Programação Paralela em Ambientes Computacionais Heterogêneos com OpenCL

César L. B. Silveira
Prof. Dr. Luiz G. da Silveira Jr.
Prof. Dr. Gerson Geraldo H. Cavalheiro

28 de outubro de 2010

www.V3D.com.br

contato@v3d.com.br

cesar@v3d.com.br

Introdução

- Ambientes computacionais atuais: CPUs e GPUs
  - Ambientes **heterogêneos**
- CPUs *multicore* e GPUs *many-core*
  - Processamento paralelo
  - Alto desempenho
- Como utilizar todo este poder computacional?

Introdução

- Programação paralela
  - CPUs
    - *Multithreading*
    - Múltiplos processos
  - GPUs
    - APIs gráficas (OpenGL, DirectX)
    - *Shaders*
    - Tecnologias proprietárias
      - NVIDIA CUDA
      - AMD Streaming SDK

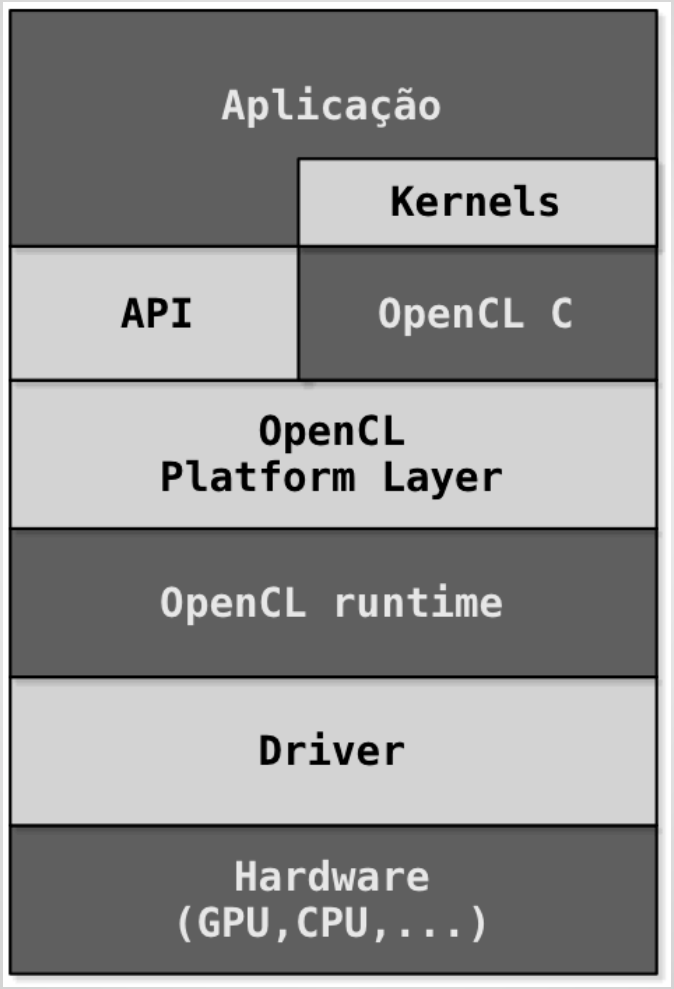
Introdução

- Problema: múltiplas ferramentas e paradigmas
  - Alto custo de implementação
  - Aplicações dependentes de plataforma
- Necessidade de um padrão para programação paralela

Introdução

- Open Computing Language (OpenCL)
  - Padrão aberto, livre de *royalties*, para programação paralela em ambientes heterogêneos
  - Mantido pelo Khronos Group desde 2008
  - Define *framework*
    - Arquitetura
    - Linguagem
    - API

Introdução



Camadas de uma aplicação OpenCL

Hello World

- 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];
    }
}
```

Hello World

- Código OpenCL (*kernel*)

```
__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

- Arquitetura abstrata de baixo nível
- Implementações mapeiam para entidades físicas
- Quatro modelos
  - Plataforma
  - Execução
  - Programação
  - Memória

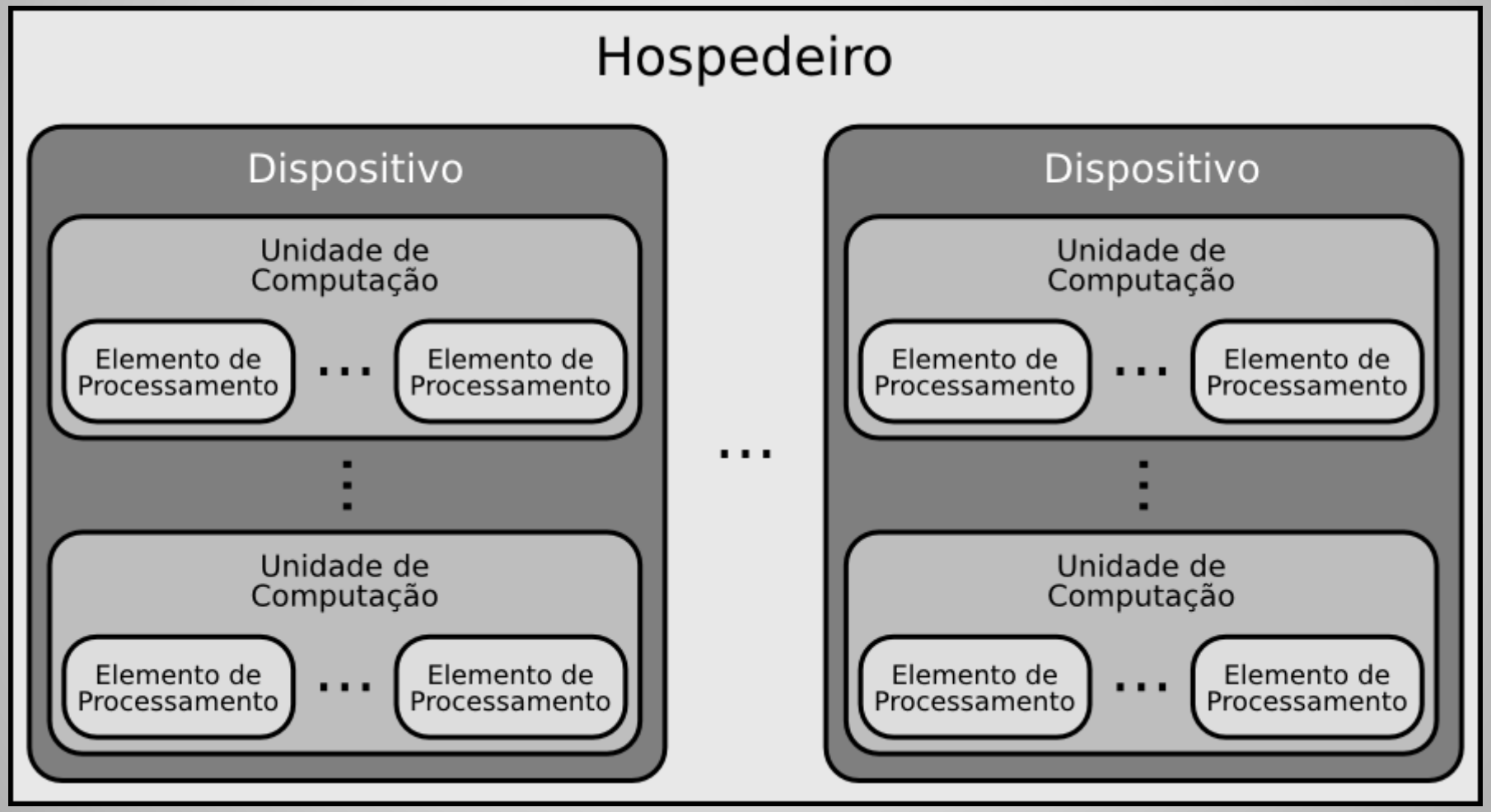
Arquitetura OpenCL

- Modelo de plataforma
  - Entidades do ambiente OpenCL
  - **Hospedeiro** (*host*)
    - Agrega **dispositivos** (*devices*) que suportam o padrão
    - Coordena execução
  - Dispositivo (*device*)
    - Composto de **unidades de computação** (*compute units*)

Arquitetura OpenCL

- Modelo de plataforma
  - Unidade de computação
    - Composta de **elementos de processamento** (*processing elements*)
  - Elemento de processamento
    - Realiza computação

Arquitetura OpenCL



Arquitetura OpenCL

- Modelo de execução
  - Descreve instanciação e identificação de *kernels*
  - Espaço de índices (*NDRange*)
    - Até 3 dimensões
    - **Itens de trabalho** (*work-items*) organizados em **grupos de trabalho** (*work-groups*)

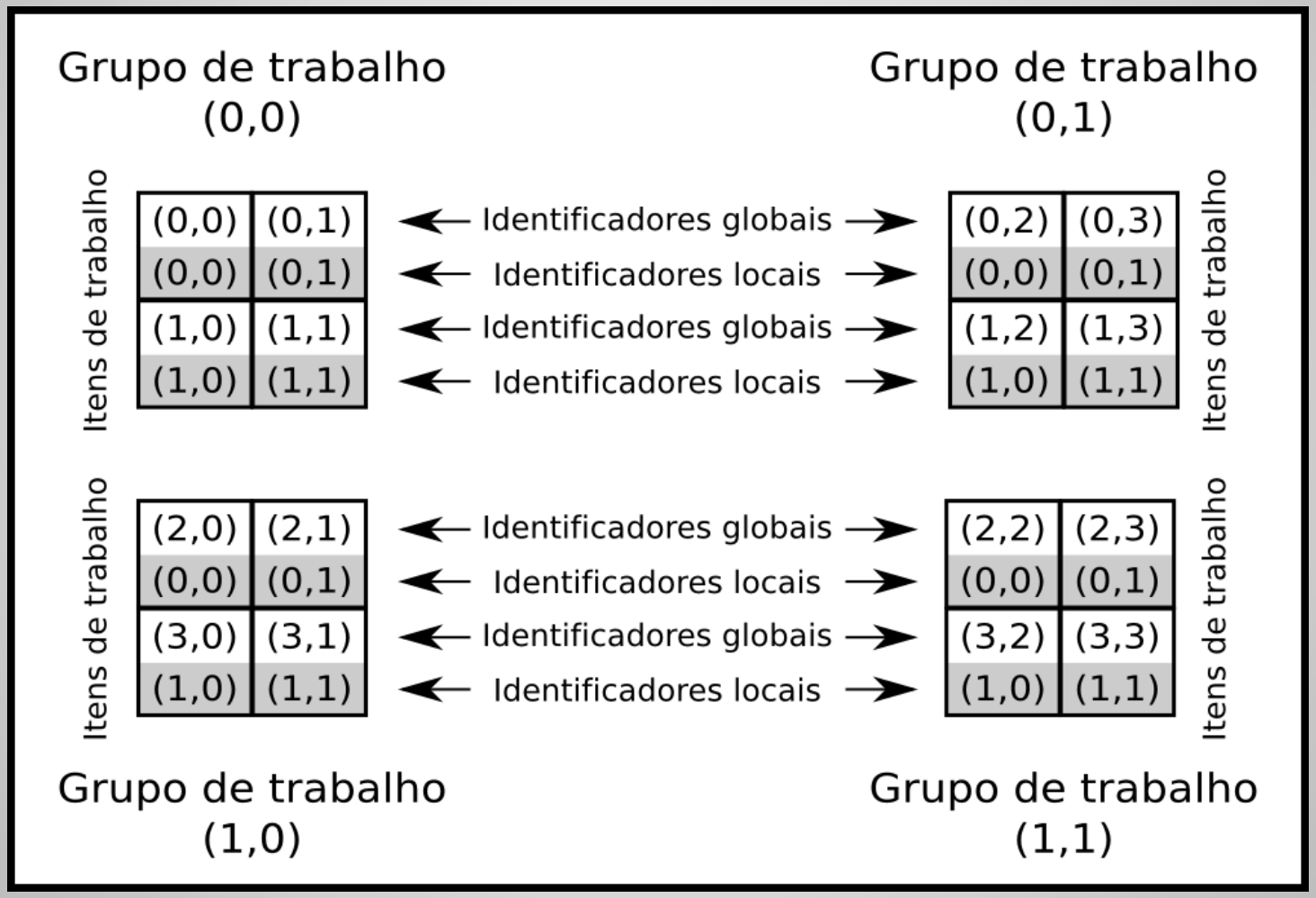
Arquitetura OpenCL

- Modelo de execução
  - Item de trabalho
    - Instância de um *kernel*
  - Grupo de trabalho
    - Permite sincronização entre itens de trabalho

Arquitetura OpenCL

- Modelo de execução: identificação
  - Item de trabalho
    - Identificadores globais no espaço de índices
    - Identificadores locais no grupo de trabalho
  - Grupo de trabalho
    - Identificadores no espaço de índices
  - **Um identificador por dimensão do espaço de índices**

Arquitetura OpenCL



Arquitetura OpenCL

- Modelo de execução: objetos
  - **Contexto** (*context*): agrega dispositivos, *kernels* e outras estruturas
  - **Fila de comandos** (*command queue*): comandos para um dispositivo
    - Comandos de I/O para memória, execução de *kernels*, sincronização e outros

Arquitetura OpenCL

- Modelo de execução: objetos
  - **Objeto de programa** (*program object*): encapsula código de um ou mais *kernels*
    - Base para compilação de *kernels*
  - **Objeto de memória** (*memory object*): comunicação com a memória dos dispositivos
    - *Buffers*: acesso direto (ponteiros)
    - *Images*: acesso especial via *samplers*, texturas

Arquitetura OpenCL

- Modelo de execução
  - Relacionado com modelo de plataforma
  - Item de trabalho: elemento de processamento
  - Grupo de trabalho: unidade de computação

Arquitetura OpenCL

- Modelo de programação
  - **Paralelismo de dados** (*data parallel*): múltiplos itens de trabalho para um *kernel*
  - **Paralelismo de tarefas** (*task parallel*): um item de trabalho para um *kernel*
    - Execução paralela de tarefas distintas
    - Útil quando há muito *overhead* de I/O entre hospedeiro e dispositivo

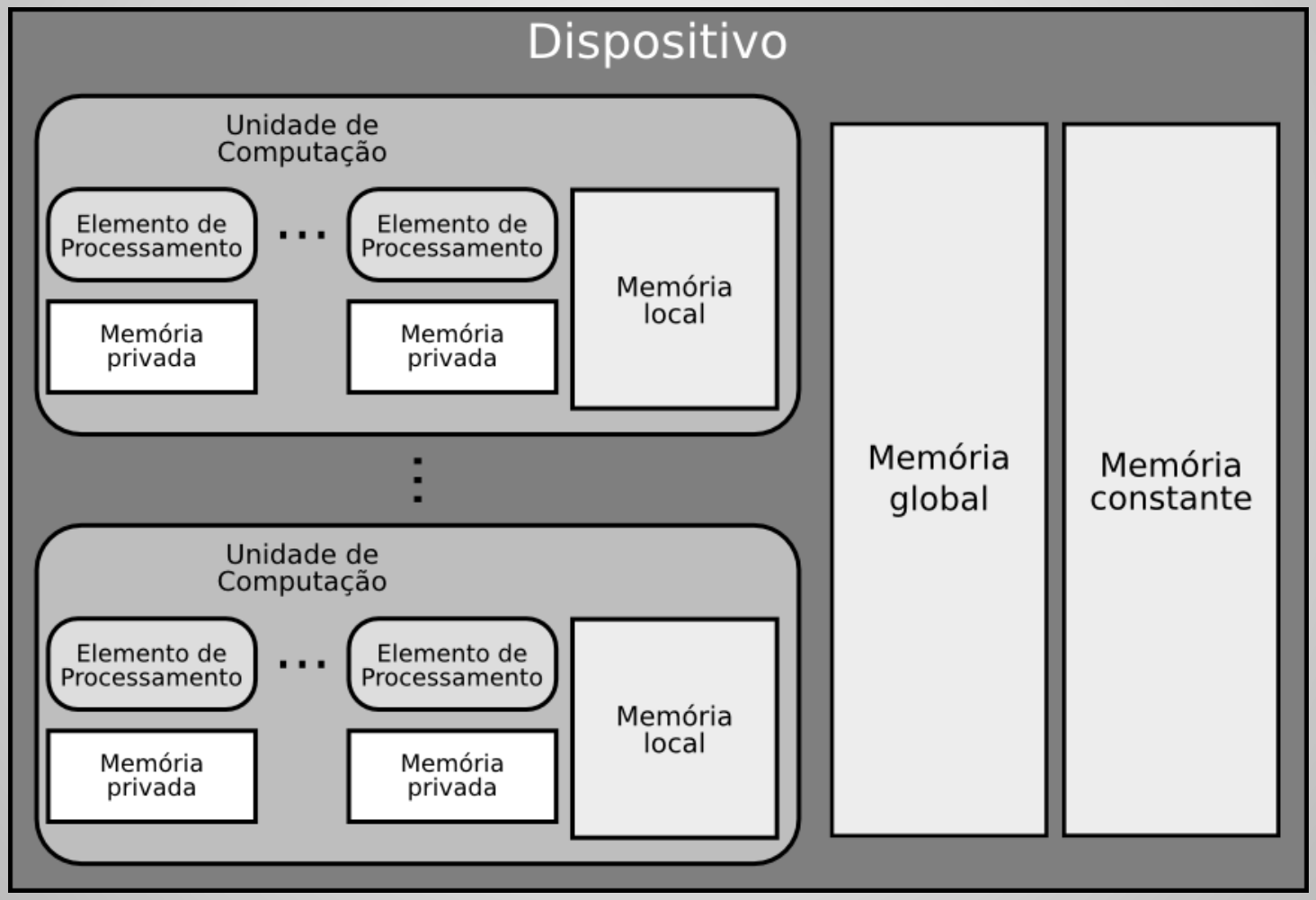
Arquitetura OpenCL

- Modelo de memória
  - **Memória global** (*global memory*)
    - Compartilhada entre todos itens de trabalho do espaço de índices
    - Escrita e leitura
  - **Memória local** (*local memory*)
    - Compartilhada apenas entre itens de trabalho do mesmo grupo de trabalho
    - Escrita e leitura

Arquitetura OpenCL

- Modelo de memória
  - **Memória privada** (*private memory*)
    - Exclusiva de cada item de trabalho
    - Escrita e leitura
  - **Memória constante** (*constant memory*)
    - Compartilhada entre todos itens de trabalho do espaço de índices
    - Somente leitura

Arquitetura OpenCL

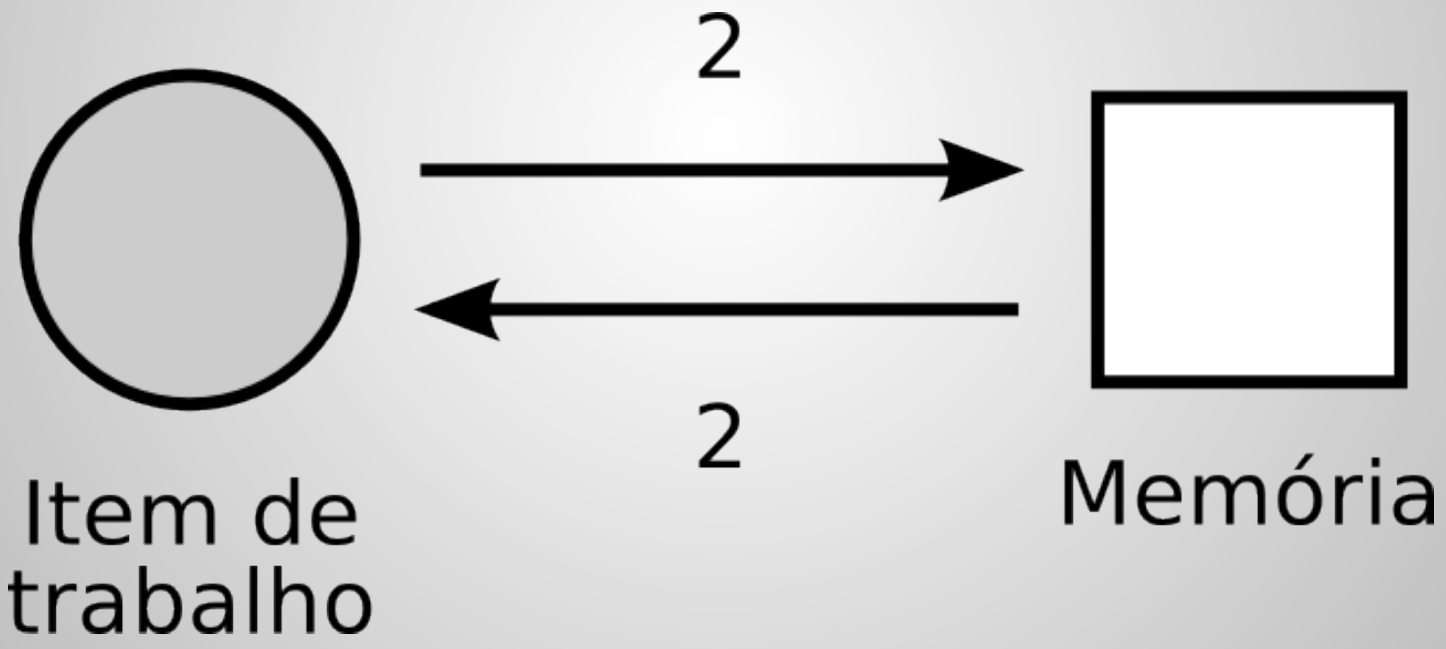


Arquitetura OpenCL

- Modelo de memória: consistência
  - Um item de trabalho lê corretamente o que outros escrevem?
  - Memória é consistente para:
    - Um único item de trabalho
    - Grupo de trabalho em uma **barreira** (sincronização)

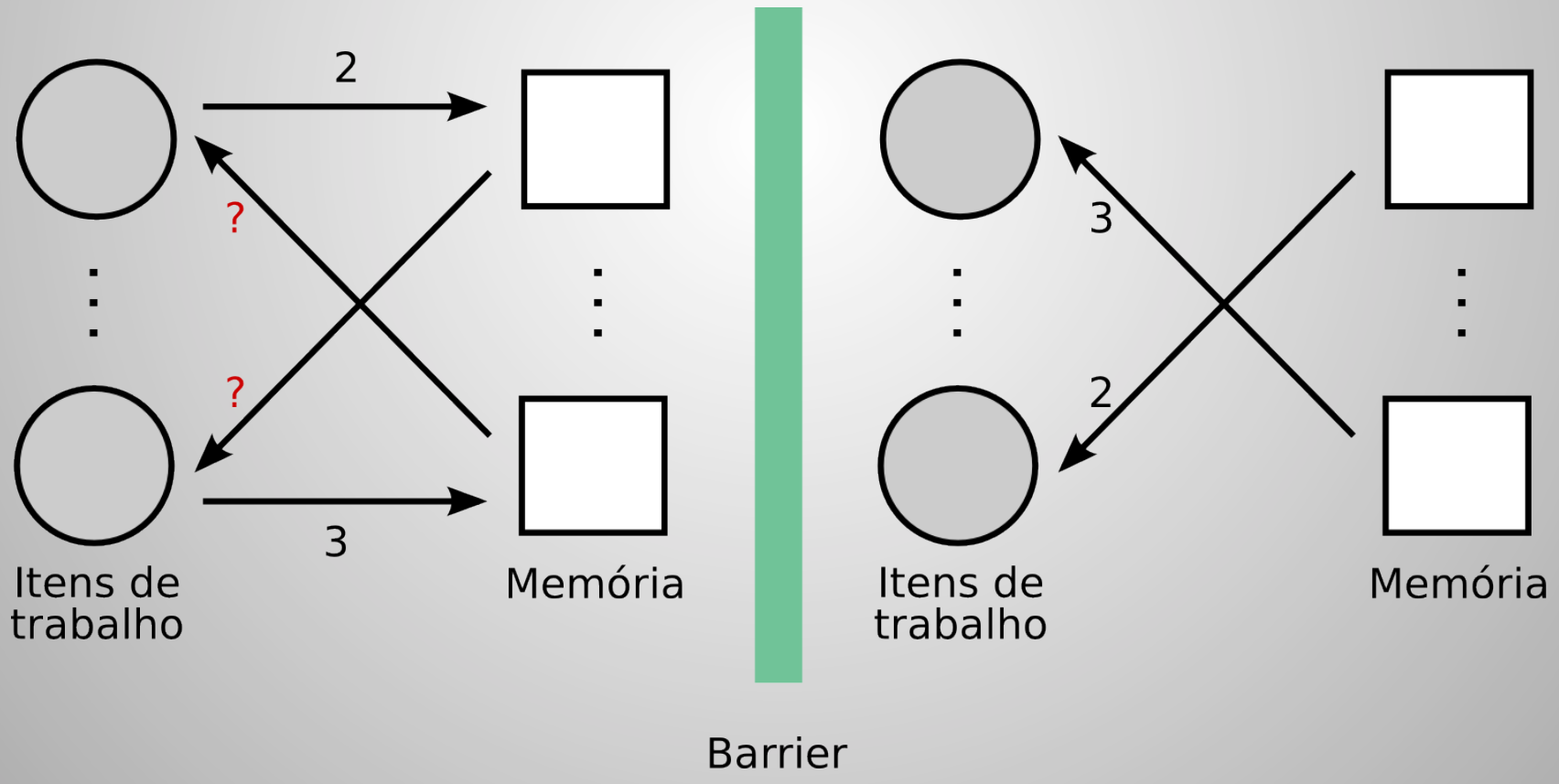
Arquitetura OpenCL

- Item de trabalho: memória consistente



Arquitetura OpenCL

- Grupo de trabalho: memória consistente após barreira (*barrier*)



Arquitetura OpenCL

- Modelo de memória: consistência
  - Não há sincronização entre grupos de trabalho

Linguagem OpenCL C

- Utilizada para a escrita de *kernels*
- Baseada no padrão C99
- Acrescenta extensões e restrições

Linguagem OpenCL C

- Extensão: tipos vetoriais
  - Notação: `tipo[# de componentes]`
  - 1, 2, 4, 8 ou 16 componentes
  - Exemplos:

```
float4
int8
short2
uchar16
```

Linguagem OpenCL C

- Operações com tipos vetoriais
  - Entre vetores de mesmo número de componentes
  - Entre vetores e escalares

```
float4 v = (float4)(1.0, 2.0, 3.0, 4.0);
float4 u = (float4)(1.0, 1.0, 1.0, 1.0);
float4 v2 = v * 2;
float4 t = v + u;
```

Linguagem OpenCL C

- Definição de *kernels*
  - Qualificador `__kernel`

```
__kernel void f(…)
{
    …
}
```

Linguagem OpenCL C

- Qualificadores de espaço de endereçamento
  - Definem nível da memória apontada por um ponteiro
  - `__global`, `__local`, `__private` e `__const`
  - *Default*: `__private`
  - **Em um *kernel*, todo argumento ponteiro deve estar acompanhado de um qualificador**

Linguagem OpenCL C

- Restrições
  - Ponteiros para função não são suportados
  - Argumentos para *kernels* não podem ser ponteiros para ponteiros
  - Funções e macros com número variável de argumentos não são suportados
  - Qualificadores `extern`, `static` e `auto` não são suportados

Linguagem OpenCL C

- Restrições
  - Não há suporte a recursão
  - Elementos de `structs` e `unions` devem pertencer ao mesmo espaço de endereçamento
  - Cabeçalhos padrão (`stdio.h`, `stdlib.h`, etc.) não estão presentes

API de suporte: *kernels*

- Funções de identificação
  - Informações sobre espaço de índices, item e grupo de trabalho

```
get_global_id(uint dimindx)
get_local_id(uint dimindx)
get_group_id(uint dimindx)
get_global_size(uint dimindx)
get_local_size(uint dimindx)
get_num_groups()
get_work_dim()
```

API de suporte: *kernels*

- Funções de sincronização

  ```
  barrier(cl_mem_fence_flags flags)
  ```

  Flags:

  ```
  CLK_LOCAL_MEM_FENCE
  CLK_GLOBAL_MEM_FENCE
  ```

API de suporte: *kernels*

- Sincronização

```
__kernel void aKernel(
     __global float* in,
     __global float* out)
{
     int gid = get_global_id(0);
      int lid = get_local_id(0);
      int lsize = get_local_size(0);

      __local float a[lsize];
```

API de suporte: *kernels*

```
    a[lid] = in[gid];

    barrier(CLK_LOCAL_MEM_FENCE);

    out[gid] =
        a[(lid + 4) % lsize] * 2;

    barrier(CLK_GLOBAL_MEM_FENCE);
}
```

API de suporte: hospedeiro

- Ilustração através de um exemplo prático
- Aplicação completa
  - *Kernel* para subtração dos elementos de dois *arrays*

Exemplo prático

- Exemplo de código executado no hospedeiro
  - Ilustra as principais etapas para o desenvolvimento de soluções OpenCL
  - Multi-plataforma

Exemplo prático

- Etapas
  - Aquisição do dispositivo (GPU)
  - Criação do contexto
  - Compilação do *kernel*
  - Criação dos *buffers* para entrada e saída
  - Escrita dos dados de entrada
  - Execução do *kernel*
  - Leitura dos dados de saída
  - Liberação de recursos

Exemplo prático

```
#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/opencl.h>
#endif
```

Exemplo prático

```
cl_platform_id platformId;
cl_device_id deviceID;

clGetPlatformIDs(1, &platformId, NULL);

clGetDeviceIDs(
    platformId,
    CL_DEVICE_TYPE_GPU, 1, &deviceId,
    NULL);
```

Exemplo prático

```
cl_context context =
  clCreateContext(
     NULL, 1, &deviceId,
     NULL, NULL, NULL);

cl_command_queue queue =
     clCreateCommandQueue(
          context, deviceId,
          NULL, NULL);
```

Exemplo prático

```
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]; \
    }";
```

Exemplo prático

```
cl_program program =
    clCreateProgramWithSource(
        context,
        1, &source,
        NULL, NULL);
```

Exemplo prático

```
clBuildProgram(
    program,
    0, NULL, NULL, NULL, NULL);

cl_kernel kernel =
    clCreateKernel(
        program,"ArrayDiff",
        NULL);
```

Exemplo prático

```
#define N …

cl_mem bufA = clCreateBuffer(
    context,
    CL_MEM_READ_ONLY,
    N * sizeof(int),
    NULL, NULL);

cl_mem bufB = …;
cl_mem bufC = …;
```

Exemplo prático

```
int* hostA;

clEnqueueWriteBuffer(
    queue,
    bufA,
    CL_TRUE,
    0,
    N * sizeof(int),
    hostA,
    0, NULL, NULL);
```

Exemplo prático

```
int* hostB;

clEnqueueWriteBuffer(
    queue,
    bufB,
    CL_TRUE,
    0,
    N * sizeof(int),
    hostB,
    0, NULL, NULL);
```

Exemplo prático

```
clSetKernelArg(
    kernel, 0,
    sizeof(cl_mem), &bufA);
clSetKernelArg(
    kernel, 1,
    sizeof(cl_mem), &bufB);
clSetKernelArg(
    kernel, 2,
    sizeof(cl_mem), &bufC);
```

Exemplo prático

```
const size_t globalSize[] =
    { N };

clEnqueueNDRangeKernel(
    queue, kernel,
    1, NULL, globalSize, NULL,
    0, NULL, NULL);

clFinish(queue);
```

Exemplo prático

```
int* hostC;

clEnqueueReadBuffer(
    queue,
    bufC,
    CL_TRUE,
    0,
    N * sizeof(int),
    hostC,
    0, NULL, NULL);
```

Exemplo prático

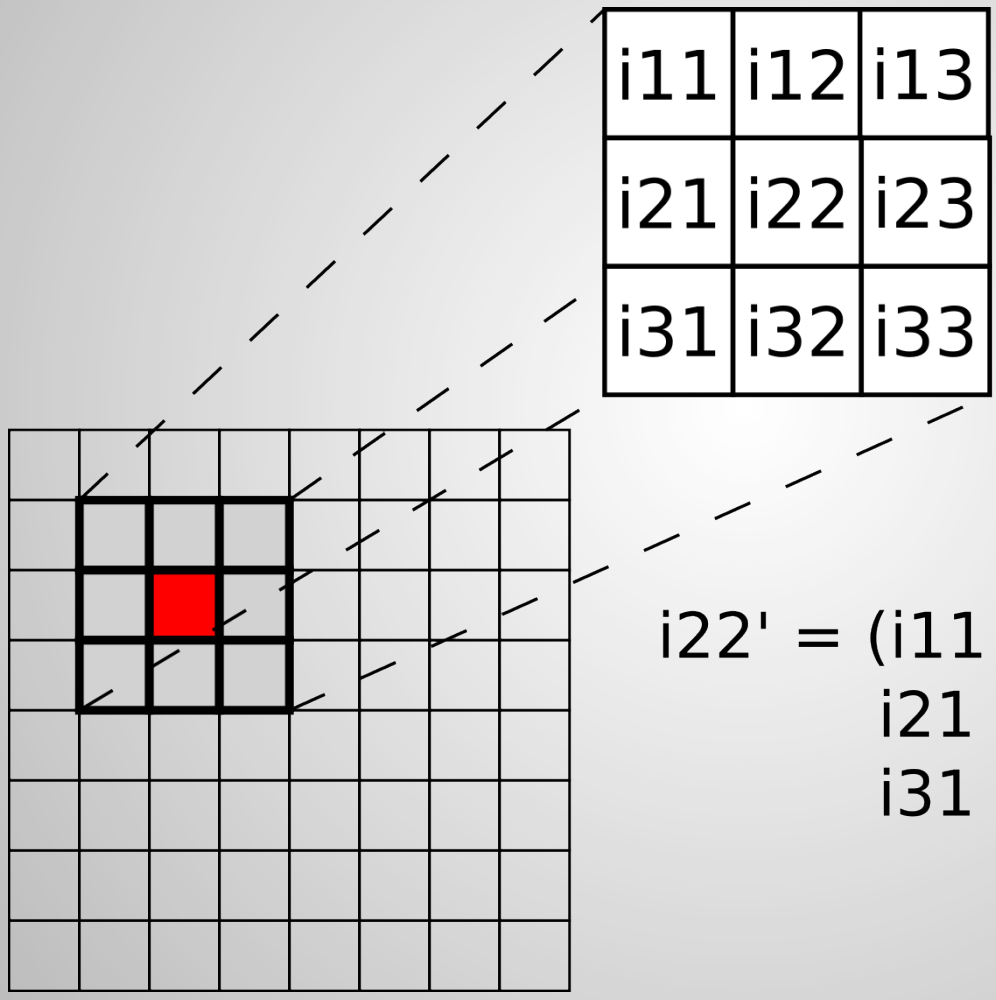
```
clReleaseMemObject(bufC);
clReleaseMemObject(bufB);
clReleaseMemObject(bufA);
clReleaseKernel(kernel);
clReleaseProgram(program);
clReleaseCommandQueue(queue);
clReleaseContext(context);
```

Exemplo prático

- Compilação
  - Instalar *toolkit* de desenvolvimento
    - Por exemplo, NVIDIA CUDA ou ATI Streaming SDK
  - Linux:

```
gcc -I/usr/local/cuda/include hello.c
-o hello –lOpenCL
```

Exemplo: filtro de imagem



$$i22' = (i11 + i12 + i13 + i21 + i22 + i23 + i31 + i32 + i33) / 9$$

Exemplo: filtro de imagem

```
__kernel void Blur(
    __global const int* in,
    __global int* out)
{
    int x = get_global_id(0);
    int y = get_global_id(1);
    int width = get_global_size(0);
    int height = get_global_size(1);
```

Exemplo: filtro de imagem

```
if (x == 0 || x == width - 1 ||
    y == 0 || y == height - 1)
{
    out[y * width + x] =
        in[y * width + x];
}
```

Exemplo: filtro de imagem

```
    else
    {
        int sum = 0;

        for (int i = -1; i <= 1; ++i)
            for (int j = -1; j <= 1; ++j)
                sum += in[(y + i) * width +
                          (x + j)];

        out[y * width + x] = sum / 9;
    }
}
```

Exemplo: filtro de imagem

- OpenCL permite o foco na solução de uma unidade do problema
  - *Runtime* é responsável pela criação dos itens de trabalho e divisão das tarefas

Comparações com outras tecnologias

- OpenMP
  - Necessidade de explicitar loops e compartilhamento de variáveis
  - Apenas CPU
- Interface nativa para *threads*
  - Gerência manual de threads
  - Apenas CPU

Comparações com outras tecnologias

- NVIDIA CUDA
  - Específico de plataforma
  - Apenas GPU

Considerações finais

- NVIDIA e ATI suportam
- ATI adotou OpenCL como linguagem oficial do Streaming SDK
- Imagination: rumores de suporte a OpenCL em futuros iPhones

Considerações finais

- Material estará disponível em

http://labs.v3d.com.br/

Dúvidas?