OpenCL

Hwansoo Han
Heterogeneous Computing

- Multiple, but heterogeneous multicores

- Use all available computing resources in system
  - Single core CPU, multicore CPU
  - GPUs, DSPs

- Parallel programming model supports both
  - Data parallelism
  - Task parallelism
OpenCL Platform

- One Host + Multiple Compute Devices
  - Each compute device consists of multiple compute units
  - Each compute unit consists of multiple processing elements
OpenCL Application (1)

- Host sends “commands” to devices via command queues
  - Data transfer between host memory and device memories
  - Execute “kernels” on devices
OpenCL Application (2)

- Serial code executed in a Host (CPU) thread
- Parallel code executed in many Device (GPU) threads
  - Multi-threading on multiple processing elements
Command Queue

- Host creates a queue for a device
  - Allow multiple queues per device
  - Host enqueues commands to execute on the device

- Command types
  - Kernel execution
  - Memory copy, map/unmap
  - Synchronization

- Command execution mode
  - In-order execution (default)
  - Out-of-order execution
OpenCL Memory Model

- Shared memory model
- Multiple distinct address space
  - Different address space for Host and Devices

- Address space for a Device
  - Private – private to a work-item
  - Local – local to work-group
  - Global – accessible by all work-items of all work-group
  - Constant – read only global space

- Relaxed memory consistency
  - Local/global memories are consistent among work-items only after work-group barrier
Kernel: Device Code (data-parallelism)

- N-dimensional computation domain (N = 1, 2, or 3)
  - Express data-parallelism
  - Each element of N-D computation domain (NDRange) is a work-item
  - N-D domain defines the total number of work-items that execute in parallel
    - E.g. Want to process 1024x1024 image (problem dimensions)
      - 2D domain with 1024x1024 work-items
      - 1M kernel executions in parallel
    - E.g 1D domain to multiply two arrays, a[n] and b[n]

Scalar

```c
void scalar_mul(int n,
    const float *a,
    const float *b,
    float *result)
{
    int i;
    for (i=0; i<n; i++)
        result[i] = a[i] * b[i];
}
```

Data Parallel

```c
kernel void dp_mul(global const float *a,
    global const float *b,
    global float *result)
{
    int id = get_global_id(0);
    result[id] = a[id] * b[id];
} // execute dp_mul over “n” work-items
```
Global and Local Dimensions

- Global dimensions
  - N-D computation domain (represents whole problem space)
- Local dimensions
  - Work-group dimensions (executed together on a compute unit)

Need to choose dimensions “best” suit your algorithm
Kernel Execution with NDRange

- clEnqueNDRangeKernel(cmd_Q, kernel, 2, NULL, gwsz, lwsz, 0, NULL, NULL)
  - gwsz[2] = {1024, 1024}; 2-dimension domain

- Total number of work-items = $G_x \times G_y$ (e.g., 1024 * 1024)
- Size of each work-group = $S_x \times S_y$ (e.g., 128 * 128)
- Each thread has own ID
  - get_global_id(), get_group_id(), get_local_id()
Kernel: Device Code (task-parallelism)

- Different, but parallel kernels are executed
  - Execute as a single work-item
  - No N-dimensional compute domain is needed
  - `clEnqueueTask(cmd_Q, kernel, 0, NULL, NULL)`
Memory Objects

- **Buffer objects**
  - 1-D data
  - Sequentially stored data on the memory
  - Kernel can access via pointers
  - Sub-buffer (OpenCL 1.1): a partial space from buffer object

- **Image objects**
  - 1-D (OpenCL 1.2), 2-D or 3-D data: texture, frame buffer, image
  - Elements are 4-component vectors from a list of predefined formats
  - Kernel can access via special built-in functions
Data Transfer: Host and Devices

- Create buffers for host and devices
  - Host buffer: C99 array or malloc’ed space
  - Device buffer (cl_mem type): created with clCreateBuffer()

- Enqueue buffer commands for read/write
  - clEnqueueWriteBuffer() : copy from host buffer to device buffer
  - clEnqueueReadBuffer() : copy from device buffer to host buffer

```c
// Create buffers on host and device
size_t size = 100000 * sizeof(int);
int* h_buffer = (int*)malloc(size);
cl_mem d_buffer = clCreateBuffer(context, CL_MEM_READ_WRITE, size, NULL, NULL);
...

// Write to buffer object from host memory
clEnqueueWriteBuffer(cmd_queue, d_buffer, CL_FALSE, 0, size, h_buffer, 0, NULL, NULL);
...

// Read from buffer object to host memory
clEnqueueReadBuffer(cmd_queue, d_buffer, CL_TRUE, 0, size, h_buffer, 0, NULL, NULL);
```
Synchronization

- Synch among work-items
  - Work-items among the same work-group
  - void barrier(flags)
    - Memory fence flags : CLK_LOCAL_MEM_FENCE, CLK_GLOBAL_MEM_FENCE

- Synch among work-groups
  - Not supported

- Synch among commands
  - Inside a context, synch among commands within a queue (or queues)
  - Command queue barrier or event synchronization
Synch: work-group barrier

- Call barrier() within a kernel
  - Each work-item should call barrier()
  - After barrier(), memory consistency is satisfied

Without barrier(), work-item n cannot guarantee to see the update from work-item n+1
Synch: command-queue barrier

- Single queue command synch
  - Host enqueues a queue-barrier command
    - `clEnqueueBarrierWithWaitList()`
  - All commands before the barrier must be completed
  - All memory object updates before the barrier must be completed
Synch: event wait

- Sync among commands in multiple queues
  - But commands and queues should be within the same context
  - All clEnqueue*() APIs have parameters for `event_wait_list` and `event`
    - `event_wait_list`: list of events to finish before current command begins
      (if `NULL`, wait to finish all previously enqueued events)
    - `event`: event for this command enqueued

![Diagram showing command queues and time]

- `cmd A` (wait for `A`, `B`)
- `cmd B`
- `cmd C` (wait for `A`, `B`)
- `cmd A`, `B` to finish
- Time line:
  - `cmd C`
Platform

- Platform
  - OpenCL implementation

- Vendor specific OpenCL implementation
  - If multiple platforms are installed on a system, need to pick one platform
  - E.g., AMD OpenCL, Intel OpenCL, NVIDIA OpenCL, ...
Context

- Context encapsulates resources into a logical group
  - Memory
    - Buffer (raw bytes)
    - Image (formatted image)
  - Programs
    - Compiled kernel objects (compiled for appropriate devices)
  - Queues
    - Asynchronous execution of commands (in-order or out-of-order)
    - Directly related to a specific device
OpenCL Execution

Compile code

Create data & Set arguments

Send to execution
OpenCL: Example (Vector Square)

- **Get Platform, Devices (CPU, GPU)**
  ```c
  cl_platform_id platform;
  cl_device_id devices[2];
  cl_uint num_platforms, num_devices[2];

  clGetPlatformIDs (1, &platform, &num_platforms);
  clGetDeviceIDs (platform, CL_DEVICE_TYPE_GPU, 1, &devices[0], &num_devices[0]);
  clGetDeviceIDs (platform, CL_DEVICE_TYPE_CPU, 1, &devices[1], &num_devices[1]);
  ```

- **Create Context, Command Queue (GPU)**
  ```c
  cl_context context;
  cl_command_queue commands;
  cl_int err; // OpenCL error code

  context = clCreateContext (NULL, 2 devices, NULL, NULL, &err);
  commands = clCreateCommandQueue (context, devices[0], 0, &err); // GPU = devices[0]
  ```
OpenCL: Example (Kernel)

- Read from source file for kernel code

```c
FILE *sourceFd = fopen("/home/OpenCL/Hello/mykernel.cl", "r");
fseek(sourceFd, 0, SEEK_END);
size_t sourceSize = ftell(sourceFd);
rewind(sourceFd);
char *sourceCode = (char *) malloc(sourceSize+1);
fwrite(sourceCode, sizeof(char), sourceSize, sourceFd);
sourceCode[sourceSize] = '\0';
```

- Kernel code
  - ISO C99
    - No fn_ptr, recursion, ...
  - Vector type
    - char2, int4, float8, ...
  - Image type
    - image2d_t

```
// FILE: mykernel.cl --- kernel code source
kernel void square(global float *in, global float *out, size_t N) {
   size_t i = get_global_id(0);
   if (i < N) out[i] = in[i] * in[i];
}
```
Create and build **Program**

```c
cl_program program;
const char buildOption[ ] = "-cl-mad-enable -Werror";

program = clCreateProgramWithSource(context, 1, (const char**)(&sourceCode), &sourceSize, &err);
err = clBuildProgram (program, 1, &device[0], buildOption, NULL, NULL);
```

Create **Kernel** for command queue

```c
cl_kernel kernel;

kernel = clCreateKernel (program, "square", &err);
```
OpenCL: Example (Buffer, Argument)

- Create **Buffers** for device code

  ```
  #define MAXSIZE 10000
  float A[MAXSIZE], B[MAXSIZE];
  cl_mem input, output;
  ```

  ```
  input = clCreateBuffer (context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                          sizeof(A), A, &err);  // copy the content of A to device buffer input
  ```

  ```
  output = clCreateBuffer (context, CL_MEM_WRITE_ONLY,
                           sizeof(float)*MAXSIZE, NULL, &err);
  ```

- Set **Arguments** for kernel

  ```
  const size_t size = MAXSIZE;
  ```

  ```
  err = clSetKernelArg (kernel, 0, sizeof(input), &input);         // 1st argument
  ```

  ```
  err = clSetKernelArg (kernel, 1, sizeof(output), &output);    // 2nd argument
  ```

  ```
  err = clSetKernelArg (kernel, 2, sizeof(size), &size);             // 3rd argument
  ```
OpenCL: Example (NDRange, Buffer)

- Enqueue Kernel to execute on NDRange (compute domain)
  
  ```c
  size_t gWorkSize[1] = { MAXSIZE };  // 1-dimensional, global work size
  size_t lWorkSize[1] = { 1 };        // 1-dimensional, local work size
  
  clEnqueueNDRangeKernel (commands, kernel, 1, NULL, gWorkSize, lWorkSize, 0, NULL, NULL);
  ```

- Read Buffer from device to host
  
  ```c
  clEnqueueReadBuffer (commands, output, CL_TRUE, 0, sizeof(B), B, 0, NULL, NULL);
  ```
OpenCL: Example (release resources)

- **Release OpenCL resources**
  - `clReleaseMemObject` (input);
  - `clReleaseMemObject` (output);
  - `clReleaseKernel` (kernel);
  - `clReleaseProgram` (program);
  - `clReleaseCommandQueue` (commands);
  - `clReleaseContext` (context);
Summary

- Heterogeneous multiprocessing
  - CPUs, GPUs, DSPs, other HW accelerators

- OpenCL applications
  - Detecting computing resources and make **Platform/Devices**
  - Create **Context** for Programs, Kernels, Memory Objects
  - Create **Command Queues** for each Device
  - Create/build **Programs** from source code (text file or text array)
  - Create **Kernel** from Program
  - Create **Buffers** as memory objects
  - Set **Arguments** for Kernel
  - Enqueue Kernel on **NDRange** or Enqueue Kernel as a **Task**
  - Enqueue **BufferRead** to fetch the result
  - Release OpenCL resources