Introduction to OpenCL™

PPAM 2009

Dominik Behr | September 15th, 2009

What is OpenCL™

Open Computing Language

OpenCL™ is open, royalty-free standard for parallel programming of heterogenous computing systems.

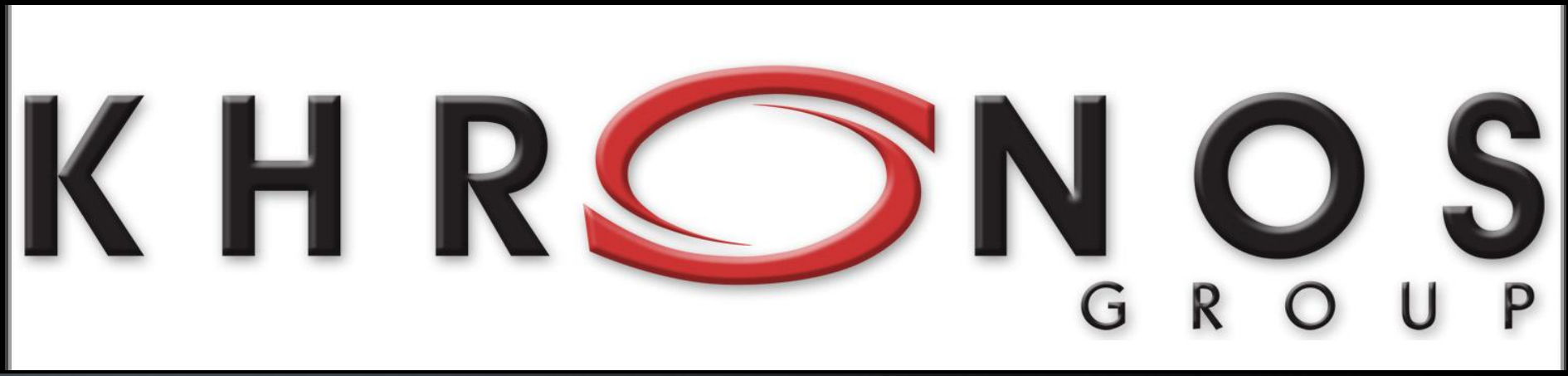
OpenCL spans range of applications starting from embedded devices to HPC solutions.

OpenCL standard defines the host API and the programming language.

Developed by Apple and the Khronos Group

The Khronos Group

http://www.khronos.org/



- The Khronos Group is an industry consortium
- Creating open standards
- Authoring and acceleration of parallel computing, graphics and dynamic media
- Variety of platforms and devices.

OpenCL™ Working Group

- Initially proposed by Apple, serving as specification editor
- Wide industry participation – hardware vendors, OEMs, middleware vendors, application developers
- Here are some of the companies in the OpenCL working group:

3DLABS, Activision Blizzard, AMD, Apple, ARM, Broadcom, Codeplay, Electronic Arts, Ericsson, Freescale, Fujitsu, GE, Graphic Remedy, HI, IBM, Intel, Imagination Technologies, Los Alamos National Laboratory, Motorola, Movidia, Nokia, NVIDIA, Petapath, QNX, Qualcomm, RapidMind, Samsung, Seaweed, S3, ST Microelectronics, Takumi, Texas Instruments, Toshiba and Vivante

What can you do with OpenCL™?

Write accelerated portable code across different devices and architectures.

Make use of CPUs, GPUs and other processors like DSPs or Cell BE to accelerate parallel computations.

Enable dramatic speedups for computationally intensive applications.

OpenCL™ Specification and Implementations

- Version 1.0 ratified on December 8th 2008

- Available at Khronos registry

http://www.khronos.org/registry/cl/

- Multiple implementations becoming available

- Shipped in Mac OS X 10.6 Snow Leopard

- Many software vendors working on applications and libraries using OpenCL

Design of OpenCL™

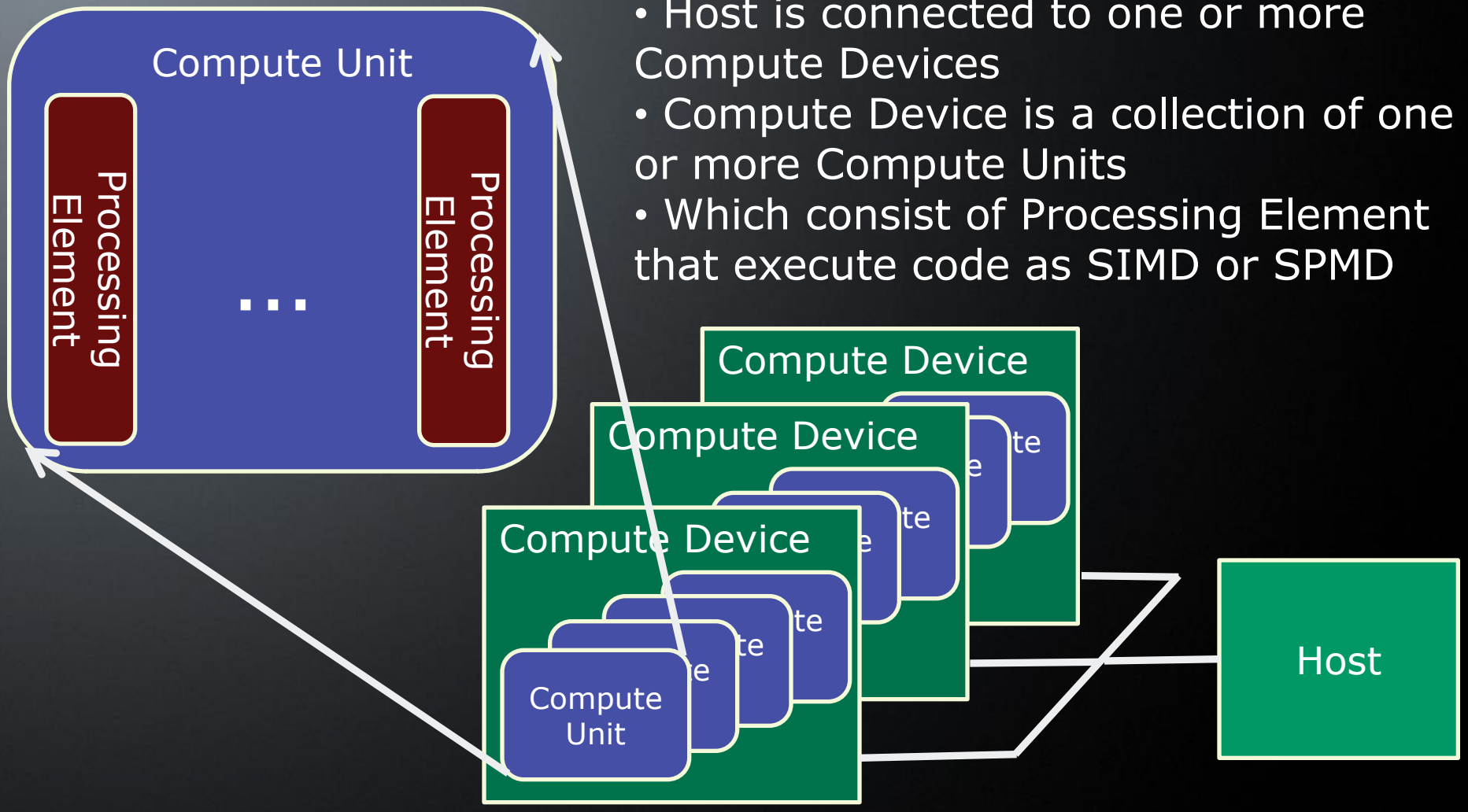
Host

- personal computer, embedded system, super computer
- provides OpenCL API and compiler
- C/C++
- bindings for other languages being created

Compute Device

- CPU, GPU, DSP
- executes OpenCL kernels
- kernels written in OpenCL language
- language based on C99

OpenCL™ Platform Model



- Host is connected to one or more Compute Devices
- Compute Device is a collection of one or more Compute Units
- Which consist of Processing Element that execute code as SIMD or SPMD

OpenCL™ Execution Model

- Kernel
  - Equivalent to C function executed on Compute Device.
  - Entry point, arguments, no return value
- Program
  - Collection of kernels and functions
  - Equivalent to a dynamically loaded library
- Command Queue
  - Enqueues kernel invocations and other OpenCL commands (like memory map/unmap/copy)
  - Enqueue in order
  - Execute in or out-of order (optionally)
- Event
  - Synchronize execution within and between queues in a context

OpenCL™ Execution Model - NDRange

Data parallel execution:

- Kernels executed across 1, 2 or 3 dimensional two-level index space called NDRange.
- Kernels are instanced as work-items ("threads") that are grouped in work-groups.
- No synchronization between work-groups, they are independent
- Barriers for synchronizing work-items within work-group
- Choose NDRange appropriate for your problem dimensions

OpenCL™ Execution Model - NDRange

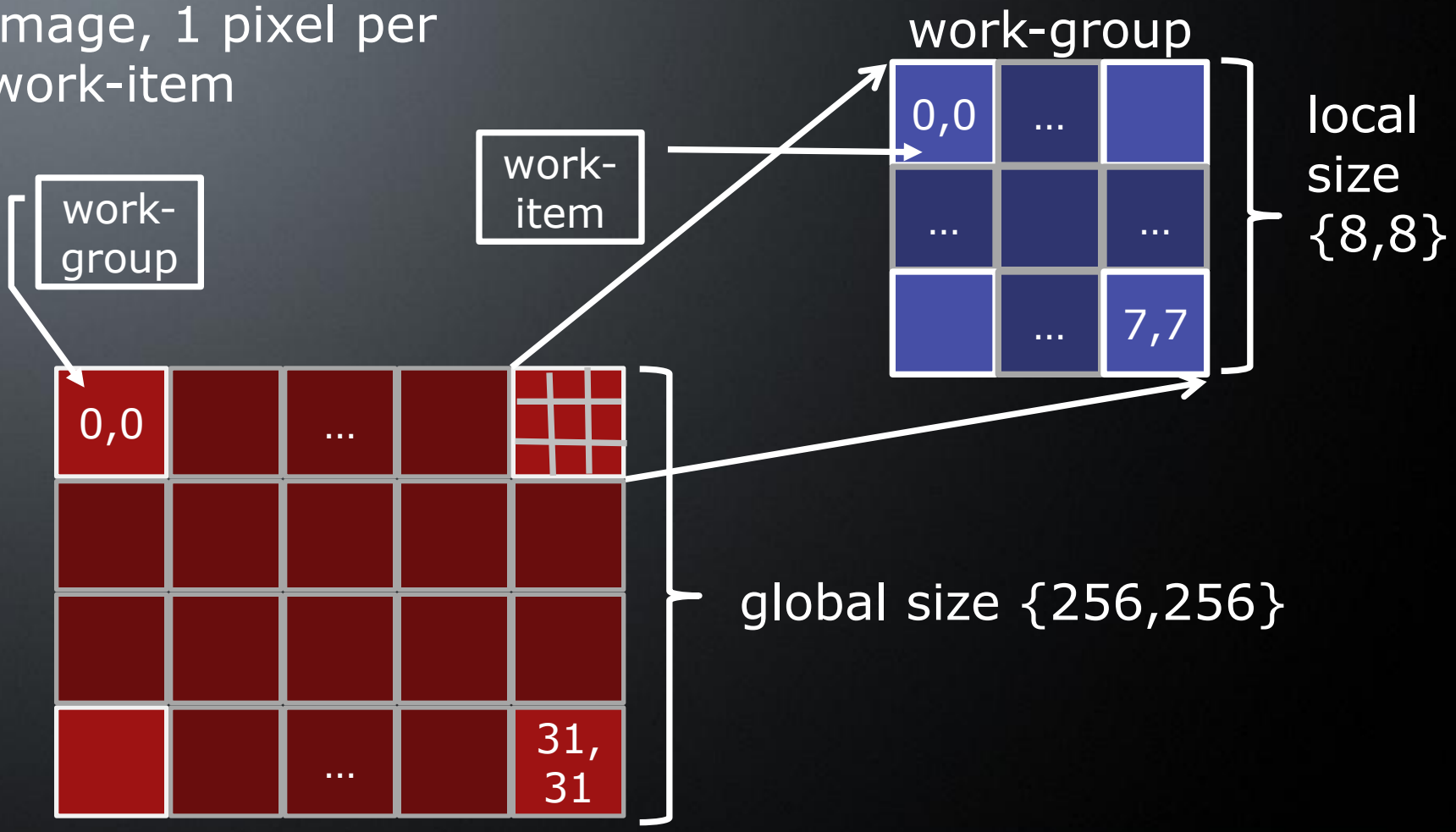
Process 256x256 image, 1 pixel per work-item

OpenCL™ Execution Model - NDRange

Process 256x256 image, 1 pixel per work-item

OpenCL™ Execution Model - NDRange

Process 256x256 image, 1 pixel per work-item



OpenCL™ Execution Model - NDRange

- Each work-item is given the same arguments but has unique local ID within group, and unique global ID.
- Each work-group has unique group ID.
- IDs and sizes are available via get_() functions.
- Global size is multiple of local size.

num_groups * local_size = global_size

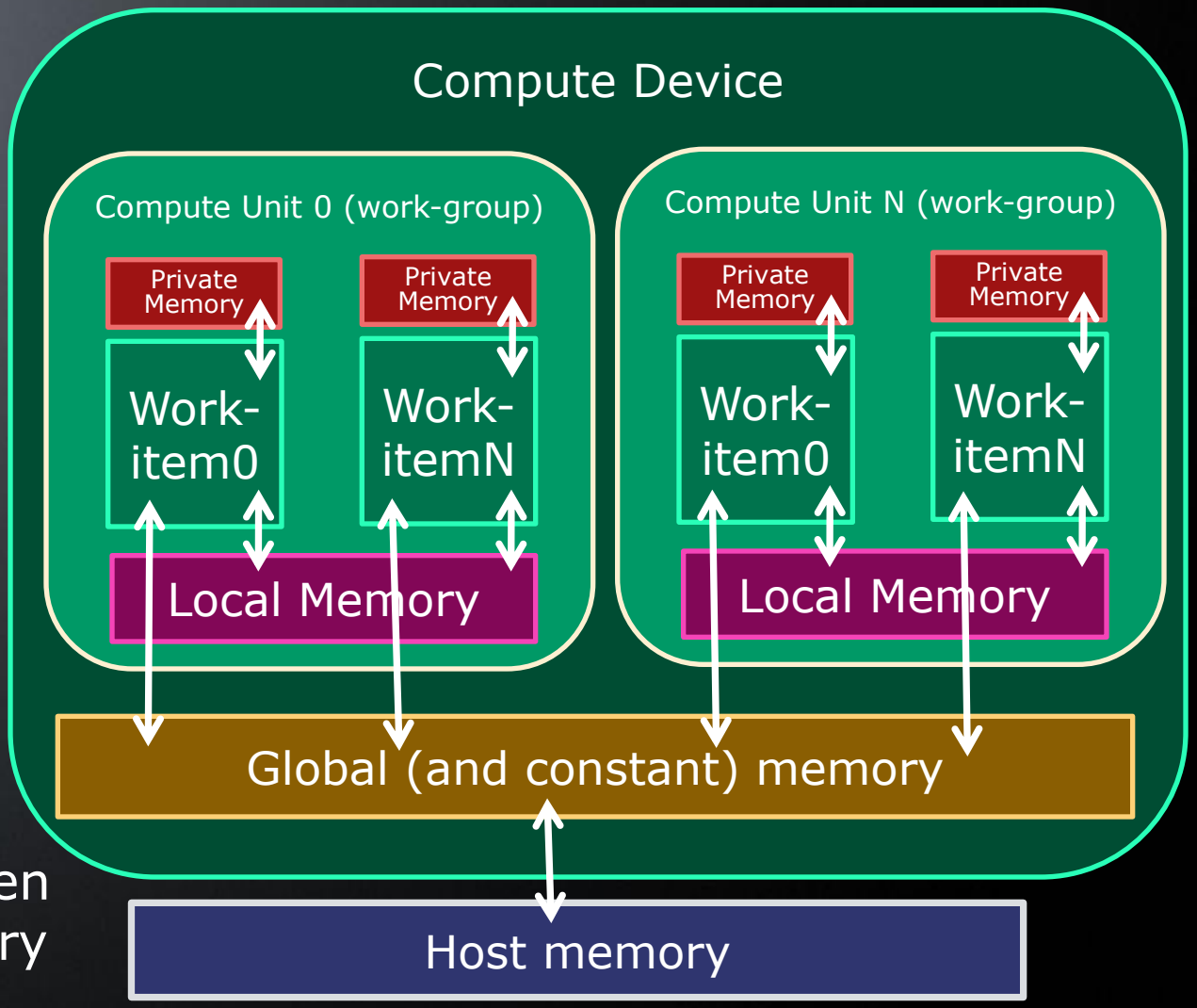
local_id + group_id * local_size = global_id

global_size % local_size = 0

OpenCL™ Memory Model

- Private Memory – accesible only by work-item on Processing Element
- Local Memory – accesible by every work-item in a work-group
- Global Memory – accessible by every work-group, persistent

Runtime copies between host and global memory



OpenCL™ Language

OpenCL language is based on C99 with limitations and extensions.

- Limitations: no recursion, no C99 headers, no bit fields, no function pointers, no variable length arrays, no byte addressable stores
- Extensions: vector types, work-items and work-groups, synchronization, address space qualifiers, image access functions, conversion and other built in functions

OpenCL™ Language: memory and synchronization

Private memory is like stack or TLS.

Local memory can be accessed and is shared by all work-items in a work-group.

Synchronization via `barrier(CLK_LOCAL_MEM_FENCE|CLK_GLOBAL_MEM_FENCE);`

Will ensure all stores are commited to local | global memory and program counter of all work-items in the work-group reached the barrier.

Simple example – Vector add

C function

```
void
vector_add(float *a,
float *b, float *c,
size_t n)
{
  size_t i;
  for(i = 0; i < n;i++)
    c[i] = a[i] + b[i];
}
```

Simple example – Vector add

C function

```
void
vector_add(float *a,
float *b, float *c,
size_t n)
{
  size_t i;
  for(i = 0; i < n;i++)
    c[i] = a[i] + b[i];
}
```

The loop becomes NDRange

The contents become the kernel

Simple example – Vector add

C function

```
void
vector_add(float *a,
float *b, float *c,
size_t n)
{
  size_t i;
  for(i = 0; i < n;i++)
    c[i] = a[i] + b[i];
}
```

OpenCL™ kernel

```
__kernel void
vector_add(
__global float *a,
__global float *b,
__global float *c)
{
  size_t i;
  i = get_global_id(0);
  c[i] = a[i] + b[i];
}
```

API: Platforms, Devices and Contexts

- Installed OpenCL™ runtime library may provide more than one platform, possibly from multiple vendors.
- Devices queried from the platform.
- Contexts are created using one or more devices.
- Other OpenCL objects created in the context.

API: Memory and Programs

• Memories (buffers, images) are created and replicated on all devices in context.

• Initialize and access using host pointer, map/unmap, read/write/copy.

• Programs created from sources or binaries. Compiled for devices.

API: Queues, Commands, Events and Synchronization

• OpenCL™ commands are sent to devices via command queues. More than one queue per device is possible.

• Almost every enqueued command can wait on list of events and produce an event too.

• Events can be only used in context in which they were created. Can be used in other queues.

• Flush, Finish, WaitForEvents

• Marker, WaitForEvents , Barrier

Barrier,

OpenCL™ Example

- Enumerate platforms
- Enumerate devices
- Create context
- Create command queue
- Create program
- Allocate and initialize memory
- Set arguments and enqueue kernel
- Sync
- Read results
- Clean up

Anatomy of a simple OpenCL™ program

Enumerate platforms:

```
clGetPlatformIDs(uPlatforms, &Platform, &uPlatforms);
```

Enumerate devices:

```
clGetDeviceIDs(Platform, CL_DEVICE_TYPE_CPU, uDevices, &Device, &uDevices);
```

Create context:

```
Context = clCreateContext(0, uDevices, &Device, NULL, NULL, &iErr);
```

Create command queue

```
Queue = clCreateCommandQueue(Context, Device, 0, &iErr);
```

Anatomy of a simple OpenCL™ program

Create program and kernel

```
 Program = clCreateProgramWithSource(Context, 1,
&pszProgram, &uProgramSize, &iErr);

    iErr = clBuildProgram(Program, 0, NULL, "",
NULL, NULL);

    Kernel = clCreateKernel(Program,
"vector_add", &iErr);
```

Anatomy of a simple OpenCL™ program

Allocate and initialize memory

```
BufA = clCreateBuffer(Context,
CL_MEM_READ_ONLY|CL_MEM_ALLOC_HOST_PTR, uNumbers
* sizeof(cl_int), NULL, &iErr);

 pA = (cl_int *)clEnqueueMapBuffer(Queue, BufA,
CL_TRUE, CL_MAP_WRITE, 0, uNumbers *
sizeof(cl_int), 0, NULL, NULL, &iErr);
```

… initialize contents of BufA

```
iErr = clEnqueueUnmapMemObject(Queue, BufA,
(void *)pA, 0, NULL, NULL);
```

… other ways to initialize memory

Anatomy of a simple OpenCL™ program

Set arguments

```
clSetKernelArg(Kernel, 0, sizeof(BufA), (void *)&BufA);

clSetKernelArg(Kernel, 1, sizeof(BufB), (void *)&BufB);

clSetKernelArg(Kernel, 2, sizeof(BufC), (void *)&BufC);
```

Invoke kernel

```
size_t uLocalSize = (uNumbers > uMaxWorkGroupSize)?uMaxWorkGroupSize:uNumbers;

size_t uGlobalSize = uNumbers;

clEnqueueNDRangeKernel(Queue, Kernel, 1, NULL, &uGlobalSize, &uLocalSize, 0, NULL, NULL);
```

Anatomy of a simple OpenCL™ program

Wait and read results

```
pC = (cl_int *)clEnqueueMapBuffer(Queue, BufC,
CL_TRUE, CL_MAP_READ, 0, uNumbers *
sizeof(cl_int), 0, NULL, NULL, &iErr);
```

... access results at *pC

```
clEnqueueUnmapMemObject(Queue, BufC, (void *)pC,
0, NULL, NULL);
```

and clean up

```
clReleaseMemObject(BufA); clReleaseMemObject(BufB);
clReleaseMemObject(BufC); clReleaseCommandQueue(Queue);
clReleaseProgram(Program); clReleaseKernel(Kernel);
clReleaseContext(Context);
```

Multi-Core x86 CPU implementation available NOW!

http://developer.amd.com/streambeta

Submitted for conformance during SIGGRAPH for Microsoft® Windows® 32-bit, Linux® 32-bit, and Linux® 64-bit

Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation in the United State and/or other jurisdictions. OpenCL is trademark of Apple Inc. used under license to the Khronos Group Inc. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.