Intel® FPGA host pipe extension for OpenCL™ applications

Michael Kinsner, Dirk Seynhaeve
IWOCL 2018
Topics

1. FPGA overview
2. Motivating application classes
3. Host pipes
4. Some data
FPGA: Fine-grained Massive Parallelism

Intel® Stratix® 10 FPGA: Over 5 Million Basic Elements!
Many Design Entry Options

OpenCL defines the full system (e.g. host, memory hierarchy, host ↔ device communication)

https://www.altera.com/products/design-software/overview.html
How Can You Compile Software to Hardware?

OpenCL™ Code

```
kernel void
add(global int* Mem) {
...
    Mem[100] += 42*Mem[101];
}
```

Instruction Level

```
add:
R0 ← Load Mem[100]
R1 ← Load Mem[101]
R2 ← Load #42
R2 ← Mul R1, R2
R0 ← Add R2, R0
Store R0 → Mem[100]
```

- Instruction sequence can either be executed sequentially in time (*temporal computing*), or executed in parallel in space (*spatial computing*)
Executing in Time vs. Executing in Space

Execute in Time
CPU, GPU

Execute in Space
FPGA

R0 $\leftarrow$ Load Mem[100]
R1 $\leftarrow$ Load Mem[101]
R2 $\leftarrow$ Load #42
R2 $\leftarrow$ Mul R1, R2
R0 $\leftarrow$ Add R2, R0
Store R0 $\rightarrow$ Mem[100]

Fine grained pipeline parallelism
Kernels consume “space”

Conceptually map to regions of the FPGA in Intel’s OpenCL implementation

- Pipeline, data, and task parallelism

- Efficient use of FPGA architecture
- “Concurrent execution”

- Data flow processing
- Fine grained on-chip communication
OpenCL 1.x – Host/Device Bulk Communication

Host system

Global Memory

FPGA

Host system

CPU

GPU

Prepare input data

clEnqueueWriteBuffer

Kernel execution

clEnqueueReadBuffer

Prepare input data

clEnqueueWriteBuffer

Kernel execution

clEnqueueReadBuffer

Latency

Use output data

or clEnqueueUnmapMemObject()

Use output data

or clEnqueueMapBuffer()

OpenCL Synchronization Point
Motivating Classes of Applications
Long Running Kernel for Bursting or Massive Data

Long running / persistent kernel

- Processing more data than fits in global memory, using a single kernel instance
  - Or for lower latency processing of data arriving piecemeal on host
- Reduced latency processing of bursting data
  - Avoid launch overhead and state reconstruction
Network Routing / Processing

Routing table updates – low rate, non-periodic

- FPGAs have rich I/Os
- Often want long-running kernels
- Polling for memory-based host updates expensive
  - Plus memory consistency challenges
- FIFO semantics ideal
Streaming Content Analysis

Low rate, non-periodic detection events signaled to host

- Rich I/Os
- Long running kernels
- Data consistency challenges
- FIFO semantics ideal
Two Use Models

High throughput streaming

Asynchronous signaling/control
Two Use Models – OpenCL 1.x Challenges

- Data availability
- Cost of memory polling
- FIFO model
OpenCL 2.0 Pipes – A Reminder

Kernels

code

```c
kernel void producer (write_only pipe uint c0) {
    for (uint i=0; i<10; i++) {
        write_pipe(c0, &i);
    }
}

c kernel void consumer (read_only pipe uint c0, global uint * restrict dst) {
    for (int i=0; i<5; i++) {
        read_pipe(c0, &dst[i]);
    }
}
```

Host

code

```c
... cl_mem pipe = clCreatePipe(context, 0, sizeof(cl_int), SIZE, NULL, &status);
status = clSetKernelArg producer, 0, sizeof(cl_mem), &pipe);
status = clSetKernelArg consumer, 0, sizeof(cl_mem), &out_buffer);
status = clSetKernelArg consumer, 1, sizeof(cl_mem), &pipe);
...```

OpenCL 2.0 pipes: Communication is between kernels
Intel® Host Pipes

Allow pipes to be read/written from the **host program** as well as in kernels
Host Pipe Extension

Small extension to OpenCL 2.x pipe API: cl_intel_fpga_host_pipe

• New flags legal in clCreatePipe():

<table>
<thead>
<tr>
<th>API Enum</th>
<th>Parent Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_MEM_HOST_READ_ONLY</td>
<td>clGetKernelArgInfo()</td>
</tr>
<tr>
<td>CL_MEM_HOST_WRITE_ONLY</td>
<td>clGetKernelArgInfo()</td>
</tr>
<tr>
<td>CL_MEM_READ_ONLY</td>
<td>clGetDeviceInfo()</td>
</tr>
<tr>
<td>CL_MEM_WRITE_ONLY</td>
<td>clGetDeviceInfo()</td>
</tr>
<tr>
<td>CL_PIPE_FULL</td>
<td>clWritePipeIntelFPGA()</td>
</tr>
<tr>
<td>CL_PIPE_EMPTY</td>
<td>clReadPipeIntelFPGA()</td>
</tr>
</tbody>
</table>

• New query / status enums:

```
cl_mem read_pipe = clCreatePipe(
    context,               
    CL_MEM_HOST_READ_ONLY, sizeof(cl_int),
    128, // Number of packets that can be buffered
    NULL, &error
);
```
Kernel Interface

Used like normal OpenCL 2.x pipes
- Additional kernel argument attribute
- No reservation functionality (the OpenCL 2.x feature)

C kernel language:

```c
kernel void foo(__attribute__((intel_host_accessible)) write_only pipe int p) { .... }
read_pipe(P, &val)
write_pipe(P, &val)
```

C++ kernel language:

```cpp
kernel void foo([[cl::intel_host_accessible]] cl::pipe<int, cl::pipe_access::write> p) { .... }
p.write(val)
p.read(&val)
```
Host Interface – Low Rate Signaling

Simple interface

- Single word read/write
- Data transferred “as soon as possible”

```c
cl_int clReadPipeIntelFPGA( cl_mem pipe, void *ptr );
cl_int clWritePipeIntelFPGA( cl_mem pipe, const void *ptr );

// Create pipes, kernels, other startup code
....

// Bind pipes to kernels
clSetKernelArg(read_kern, 0, sizeof(cl_mem), (void *)&write_pipe);
clSetKernelArg(write_kern, 0, sizeof(cl_mem), (void *)&read_pipe);

// Enqueue kernels
....

int float2;
if ( !clReadPipeIntelFPGA( read_pipe, &val ) ) {
    int result = clWritePipeIntelFPGA( write_pipe, (int)(val.x + val.y));
    // Check write success/failure and handle
    ....
}
```
void * clMapHostPipeIntelFPGA(
    cl_mem pipe,
    cl_map_flags map_flags,
    size_t requested_size,
    size_t * mapped_size,
    cl_int * errcode_ret);

cl_int clUnmapHostPipeIntelFPGA(
    cl_mem pipe,
    void * mapped_ptr,
    size_t requested_size,
    size_t * unmapped_size);

... cl_int *buffer;

buffer = (cl_int*) clMapHostPipeIntelFPGA( pipe, 0, ask_size, &got_size, &status );

// Write data to buffer
...

clUnmapHostPipeIntelFPGA( pipe, buffer, buffer_size, NULL );
FIFO Access Within Kernels

Checking FIFO for data availability is cheap

- Implicit control signals (ready/full), and low latency

```c
kernel void foo ( global int *G, ... ) {
    if (G[ get_local_id(0) ]) { ... }
}

kernel void
foo ( read_only pipe int4 P ... ) {
    int4 val;
    if (0 == read_pipe(P, &val)) { ... }
}
```

![Diagram showing FIFO access within kernels with controls for loading and checking availability](diagram.png)
Visibility and Latency

Additional memory model guarantee

- Data written to a pipe will eventually be visible on the read endpoint, without an OpenCL synchronization point. It is understood that an OpenCL implementation will make the data visible to the read endpoint “as soon as possible”
- No synchronization side effects with other pipes or memory

The host pipe API supports low latency communication

- An OpenCL extension is not enough to guarantee latency
  - Board support package
  - Drivers/OS
  - System load
- The host pipe API was designed to enable latency-sensitive applications
  - Talk to board and system provider if guarantees are required
Host Pipe Microbenchmark

- Results from an Intel® Arria® 10 GX FPGA Development Kit
- Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50GHz
- Two CPU threads, each managing one host pipe direction. Loopback kernel
- 'aocl diagnose': Buffer transfer speed µbenchmark, that ships with the Intel® FPGA SDK for OpenCL™

Host pipe microbenchmark

<table>
<thead>
<tr>
<th>Map size [KB]</th>
<th>Full duplex speed [GB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host pipe loopback</td>
<td>0</td>
</tr>
<tr>
<td>(sum tx and rx) Buffer xfer µbench peak (aocl diagnose)</td>
<td></td>
</tr>
<tr>
<td>Theoretical link (no protocol)</td>
<td>18</td>
</tr>
</tbody>
</table>

Host pipes require platform (BSP) support
Now Available!

Host pipes are shipping in the Intel® FPGA SDK for OpenCL™ 18.0
- Reference platform has some minor restrictions that will be relaxed in the future
  - # host pipes, width of host pipes
  - Some queries
Legal Notice and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks.

Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

© 2018 Intel Corporation.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

* Other names and brands may be claimed as the property of others.
Legal Disclaimer and Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2018, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804