Comparativo de desempenho de execução de Algoritmos no CUDA e no OpenCL

Antonio Raian de Lima Mendes[1]; Angelo Amâncio Duarte[2]

1. Bolsista PIBIC/FAPESB, Graduando em Engenharia de Computação, Universidade Estadual de Feira de Santana, e-mail: raymendesjr2013@gmail.com

2. Orientador, Departamento de DETEC, Universidade Estadual de Feira de Santana, e-mail: angeloduarte@uefs.br

PALAVRAS-CHAVE: CUDA; OpenCL; Computação Paralela.

INTRODUÇÃO

Com o avanço tecnológico surgiu a necessidade de processar dados de forma mais rápida e eficaz. Dando origem à *Computação Paralera* (CP) (ROCHA, 2007/2008) e os processadores capazes de atender as necessidades da mesma que são os *multicore* ou os *manycore*. Um exemplo de processadores que se assemelham aos *manycore*, possuindo dezenas de unidades de processamento, são as *Placas Gráficas* (GPU). As *GPUs* foram projetadas para auxiliar em cálculos de uso intensivos dos processadores comuns, elas possuem muito mais núcleos de processamentos que permite a execução simultânea de operações idênticas sobre dados diferentes (SIMD) (DE PAULA, 2014).

Novas plataformas para gerenciar a implementação de algoritmos foram criadas e com elas outras tecnologias como *Application Programming Interface* (API) e *bibliotecas* foram aprimoradas. APIs e *bibliotecas* são interfaces usadas para acessar algumas funções existentes sem necessariamente saber como o código fonte funciona, a diferença básica é que as primeiras são mantidas por entidades e isso agrega uma confiabilidade maior aos serviços. Atualmente, as plataformas que mais se destacam para computação paralela usando GPU são: CUDA (*Compute Unified Device Architecture*), OpenCL (*Open Computing Language*) e Direct Compute (Microsoft).

A *CUDA* foi criada pela NVIDIA em 2006, com o intuito de otimizar a execução de algoritmos nas placas desenvolvidas por esta empresa. O *OpenCL*, desenvolvido pela *Khronos*, é uma plataforma aberta que comporta a paralelização de aplicações e, também, admite a elaboração de códigos heterogêneos podendo assim aproveitar tanto a CPU quanto a GPU (TSUCHIYAMA, 2010). Visto que no Laboratório de Computação de Alto Desempenho (LaCAD) da Universidade Estadual de Feira de Santana (UEFS) existem GPU da NVIDIA, foi possível utilizar ambas as plataformas. Dessa forma, esse trabalho teve o objetivo de testar e identificar, dentre elas, a que potencializa a criação de sistemas de alto desempenho para atender as necessidades do laboratório.

MATERIAIS E MÉTODOS

Inicialmente, para a obtenção do êxito da pesquisa através da fase de testes dividimos o trabalho em três etapas: A primeira consistiu em encontrar trabalhos já realizados que pudessem nortear nossa pesquisa. A segunda etapa configuramos os computadores com as devidas APIs e realizamos os testes. Finalmente, na terceira etapa usamos os dados encontrados para definir a plataforma conveniente para as

necessidades do LaCAD. Durante o tempo da pesquisa usamos a abordagem teórica de Lauro de Paula (DE PAULA, 2014) e a comparação de performance de Karime, Dikson e Hamze (KARIME, 2010) que conclui que o CUDA possui algumas vantagens sobre a OpenCL por conter algumas ferramentas que ajudam na criação de códigos e na alocação e recuperação de dados da GPU. Porém, o CUDA só pode ser utilizado em placas da NVIDIA enquanto a plataforma da *Khronos* é genérica e podendo ser empregada em placas de outras fabricantes.

Na etapa seguinte configuramos o ambiente para dar inicio à fase de testes e nos deparamos com alguns problemas de compatibilidade de hardware. Uma das placas disponíveis no laboratório, a GT 218, foi fabricada em 2008 e a NVIDIA deixou de dar suporte a este hardware, nos forçando a usar versões mais antigas da biblioteca CUDA. A configuração do ambiente gerou um tutorial que está disponível nos anexos do relatório (Anexo I) e no site do laboratório (lacad.uefs.br).

O procedimento utilizado para determinar a melhor plataforma foi o uso de *benchmarks* que são softwares que possuem testes computacionais para quantificar a diferença de desempenho, seja ele de cômputo, gráfico ou de transferência de dados, entre dois ou mais computadores ou sistemas computacionais (CANALTECH, 2014). Esses *softwares* buscam extrair o máximo desempenho do computador, geralmente com operações algébricas usando matrizes com muitos números de linhas e colunas, computação gráfica e testes de transferência de dados entre os processadores (*CPU* e *GPU).*

O *benchmark* que escolhemos para automatizar os testes foi o *ViennaCL* (VIENNACL, 2017), por possuir diversos testes escritos em linguagem C para CUDA e para OpenCL. Dessa maneira, pudemos comparar a mesma funcionalidade para ambas as plataformas.

O teste que utilizamos foi o *dense_blas* (Anexo II), que manipula matrizes densas e esparsas com muitas linhas e colunas. O dense_blas utiliza *pontos flutuantes* (*float* e *double*, forma computacional de representar valores reais), em processos aritméticos de multiplicação por valores, vetores e entre matrizes quantificando o cômputo e a taxa de transferência de dados entre as operações. Os principais métodos do *dense_blas* são mostrados na Tabela 1.

Tabela 1. Principais métodos do *dense_blas.*

Método	**Descrição**
bench	Esse método é o principal do *benchmark*, responsável por receber os parâmetros que são usados no métodos seguintes. Os parâmetros são: O tamanho dos vetores e os tamanhos das linhas e colunas das matrizes.
bench – BLAS_1	Esse método faz a cópia de um vetor para outro (COPY), a multiplicação de um vetor por um número (AXPY) e a multiplicação de dois vetores (DOT). Sempre usando o parâmetro BLAS1_N como tamanho dos vetores.
bench – BLAS_2:	Usando os parâmetros BLAS2_N e BLAS_2M, cria-se dois vetores

	X e Y e uma matriz A. Inicialmente multiplica-se o X pela matriz A (GEMV-N) e em seguida multiplica o vetor resultado Y pela matriz transposta de A (GEMV-T).
bench – BLAS_3	Com os parâmetros BLAS3_M, BLAS3_N, BLAS3_K cria-se três matrizes, A, B e C. Em seguida é feita a multiplicação de A por B (GEMM-NN), de A pela transposta de B (GEMM-NT), da transposta de A por B (GEMM-TN) e das transpostas de A e B (GEMM-TT). C é a matriz resultante das multiplicações.

RESULTADOS E DISCUSSÃO

A placa GT 218 não suporta cálculos usando os números de pontos flutuantes do tipo *double* e, por isso, os códigos não puderam ser executados por completo. Na execução com o CUDA (Figura 1), quando o código é inspecionado para verificar a compatibilidade com a placa observa-se a presença de cálculos com *double*, o qual aborta a execução do algoritmo e emite uma *exceção* de incompatibilidade. No entanto, na execução com o OpenCL (Figura 2), não é feita uma verificação prévia de compatibilidade e a *exceção* aparece somente quando tenta-se realizar os cálculos com double.

```
[antonio@localhost build]$ examples/benchmarks/dense_blas-bench-cuda
Benchmark : BLAS
-----------------
terminate called after throwing an instance of 'viennacl::backend::cuda::cuda_ex
ception'
  what():  /home/antonio/Desktop/BenchMark/ViennaCL-1.7.1/viennacl/linalg/cuda/v
ector_operations.hpp(820): : getLastCudaError() CUDA error 8: invalid device fun
ction  @ vector_assign_kernel

Aborted (core dumped)
```

Figura 1: Execução do *dense_blas* CUDA na placa GT 218.

```
[antonio@localhost build]$ examples/benchmarks/dense_blas-bench-opencl

------------------------------------------------
              Device Info
------------------------------------------------

Name:                GeForce 310
Vendor:              NVIDIA Corporation
Type:                GPU
Available:           1
Max Compute Units:   2
Max Work Group Size: 512
Global Mem Size:     536084480
Local Mem Size:      16384
Local Mem Type:      1
Host Unified Memory: 0


Benchmark : BLAS
-----------------
SCOPY : 6.34 GB/s
SAXPY : 6.87 GB/s
SDOT : 7.18 GB/s
SGEMV-N : 6.19 GB/s
SGEMV-T : 3.3 GB/s
Build Status = -2 ( Err = -5 )
Log:

Sources:
  __attribute__((reqd_work_group_size(16,16,1)))
__kernel void prod_NN(unsigned int M, unsigned int N, unsigned int K, __global float* obj0_pointer,unsigned int obj0_ld,unsigned
unsigned int obj0_stride2, __global float* obj1_pointer,unsigned int obj1_ld,unsigned int obj1_start1,unsigned int obj1_start2,uns
3_pointer,unsigned int obj3_ld,unsigned int obj3_start1,unsigned int obj3_start2,unsigned int obj3_stride1,unsigned int obj3_stri
```

Figura 2: Execução do *dense_blas* OpenCL na placa GT 218.

Tendo em vista que a execução não ocorreu como deveria na placa GT 218, usamos uma segunda placa da NVIDIA disponível no laboratório, a Tesla C2070. Essa placa por ser mais recente e conter um poder computacional maior permitiu executar os códigos por completo. Os dados gerados na execução do *dense_blas* na Tesla C2070

estão apresentados em forma de gráficos na figura 3 e ao analisar o gráfico da esquerda, constatamos que a quantidade de dados transferidos por segundo é quase a mesma entre as duas plataformas e no gráfico da direita, notamos que a diferença de Gflops (quantidade de operações por segundo) do código em OpenCL é consideravelmente maior que o do CUDA.

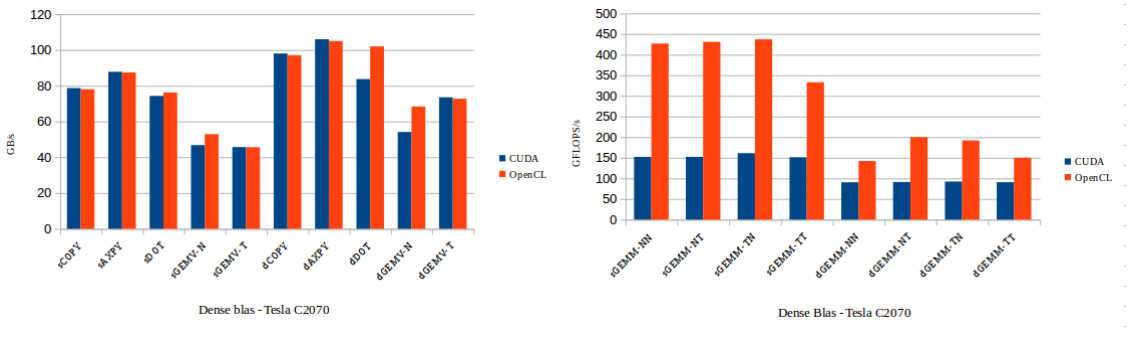

Figura 3: Gráfico dos dados gerados na execução do *dense_blas* na GPU Tesla C2070.

Com base nos dados coletados na execução do *dense_blas* na Tesla, podemos notar que em desempenho a plataforma OpenCL mostrou-se melhor. Portanto, levando em consideração as necessidades do laboratório e o que foi apresentado com os testes, podemos afirmar que o uso do OpenCL é mais adequado para o LaCAD, pois não é inferior em desempenho e possui compatibilidade com GPUs de vários fabricantes.

CONSIDERAÇÕES FINAIS

O propósito principal do trabalho realizado foi a definição da plataforma de criação de algoritmos de alto desempenho que melhor se adéqua aos projetos do LaCAD. Tendo em vista que existem diversas plataformas que auxiliam a criação de algoritmos de alto desempenho para execução em GPU, escolhemos as duas mais comuns atualmente, o CUDA e o OpenCL, criadas pela NVIDIA e pela Khronos, respectivamente.

Comparamos o desempenho de cômputo e da transferência de dados do CUDA e do OpenCL e para isto utilizamos o código *dense_blas* (Anexo II) do pacote de *benchmark*s do ViennaCL (VIENNACL, 2017) implementado para ambas as plataformas. Por fim, identificamos que o desempenho das duas plataformas (CUDA e OpenCL) são similares. O OpenCL se sobressaiu apenas nos algoritmos que demandam maior volume de cômputo, como multiplicação de matrizes e os possíveis motivos para tal resultado são as formas de implementação dos algoritmos e/ou a compatibilidade entre a versão do *benchmark* e as versões CUDA e OpenCL.

Tendo em vista todos os dados coletados durante a pesquisa podemos notar que o OpenCL é a opção mais factível, pois mesmo com menos ferramentas de criação ela atende as necessidades do laboratório permitindo a criação de códigos para diversas placas gráficas independente do fabricante.

REFERÊNCIAS

CANALTECH. O que é benchmark? 2014. Acesso em: 01 de Mar. De 2018. Disponível em: https://canaltech.com.br/produtos/O-que-e-benchmark--26350/.

DE PAULA, Lauro C. M. 2014. CUDA vs. OpenCL: uma comparação teórica e tecnológica. ForScience, v. 2, n. 1, p. 31-46, 2014. Acesso em: Março de 2017. Disponível em: http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/Cuda_x_OpenCL.pdf

KARIMI, K.; DICKSON, N.; HAMZE, F. 2010. A performance comparison of CUDA and OpenCL. arXiv preprint arXiv:1005.2581.

ROCHA, Ricardo. 2007/2008. Programação Paralela e Distribuída. Acesso: 17 de Mar. de 2017 Disponível em: https://www.dcc.fc.up.pt/~ricroc/aulas/0708/ppd/apontamentos/fundamentos.pdf.

TSUCHIYAMA, R. 2010. The OpenCL Programming Book. [S.l.]: Fixstars.

VIENNACL. About ViennaCL. 2017. Acesso em: 01 de Março de 2018. Disponível em: http://viennacl.sourceforge.net/viennacl-about.html.