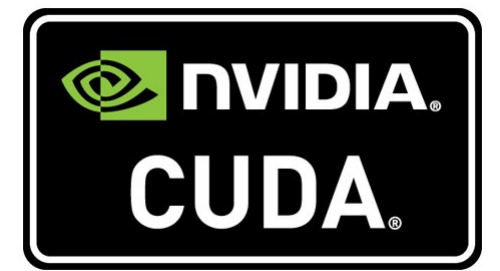
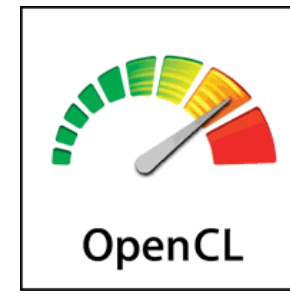




# Comparação de Desempenho entre OpenCL e CUDA

Thiago de Gouveia Nunes  
Orientador: Prof. Marcel P. Jackowski



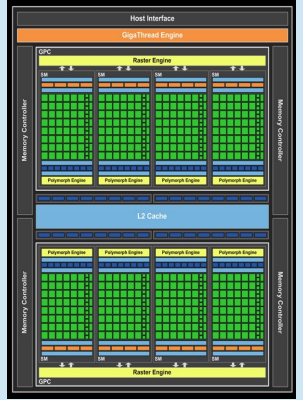
## Introdução

GPGPU ( General-purpose computing on graphics processing units ) é utilizar uma GPU (Graphic Processing Unit) ao invés de uma CPU (Central Processing Unit) para realizar as computações de uma aplicação.

Duas linguagens são muito utilizadas no mercado para programação GPGPU. São elas:

- CUDA (Compute Unified Device Architecture), desenvolvida pela NVIDIA. Hoje está na versão 5.0;
- OpenCL (Open Computing Language), uma alternativa open source desenvolvida em conjunto por várias empresas. Hoje está na versão 1.2.

As GPUs seguem o paradigma de execução SIMD (Single Instruction, Multiple Data), onde cada thread na GPU executa o mesmo código, mas com dados diferentes. Chamamos de Kernel o código que será executado na GPU.



Representação do hardware de uma GPU GTX 460 SE.

GPGPU nasceu do hardware diferenciado da GPU, que permite que problemas que são altamente paralelizáveis sejam executados rapidamente.



Representação de uma unidade de processamento da GTX 460 SE.

## Objetivos

- Comparar o desempenho das duas linguagens
- Comparar as abstrações que cada linguagem usa para mapear a memória e processamento da GPU.
- Comparar os arquivos gerados pela compilação de kernels escritos nessas linguagens.

## Metodologia

Para qualificar as duas linguagens, foram montados 2 tipos de kernels.

- O primeiro é Memory Bound, ou seja, seu tempo de execução é ditado pela velocidade de acesso aos dados. Logo ele foi usado para verificar a velocidade de acesso a memória das linguagens. Esse kernel copia uma matriz 4096x4096 de floats para outra.
- O segundo kernel é Compute Bound, ou seja, seu tempo de execução é determinado unicamente pela velocidade de processamento. Ele é usado para avaliar o método de execução das duas linguagens. Esse kernel multiplica duas matrizes 1024x1024 de floats e depois guarda o valor numa terceira.

## Comparação das abstrações

As duas linguagens são sintaticamente parecidas com o C, mas cada uma delas tem extensões bem diferentes. Enquanto o CUDA otimiza suas funcionalidades para GPUs, o OpenCL é mais genérico, por se tratar de uma linguagem para sistemas baseados tanto em GPU como CPU. Abaixo temos dois kernels de exemplo, os Compute Bound.

### Kernel Compute Bound em OpenCL

```

__kernel void matrixmulti(__global float* MatrixA,
__global float* MatrixB, __global float* MatrixC,
__global int* N)
{
    unsigned i = get_global_id(0);
    unsigned j = get_global_id(1);
    unsigned k;
    MatrixC[i*(*N)+j] = 0;
    for (k = 0; k < (*N); k++)
        MatrixC[i*(*N)+j] +=
            MatrixA[i*(*N)+k]*MatrixB[j+k*(*N)];
}

```

### Kernel Compute Bound em CUDA

```

__global__ void MatrixMult (float* MatrixA, float* MatrixB, float* MatrixC, int N) {
    int j = blockIdx.x*blockDim.x+threadIdx.x;
    int i = blockIdx.y*blockDim.y+threadIdx.y;
    int k;
    MatrixC[i*N+j] = 0;
    for (k = 0; k < N; k++)
        MatrixC[i*N+j] += MatrixA[i*N+k]*MatrixB[k*N+j];
}

```

## Comparação dos PTX

PTX (Parallel Thread Execution) é uma máquina virtual que fica entre a aplicação e as GPUs NVIDIA. Ao se compilar um kernel ele é primeiramente convertido para a linguagem PTX, para só depois executar na VM. Abaixo um exemplo de código PTX.

### Kernel Memory Bound do CUDA Compilado para PTX

```

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

ld.param.u64
    %r1, [_Z10MatrixCopyPfs_ii_param_0];
ld.param.u64
    %r2, [_Z10MatrixCopyPfs_ii_param_1];
ld.param.u32
    %r1, [_Z10MatrixCopyPfs_ii_param_3];
cvta.to.global.u64
    %r3, %r2;
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;
mad.lo.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;

```

## Conclusão

O CUDA executou, em média, 15 vezes mais rápido que o OpenCL.

A causa mais evidente é o PTX. Como o CUDA foi desenvolvido para executar em GPUs NVIDIA, ele otimiza a sua passagem para o PTX, enquanto o OpenCL, uma linguagem para sistemas híbridos, vê essa passagem para o PTX como um nível adicional entre o software e o hardware, fazendo a passagem com perda de desempenho.

Outra causa é o driver usado pelo OpenCL para se comunicar com as GPUs NVIDIA. Hoje, o OpenCL, na sua versão 1.2, é obrigado a usar um driver feito a 3 anos atrás, quando ele estava na versão 1.0.

Essas diferenças são visíveis nos arquivos .ptx. Além do CUDA usar um conjunto de comandos mais avançados e eficientes, ele otimiza seus .ptx de uma maneira melhor que o OpenCL.

Por exemplo, no .ptx resultante da compilação dos kernels Compute Bound, o CUDA faz somente um acesso a memória global por loop, enquanto o OpenCL faz 4.

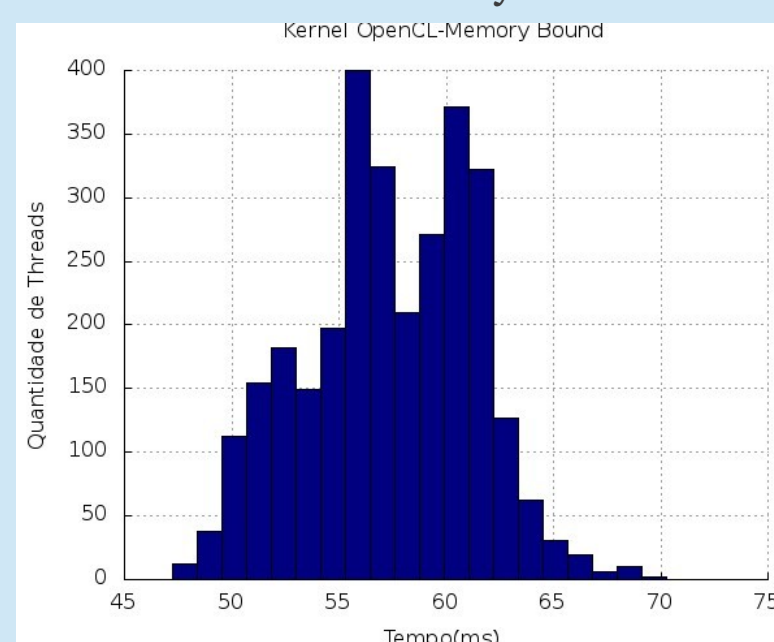
Agora imagine essa diferença quando temos dezenas de threads executando em paralelo tentando acessar a mesma posição de memória, um cenário comum em GPGPU. A perda em desempenho é enorme.

## Comparação de desempenho

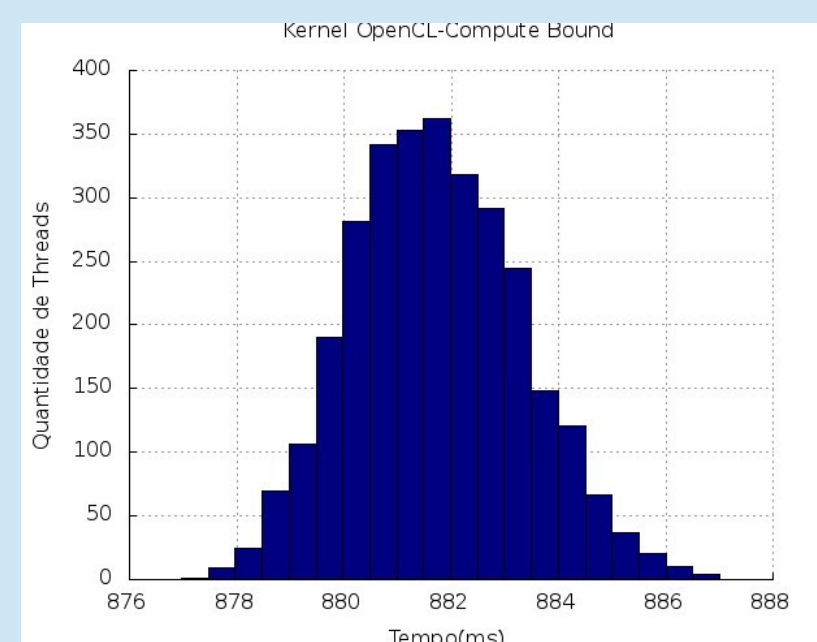
Para colher resultados dos testes cada um dos kernels foi executado 3.000 vezes. A GPU usada para os testes foi uma GTX 460 SE.

Os histogramas abaixo mostram o tempo de execução dos kernels, em milissegundos:

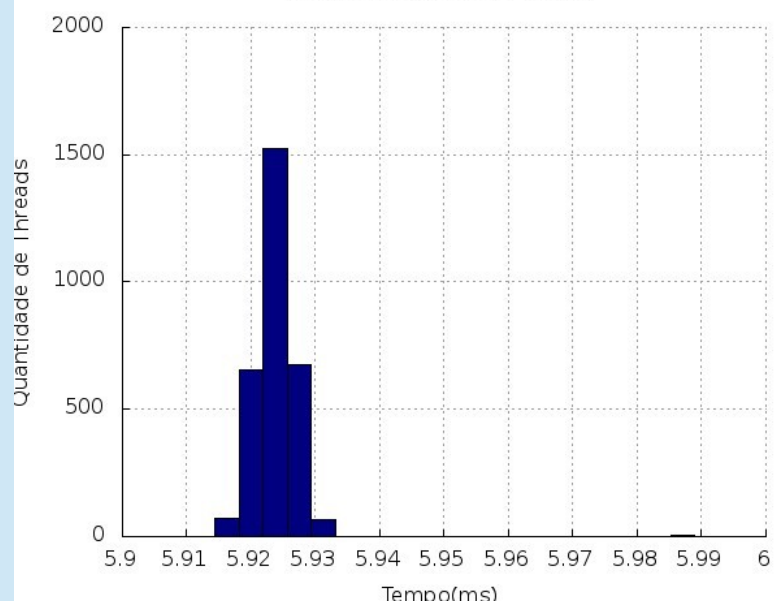
### Kernels Memory Bound



### Kernels Compute Bound



### Kernel CUDA-Memory Bound



### Kernel CUDA-Compute Bound

