Abstração OpenCL 1.2

INTRODUÇÃO

Inicialmente é necessário deixar claro que o material disponibilizado nesse documento foi criado para introduzir as principais características da linguagem OpenCL, de forma reduzida e simplificada, o que não substitui a necessidade de leitura da especificação OpenCL disponível em: https://www.khronos.org/registry/OpenCL/. O documento 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 a explicação a versão 1.2 não torna o documento desatualizado, pois as características exemplificadas aqui se mantém na versão 2.0.

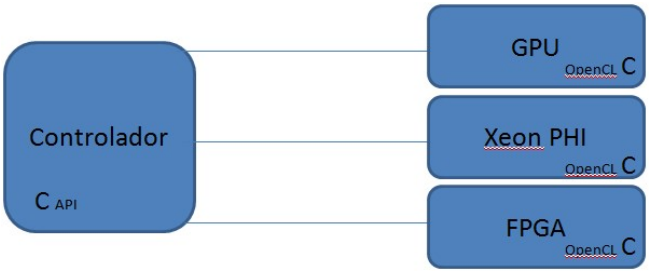
A linguagem de programação OpenCL, disponibilizada pelo consórcio Khronos Group, surge com o propósito de unificar o cenário de programação paralela e heterogênea, através de uma linguagem livre, tendo como objetivo o aumento de produtividade dos programadores evitando que os mesmos tenham que se adaptar aos diferentes fabricantes de hardware. 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.

Essa explicação é dividida da seguinte forma: Na primeira sessão 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 sessão 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. Desta forma não será dado ênfase à explicação dos detalhes arquiteturais. Na terceira sessão é apresentada uma explicação a cerca do mapeamento do processamento paralelo para os elementos de processamento (PE).

I. SESSÃO

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

FIGURA 1 - PRINCIPAIS COMPONENTES OPENCL



FONTE: PRÓPRIO AUTOR

A. *Visão geral*

Como exemplificado na Figura 1, a proposta da programação em OpenCL é extrair eficientemente desempenho de uma arquitetura heterogênea, isso é capaz 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. Dessa forma é possível que trechos dos códigos sejam executados em hardwares diferentes, como por 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 possibilitando, assim, extrair as qualidades de diferentes propostas de arquitetura.

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.

A API controladora é utilizada para coordenar o funcionamento de todos os dispositivos, sendo assim o programador utiliza essa 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.

II. SESSÃO

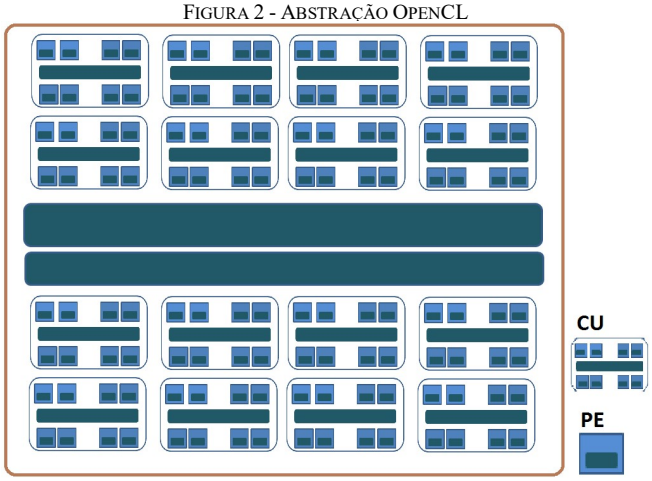
Após entender que a programação em OpenCL é uma forma de se extrair desempenho através da paralelização dos códigos e da utilização eficiente das diferentes características de um ambiente heterogêneo, é necessário agora aprender como OpenCL viabiliza a paralisação desses códigos e , como ele possibilita que se gerencie os dispositivos de diferentes fabricantes afim de extrair deles suas melhores características. Ao aprender como realizar ambas as tarefas mencionadas anteriormente é possível evitar o enorme desperdício de potencial de hardware como atualmente grande parte das aplicações o fazem.

Sendo assim, nessa sessão será apresentada de forma resumida alguns elementos que possibilitam e influenciam na hora de se programar em 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.

A. *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.

FIGURA 2 - ABSTRAÇÃO OPENCL



FONTE: PRÓPRIO AUTOR

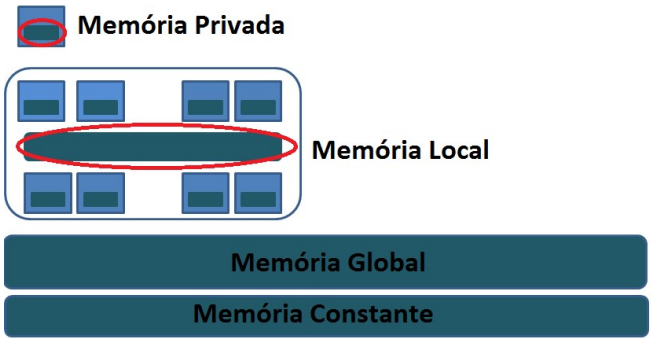
1) ***Elemento de processamento (PE):*** Os elementos de processamento (PE) são os responsáveis por executar todas as linhas de código desenvolvidas em OpenCL C, ou seja, são eles os responsáveis por executar todas as instruções. Se você tem 128 elementos de processamento na sua placa de vídeo aceleradora você é capaz de executar 128 "threads" em paralelo se desejar e a aplicação permitir. Outra opção é dividir o trabalho entre as unidades de computação (CU), dessa forma 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.

2) ***Unidade de Computação (CU):*** A unidade de computação é um aglomerado de elementos de processamento (PE). 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.

B. *Memória*

A abstração OpenCL especifica quatro diferentes regiões de memória a saber: a memória global, privada, constante e local, que podem ser visualizadas na Figura 3. A utilização correta das memórias resulta em aumento de performance, pois você pode melhorar a comunicação e diminuir a latência.

FIGURA 3 - REGIÕES DE MEMÓRIA



FONTE: PRÓPRIO AUTOR

1) ***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.

2) ***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.

3) ***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, porém não vamos entrar nesse nível de detalhamento.

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

III. SESSÃO

A. *ND-Range*

Para quebrar o paradigma sequencial e iniciar a programação paralela a linguagem OpenCL utiliza o conceito ND-Range. Esse 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 esse conceito inicialmente é pensar que esse conjunto de índices são uma estrutura de repetição FOR, onde cada índice é uma iteração e a quantidade de índices gerados (*WorkSize*) é igual ao limite de repetições definidos no FOR. 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 interação anterior tenha acontecido para que a próxima aconteça. Para facilitar o entendimento a Figura 4 ilustra essa comparação.

FIGURA 4 - COMPARAÇÃO FOR / ND-RANGE

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

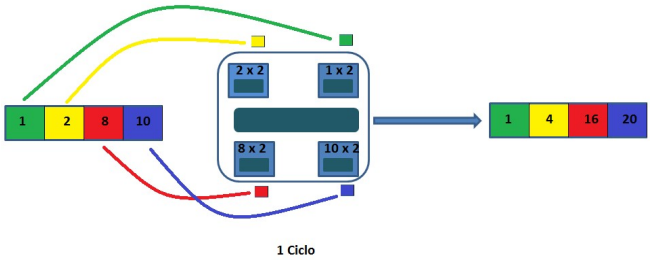
FONTE: PRÓPRIO AUTOR

B. *Mapeamento ND-Range*

Para facilitar o entendimento do mapeamento dos Kernels para os elementos de processamento, 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



FONTE: PRÓPRIO AUTOR

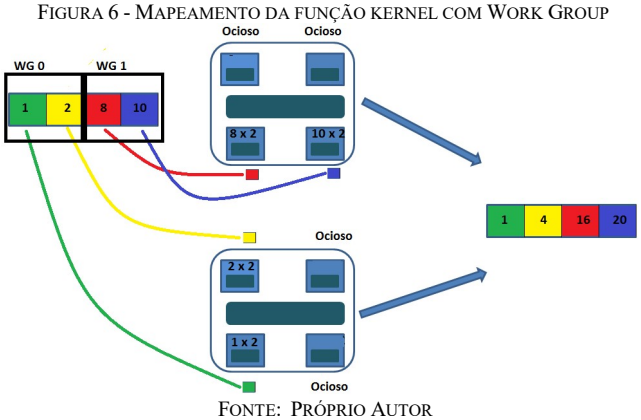
C. *Work Group*

Como explicado anteriormente a paralelização em OpenCL utiliza o conceito ND-Range. Esse 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. Esses 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, dessa forma ele garante que nenhuma CU ficará ociosa. A divisão do trabalho é realizada criando grupos de trabalho (WG). A Figura 6 ilustra essa divisão para fins didáticos deixando núcleos ociosos propositalmente afim de demonstrar possíveis desperdício 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.

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. Esse documento não irá entrar nesse nível de detalhamento por se tratar de algo que exige maior proximidade com os conceitos de OpenCL e de arquitetura. O documento recomenda que inicialmente utilize a quantidade máxima disponibilizada pelo fabricante. (Esse desempenho pode ser verificado nos gráficos de resultado do artigo que menciona esse documento).

FIGURA 6 – MAPEAMENTO ND-RANGE COM WG

FIGURA 6 - MAPEAMENTO DA FUNÇÃO KERNEL COM WORK GROUP



FONTE: PRÓPRIO AUTOR

CONSIDERAÇÕES FINAIS

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 documento, 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.