OpenCL: uma Alternativa para o Ensino de Arquitetura de Computadores

Fábio D. Pereira, Allan M. de Souza, #Edward David Moreno
COMPSI – Lab. Pesq. Computação e S. Informação
Centro Universitário Eurípides de Marília – UNIVEM, Marilia, Brasil

#DCOMP/UFS - Departamento de Computação da Universidade Federal de Sergipe, Aracaju, Brasil

{dacencio, allan-ms} @univem.edu.br, edwdavid@gmail.com

Resumo— O ensino de arquitetura de computadores, tanto conceitualmente e na sua prática, é adequado para novas arquiteturas multi/many cores. Este artigo destaca a aplicação da plataforma OpenCL no ensino prático de arquitetura de computadores considerando o uso de CPU e GPU. O artigo também demonstra exemplo de aplicações e comparação de desempenho entre CPU e GPU.

Palavras-chave: OpenCL, computação de alto desempenho, paralelismos de instruções e dados .

I. INTRODUÇÃO

O ensino teórico de arquitetura de computadores está presente em um conjunto comum de livros fortemente referenciados como bibliografia básica de disciplinas da graduação em Ciência da Computação e Engenharia.

A metodologia clássica de apresentação dos conceitos e fundamentos sobre arquitetura de computadores é semelhante entre os cursos citados. Apesar de não podermos apresentar dados quantitativos, pode-se constatar em uma simples pesquisa na internet essa semelhança.

No entanto, quando esse conteúdo é explorado na prática as técnicas e ferramentas são distintas para cada universidade, com influência direta da formação do corpo docente, laboratórios de aula e pesquisa disponíveis na instituição de ensino.

Neste artigo é apresentada a plataforma OpenCL e sua utilização para o ensino de arquitetura de computadores. Um estudo de caso é apresentado e conclui que o uso do OpenCL auxilia na compreensão prática dos conceitos e fundamentos de arquitetura de computadores, assim como, o relacionamento outras disciplinas: programação de computadores, sistemas operacionais, sistemas paralelos e sistemas distribuídos.

Alguns artigos como "Impactos, Oportunidades e Desafios no contexto da Educação em Arquitetura de Computadores causados pelos Processadores com Múltiplos Núcleos" [3], "Ferramenta para Simulação de Multiprocessadores Superescalares de Memória Compartilhada" [4] e "Novas Perspectivas no Contexto da Arquitetura de Computadores – Softwares Educacionais para o Paradigma da Computação Paralela" [5] publicados no Workshop sobre Educação em Arquitetura de Computadores (WEAC) de anos anteriores apresentam estudos correlatos. No entanto, os artigos são direcionados apenas para o ensino de arquitetura avançada de computadores. Diferente da abordagem proposta neste artigo, onde ressalta-se o uso de OpenCL para o ensino de conceitos básicos de arquitetura de computadores até tópicos avançados, em um mesmo ambiente de desenvolvimento.

II. PLATAFORMA OPENCL

OpenCL é uma plataforma para a programação de arquiteturas heterogêneas, que podem ser formadas por coleções de CPUs, GPUs e outros dispositivos computacionais organizados em uma única arquitetura [1].

Programas únicos escritos em OpenCL podem ser executados em uma ampla gama de sistemas, a partir de telefones celulares, computadores portáteis até nós de supercomputadores. Nenhum outro padrão de programação paralela tem essa capacidade de portabilidade [2].

A arquitetura básica do OpenCL pode ser descrita usando modelos hierárquicos. Modelo de plataforma (A), modelo de execução (B), modelo de memória (C) e modelo de programação (D).

A. Modelo de plataforma

O modelo de plataforma consiste em um *host* que está conectado a um ou mais dispositivos OpenCL (CPUs, GPUs, PDAs), os dispositivos OpenCL são divididos em uma ou mais unidades de computação (UCs), que são divididos em um ou mais elementos de processamento (PEs). Os cálculos executados nos dispositivos OpenCL ocorrem dentro dos elementos de processamento [1].

A figura 1 ilustra o modelo de plataforma OpenCL que foi descrito.

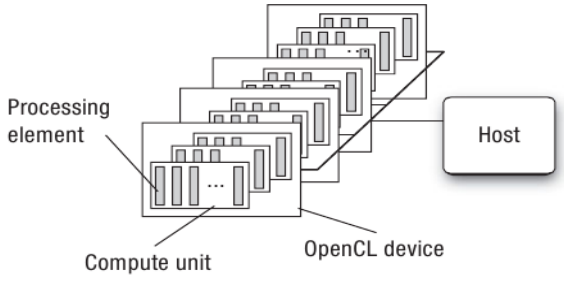


Figura 1: OpenCL Modelo de plataforma [2].

B. Modelo de execução

A execução de um programa OpenCL ocorre em duas partes: (i) *kernels*, que são funções executadas em um ou mais dispositivos OpenCL e (ii) programa de *host*, que

executa partes seuquenciais do programa. O programa de *host* define o contexto e os parâmetros para os *kernels* e também gerencia a sua execução [1].

O modelo de execução OpenCL é definido pela forma como seus *kernels* são executados. Quando o programa de *host* apresenta um *kernel* para a execução, um espaço de índices é definido chamado *NDRange*, onde estes índices podem ser de uma dimensão (1D), duas dimensões (2D) ou três dimensões (3D). Cada ponto no espaço de índice é chamado de *work-item* e cada *work-item* é uma instância do *kernel* e possuem um índice único (identificador global) para calcular endereços de memória e de tomar decisões de controle.

Os *work-items* são organizados em *work-groups*, fornecem uma decomposição do espaço índice. Aos *work-groups* é atribuído a um identificador de grupo único, com a mesma dimensão do espaço de índice utilizado para os *work-items*, é atribuído também, um identificador único local dentro de cada *work-group* para que cada *work-item* possa ser identificado exclusivamente por seu identificador global ou por uma combinação de seu identificador local e grupo. *Work-items* em um determinado *work-group* executam concorrentemente sobre os elementos de processamento de uma única unidade de computação [2].

A figura 2 é um exemplo da forma como os identificadores globais, locais e de grupos estão relacionados por um *NDRange* bidimensional. Outros parâmetros do espaço de índice está definido na figura. O bloco sombreado tem uma identificação global de (GX, GY) = (6, 5), com um identificação de grupo mais uma identificação local (WX, WY) = (1, 1) e (lx, ly) = (2, 1).

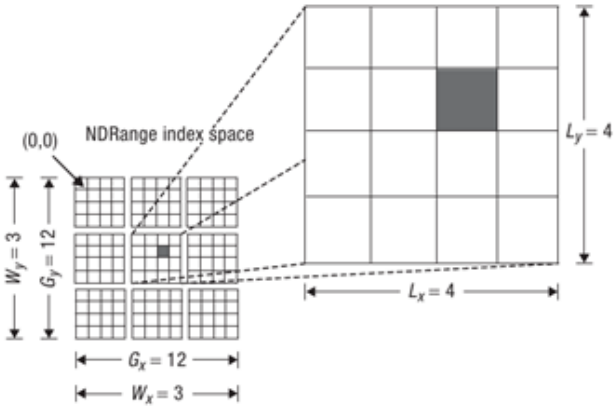


Figura 2: Modelo de excução OpenCL[2].

C. Modelo de memória

Work-items executando um *kernel* têm acesso a cinco regiões de memória distintas [2].

- **Memória de host:** Esta região de memória só é visível para o *host*. Tal como acontece com a maioria dos detalhes sobre o *host*, OpenCL define apenas como a memória do *host* interage com objetos OpenCL.
- **Memória global:** Esta região de memória permite acesso de leitura e escrita a todos os *work-items* em todos os *work-groups*. Cada *work-item* pode ler ou escrever em qualquer elemento de um objeto de memória. Leitura e escrita para a memória global pode ser armazenada em cache, dependendo das capacidades do dispositivo.
- **Memória constante:** Esta região de memória global permanece constante durante a execução de um *kernel*. O *host* aloca e inicializa objetos de memória colocado na memória constante. *Work-items* têm acesso somente leitura a esses objetos.
- **Memória local:** Esta região de memória é local para cada *work-item*. Esta região de memória pode ser usada para alocar variáveis que são compartilhadas por todos os *work-items* do mesmo *work-group*. Ele pode ser implementado como regiões dedicadas de memória no dispositivo OpenCL. Alternativamente, a região de memória local pode ser mapeado sobre seções da memória global.
- **Memória privada:** Esta região de memória é privada a cada *work-item*. As variáveis definidas na memória particular de um *work-items* não são visíveis para os outros *work-items*.

D. Modelo de programação

OpenCL inclui uma linguagem baseada em C99 para escrever o código do *kernel*, e o programa de *host* pode ser escrito em outras linguagens, tais como: C/C++, Java e Python. O modelo de programação OpenCL suporta paralelismo de dados e processos, bem como implementações híbridas destes dois modelos.

III. ESTUDO DE CASO DE ENSINO NA GRADUAÇÃO

Nesta seção apresenta-se um estudo de caso do uso de plataforma OpenCL em um curso de Bacharelado em Ciência da Computação. Para sua aplicabilidade e eficiência, o estudante deve ter uma base de paradigmas e linguagens de programação e conceitos básicos de organização e arquitetura de computadores. O ensino pode ser explorado em quatro etapas, descritas e comentadas a seguir:

(i) Conceitos de arquiteturas paralelas: opcionalmente, podem ser apresentados ou revisados os conceitos arquiteturais de SMP, CMP, SIMD, MIMD, NUMA, bem como as principais premissas de software envolvidas na programação paralela, tais como o paralelismo no nível de instruções, dependência de dados, recursos compartilhados, uso de recursos compartilhados, comunicação e sincronização entre processos, entre outros conceitos.

(ii) Ensino da plataforma OpenCL: etapa conceitual e prática para demonstrar o processo de desenvolvimento em OpenCL, detalhando as técnicas de programação e compilação para esta plataforma. Os conceitos assimilados na etapa (i) são justificados nesta etapa e os estudantes podem ver na prática o impacto da sua aplicação.

(iii) Paralelização de problemas clássicos: de maneira evolutiva e didática apresentar técnicas para explorar soluções para computação de alto desempenho. Partindo de problemas simples de paralelização como uma operação de multiplicação de matrizes até otimizações para algoritmos de processamento de imagem. Na seção IV é

apresentado resultados alcançados por alunos de graduação na disciplina de organização e arquitetura de computadores II.

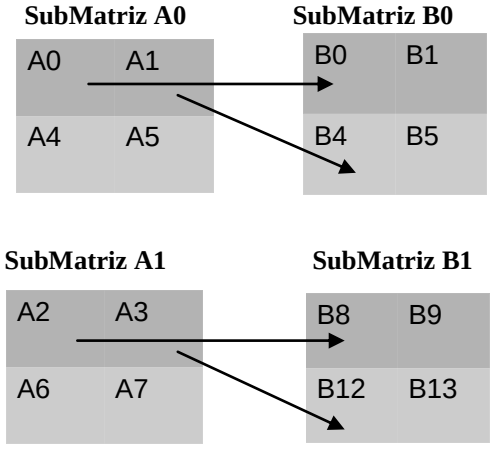
(iv) Paralelização de algoritmos especiais: são considerados algoritmos especiais novas soluções para áreas especificas. Como exemplo, este ano letivo foi trabalhado novo padrão de algoritmo para função de *hash* SHA-3 (Keccak) [9]. Na seção IV apresenta-se resultados obtidos em OpenCL com execução em GPUs. Os resultados obtidos de algoritmos especiais pode gerar publicações em eventos científicos e técnicos na área.

IV. EXEMPLOS DE ALGORITMOS DESENVOLVIDOS E RESULTADOS

Nesta seção são apresentados dois algoritmos desenvolvidos e a comparação de desempenho entre CPU e GPU das implementações propostas. O primeiro algoritmo apresentado é uma paralelização de uma simples multiplicação de matrizes e o segundo algoritmo apresentado é a paralelização de dados do novo padrão de algoritmo para funções *hash* SHA-3 (Keccak), definido em outubro de 2012 pelo *National Institute of Standards and Technology* (NIST).

Para a paralelização da multiplicação de matrizes A x B = C, cada matriz é dividida em sub-matrizes, ou seja, em uma multiplicação de A[4][4] x B[4][4], gera-se:
$subMatrizA_0[2][2] = \{\{A_0, A_1\},\{A_4, A_5\}\}$,
$subMatrizA_1[2][2] = \{\{A_2, A_3\},\{A_6, A_7\}\}$,
$subMatrizA_2[2][2] = \{\{A_8, A_9\},\{A_{12}, A_{13}\}\}$,
$subMatrizA_3[2][2] = \{\{A_{10}, A_{11}\},\{A_{14}, A_{15}\}\}$
e
$subMatrizB_0[2][2] = \{\{B_0, B_1\},\{B_4, B_5\}\}$,
$subMatrizB_1[2][2] = \{\{B_8, B_9\},\{B_{12}, B_{13}\}\}$,
$subMatrizB_2[2][2] = \{\{B_2, B_3\},\{B_6, B_7\}\}$,
$subMatrizB_3[2][2] = \{\{B_{10}, B_{11}\},\{B_{14}, B_{15}\}\}$.

Desse modo, é criado um *NDRange* de duas dimensões de tamanho 4x4, também é criado *work-groups* de dimensões iguais ao *NDRange* de tamanho 2x2, onde, cada *work-item* instanciado em cada *work-group* calcula as multiplicações referentes ao seus identificadores. A figura 4 demonstra um exemplo do cálculo do primeiro elemento da matriz resultante A x B = C, calculado pelas sub-matrizes geradas, como descrito nesta seção.



Tem-se,
C [0] = SubMatrizA0[0] * SubMatrizB0[0] + SubMatrizA0[1] * SubMatrizB0[2] + subMatrizA1[0] * SubMatrizB1[0] + SubMatrizA1[1] * SubMatrizB1[2];

Figura 3: Exemplo de cálculo do primeiro elemento da matriz resultante com sub matrizes com *work-groups* de dimensão 2x2.

Nesse escopo, foi implementado um kernel OpenCL de uma multiplicação de matrizes de 4800 x 4800 elementos com work-groups de duas dimensões de tamanho 16 x 16. O programa de host foi implementado, nas três linguagens suportadas pela plataforma (C/C++, Java e Python) para comparação de desempenho entre as mesmas. A tabela I apresenta os resultados obtidos dos testes de desempenhos em segundos, executados em uma CPU Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz e em uma GPU AMD/ATI Radeon HD 6400M Grafics Series.

TABELA I. RESULTADOS DA IMPLEMENTAÇÃO DA MULTIPLICAÇÃO DE MATRIZES

Implementação	**Tempo em segundos**	
	CPU	**GPU**
Java + OpenCL	859,937	14,397
Python + OpenCL	841,851	14,257
C/C++ + OpenCL	868,585	14,139

Os resultados obtidos mostram um aumento de desempenho de aproximadamente 60 vezes entre GPU e CPU, os resultados também mostram que o tempo de execução das implementações são próximos, indicando assim, que o programa de *host* não influência no desempenho da aplicação, ou seja, o desempenho da aplicação depende do dispositivo no qual está sendo executado o *kernel* OpenCL.

Para a implementação do algoritmo de funções *hash* SHA-3 (Keccak), a proposta foi utilizar a paralelização de dados (SIMD – *Single instruction Multiple Data*), de modo que, cada instância execute paralelamente o *core* do algoritmo (keccak-f).

A principal função executada pelo algoritmo, é a keccak-f [b], onde b {25, 50, 100, 200, 400, 800, 1600} é a largura de a permutação. E recebe como entrada um *state* que é uma matriz 5x5, com comprimento w {1, 2, 4, 8, 16, 32, 64} (b = 25 W), onde aplicam-se os quatro passos do algoritmo (Θ, $\rho\pi$, χ, ι) e retorna uma matriz de 5x5 com comprimento w como saída. A estrutura original do Keccak foi quase totalmente mantida nesta solução, mesmo que tenham sido feitos alguns ajustes para maximizar o desempenho em GPU. Sendo assim para implementar a paralelização de dados,

Na função OpenCL, é recebido como entrada um vetor de X elementos, sendo X multiplo de 25, onde será instânciada em um *NDRange* de tamanho X/25 *work-items*, cada *work-item* receberá como entrada espaços diferentes do vetor de entrada de modo que cada entrada recebida forme uma matriz 5x5 ou um vetor de 25 posições, formando assim diferentes *states* para aplicar a função principal do algoritmo.

A figura 4 mostra o modelo de distribuição dados e execução do *kernel* OpenCL conforme foi descrito acima. De modo que recebe como entrada um vetor de tamanho x onde x é múltiplo de 25 e é instanciada um *NDRange* com x/25 *work-item*. Cada *work-item* instância um *kernel* OpenCL, copia um espaço do vetor de entrada para sua memória local formando um *state* e, aplica a permutação keccak-f nesse *state*, onde depois é copiado para o vetor de saída.

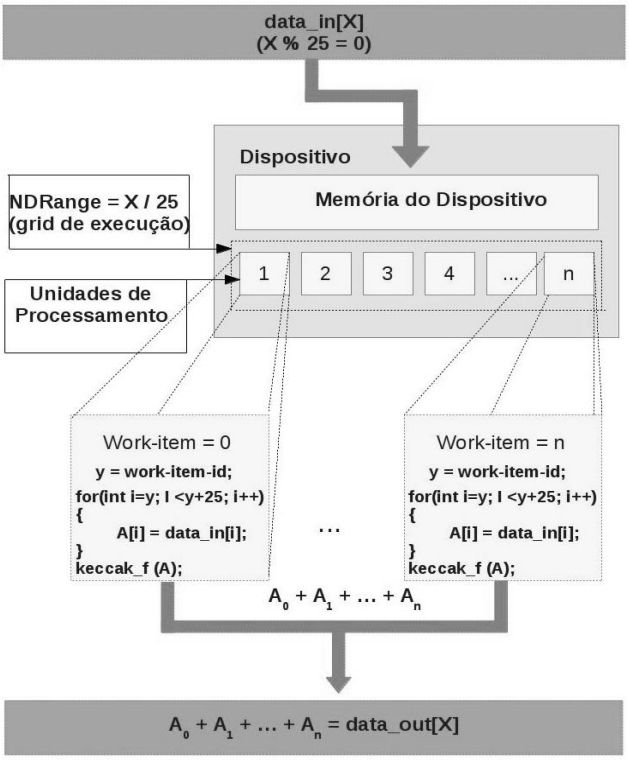


Figura 4: Modelo de distribuição dados e execução do *kernel* OpenCL implementado

Com esse cenário, foram realizados testes de desempenho com diferentes números de instâncias do kernel OpenCL, onde os resultados foram comparados entre CPU e GPU. Tabela II mostra o número de work-items e o tempo que todos levaram para executar o algoritmo.

TABELA II. RESULTADOS DA IMPLEMENTAÇÃO KECCAK SHA-3 EM OPENCL

No. work-items	**Tempo em segundos**	
	CPU Intel core I5	GPU AMD Radeon HD 6400M
$5*10^5$	0,63937	0,06292
$1*10^6$	1,29261	0,12376
$5*10^6$	62,93171	6,18758
$1*10^8$	125,82469	12,03659
$5*10^8$	628,15016	60,47335
$1*10^9$	1258,75649	119,85742

Os resultados desta primeira implementação mostram que a execução da GPU é aproximadamente 10 vezes mais rápida do que a execução da CPU. Porém esses dados podem ser melhorados gerenciando as memórias locais e privadas dos dispositivos e *work-items* e também dimensionando diferentes *work-groups*.

Diferentes testes já foram executados nessa arquitetura, como explorar a paralelização do algoritmo, de forma que múltiplos *work-items* instanciados em um único *work-group* executem sub-funções do algoritmo. Uma paralelização também já implementada é a paralelização do algoritmo juntamente com a paralelização dos dados executados, onde, é instanciado múltiplos *work-items* em múltiplos *work-groups*, de modo que, cada *work-group* calcule diferentes *hash*.

Outros testes também podem ser executados, como implementação do algoritmo em pipeline, consumo de energia dos dispositivos, gerenciamento do acesso as memórias locais e privadas dos *work-groups* e *work-items*, e também diferentes arquiteturas comparadas, como executar esses mesmos testes na plataforma CUDA. Importante salientar que os resultados parciais dos testes descritos acima já foram publicados em alguns eventos na área [6] [7] [8].

V. CONCLUSÃO

Este artigo destaca a aplicação da plataforma OpenCL no ensino prático de arquitetura de computadores considerando o uso de CPU e GPUs. Também foram demonstrados exemplos de aplicações, sendo um mais simples, a paralelização de uma operação de multiplicação de matrizes, outro mais robusto, a implementação da função de *hash* SHA-3. A possibilidade de gerar artigos a partir de resultados alcançados na disciplina de arquitetura de computadores motivou docentes e discentes do curso na escrita deste artigo.

REFERÊNCIAS

[1] Khronos OpenCL Working Group, The OpenCL Specification, 2011.

[2] Aaftab Munshi, et al. OpenCL Programming Guide, 2011.

[3] Silva, L. G., Souza M. O. S., Impactos, Oportunidades e Desafios no contexto da Educação em Arquitetura de Computadores causados pelos Processadores com Múltiplos Núcleos, Workshop sobre Educação em Arquitetura de Computadores - WEAC 2009.

[4] Perissatto, M. G., et al, Ferramenta para Simulaçãoo de Multiprocessadores Superescalares de Memória Compartilhada, Workshop sobre Educação em Arquitetura de Computadores- WEAC, 2007.

[5] Diego Marinho de Oliveira1, Fernando Augusto Fernandes Braz2, Tiago Figueiredo de Carvalho Novas Perspectivas no Contexto da Arquitetura de Computadores – Softwares Educacionais para o Paradigma da Computacão Paralela, Workshop sobre Educação em Arquitetura de Computadores- WEAC, 2009

[6] Pereira, F. D. ; Ordonez, E. D. M.; Sakai, I. D. Hash function keccak: exploring parallelism with pipeline. In: PDCS- Parallel and Distributed Computing and Systems, 2011.

[7] PEREIRA, Fábio Dacêncio; ORDONEZ, Edward David Moreno; SOUZA, A. M. . Exploiting Parallelism on Keccak: FPGA and GPU Comparison. Parallel & Cloud Computing, v. 2, p. 1, 2013

[8] PEREIRA, Fábio Dacêncio ; ORDONEZ, Edward David Moreno; SOUZA, A. M. Exploiting Heterogeneous Systems: Keccak on OpenCL. PDPTA'13 - Int'l Conf on Parallel & Distributed Processing Techniques & Applications, July 22-25, 2013, Las Vegas, USA.

[9] FIPS 180-3, Secure Hash Standard, Cryptographic Hash Algorithm Competition, , available from http://csrc.nist.gov/groups/ST/hash/sha-3/index.html, 2013