OpenCL: FPGA Acceleration for Software Programmers

Marco Mauri
Agenda

- FPGA Overview for software programmers
- Technology trends & challenges
  - Why FPGA & OpenCL?
- OpenCL and Altera SDK Overview
FPGA Overview for Software Programmers
FPGA Architecture: Fine-grained Massively Parallel

Let’s zoom in

- Millions of reconfigurable logic elements
- Thousands of 20Kb memory blocks
- Thousands of Variable Precision DSP blocks
- Dozens of High-speed transceivers
- Multiple High Speed configurable Memory Controllers
- Multiple ARM® Cores
FPGA Architecture: Basic Elements

Configured to perform any 1-bit operation:
AND, OR, NOT, ADD, SUB
FPGA Architecture: Flexible Interconnect

Basic Elements are surrounded with a flexible interconnect
Wider custom operations are implemented by configuring and interconnecting Basic Elements.
Wider custom operations are implemented by configuring and interconnecting Basic Elements.

16-bit add

32-bit sqrt

Your custom 64-bit bit-shuffle and encode
FPGA Architecture: Memory Blocks

Can be configured and grouped using the interconnect to create various cache architectures
FPGA Architecture: Memory Blocks

Memory Block
20 Kb

can be configured and grouped using the interconnect to create various cache architectures

Lots of smaller caches

Few larger caches
FPGA Architecture: Floating Point Multiplier/Adder Blocks

Dedicated floating point multiply and add blocks
FPGA Architecture: Configurable Routing

Blocks are connected into a custom data-path that matches your application.
Mapping a simple program to an FPGA

High-level code

Mem[100] += 42 * Mem[101]

CPU instructions

R0 ← Load Mem[100]
R1 ← Load Mem[101]
R2 ← Load #42
R2 ← Mul R1, R2
R0 ← Add R2, R0
Store R0 → Mem[100]
First let’s take a look at execution on a *simple* CPU

Fixed and general architecture:

- General “cover-all-cases” data-paths
- Fixed data-widths
- Fixed operations
Load constant value into register

Very inefficient use of hardware!
CPU activity, step by step

R0 ← Load Mem[100]

R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]
On the FPGA we unroll the CPU hardware...

R0 ← Load Mem[100]

R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]
… and specialize by position

R0 ← Load Mem[100]

R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]

1. Instructions are fixed. Remove “Fetch”
... and specialize

1. Instructions are fixed.
   Remove “Fetch”
2. Remove unused ALU ops

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]
... and specialize

1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store

R0 ← Load Mem[100]

R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]
... and specialize

1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store
4. Wire up registers properly! And propagate state.

R0 ← Load Mem[100]

R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]
... and specialize

1. Instructions are fixed.
   Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store
4. Wire up registers properly!
   And propagate state.
5. Remove dead data.

R0 ← Load Mem[100]
R1 ← Load Mem[101]
R2 ← Load #42
R2 ← Mul R1, R2
R0 ← Add R2, R0
Store R0 → Mem[100]
... and specialize

1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store
4. Wire up registers properly! And propagate state.
5. Remove dead data.
6. Reschedule!

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]
Custom data-path on the FPGA matches your algorithm!

High-level code

\[
\text{Mem}[100] += 42 \times \text{Mem}[101]
\]

Custom data-path

Build exactly what you need:
- Operations
- Data widths
- Memory size & configuration

Efficiency:
- Throughput / Latency / Power
Technology Trends & challenges
High Level Market Trends

- Application complexity and compute requirements is ever increasing
  - Larger FPGAs (5.5 Million LEs) require longer development times for FPGA developers
  - Software development resources are growing while hardware engineers are diminishing
  - Faster to develop in software and accelerate in hardware

- Distributed computing required to solve growing data requirements
  - SoC and embedded computing is becoming common place in the industry
General Challenges

- Distributed computing challenged by data transfers among the nodes and datacenter
  - infrastructure / data center bottleneck

- Maintaining coherency throughout scalable system

- Overloading CPUs capabilities
  - Frequencies are capped
  - Processors keep adding more cores
  - Need to coordinate all the cores and manage data

- Product life cycles are long
  - SW Updates to existing HW
  - GPUs lifespan is short
    - Require re-optimization and regression testing between generations

- Power dissipation of CPUs and GPUs limits system size
OpenCL and FPGAs Address Markets Challenges

- **Power efficient acceleration**
  - Typically 1/5 power of GPU and orders of magnitude more performance per watt of CPU

- **FPGA lifecycle over 15 years**
  - GPUs lifespan is short
    - Require re-optimization testing between generations
    - FPGA OpenCL code retargeted to future devices without modification

- **Our OpenCL flow abstracts away FPGA hardware flow**
  - Puts FPGA into software engineers hands

- **Shared virtual memory to keep data coherency among kernels**
  - IBM CAPI and Intel QPI

- **Our OpenCL SDK allows for streaming IO channels and kernel channels**
  - Data movement without host involvement
  - Low latency data transmissions to accelerator
OpenCL and Altera SDK

Overview
What is OpenCL?

A software programming model for software engineers and a software methodology for system architects
- First industry standard for heterogeneous computing

Provides increased performance with hardware acceleration
- Low Level Programming language
- Based on ANSI C

Open, royalty-free, standard
- Managed by Khronos Group
- Altera active member
- Conformance requirements
  - V1.0 is current reference
  - V2.0 is current release
- [http://www.khronos.org](http://www.khronos.org)
OpenCL Use Model: Abstracting the FPGA away

Host Code

```c
main() {
    read_data( ... );
    manipulate( ... );
    clEnqueueWriteBuffer( ... );
    clEnqueueNDRange(...,sum,...);
    clEnqueueReadBuffer( ... );
    display_result( ... );
}
```

OpenCL Accelerator Code

```c
__kernel void sum
(__global float *a,
__global float *b,
__global float *y)
{
    int gid = get_global_id(0);
    y[gid] = a[gid] + b[gid];
}
```
Superior Software Development Flow

1. Modify kernel.cl
2. x86 Emulator (sec)
3. Optimization Report (sec)
4. Profiler (hours)

System performance met?

Functional Bugs?

Stall-free pipeline? Memory coalesced?

DONE!
The Only Custom Accelerator Solution: Platforms

**Platforms**
- PCIe gen2x8 Host Interface
- OpenCL Domain
- DDR3 Memory Interface
- QDRII Memory Interface
- 10Gb MAC/UOE Data Interface
- PCIe gen2x8 Host Interface

**IO Infrastructure**

- DDR
- DDR
- QDR
- QDR
- QDR
- 10G Network
- Host

**Prebuilt BSP with standard HDL Tools by FPGA Developer**

**Built with Altera OpenCL Compiler**
Channels Advantage

Standard OpenCL

Altera Vendor Extension
IO and Kernel Channels

channel int DataChannel;

kernel producer(...) {
    write_channel_altera(DataChannel, value);
}

kernel consumer(...) {
    value = read_channel_altera(DataChannel);
}
Altera SDK for OpenCL Competitive Differentiator

Altera’s SDK for OpenCL has proven to be a powerful solution for many vendors

- Won *design tool and development software Elektra* award in Europe
- Won *Ultimate Product of the Year* for 2014
- Actively being used today:

  “I was extremely happy to get a great performance with such low effort. I was so impressed with how powerful the Altera tool was! 😊”

--- Senior Engineer, Altera OpenCL Customer
Conformant to OpenCL

- OpenCL v1.0 specification
  - >8500 Programs tested
- Supports Arm Host
  - CV and AV SoC

Conformant Products

<table>
<thead>
<tr>
<th>Company</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>2013-09-11</td>
<td>OpenCL 1.2</td>
</tr>
<tr>
<td>Windows 8.1 64-bit pre-release build running on x86_32 with minimum requirement of SSE 4_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Confidential Product</td>
<td>Windows 8 64-bit RTM</td>
<td></td>
</tr>
<tr>
<td>Intel</td>
<td>2013-08-23</td>
<td>OpenCL 1.2</td>
</tr>
<tr>
<td>Windows 8 64-bit 9200 (RTM) running on Confidential x86_32 with minimum requirement of SSE 4_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Altera Corporation</td>
<td>2013-08-03</td>
<td>OpenCL 1.0</td>
</tr>
<tr>
<td>Nallatech PCIe-385N (Altera Stratix V A7)</td>
<td>Altera Stratix V FPGA</td>
<td></td>
</tr>
<tr>
<td>Intel</td>
<td>2013-07-23</td>
<td>OpenCL 1.2</td>
</tr>
</tbody>
</table>

http://www.khronos.org/conformance/adopters/conformant-companies
http://www.khronos.org/conformance/adopters/conformant-products
Additional Altera OpenCL Collateral

- White papers on OpenCL
- OpenCL online demos
- OpenCL design examples
- Instructor-Led training
  - Parallel Computing with OpenCL Workshop by Altera – (1 Day)
  - Optimization of OpenCL for Altera FPGAs Training by Altera – (1 Day)
- Online training
  - Introduction to Parallel Computing with OpenCL
  - Writing OpenCL Programs for Altera FPGAs
  - Running OpenCL on Altera FPGAs
  - Single-Threaded vs. Multi-Threaded Kernels
  - Building Custom Platforms for Altera SDK for OpenCL
- OpenCL board partners page
Thank You