GPU Teaching Kit

Accelerated Computing

Module 20 – Related Programming Models: OpenCL

Lecture 20.3 - OpenCL Host Code

Objective

- To learn to write OpenCL host code
  - Create OpenCL context
  - Create work queues for task parallelism
  - Device memory Allocation
  - Kernel compilation
  - Kernel launch
  - Host-device data copy

OpenCL Context

- Contains one or more devices
- OpenCL memory objects are associated with a context, not a specific device
- clCreateBuffer() is the main data object allocation function
  - error if an allocation is too large for any device in the context
- Each device needs its own work queue(s)
- Memory copy transfers are associated with a command queue (thus a specific device)

OpenCL Context Setup Code (simple)

```
cl_int clerr = CL_SUCCESS;
cl_context clctx = clCreateContextFromType(0, CL_DEVICE_TYPE_ALL, NULL,
NULL, &clerr);

size_t parmsz;
clerr = clGetContextInfo(clctx, CL_CONTEXT_DEVICES, 0, NULL, &parmsz);

cl_device_id* cldevs = (cl_device_id *) malloc(parmsz);
clerr = clGetContextInfo(clctx, CL_CONTEXT_DEVICES, parmsz, cldevs,
NULL);

cl_command_queue clcmdq = clCreateCommandQueue(clctx, cldevs[0], 0,
&clerr);
```

OpenCL Kernel Compilation: vadd

```
const char* vaddsrc =
   “__kernel void vadd(__global float *d_A, __global float *d_B,
__global float *d_C, int N) { \n“   […etc and so forth…]

cl_program clpgm;
clpgm = clCreateProgramWithSource(clc
&clerr);

char clcompileflags[4096];
sprintf(clcompileflags, “-cl-mad-enab
clerr = clBuildProgram(clpgm, 0, NULL
NULL);
cl_kernel clkern = clCreateKernel(clpgm, “vadd", &clerr);
```

OpenCL kernel source code as a big string

Gives raw source code string(s) to OpenCL

Set compiler flags, compile source, and retrieve a handle to the “vadd” kernel

OpenCL Device Memory Allocation

- clCreateBuffer();
  - Allocates object in the device Global Memory
  - Returns a pointer to the object
  - Requires five parameters
    - OpenCL context pointer
    - Flags for access type by device (read/write, etc.)
    - Size of allocated object
    - Host memory pointer, if used in copy-from-host mode
    - Error code
- clReleaseMemObject()
  - Frees object
    - Pointer to freed object

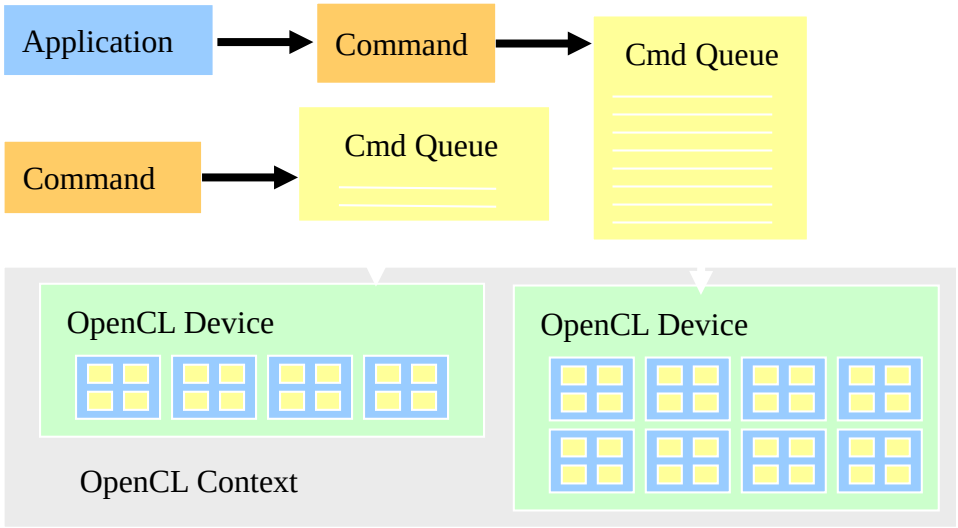
OpenCL Device Memory Allocation (cont.)

- Code example:
  - Allocate a 1024 single precision float array
  - Attach the allocated storage to d_a
  - “d_” is often used to indicate a device data structure

```
VECTOR_SIZE = 1024;
cl_mem d_a;
int size = VECTOR_SIZE* sizeof(float);

d_a = clCreateBuffer(clctx,
    CL_MEM_READ_ONLY, size, NULL, NULL);
…
clReleaseMemObject(d_a);
```

OpenCL Device Command Execution



OpenCL Host-to-Device Data Transfer

- `clEnqueueWriteBuffer();`
  - Memory data transfer to device
  - Requires nine parameters
    - OpenCL command queue pointer
    - Destination OpenCL memory buffer
    - Blocking flag
    - Offset in bytes
    - Size (in bytes) of written data
    - Source host memory pointer
    - List of events to be completed before execution of this command
    - Event object tied to this command

OpenCL Device-to-Host Data Transfer

- `clEnqueueReadBuffer();`
  - Memory data transfer to host
  - requires nine parameters
    - OpenCL command queue pointer
    - Source OpenCL memory buffer
    - Blocking flag
    - Offset in bytes
    - Size of bytes of read data
    - Destination host memory pointer
    - List of events to be completed before execution of this command
    - Event object tied to this command

OpenCL Host-Device Data Transfer (cont.)

- Code example:
  - Transfer a 64 * 64 single precision float array
  - a is in host memory and d_a is in device memory

```
clEnqueueWriteBuffer(clcmdq, d_a, CL_FALSE, 0,
                     mem_size, (const void * )a, 0,
      0, NULL);

clEnqueueReadBuffer(clcmdq, d_result,
      CL_FALSE, 0,
                     mem_size, (void * )
      host_result, 0, 0, NULL);
```

OpenCL Host-Device Data Transfer (cont.)

- `clCreateBuffer` and `clEnqueueWriteBuffer` can be combined into a single command using special flags.
- Eg:

```
d_A=clCreateBuffer(clctxt,
CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
mem_size, h_A, NULL);
```

  - Combination of 2 flags here. `CL_MEM_COPY_HOST_PTR` to be used only if a valid host pointer is specified.
  - This creates a memory buffer on the device, and copies data from h_A into d_A.
  - Includes an implicit `clEnqueueWriteBuffer` operation, for all devices/command queues tied to the context clctxt.

Device Memory Allocation and Data Transfer for vadd

```
float *h_A = …,   *h_B = …;
    // allocate device (GPU) memory
   cl_mem d_A, d_B, d_C;
   d_A = clCreateBuffer(clctx, CL_MEM_READ_ONLY |
          CL_MEM_COPY_HOST_PTR, N *sizeof(float), h_A, NULL);
   d_B = clCreateBuffer(clctx, CL_MEM_READ_ONLY |
          CL_MEM_COPY_HOST_PTR, N *sizeof(float), h_B, NULL);
   d_C = clCreateBuffer(clctx, CL_MEM_WRITE_ONLY,
   N *sizeof(float), NULL, NULL);
```

Device Kernel Configuration Setting for vadd

```
clkern=clCreateKernel(clpgm, “vadd", NULL);
…
clerr= clSetKernelArg(clkern, 0, sizeof(cl_mem),(void *)&d_A);
clerr= clSetKernelArg(clkern, 1, sizeof(cl_mem),(void *)&d_B);
clerr= clSetKernelArg(clkern, 2, sizeof(cl_mem),(void *)&d_C);
clerr= clSetKernelArg(clkern, 3, sizeof(int), &N);
```

Device Kernel Launch and Remaining Code for vadd

```
cl_event event=NULL;
clerr= clEnqueueNDRangeKernel(clcmdq, clkern, 2, NULL,
     Gsz, Bsz, 0, NULL, &event);
clerr= clWaitForEvents(1, &event);
clEnqueueReadBuffer(clcmdq, d_C, CL_TRUE, 0,
     N*sizeof(float), h_C, 0, NULL, NULL);
clReleaseMemObject(d_A);
clReleaseMemObject(d_B);
clReleaseMemObject(d_C);
}
```

GPU Teaching Kit

Accelerated Computing