Towards Efficient OpenCL Pipe Specification for Hardware Accelerators

Topi Leppänen, Tampere University, Finland

Joonas Multanen, Leevi Leppänen, Pekka Jääskeläinen
Tampere University                  Tampere University                  Tampere University & Intel
Outline

- Background on OpenCL implementations for FPGA
- Issues with the pipe specification in regards to streaming-style execution
  - Suggestions to improve the specification
- A case study on a more dynamic hardware pipe implementation
Outline

- Background on OpenCL implementations for FPGA
- Issues with the pipe specification in regards to streaming-style execution
  - Suggestions to improve the specification
- A case study on a more dynamic hardware pipe implementation
Generic OpenCL Platform
FPGA as OpenCL Device

Host CPU

PCIe

DDR/ HBM

FPGA Card

Static Region

Reconfigurable region

Kernel A

Kernel B

Kernel C

Memory Interconnect

FPGA chip
FPGA as OpenCL Device

- Kernels are pre-compiled with FPGA vendor tooling
- Circuit descriptions need to be synthesized, placed and routed
- The compile-time measured in hours
- The reconfigurable region in the FPGA is programmed with `clCreateProgramWithBinary`
OpenCL Task Pipeline on FPGA

OpenCL host software point-of-view

<table>
<thead>
<tr>
<th>kernel</th>
<th>buffer</th>
<th>buffer</th>
<th>buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>sobel3x3</td>
<td>phase</td>
<td>magnitude</td>
<td>nonmax</td>
</tr>
</tbody>
</table>

Hardware implementation

<table>
<thead>
<tr>
<th>kernel</th>
<th>buffer</th>
<th>buffer</th>
<th>buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA Sobel3x3</td>
<td>DMA Phase</td>
<td>DMA Magnitude</td>
<td>DMA Non-maximum suppression</td>
</tr>
</tbody>
</table>

Since the connectivity is static, there's not much for the programmer to control. The closest OpenCL compliant way to describe this hardware system is to call it a one big kernel.

Separate kernels with independent global memory access.
OpenCL Task Pipeline on FPGA

OpenCL host software point-of-view

- sobel3x3
- phase
- magnitude
- nonmax

buffer
buffer
buffer
buffer

Hardware implementation

- DMA Sobel3x3
- DMA Phase
- DMA Magnitude
- DMA Non-maximum suppression

4K image (~8MB)
4K image (~8MB)
4K image (~8MB)

Configuration #1
sobel3x3 phase magnitude nonmax
Broadcast
FIFO
Global Memory
x2
Configuration #2
Configuration #3
OpenCL Task Pipeline

The specification:
“A command submitted to a device will not launch until prerequisites that constrain the order of commands have been resolved.

…
The command will wait and not launch until all the events in the list are in the state CL_COMPLETE ...
”
Streaming-style execution in OpenCL
Streaming execution

- Task pipelines with a **finer** grain of synchronization than events
  - E.g. partial frame is already sent for the next kernel, while the entire frame is not yet processed
  - Minimizes single-frame latency
  - Minimizes intermediate storage requirement
- Forever-running kernels that do not need to be regularly launched
  - E.g. microphone generates continuous data, no need to launch the processing kernels every n seconds

![Diagram showing streaming execution with Producer and Consumer](image)
OpenCL Pipe Specification

- OpenCL memory object just like Buffer or Image
  - Can be set as kernel arguments
- FIFO-like
- Kernels use read_pipe and write_pipe to push and pop packets
  - Reserve multiple of packets at work-item or work-group level
- Not accessible to host
- Introduced in OpenCL 2.0
  - Made optional in OpenCL 3.0
OpenCL Pipe Memory Model

Producer

Pipe

Consumer

clCreateKernel

clSetKernelArg

clCreatePipe

read_pipe

write_pipe

Producer

Pipe

Consumer

Time
__kernel void producer(__write_only pipe int out_pipe, __global int* A) {
    for (int i = 0; i < N; i++) {
        while(write_pipe(out_pipe, &A[i]));
    }
}

__kernel void consumer(__read_only pipe int in_pipe) {
    int data = 0;
    while(read_pipe(out_pipe, &data));
    // ...compute…
}
The target

Should we include an event dependency between the kernels when submitting?

```
__kernel void producer(__write_only pipe int out_pipe,
                      __global int* A)
{
   for (int i = 0; i < N; i++)
   {
      while(write_pipe(out_pipe, &A[i]));
   }
}

__kernel void consumer(__read_only pipe int in_pipe)
{
   int data = 0;
   while(read_pipe(out_pipe, &data));
   // ...compute…
}
```
The target

No good solutions

With the event dependency:

Without the event dependency:
No good solutions

With the event dependency:

- The specification:
  “A command submitted to a device will not launch until prerequisites that constrain the order of commands have been resolved.

  …

  The command will wait and not launch until all the events in the list are in the state CL_COMPLETE

  …”

Without the event dependency:
No good solutions

With the event dependency:

- The specification:
  
  “A command submitted to a device will not launch until prerequisites that constrain the order of commands have been resolved.
  
  ... The command will wait and not launch until all events in the list are in state CL_COMPLETE ...
  
Without the event dependency:

- Increases latency
- Pipe has to be large enough to hold all the data
No good solutions

With the event dependency:

• The specification:
  “A command submitted to a device will not launch until prerequisites that constrain the order of commands have been resolved.

  …
  The command will wait and not launch until all the events in the list are in the state CL_COMPLETE ...
”

Without the event dependency:

• The specification:
  “The pipe state i.e. contents of the pipe across kernel-instances (on the same or different devices) is enforced at a synchronization point.”

• No guarantee that both of these kernels will make concurrent progress
No good solutions

With the event dependency:

- The specification:
  “A command submitted to a device will not launch until prerequisites that constrain the order of commands have been resolved.

  ...The command will wait and not launch until all the events in the list are in the state CL_COMPLETE ...
”

Without the event dependency:

- The specification:
  “The pipe state i.e. contents of the pipe across kernel-instances (on the same or different devices) is enforced at a synchronization point.”

- No guarantee that both of these kernels will make concurrent progress

- Good chance for a deadlock
OpenCL Pipe Memory Model

- SYCL pipe extension proposal discusses these issues more
- Similar change needed for OpenCL

Proposal #1 for improving the OpenCL pipe specification

Declare in the memory consistency model that pipe read and write operations are eventually visible from the producer to the consumer end of the pipe, without requiring to wait for the whole buffer synchronization points.
OpenCL Pipe Memory Model

- All the data that the kernel needs no longer needs to be ready in global memory when the kernel is launched
  → implementation must support multiple kernel instances *RUNNING* at the same time
- Possible to construct pipe graphs of any size at run-time
  → arbitrarily many concurrent running kernels

**Proposal #1 for improving the OpenCL pipe specification**

Declare in the memory consistency model that pipe read and write operations are eventually visible from the producer to the consumer end of the pipe, without requiring to wait for the whole buffer synchronization points.
Pipes Between Built-in Kernels

- There is no limit to how many built-in kernel instances could be chained together.
- The number of concurrent `RUNNING` kernels can grow arbitrarily large → large number of HW contexts.
Pipes Between Built-in Kernels

Proposal #2 for improving the OpenCL pipe specification

Add a device query `CL_DEVICE_BUILT_IN_KERNELS_RESOURCES` parameter to `clGetDeviceInfo` which would return a list of a number of concurrent hardware contexts for each built-in kernel.
Software Kernel Pipes with Limited HW Contexts

- Propose a new device query parameter `CL_DEVICE_MAX_CONCURRENT_PIPE_KERNEL_INSTANCES` to limit the total number of concurrent HW contexts (the size of pipe graph).
- Implementation can set to 1 if they want to keep the old behavior as defined in the current OpenCL specification.
- Producer-consumer kernels connected with `event` do not count towards this limit.
- Old programs with the `event` synchronization would still work as before.
- User can use `events` to split large graphs into multiple smaller ones.

Proposal #3 for improving the OpenCL pipe specification

Add a device query `CL_DEVICE_MAX_CONCURRENT_PIPE_KERNEL_INSTANCES` and allow `clEnqueueNDRangeKernel` to fail if more than that many concurrent instances are enqueued.
Dynamic Pipe Component
OpenCL Pipe on FPGAs

- Kernels connected together with streaming interfaces
  - Ready-valid-signaling
- FPGA vendor OpenCL implementations only support "static pipe"
  - Pipe connectivity, depth and width need to be defined at compile-time
    - Not spec-compliant
Static Pipe Connectivity

- Kernels connected together with streaming interfaces
  - Ready-valid-signaling
- FPGA vendor OpenCL implementations only support “static pipe”
  - Pipe connectivity, depth and width need to be defined at compile-time
    - Not spec-compliant

- Two options:
  1) Standardize the static pipe
  2) More dynamic pipe implementation
Runtime-defined Pipe Connectivity

- Kernels are connected together with pipes using `clSetKernelArg`
- Connectivity defined at runtime
- *After* the program has been built
- `clCreatePipe` calls are also independent of the program object
- Runtime-defined pipe depth and width
- How to make this work on FPGA?

![Diagram](image-url)
Runtime-defined Pipe Connectivity

- Kernels are connected together with pipes using `clSetKernelArg`
- Connectivity defined at runtime
- After the program has been built
- `clCreatePipe` calls are also independent of the program object
- Runtime-defined pipe depth and width
- How to make this work on FPGA?
Runtime-defined Pipe Connectivity

<table>
<thead>
<tr>
<th>Dynamic pipe parameter</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>Fail any <code>clCreatePipe</code>-call that is larger then HW FIFO size</td>
</tr>
<tr>
<td>Width</td>
<td>Manage interfacing to the fixed width AXI Stream from the kernel-side</td>
</tr>
<tr>
<td>Connectivity</td>
<td>AXI Stream TDEST-based routing of packets</td>
</tr>
</tbody>
</table>
Dynamic Kernel Pipeline

- Hardware accelerators exposed to OpenCL host as built-in kernels
Dynamic Kernel Pipeline

- Computation graph can now be constructed at run-time
- Processing pipeline can be changed based on e.g. environmental conditions or user input
- E.g. add pre-processing kernels dynamically
Dynamic Kernel Pipeline

- In this example, `CL_DEVICE_BUILT_IN_KERNELS_RESOURCES` for
  - `Broadcast`-kernel is 2
  - Every other kernel has 1

### OpenCL host software point-of-view

- `sobel3x3`
- `phase`
- `magnitude`
- `nonmax`

### Hardware implementation

<table>
<thead>
<tr>
<th></th>
<th>Buffer 2 pipe</th>
<th>Pipe 2 buffer</th>
<th>Broadcast x2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sobel3x3</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>phase</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>magnitude</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>nonmax</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- `DMA`
- `MM2S`
- `S2MM`
- `DMA`
- `Broadcast`
- `Phase`
- `Magnitude`
- `Non-maximum suppression`

Global Memory
Evaluation

OpenCL host software point-of-view

Configuration #1
Separate kernels with independent global memory access
sobel3x3 phase magnitude nonmax
DMA Global Memory

Configuration #2
Static AXI Stream connectivity between kernels
Since the connectivity is static, there’s not much for the programmer to control. The closest OpenCL compliant way to describe this hardware system is to call it a one big kernel.
sobel3x3 phase magnitude nonmax
DMA Global Memory

Configuration #3
Dynamic AXI Stream connectivity
DMA MM2S

Hardware implementation

Global Memory

Sobel3x3
buffer
DMA

Sobel3x3
buffer
DMA

Phase
DMA

Magnitude
DMA

Non-maximum suppression
DMA

FIFO

Non-maximum suppression
DMA

S2MM
DMA

Canny
buffer
DMA

Sobel3x3
buffer
DMA

Phase
DMA

Magnitude
DMA

Non-maximum suppression
DMA

FIFO

Non-maximum suppression
DMA

S2MM
DMA

Broadcast

Global Memory

Configuration #2
Configuration #3
Evaluation

<table>
<thead>
<tr>
<th>Configuration #1</th>
<th>OpenCL host software point-of-view</th>
<th>Hardware implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate kernels with independent global memory access</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sobel3x3 buffer</td>
<td>DMA Sobel3x3</td>
<td>DMA Non-maximum suppression</td>
</tr>
<tr>
<td>phase buffer</td>
<td>DMA Phase</td>
<td>DMA Magnitude</td>
</tr>
<tr>
<td>magnitude buffer</td>
<td>DMA Broadcast</td>
<td>DMA Non-maximum suppression</td>
</tr>
<tr>
<td>nonmax buffer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Since the connectivity is static, there’s not much for the programmer to control. The closest OpenCL compliant way to describe this hardware system is to call it a one big kernel.

<table>
<thead>
<tr>
<th>Configuration #2</th>
<th>Static AXI Stream connectivity between kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Since the connectivity is static, there’s not much for the programmer to control. The closest OpenCL compliant way to describe this hardware system is to call it a one big kernel.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration #3</th>
<th>Dynamic AXI Stream connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer2pipe pipe2buffer broadcast x2</td>
<td>DMA MM2S FIFO FIFO</td>
</tr>
<tr>
<td>sobel3x3 pipe buffer</td>
<td>AXI Stream Crossbar</td>
</tr>
<tr>
<td>phase buffer pipe</td>
<td>AXI Stream Crossbar</td>
</tr>
<tr>
<td>magnitude buffer pipe</td>
<td>AXI Stream Crossbar</td>
</tr>
<tr>
<td>nonmax buffer pipe</td>
<td>S2MM DMA</td>
</tr>
</tbody>
</table>

Table 1: Latency and area results for (partial) Canny edge detection of a 4K image on Alveo U280 FPGA.

<table>
<thead>
<tr>
<th>Latency</th>
<th>LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.9 ms (100 FPS)</td>
<td>135150 (10.4%)</td>
</tr>
<tr>
<td>2.5 ms (400 FPS)</td>
<td>127549 (9.78%)</td>
</tr>
<tr>
<td>3.9 ms (260 FPS)</td>
<td>185069 (14.2%)</td>
</tr>
</tbody>
</table>
Spatial Pipelines with Compiled Kernels

\[ \text{CL\_DEVICE\_MAX\_CONCURRENT\_PIPE\_KERNEL\_INSTANCES} = 4 \]
Conclusion

- The current OpenCL pipe specification is not well-suited for parallel, spatial pipelines
- Fixing the pipe specification could enable novel, spatial architectures programmable via OpenCL

A dynamic pipe component to implement the runtime-defined pipe connectivity
Towards Efficient OpenCL Pipe Specification for Hardware Accelerators

Topi Leppänen, Joonas Multanen, Leevi Leppänen, Pekka Jääskeläinen
Tampere University
topi.leppanen@tuni.fi

github.com/cpc/AFOCL