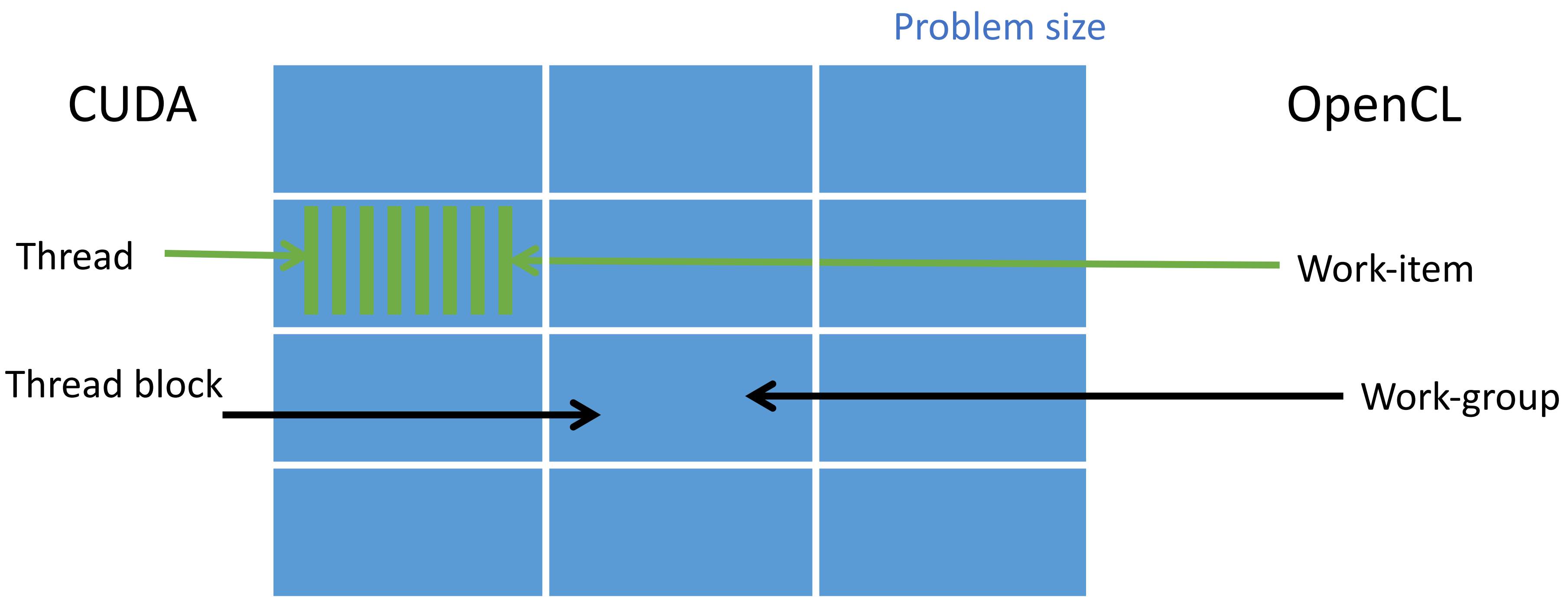
CPSC/ECE 4780/6780

General-Purpose Computation on Graphical Processing Units (GPGPU)

Lecture 12: (OpenCL) CUDA and OpenCL by Comparison

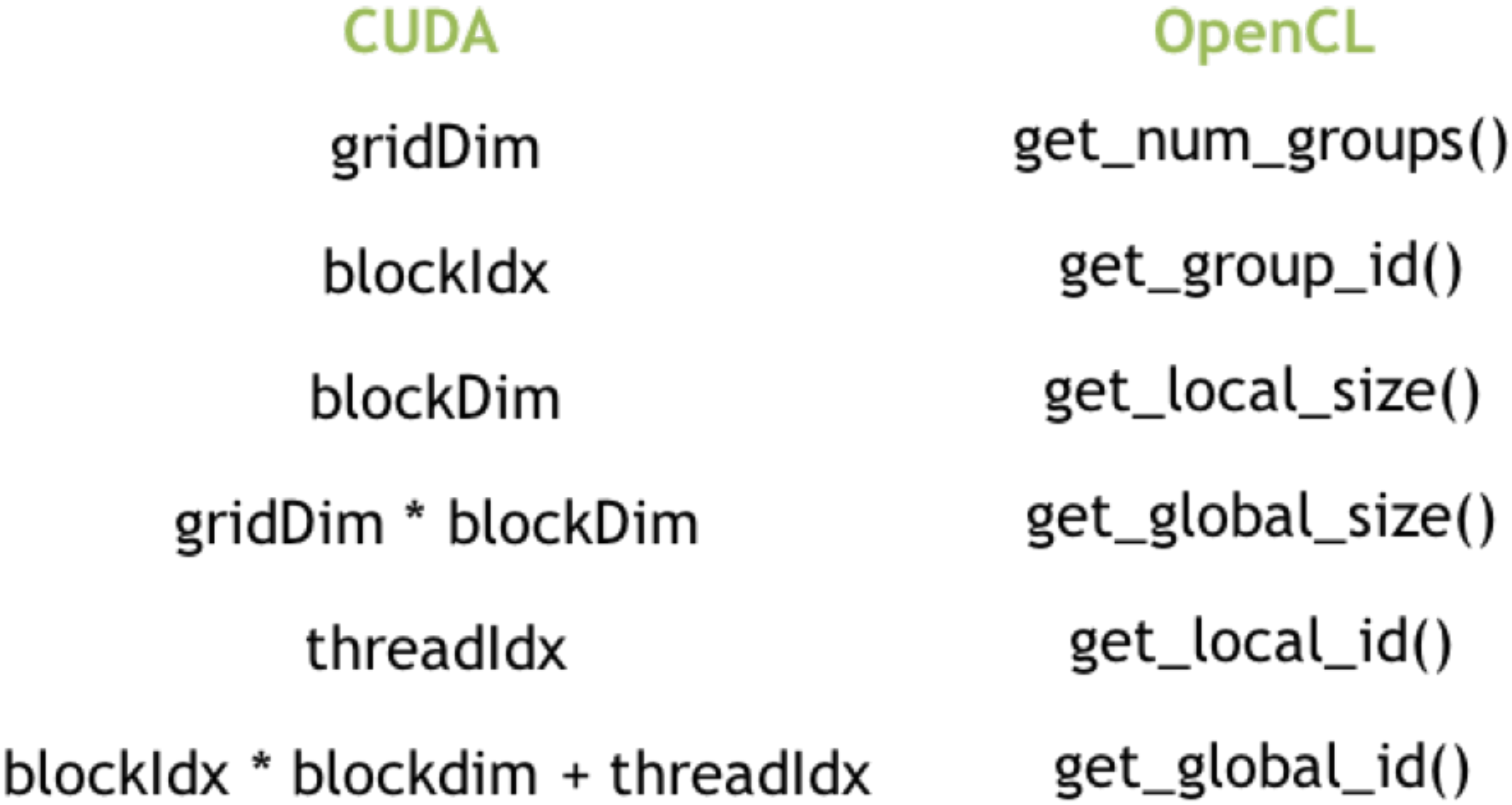
Execution Model



Kernels

CUDA	OpenCL
Denote by __global__	Denote by __kernel
A function in the host code	Either a string (const char*), or read from a file
Compile with compilation of host code	Compile at runtime

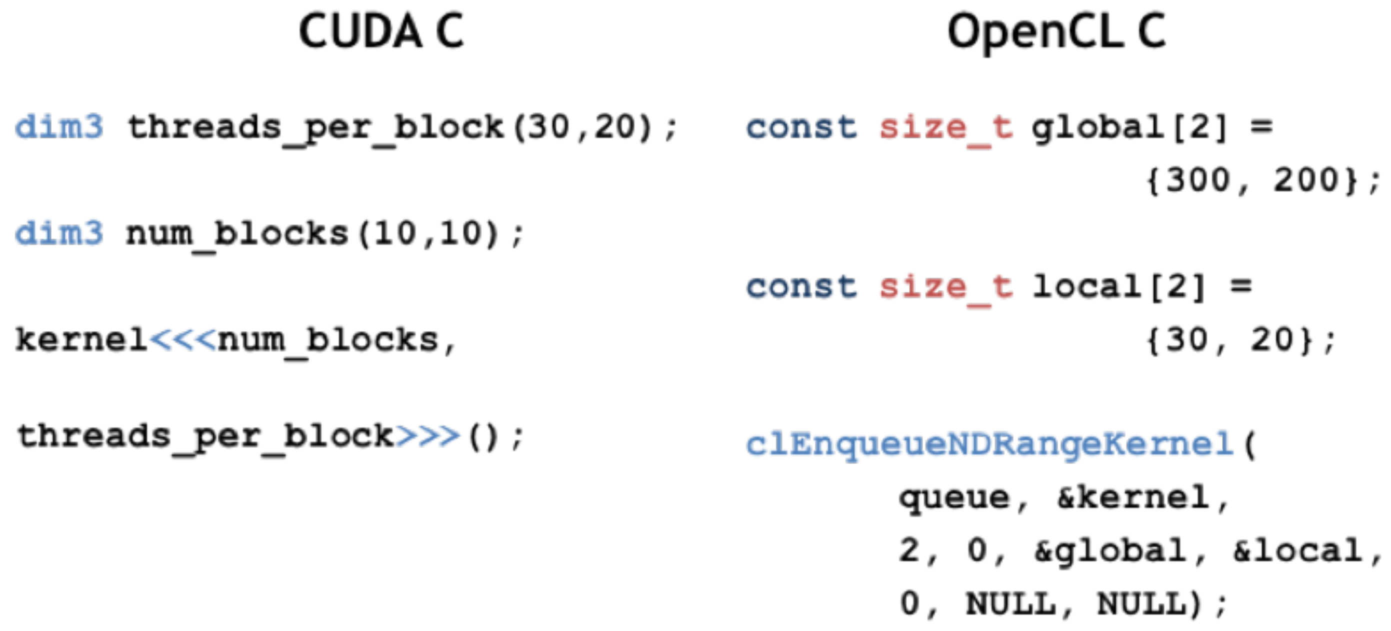
Kernel Indexing



CUDA	OpenCL
gridDim	get_num_groups()
blockIdx	get_group_id()
blockDim	get_local_size()
gridDim * blockDim	get_global_size()
threadIdx	get_local_id()
blockIdx * blockdim + threadIdx	get_global_id()

Enqueue a Kernel

- To enqueue the kernel
  - CUDA – specify the number of thread blocks and threads per block
  - OpenCL – specify the problem size and (optionally) number of work-items per work-group



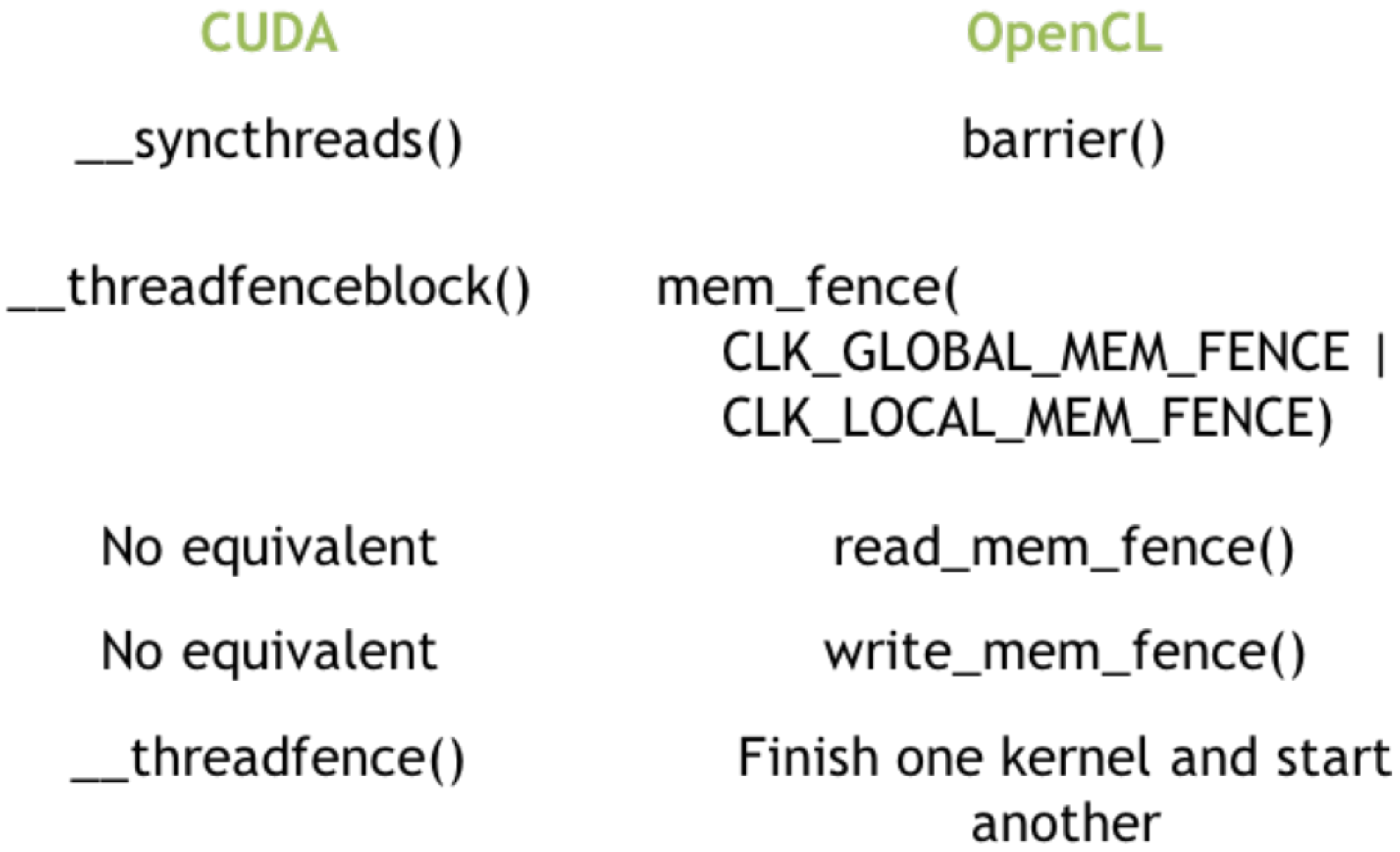
CUDA C

```
dim3 threads_per_block(30,20);

dim3 num_blocks(10,10);

kernel<<<num_blocks,

threads_per_block>>>();
```

OpenCL C

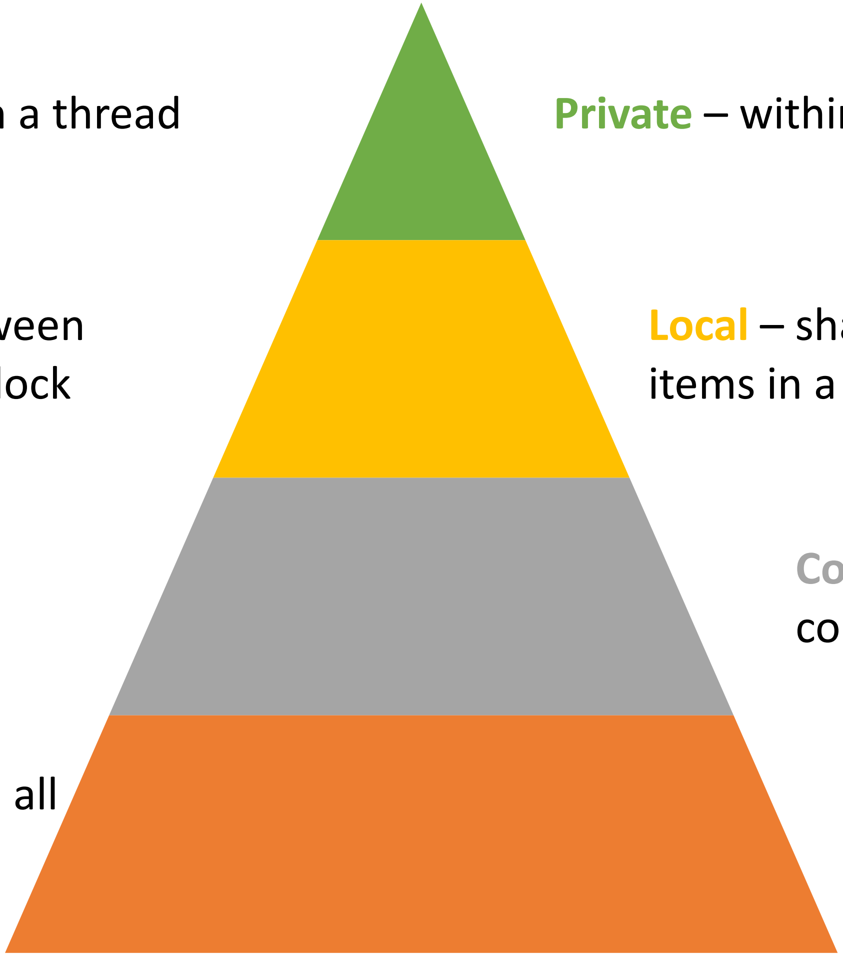
```
const size_t global[2] =
                    {300, 200};

const size_t local[2] =
                    {30, 20};

clEnqueueNDRangeKernel(
       queue, &kernel,
       2, 0, &global, &local,
       0, NULL, NULL);
```

Kernel Synchronization



CUDA	OpenCL
__syncthreads()	barrier()
__threadfenceblock()	mem_fence(CLK_GLOBAL_MEM_FENCE | CLK_LOCAL_MEM_FENCE)
No equivalent	read_mem_fence()
No equivalent	write_mem_fence()
__threadfence()	Finish one kernel and start another

Memory Hierarchy Terminology



Local – within a thread

Shared – shared between threads in a thread block

Constant – a cache for constant memory

Global – shared between all thread blocks

CUDA

Private – within a work-item

Local – shared between work-items in a work-group

Constant – a cache for constant memory

Global – shared between all work-groups

OpenCL

Allocating and Copying Memory

	CUDA C	OpenCL C

Allocate	`float* d_x;`
	`cudaMalloc(&d_x,`
	`sizeof(float)*size);`
		`cl_mem d_x =`
		`clCreateBuffer(context,`
		`CL_MEM_READ_WRITE,`
		`sizeof(float)*size,`
		`NULL, NULL);`

Host to Device	`cudaMemcpy(d_x, h_x,`
	`sizeof(float)*size,`
	`cudaMemcpyHostToDevice);`
		`clEnqueueWriteBuffer(queue, d_x,`
		`CL_TRUE, 0,`
		`sizeof(float)*size,`
		`h_x, 0, NULL, NULL);`

Device to Host	`cudaMemcpy(h_x, d_x,`
	`sizeof(float)*size,`
	`cudaMemcpyDeviceToHost);`
		`clEnqueueReadBuffer(queue, d_x,`
		`CL_TRUE, 0,`
		`sizeof(float)*size,`
		`h_x, 0, NULL, NULL);`

Declaring Dynamic Local/Shared Memory

CUDA C

1. Define an array in the kernel source as extern

```
__shared__ int array[];
```

2. When executing the kernel, specify the third parameter as size in bytes of shared memory

```
func<<<num_blocks,
 num_threads_per_block,
 shared_mem_size>>>(args);
```

OpenCL C

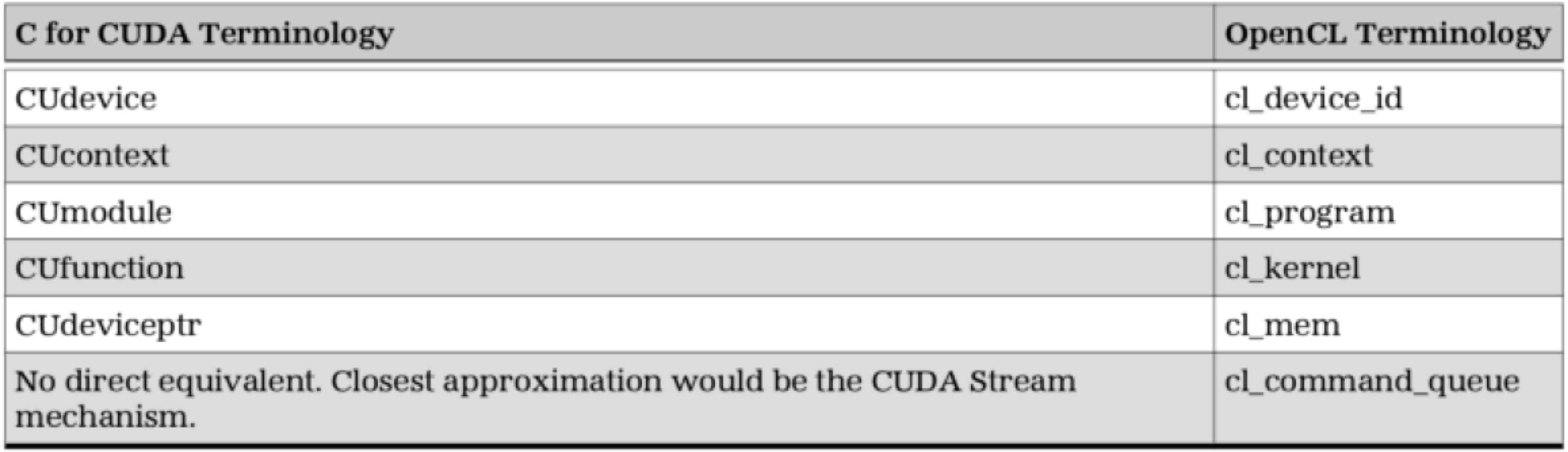
1. Have the kernel accept a local array as an argument

```
__kernel void func(
            __local int *array)
{}
```

2. Specify the size by setting the kernel argument

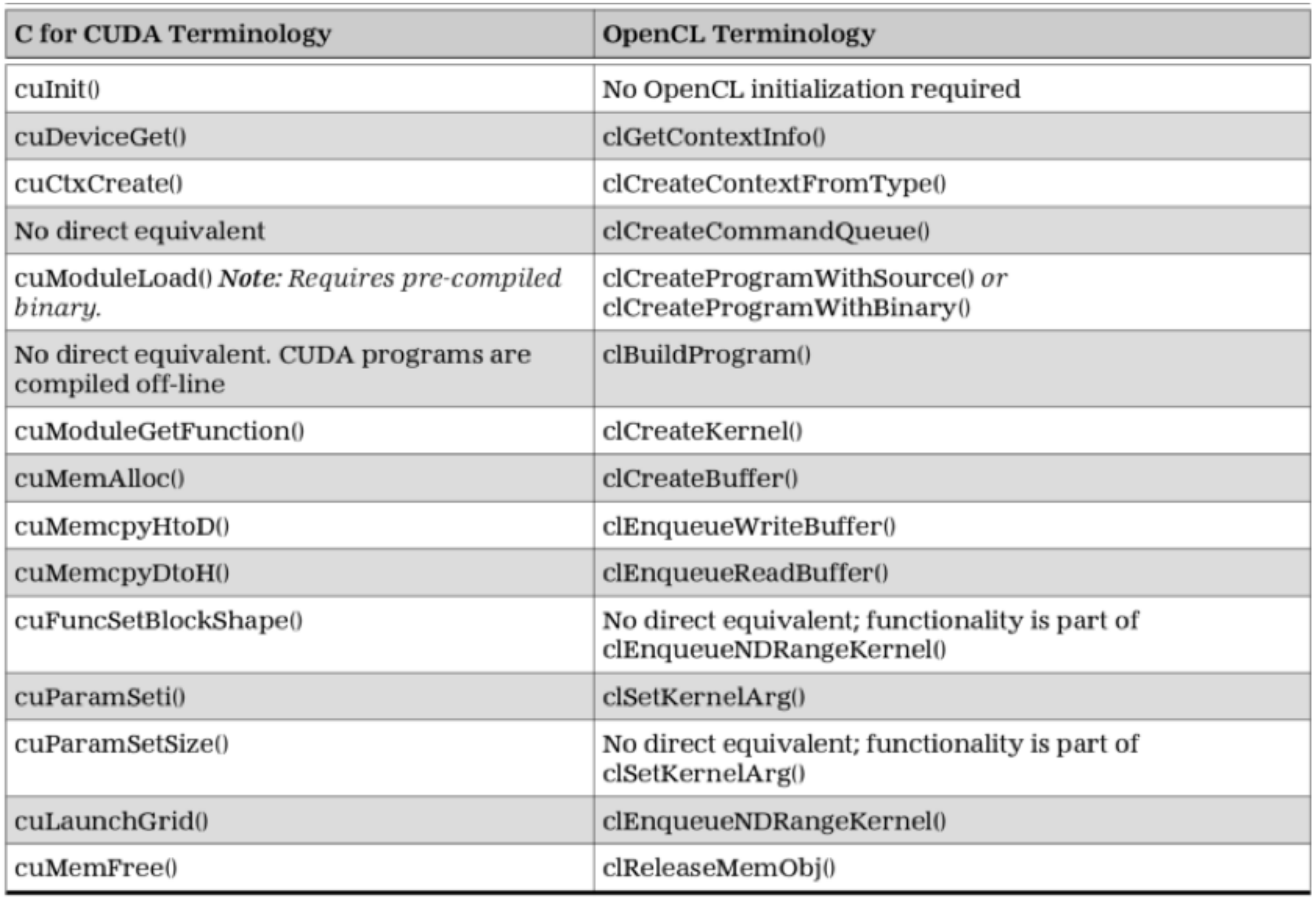
```
clSetKernelArg(kernel, 0,
   sizeof(int)*num_elements,
   NULL);
```

General API Terminology



C for CUDA Terminology	OpenCL Terminology
CUdevice	cl_device_id
CUcontext	cl_context
CUmodule	cl_program
CUfunction	cl_kernel
CUdeviceptr	cl_mem
No direct equivalent. Closest approximation would be the CUDA Stream mechanism.	cl_command_queue

Important API Calls



C for CUDA Terminology	OpenCL Terminology
cuInit()	No OpenCL initialization required
cuDeviceGet()	clGetContextInfo()
cuCtxCreate()	clCreateContextFromType()
No direct equivalent	clCreateCommandQueue()
cuModuleLoad() ***Note:*** *Requires pre-compiled binary.*	clCreateProgramWithSource() *or* clCreateProgramWithBinary()
No direct equivalent. CUDA programs are compiled off-line	clBuildProgram()
cuModuleGetFunction()	clCreateKernel()
cuMemAlloc()	clCreateBuffer()
cuMemcpyHtoD()	clEnqueueWriteBuffer()
cuMemcpyDtoH()	clEnqueueReadBuffer()
cuFuncSetBlockShape()	No direct equivalent; functionality is part of clEnqueueNDRangeKernel()
cuParamSeti()	clSetKernelArg()
cuParamSetSize()	No direct equivalent; functionality is part of clSetKernelArg()
cuLaunchGrid()	clEnqueueNDRangeKernel()
cuMemFree()	clReleaseMemObj()