Conceitos básicos para OpenCL

Por Raphael Costa e Rafael Corsi

O OpenCL é uma arquitetura para programação de uma coleção heterogênea de CPUs, GPUs e outros dispositivos (que inclui FPGA) para computadores, organizada em uma só plataforma [1]. É um framework para programação paralela, que, através de sua linguagem, API, bibliotecas e sistemas *runtime*, possibilita desenvolver programas que sejam portáveis e ainda sejam eficientes.

Contexto do OpenCL para FPGAs

A Altera Corporation era uma fabricante americana líder na produção de Dispositivos Lógicos Reprogramáveis, existindo desde 1984. No dia 01 de Junho de 2015, a Intel e Altera anunciaram que a Intel adquiriria a Altera em uma negociação envolvendo aproximadamente $16.7 bilhões de dólares, e assim o negócio foi fechado oficialmente no final deste mesmo ano. [2]

Hoje a Intel dá continuidade a linha de produtos principal da Altera, como as placas da série Stratix, Arria e Cyclones e o Software de Design chamado Quartus.

Intel FPGA SDK for OpenCL

Até a compra da Altera, a Intel não dominava este mercado e nem esta Tecnologia. Assim, a empresa resolveu por criar uma maneira rápido e fácil das pessoas com afinidade a OpenCL começarem a fazerem seu uso para FPGAs. A empresa disponibiliza o *Intel FPGA SDK for OpenCL* [3], que é um kit de desenvolvimento de software, funcionando como uma API para programar FPGAs, GPUs e CPUs Intel através de códigos arquitetados em OpenCL.

Utilizando o OpenCL, podemos caracterizar dois benefícios principais: programacao em linguagem mais usual do que as que são utilizadas para a sintetização de Hardware (VHDL, Verilog, etc). Ou seja, podemos transferir um programa em C para a FPGA diretamente sem perder a eficiência de um programa em HDL (Hardware Description Language); Ainda, este programa contará com execução em paralelo, que é muito mais eficiente quando comparado a um programa de execução concorrente. Além disso, podemos transferir o mesmo programa para CPUs Intel, executando-o normalmente.

Funcionamento e conceitos básicos

Como um programa roda

Primeiro, vamos definir 3 conceitos básicos para podermos utilizar nas explicações que virão:

1) Host. Host é o computador que hospeda o dispositivo acelerador, na maioria das vezes, é o seu Desktop, notebook, etc.
2) Device. Device é o próprio dispositivo acelerador. Neste contexto, é a nossa FPGA. Porém, em outros contextos, pode ser uma GPU por exemplo.
3) Kernel. O kernel de um OpenCL é o programa que será rodado no device escolhido. É usualmente escrito em C. Daremos um exemplo posteriormente.

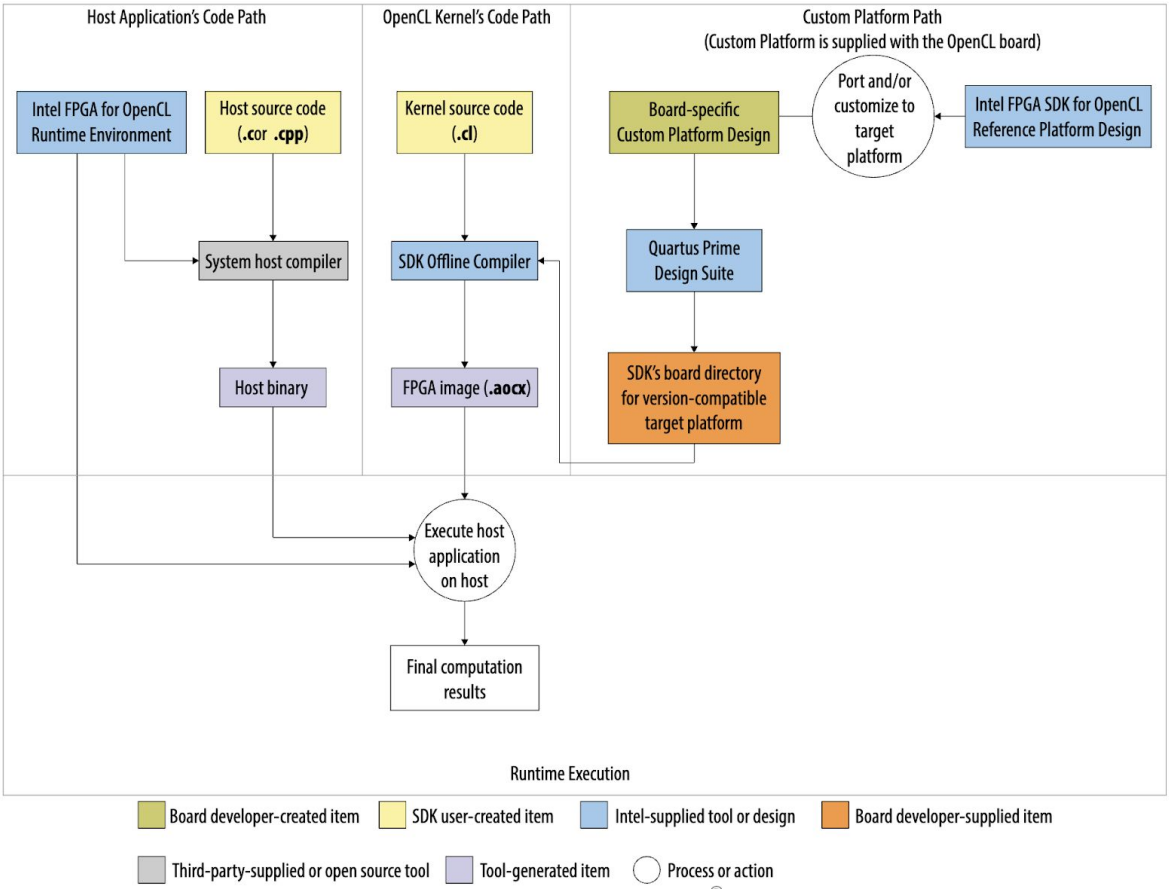


Imagem 1: diagrama de funcionamento de um programa em OpenCL

Como pode-se observar na imagem acima, o OpenCL funciona basicamente através de 3 ferramentas:

- O Computador e o compilador do Host (Desktop)
- O kernel do OpenCL e seu respectivo compilador
- A plataforma custom (FPGA)

O compilador do OpenCL pode compilar seu código de diversas formas, sendo elas em um processo único ou até multiprocesso (Isso mesmo, até a compilação do código pode ser paralelizada). Esta configuração depende diretamente da complexidade do kernel que você está executando. Um arquivo kernel do OpenCL (.cl) contém o source do código que será executado na FPGA, como citado anteriormente. O compilador agrupa diversos kernels em arquivos temporários e então os compila para gerar basicamente dois arquivos:

- Um .aoco object file, basicamente o .o usual de uma compilação do GCC, arquivo intermediário que será utilizado em estágios posteriores de compilação.
- Um .aocx image file, que é a configuração do hardware que será implementado na FPGA para rodar tais kernels e algumas informações essenciais do sistema para a troca de informações entre FPGA e sistema host.

Uma das principais diferenças entre a execução de um programa sequencial e de um programa em OpenCL é a de que na implementação sequencial, em processadores convencionais, no mais baixo nível, é o Program Counter que dita a sequência e qual instrução será executada no hardware. No tipo de programa que lidamos aqui, como programas implementados pelo SDK do OpenCL, as instruções são executadas assim que os pré requisitos para tal estão prontos.

Threads, work-items e work-groups

No contexto de OpenCL, cada thread é chamada de work-item e múltiplos work-items são agrupados para formar um work-group. Durante a execução de uma aplicação, as várias threads são distribuídas formando múltiplos work-groups. Dentro de cada work-group, seus work-items são sincronizados utilizando barreiras e dentro de um work-group, os dados podem ser compartilhados através das memórias mais rápidas do chipset. Em contrapartida, para compartilhar dados entre work-groups é necessário utilizar memórias mais lentas. O número de work-items dentro de um work-group, no contexto do código, é chamado de local work size, enquanto que o número total de work-items necessários para executar uma aplicação é chamado de global work size. Por último e não menos importante, work-items e work-groups podem ser arranjados de forma multidimensional (até 3 dimensões), dentro de um espaço que recebe o nome de NDRange ou, visualmente, chamamos de Grid.

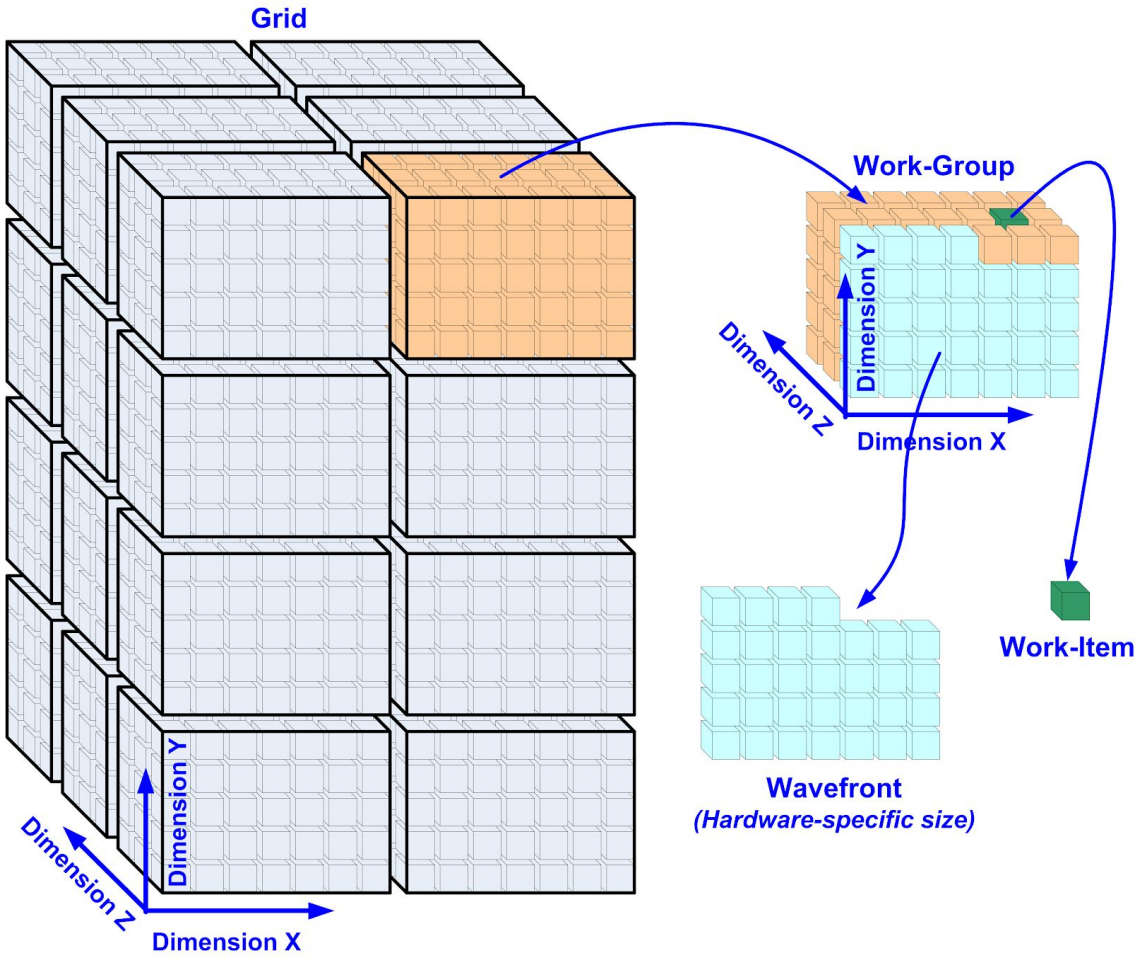


Imagem 2: Abstração de um grid

Tipos de memória

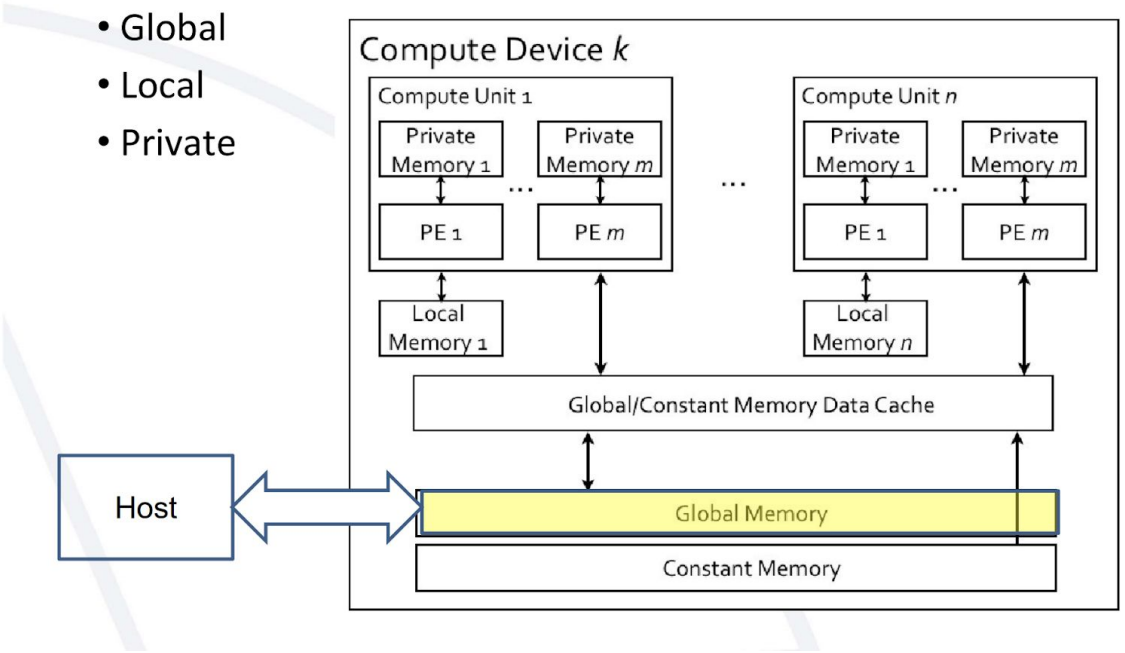


Imagem 3: Diagrama de memórias em um OpenCL

Global (Gb): esta região de memória é uma região externa ao chip e localizada no device. É, usualmente a maior memória do device, porém, a mais lerda quando comparada aos outros tipos de memória listados abaixo. Está memória é visível a todos os work-items de todos os workgroups. Ainda, é a região de memória que o Host tem acesso. Em analogia, a memória Global é parecida com o HD de um computador usual.

Local(Mb): esta região de memória reside dentro do chip de processamento e é compartilhada entre os work-items de um work-group. Cada work-group possui sua região de memória local, portanto, outros work-groups não possuem acesso a memórias que não sejam destinadas a ele. Em analogia, a memória Local é parecida com a RAM de um computador usual.

Private (Kb): Esta região de memória é aquela destinada a um único work-item. É, de longe, a memória mais rápida do chip, porém a menor. Em analogia, a memória Private é parecida com o Cache de um processador usual.

Técnicas básicas de aceleração de código

Unrolling a Loop (unroll Pragma)

O unroll de loops é uma das práticas mais utilizadas em aceleração de código. Basicamente, a nível de código de máquina, você transforma o que seria um loop em várias vezes o mesmo código (mostrar exemplo de assembly que fica pulando para flags e executando).

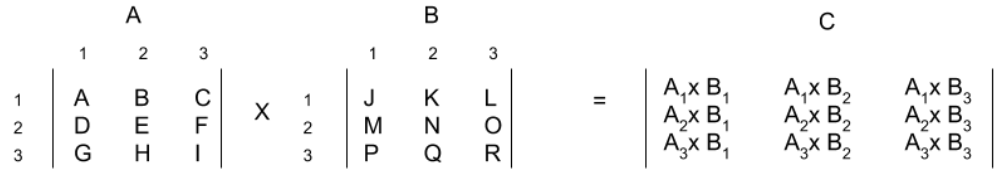
Deve-se tomar muito cuidado com este tipo de instrução já que só funciona caso não haja nenhum tipo de dependência de código (conhecidas como loop-carried dependencies). Para estes casos, o unroll de loops ajuda a reduzir a latência e o overhead de código nas FPGAS.

Uso: `#pragma unroll <N>`

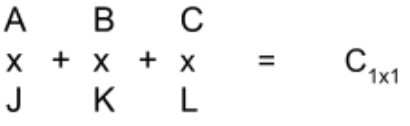
Onde N é o número de vezes que se deseja "desenrolar" o loop. Caso o N não seja especificado dizemos que o loop foi inteiramente desenrolado *(Fully unroll a loop*).

SIMD (Single Instruction, Multiple Data)

Nesta técnica, como o próprio nome sugere, a mesma instrução é aplicada simultaneamente a diversos dados para produzir mais resultados. Assim, exemplificando, vamos multiplicar uma matriz qualquer A por uma matriz B:



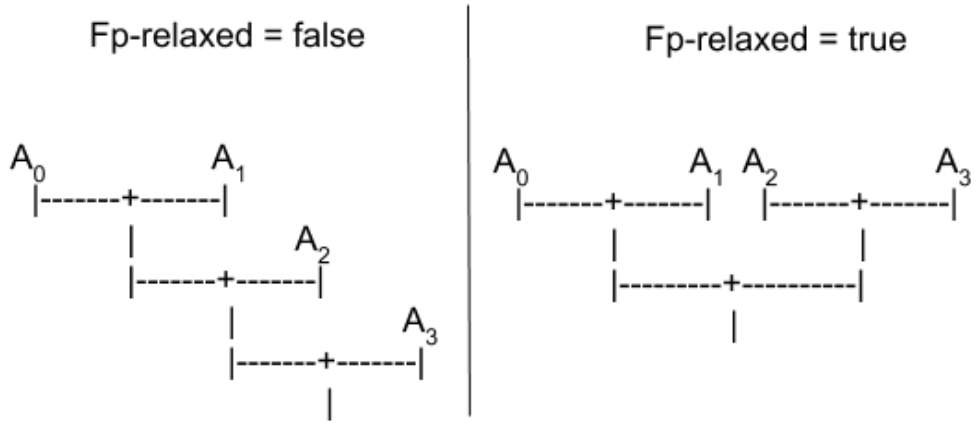
No OpenCL, além de paralelizar o cálculo da matriz C, de forma que cada work-item calcula uma posição da matriz resultado, utilizamos instruções do tipo SIMD para paralelizar a multiplicação de cada linha por coluna. Assim, para o cálculo da posição 1x1, por exemplo, em uma instrução do tipo SIMD, a multiplicação ocorre paralelamente dentro de um mesmo work-item.



Fp-relaxed

Esta técnica é adicionada através de uma flag no ato da compilação, e se resume em otimizar a forma em que fazemos uma soma composta:

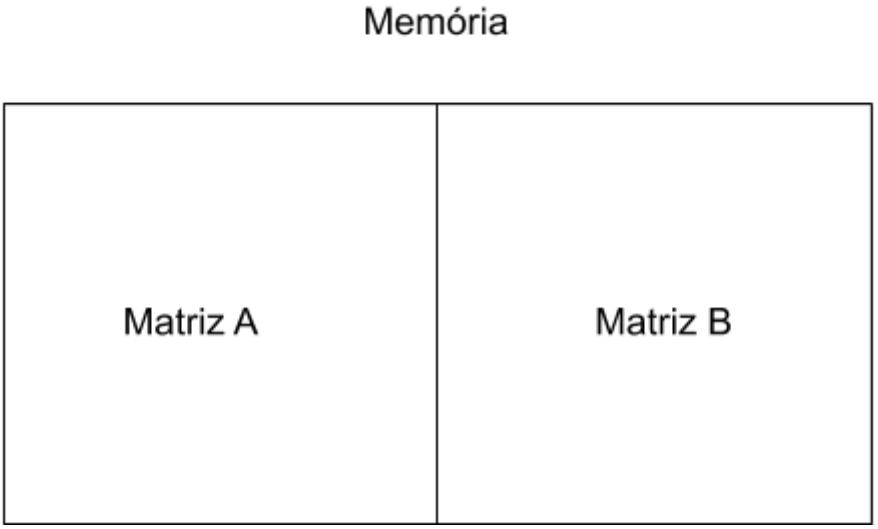
$$A_0 + A_1 + A_2 + A_3$$



Desta forma, quando o fp-relaxed está ativado, as somas são mais eficientes.

No-intervaling

Esta técnica também é adicionada através de uma flag na compilação e consiste em dividir a memória em 2 partes distintas permitindo ler a matriz A e a matriz B simultaneamente sem que haja problemas de acesso.



Bibliografia

[1] - The OpenCL Specification
<https://www.khronos.org/registry/OpenCL/specs/2.2/html/OpenCL_API.html>\

[2] - Intel Acquisition of Altera
https://newsroom.intel.com/press-kits/intel-acquisition-of-altera/#gs.kxoqod

[3] - Intel FPGA SDK for OpenCL
https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html
[4] - Intel FPGA SDK for OpenCL Overview
https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807965224.html#mwh1391807939093

[5] - Reza Zohouri,Hamid, et al. “High Performance Computing with FPGAs and OpenCL”. Agosto 2015.
https://arxiv.org/pdf/1810.09773.pdf