Introduction to OpenCL

Alexey A. Romanenko
arom@ccfit.nsu.ru
Novosibirsk State University

OpenCL (Open Computing Language)

* A standard based upon C for portable parallel applications.

* Task parallel and data parallel applications

* Focuses on multi platform support (multiple CPUs, GPUs, …)

* Development initiated by Apple.

* Developed by Khromos group who also managed OpenGL
* OpenCL 1.0 2008. Released with Max OS 10.6 (Snow Leopard)
* OpenCL 1.1 June 2010

* Similarities with CUDA.

OpenCL Timeline

- **Six months from proposal to released specification**
  - Due to a strong initial proposal and a shared commercial incentive to work quickly
- **Apple's Mac OS X Snow Leopard will include OpenCL**
  - Improving speed and responsiveness for a wide spectrum of applications
- **Multiple OpenCL implementations expected in the next 12 months**
  - On diverse platforms

Apple works with AMD, Intel, NVIDIA and others on draft proposal

Jun08

Apple proposes OpenCL working group and contributes draft specification to Khronos

OpenCL working group develops draft into cross-vendor specification

Oct08

Working Group sends completed draft to Khronos Board for Ratification

Khronos publicly releases OpenCL as royalty-free specification

Dec08

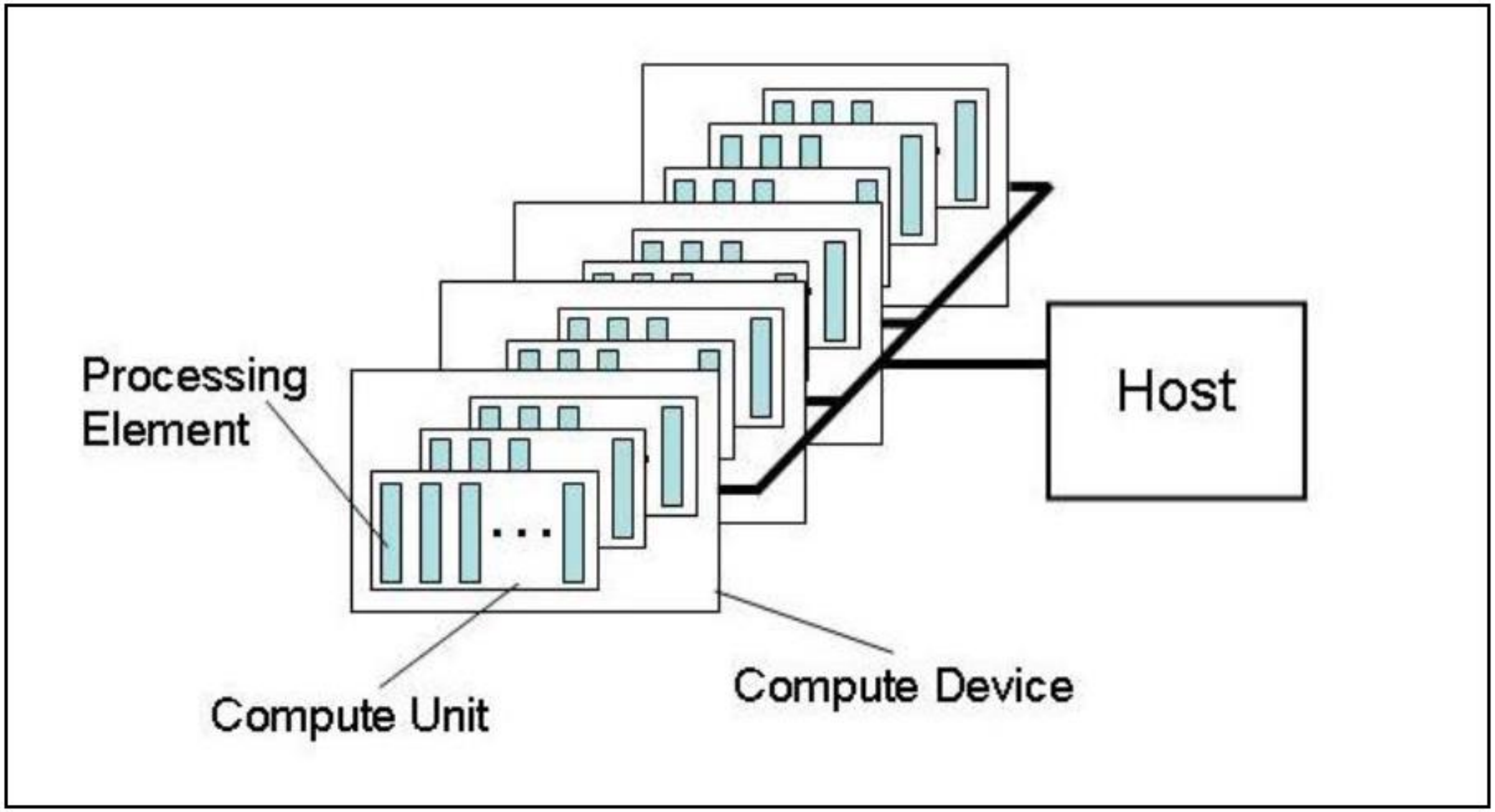
May09

Khronos to release conformance tests to ensure high-quality implementations

Модель OpenCL

- Platform Model
- Memory Model
- Execution Model
- Programming Model

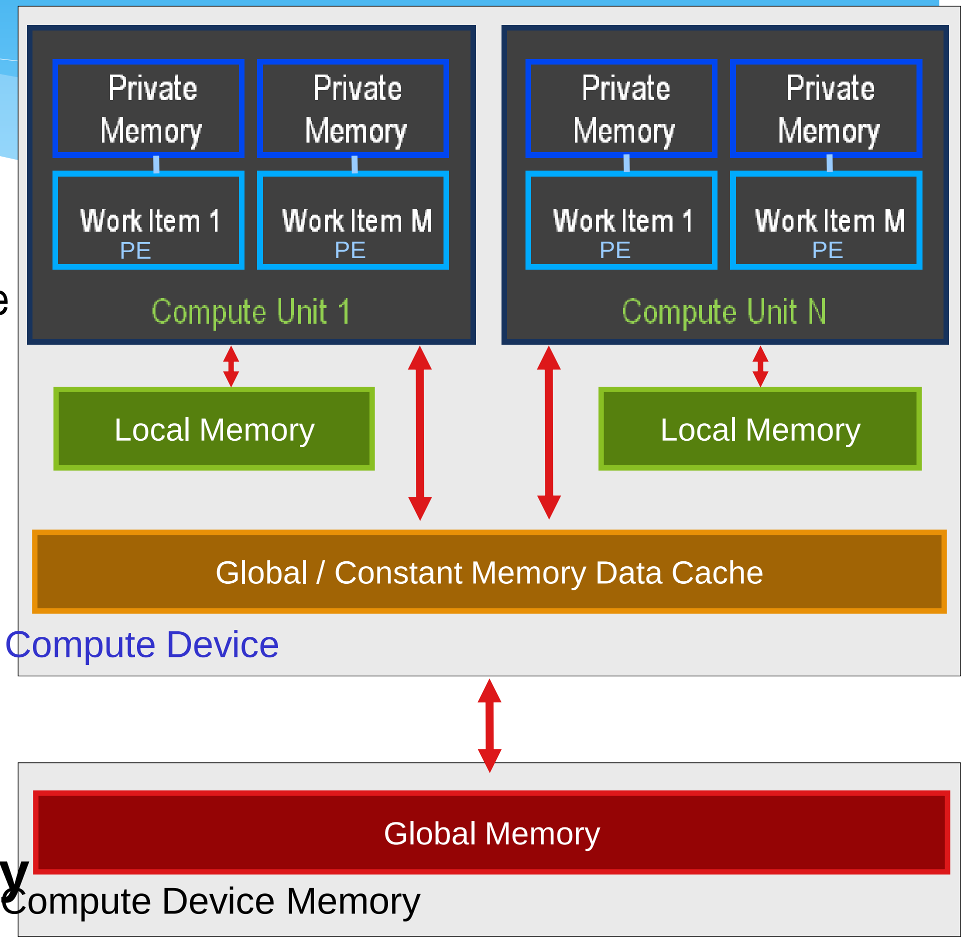
OpenCL Platform Model



- **One <u>Host</u> + one or more <u>Compute Devices</u>**
  - Each **Compute Device** is composed of one or more **Compute Units**
    - Each **Compute Unit** is further divided into one or more **Processing Elements**

OpenCL Memory Model

- **Shared memory model**
  - Relaxed consistency
- **Multiple distinct address spaces**
  - Address spaces can be collapsed depending on the device's memory subsystem
- **Address spaces**
  - Private - private to a *work-item*
  - Local - local to a *work-group*
  - Global - accessible by all work-items in all work-groups
  - Constant - read only global space
- **Implementations map this hierarchy**
  - To available physical memories



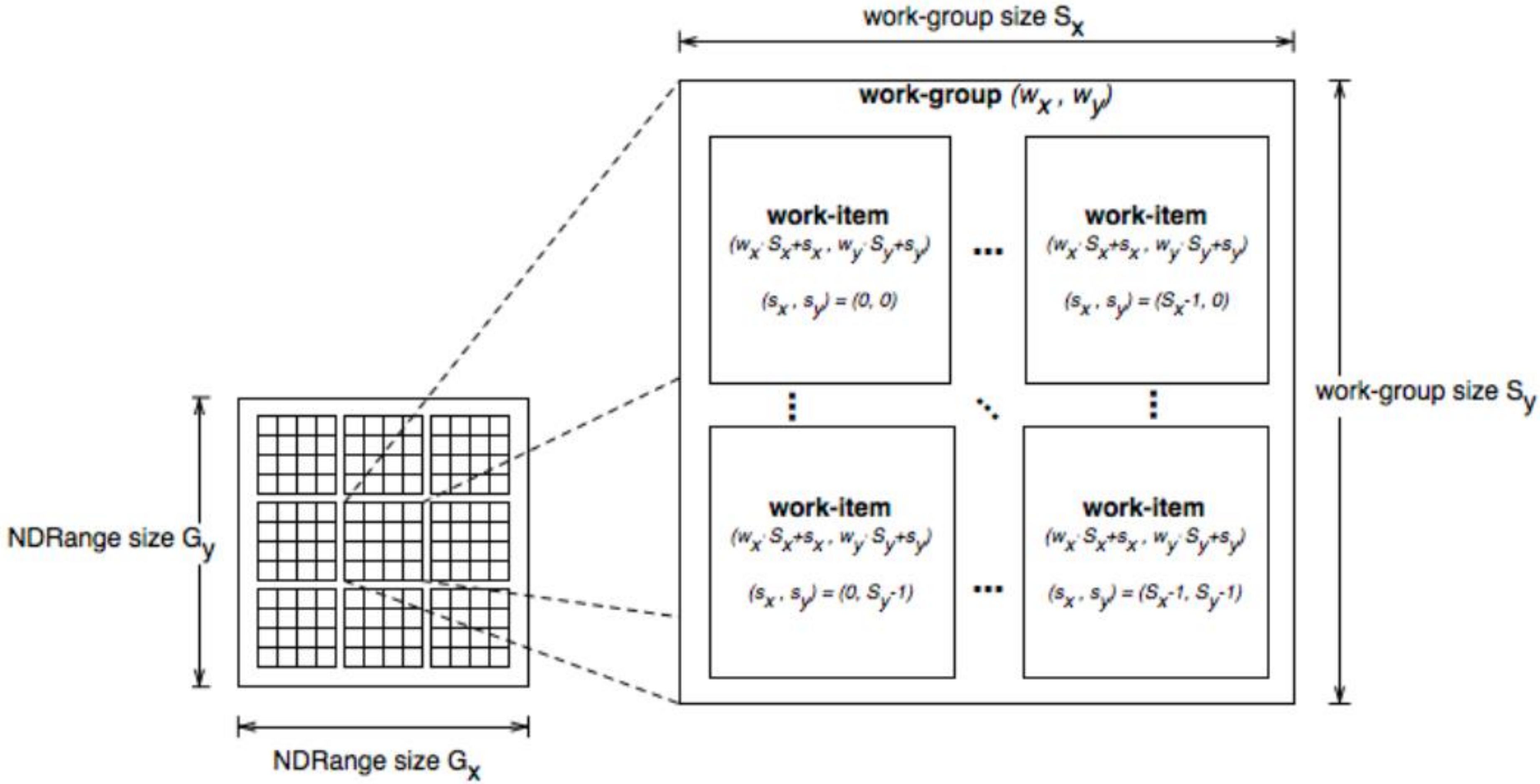
Memory Consistency

- **OpenCL uses a "relaxed consistency memory model"**
  - State of memory visible to a work-item ***not*** guaranteed to be consistent across the collection of work-items at all times
- **Memory has load/store consistency within a *work-item***
- **Local memory has consistency across work-items within a *work-group* at a barrier**
- **Global memory is consistent within a *work-group* at a barrier, but not guaranteed across different work-groups**
- **Memory consistency for objects shared between commands enforced at synchronization points**

OpenCL Execution Model

- **OpenCL *Program*:**
  - Kernels
    - Basic unit of executable code — similar to C functions, CUDA kernels, etc.
    - Data-parallel or task-parallel
  - Host Program
    - Collection of compute kernels and internal functions
    - Analogous to a dynamic library
- ***Kernel* Execution**
  - The host program invokes a kernel over an index space called an ***NDRange***
    - NDRange, “N-Dimensional Range”, can be a 1D, 2D, or 3D space
  - A single kernel instance at a point in the index space is called a ***work-item***
    - Work-items have unique global IDs from the index space
  - Work-items are further grouped into ***work-groups***
    - Work-groups have a unique work-group ID
    - Work-items have a unique local ID within a work-group

Kernel Execution



- **Total number of work-items = G_x * G_y**
- **Size of each work-group = S_x * S_y**
- **Global ID can be computed from work-group ID and local ID**

Contexts and Queues

- ***Contexts* are used to contain and manage the state of the "world"**

- **Kernels are executed in contexts defined and manipulated by the host**
  - Devices
  - Kernels - OpenCL functions
  - Program objects - kernel source and executable
  - Memory objects

- **Command-queue** - coordinates execution of kernels
  - Kernel execution commands
  - Memory commands: Transfer or map memory object data
  - Synchronization commands: Constrain the order of commands

- **Applications queue instances of compute kernel execution**
  - Queued in-order
  - Executed in-order or out-of-order
  - Events are used to synchronization execution instances as appropriate

Programming Model

Data-Parallel Model

- **Must be implemented by *all* OpenCL compute devices**
- **Define N-Dimensional computation domain**
  - Each independent element of execution in an N-Dimensional domain is called a *work-item*
  - N-Dimensional domain defines total # of work-items that execute in parallel
    = *global work size*
- **Work-items can be grouped together — *work-group***
  - Work-items in group can communicate with each other
  - Can synchronize execution among work-items in group to coordinate memory access
- **Execute multiple work-groups in parallel**
  - Mapping of global work size to work-group can be implicit or explicit

Programming Model

Task-Parallel Model

- **Some compute devices can also execute task-parallel compute kernels**

- **Execute as a *single* work-item**
  - A compute kernel written in OpenCL
  - A native C / C++ function

Basic OpenCL Program Structure

- **Host program**
  - Query compute devices
  - Create contexts
  - Create memory objects associated to contexts
  - Compile and create kernel program objects
  - Issue commands to command-queue
  - Synchronization of commands
  - Clean up OpenCL resources
- **Kernels**
  - C code with some restrictions and extensions

Platform Layer

Runtime

Language

OpenCL C Language Restrictions

- Pointers to functions not allowed
- Pointers to pointers allowed within a kernel, but not as an argument
- Bit-fields not supported
- Variable-length arrays and structures not supported
- Recursion not supported
- Writes to a pointer of types less than 32-bit not supported
- Double types not supported, but reserved
- 3D Image writes not supported

- Some restrictions are addressed through extensions

OpenCL vs. CUDA

- C for CUDA Kernel Code:

```
__global__ void
   vectorAdd(const float * a, const float * b, float * c){
       // Vector element index
       int nIndex = blockIdx.x * blockDim.x + threadIdx.x;
       c[nIndex] = a[nIndex] + b[nIndex];
   }
```

- OpenCL Kernel Code

```
__kernel void
   vectorAdd(__global const float * a,
             __global const float * b,
             __global       float * c){
       // Vector element index
       int nIndex = get_global_id(0);
       c[nIndex] = a[nIndex] + b[nIndex];
   }
```

Group and grid size in OpenCL

- `get_local_id()`
- `get_work_dim()`
- `get_global_size()`
- `get_global_id()`

OpenCL vs. CUDA. Initialization

- CUDA

```
cuInit(0);
cuDeviceGet(&hDevice, 0);
cuCtxCreate(&hContext, 0, hDevice);
```

- OpenCL

```
cl_context hContext;
  hContext = clCreateContextFromType(0, CL_DEVICE_TYPE_GPU,
                                      0, 0, 0);
  size_t nContextDescriptorSize;
  clGetContextInfo(hContext, CL_CONTEXT_DEVICES,
                   0, 0, &nContextDescriptorSize);
  cl_device_id * aDevices = malloc(nContextDescriptorSize);
  clGetContextInfo(hContext, CL_CONTEXT_DEVICES,
                   nContextDescriptorSize, aDevices, 0);
  cl_command_queue hCmdQueue;
  hCmdQueue = clCreateCommandQueue(hContext, aDevices[0],
  0,0);
```

OpenCL vs. CUDA. Creating kernel

- CUDA

```
CUmodule hModule;
cuModuleLoad(&hModule, “vectorAdd.cubin”);
cuModuleGetFunction(&hFunction, hModule, "vectorAdd");
```

- OpenCL

```
cl_program hProgram;
hProgram = clCreateProgramWithSource(hContext, 1,
                              sProgramSource, 0, 0);
clBuildProgram(hProgram, 0, NULL, NULL, NULL, NULL);

cl_kernel hKernel;
hKernel = clCreateKernel(hProgram, “vectorAdd”, 0);
```

Количество строк (→ 1)

Код ошибки (→ 0 in clCreateProgramWithSource)

Код ошибки (→ 0 in clCreateKernel)

OpenCL vs. CUDA. Memory allocation

- CUDA

```
CUdeviceptr pDevMemA, pDevMemB, pDevMemC;
cuMemAlloc(&pDevMemA, cnDimension * sizeof(float));
cuMemAlloc(&pDevMemB, cnDimension * sizeof(float));
cuMemAlloc(&pDevMemC, cnDimension * sizeof(float));
// copy host vectors to device
cuMemcpyHtoD(pDevMemA, pA, cnDimension * sizeof(float));
cuMemcpyHtoD(pDevMemB, pB, cnDimension * sizeof(float));
```

OpenCL vs. CUDA. Memory allocation

- OpenCL

```
cl_mem hDevMemA, hDevMemB, hDevMemC;
  hDevMemA = clCreateBuffer(hContext,
                            CL_MEM_READ_ONLY |
  CL_MEM_COPY_HOST_PTR,
                            cnDimension * sizeof(cl_float),
                            pA,
                            0);
hDevMemB = clCreateBuffer(hContext,
                          CL_MEM_READ_ONLY |
  CL_MEM_COPY_HOST_PTR,
                          cnDimension * sizeof(cl_float),
                          pA,
                          0);
hDevMemC = clCreateBuffer(hContext,
                            CL_MEM_WRITE_ONLY,
                            cnDimension * sizeof(cl_float),0, 0);
```

Код ошибки

Код ошибки

OpenCL vs. CUDA. Kernel parameters

- CUDA

```
cuParamSeti(cuFunction, 0,  pDevMemA);
cuParamSeti(cuFunction, 4,  pDevMemB);
cuParamSeti(cuFunction, 8,  pDevMemC);
cuParamSetSize(cuFunction,  12);
```

- OpenCL:

```
clSetKernelArg(hKernel, 0, sizeof(cl_mem),
                         (void *)&hDevMemA);
clSetKernelArg(hKernel, 1, sizeof(cl_mem),
                         (void *)&hDevMemB);
clSetKernelArg(hKernel, 2, sizeof(cl_mem),
                         (void *)&hDevMemC);
```

OpenCL vs. CUDA. Launching kernel

- CUDA

```
cuFuncSetBlockShape(cuFunction, cnBlockSize, 1, 1);
cuLaunchGrid       (cuFunction, cnBlocks, 1);
```

- OpenCL

```
clEnqueueNDRangeKernel(hCmdQueue, hKernel, 1, 0,
                &cnDimension, &cnBlockSize, 0, 0, 0);
```

OpenCL vs. CUDA. Copy result back

- CUDA

```
cuMemcpyDtoH((void*)pC, pDevMemC,
  cnDimension*sizeof(float));
```

- OpenCL

```
clEnqueueReadBuffer(hContext, hDeviceC, CL_TRUE, 0,
                    cnDimension * sizeof(cl_float),
                    pC, 0, 0, 0);
```

Release resourses

- OpenCL

```
clReleaseMemObject(hDevMemA);
clReleaseMemObject(hDevMemB);
clReleaseMemObject(hDevMemC);
free (aDevices);
clReleaseKernel (hKernel);
clReleaseProgram (hProgram);
clReleaseCommandQueue (hCmdQueue);
clReleaseContext (hContext);
```

Recourses OpenCL

- Khronos OpenCL Homepage
  http://www.khronos.org/opencl
- OpenCL 1.0 Specification
  http://www.khronos.org/registry/cl
- OpenCL at NVIDIA
  http://www.nvidia.com/object/cuda_opencl.html