2017 年春季 - 并行与分布式计算导论

ASSIGNMENT 3

Assigned: 05/26/2017, Due: 06/11/2017

Instructor: Guojie Luo (gluo@pku.edu.cn)

Problem 1 – OpenCL practice

OpenCL represents OpenCL C++, and it consists of two parts: host and device.
The host is an abstraction that defines where an application is executed, normally the operating system and CPU. The host can access devices that support OpenCL, normally accelerators such as GPU. OpenCL provides kernel languages that are used to write programs that can run on OpenCL devices.
Let's look at our simple example:
Host code (hello.cpp):

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <CL/cl.h>
4.
5.  #define MEM_SIZE (128)
6.  #define MAX_SOURCE_SIZE (0x100000)
7.
8.  int main()
9.  {
10.     cl_device_id device_id = NULL;
11.     cl_context context = NULL;
12.     cl_command_queue command_queue = NULL;
13.     cl_mem memobj = NULL;
14.     cl_program program = NULL;
15.     cl_kernel kernel = NULL;
16.     cl_platform_id platform_id = NULL;
17.     cl_uint ret_num_devices;
18.     cl_uint ret_num_platforms;
19.     cl_int ret;
20.
21.     char string[MEM_SIZE];
22.
23.     FILE *fp;
24.     char fileName[] = "./hello.cl";
25.     char *source_str;
```

```
26.     size_t source_size;
27. 
28.     /* Load the source code containing the kernel*/
29.     fp = fopen(fileName, "r");
30.     if (!fp) {
31.         fprintf(stderr, "Failed to load kernel.\n");
32.         exit(1);
33.     }
34.     source_str = (char*)malloc(MAX_SOURCE_SIZE);
35.     source_size = fread(source_str, 1, MAX_SOURCE_SIZE, fp);
36.     fclose(fp);
37. 
38.     /* Get Platform and Device Info */
39.     ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);
40.     ret = clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_DEFAULT, 1, &device_id,
    &ret_num_devices);
41. 
42.     /* Create OpenCL context */
43.     context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &ret);
44. 
45.     /* Create Command Queue */
46.     command_queue = clCreateCommandQueue(context, device_id, 0, &ret);
47. 
48.     /* Create Memory Buffer */
49.     memobj = clCreateBuffer(context, CL_MEM_READ_WRITE,MEM_SIZE * sizeof(char),
    NULL, &ret);
50. 
51.     /* Create Kernel Program from the source */
52.     program = clCreateProgramWithSource(context, 1, (const char **)&source_str,
53. (const size_t *)&source_size, &ret);
54. 
55.     /* Build Kernel Program */
56.     ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
57. 
58.     /* Create OpenCL Kernel */
59.     kernel = clCreateKernel(program, "hello", &ret);
60. 
61.     /* Set OpenCL Kernel Parameters */
62.     ret = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *)&memobj);
63. 
64.     /* Execute OpenCL Kernel */
65.     ret = clEnqueueTask(command_queue, kernel, 0, NULL,NULL);
66. 
67.     /* Copy results from the memory buffer */
```

```
    ret = clEnqueueReadBuffer(command_queue, memobj, CL_TRUE, 0,
    MEM_SIZE * sizeof(char),string, 0, NULL, NULL);

    /* Display Result */
    puts(string);

    /* Finalization */
    ret = clFlush(command_queue);
    ret = clFinish(command_queue);
    ret = clReleaseKernel(kernel);
    ret = clReleaseProgram(program);
    ret = clReleaseMemObject(memobj);
    ret = clReleaseCommandQueue(command_queue);
    ret = clReleaseContext(context);

    free(source_str);

    return 0;
}
```

Device Code (hello.cl):

```
__kernel void hello(__global char* string)
{
    string[0] = 'H';
    string[1] = 'e';
    string[2] = 'l';
    string[3] = 'l';
    string[4] = 'o';
    string[5] = ',';
    string[6] = ' ';
    string[7] = 'W';
    string[8] = 'o';
    string[9] = 'r';
    string[10] = 'l';
    string[11] = 'd';
    string[12] = '!';
    string[13] = '\0';
}
```

Compile & Run (on our Server):

```
g++ -I/usr/local/cuda/include/ -L/usr/local/cuda/lib64/ -lOpenCL hello.cpp -o hello
./hello
```

Result:

```
[user@ceca24 ~]$ ./hello
```

```
Hello, World!
```

You can organize these commands as a Makefile or a shell file for convenience.

Task: Matrix-Vector Multiplication, take one matrix and one vector as input, and output the result.

Example Input:

Matrix:

```
1  2  3
2  3  4
```

Vector:

```
3
2
1
```

Example Output:

```
10  16
```

1. Use OpenCL to implement an effective solution with high performance. Analyze the performance and scalability.
2. Learn more about global_id, local_id, and work_item, and balance the workload. For example, what performance will be if maximum workgroup reached?