13th International Workshop on OpenCL and SYCL

IWOCL 2025

Latency Reduction Potential of Server-Side Command Buffers in OpenCL-Based Edge Offloading

Jan Solanti, Tampere University

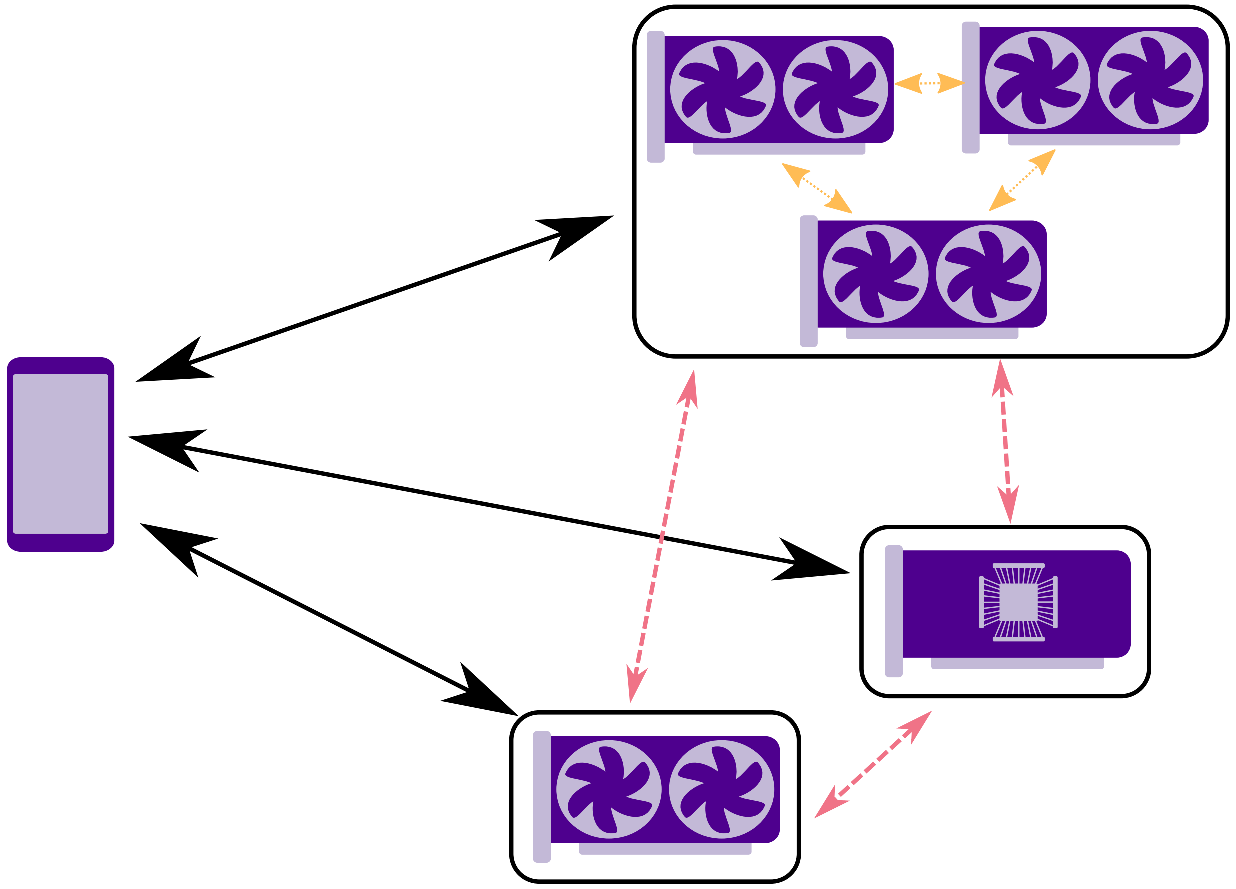
Jan Solanti (Tampere University), Pekka Jääskeläinen (Tampere University, Intel Finland Oy)

Remote OpenCL Offloading

- Sometimes you want to do heavy GPGPU work on mobile devices
  - Machine Learning, Image Recognition/Classification, ...
- Might not have a GPU or other accelerators
  - And even if, they would be impractically slow and drain the battery too fast

➠ Offload it to something with more power

PoCL-Remote

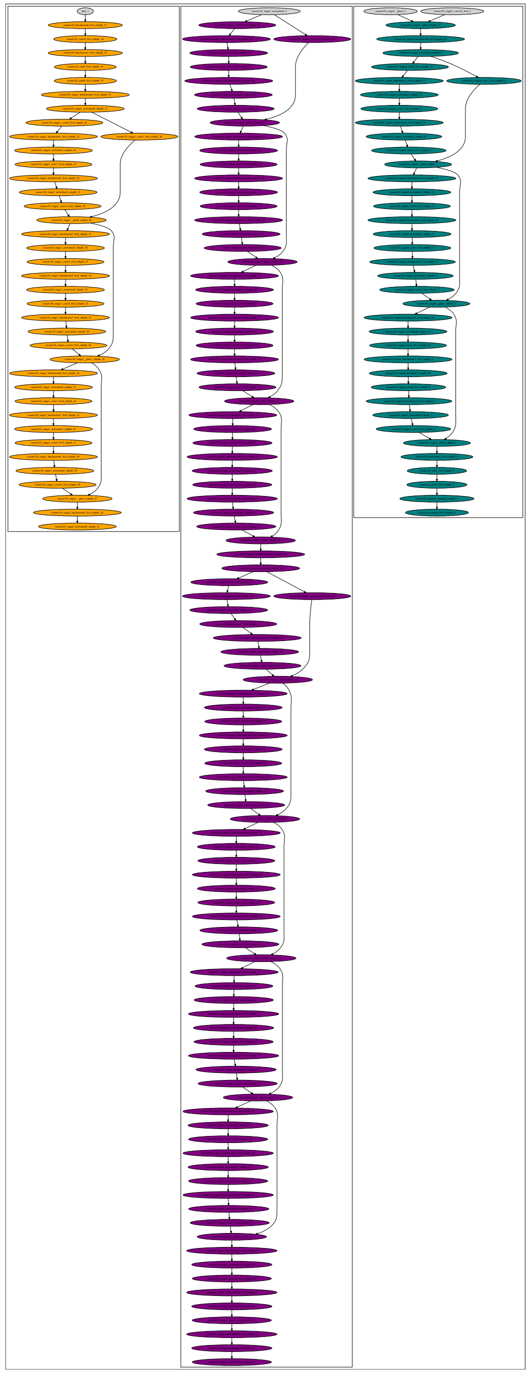


Why OpenCL for Remote Offloading?

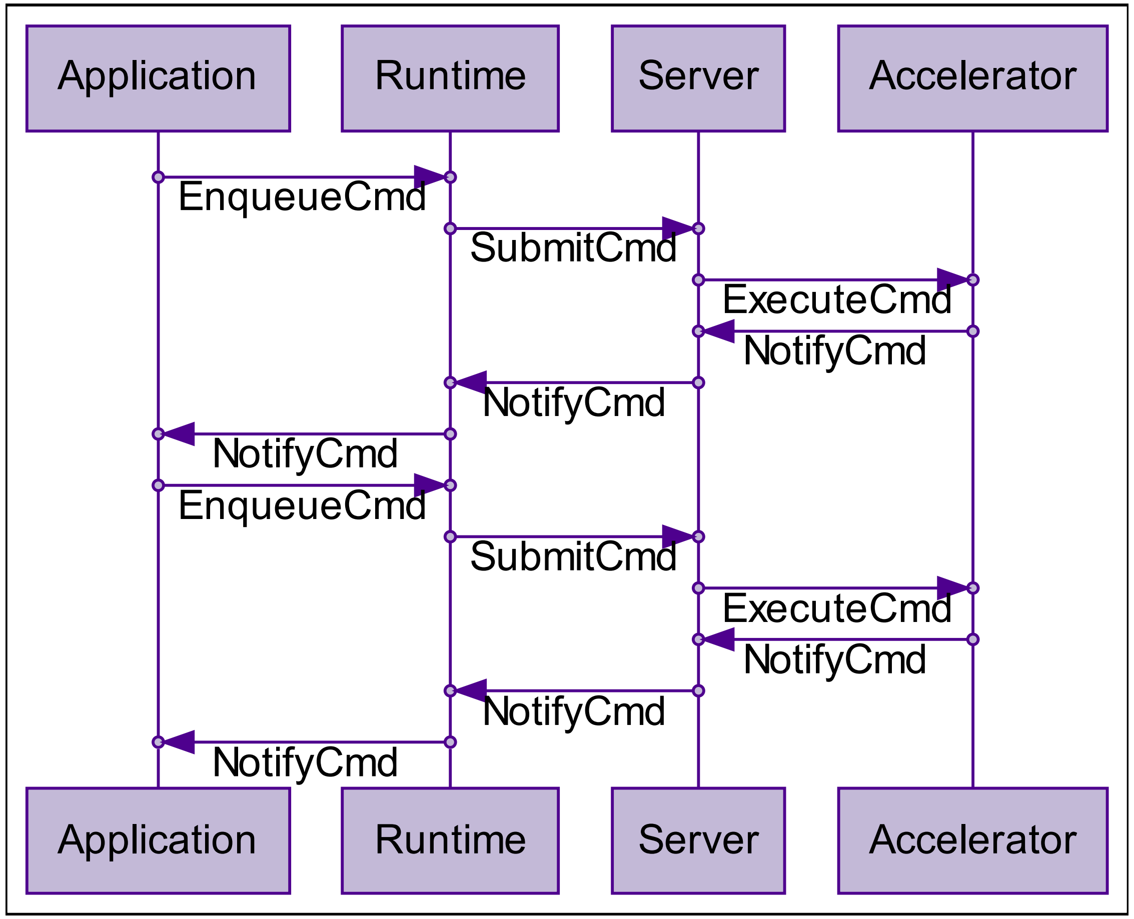
- Bespoke high level APIs built on top of e.g. HTTP are very common
- However, they are usually application-dependent
  - Can't easily update to different application logic without versioned endpoints
  - Difficult to do work partitioning on the fly
    - Dynamic performance vs power optimization
    - Privacy (e.g. split inference)

Example: ResNet50-v2-7

- Split into 3 parts for U-shaped split inference
  - Server sees neither input data nor final results
- Splitting done programmatically, exact positions chosen at runtime



Remote OpenCL Offloading



Tampere University

Remote OpenCL Offloading

- 1 problem solved
- 1 problem created

- Transferring data over an IP network takes a lot longer than over a PCIe or SoC interconnect

    ➠ Compression, yes, but...

- Even tiny data transfers are subject to network latency
    - Latencies of multiple transfers add up quickly

OpenCL Command Buffers

- cl_khr_command_buffer
  - Largely analogous to CUDA and SYCL "(task) graphs"
- Provisional extension since November 2021
- Several additions on top of the base extension
- Constructed at application runtime
- Cheap to rebuild on the fly for different partitioning

Command Buffer Implementation

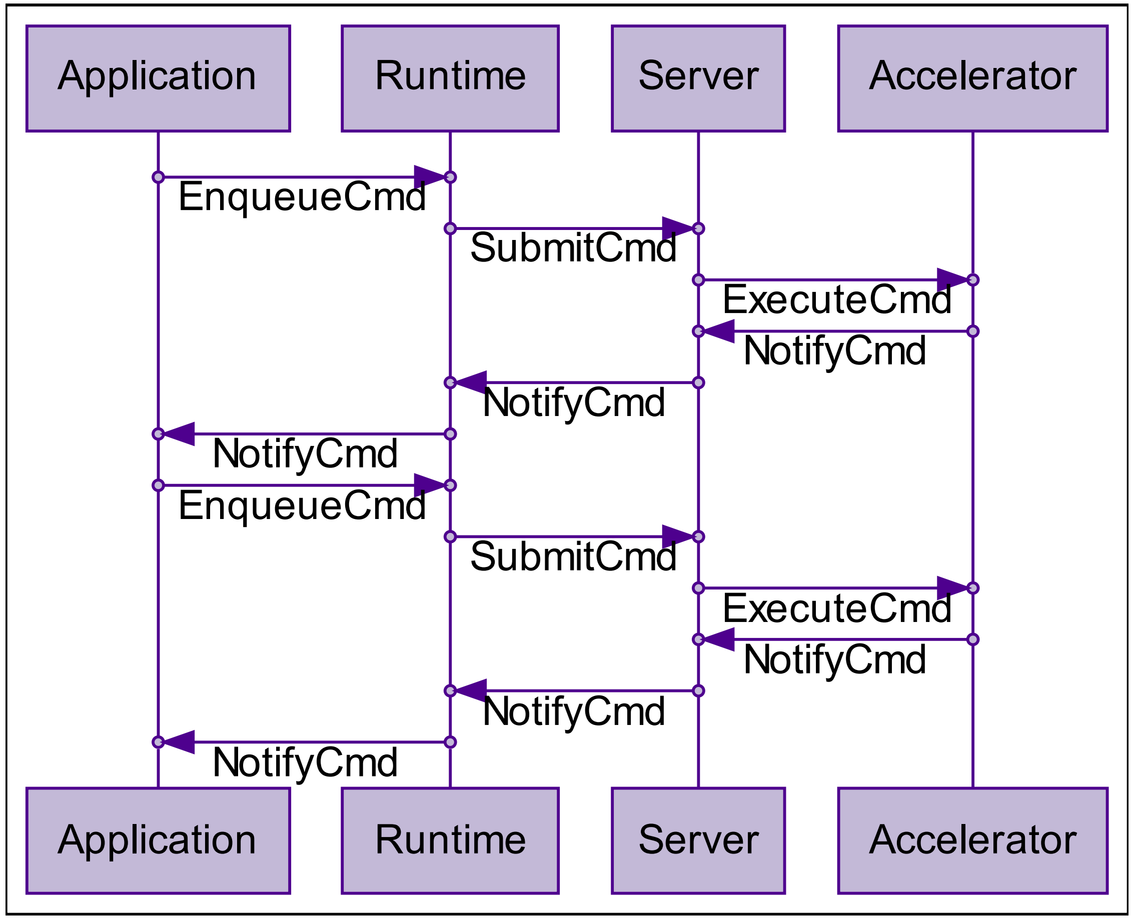
- Baseline is very straightforward
  - Store a list of `clCommand*` calls and issue corresponding `clEnqueue*` calls whenever the buffer is executed
  - (a bit of manual work needed wrt kernel arguments)
- Can be done with an ICD layer on top of any conformant driver[1]

[1] Emulating Command Buffer Extensions with OpenCL Layers, Ewan Crawford, James Brodman, Ben Ashbaugh, IWOCL 2024

Command Buffers in PoCL

- Parameter validation is virtually identical between `clCommand*` and `clEnqueue*` functions
  - ➠ Share validation code between corresponding function pairs
- In PoCL almost all command life cycle management code got reused as-is for commands recorded to command buffers
  - ➠ Implementation for generating live commands from recorded ones when enqueueing a command buffer is **very** simple

Remote Offloading With Naïve Command Buffers



Tampere University

Remote Offloading With Naïve Command Buffers

- Easy to implement
- Convenient, no need for specialized code paths in applications nor OpenCL runtimes
- Still has to synchronize every recorded command with the client -> latency overhead

Server-Side Command Buffers

- Unlike command queues, command buffers have an unambiguous end: `clFinalizeCommandBufferKHR`
- Natural point for performing command graph optimizations
- Also a natural point for batching commands in order to send them to a remote in bulk
  - Also well suited for e.g. compression on the wire

Reduces communication to just one (1) roundtrip per command buffer invocation (plus one to create the buffer)

Device-Side Command Buffers in PoCL

- Additional command buffer implementation(s) in PoCL
- Command recording shared as-is with the “overlay” command buffer implementation
- `clFinalizeCommandBufferKHR` detects if target device has a specialized implementation and calls that to construct a device side command buffer
- `clEnqueueCommandBufferKHR` checks if target device has a specialized implementation and calls that instead of submitting individual commands

Server-Side Command Buffers in PoCL-Remote

- Finalize handler packs all commands in a memory buffer
  - Tightly packed, exactly how they would be written to the TCP socket
- Buffer is sent as payload of a "create command buffer" command

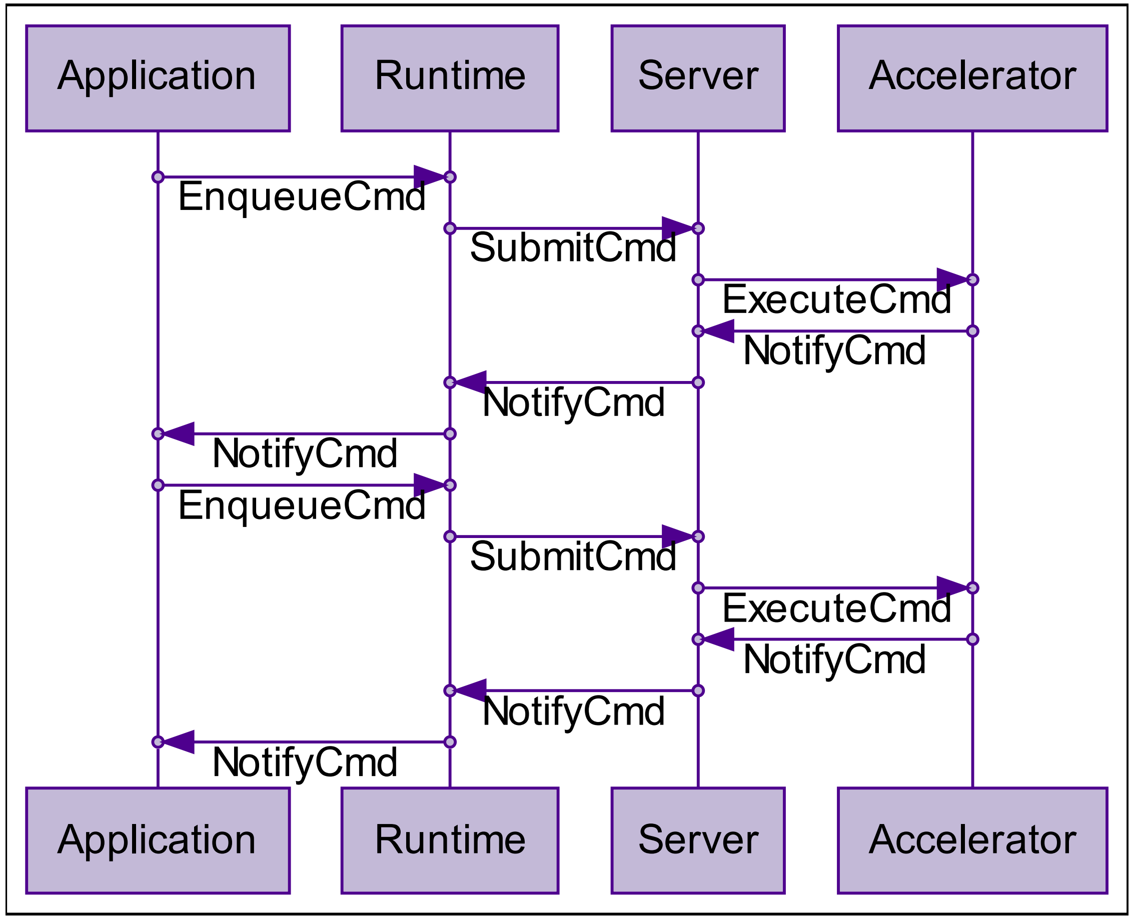
Constructing Command Buffers in `pocld`

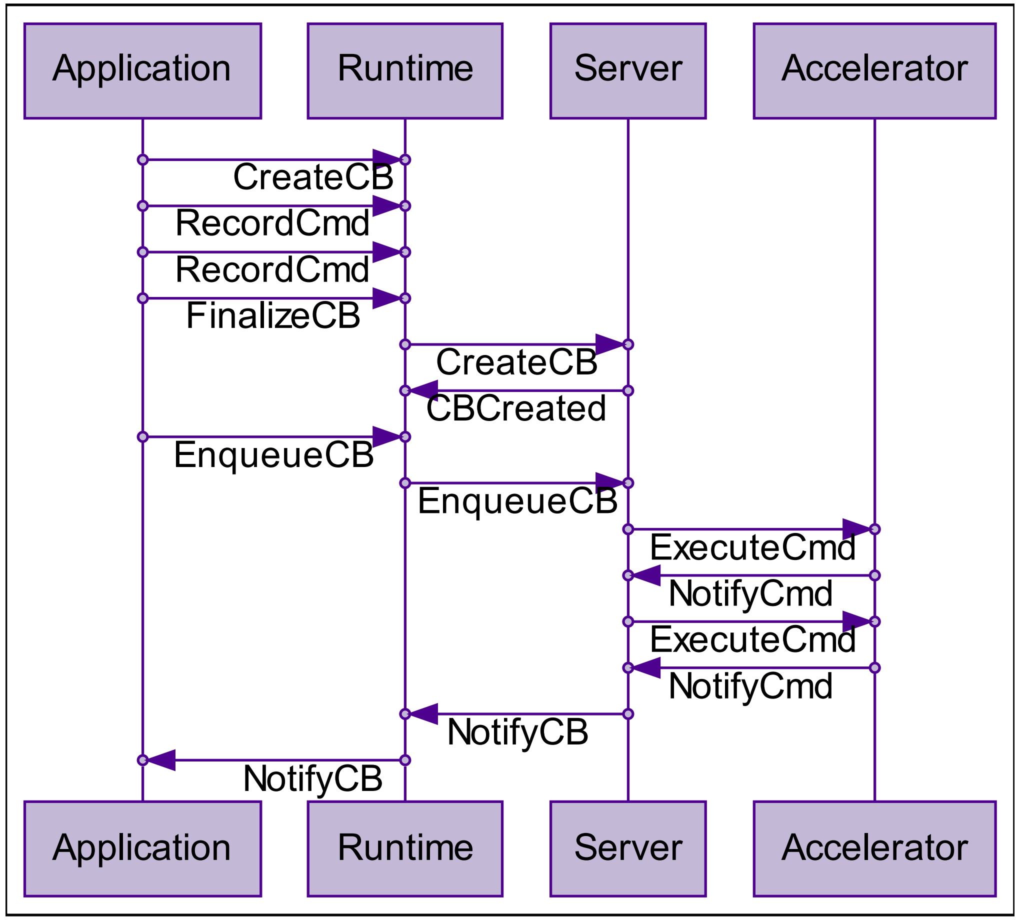
- Server feeds “create command buffer” payload into the command reading logic to generate in-memory representation
  - If underlying driver supports command buffers, creates a “native” command buffer
  - Otherwise simply stores commands in a vector

Executing Command Buffers in `pocld`

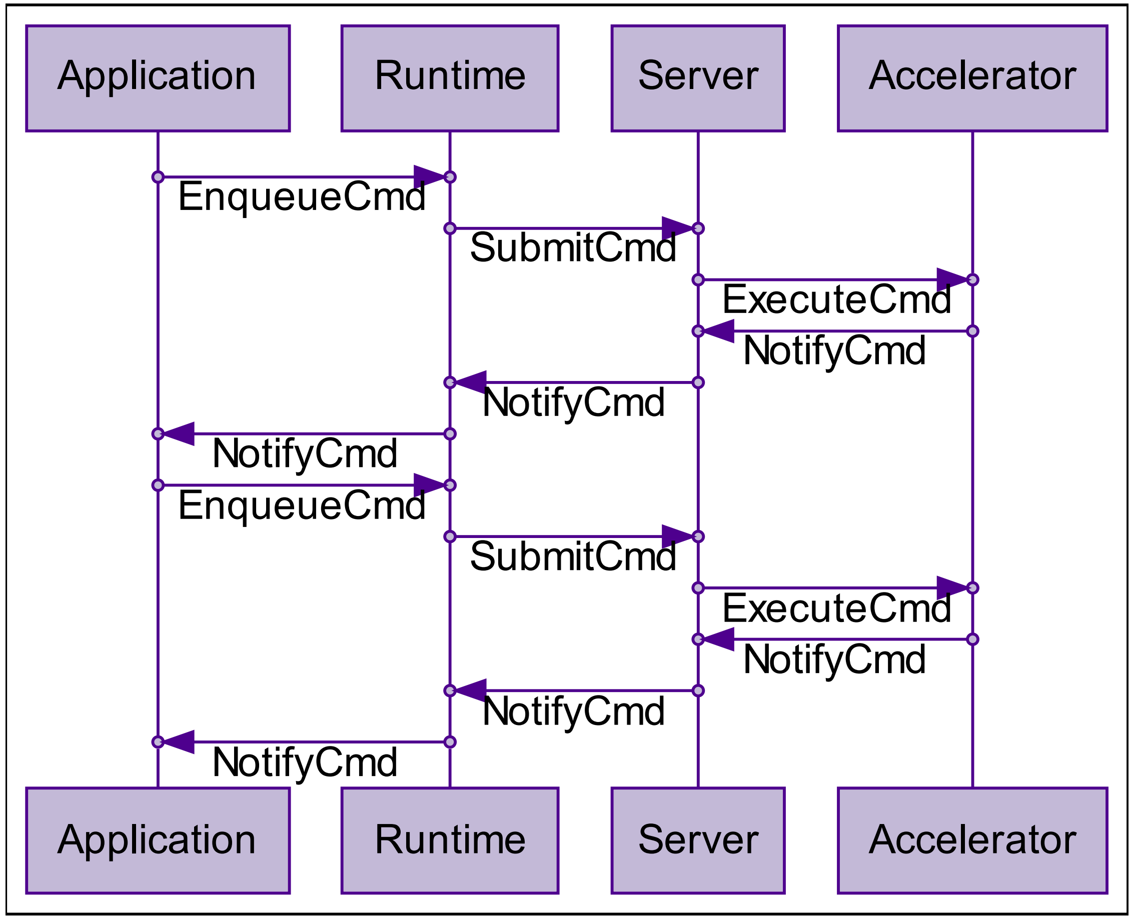
- Upon receiving an “enqueue command buffer” command, call `clEnqueueCommandBufferKHR` with the stored buffer, if available
- If “native” command buffers are not available, iterate through the vector and enqueue commands one by one
  - But don’t send completion notifications etc to the client
  - After all commands are enqueued, enqueue a *marker*
  - Completion of the marker gets reported back to the client with the event ID of the corresponding `clEnqueueCommandBufferKHR` call

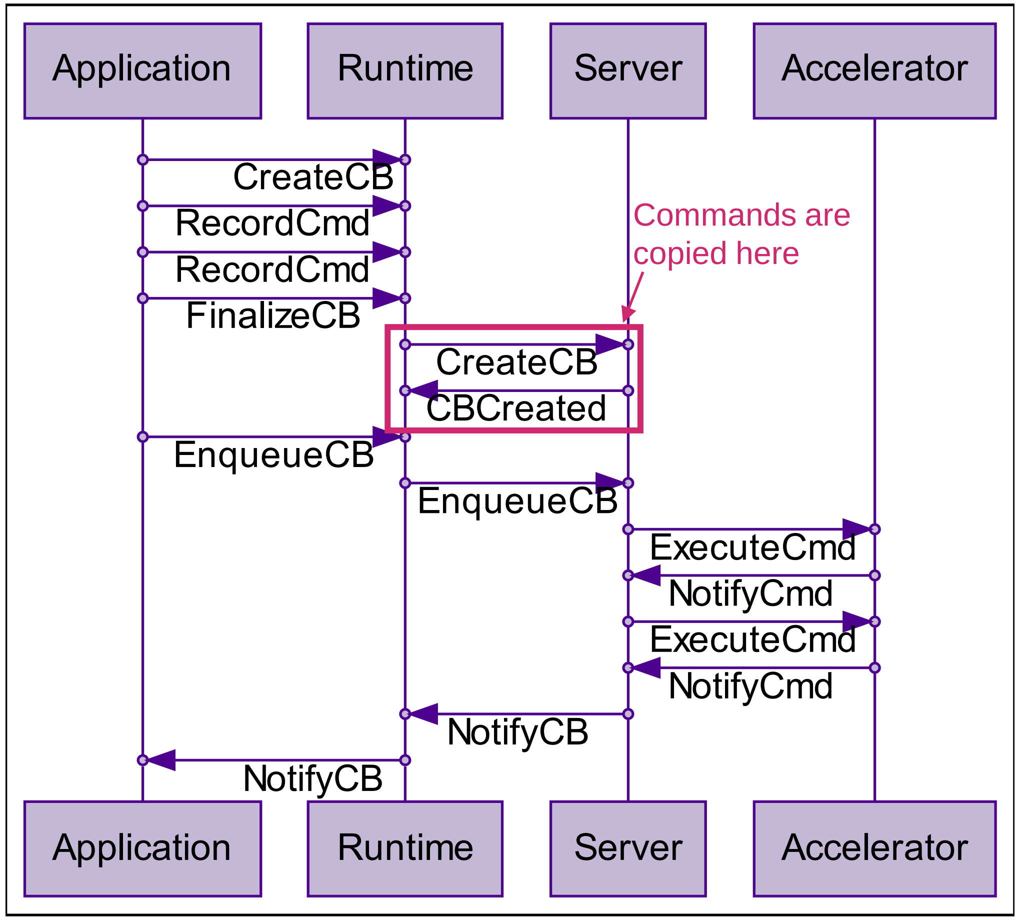
Direct vs Buffered Command Flow





Direct vs Buffered Command Flow

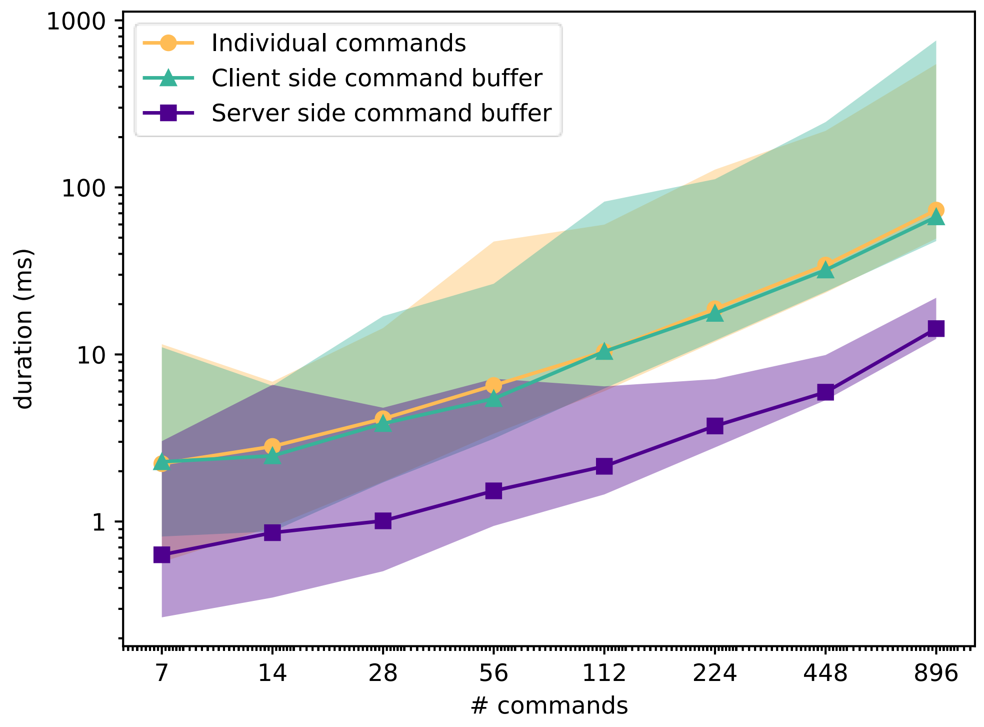




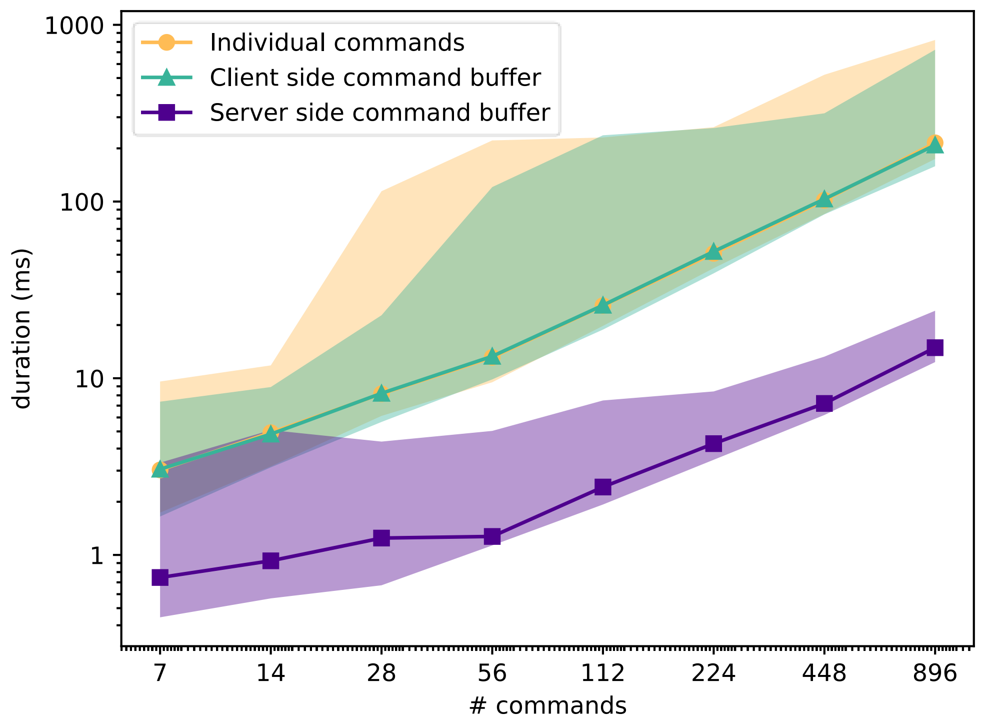
Tampere University

Performance Overview (log axes)

Local Loopback



Gigabit Ethernet



But What About the Buffer Finalize Step?

- Not accounted for in the previous slide
  - For most applications it's a one off operation and the main thing of interest is dispatching the command buffer
- Could still be relevant
  - Dynamically changing command graph
  - One-off command buffers
- Server-side command buffers are sent to server in the finalize step
  - That’s potentially a lot of data, no?

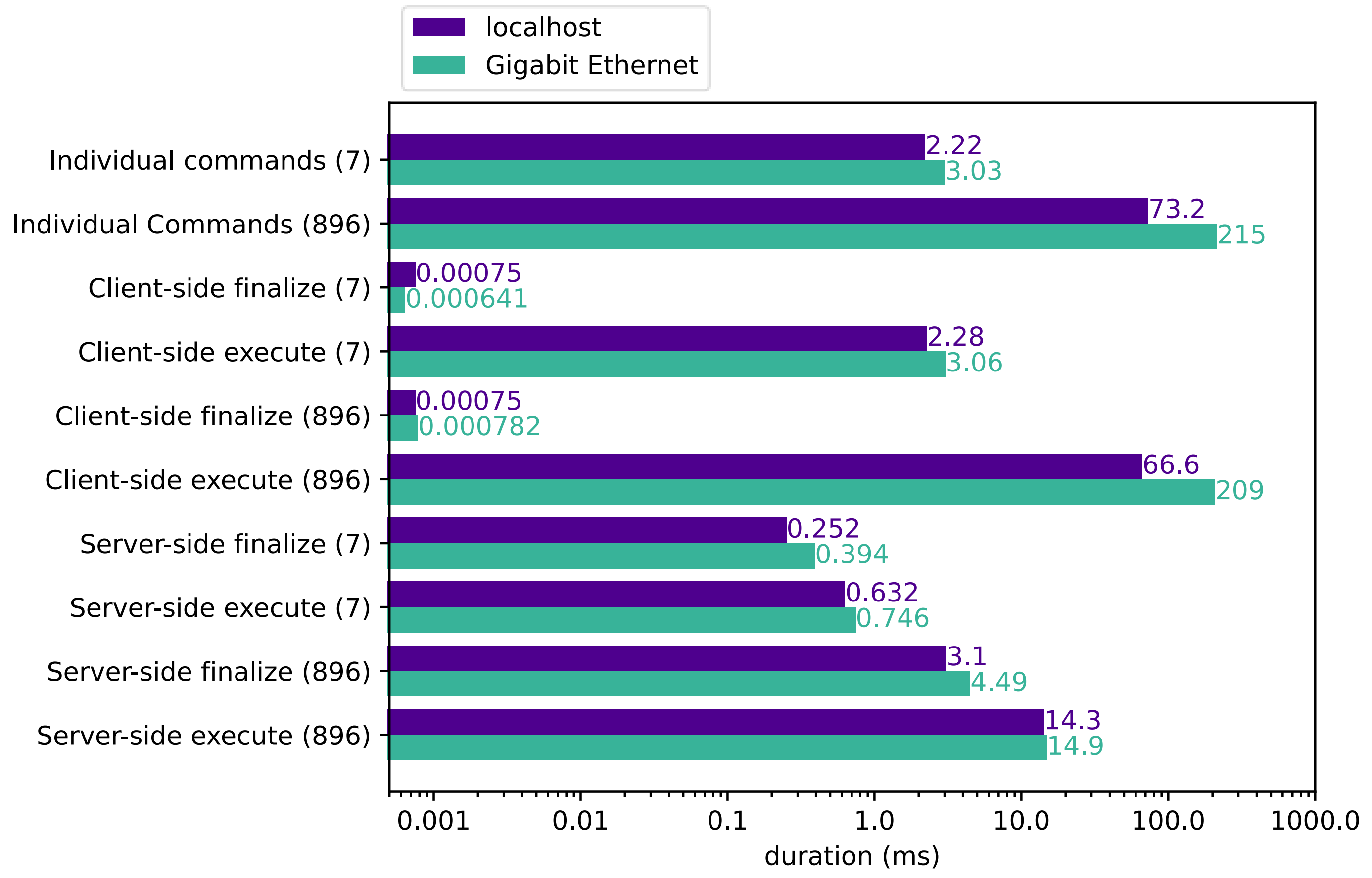
About the Buffer Finalize Step

- Hypothesis: it's negligible in terms of performance
  - Finalization does very little work (locally)
  - Input validation is (mostly) done already while recording
- Server-side command buffers are sent to server in finalize step
  - Expectation: 1 large bulk transfer here is faster than multiple network roundtrips while executing the commands

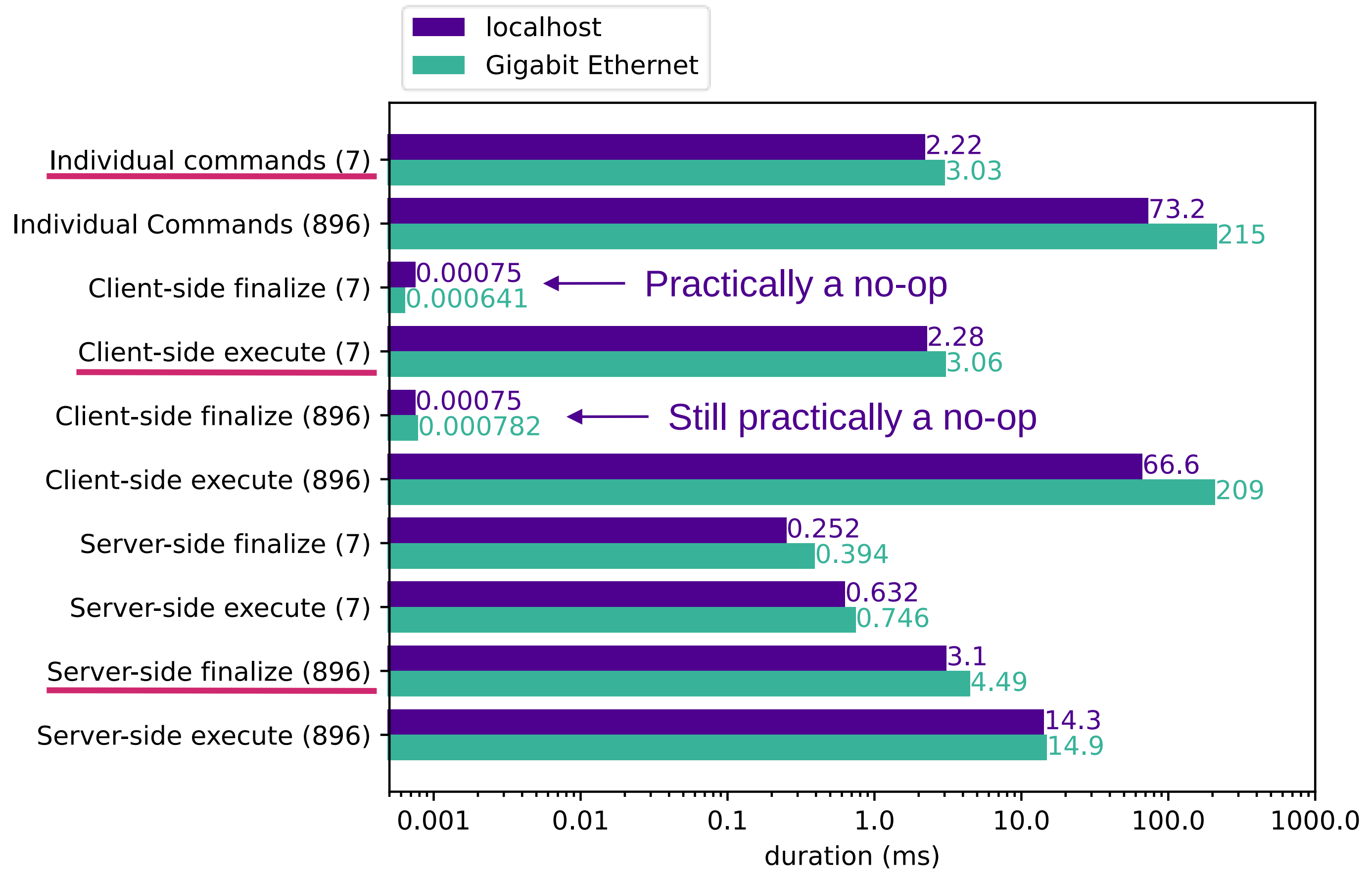
    Note: no compression is applied to the command buffer in transit

Tampere University

Performance Details



Performance Details



Conclusion

- Command buffers are very beneficial to remote OpenCL offloading
  - Up to an order of magnitude faster than submitting commands immediately
- Bulk uploads are fast enough that even single-use command buffers can make a difference

Tampere University

Thank You!

I am Jan Solanti from…

Customized Parallel Computing group at Tampere University

tuni.fi/cpc

Working with…

PoCL project

portablecl.org