Programação introdutória em OpenCL e aplicações em realidade virtual e aumentada

César Leonardo Blum Silveira
Prof. Dr. Luiz Gonzaga da Silveira Jr.

V3D

24 de maio de 2010

Soluções em visualização e visão computacional

contato@v3d.com.br

Introdução

Motivação

Breve ilustração

Arquitetura OpenCL

Linguagem OpenCL C

API OpenCL

OpenCL na prática

Exemplos em Realidade Virtual e Aumentada

Otimização com OpenCL

Considerações finais

Introdução

- OpenCL™(*Open Computing Language*) é um padrão aberto para programação em ambientes computacionais heterogêneos
  - Escrita de código para execução em CPUs, GPUs, DSPs e outros dispositivos
- Criado pela Apple, entregue ao Khronos Group para padronização
- Mantido por conjunto de empresas da área
- Aberto, *royalty-free*

Introdução

- Padrão descreve
  - Arquitetura
  - Camada de plataforma
  - *Runtime*
  - Linguagem
  - API

Introdução

- Objetivos deste mini-curso
  - Apresentar a arquitetura OpenCL
  - Introduzir conceitos de programação em OpenCL C
  - Exemplificar aplicações em Realidade Virtual e Aumentada
  - Discutir aspectos de otimização

Motivação

- CPUs
  - Ganho de desempenho com o aumento do clock
  - Limite atingido
  - Solução: adicionar mais *cores*
    - Ganho de desempenho deixou de ser “gratuito”
    - Técnicas e ferramentas para programação paralela

Motivação

- GPUs
  - Processadores paralelos no pipeline gráfico
  - *Shaders*
  - *Toolkits* de acesso aos recursos de processamento
    - CUDA (NVIDIA)
    - Streaming SDK (ATI)

Motivação

- Problemas
  - CPUs e GPUs programadas de formas distintas
  - Acesso ao hardware gráfico diferente para cada fabricante
- Solução: OpenCL
  - Programação baseada na escrita de *kernels*
    - Código a ser executado por unidade da solução
  - Distribuição das tarefas é abstraída

Breve ilustração

Código sequencial

```
void ArrayDiff(const int* a,
               const int* b,
               int* c,
               int n)
{
    for (int i = 0; i < n; ++i)
    {
        c[i] = a[i] - b[i];
    }
}
```

Breve ilustração

Kernel OpenCL

```
__kernel void ArrayDiff(__global const int* a,
                        __global const int* b,
                        __global int* c)
{
    int id = get_global_id(0);
    c[id] = a[id] - b[id];
}
```

Arquitetura OpenCL

- Abstração de baixo-nível do hardware
- Diversos conceitos
  - Inspirados no hardware gráfico
- Camada de plataforma e *runtime*

Arquitetura OpenCL

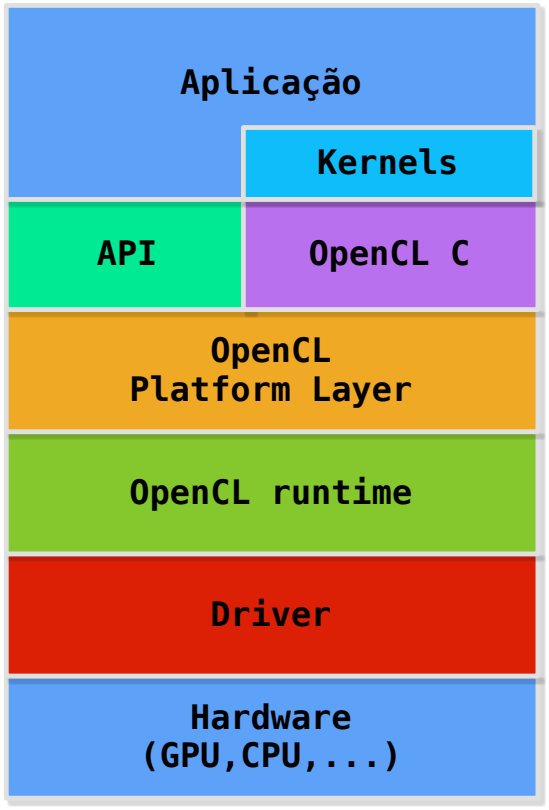


Figura: Camadas de uma aplicação OpenCL

Arquitetura OpenCL

Modelo de plataforma

- *Device*
  - Dispositivo que suporta OpenCL
  - CPU, GPU, DSP, etc.
  - Possui uma ou mais *compute units*
    - Possuem um ou mais *processing elements*

Arquitetura OpenCL

Modelo de plataforma

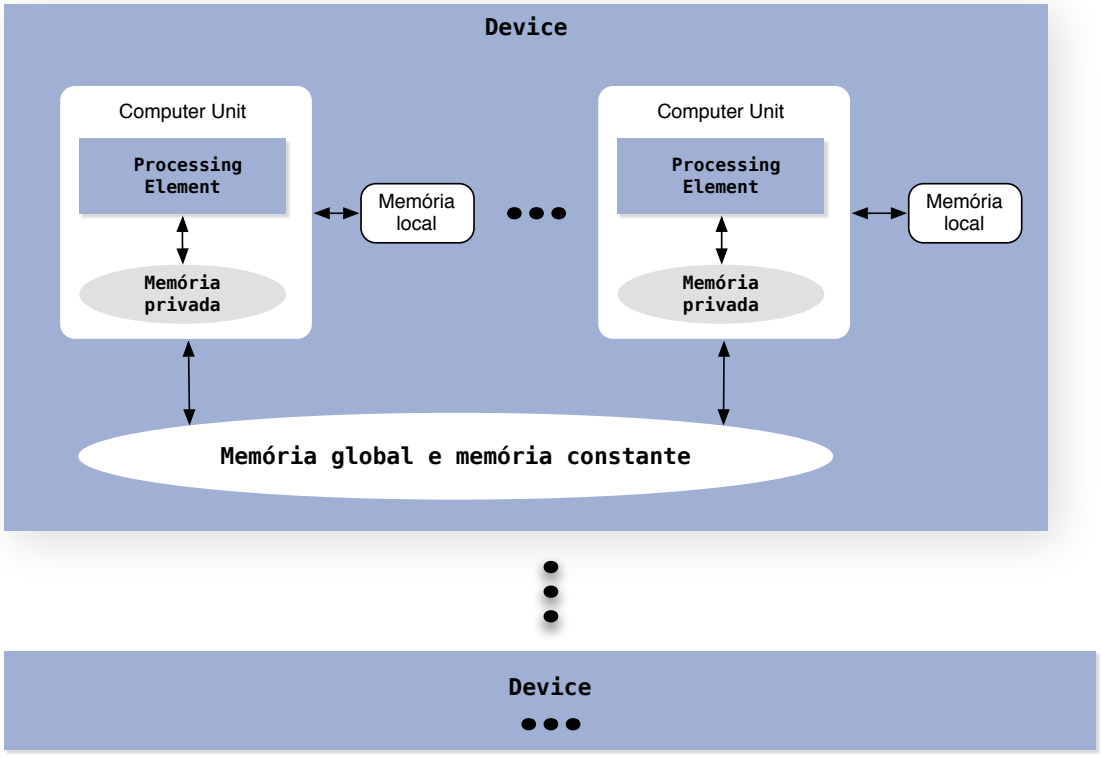


Figura: Modelos de plataforma e de memória da arquitetura OpenCL

Arquitetura OpenCL

Modelo de plataforma

- *Host*
  - Sistema que agrega vários *devices*
  - Coordena
    - Inicialização
    - Compilação de *kernels*
    - Distribuição da computação
    - Recuperação de resultados

Arquitetura OpenCL

Kernel

- Também *kernel function* ou *compute kernel*
- Instruções a serem executadas em um *device*
- Compilados a partir de programas OpenCL C

Arquitura OpenCL

Modelo de execução

- *Work-item*
  - Instância de um *kernel*
  - Associado a *processing element*
- *Work-group*
  - Grupo de *work-items*
  - Associado a *compute unit*

Arquitura OpenCL

Modelo de execução

- NDRange (*N-dimensional range*)
  - Espaço de índices
    - Até 3 dimensões
  - *Work-items* recebem índices globais e locais em cada dimensão
  - *Work-groups* também recebem um índice em cada dimensão

Arquitetura OpenCL

Modelo de execução

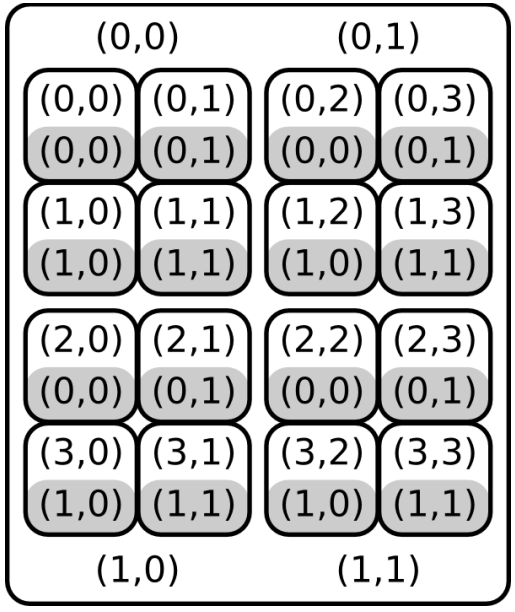


Figura: Exemplo de espaço de índices de duas dimensões

Arquitetura OpenCL

Modelo de programação

- *Data parallel*: um *kernel*, múltiplos *work-items*
- *Task-parallel*: um kernel, um *work-item*

Arquitetura OpenCL

Program object

- Encapsula o código de um ou mais *kernels* e outras funções
- Criado a partir do código-fonte ou binário pré-compilado

Arquitetura OpenCL

Memory object

- Alocação e acesso à memória dos *devices*
- Entrada e saída de dados em *kernels*
- Duas categorias: *buffers* e *images*

Arquitetura OpenCL

Buffer object

- Armazenamento sequencial
- Qualquer tipo de dado
- Acessível por ponteiro

Arquitetura OpenCL

Image object

- Textura
- Ponto-flutuante ou inteiro vetorial
- Acessível através de *samplers*
- Não suportado por todas as implementações

Arquitetura OpenCL

Command queue

- Fila de comandos para um *device*
  - I/O, execução de *kernels*, sincronização
- Comandos iniciados em ordem
- Término fora de ordem
  - *Event objects* permitem aguardar pelo término de comandos específicos

Arquitetura OpenCL

Context

- Ambiente de execução para *kernels*
- Gerencia *devices*, *program objects*, *kernels*, *memory objects* e *command queues*

Arquitetura OpenCL

Modelo de memória

- Memória global
  - Compartilhada por todos os *work-items*
  - Escrita e leitura
- Memória local
  - Compartilhada em um *work-group*
  - Escrita e leitura

Arquitetura OpenCL

Modelo de memória

- Memória privada
  - Restrita a cada *work-item*
  - Escrita e leitura
- Memória constante
  - Compartilhada por todos os *work-items*
  - Somente leitura

Arquitetura OpenCL

Modelo de memória

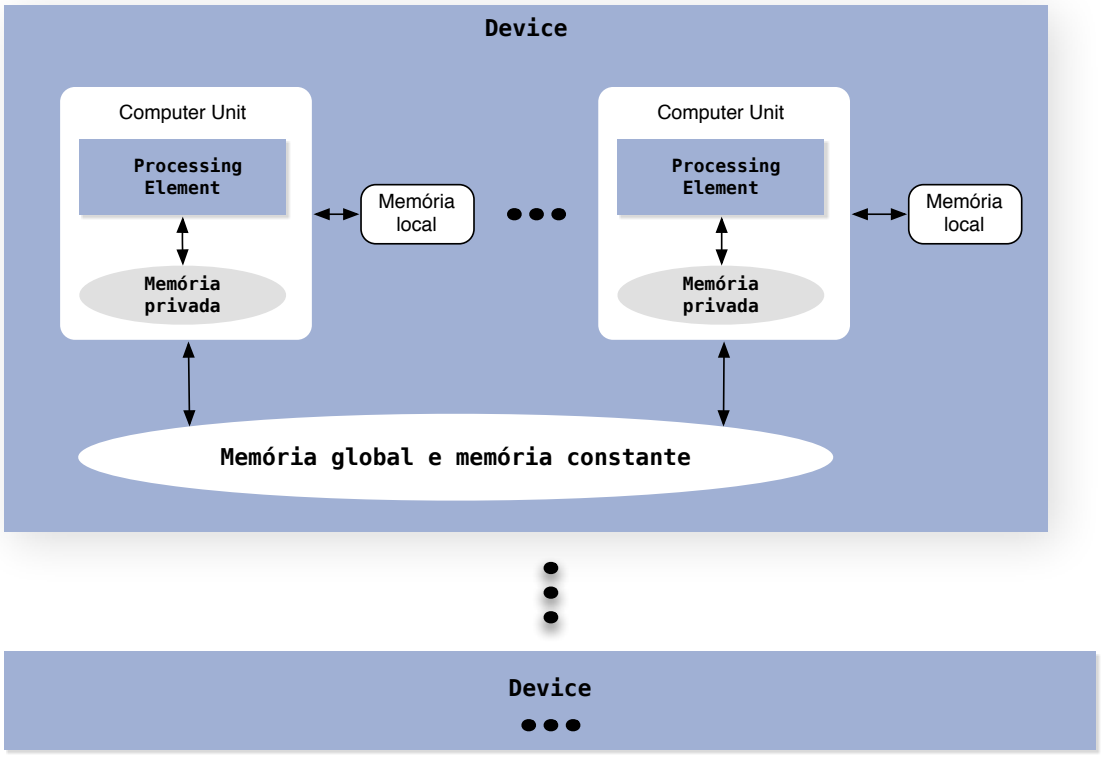


Figura: Modelos de plataforma e de memória da arquitetura OpenCL

Arquitetura OpenCL

Modelo de memória

- Desempenho
  - Na prática, latências variam para cada categoria
  - Dependente de fabricante

Arquitetura OpenCL

Modelo de memória

- Consistência de memória
  - Garantia da leitura correta da memória após escrita
  - *Work-item*: consistência de escrita e leitura
    - Lê último valor escrito
  - *Work-group*: consistência após sincronização
  - Não há sincronização entre *work-groups*
    - Impossível garantir consistência

Linguagem OpenCL C

- Baseada na especificação ISO C99
- Extensões e restrições

Linguagem OpenCL C

Tipos de dados escalares

- Vazio: `void`
- Booleano: `bool`
- Inteiros: [u]`char`, [u]`short`, [u]`int`, [u]`long`
- Ponto-flutuante: `float`, `half`
- Outros: `size_t`, `image2d_t`, `image3d_t`, `sampler_t`, `event_t`

Linguagem OpenCL C

Tipos de dados vetoriais

- Representação de vetores
- `<tipo escalar><# de componentes>`
  - Exemplos: `int2`, `float4`, `long8`
  - Não são permitidos: `bool`, `half`, `size_t` e `void`

Linguagem OpenCL C

Tipos de dados vetoriais

- Permitem 1, 2, 4, 8 ou 16 componentes
- Acesso: `x`, `y`, `z`, `w`
- Ou: `<variável>.s[0-f]`*
  - Exemplos: `vec.s05af`, `vec.s0123`, `vec.s22`, `vec.se0`
- Atribuições entre vetores de dimensões diferentes

```
float2 v = (float2)(1.0f,2.0f);
float4 u = v.s0011;
float4 s = (float4)(10.0f,20.0f,30.0f,40.0f);
float2 t = s.xy;
```

Linguagem OpenCL C

Tipos de dados definidos pelo usuário

- `struct`
- `union`

Linguagem OpenCL C

Qualificador `__kernel`

- Definição de kernels

```
__kernel void f(...)
{
    ...
}
```

Linguagem OpenCL C

Qualificadores de espaço de endereçamento

- `__global`, `__local`, `__private` e `__constant`
- `__private` assumido por *default*
- Exemplos
  - `__global int* arr`
  - `__global const float* data`
  - `__local uchar* ldata`

Linguagem OpenCL C

Qualificadores de espaço de endereçamento - Restrições

- Argumentos ponteiros de um *kernel* devem ser `__global`, `__local` ou `__const`
  - Legal

```
__kernel void f(__global int* in,
                __global int* out) {
    ...
}
```

  - Ilegal

```
__kernel void f(int* in,
                int* out) {
    ...
}
```

Linguagem OpenCL C

Qualificadores de espaço de endereçamento - Restrições

- Somente são permitidas atribuições entre ponteiros de um mesmo espaço de endereçamento
  - Legal

```
__kernel void f(__global int* in,
                __global int* out) {
    __global int* pIn = in;
    ...
}
```

  - Ilegal

```
__kernel void f(__global int* in,
                __global int* out) {
    __local int* pIn = in;
    ...
}
```

Linguagem OpenCL C

Operadores

- Mesmos operadores da linguagem C99
- Aritméticos binários: `+`, `-`, `*`, `/`, `%`
- Aritméticos unários: `+`, `-`, `++`, `--`
- Relacionais: `>`, `>=`, `<`, `<=`, `==`, `!=`
- Lógicos binários: `&&`, `||`
- Lógicos unários: `!`
- Bit-a-bit: `&`, `|`, `^`, `~`, `<<`, `>>`
- Atribuição: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `>>=`

Linguagem OpenCL C

Operadores - Semântica

- Operações entre escalares e vetores
  - Retornam vetores com operação aplicada sobre cada componente
  - Exemplo

```
// (int4)(3, 4, 5, 6)
int4 v = (int4)(1, 2, 3, 4) + 2;
```

  - Escalar deve ser do mesmo tipo ou de menor tamanho que o vetor

```
// Erro, tipo de n maior que short
long n = 10;
short4 v = (short4)(1, 2, 3, 4) + n;
```

Linguagem OpenCL C

Operadores - Semântica

- Operações unárias sobre vetores
  - Retornam vetores com operação aplicada sobre cada componente
  - Exemplo

```
// (char4)(-1, -2, -3, -4)
char4 v = -(char4)(1, 2, 3, 4);

// (uchar2)(0xff, 0x00)
uchar4 v = ~(uchar4)(0x00, 0xff);
```

Linguagem OpenCL C

Operadores - Semântica

- Operações entre vetores
  - Vetores devem ter o mesmo número de componentes
  - Exemplo

```
float2 v = (float2)(2.0f, 4.0f);
float2 u = (float2)(3.0f, 5.0f);
float4 a = (float4)(1.0f, 1.0f, 1.0f, 1.0f);

// (float2)(6.0f, 20.0f)
float2 t = v * u;

// Erro, vetores de tamanhos diferentes!
float4 r = a + t;
```

Linguagem OpenCL C

Operadores - Semântica

- Operações lógicas e relacionais com operando vetorial
  - Retornam vetor do mesmo tipo com operação aplicada componente-a-componente
  - Resultado 0 é falso, -1 é verdadeiro
  - Exemplo

```
// (int4)(-1, 0, 0, 0)
int4 v = 2 > (int2)(1, 3, 5, 7);

// (short2)(-1, 0)
short2 u = (short2)(1, 10) < (5, 7);
```

Linguagem OpenCL C

Restrições

- Não são permitidos
  - Ponteiros para funções
  - Escrita em ponteiros ou *arrays* cujo tipo tenha tamanho inferior a 32 bits

```
// Escritas invalidas
char arr[10];
arrc[0] = 'a';
char* p = arr;
p[1] = 'b';
```

  - Argumentos para *kernels* que sejam ponteiros para ponteiros

```
// Ilegal
__kernel void f(__global int** a) {
    ...
}
```

Linguagem OpenCL C

Restrições

- Não são suportados
  - Recursão
  - Funções e macros com número variável de argumentos
  - Qualificadores `extern`, `static` e `auto`
  - `struct`s e `union`s com elementos em espaços de endereçamento distintos
    - Mesmo qualificador para todos elementos

API OpenCL

- API de suporte para *kernels*
- Auxílio para o programador

API OpenCL

- Categorias de funções
  - **Funções de identificação**
  - Funções matemáticas
    - Raíz quadrada, potência, trignométricas, arredondamento, produtos vetorial e escalar, mínimo, máximo, etc.
  - Funções utilitárias
    - *Clamping*, conversões
  - Funções de acesso a *image objects*
  - **Funções de Sincronização**

API OpenCL

Funções de identificação

- `size_t get_global_id(uint dimindx)`
- `size_t get_local_id(uint dimindx)`
- `size_t get_group_id(uint dimindx)`
  - Retornam posição no espaço de índices na dimensão informada

API OpenCL

Funções de identificação

- `size_t get_global_size(uint dimindx)`
- `size_t get_local_size(uint dimindx)`
  - Retornam tamanho global e local (*work-group*) da dimensão informada

API OpenCL

Funções de identificação

- `size_t get_num_groups()`
  - Retorna número de *work-groups* em execução
- `uint get_work_dim()`
  - Retorna número de dimensões do espaço de índices

API OpenCL

Sincronização

- `void barrier(cl_mem_fence_flags flags)`
  - Barreira de sincronização
  - Execução só continua após todos *work-items* de um *work-group* atingirem a barreira
  - Barreira garante consistência de memória
  - *Flags*
    - `CLK_LOCAL_MEM_FENCE`
    - `CLK_GLOBAL_MEM_FENCE`

API OpenCL

Sincronização - Exemplo

```
__kernel void aKernel(__global float* in,
                      __global float* out)
{
    // Indice global e local do work-item
    int gid = get_global_id(0);
    int lid = get_local_id(0);
    int lsize = get_local_size(0);

    // Array compartilhado pelo work-group
    __local float a[lsize];
```

API OpenCL

Sincronização - Exemplo

```
    // Carrega dados no array local
    a[lid] = in[gid];

    // Consistencia no array local
    barrier(CLK_LOCAL_MEM_FENCE);

    // Escreve saida na memoria global
    out[gid] = a[(lid + 4) \% lsize] * 2;

    // Consistencia na memoria global
    barrier(CLK_GLOBAL_MEM_FENCE);

    ...
}
```

API OpenCL

Sincronização - Exemplo

- *Work-items* lêem dados escritos por outros *work-items* do mesmo *work-group*
- Sem `barrier`s, valores lidos na sequência podem ser incorretos

OpenCL na prática

- Código do *host*
- Exemplo: diferença entre *arrays*

OpenCL na prática

Cabeçalho

- Mac OS X

```
#include <OpenCL/opencl.h>
```

- Demais sistemas

```
#include <CL/opencl.h>
```

OpenCL na prática

Código-fonte do *kernel*

```
const char* source =
    "__kernel void ArrayDiff( \
        __global const int* a, \
        __global const int* b, \
        __global int* c) \
    { \
        int id = get_global_id(0); \
        c[id] = a[id] - b[id]; \
    }";
```

OpenCL na prática

Obtenção do *device*

```
cl_platform_id platformId;
clGetPlatformIDs(
    1,              // No. de IDs desejados
    &platformId,    // Onde armazenar IDs
    NULL            // No. de IDs retornados
);

cl_device_id deviceId;
clGetDeviceIDs(
    platformId,             // Platform ID
    CL_DEVICE_TYPE_GPU,     // Tipo de dispositivo
    1,                      // No. de IDs desejados
    &deviceId,              // Onde armazenar IDs
    NULL                    // No. de IDs encontrados
);
```

OpenCL na prática

Obtenção do *device* - alternativa

```
cl_device_id deviceId;
clGetDeviceIDs(
    NULL,                   // Platform ID
    CL_DEVICE_TYPE_GPU,     // Tipo de dispositivo
    1,                      // No. de IDs desejados
    &deviceId,              // Onde armazenar IDs
    NULL                    // No. de IDs encontrados
);
```

- Não funciona em todas as implementações!

OpenCL na prática

Criação do *context*

```
cl_context context = clCreateContext(
    0,           // Propriedades do context
    1,           // No. de devices no context
    &deviceId,   // Array de IDs de devices
    NULL,        // Funcao de notificacao
    NULL,        // Dado p/ funcao de notificacao
    NULL,        // Codigo de erro
);
```

OpenCL na prática

Criação da *command queue*

```
cl_command_queue queue = clCreateCommandQueue(
    context,
    deviceId,
    0,          // Propriedades da command queue
    NULL        // Codigo de erro
);
```

OpenCL na prática

Criação do *program object*

```
cl_program program = clCreateProgramWithSource(
    context,
    1,          // No. de strings no array
    &source,    // Array de strings do fonte
    NULL,       // Array de tamanhos de strings
    NULL,       // Codigo de erro
);
```

OpenCL na prática

Compilação do *program object*

```
clBuildProgram(
    program,
    0,    // No. de devices p/ os quais compilar
          // 0 p/ todos devices do context
    NULL, // IDs dos devices; NULL p/ todos
    NULL, // Opcoes do compilador
    NULL, // Funcao de notificacao
    NULL  // Dados p/ funcao de notificacao
);
```

OpenCL na prática

Criação do *kernel*

```
cl_kernel kernel = clCreateKernel(
    program,
    "ArrayDiff", // Nome da funcao __kernel
    NULL         // Codigo de erro
);
```

OpenCL na prática

Criação dos *buffer objects*

```
#define ARRAY_LENGTH 1000000
#define BUFSIZE (ARRAY_LENGTH * sizeof(int))

cl_mem bufA = clCreateBuffer(
    context,
    CL_MEM_READ_WRITE, // Flags
    BUFSIZE,           // Tamanho
    NULL,              // Dados iniciais
    NULL               // Codigo de erro
);
```

- Outras flags
  - `CL_MEM_READ_ONLY`
  - `CL_MEM_WRITE_ONLY`

OpenCL na prática

Criação dos *buffer objects*

```
cl_mem bufB = clCreateBuffer(...);
cl_mem bufC = clCreateBuffer(...);
```

OpenCL na prática

Transferência das entradas

```
int* hostA = ...;
int* hostB = ...;

clEnqueueWriteBuffer(
    queue,
    bufA,
    CL_TRUE, // Bloquear ate terminar
    0,       // Offset
    BUFSIZE, // Tamanho
    hostA,   // Ponteiro para dados no host
    0,       // No. de eventos a aguardar
    NULL,    // Lista de eventos
    NULL     // Evento retornado
);
```

OpenCL na prática

Transferência das entradas

```
clEnqueueWriteBuffer(
        ...
        bufB,
        ...
        hostB,
        ...
);
```

OpenCL na prática

Configuração dos argumentos do *kernel*

```
clSetKernelArg(
    kernel,
    0,                  // Posicao
    sizeof(cl_mem), // Tamanho
    &bufA               // Ponteiro p/ dados no host
);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &bufB);
clSetKernelArg(kernel, 2, sizeof(cl_mem), &bufC);
```

OpenCL na prática

Argumentos para *kernels* - outros tipos

- *Kernel* exemplo

```
__kernel void f(__global int* a,
                int n)
{
    ...
}
```

- Configuração dos argumentos no *host*

```
cl_kernel k;
cl_mem buf;
int n;

clSetKernelArg(k, 0, sizeof(cl_mem), &bufA);
clSetKernelArg(k, 1, sizeof(int), &n);
```

OpenCL na prática

Execução do *kernel*

```
size_t globalSize[1] = { ARRAY_LENGTH };
cl_event event;

clEnqueueNDRangeKernel(
    queue,
    kernel,
    1,              // No. de dimensoes
    NULL,           // Offset indices (nao suportado)
    globalSize,     // Dimensoes do espaco de indices
    NULL,           // Dimensoes dos work-groups
    0,              // No. de eventos a aguardar
    NULL,           // Lista de eventos
    &event          // Evento retornado
);
```

OpenCL na prática

Execução do *kernel*

- Dimensões do *work-group* devem dividir inteiramente dimensões do espaço de índices

OpenCL na prática

Sincronização

```
clWaitForEvents(
    1,      // No. de eventos
    &event // Array de eventos
);
```

OpenCL na prática

Sincronização - alternativa

- Execução do *kernel*

```
clEnqueueNDRangeKernel(
    ...
    NULL // Evento retornado
);
```

- Sincronização

```
// Aguarda termino de todos os comandos
clFinish(queue);
```

OpenCL na prática

Leitura dos resultados

```
clEnqueueReadBuffer(
    queue,
    bufC,
    CL_TRUE, // Bloquear host ate terminar
    0,       // Offset
    BUFSIZE, // Tamanho
    hostC,   // Onde armazenar dados
    0,       // No. de eventos a aguardar
    NULL,    // Eventos
    NULL     // Evento retornado
);
```

OpenCL na prática

Liberação de recursos

```
clReleaseMemObject(bufC);
clReleaseMemObject(bufB);
clReleaseMemObject(bufA);
clReleaseKernel(kernel);
clReleaseProgram(program);
clReleaseCommandQueue(queue);
clReleaseContext(context);
```

Exemplos em Realidade Virtual e Aumentada

- Foco de OpenCL está no desempenho da computação
- Auxílio em
  - Algoritmos de base
  - Simulações
  - Processamento de dados diversos
- *Offload* de parte da computação

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual

- OpenCL interopera com OpenGL
- Processamento de dados para visualização
- Exemplo: animação de malha 3D

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Definir propriedades para um *context* para interoperação

  Linux

```
cl_context_properties props[] = {
  CL_GL_CONTEXT_KHR,
  (cl_context_properties)glXGetCurrentContext(),
  CL_GLX_DISPLAY_KHR,
  (cl_context_properties)glXGetCurrentDisplay(),
  CL_CONTEXT_PLATFORM,
  (cl_context_properties)cpPlatform,
  0
};
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

Windows

```
cl_context_properties props[] = {
  CL_GL_CONTEXT_KHR,
  (cl_context_properties)wglGetCurrentContext(),
  CL_WGL_HDC_KHR,
  (cl_context_properties)wglGetCurrentDC(),
  CL_CONTEXT_PLATFORM,
  (cl_context_properties)cpPlatform,
  0
};
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

Mac OS X

```
CGLContextObj kCGLContext = CGLGetCurrentContext();
CGLShareGroupObj kCGLShareGroup =
  CGLGetShareGroup(kCGLContext);

cl_context_properties props[] = {
  CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
  (cl_context_properties)kCGLShareGroup,
  0
};
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Criar o *context*

```
cl_device_id deviceId;
...
cl_context context = clCreateContext(
    props,      // Propriedades
    1,
    &deviceId,
    NULL,
    NULL,
    NULL
);
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Criar um VBO (*Vertex Buffer Object*) OpenGL

```
// Dimensoes da malha
unsigned int m_width, m_height;

// Tamanho do VBO
unsigned int size =
    m_width * m_height * 4 * sizeof(float);

GLuint vbo;
glGenBuffers(1, vbo);
glBindBuffer(GL_ARRAY_BUFFER, *vbo);
glBufferData(
    GL_ARRAY_BUFFER, size, 0, GL_DYNAMIC_DRAW);
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Criar um *buffer object* OpenCL a partir do VBO

```
cl_mem vbo_cl = clCreateFromGLBuffer(
    context,
    CL_MEM_WRITE_ONLY,
    *vbo,
    NULL
);
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Kernel

```
__kernel void sine_wave(
    __global float4* pos,
    uint width,
    uint height,
    float time)
{
    uint x = get_global_id(0);
    uint y = get_global_id(1);

    float u = x / (float) width;
    float v = y / (float) height;
    u = u*2.0f - 1.0f;
    v = v*2.0f - 1.0f;
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- *Kernel*

```
        float freq = 4.0f;
        float w =
            sin(u*freq + time) *
            cos(v*freq + time) *
            0.5f;

        pos[y*width+x] = (float4)(u, w, v, 1.0f);
    }
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Configurar argumentos fixos do *kernel*

```
clSetKernelArg(
  kernel, 0, sizeof(cl_mem), (void *) &vbo_cl);
clSetKernelArg(
  kernel, 1, sizeof(unsigned int), &m_width);
clSetKernelArg(
  kernel, 2, sizeof(unsigned int), &m_height);
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Loop
  - Atualizar *timestep*
  - Executar *acquire* no VBO
  - Executar *kernel*
  - Executar *release* no VBO

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Atualizar *timestep*

```
float timestep = 0.0f;

// Funcao executada periodicamente pelo host
void updateFunc()
{
    timestep += 0.1f;
    ...
}
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Executar *acquire* no VBO

```
clEnqueueAcquireGLObjects(
    queue,
    1,          // No. de objetos OpenGL
    &vbo_cl, // Array de objetos
    NULL,       // No. de eventos a aguardar
    NULL,       // Array de eventos
    NULL        // Evento retornado
);
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Configurar argumento `time` do *kernel* e executá-lo

```
size_t globalSize[2] = { m_width, m_height };
clSetKernelArg(
    kernel, 3, sizeof(float), &timestep);
clEnqueueNDRangeKernel(
    queue,
    kernel,
    2,
    NULL,
    globalSize,
    NULL,
    NULL,
    NULL,
    NULL
);
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Executar *release* no VBO

```
clEnqueueReleaseGLObjects(
    queue,
    1,          // No. de objetos OpenGL
    &vbo_cl,    // Array de objetos
    NULL,       // No. de eventos a aguardar
    NULL,       // Array de eventos
    NULL        // Evento retornado
);
```

Exemplos em Realidade Virtual e Aumentada

Realidade Virtual - Exemplo: malha 3D

- Demonstração

Exemplos em Realidade Virtual e Aumentada

Realidade Aumentada

- Processamento de imagens e visão computacional
- Algoritmos computacionalmente custosos
- Operações por *pixel*
- Exemplo: detecção de bordas com Laplace

Exemplos em Realidade Virtual e Aumentada

Realidade Aumentada - Detecção de bordas

- Operador de Laplace calcula segunda derivada
- Invariante a rotações

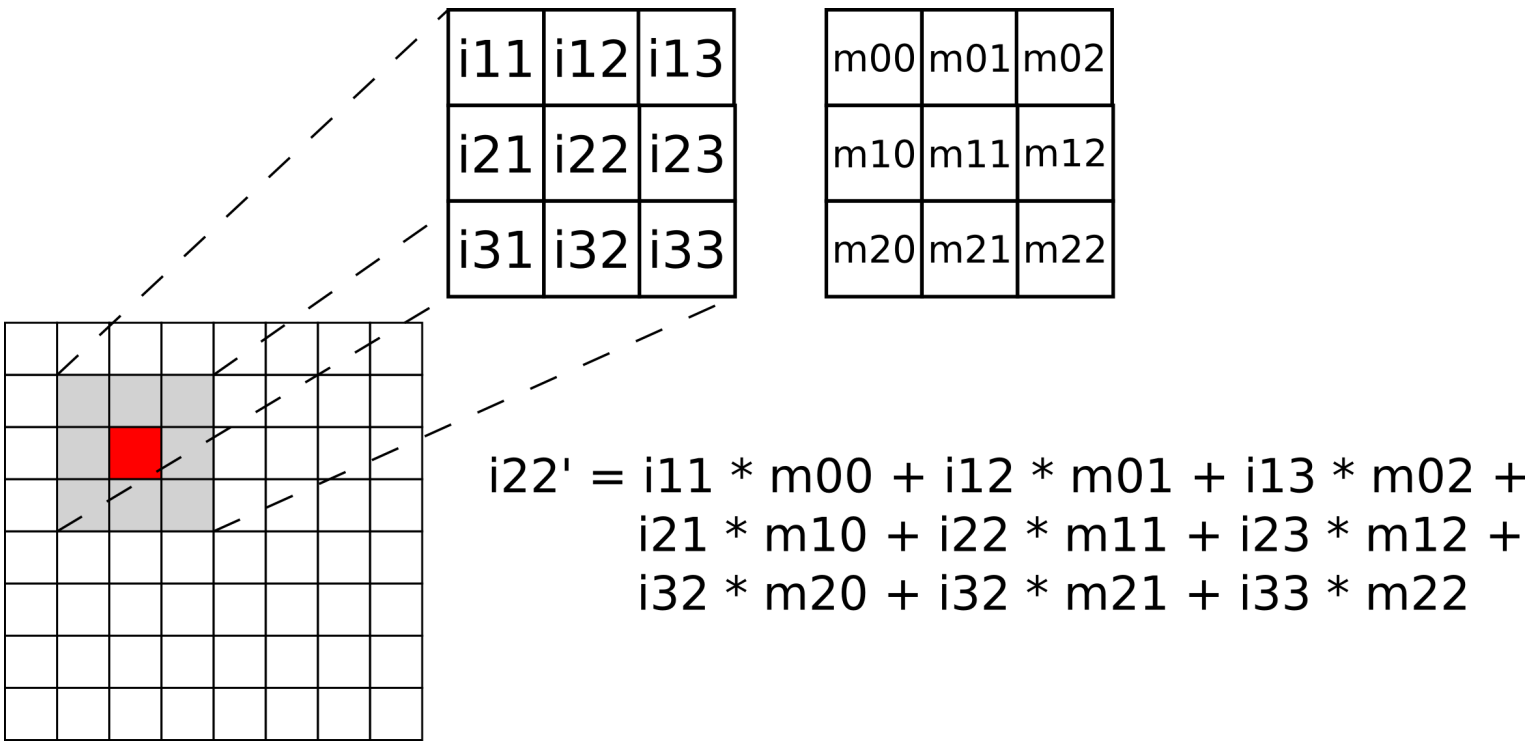
0	1	0
1	-4	1
0	1	0

Figura: Operador de Laplace

Exemplos em Realidade Virtual e Aumentada

Realidade Aumentada - Detecção de bordas

- Convolução



Exemplos em Realidade Virtual e Aumentada

Realidade Aumentada - Detecção de bordas

- *Kernel*

```
__kernel void Laplacian(
    __global const int* image,
    __global int* laplacian)
{
    int x = get_global_id(0);
    int y = get_global_id(1);
    int width = get_global_size(0);
    int height = get_global_size(1);
```

Exemplos em Realidade Virtual e Aumentada

Realidade Aumentada - Detecção de bordas

- *Kernel*

```
    if (x > 0 && x < width - 1 &&
        y > 0 && y < height - 1)
    {
        laplacian[y * width + x] =
            -4 * image[y * width + x] +
            image[(y - 1) * width + x] +
            image[(y + 1) * width + x] +
            image[y * width + (x - 1)] +
            image[y * width + x + 1];
    }
    else laplacian[y * width + x] = 0;
}
```

Exemplos em Realidade Virtual e Aumentada

Realidade Aumentada - Detecção de bordas

- Executar *kernel* em espaço de índices de duas dimensões
- Dimensões iguais às da imagem
- Opera sobre imagem em escala de cinza

Otimização com OpenCL

Identificação de problemas paralelizáveis

- Tarefas independentes
- Resultados independentes na mesma tarefa

Otimização com OpenCL

Identificação de problemas paralelizáveis

- Exemplos
  - Paralelizável

```
for (int i = 0; i < N; ++i)
{
    data[x] = work(x);
}
```

  - Não-paralelizável

```
for (int i = 1; i < N; ++i)
{
    // dependencia
    data[x] = work(data[x - 1], x);
}
```

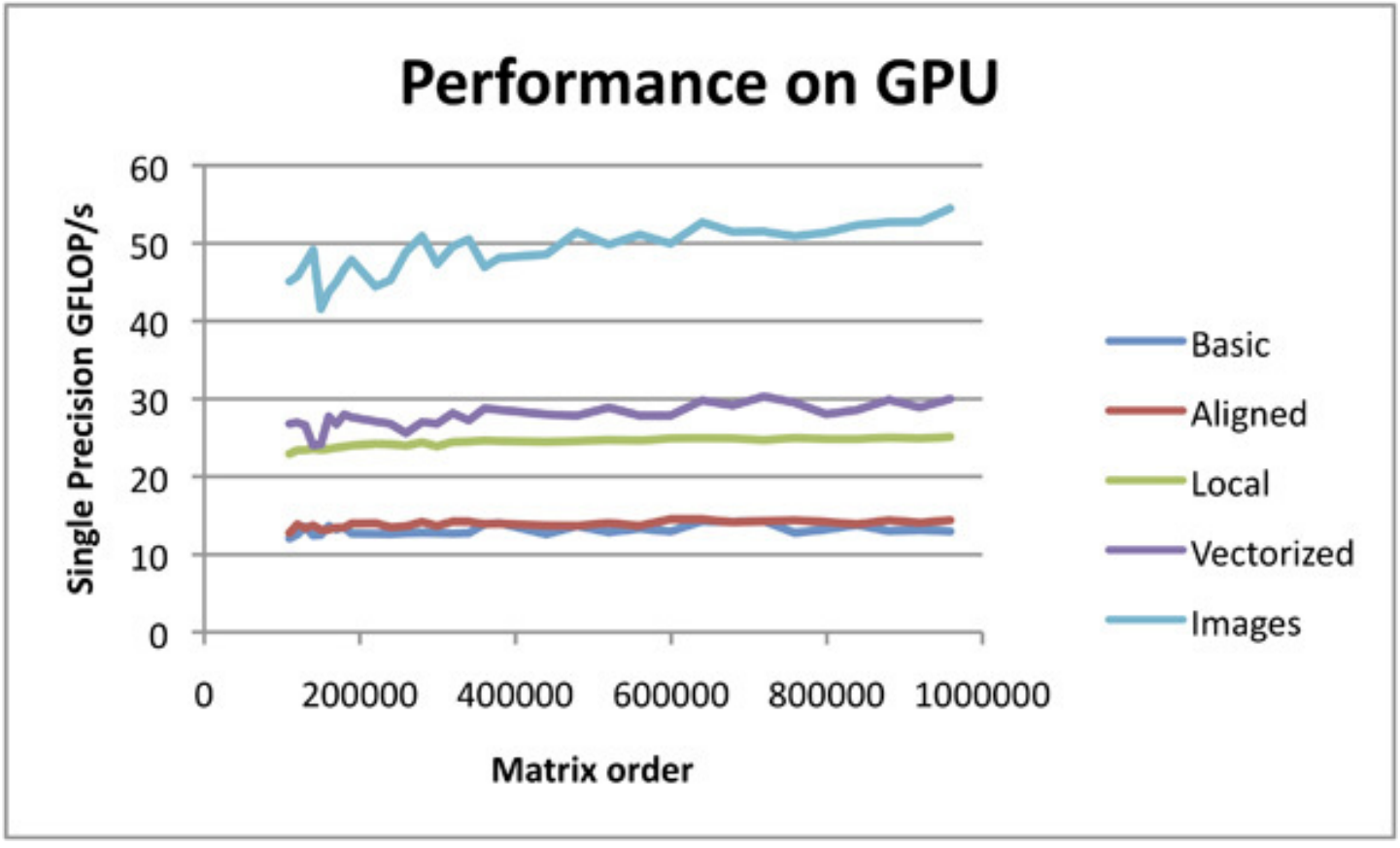
Otimização com OpenCL

Otimização de *kernels*

- Dependente de fabricante
- Alguns aspectos
  - Uso de `__local`
  - Dados `float`
  - Vetores
  - *Image objects*

Otimização com OpenCL

Exemplo de ganho de desempenho



Otimização com OpenCL

Links recomendados

- NVIDIA Developer Zone - OpenCL
  - http://developer.nvidia.com/object/opencl.html
- AMD Developer Central - OpenCL Zone
  - http://developer.amd.com/zones/OpenCLZone/Pages/default.aspx
- Exemplos e guias

Considerações finais

- OpenCL é uma abordagem multiplataforma para processamento paralelo
- Espera-se crescimento da adoção do padrão

Considerações finais

- ATI adotou OpenCL no Streaming SDK
- NVIDIA provavelmente manterá CUDA e OpenCL paralelamente
  - OpenCL roda sobre CUDA
  - Porém CUDA oferece algumas vantagens adicionais
- Dispositivos móveis

Considerações finais

Disponibilização do material

- http://labs.v3d.com.br
- A partir da próxima semana

Considerações finais

Contato com os autores

- cesar@v3d.com.br
- lgonzaga@v3d.com.br

Programação introdutória em OpenCL e aplicações em realidade virtual e aumentada

César Leonardo Blum Silveira
Prof. Dr. Luiz Gonzaga da Silveira Jr.

V3D

24 de maio de 2010