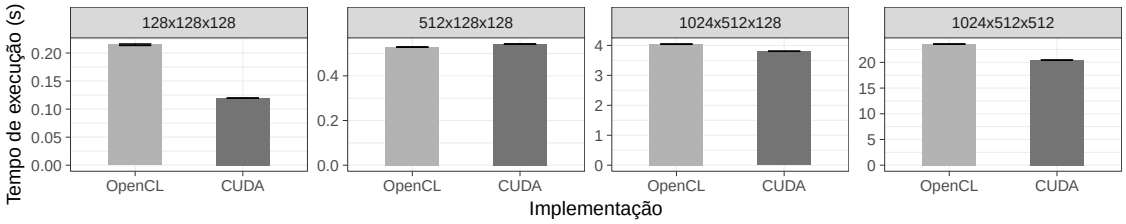

Figura 1. Tempos de execução das implementações baseadas em GPU

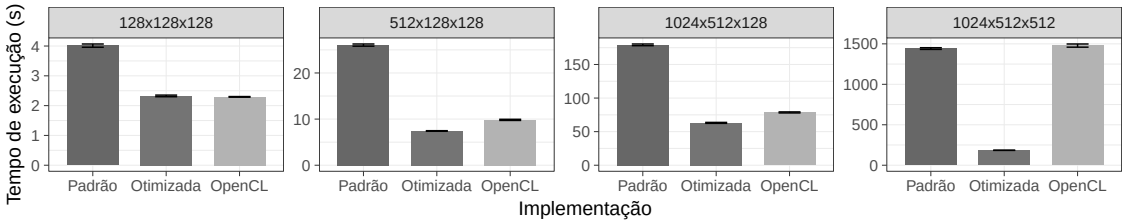

Figura 2. Tempos de execução das implementações baseadas em CPU

4. Conclusão

Este trabalho relata a implementação de uma versão em OpenCL de uma aplicação geofísica e analisa seu desempenho com uma GPU e CPU como plataformas, comparando à versões equivalentes em CUDA e OpenMP. O desempenho pode ser considerado satisfatório pois em muitos casos foi semelhante às outras versões, possuindo uma portabilidade ausente nas demais.

Como trabalho futuro, pretende-se adaptar a aplicação para utilizar as memórias específicas e mais rápidas das GPUs, e estudar como garantir um padrão de acesso à memória que forneça um melhor uso da cache quando executada em uma CPU.

Referências

Fang, J., Varbanescu, A. L., and Sips, H. (2011). A comprehensive performance comparison of cuda and opencl. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 216–225. IEEE.

Nickolls, J. and Dally, W. J. (2010). The gpu computing era. *IEEE micro*, 30(2).

Serpa, M. S., Cruz, E. H., Diener, M., Krause, A. M., Farres, A., Rosas, C., Panetta, J., Hanzich, M., and Navaux, P. O. (2017). Strategies to improve the performance of a geophysics model for different manycore systems. In *2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pages 49–54. IEEE.

Stone, J. E., Gohara, D., and Shi, G. (2010). Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66–73.