Abordagem para Ensino de Programação Paralela em Ambientes Heterogêneos Usando OpenCL

Lucas Henrique Silva Valentim, Henrique Cota de Freitas
Departamento de Ciência da Computação
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Belo Horizonte, Brasil
lucas.h.valentim@gmail.com, cota@pucminas.br

***Resumo*—Este artigo apresenta uma abordagem para ensino de programação paralela em ambientes heterogêneos a ser utilizada nos cursos de graduação de computação. O principal objetivo dessa abordagem é possibilitar uma transição suave entre o paradigma de programação sequencial para a programação paralela em ambientes heterogêneos, capacitando os alunos de graduação a extrair melhor desempenho das arquiteturas atuais. A linguagem de programação utilizada é OpenCL, devido a sua alta portabilidade, ser um padrão de linguagem livre de programação e por ser uma linguagem de programação paralela que possibilita utilizar todos os recursos de uma arquitetura heterogênea. A abordagem de ensino é divida em dois componentes, onde o componente introdutório aborda as principais características da linguagem de programação OpenCL, a identificação de hardware com suporte a OpenCL em um ambiente heterogêneo e a configuração do ambiente de desenvolvimento. O Componente de transição é composto por aplicações com nível de complexidade crescente, afim de possibilitar o ensino prático da programação paralela em ambientes heterogêneos.**

Palavras-chave— OpenCL, programação paralela, arquiteturas paralelas, computação heterogênea, ensino

I. INTRODUÇÃO

Grande parte dos estudos científicos em arquiteturas na década de 1980, até meados da década de 1990, estavam focados em viabilizar o aumento de frequência dos processadores. Esse período ficou conhecido como a corrida pelo aumento da frequência. A estratégia de aumento da frequência não vem sendo mais utilizada nos moldes anteriores devido a limitações tais como: aumento do consumo de energia para que haja ganho de desempenho, dificuldade para conter dissipação de calor e pelo hiato criado entre a velocidade em que o processador processa as informações e a velocidade em que a memória consegue entregar essas informações ao processador [3, 4]. Essa diferença de desempenho é conhecida como Gargalo de von Neumann [5].

Para continuar atendendo a demanda por desempenho, que é crescente e contínua, contornando as limitações encontradas pelo grande aumento de frequência dos processadores, surge a estratégia de fabricar *chips* com mais de um processador. Essa estratégia tem como resultado os processadores de propósito geral *multi-core* e consequentemente os *many-core*. Os computadores atualmente são compostos por uma arquitetura heterogênea contendo tanto processador de propósito específico, e.g., núcleos de processamento de imagens, quanto de propósito geral, o que torna possível extrair de cada aplicação o melhor desempenho.

Essa classificação abrange quatro diferentes classes. Para as arquiteturas sequenciais onde um único fluxo de instruções é executado sobre um único fluxo de dados, a classe é *Single Instruction Single Data* (SISD). Para classificar os processadores vetoriais, onde um mesmo fluxo de instruções é executado de forma paralela sobre vários dados, a classe é *Single Instruction Multiple Data* (SIMD). Os processadores com múltiplos fluxos de instruções sobre um mesmo fluxo de dados, a classe é *Multiple Instruction Single Data* (MISD) e para os processadores onde múltiplos fluxos de instruções são executados sobre múltiplos fluxos de dados, a classe é chamada de *Multiple Instruction Multiple Data* (MIMD) [19]. Das quatro classes disponibilizadas por Flynn, somente três são encontradas nos processadores atuais, haja vista que a classe MISD não é funcional para a demanda das aplicações existentes. Neste trabalho, a arquitetura heterogênea utilizada para simulação dos resultados é composta por processadores que podem ser classificados como MIMD e SIMD.

A mudança do cenário onde os processadores eram sequenciais e sua arquitetura homogênea para computadores paralelos e compostos por processadores de propósito específico e geral, modifica o ensino de programação e de arquitetura de computadores. No cenário atual, para extrair o desempenho máximo das arquiteturas é necessário saber programar de forma paralela [16] e ter conhecimento sobre as diferentes propostas de arquiteturas e suas características [6, 9]. Sendo assim, este trabalho tem como objetivo discutir uma abordagem desenvolvida na linguagem OpenCL, assim como apresentar um conjunto de aplicações para iniciar o ensino de programação paralela heterogênea nos cursos de computação.

O ensino de programação sequencial é ainda muito forte nos cursos de computação. Quebrar esse paradigma no cenário atual, onde a tendência de arquiteturas paralelas e heterogêneas vem se confirmando, é necessário, uma vez que a programação sequencial desperdiça um enorme potencial de hardware [2, 6]. Alguns cursos de graduação possuem ainda uma programação voltada para o modelo sequencial onde o ensino de programação paralela e também heterogênea se faz necessário visto a evolução das arquiteturas de computadores, inclusive pessoais [8]. Uma característica que dificulta o ensino de programação paralela em ambientes heterogêneos é que além dos conhecimentos necessários intrínsecos a programação, existe também a necessidade de compreensão de conceitos

de arquitetura como: distribuição de cargas de trabalho, análise de comportamento da aplicação em diferentes arquiteturas, utilização eficiente dos núcleos de processamento, atenção às dependências de dados [16, 17], entre outras diversas características que influenciam no desempenho e bom funcionamento da aplicação. Neste cenário podemos concluir que é necessário que sejam desenvolvidas pesquisas que disponibilizem técnicas e ferramentas que cubram todas as especificidades deste tipo de programação.

As próximas seções deste artigo estão organizadas da seguinte maneira: na Seção II será exibida uma visão geral da linguagem OpenCL, na Seção III serão discutidos os trabalhos relacionados, na Seção IV serão identificados os desafios no ensino da programação paralela em ambientes heterogêneos, a Seção V demonstra a abordagem proposta neste artigo, e na Seção VI são feitas análises de desempenho acerca de algumas das aplicações usadas na abordagem. Por fim, na Seção VII é apresentada a conclusão e os trabalhos futuros.

II. VISÃO GERAL DE OPENCL

A linguagem de programação OpenCL disponibilizada pelo consórcio Khronos Group, surge com o propósito de unificar o cenário de programação paralela e heterogênea, por meio de uma linguagem livre, tendo como objetivo o aumento de produtividade dos programadores, evitando que os mesmos tenham que se adaptar aos diferentes fabricantes de hardware [8]. Para possibilitar a portabilidade do código OpenCL entre os diferentes fabricantes de hardware, que fazem parte do consórcio Khronos Group, entre outros requisitos que viabilizam o bom funcionamento do código, é necessário que estes implementem o suporte a abstração arquitetural exigida pela linguagem [8].

Um dos propósitos do conteúdo dessa seção é introduzir as principais características da linguagem OpenCL de forma reduzida e simplificada, o que não substitui a necessidade de leitura da especificação OpenCL. Esta seção se restringe a explicar as principais características da versão OpenCL 1.2, pois nem todos os fabricantes suportam as versões superiores. Vale ressaltar que restringir à explicação a versão 1.2 não torna o documento desatualizado, pois as características exemplificadas aqui se mantêm na versão 2.0.

Essa explicação é dividida da seguinte forma: Na primeira subseção é apresentada uma visão macro acerca dos elementos que compõem o desenvolvimento a fim de tornar claro o cenário de programação em OpenCL. Na segunda subseção é apresentada uma representação gráfica da abstração arquitetural que não necessariamente é fiel a especificação em sua totalidade, pois o principal objetivo é destacar as principais características dessa abstração, que tem forte influência no desenvolvimento das aplicações. Sendo assim, não será dada ênfase à explicação dos detalhes arquiteturais. Na terceira subseção é apresentada uma explicação a cerca do mapeamento do processamento paralelo para os elementos de processamento (PE).

A. Primeira parte

Nesta subseção é apresentada uma visão macro do cenário, onde os principais componentes OpenCL são explicados. O entendimento desta etapa é de grande importância para a continuidade do aprendizado. A Figura 1 ilustra os principais componentes.

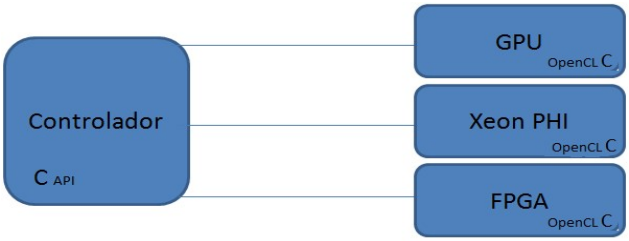

FIGURA 1 – PRINCIPAIS COMPONENTES OPENCL

1) ***Visão Geral****:* Como ilustrado na Figura 1, a proposta da programação em OpenCL é extrair de maneira eficiente desempenho de uma arquitetura heterogênea. Isso é possível, pois diferentes hardwares como CPU, GPU, Xeon PHI e FPGA são programáveis através da linguagem de programação OpenCL C e possibilitam que sejam coordenados pelo controlador através de uma API [7]. Portanto, é possível que trechos dos códigos sejam executados em hardwares diferentes. Como exemplo um trecho com baixo potencial paralelo é executado no próprio Host sequencialmente e outro trecho com alto potencial paralelo é escrito em OpenCL C e enviado para ser executado na GPU (*Graphics Processing Unit*) possibilitando, assim, extrair as qualidades de diferentes propostas arquiteturais.

A programação em OpenCL é composta por duas linguagens de programação semelhantes, são elas: API Controladora que é desenvolvida em C e a linguagem de programação OpenCL C que herdou grande parte das características da linguagem C99 porém, com restrições tais como não permitir o uso de recursividade [7].

A API controladora é utilizada para coordenar o funcionamento de todos os dispositivos. Sendo assim, o programador utiliza a API para informar aos dispositivos o que eles devem fazer, por exemplo: quantas vezes devem repetir a execução de um trecho de código, quais regiões de memória estão disponíveis para os dispositivos, se um comando enviado deve esperar pela finalização de outro, entre outras diversas formas disponíveis para realizar a coordenação dos dispositivos.

A linguagem de programação OpenCL C é a linguagem utilizada para escrever o código que será executado pelos dispositivos. Essa linguagem utiliza o marcador Kernel para identificar as funções que serão executadas pelos nós de processamento dos dispositivos. Um Kernel é a menor instância de execução em um processo paralelo e, para se atingir o resultado final, na maioria das vezes, uma mesma função Kernel é chamada repetidas vezes de forma similar a uma estrutura de repetição *for* comumente utilizada nas linguagens sequenciais.

B. Segunda parte

Após entender que a programação em OpenCL é uma forma de se extrair desempenho através da paralelização dos códigos, e da utilização eficiente das diferentes características de um ambiente heterogêneo, é necessário aprender como OpenCL viabiliza a paralelização desses

códigos e como ele possibilita que se gerencie os dispositivos de diferentes fabricantes, afim de extrair deles suas melhores características. Por fim, torna-se necessário realizar ambas as tarefas mencionadas para evitar o enorme desperdício de potencial de hardware como atualmente grande parte das aplicações o fazem.

Sendo assim, nesta subseção será apresentada de forma resumida alguns elementos que possibilitam e influenciam a programação utilizando a linguagem OpenCL. O foco não será na sintaxe de criação de código e sim, nos conceitos arquiteturais que influenciam na portabilidade e bom funcionamento das aplicações desenvolvidas nesse cenário.

1) ***Dispositivo:*** A abstração OpenCL, citada anteriormente e ilustrada na Figura 2, está fortemente relacionada com os dispositivos, pois são eles que implementam essa abstração tornando possível que os programadores se preocupem com um único modelo arquitetural sem se preocupar com as características arquiteturais de cada fabricante.

a) ***Elemento de processamento (PE)****:* Os elementos de processamento (PE) são os responsáveis por executar todas as linhas de código desenvolvidas em OpenCL C, ou seja, são eles os responsáveis por executar todas as instruções [7]. Se há 128 elementos de processamento na sua placa de vídeo aceleradora, então, é possível executar 128 *threads* em paralelo. Outra opção é dividir o trabalho entre as unidades de computação (CU). Sendo assim, os elementos de processamento trabalham em conjunto ao processar uma *thread* e compartilham informação entre si. A escolha acerca da forma de execução é altamente dependente das características da aplicação.

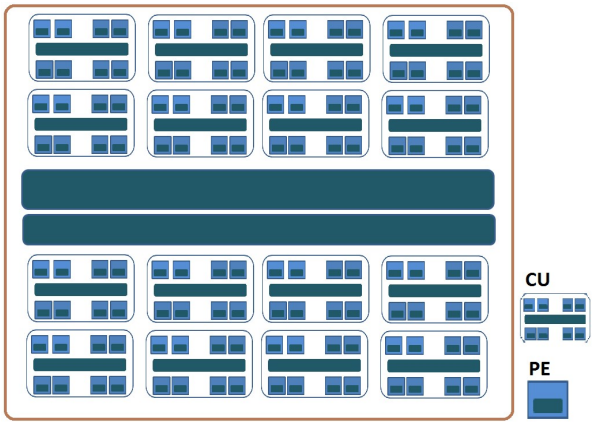

FIGURA 2 – ABSTRAÇÃO OPENCL

b) ***Unidade de Computação (CU)****:* A unidade de computação é um aglomerado de elementos de processamento (PE) [7]. A unidade de computação em si não executa nenhuma instrução. O seu papel é criar um conjunto que pode dividir a carga de trabalho, se assim desejar o programador, e se comunicar através de uma memória compartilhada entre todos os PE's de um mesmo CU.

2) ***Memória:*** A abstração OpenCL especifica quatro diferentes regiões de memória, a saber: memória global, privada, constante e local [7], que podem ser visualizadas na Figura 3. A utilização correta das memórias resulta em aumento de desempenho, pois pode melhorar a comunicação e diminuir a latência.

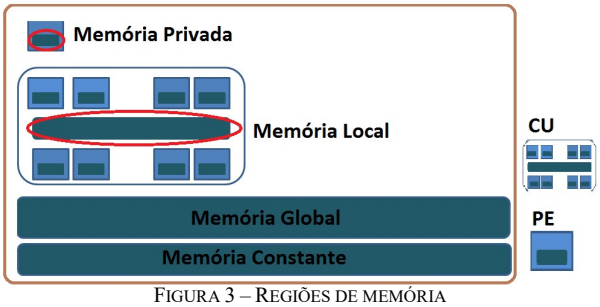

FIGURA 3 – REGIÕES DE MEMÓRIA

a) ***Memória Privada:*** cada elemento de processamento (PE) tem sua própria memória privada e nenhum outro elemento de processamento ou mesmo o controlador tem acesso a essa região de memória.

b) ***Memória Local:*** Toda unidade de computação (CU) tem sua memória local. Essa memória é compartilhada exclusivamente entre os elementos de processamento (PE) que estão contidos nessa CU.

c) ***Memória Global:*** A memória Global é acessada tanto pelo controlador, quanto por todos os elementos de processamento (PE). Existe a diferença de persistência dos dados entre a memória Global e as demais.

d) ***Memória Constante:*** A memória Constante é acessível a todos os elementos de processamento (PE) porém somente para leitura.

C. Terceira parte

Essa subseção apresenta os conceitos necessários para o entendimento da estrutura de repetição paralela utilizada na linguagem OpenCL, e como balancear sua carga entre os diversos núcleos de forma a evitar que núcleos fiquem ociosos.

1) ***ND-****Range:* Para quebrar o paradigma sequencial e iniciar a programação paralela, a linguagem OpenCL utiliza o conceito ND-Range. Este conceito consiste em disponibilizar um conjunto de índices onde cada índice representa uma instância da função Kernel, que deve ser executada. A melhor forma para entender este conceito é pensar que o conjunto de índices é uma estrutura de repetição *for*, onde cada índice é uma iteração e a quantidade de índices gerados é igual ao limite de repetições definidos no *for*, este conjunto de índices é denomidado como *WorkSize* (WS). Lembrando que essa comparação é somente uma técnica para facilitar o entendimento, pois os índices diferentemente do *for* são executados de forma paralela, não precisando que a iteração anterior tenha acontecido para que a próxima aconteça. Para facilitar o entendimento a Figura 4 ilustra essa comparação.

C	OpenCL C
For(int i = 0; i < 3; i ++) { }	Índices gerados: 0,1 e 2

FIGURA 4 – COMPARAÇÃO FOR / ND-RANGE

2) ***Mapeamento ND-Range:*** Para facilitar o entendimento do mapeamento dos Kernels aos elementos de processamento (PE) imagine que uma placa de vídeo que tem suporte a OpenCL, contenha uma única unidade de computação (CU) e somente quatro elementos de processamento (PE). A aplicação necessita que quatro posições de um vetor sejam multiplicadas por uma constante de valor 2. A Figura 5 ilustra o mapeamento da função Kernel Y * 2 e quais as posições do vetor cada elemento de processamento irá multiplicar. O módulo de transição contém esse código implementado e pode ser utilizado como ponto de partida para iniciar a programação em OpenCL.

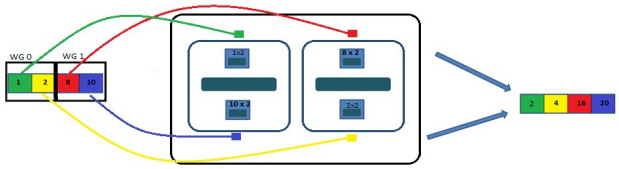

FIGURA 5 – MAPEAMENTO DA FUNÇÃO KERNEL Y * 2

3) ***WorkGroup:*** Como explicado anteriormente, a paralelização em OpenCL utiliza o conceito ND-Range. Este conceito possibilita a paralelização através de índices. Quem determina a quantidade de índices é o tamanho do trabalho (WS). O WS deve ser informado pelo programador ao escrever o código através da API C. Os índices gerados devem ser divididos e distribuídos nas unidades de computação (CU's) para serem executados pelos elementos de processamento (PE's). É recomendado que o próprio programador realize essa divisão pois, assim, ele garante que nenhuma CU ficará ociosa. A divisão do trabalho é realizada criando grupos de trabalho (WG). A Figura 6 ilustra a divisão para fins didáticos, deixando núcleos ociosos propositalmente afim de demonstrar possíveis desperdícios de potencial de hardware. O tamanho definido para o WG tem impacto no desempenho final do trecho paralelo, pois tem influência direta na utilização dos elementos de processamento e das regiões de memória.

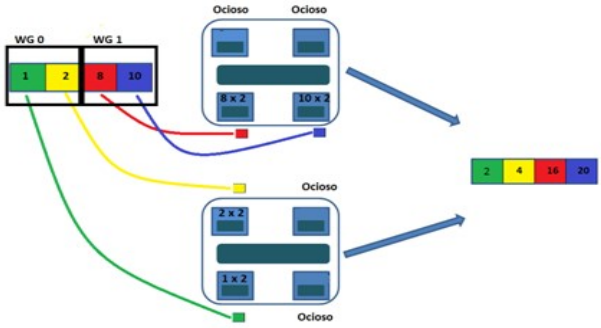

FIGURA 6 – MAPEAMENTO DA FUNÇÃO KERNEL COM WORK GROUP

Todo hardware que suporta OpenCL disponibiliza o tamanho máximo de WG suportado, porém, apesar de evitar que núcleos fiquem ociosos, nem sempre a quantidade máxima é a que terá melhor desempenho. Este artigo não irá entrar neste nível de detalhamento por se tratar de algo que exige maior proximidade com os conceitos de OpenCL e de arquitetura. Recomenda-se que inicialmente seja utilizada a quantidade máxima disponibilizada pelo fabricante.

III. TRABALHOS RELACIONADOS

A. WebGPU

Dakkak et al. [1] desenvolveram uma plataforma escalável de desenvolvimento *online* para cursos de programação em GPU. Os usuários da plataforma têm acesso a um ambiente onde podem desenvolver e executar códigos paralelos em OpenCL, OpenACC, CUDA e MPI. O desenvolvimento dessa plataforma resolve alguns problemas inerentes aos cursos gratuitos e *online* de programação que são: reduz a exigência de que todos os alunos tenham um computador pessoal semelhante aos comercializados atualmente, ambiente de programação com diversos fabricantes e diferentes configurações para iniciar o desenvolvimento, capacidade de armazenar os exercícios executados pelos alunos e as modificações realizadas ao longo do tempo, entre outros inúmeros benefícios disponibilizados por essa pesquisa. Porém, nenhum desses benefícios cobre a proposta deste artigo, que é disponibilizar um conjunto de aplicações que auxiliem no início da programação paralela em ambientes heterogêneos e no entendimento de suas características arquiteturais, pois a plataforma WebGPU disponibiliza somente o ambiente que viabiliza esse aprendizado e não o material didático a ser utilizado nos cursos.

B. Tetra

Finlayson et al. [2] acreditam que os principais obstáculos para o ensino de programação paralela na graduação são as linguagens de alto desempenho tais como: C, C++, Fortran, OpenCL e CUDA, classificando-as como linguagens de difícil utilização. Outro obstáculo, segundo os autores, é que os ambientes de desenvolvimento integrados (IDEs) e as ferramentas de depuração de código existentes para programação paralela disponíveis atualmente, são difíceis de usar, proprietárias ou contém poucos recursos de depuração de código. Para solucionar os obstáculos identificados anteriormente Finlayson et al. propõem a linguagem de programação Tetra, através de um ambiente de desenvolvimento integrado. A linguagem Tetra vem para facilitar o aprendizado de programação paralela, abrindo mão do desempenho, mas com foco na simplicidade, contendo características de linguagem de alto nível e realizando a paralelização por trechos através de marcadores, não exigindo inicialmente grandes mudanças no código sequencial, diferentemente de OpenCL e CUDA. O ambiente de desenvolvimento proposto vem com o objetivo de facilitar a execução e depuração do código paralelo, característica relevante para entendimento da execução dos diversos trechos paralelos.

Apesar de muito relevante, o esforço realizado pelos autores ao desenvolver a linguagem Tetra e o ambiente que possibilite principalmente depurar a execução de códigos paralelos, não foca no aprendizado de programação paralela para alto desempenho como a pesquisa realizada nesse artigo. Entretanto, mesmo não tendo o mesmo foco,

a linguagem Tetra e o conjunto de aplicações apresentadas nesse artigo desenvolvido em OpenCL, podem ser utilizados em conjunto para o ensino de programação paralela. Sendo a linguagem Tetra responsável por apresentar o universo paralelo, e o conjunto de aplicações proposto nesse artigo responsável por realizar uma transição suave entre a programação sequencial para a programação de alto desempenho paralela e heterogênea.

C. *Ensino de programação paralela usando Java*

Shafi et al. [10] desenvolveram as etapas e conteúdos de um curso para ensino de programação paralela, com o intuito de possibilitar que engenheiros de software tivessem contato com o novo paradigma de programação ao cursar o bacharelado em Engenharia de Software na *National University of Sciences and Technology* - (NUST) no Paquistão. O curso no primeiro momento introduz os conceitos de programação paralela, hardwares aceleradores, métricas para análise de desempenho e sistemas de memória distribuída e compartilhada. Após a introdução, o curso é dividido em três seções. A primeira seção cobre técnicas de programação paralela em sistemas de memória compartilhada, utilizando as linguagens de programação OpenMP [12], Intel Cilk Plus [13] e Java threads [11]. A segunda seção apresenta ferramentas de programação e APIs para programação paralela, em sistemas com memória distribuída, tendo como foco a utilização do software MPJ Express [14] para atividades práticas de desenvolvimento das aplicações paralelas. A terceira e última seção introduz a programação paralela avançada através da utilização de aceleradores de propósito geral GPU e o modelo de programação MapReduce através da utilização do Hadoop [15].

O trabalho desenvolvido por Shafi et al. é de grande valia para o meio acadêmico, pois aborda temas atuais que se relacionam com a programação paralela e disponibiliza uma metodologia para ensino destes conceitos. Porém, diferentemente da proposta desse artigo, a utilização de aceleradores é superficial segundo os próprios autores [10].

IV. Desafios de ensino

Diferente da estratégia para extrair maior desempenho dos processadores por meio do grande aumento de frequência, a estratégia em arquiteturas *multi/many-core* exige modificação do software para atingir o potencial do hardware disponível [6]. Sendo assim, os programas sequenciais que aumentavam de desempenho simplesmente trocando o processador para um com maior frequência, agora precisam ser reprogramados de forma a tornar possível a utilização de todos os núcleos. Além da necessidade de alteração do código sequencial para o código paralelo, com a introdução das arquiteturas heterogêneas, é necessário também identificar quais propostas de arquitetura executam melhor um trecho de código da aplicação. Como por exemplo, trechos onde a execução em aceleradores como GPU têm ganho de desempenho sobre a execução em CPU.

Neste novo contexto, aumenta-se a exigência acerca dos conhecimentos necessários para o desenvolvimento de aplicações que não desperdicem potencial de hardware. Portanto, um dos obstáculos para o ensino de programação nesse contexto está relacionado com a diminuição da abstração da arquitetura para aplicações de alto nível.

Outro obstáculo para o ensino é a quebra de paradigma de programação sequencial, sendo necessário introduzir as boas práticas de programação, a análise de trechos com potencial paralelo e análise de dependência de dados. Esse conjunto de exigências cria um conjunto de barreiras para o ensino em cursos de graduação, sendo necessário o desenvolvimento de técnicas e ferramentas que auxiliem na realização de uma transição suave entre a programação sequencial e a programação paralela de alto desempenho.

V. Abordagem Proposta

A abordagem proposta é composta por dois componentes. O primeiro é o Componente Introdutório, que é composto por uma aplicação que auxilia o programador a identificar os elementos de hardware que compõem a sua arquitetura, um resumo que auxilia a entender conceitualmente os modelos da linguagem OpenCL e como programar esses modelos para extração de maior desempenho da arquitetura alvo e, por fim, por instruções que possibilitam a reprodução de todos os resultados obtidos ao executar as aplicações do componente de transição, para fins de análise ou preparação de ambiente para iniciar a programação paralela. Já o Componente de Transição é composto por quatro aplicações com níveis de complexidade diferentes, iniciando com uma aplicação bem simples até uma aplicação de média complexidade. Todas as aplicações desenvolvidas que integram o componente de transição, implementam os conceitos de otimização de código com foco nas arquiteturas disponíveis no componente introdutório. A linguagem de programação OpenCL foi escolhida por ser uma linguagem de código aberto e ter alta portabilidade entre diferentes fabricantes de hardware, evitando assim, restringir o ambiente de desenvolvimento dos usuários da abordagem proposta.

A. *Componente Introdutório*

Identificação do ambiente: A identificação do ambiente é feita através da execução da aplicação desenvolvida para tal. Essa aplicação identifica as plataformas OpenCL, os dispositivos disponíveis e seus fabricantes. É utilizada por todas as outras para viabilizar a criação do contexto solicitado ao se programar em OpenCL.

Resumo da abstração OpenCL: OpenCL é uma linguagem com alta portabilidade. Essa portabilidade é possível devido ao consórcio criado pelo Khronos Group que é integrado por empresas como AMD, Intel, Altera (adquirida pela Intel) Nvidia, IBM, entre outros fabricantes de hardware que concordaram em suportar a linguagem OpenCL. Para que o programador não tenha que se preocupar com as diferenças entre as arquiteturas de diferentes fabricantes, OpenCL disponibiliza uma abstração de arquitetura, onde todos os fabricantes devem implementar essa abstração. Para que o programador inicie a programação em OpenCL é necessário que ele tenha compreendido os conceitos da mesma. Para auxiliar os estudantes é disponibilizado um resumo[1] que aborda as principais características dessa abstração, viabilizando assim, o início da prática de programação.

Instruções para reprodução dos resultados: As aplicações disponibilizadas no componente de transição ***foram*** executadas utilizando a ferramenta de

[1]https://github.com/cart-pucminas/OpenCL-freshmen

desenvolvimento Microsoft Visual Studio Comunnity 2017, uma placa aceleradora AMD Radeon R7 360 e um processador Xeon E3-1240. Para que seja possível reproduzir os resultados obtidos, as configurações necessárias foram disponibilizadas no arquivo "Configuração do ambiente de desenvolvimento" que está contido no componente introdutório[1]. Instruções também podem ser utilizadas para configuração do ambiente para iniciar a programação de outras aplicações.

Introdução de conceitos essenciais: Para compreensão do ganho de desempenho, e do modelo de programação, são apresentados conceitos de paralelismo, dependência de dados, concorrência, balanceamento de carga, paralelismo de dados e de tarefas. Conceitos de arquitetura também são apresentados, tais como: hierarquia de memória, memória distribuída e centralizada e arquitetura de von Neumann.

B. *Componente de transição*

Multiplicação de vetor simples: Esta aplicação consiste na multiplicação de todos os elementos de um vetor por uma constante de forma paralela, ou seja, cada núcleo de processamento é responsável por realizar a multiplicação de um elemento do vetor. Esta aplicação foi escolhida com o objetivo de introduzir a programação paralela, pois além de sua simplicidade, é uma aplicação que facilmente se identifica a não existência de dependência de dados, possibilitando a divisão de trabalho de forma paralela. A aplicação multiplicação de vetor simples é a aplicação de menor complexidade do conjunto proposto nesse artigo, sendo a aplicação recomendada para o primeiro contato do programador com a prática de programação paralela.

Multiplicação de vetor x vetor: A multiplicação de vetor por vetor contém todas as características que a multiplicação de vetor simples possui, porém com pequenas diferenças que aumentam um pouco a complexidade da aplicação. Nessa aplicação, cada elemento de um vetor é multiplicado pelo mesmo elemento de outro vetor, tendo como saída um vetor com o resultado da multiplicação. Essa aplicação foi escolhida para compor o conjunto proposto justamente para que o aumento de complexidade fosse crescente e possibilitasse uma transição suave entre os diferentes paradigmas de programação.

Multiplicação matriz x constante: Diferentemente da aplicação multiplicação de vetores, a multiplicação de matrizes utilizando a linguagem OpenCL exige que se utilize um recurso disponibilizado pela linguagem que é a possibilidade de lidar com mais de uma dimensão em um trecho paralelo. Por conta dessa diferença, essa aplicação foi escolhida para fazer parte da abordagem.

Ordenação: A aplicação de ordenação é um algoritmo de força bruta sem dependência de dados. Esse algoritmo foi escolhido para demonstrar o enorme potencial de hardware que pode ser desperdiçado ao se programar de forma sequencial.

Redução numérica: A aplicação de redução numérica consiste em retornar a soma de todos os elementos de um vetor de forma paralela. Essa aplicação foi escolhida, pois nela é possível introduzir os conceitos de corrida e dependência de dados, aumentando assim a complexidade da aplicação e possibilitando o ensino de conceitos importantes da programação paralela.

VI. ANÁLISE DE DESEMPENHO

Esta seção é dividida em duas partes, onde a primeira detalha o ambiente utilizado para produção dos resultados, e a segunda apresenta três resultados obtidos utilizando duas aplicações da proposta com o objetivo de conscientizar, tanto a necessidade de desenvolvimento com foco na arquitetura, quanto na necessidade de evitar o desperdício de hardware através da paralelização dos códigos sequenciais.

A. *Materiais*

Para que o controlador consiga enviar os comandos aos dispositivos, foi necessário instalar a implementação OpenCL da AMD e configurar a IDE Microsoft Visual Studio Community 2017. O sistema operacional utilizado foi o Windows 10 de 64 bits, a arquitetura foi composta por um processador de propósito geral Intel Xeon E3-1240, uma placa aceleradora gráfica AMD Radeon R7 360 e 64 Gigabytes de memória principal.

A GPGPU AMD Radeon R7 360 contém 768 elementos de processamento. O processo de fabricação do chip é de 28nm, o Clock dos elementos de processamento é de 1050MHz, a largura de banda de comunicação com a memória é de 128-bit com capacidade máxima de transferência de 104 GB/s e contém capacidade de armazenamento de 2GB GDDR5 em sua memória.

O processador Intel Xeon E3-1240 contém 4 núcleos de processamento com a tecnologia SMT - *Simultaneous Multithreading* que possibilita a execução de 8 threads em paralelo. O processo de fabricação do chip é de 32 nm, a frequência máxima dos núcleos é de 3,7 GHz e tem capacidade de armazenamento de 128 kB em sua *cache* nível 1, 1024 kB em sua *cache* nível 2 e 8 MB em sua memória *cache* nível 3.

As análises de desempenho realizadas utilizando as aplicações de ordenação e de multiplicação simples tiveram carga de trabalho fixa e foram executadas vinte vezes, a unidade de tempo está em milissegundos para todas as avaliações.

B. *Resultados de desempenho*

A primeira análise mostra o forte impacto ao se programar em OpenCL sem se preocupar com a arquitetura dos dispositivos que estão sendo utilizados. A segunda análise apresenta o impacto de uma característica da aplicação multiplicação de vetor simples sobre seu desempenho ao ser paralelizada em um ambiente heterogêneo e a terceira análise mostra o enorme desperdício de hardware ao se programar de forma sequencial.

Impacto do grupo de trabalho: Ao se programar em OpenCL e não distribuir a carga de trabalho entre as unidades de computação do dispostivo corretamente, é possível que potencial de hardware seja desperdiçado. A Figura 7 ilustra o possível desperdício de hardware caso o programador não distribua corretamente a carga de trabalho para a aplicação “multiplicação de vetor

[1]https://github.com/cart-pucminas/OpenCL-freshmen

simples", pois ao se dimensionar corretamente a carga de trabalho para as unidades de computação evita-se que alguma unidade de computação ou elemento de processamento fique ocioso. No caso onde a carga é dimensionada corretamente, é possível melhorar o desempenho da aplicação "multiplicação de vetor simples" em aproximadamente três vezes.

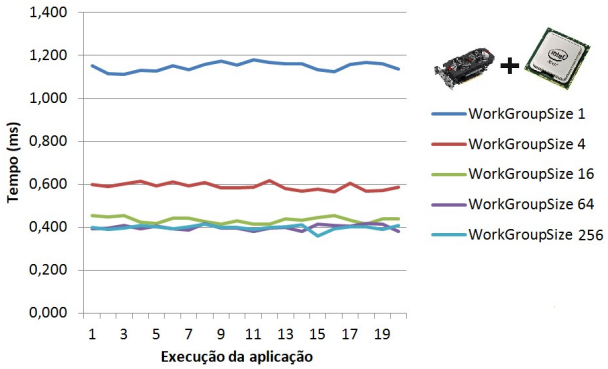

FIGURA 7 – INFLUÊNCIA DO WORKGROUP SIZE

1) ***Análise da aplicação multiplicação de vetor simples:*** Para que uma aplicação tenha um bom ganho de desempenho ao ser executada em um acelerador como a arquitetura de GPU utilizada, uma das características necessárias da aplicação é que a latência de acesso à memória seja ocultada pela carga de trabalho que será realizada sobre os dados.

A aplicação "multiplicação de vetor simples" realiza poucas operações lógicas após transferir todo o vetor para a memória do dispositivo. Sendo assim, o seu ganho de desempenho sobre a aplicação sequencial é ocultado pelo tempo de leitura e transferência de dados para a memória do dispositivo. Esse ganho de aproximadamente 1,5x pode ser visualizado na Figura 8, onde as aplicações sequencial e paralela realizaram a multiplicação de 196.777.216 posições de um vetor pela constante 8.

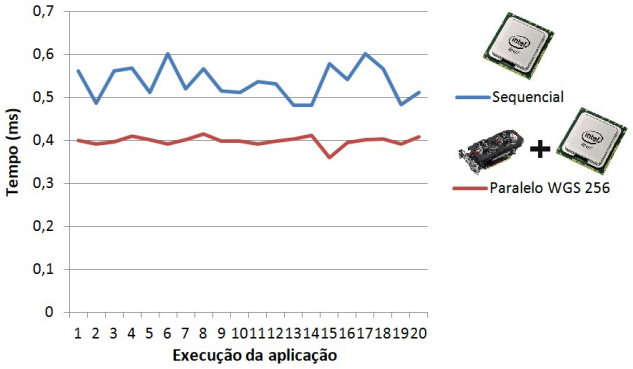

FIGURA 8 – MULTIPLICAÇÃO DE VETOR POR CONSTANTE

2) ***Desperdício de potencial de hardware:*** A aplicação "ordenação por força bruta" obtém enorme ganho de desempenho quando programada em OpenCL de forma paralela. Esse resultado é obtido pois essa aplicação contém características que contribuem para um ótimo ganho de desempenho ao ser executada em placas aceleradoras com arquiteturas SIMD. A aplicação contém caraterísticas como: i) mesma operação sobre diferentes dados, ii) tempo de transferência dos dados muito inferior ao tempo de processamento e iii) não existência de dependência de dados nem condição de corrida. Aplicações com essas características obtém bons resultados ao serem paralelizadas. A Figura 9 ilustra o ganho de desempenho da aplicação de "ordenação por força bruta" sobre a mesma aplicação sequencial ao ordenar um vetor com cem mil posições.

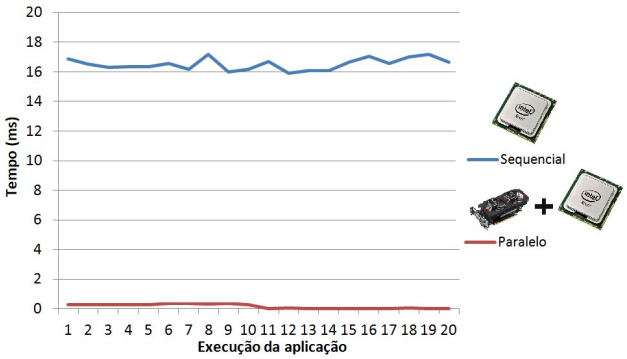

FIGURA 9 – ORDENAÇÃO POR FORÇA BRUTA

VII. CONCLUSÃO

A tendência de processadores com mais de um núcleo já é realidade e infelizmente os cursos de graduação na área de computação, possuem dificuldade para introduzir a computação paralela em ambientes heterogêneos na sua grade curricular. Essa dificuldade resulta em produção de códigos sequenciais para serem executados em máquinas paralelas tendo como consequência enorme desperdício de hardware. Diminuir os desafios do ensino de programação paralela é essencial para modificar esse cenário. Sendo assim, trabalhos de pesquisa precisam dar continuidade a essa linha de trabalho disponibilizando ferramentas e técnicas que viabilizem o ensino de programação paralela em ambientes heterogêneos nos cursos de graduação.

Iniciar a programação em linguagens paralelas é um desafio e exige a quebra do paradigma de programação sequencial porém, no cenário atual onde a era *many-core* é uma realidade, programar de forma sequencial é um enorme erro. OpenCL permite ir além, você não só programa de forma paralela, mas programa usufruindo de todos os benefícios de uma arquitetura heterogênea com alto nível de portabilidade entre diferentes fabricantes. A explicação acerca da abstração OpenCL disponibilizada nesse artigo, possibilita que as próximas etapas de estudo a cerca da linguagem sejam menos complexas, pois dominando os conceitos básicos, os demais são adquiridos de forma gradual.

O próximo passo é validar a curva de aprendizado ao se utilizar a abordagem proposta em cursos de graduação em Computação. Outro passo importante é o desenvolvimento de dois novos componentes para a abordagem proposta, chamados de intermediário e avançado com novas aplicações para possibilitar a continuidade do estudo de forma mais abrangente. O componente intermediário será composto por aplicações mais complexas que o introdutório, como por exemplo a aplicação K-means. Já o componente avançado abordará a escolha de algoritmos paralelos de acordo com o problema a ser solucionado,

levando em consideração as características da arquitetura paralela alvo.

AGRADECIMENTOS

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001, além do apoio do CNPq, FAPEMIG e PUC Minas.

REFERÊNCIAS

[1] A. Dakkak, C. Pearson and W. M. Hwu, "WebGPU: A Scalable Online Development Platform for GPU Programming Courses," 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL, 2016, pp. 942-949 .

[2] I. Finlayson, J. Mueller, S. Rajapakse and D. Easterling, "Introducing Tetra: An Educational Parallel Programming System," 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, Hyderabad, 2015, pp. 746-751.

[3] S. Borkar, 2007. Thousand core chips: a technology perspective. Proceedings of Annual Design Automation Conference (DAC) pp.746-749.

[4] R. W. Keyes, "Fundamental limits of silicon technology," in Proceedings of the IEEE, vol. 89, no. 3, pp. 227-239, Mar 2001.

[5] C. H. Lu, C. S. Lin, H. L. Chao, J. S. Shen and P. A. Hsiung, "Reconfigurable Multi-core Architecture -- A Plausible Solution to the Von Neumann Performance Bottleneck," 2013 IEEE 7th International Symposium on Embedded Multicore Socs, Tokyo, 2013,pp.159-164. doi: 10.1109/MCSoC.2013.32.

[6] R. Muresano, D. Rexachs, E. Luque, "Learning parallel programming: a challenge for university students", Procedia Computer Science, Volume 1, Issue 1, 2010, pp. 875-883.

[7] A.MUNSHI, "The opencl specification". In: Hot Chips 21 Symposium (HCS). IEEE, 2009. p. 1-314.

[8] M. Paprzycki, "Education: Integrating Parallel and Distributed Computing in Computer Science Curricula," in *IEEE Distributed Systems Online*, vol. 7, no. 2, pp. 6-6, Feb. 2006

[9] A. Marowka, "Think Parallel: Teaching Parallel Programming Today," in *IEEE Distributed Systems Online*, vol. 9, no. 8, pp. 1-1, Aug. 2008

[10] A. Shafi, A. Akhtar, A. Javed and B. Carpenter, "Teaching Parallel Programming Using Java," *2014 Workshop on Education for High Performance Computing*, New Orleans, LA, 2014, pp. 56-63

[11] S. Oaks and H. Wong, Java Threads, Third Edition, 3rd ed. O'Reilly Media, Inc., 2004

[12] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, Parallel Programming in OpenMP.Morgan Kaufmann Publishers Inc., 2001.

[13] M. Frigo, C. E. Leiserson, and K. H. Randall, "The implementation of the Cilk-5 multithreaded language," in Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation (PLDI),1998, pp. 212–223.

[14] A. Shafi, B. Carpenter, and M. Baker, "Nested parallelism for multi-core HPC systems using Java," Journal of Parallel and Distributed Computing, vol. 69, no. 6, pp. 532 – 545, 2009.

[15] T. White, Hadoop: The Definitive Guide, 1st ed. O'Reilly Media, Inc., 2009

[16] V. Kumar, A. Grama, A. Gupta, and G. Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/ Cummings Publishing Company, 1994.

[17] W. Gropp, K. Kennedy, L. Torczon, A. White, J. Dongarra, I. Foster, and G. C. Fox. The Sourcebook of Parallel Computing (The Morgan Kaufmann Series in Computer Architecture and Design). Morgan Kaufmann, November 2002.

[18] HENNESSY, John L.; PATTERSON, David A. Computer architecture: a quantitative approach. Elsevier, 2011.

[19] FLYNN, M. J.; RUDD, K. W. Parallel Architectures. ACM Computing Surveys (CSUR), 1996,pp. 67-70 .