

# Comparação de eficiência entre OpenCL e CUDA

Thiago de Gouveia Nunes

11 de novembro de 2012

# GPGPU

O que é GPGPU?

# GPGPU

Existem 2 linguagens populares no mercado para GPGPU, o **CUDA** (Compute Unified Device Architecture) feita pela *NVIDIA*, e o **OpenCL** (Open Computing Language), iniciativa open source de um conjunto de empresas.

# OpenCL



O OpenCL é uma linguagem de programação paralela para sistemas híbridos. Atualmente o OpenCL está na versão 1.2.

# CUDA



CUDA é uma linguagem proprietária para programação paralela em GPUs desenvolvida pela NVIDIA. O CUDA está na versão 5.0 atualmente.

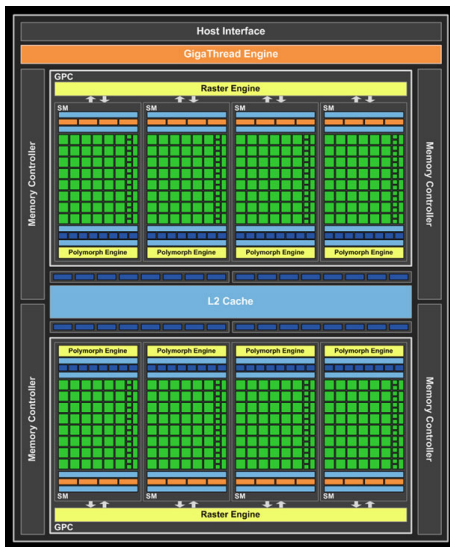
# Como comparar?

Bem, para comparar as linguagens, vamos observar 2 aspectos delas:

1. Taxa de acesso a memória;
2. Método de execução.

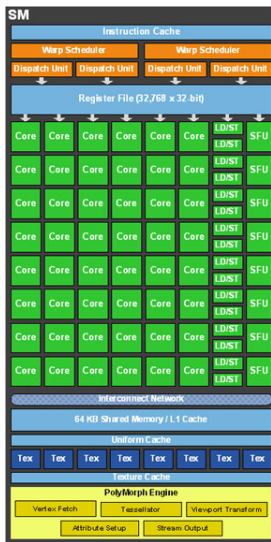
Para entender o impacto desses aspectos no tempo de execução, vamos dar uma olhada no hardware de uma GPU...

# GPU



Representação do hardware de uma GTX 460 SE da NVIDIA.

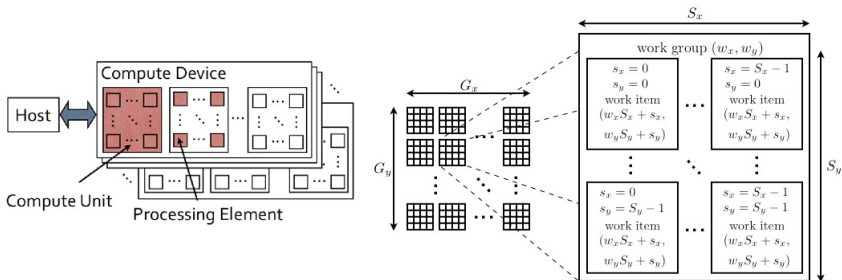
# Arquitetura GeForce GTX 460 SE



A GPU é subdividida em vários Streaming Multiprocessor, como o do lado, que agrupam 48 processadores.



# Como um programa roda na GPU?



# Semelhanças

Alguns elementos são iguais para as duas linguagens.

- ▶ Para iniciar a execução num device é necessário que um programa chamado de host inicie o ambiente de execução na GPU.
- ▶ As threads executando no device são identificadas por índices.
- ▶ As threads são agrupadas em conjuntos antes de serem enviadas para execução no device.
- ▶ A alocação e preenchimento da memória no device é controlada pelo host.
- ▶ A execução dos kernels pode ser síncrona ou assíncrona com o a execução do host.

# Semelhanças

- ▶ Existem 4 locais diferentes para a memória que é enviada para o device:
  1. Global Memory - Toda e qualquer thread tem acesso a essa memória.
  2. Constat Memory - Memória que permanece fixa ao andar da execução.
  3. Local Memory - Região da memória dividida pelas threads de um mesmo SM.
  4. Private Memory - Região privada para cada thread.

# Diferenças - Plataforma

No OpenCL existem 2 tipos de execução diferentes:

1. Data Parallel
2. Task Parallel

O CUDA implementa o modelo SIMT (*Single Instruction, Multiple Thread*).

# Ideia

Para comparar a performance das duas linguagens foram usados dois tipos de kernel, um em que o desempenho está ligado ao acesso a memória (memory bound) e outro que está ligado à velocidade de processamento (compute bound).

# Kernel Memory bound

Para comparar o acesso a memória, foi usado um kernel que faz a cópia de uma matriz de floats para outras.

# Kernel Memory Bound

```
1  __kernel void matrixmulti(__global float* a,  
2      __global float* b,  
3      __global int* rowSize,  
4      __global int* columnSize) {  
5  
6      unsigned int row = get_global_id(0);  
7      unsigned int column = get_global_id(1);  
8  
9      b[row+column*( *rowSize )]  
10         = a [row+column*( *rowSize )];  
11 }
```

# Kernel Compute bound

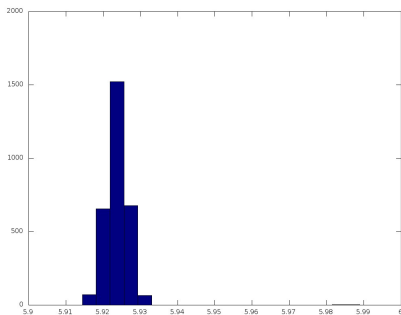
Para comparar o processamento, um kernel que multiplica duas matrizes de floats e guarda o valor numa terceira foi usado.



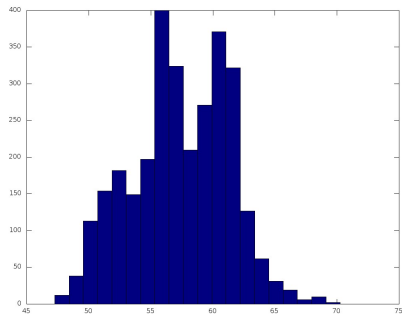
# Kernel Compute bound

```
1  __kernel void matrixmulti(__global int* a,  
3                               __global int* b,  
5                               __global int* c,  
6                               __global int* size) {  
7      unsigned row = get_global_id(0);  
8      unsigned column = get_global_id(1);  
9      unsigned i;  
10     row *= (*size);  
11     c[row+column] = 0;  
12     for( i = 0; i < (*size); i++ )  
13         c[row+column] +=  
14         a[row+i]*b[i*(*size)+column];  
15 }
```

# Estatística Memory bound

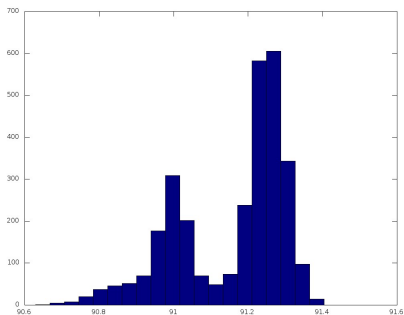


CUDA

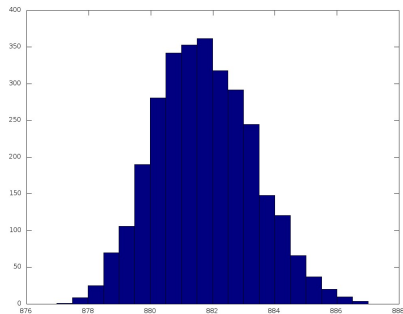


OpenCL

# Estatística Process bound



CUDA



OpenCL

# Explicação dos PTX

Para melhorar a compatibilidade dos programas rodando em GPUs diferentes, a NVIDIA implementou uma máquina virtual, a Parallel Thread Execution (PTX).

# Arquivo .ptx

## Kernel Memory Bound do CUDA Compilado para PTX

```
.reg .f32      %f<2>;
.reg .s32      %r<13>;
.reg .s64      %r1<8>;

ld.param.u64
    %r11, [_Z10MatrixCopyPfS_ii_param_0];
ld.param.u64
    %r12, [_Z10MatrixCopyPfS_ii_param_1];
ld.param.u32
    %r1, [_Z10MatrixCopyPfS_ii_param_3];
cvta.to.global.u64    %r13, %r12;
mov.u32      %r2, %ntid.x;
mov.u32      %r3, %ctaid.x;
mov.u32      %r4, %tid.x;
mad.lo.s32   %r5, %r2, %r3, %r4;
mov.u32      %r6, %ntid.y;
mov.u32      %r7, %ctaid.y;
mov.u32      %r8, %tid.y;
mad.lo.s32   %r9, %r6, %r7, %r8;
mad.lo.s32   %r10, %r5, %r1, %r9;
cvta.to.global.u64    %r14, %r11;
mul.wide.s32 %r15, %r10, 4;
add.s64      %r16, %r14, %r15;
add.s64      %r17, %r13, %r15;
ld.global.f32 %f1, [%r16];
st.global.f32 [%r17], %f1;
ret;
```

## Kernel Memory Bound do OpenCL Compilado para PTX

```
.reg .f32      %f<2>;
.reg .s32      %r<21>;

ld.param.u32
    %r9, [matrixmulti_param_0];
ld.param.u32
    %r10, [matrixmulti_param_1];
ld.param.u32
    %r11, [matrixmulti_param_2];
mov.u32      %r1, %envreg3;
mov.u32      %r2, %ntid.x;
mov.u32      %r3, %ctaid.x;
mov.u32      %r4, %tid.x;
add.s32      %r12, %r4, %r1;
mad.lo.s32   %r13, %r3, %r2, %r12;
mov.u32      %r5, %envreg4;
mov.u32      %r6, %ntid.y;
mov.u32      %r7, %ctaid.y;
mov.u32      %r8, %tid.y;
add.s32      %r14, %r8, %r5;
mad.lo.s32   %r15, %r7, %r6, %r14;
ld.global.u32 %r16, [%r11];
mad.lo.s32   %r17, %r15, %r16, %r13;
shl.b32      %r18, %r17, 2;
add.s32      %r19, %r9, %r18;
add.s32      %r20, %r10, %r18;
ld.global.f32 %f1, [%r19];
st.global.f32 [%r20], %f1;
ret;
```

# Comparação dos PTX

Pelos .ptx é possível verificar algumas das diferenças entre as abstrações das linguagens:

1. O OpenCL usa um registrador a mais que o CUDA para calcular o índice de uma thread.
2. O OpenCL faz mais leituras da memória padrão que o CUDA.
3. O OpenCL não tem acesso a todos os comandos do PTX.

# Conclusões

O CUDA é mais rápido que o OpenCL em GPUs NVIDIA...