Comparação de eficiência entre OpenCL e CUDA

Thiago de Gouveia Nunes

19 de outubro de 2012

Motivação

- Em GPGPU, é natural se preocupar com a máxima performance dos programas. Esse trabalho consiste na comparação de desempenho e das abstração de uma GPU entre duas plataformas para programação paralela em GPUs, **OpenCL** e **CUDA**.

GPGPU

É usar uma GPU, pricipalmente usada para Computação Gráfica, para computar aplicações que seriam mandadas para uma CPU. A principal vantagem disso é o paralelismo. A GPU tem vários cores SIMD, que são usados para resolver problemas que levariam muito mais tempo numa CPU (como uma multiplicação de matriz) em poucos milisegundos.

OpenCL

O OpenCL é uma framework de programação paralela para sistemas híbridos. Isso quer dizer que podemos rodar aplicações que usam OpenCL tanto em CPUs como em GPUs. OpenCL foi uma iniciativa da Apple, e depois sofreu várias melhorias por times de várias empresas, entre elas Intel e Nvidia. OpenCL pode ser usado em C e C++.

CUDA

CUDA é uma plataforma para programação paralela em GPUs desenvolvida pela Nvidia. Ela suporta C, C++, Java, Python, Fortran, etc.

Termos Técnicos

A explicação dos termos técnicos que serão usados:

Kernel Função que será executada em cada processador da placa de vídeo.

Host Programa que será executado na CPU e tem a função de preparar o ambiente para o kernel e rodá-lo na GPU.

SIMD *Single Instrution, Multiple Data.* Os processadores das GPUs são desse tipo. Cada um recebe o mesmo código, mas roda usando dados diferentes.

Device GPU ou CPU.

CU *Computer Unit*, um processador.

PE *Processing Element*, um core do processador.

Modelos

OpenCL usa uma hierarquia de 4 modelos:

1. Modelo de Plataforma
2. Modelo de Execução
3. Modelo de Memória
4. Modelo de Programação

Modelo de Plataforma

O modelo de plataforma é responsável por representar o Host ligado a um ou mais Devices, cada com um ou mais CU's, cada CU com um ou mais PE's. A aplicação do OpenCL fica no Host, que é responsável por direcionar os kernel's para os PE's em cada Device

Modelo de Execução

O modelo de execução representa o programa que roda no Host, e todos os Kernel's. O host prepara um contexto para a execução do kernel e o roda.
Quando o Kernel começa a executar, um espaço de indices é criado, onde cada indice representa uma cópia do kernel rodando. Esse espaço pode ter dimensão 1, 2 ou 3.

Contexto

O contexto onde o kernel é executado tem os seguintes recursos:

- Devices
- Kernels
- Program Objects: O código fonte do kernel.
- Memory Objects: Objetos que representão a memória que será modificada pelo kernel. Tanto o host como o kernel tem acesso a ela.
- Command-Queue: Fila que cuida da ordem em que tudo será executado num device.

Modelo de Memória

Cada instância do kernel tem acesso a 4 tipos de memórias distintas:

1. Global Memory - Toda e qualquer instância de um kernel tem acesso a essa memória.
2. Constat Memory - Memória que permanece fixa ao andar da execução.
3. Local Memory - Região da memória dividida pelos kernel's de um mesmo CU.
4. Private Memory - Região privada para cada instância de um kernel.

Modelo de Programação

O OpenCL suporpa dois modelos de programação, o Data Parallel e o Task Parallel.

1. Data Parallel - É o modelo padrão do OpenCL, cria várias instâncias iguais do kernel, cada uma recebendo argumentos diferentes.
2. Task Parallel - Nesse modelo, cada kernel tem só uma instância, e o programador tem que usar outros objetos (Ex: vetores) para expressar o paralelismo do kernel.

Ideia

Para comparar a performance das duas plataformas, a ideia inicial é usar dois kernel's, um para comparar o tempo que cada um leva pra computar uma operação com muita movimentação de memória, e outro com alto processamento. Os kernel's em OpenCL já estão prontos e os de CUDA estão em andamento.

Kernel Memory bound

Para comparar a cópia de memória, usamos um kernel que usa a GPU para copiar uma matriz de double em outra.

Kernel Computing bound

Para comparar o processamento, usamos um kernel que usa a GPU multiplicar duas matrizes de doubles. Obtemos o tempo real do processamento tirando o tempo de execução do kernel memory bound do tempo desse kernel.