OpenCL Workshop Language

Ljubljana, May 2019

OpenCL

- targets multi-core CPUs (desktop or embedded), GPUs, DSPs, FPGAs and other accelerators
- open standard, maintained by the Khronos Group
- notable non-scientific applications that use OpenCL
  - Adobe Photoshop, GIMP, ImageMagick
  - Autodesk Maya, Blender
  - Final Cut, OpenCV, FFmpeg
  - LibreOffice Calc
- the API is officially supported in C & C++
  - wrappers for Pyton, Julia, Java, ..
- the device code can be written in C or C++
  - C++ as of May 2017, still not widely supported

Resources

- This tutorial is based on the OpenCL C API ver 1.2
- OpenCL 1.2 specification: https://www.khronos.org/registry/OpenCL/specs/opencl-1.2.pdf
- Reference card: https://www.khronos.org/files/opencl-1-2-quick-reference-card.pdf
- Resources: https://github.com/bstatcomp/OpenCL-Workshop

The Platform layer

- Platform
    - A vendors OpenCL implementation
        - AMD: AMD/ATI GPUs & Intel CPUs
        - Intel: CPUs & GPUs
        - Nvidia: GPUs
    - enables us to list devices, select devices & their information
- Devices
    - represents actual devices inside a platform
    - used to create a context and command queues
- Contexts
    - manages command queues, memory objects (buffers), program, kernel objects, ...

The Platform layer - code

```
cl_platform_id platform_id = NULL;
cl_device_id device_id = NULL;
cl_uint ret_num_devices;
cl_uint ret_num_platforms;
// retrieve the platform and device
cl_int ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);
ret = clGetDeviceIDs( platform_id, CL_DEVICE_TYPE_GPU, 1, &device_id,
&ret_num_devices);
// create an OpenCL context
cl_context context = clCreateContext( NULL, 1, &device_id, NULL, NULL, &ret);
```

The Runtime layer - queues & buffers

- Command queue
  - the host enqueues actions to the command queue to execute them on the device
    - copying data to and from the devices
    - device code (kernels)
  -
- Buffers
  - represent memory objects on the OpenCL device
  - can be read-only, write-only or read/write
  - point to space allocated on the devices or the host memory space

The Runtime layer - code

```
// Create a command queue
cl_command_queue command_queue = clCreateCommandQueue(context, device_id, 0,
&ret);

// Create a memory buffer on the device
cl_mem a_mem_obj = clCreateBuffer(context, CL_MEM_READ_WRITE, LIST_SIZE *
sizeof(float), NULL, &ret);
```

Copying data

```
// copy the data from the CPU global memory to the GPU global memory
ret = clEnqueueWriteBuffer(command_queue, a_mem_obj, CL_TRUE, 0, LIST_SIZE *
sizeof(float), A, 0, NULL, NULL);

// copy the data to the CPU global memory from the GPU global memory
ret = clEnqueueReadBuffer(command_queue, c_mem_obj, CL_TRUE, 0, LIST_SIZE *
sizeof(float), C, 0, NULL, NULL);
```

Kernels

- functions that are executed on the OpenCL device

```
__kernel void kernel add_scalar(global float* A, const float b) {
   const int id = get_global_id(0);
   A[id] = A[id] + b;
}
```

Compiling kernels - code

```
// create a program from the kernel source
cl_program program = clCreateProgramWithSource(context, 1,
      (const char **)&source_str, (const size_t *)&source_size, &ret);

// build the program
ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);

// create the OpenCL kernel
cl_kernel kernel = clCreateKernel(program, "add_scalar", &ret);
```

Kernels - code

```
// set the arguments of the kernel
ret = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *)&a_mem_obj);
ret = clSetKernelArg(kernel, 1, sizeof(float), scalar);

// Execute the OpenCL kernel on the list
size_t global_item_size = LIST_SIZE;
size_t local_item_size = 64;
ret = clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL,
                             &global_item_size, &local_item_size, 0, NULL, NULL);
```

Kernel C language

- supported data types: char, short, int, long (signed/unsigned), float, double, half, size_t, void
- memory types: global, local, private
- operators: +, -, *, /, %, ++, --, <, >, >=, <=, bitwise operators, logical operators
- most C math functions are supported
- built-in functions:
    - get_global_id(int dimensionID), get local_id(int dimensionID), get_group_id(int dimensionID)
    - get_global_size(), get_local_size(), get_num_groups()
    - get_work_dim()
    - get_global_offset()

Kernel C language

- synchronization of threads in a block
  - barrier(CLK_LOCAL_MEM_FENCE), barrier(CLK_GLOBAL_MEM_FENCE)
- atomic operations
  - add, sub, xchg, inc, dec, min, max, or, xor