TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

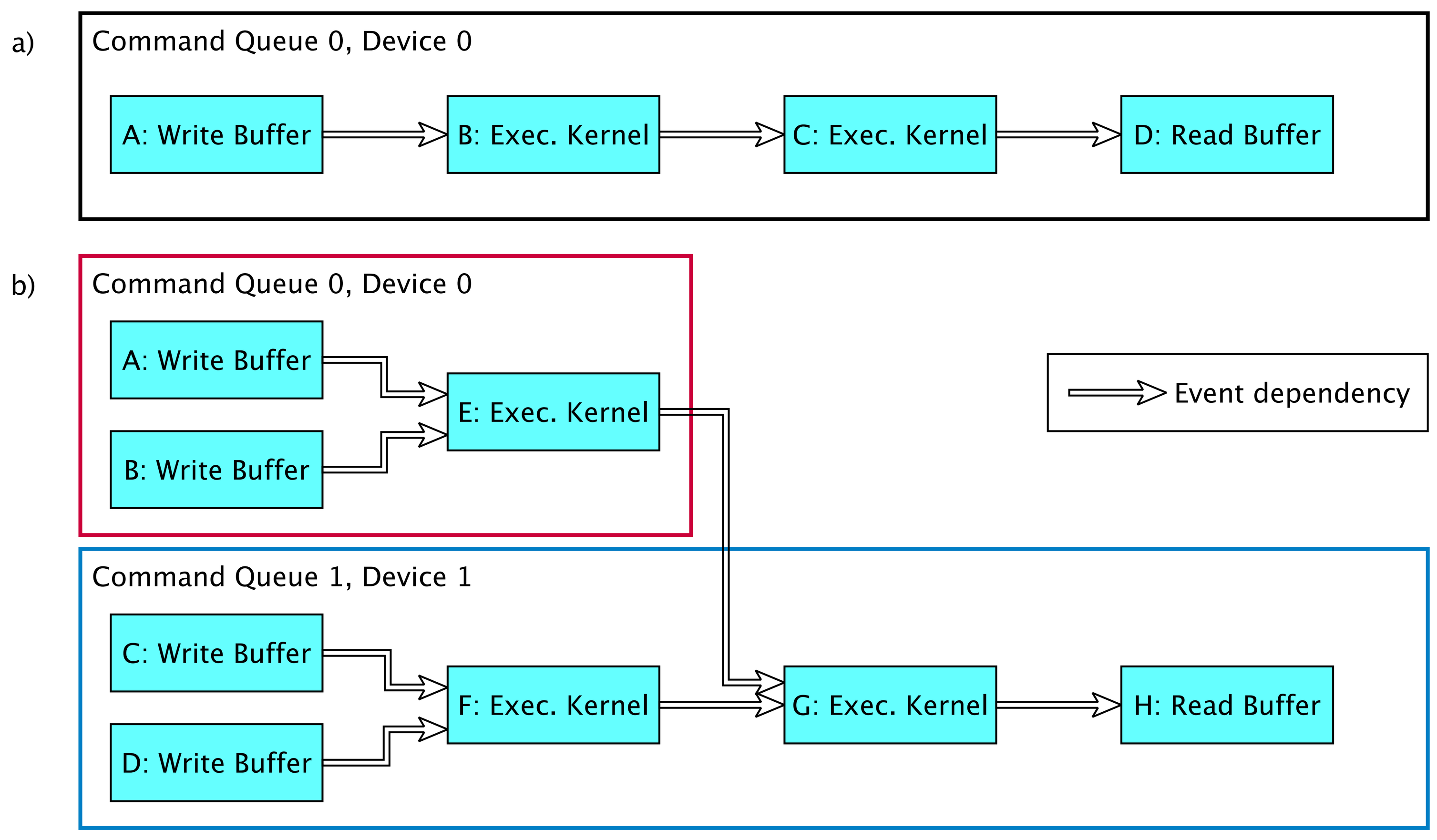
KALRAY

Out of Order Execution Framework for OpenCL Implementation

Ville Korhonen (ville.t.korhonen@tut.fi), Pekka Jääskeläinen (pekka.jaaskelainen@tut.fi) / Tampere University of Technology, Finland
Clément Léger (clement.leger@kalray.eu) / Kalray, France

Task Parallelism in OpenCL

- Task level parallelism can be expressed with out of order command queues.
- Scheduling freedom of commands when respecting the explicit synchronization and event dependencies.
- Command queues are synchronised across all devices by the runtime, which simplifies host application control logic.



a) No task level parallelism
b) At most 4 tasks (A, B, C, D) can be executed in parallel, if there are computational resources

Goals

- Flexible framework for implementing out of order command queues (OoOCQ).
- Enable distribution of scheduling and synchronisation overhead to devices
- Support different degrees of host orchestration vs. independent task graph execution in the device.

Framework Functions

An extension to PoCL host-device/driver interface.

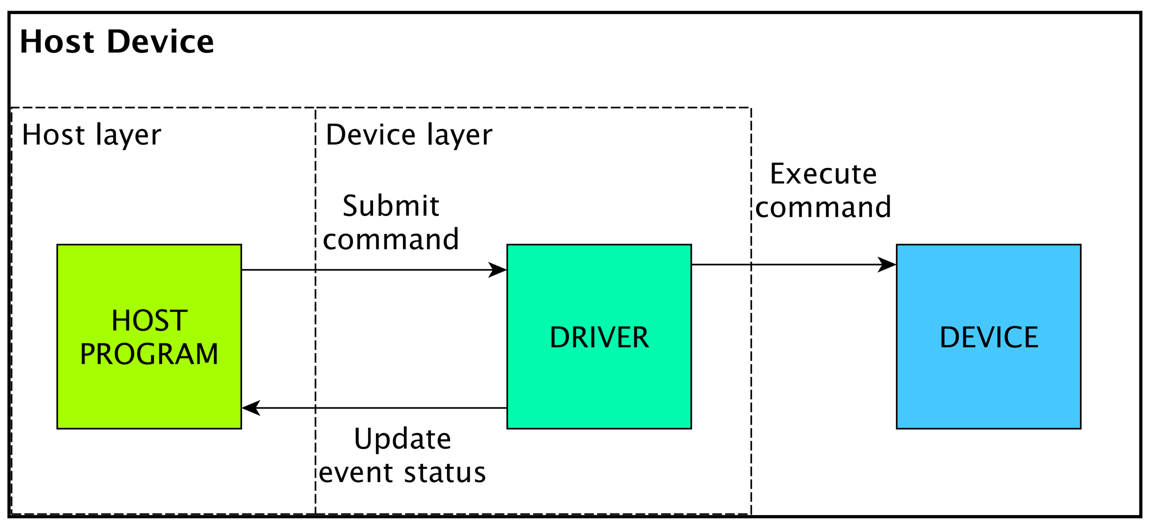
OoOE support can be implemented for a new device by redefining some or all of the framework functions:

submit()	Submits a command to the device driver.
flush()	Flushes commands to the device.
join()	Used by clFinish to ensure that commands will be executed.
broadcast()	When command is completed a notification is broadcasted for all devices that have commands waiting completed command.
notify()	Used for notifying device driver that a waited event has been completed.

Example Implementations

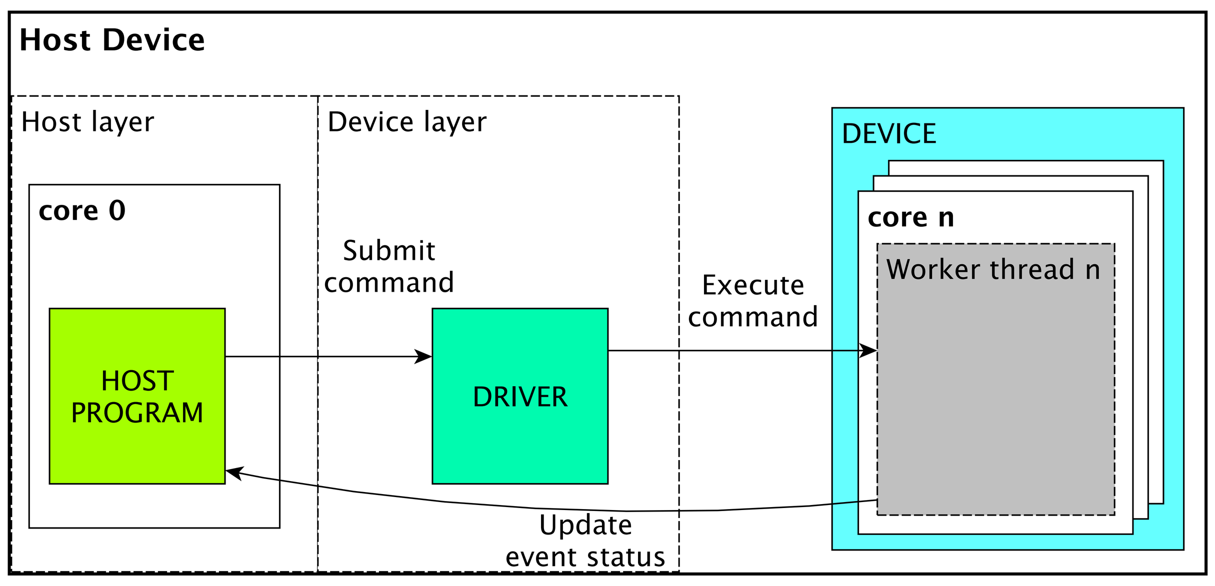
Standalone Single Core

- For small embedded devices and soft cores
- No threading support assumed
- Host + kernels possibly compiled to the same image



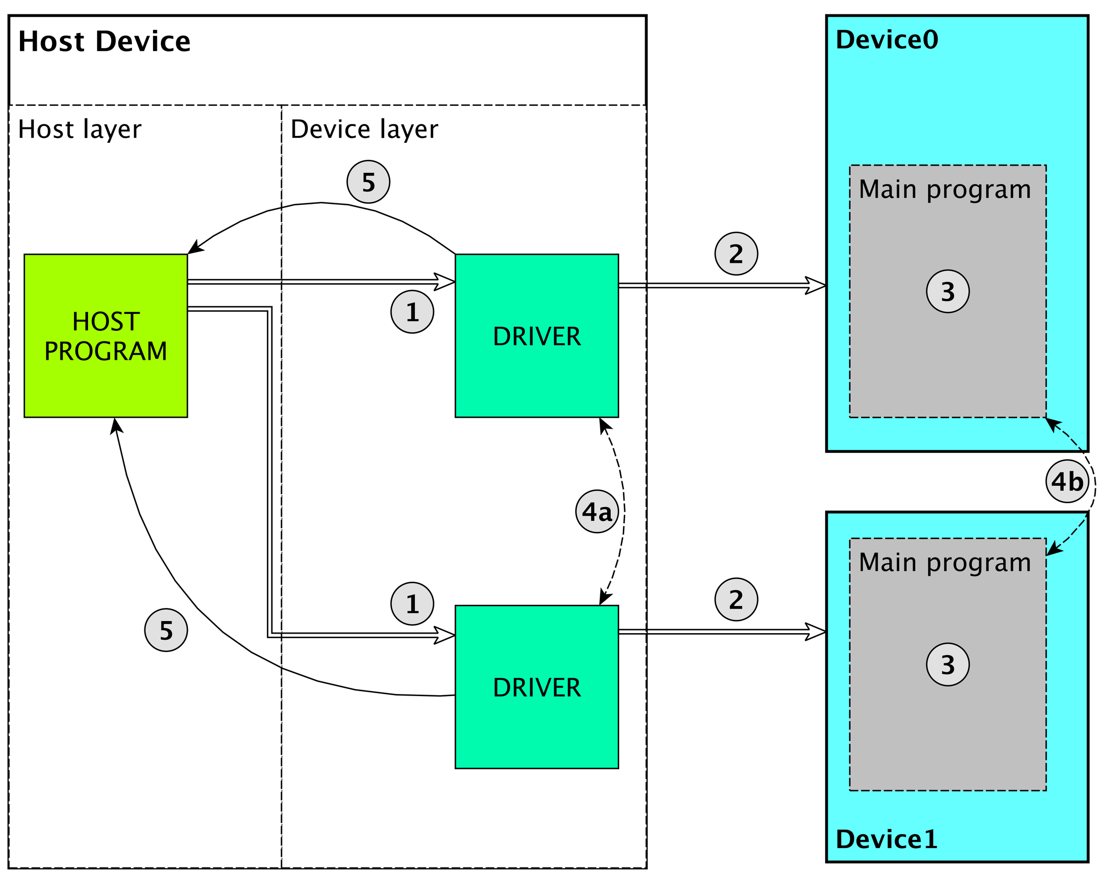
Homogeneous CPU with Multiple Cores and/or HW Threads

- By default one worker thread per core.
- Independent task graph execution.
- Memory shared with the host, low overheads.
- Load balancing across cores.



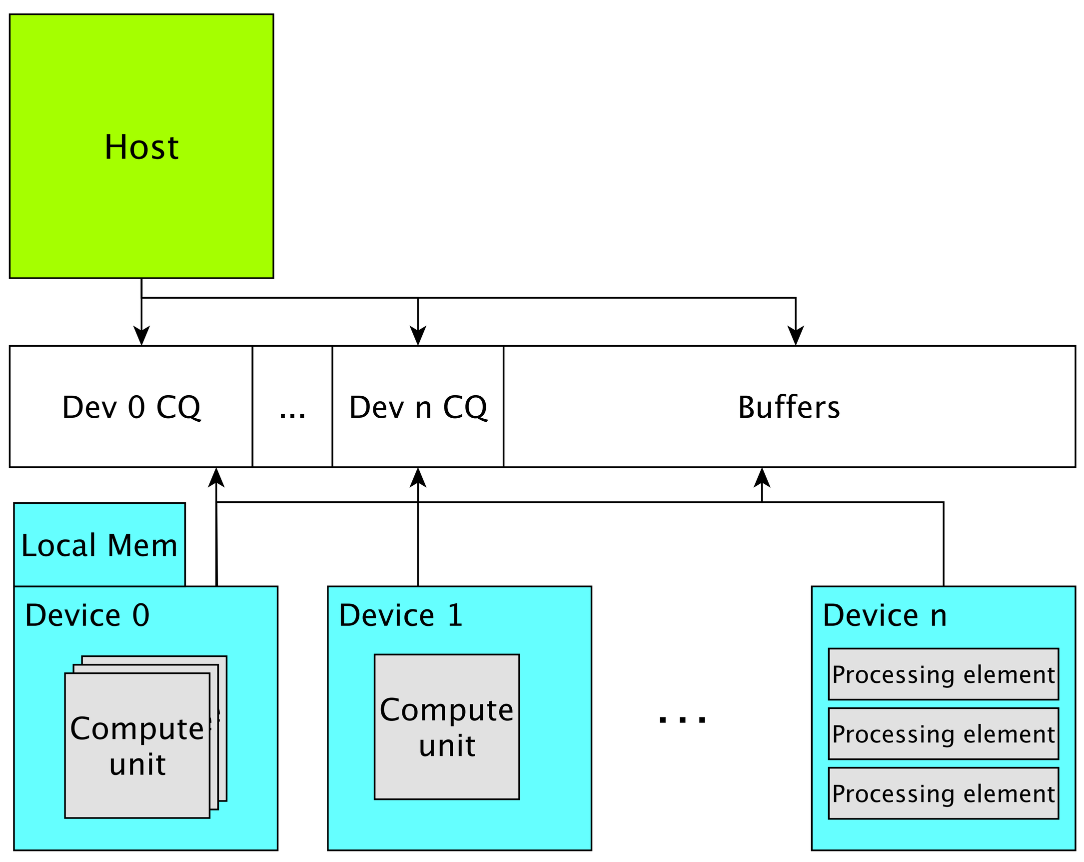
Devices with Task Graph Execution Capabilities

1. CQ is pushed to the device driver.
2. Driver pushes CQ to device's command buffer.
3. Events are notified to host by rasing interrupt.
4a. Events notified to the host that broadcasts the events to the listeners OR
4b. Device notifies peers independently.
5. Driver updates host side event status in any case.



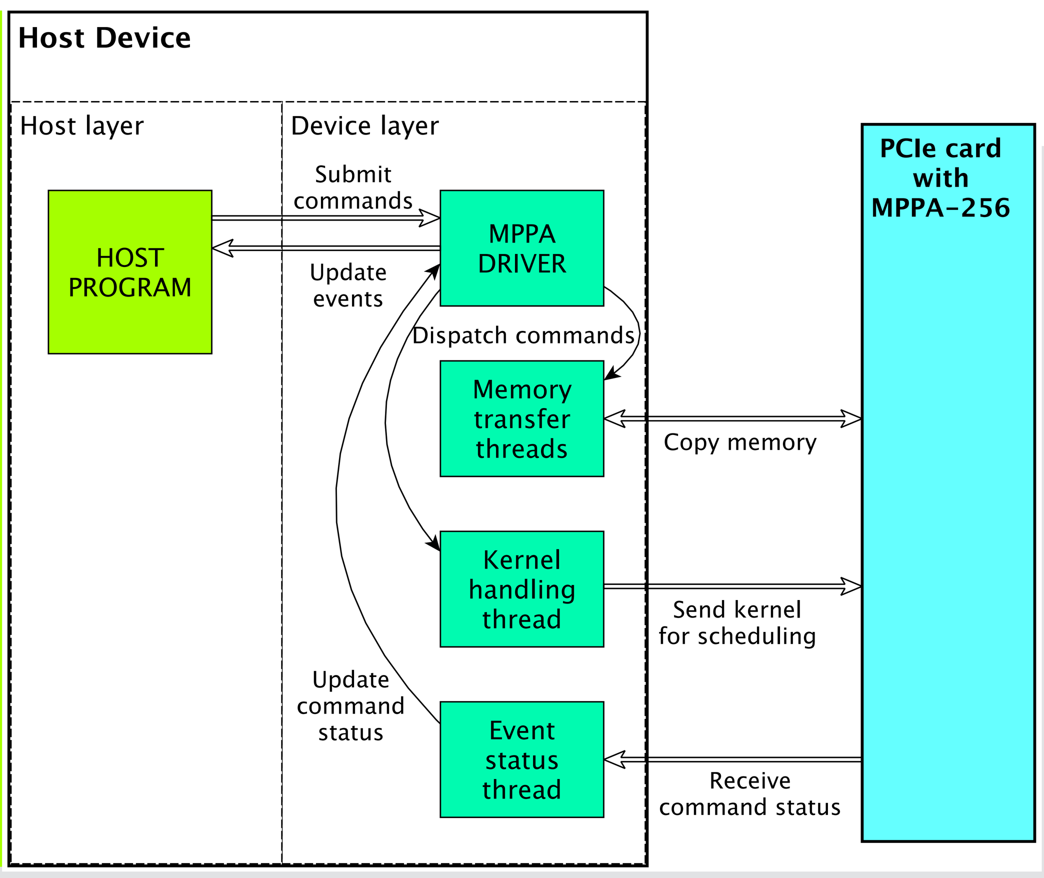
Heterogeneous Platform with Shared System Address Space

- Each device can be with different ISA.
- Each device can have a local memory.
- Global memory buffers reside in the shared host accessible global memory.
- Commands are submitted to devices command queues in global memory.
- Events are notified to peers by modifying commands waitlist in other devices CQ.



Kalray MPPA-256 Manycore Processor

- PCIe accelerator card with MPPA-256 processor.
- Multiple kernels enqueued and ran simultaneously.
- Threads to handle transfers to/from global memory asynchronously from command queue execution
- Transfers can be done while kernels are executed to hide latency
- Kernel scheduling offloading on device for special cases (kernel-to-kernel dependencies)



pocl

This work will be contributed to
Portable Computing Language - an open source OpenCL implementation available at http://portablecl.org