Comparação de Desempenho entre OpenCL e CUDA

IME - Instituto de Matemática e Estatística

Thiago de Gouveia Nunes

Orientador: Prof. Marcel P. Jackowski

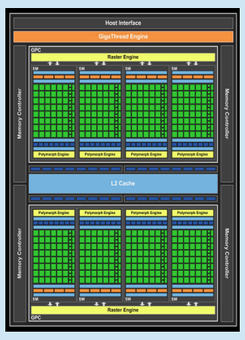
Introdução

GPGPU (General-purpose computing on graphics processing units) é utilizar umas 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:

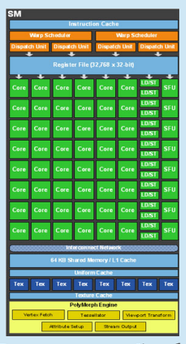
- CUDA (Compute Unified Device Architecture), desenvolvida pela NVIDIA;
- OpenCL (Open Computing Language), uma alternativa open source desenvolvida em conjunto por várias empresas.

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 um Streaming Multiprocessor

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       %rl<8>;

ld.param.u64
        %rl1, [_Z10MatrixCopyPfS_ii_param_0];
ld.param.u64
        %rl2, [_Z10MatrixCopyPfS_ii_param_1];
ld.param.u32
        %r1, [_Z10MatrixCopyPfS_ii_param_3];
cvta.to.global.u64      %rl3, %rl2;
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      %rl4, %rl1;
mul.wide.s32    %rl5, %r10, 4;
add.s64         %rl6, %rl4, %rl5;
add.s64         %rl7, %rl3, %rl5;
ld.global.f32   %f1, [%rl6];
st.global.f32   [%rl7], %f1;
ret;
```

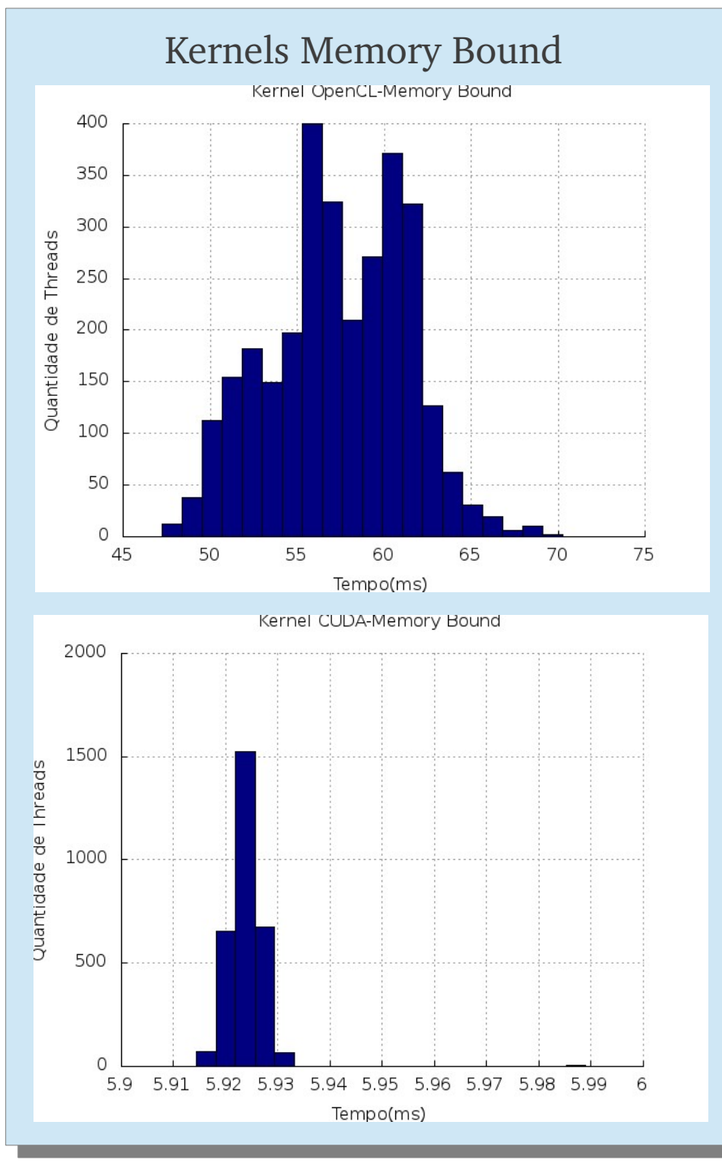
Kernel Memory Bound do OpenCL Compilado para PTX0

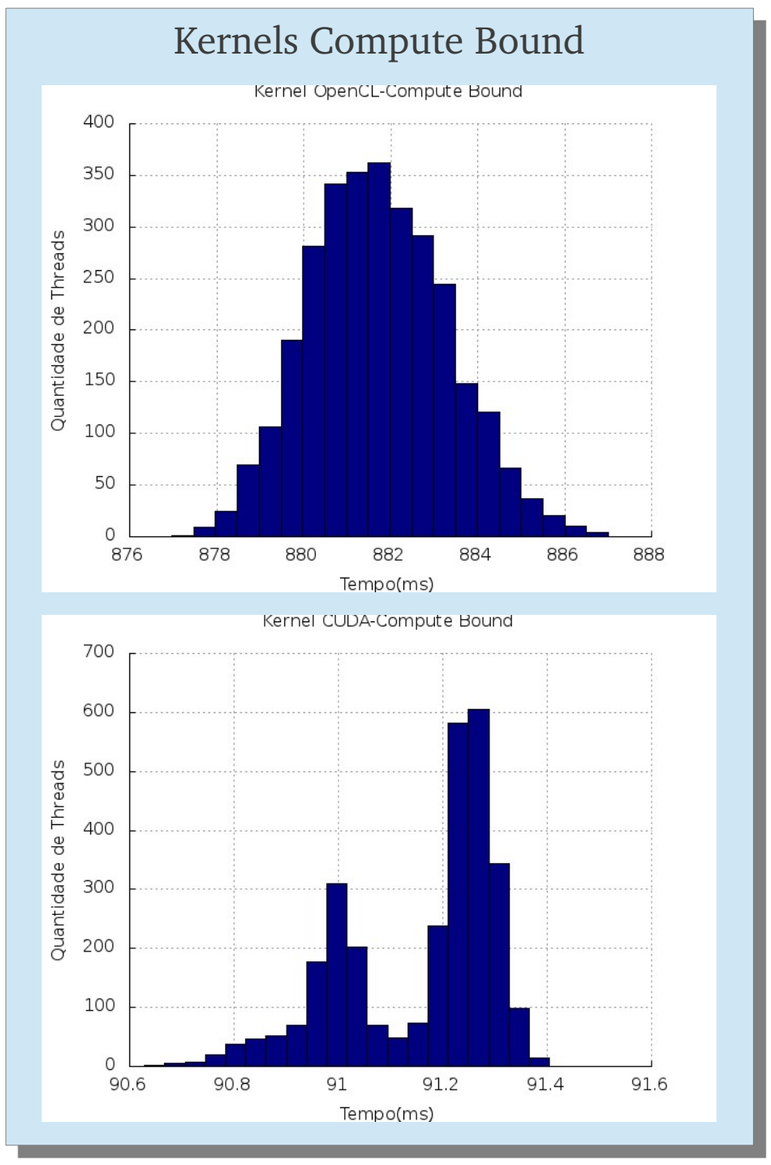
```
.reg .f32       %f<2>;
.reg .s32       %r<21>;

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 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:





Conclusão

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

Uma das causas da diferença de desempenho entre as linguagens é o paradigma usado por elas ao abstrair a execução das threads na GPU.

O OpenCL foi desenvolvido para rodar em sistema híbridos, em que tanto GPUs e CPUs são usados, e por essa abrangência ele paga com desempenho. Ele não reconhece várias funcionalidades que as placas NVIDIA oferecem, enquanto o CUDA tem acesso a essas funcionalidades.

A outra causa está na compilação das linguagens para a máquina virtual das GPUs NVIDIA, a PTX (Parallel Thread Execution). Essa máquina virtual é usada para garantir a compatibilidade de um mesmo programa para GPUs diferentes.

É possível compilar os kernels para um arquivo .ptx, que depois será executado nessa máquina virtual.

Ao comparar esses dois arquivos, notou-se que os kernels do CUDA fazem menos acesso a memória global da GPU do que os do OpenCL. Isso faz uma grande diferença, já que essa memória deve ser acessada por todas as threads executando dentro da GPU, gerando um acesso sequêncial para esse acesso, que acaba diminuindo muito a velocidade de execução das threads.