Politecnico di Milano

ACA Project : Implementation of a subset of OpenCL wrapper for Mango API

June 10, 2017

Authors:

- Domenico FAVARO (Mat. 837995)
- Matheus FIM (Mat. 876069)

Prof. Giovanni Agosta

Contents

1 Mango API - OpenCL

The Mango API is a tool to manage resources and parallelism that aims to attain higher resource efficiency to be used by HPC (High Performance Computing) applications.
As OpenCL is a framework to manage resources as devices, CPUs and GPUs, to control the platform and execute programs on these devices. Providing an interface for parallel computing using task- and data-based parallelism.

2 Objective

On the implementation of the wrapper for OpenCL compliant code we need to use the Mango API for parallel programming. Calls to functions should be in OpenCL format and then translated wrapped around to its Mango equivalent.
The OpenCL functions will be presented with their equivalent Mango function or translated in a way to do what OpenCL wants using the available Mango functions. If none can be achieved then the OpenCL function will be marked as not wrappable.

3 Mango Elements

3.1 Context

Created automatically when `mango_init()`is called

3.2 KernelFunction

This is an array of function pointers to support multiple versions of the kernel. Data structure that'll store what program will be executed in which kernel.

3.3 Kernels

A kernel represents a function that'll be executed, the concept is similar to OpenCL kernel. A Mango kernel however will be included in a task graph.

3.4 Buffers

Used to transfer datao to and from GN (Guest) and HN (Host). Mango implements a further FIFO Buffer for a a burst-mode data transfer.

3.5 TaskGraph

Data structure representing an execution (task) that can include one or several kernels defining which buffers and events it will use.

3.6 Events

Used to synchronize executions between kernels. A typical event symbolizes the completion of a Kernel.

3.7 Arguments

Typically pointers to buffers, with the appropriate size or alternately, they can be scalar values or events to be passed to the kernel.

3.8 Mango_Types and Error_Types

Define the types that will be used inside Mango (ex. File Types) and more interestingly the Errors that can happen during execution, that are significantly less than OpenCL.

4 Mango Program Flow

An usual program executed in Mango API follows a setup that prepares the elements needed to be executed in parallel by creating one or more kernels to which resources (devices) are allocated. The flow is the following:

- Init Mango: `mango_init()`
- Kernel Declarations: `mango_kernelfunction_init(); mango_load_kernel(); mango_kernel_t = mango_register_kernel();`
- Registering buffers: `mango_buffer_t = mango_register_memory()`
- Create a Task Graph (returns an event): `mango_task_graph_t = mango_task_graph_create()`
- Resource Allocation: `mango_resource_allocation()`
- Declare Arguments that will be passed to the kernel: `mango_arg_t = mango_arg()`
- Transfer buffers from host to device: `mango_write()`
- Start the kernels (returns an event) and execute synchronization tasks between events: `mango_event_t = mango_start_kernel()`

- Read the result from device to host: `mango_read()`
- Deallocate resources, destroy task graph and release Mango: `mango_resource_deallocation()` `mango_task_graph_destroy_all(); mango_release();`
- Continue offline code...

5 OpenCL Elements - Wrapper

5.1 Platform

Not applicable as the platform used will always be Mango.

5.2 Devices

Mango = Emulate?

5.3 Context

Mango Context will be used.

5.4 Buffer Objects - Memory Objects

Equivalent to Buffers, used to read write data.

5.5 Programs

Mango = KernelFunction

5.6 Kernels

Mango = Kernel

5.7 Events

Mango = Events

5.8 Command Queue

Mango = TaskGraph?

5.9 Exceptions

Mango = ErrorTypes

5.10 Images

As the wrapper will limit itself outside of graphic objects, no Images (2D or 3D) will be used.

6 OpenCL Flow - Wrapper

OpenCL flow offers three types of task parallelism:

- Internal to the task: won't be addressed directly in the wrapper.
- Kernels executing tasks concurrently in an out-of-order queue: This can be wrapped on Mango.
- Use of events synchronization: This is done by task graphs in Mango and it provides a set of tools to sync the queues. OpenCL does not address this specifically as it has no concept of Task Graph but the event synchronization concept is the same and thus can be wrapped.

An example of OpenCL program flow is the following:

1. Get available Platform
2. Get available Devices
3. Create Context
4. Create Command Queue
5. Create Buffers
6. Create and Build Program
7. Create Kernel
8. Set Kernel Arguments
9. Queue Buffers
10. Queue and execute Kernels
11. Read the result from read buffer
12. Release all resources, program, kernel, buffers and context
13. Continue offline code...

We'll present the wrapping for each of these steps.

6.1 Get available Platform

No

6.2 Get available Devices

???

6.3 Create Context

`mango_init()`

6.4 Create Command Queue

`mango_task_graph_t = mango_task_graph_create()`

6.5 Create Buffers

`mango_buffer_t = mango_register_memory()`

6.6 Create and Build Program

`mango_kernelfunction_init(); mango_load_kernel();`

6.7 Create Kernel

`mango_kernel_t = mango_register_kernel();`

6.8 Set Kernel Arguments

`mango_arg_t = mango_arg()`

6.9 Queue Buffers

`mango_write()`

6.10 Queue and execute Kernels

`mango_event_t = mango_start_kernel()`

6.11 Read the result from read buffer

`mango_read()`

6.12 Release all resources, program, kernel, buffers and context

`mango_resource_deallocation(); mango_task_graph_destroy_all(); mango_release();`

7 Documentation

- **Mango:**`http://www.mango-project.eu/`
- **OpenCL:** OpenCL Programming Guide - by Benedict Gaster and Timothy G. Mattson
- **Khronos OpenCL:** `https://www.khronos.org/opencl/`

8 Changelog

- **Version 1.1:** 30/06/2017