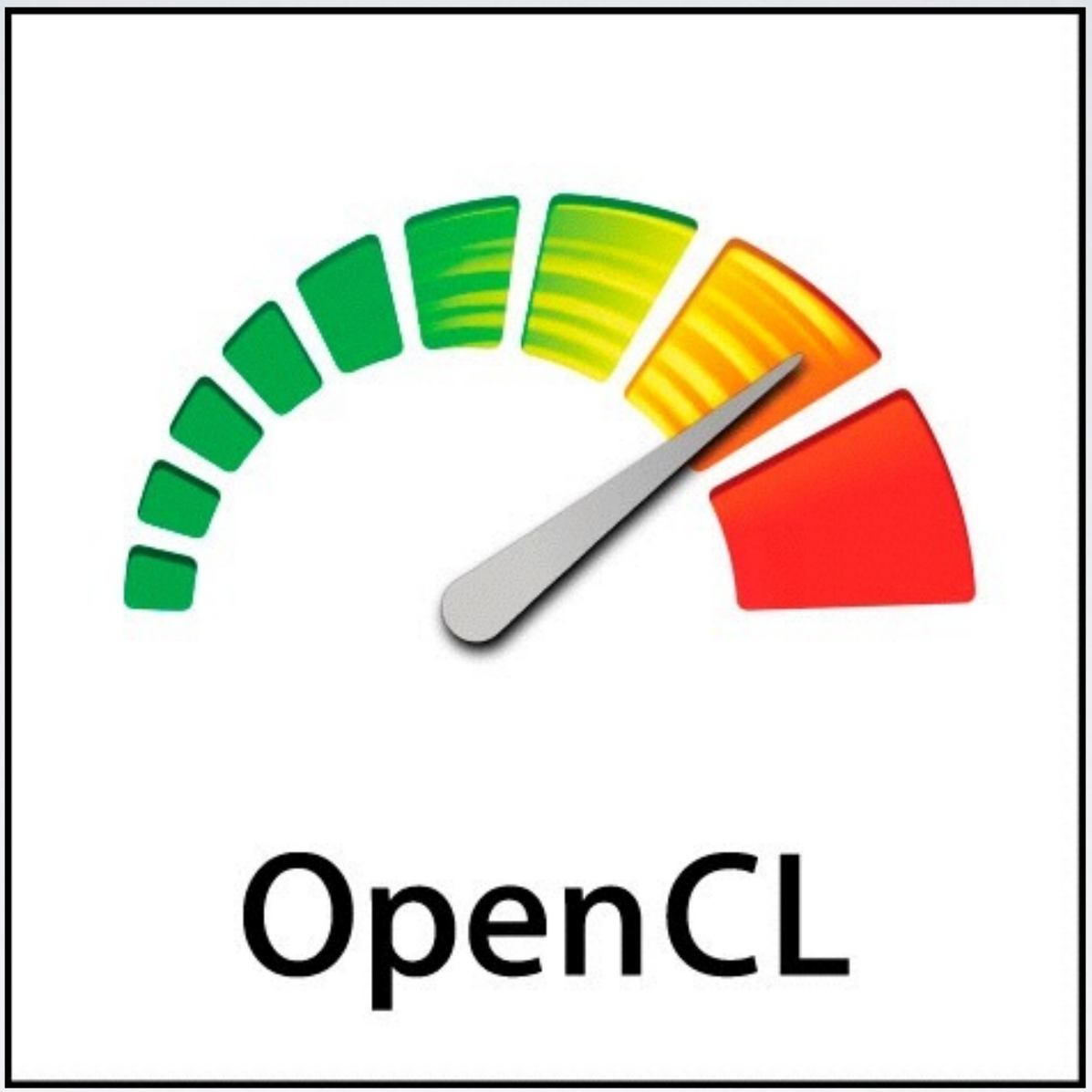


INTRODUCTION TO OPENCL

Jason B. Smith, Hood College

May 4 2011

WHAT IS IT?

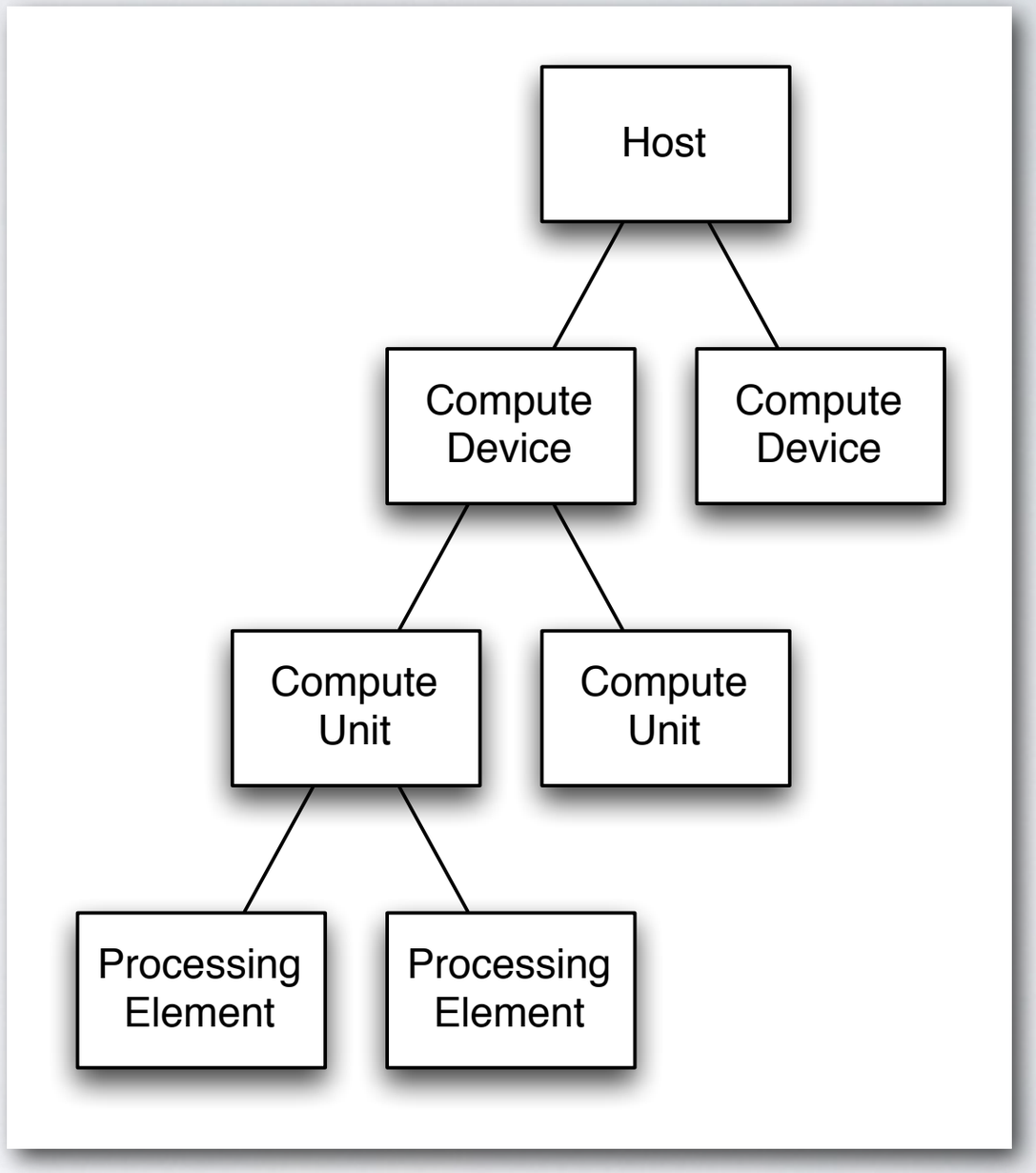
- Use **heterogeneous** computing platforms
- Specifically for **computationally intensive** apps
- Provide a means for **portable parallelism**
- Kernels written in a subset of **C99**

A LITTLE HISTORY

- Managed by non-profit tech consortium Khronos Group
- Originally developed by Apple
- OpenCL 1.0 - December 8, 2008
  - First Demo on GPU - NVIDIA December 12, 2008
- OpenCL 1.1 - June 14, 2010

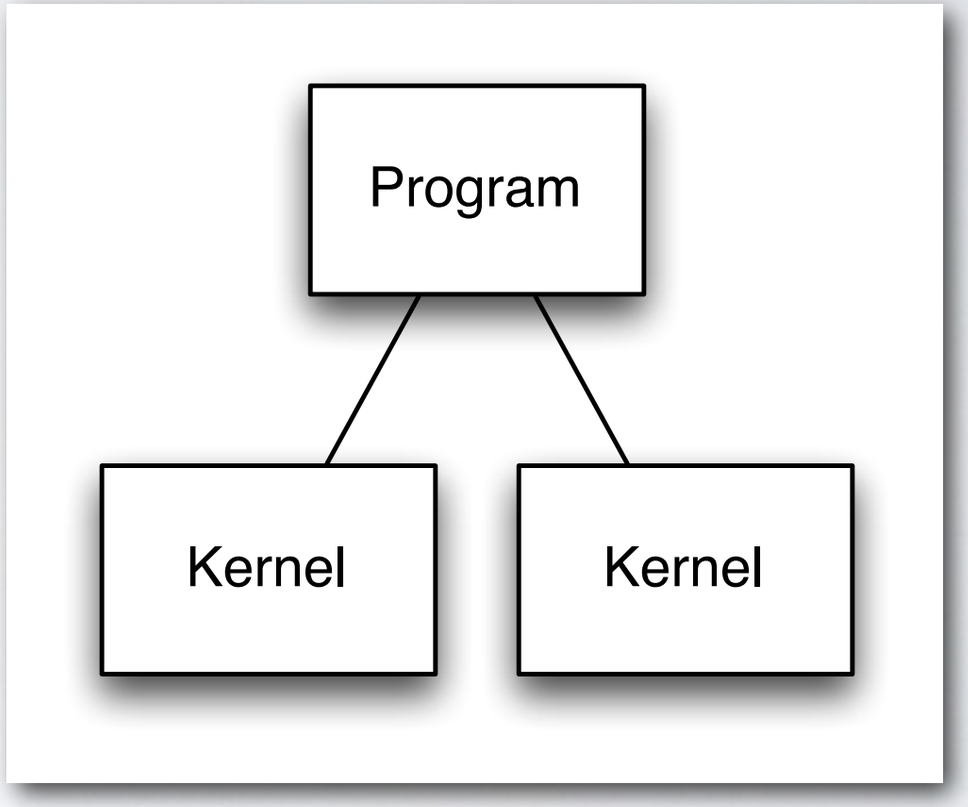
PLATFORM MODEL

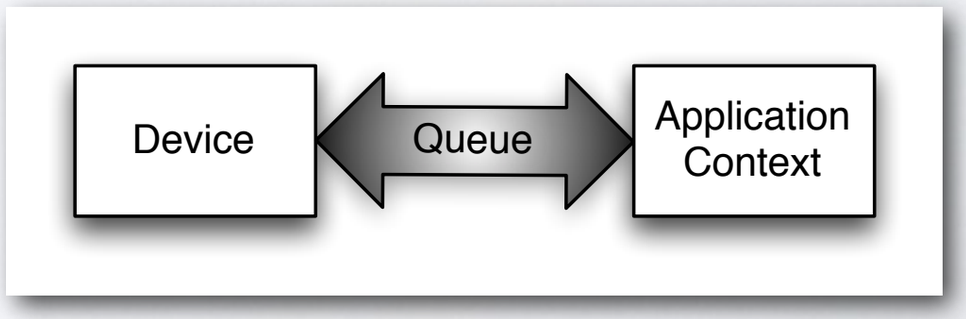
- Host
- Compute Devices (GPU/CPU)
- Compute Unit (~ Core)
- Processing Element



EXECUTION MODEL

- Program
- Kernel
- Context
- Queue



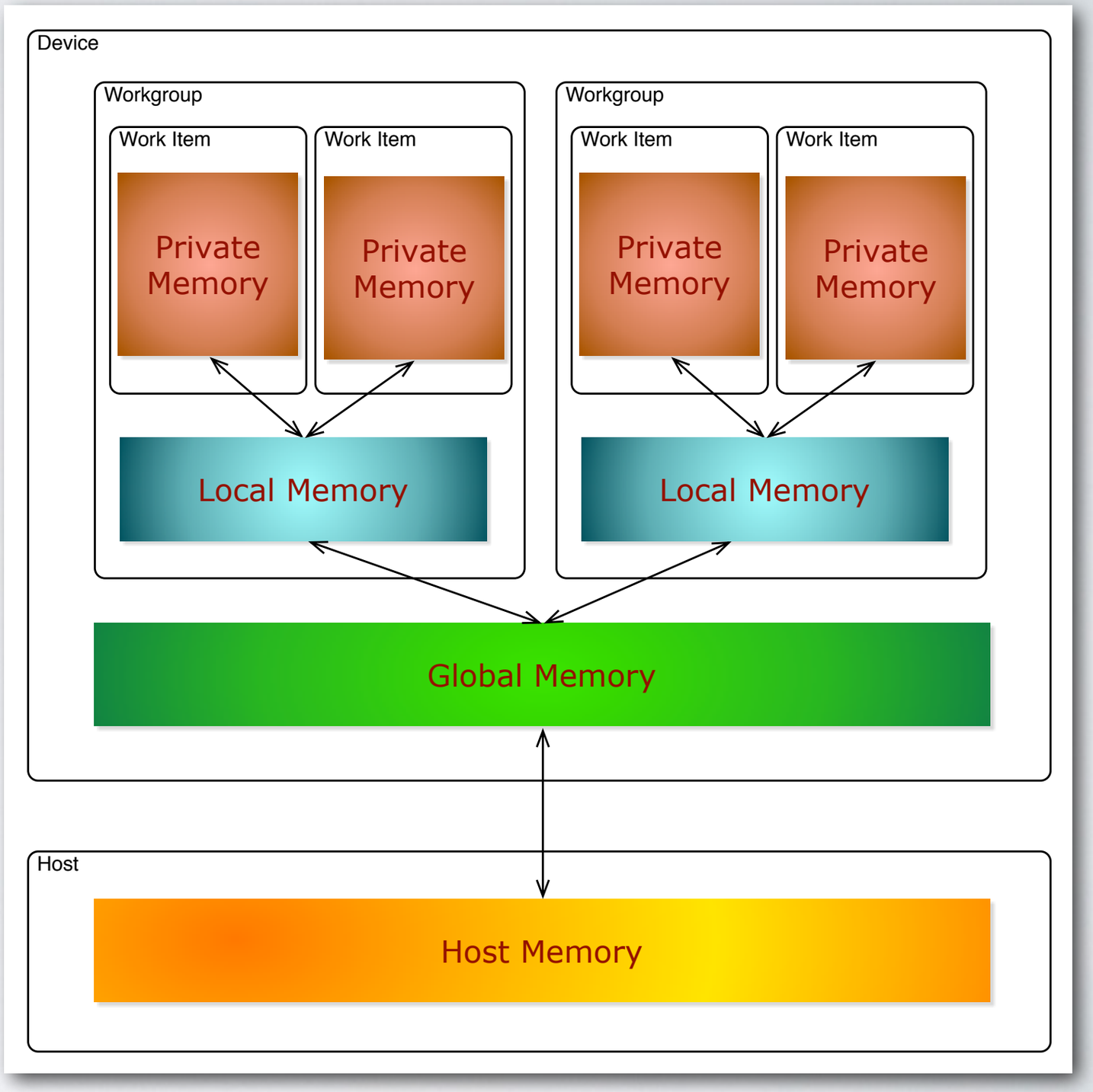


CONTEXT

- Created with one or more devices
- Manage queues, memory, programs and kernels
- Allow kernel execution on one or more devices

MEMORY MODEL

- Host
- Local Global/Constant
- Local Memory
- Private



MEMORY SPACES

- **Global** - Read/Write access for all work items
- **Constant** - host allocated read only memory
- **Local** - allocated to work group (shared)
- **Private** - visible to individual work item only

INTERLUDE: GETTING INFO

CONTRASTING C & C++

VARIABLES

```
#if defined __APPLE__ || defined(MACOSX)
        #include <OpenCL/opencl.h>
#else
        #include <CL/opencl.h>
#endif

int main(int argc, char *argv[]) {

        cl_int                  error = 0;
        cl_platform_id  platforms[10];
        cl_platform_id  platform;
        cl_uint num_platforms;
        char param[255];
        char *param_name[] = {
                "CL_PLATFORM_PROFILE", "CL_PLATFORM_VERSION",
"CL_PLATFORM_NAME","CL_PLATFORM_VENDOR", "CL_PLATFORM_EXTENSIONS" };
        cl_platform_info params[] = {
                CL_PLATFORM_PROFILE, CL_PLATFORM_VERSION,
CL_PLATFORM_NAME,  CL_PLATFORM_VENDOR, CL_PLATFORM_EXTENSIONS};

        cl_device_id devices[10];
        cl_device_id device;
        cl_uint num_devices;
```

PLATFORMS

```
35          error = clGetPlatformIDs(
36                  10,
37                  platforms,
38                  &num_platforms);
39          if (error != CL_SUCCESS) {
40                  printf("Error getting platform id: %d", error);
41                  exit(-1);
42          }
43          printf("Number of platforms: %d\n", num_platforms);
```

PLATFORM INFO

```
//
50                  // Loop for specific platform paramater
51                  //
52                  for (int j=0; j<5; j++) {
53                          error = clGetPlatformInfo(
54                                                  platforms[i],
55                                                  params[j],
56                                                  255,
57                                                  param,
58                                                  NULL);  // assume 255 is enough
59                          if (error != CL_SUCCESS) {
60                                  printf("Error getting platform id: %d", error);
61                                  exit(-1);
62                          }
63                          printf("%s:\t%s\n", param_name[j], param);
64                  }
```

DEVICES

```
//
67                  // Device IDs for platform
68                  //
69                  error = clGetDeviceIDs(
70                          platforms[i],
71                          CL_DEVICE_TYPE_ALL,
72                          10,
73                          devices,
74                          &num_devices);
75                  if (error != CL_SUCCESS) {
76                          printf("Error getting platform id: %d", error);
77                          exit(-1);
78                  }
79                  printf("Found %d devices\n", num_devices);
```

ALTERNATIVE: C++

- Use C++ Standard Libraries
- OpenCL Specific Containers
  - OpenCL versions of String & Vector
- Can use C++ exception handling
- RAII

VARIABLES

```
 5 #include <algorithm>
 6 #include <iostream>
 7 #include "cl.hpp"
 8
 9 #define __CL_ENABLE_EXCEPTIONS
10
11 using namespace std;


47 int main(int argc, char *argv[]) {
48         cout << "Starting environment info" << endl;
49         vector<cl::Platform> platforms;
50
51         //
52         // Query platform info
53         //
54         cl::Platform::get(&platforms);
55         cout << "Found " << platforms.size() << " available platforms" <<
endl;
56         for_each (platforms.begin(), platforms.end(), show_platform_info);
57
58         cout << "Finished" << endl;
59         return 0;
60 }
```

PLATFORMS

```
29 void show_platform_info(cl::Platform &platform) {
30         cl::STRING_CLASS info;
31         vector<cl::Device> devices;
32 
33         cout << "Platform Profile: " << platform.getInfo<CL_PLATFORM_PROFILE>() << endl;
34         /*  Alternate (non-traits) version
35         platform.getInfo(CL_PLATFORM_PROFILE, &info);
36         cout << "OpenCL Platform Profile: " << info << endl;
37         */
38         cout << "Platform Version: " << platform.getInfo<CL_PLATFORM_VERSION>() << endl;
39         cout << "Platform Name   : " << platform.getInfo<CL_PLATFORM_NAME>() << endl;
40         cout << "Platform Vendor : " << platform.getInfo<CL_PLATFORM_VENDOR>() << endl;
41 
42         platform.getDevices(CL_DEVICE_TYPE_ALL, &devices);
43         cout << "Found " << devices.size() << " for this platform\n" << endl;
44         for_each(devices.begin(), devices.end(), show_device_info);
45 }
```

DEVICES

```
13 void show_device_info(cl::Device &device) {
14         cout << "\n\tVendor: " << device.getInfo<CL_DEVICE_VENDOR>() << endl;
15         cout << "\tType: " << device.getInfo<CL_DEVICE_TYPE>() << endl;
16         cout << "\tMax Work Items: " <<
device.getInfo<CL_DEVICE_MAX_COMPUTE_UNITS>() << endl;
17         cout << "\tMax Work Item Dimensions: " <<
device.getInfo<CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS>() << endl;

18         vector<size_t> dim = device.getInfo<CL_DEVICE_MAX_WORK_ITEM_SIZES>();
19         cout << "\tMax Work Item Sizes: ";

20         for (unsigned int i=0;i<dim.size();i++) { cout << " " << dim[i]; }
21         cout << endl;
22         cout << "\tMax Work Group Size: " <<
device.getInfo<CL_DEVICE_MAX_WORK_GROUP_SIZE>() << endl;
23         long max_mem = device.getInfo<CL_DEVICE_GLOBAL_MEM_SIZE>();
24         cout << "\tMax Device Global Mem Size: " <<  max_mem << " bytes ("
25                 << max_mem/1024/1024 << " mb)"<< endl;
26
27 }
```

RESULTS

```
Starting environment info
Found 1 available platforms
Platform Profile: FULL_PROFILE
Platform Version: OpenCL 1.0 (Dec 26 2010 12:52:21)
Platform Name   : Apple
Platform Vendor : Apple
Found 2 for this platform

        Vendor: AMD
        Type: 4
        Max Work Items: 5
        Max Work Item Dimensions: 3
        Max Work Item Sizes:  1024 1024 1024
        Max Work Group Size: 1024
        Max Device Global Mem Size: 536870912 bytes (512 mb)

        Vendor: Intel
        Type: 2
        Max Work Items: 8
        Max Work Item Dimensions: 3
        Max Work Item Sizes:  1 1 1
        Max Work Group Size: 1
        Max Device Global Mem Size: 3221225472 bytes (3072 mb)
Finished
```

RESULTS - HOOD GPU

```
Starging environment info
Found 1 available platforms
Platform Profile: FULL_PROFILE
Platform Version: OpenCL 1.0 CUDA 4.0.1
Platform Name   : NVIDIA CUDA
Platform Vendor : NVIDIA Corporation
Found 2 for this platform

        Vendor: NVIDIA Corporation
        Type: 4
        Max Work Items: 14
        Max Work Item Dimensions: 3
        Max Work Item Sizes:  1024 1024 64
        Max Work Group Size: 1024

        Vendor: NVIDIA Corporation
        Type: 4
        Max Work Items: 2
        Max Work Item Dimensions: 3
        Max Work Item Sizes:  1024 1024 64
        Max Work Group Size: 1024
Finished

GPU 0: Tesla C2050 (S/N: 0321310040087)
GPU 1: GeForce GT 430 (UUID: N/A)
```

USING OPENCL

- **Setup** - Query & configure execution environment
- **Memory** - Allocate buffers/images
- **Execution** - Run program and kernels

OPENCL UML DIAGRAM

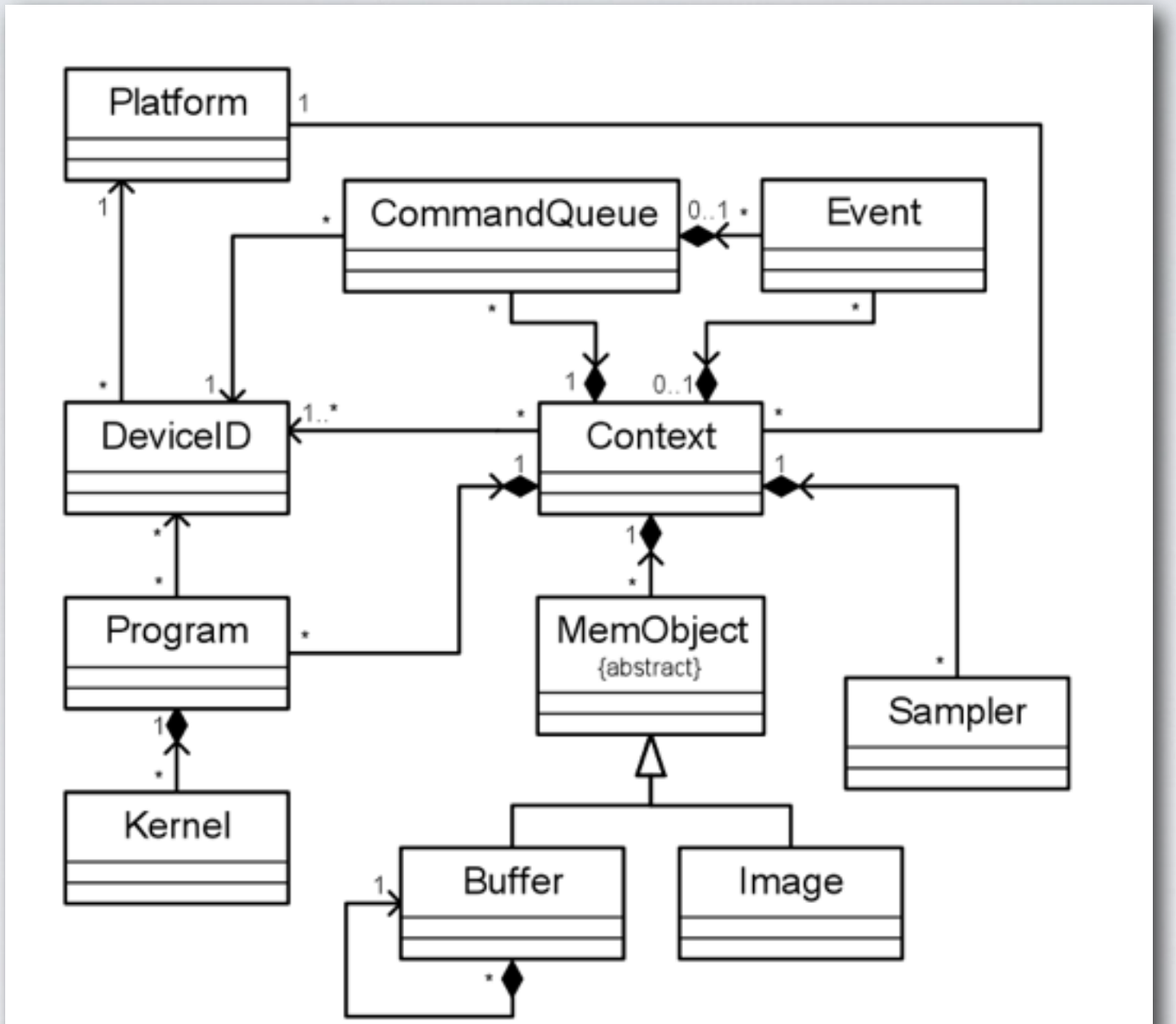


Figure 2.1 - *OpenCL UML Class Diagram*

MEMORY OBJECTS

- **Buffers** - One dimensional sequential memory structures
  - scalars, vectors, structs
- **Images** - 1D/2D/3D memory structures
  - frame buffer, images, use **built in** functions to address
  - channel format (RGB, RGBA + INT8/FLOAT, etc.)

EXECUTION

- **Programs** - libraries or modules of kernels
  - Compiled from Source or Binary
    - Can specify build options
- **Kernels** - "lowest" level of execution
  - Work Items are individual instances of a kernel
- **Events** - used for synchronization and coordination

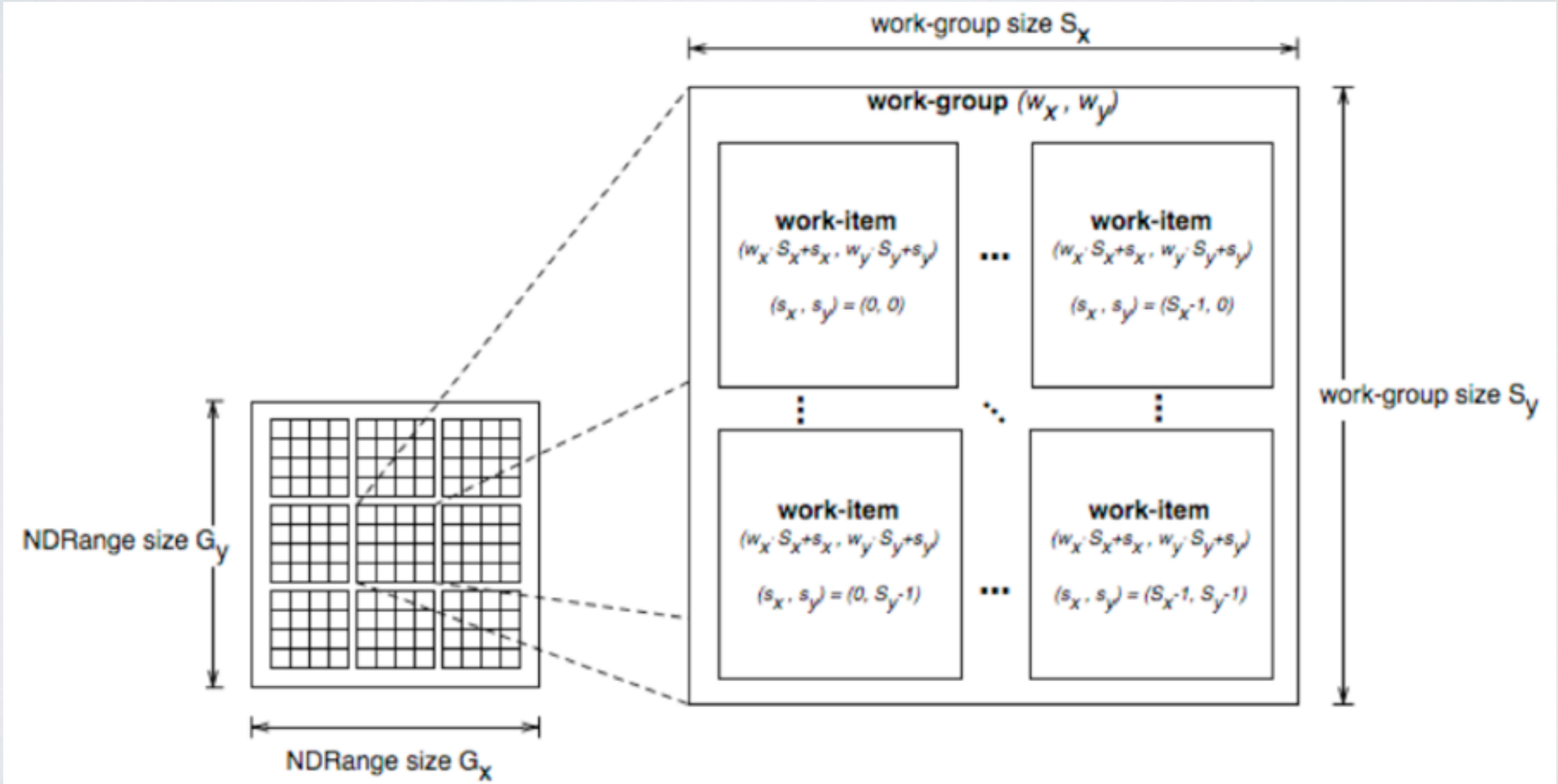
KERNELS

- A kernel instance is called a **work item**
- Work items are organized into **work groups**
  - # of work groups = Global Size / Work Group Size
- Addressing uses the following ids:
  - global, workgroup index, local

KERNEL INDEXES

- Index space defined when kernel submitted for execution
- **NDRange** - "N" Dimensional Range
- Used for defining both **global** and **local** work topologies

KERNEL DIMENSIONS



EVENTS

- Can refer to different execution commands and objects
  - enqueueX (kernel, memory, etc.)
  - track execution status
  - profiling

QUEUES

- Mechanism for performing operations on:
  - memory
  - program & kernel
- Allow multiple operations without explicit synchronization

DATA TYPES

- Standard C scalars
  - Most also have CL versions
  - CL Vector and CL String

VECTORS

- **Vector** types (2,3,4,8,16)
  - char*n*, int*n*, float*n*, etc.
  - Literal: (uint3)(1) => (1,1,1)
  - (ufloat2)(1.0f, 2.3f)
- Address up to 4 with x,y,z,w
  - float2 point; point.x = 2.4f; point.y = 3.3f;

BUILT-IN FNS: WORK ITEMS

- get_work_dim()
- get_global_size()
- get_global_id()
- get_local_size()
- get_local_id()
- get_num_groups()
- get_group_id()

BUILT-IN FNS: MATH

- trig functions (float)
  - some pi operators, hypotenuse
- exponents, logs in bases of *e*, 2, 10
- fmin/fmax, modulus, min/max magnitude
- Some native versions of these, e.g: native_cos

BUILT-IN FNS: COMMON

- clamp on interval
- min/max
- mixing [x + (y-x)*a]
- degree/radian
- distance, length, vector normalization, cross product

IN-KERNEL SYNC

- Makes use of Memory Barriers
  - **CLK_LOCAL_MEM_FENCE**
  - **CLK_GLOBAL_MEM_FENCE**
- barrier(cl_mem_fence *fence_flags)*
- mem_fence (store/load), read_mem_fence, write_mem_fence

ATOMIC OPERATIONS

- Capability for safe modification of global or local memory
- atomic_inc, atomic_dec, atomic_xchg, etc.
- Think about the consequences!

PROGRAM STRUCTURE

- **Platform Layer**
  - Query Compute Devices
  - Create Contexts
- **Runtime**
  - Create Ctx/Memory
  - Compile Kernels
  - Create/Use Queues
  - Sync Resources

SETUP ENVIRONMENT

```
cl::vector<cl::Platform> platforms;
cl::vector<cl::Device> devices;
cl::Device device;

// Select first platform
cl::Platform::get(&platforms);
assert (platforms.size() > 0);

// Create context properties for the first platform
cl_context_properties ctx_props[3] = {
    CL_CONTEXT_PLATFORM,
    (cl_context_properties)(platforms[0])(),
    0
};

// Create GPU Context with properties
cl::Context context(CL_DEVICE_TYPE_GPU, ctx_props);
devices = context.getInfo<CL_CONTEXT_DEVICES>();
assert (devices.size() > 0);

// Create a command queue using first device
cl::CommandQueue queue = cl::CommandQueue(context, devices[0]);
```

CREATE PROGRAM

```
 // Read source file and compile program
std::ifstream kernelFile("/Users/jsmith/projects/OpenCL/opencl/identity/
identity_kernel.cl");
assert (kernelFile.is_open());

std::string prog(std::istreambuf_iterator<char>(kernelFile),
  (std::istreambuf_iterator<char>()) );

cl::Program::Sources source(1, std::make_pair(prog.c_str(), prog.length()+1));
cl::Program program(context, source);

try {
  program.build(devices);
}
catch (cl::Error err) {
  cl::STRING_CLASS buildlog;
  buildlog = program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(devices[0]);
  std::cout << buildlog << std::endl;
  exit(-1);
}
```

CREATE KERNEL

```
// Construct Kernel
cl::Kernel kernel(program, "identity");

// Set arguments to kernel
kernel.setArg(0, devA);
kernel.setArg(1, N);
kernel.setArg(2, lsize);

// Create memory buffers
cl::Buffer devA = cl::Buffer(context, CL_MEM_READ_WRITE, N * sizeof(int));

int gsize = 64;
int lsize  = 64;
```

RUN KERNEL

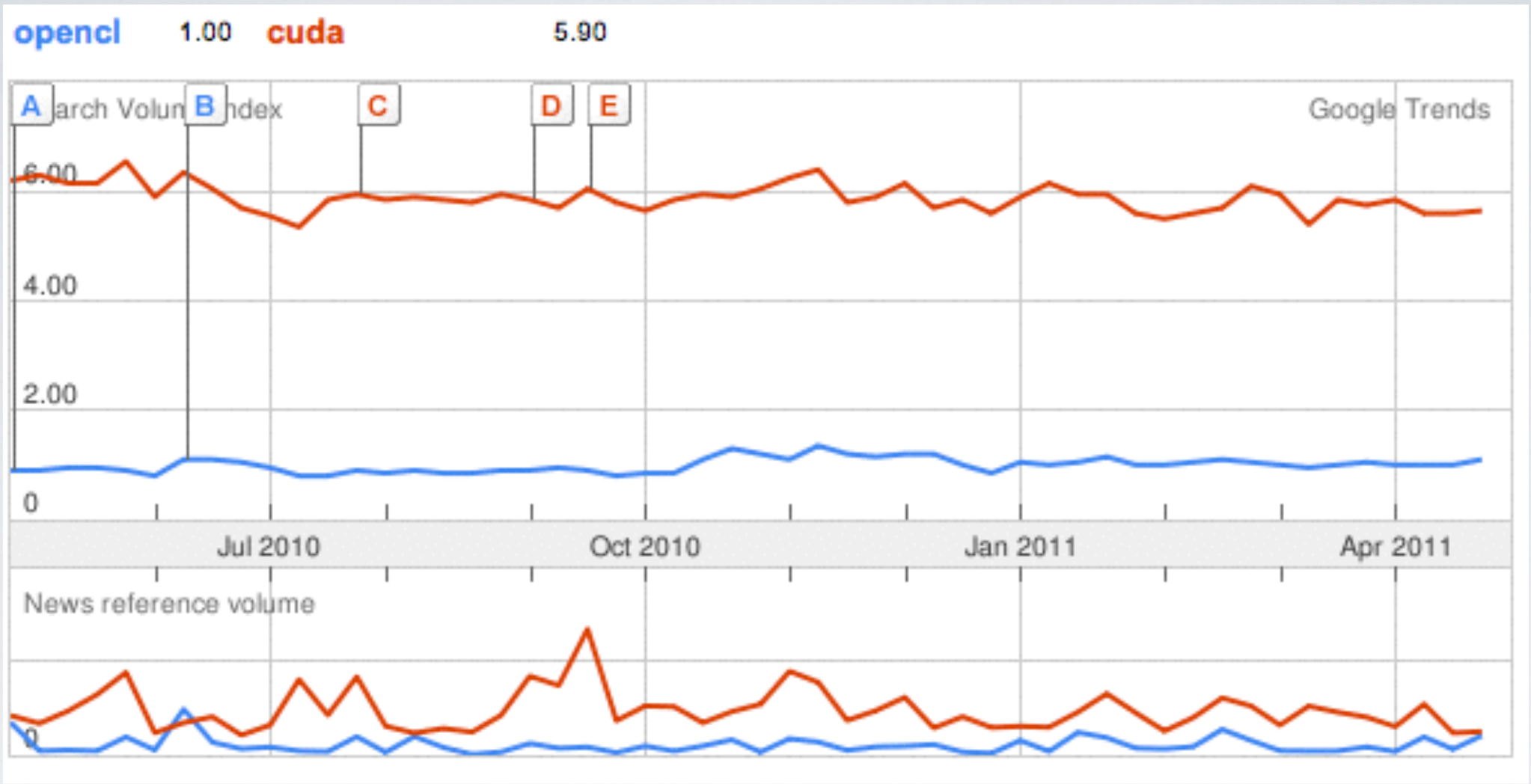
```
int gsize = 64;
int lsize  = 64;

// Run kernel on ND range
cl::NDRange global(gsize);
cl::NDRange local(lsize);

queue.enqueueNDRangeKernel(kernel, cl::NullRange, global, local);


// Read buffer into local, blocking
int *A = new int[N];
queue.enqueueReadBuffer(devA, CL_TRUE, 0, N * sizeof(int), A);
```

SIMPLE KERNEL

```
__kernel void identity(__global int *A, __global int N, __global int m)
{
   int i = get_global_id(0);
   if (i >= N) return;
   A[i] = i*m;
}
```

SUMMING UP

- OpenCL provides an environment for **portable** and **heterogenous** parallel processing
- Documentation is still sparse:
  - Use the OpenCL C API as a guide for function invocation regardless of the interface you are using


opencl 1.00 cuda 5.90
A
B
C
D
E
Google Trends
6.00
4.00
2.00
0
Jul 2010
Oct 2010
Jan 2011
Apr 2011
News reference volume
0

POPULARITY?

FINAL THOUGHTS

- Other implementations and competitors showing up!
  - Intel OpenCL, GPUOcelot, others?
- Cross platform is the main feature
  - Can it compete and gain mindshare vs. CUDA?