Introduction to OpenCL

GoCode Thessaly #3

What is OpenCL

- Open Computing Language
- OpenCL 1.0 released on 8/12/08, by Khronos OpenCL working group (www.khronos.org)
- Parallel programming of **heterogeneous** systems
- Goal: Use **all** computational resources in a system

OpenCL API

- Platform Layer
  - Select devices, initialization, contexts, work-queues
- Runtime
  - Launch **kernels**, memory management, scheduling etc
- Compiler
  - C99 based, execute **online** or **offline** program executable

OpenCL program structure

- Host program
  - Create memory objects, command queues, compile and create kernel objects, create contexts, clean up etc.
- Kernel
  - Runs on device

What is a kernel

- The part of executable code running on a **device**

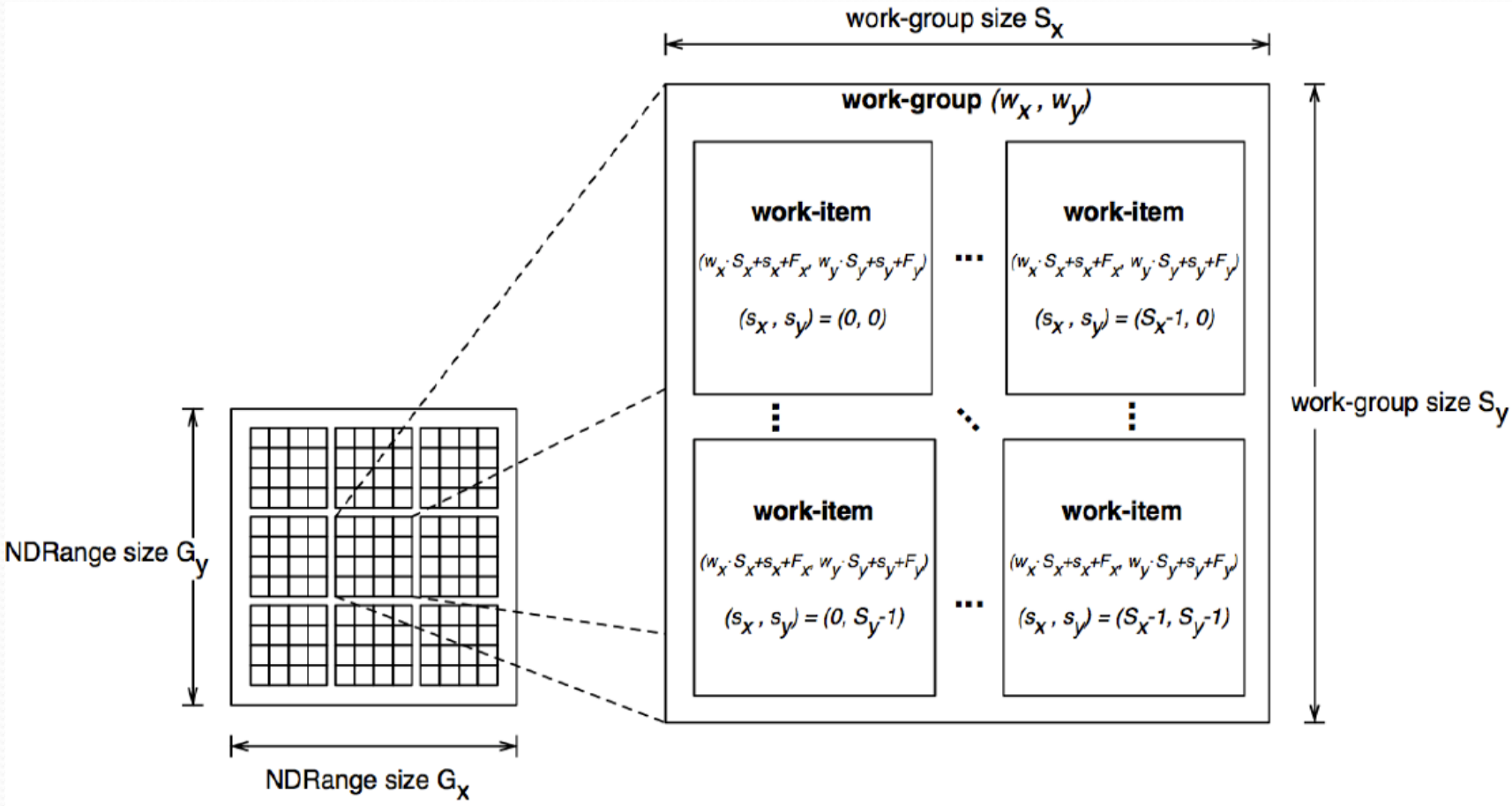
- Similar to a C function

```
12  // OpenCL Kernel Function for element by element vector addition
13 __kernel void VectorAdd(__global const float* a, __global const float* b, __global float* c, int iNumElements)
14 {
15     // get index into global data array
16     int iGID = get_global_id(0);
17
18     // bound check (equivalent to the limit on a 'for' loop for standard/serial C code
19     if (iGID >= iNumElements)
20     {
21         return;
22     }
23
24     // add the vector elements
25     c[iGID] = a[iGID] + b[iGID];
26 }
```

Kernel Execution

- **Host** program launches kernel in NDRange index space
  - Dimension N can be, 1-2-3
- Work item: A single kernel instance
  - Each work item executes same kernel on **different data**
  - **Unique global IDs**
- Work group: A group of work items
  - Unique Work Group IDs
  - Work items have a **unique local ID** within work group

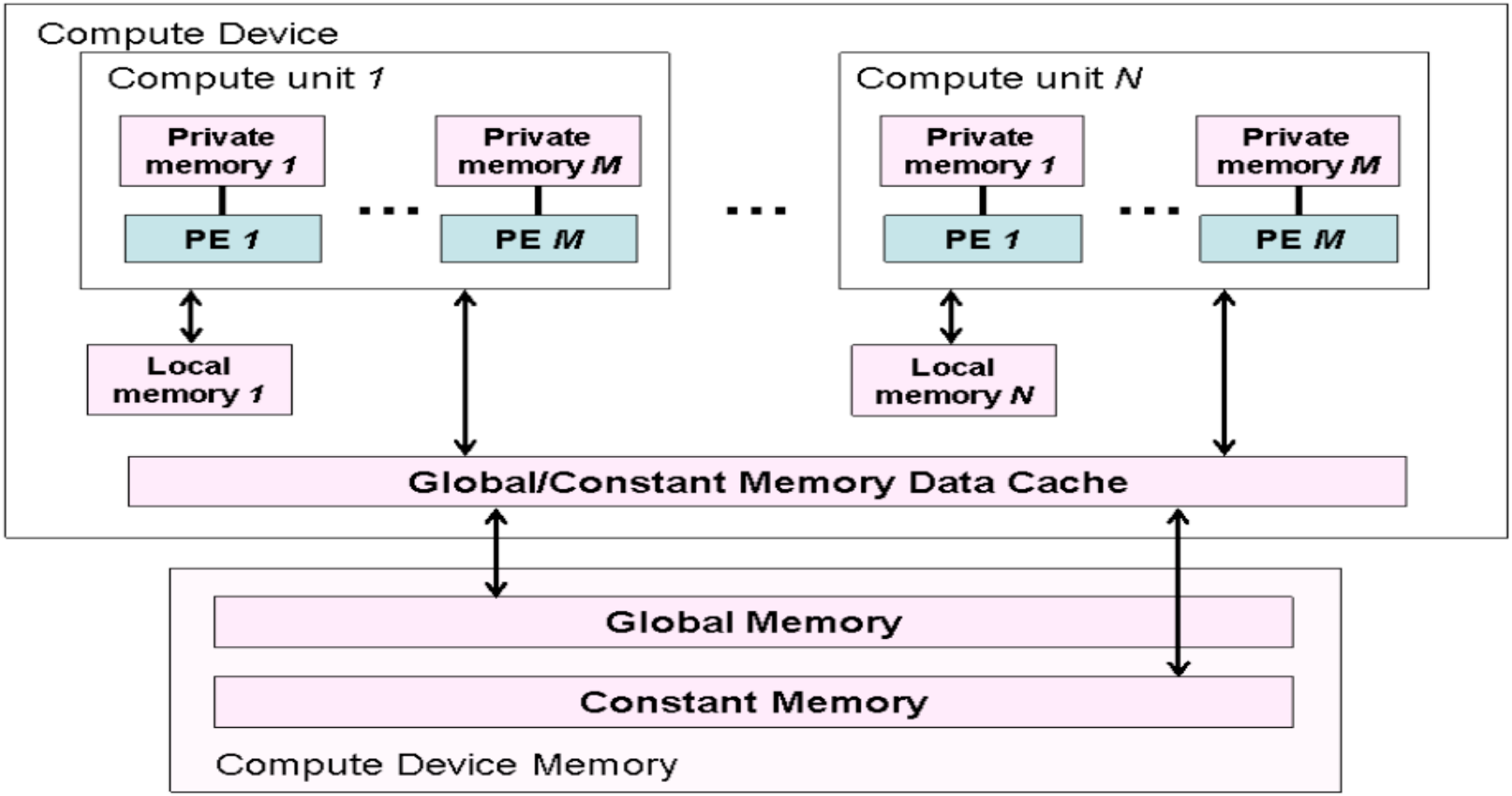
Kernel Execution



Memory Model

- Global Memory (R/W, _global)
  - Accessible from **all** work items in all work groups
- Constant Memory (R, _constant)
  - Remains **constant** during the kernel execution
- Local Memory (R/W, _local)
  - Accessible only from a **work group**.
- Private memory (R/W, _private)
  - Accessible only to a **work item**

Memory Model



Example: Vector Addition

- **Kernel Code:**

```
12  // OpenCL Kernel Function for element by element vector addition
13 __kernel void VectorAdd(__global const float* a, __global const float* b, __global float* c, int iNumElements)
14 {
15     // get index into global data array
16     int iGID = get_global_id(0);
17
18     // bound check (equivalent to the limit on a 'for' loop for standard/serial C code
19     if (iGID >= iNumElements)
20     {
21         return;
22     }
23
24     // add the vector elements
25     c[iGID] = a[iGID] + b[iGID];
26 }
```

- **In C (simple CPU code):**

```
int elements = 20000;
for ( i = 0; i < elements; i++ )
    C[i] = A[i] + B[i];
```

Example: Vector Addition (Host)

- **Declarations:**

```
40 // OpenCL Vars
41 cl_context cxGPUContext;          // OpenCL context
42 cl_command_queue cqCommandQueue;// OpenCL command que
43 cl_platform_id cpPlatform;        // OpenCL platform
44 cl_device_id cdDevice;           // OpenCL device
45 cl_program cpProgram;            // OpenCL program
46 cl_kernel ckKernel;              // OpenCL kernel
47 cl_mem cmDevSrcA;                // OpenCL device source buffer A
48 cl_mem cmDevSrcB;                // OpenCL device source buffer B
49 cl_mem cmDevDst;                 // OpenCL device destination buffer
50 size_t szGlobalWorkSize;         // 1D var for Total # of work items
51 size_t szLocalWorkSize;        // 1D var for # of work items in the work group
52 size_t szParmDataBytes;      // Byte size of context information
53 size_t szKernelLength;       // Byte size of kernel code
54 cl_int ciErr1, ciErr2;       // Error code var
55 char* cPathAndName = NULL;       // var for full paths to data, src, etc.
56 const unsigned char* cSourceCL = NULL;         // Buffer to hold source for compilation
57 const char* cExecutableName = NULL;
```

```
1  // set Local work size dimensions
2  szLocalWorkSize = 256;
3
4  // set Global work size dimensions (rounded up to the nearest multiple of LocalWork
5  // Size using C++ helper function)
6   szGlobalWorkSize = shrRoundUp ((int) szLocalWorkSize, iNumElements);
7
8   // Allocate host arrays
9   srcA = (void *) malloc (sizeof (cl_float) * szGlobalWorkSize);
10  srcB = (void *) malloc (sizeof (cl_float) * szGlobalWorkSize);
11  dst = (void *) malloc (sizeof (cl_float) * szGlobalWorkSize);
12
13  // Init host arrays using C++ helper functions
14  shrFillArray (( float*) srcA, iNumElements);
15  shrFillArray (( float *) srcB, iNumElements);
16
17  // Create the OpenCL context on a GPU device
18  cxGPUContext = clCreateContextFromType(0, CL_DEVICE_TYPE_GPU, NULL, NULL, NULL);
19
20  // Get the list of GPU devices associated with context
21  clGetContextInfo(cxGPUContext, CL_CONTEXT_DEVICES, 0, NULL, &szParmDataBytes);
22  cdDevices = (cl_device_id*) malloc (szParmDataBytes);
23  clGetContextInfo(cxGPUContext, CL_CONTEXT_DEVICES, szParmDataBytes,cdDevices, NULL);
24
25  // Create a command-queue
26  cqCommandQue = clCreateCommandQueue(cxGPUContext, cdDevices[0], 0, NULL);
27
28  // allocate the first source buffer memory object
29  cmDevSrcA = clCreateBuffer(cxGPUContext, CL_MEM_READ_ONLY,sizeof(cl_float) * iNumElements, NULL, NULL);
30  // allocate the second source buffer memory object
31  cmDevSrcB= clCreateBuffer(cxGPUContext, CL_MEM_READ_ONLY, sizeof(cl_float) * iNumElements, NULL, NULL);
32  // allocate the destination buffer memory object
33  cmDevDst = clCreateBuffer(cxGPUContext, CL_MEM_WRITE_ONLY, sizeof(cl_float) * iNumElements, NULL, NULL);
34
35  // Read the OpenCL kernel in from source file using helper C++ functions
36  cPathAndName = shrFindFilePath(cSourceFile, argv[0]);
37  cSourceCL = oclLoadProgSource(cPathAndName, "", &szKernelLength);
38
39
```

```
39  // Create the program
40  cpProgram = clCreateProgramWithSource(cxGPUContext, 1, (const char **)&cSourceCL, &szKernelLength, NULL);
41
42  // Build the program
43  clBuildProgram(cpProgram, 0, NULL, NULL, NULL, NULL);
44
45  // Create the kernel
46  ckKernel = clCreateKernel (cpProgram, "VectorAdd ", NULL);
47
48  // Set the Argument values
49  clSetKernelArg(ckKernel, 0, sizeof(cl_mem), (void*)&cmDevSrcA);
50  clSetKernelArg(ckKernel, 1, sizeof(cl_mem), (void*)&cmDevSrcB);
51  clSetKernelArg(ckKernel, 2, sizeof(cl_mem), (void*)&cmDevDst);
52  clSetKernelArg(ckKernel, 3, sizeof(cl_int), (void*)&iNumElements);
53
54  // Copy input data to GPU, compute, copy results ba ck
55  //  Runs asynchronous to host, up until blocking read at end
56
57  // Write data from host to GPU
58  clEnqueueWriteBuffer(cqCommandQue, cmDevSrcA, CL_FALSE, 0, sizeof(cl_float) * szGlobalWorkSize, srcA, 0, NULL, NULL);
59  clEnqueueWriteBuffer(cqCommandQue, cmDevSrcA, CL_FALSE, 0, sizeof(cl_float) * szGlobalWorkSize, srcB, 0, NULL, NULL);
60
61  // Launch kernel
62  clEnqueueNDRangeKernel(cqCommandQue, ckKernel, 1, NULL,&szGlobalWorkSize, &szLocalWorkSize, 0, NULL, NULL);
63
64  // Blocking read of results from GPU to Host
65  clEnqueueReadBuffer(cqCommandQue, cmDevDst, CL_TRUE, 0, sizeof(cl_float) * szGlobalWorkSize, dst, 0, NULL, NULL);
66
67  // Cleanup allocated objects
68  clReleaseKernel(ckKernel);
69  clReleaseProgram(cpProgram);
70  clReleaseCommandQueue(cqCommandQue);
71  clReleaseContext(cxGPUContext);
72  clReleaseMemObject(cmDevSrcA);
73  clReleaseMemObject(cmDevSrcB);
74  clReleaseMemObject(cmDevDst);
75  free (cdDevices);
76  free (cPathAndName);
77  free (cSourceCL);
78  // Free host memory
79  free(srcA);
80  free(srcB);
81  free (dst);
```

Is that all?

- Well..
  - Synchronization
  - Image objects (2d-3d)
  - Events
  - And now with OpenCL 2.0:
    - Shared Virtual Memory
    - Dynamic Parallelism
    - Android Client etc.

- Not in this presentation!

Why bother with OpenCL and parallel computing?

- Less power consumption
- Every machine is now parallel (cpus, gpus etc.)
- Moore's law can't go on forever
  - Fundamental limitations (c)

Links

- http://www.khronos.org/opencl/
- http://software.intel.com/en-us/vcsource/tools/opencl-sdk
- https://developer.nvidia.com/opencl