CoRAM++: Supporting Data-Structure-Specific Memory Interfaces for FPGA Computing

Gabriel Weisz and James C. Hoe
The FPGA vs GPU Debate at ISCA

Whither toolchains/libraries for FPGAs?
CoRAM++ Provides Convenience and Performance

Application Component

Application Component

Application Component

Data Interface

Data Interface

Data Interface

Auto-Generated Data Distribution Network

FPGA Chip Boundary

Application developer chooses data-structure-specific interfaces

Built-in Interface (Fixed Functionality)

DRAM Interface

DRAM Interface
CoRAM++ Provides Convenience and Performance

Application Component 

Stream Interface

Application Component

Array Interface

Application Component

Linked List Interface

Auto-Generated Data Distribution Network

Linked List Engine

FPGA Chip Boundary

Application developer chooses data-structure-specific interfaces

Built-in Interface (Fixed Functionality)

Auto-Generated Data Distribution Network

Pointer Chasing Module Connected to DRAM Interface

DRAM Interface

DRAM Interface
Outline

• Introduction

• The CoRAM++ Programming Environment

• Evaluation

• Conclusion
The CoRAM++ Programming Environment

- CoRAM++ applications are decomposed into hardware kernels and control threads
- Hardware kernels are created using any hardware design methodology
- Hardware kernels only interact with local data
- Control threads are multi-threaded software modules that manage data transfers and kernel invocations
- Control threads are compiled to state machines

Control Thread

Hardware Kernel

Data Interface

DRAM Interface
CoRAM++ Data-Structure-Specific Interfaces

<table>
<thead>
<tr>
<th>Control Thread Commands</th>
<th>Hardware Kernel Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>write_stream</td>
<td>FIFO-style ports</td>
</tr>
<tr>
<td>read_stream</td>
<td></td>
</tr>
<tr>
<td>check_stream</td>
<td></td>
</tr>
<tr>
<td>write_array</td>
<td>SRAM or FIFO-style ports</td>
</tr>
<tr>
<td>read_array</td>
<td></td>
</tr>
<tr>
<td>write_sequence</td>
<td></td>
</tr>
<tr>
<td>read_sequence</td>
<td></td>
</tr>
<tr>
<td>check_array</td>
<td></td>
</tr>
<tr>
<td>list_to_stream</td>
<td>FIFO-style ports</td>
</tr>
<tr>
<td>list_from_stream</td>
<td></td>
</tr>
<tr>
<td>merge_lists</td>
<td></td>
</tr>
<tr>
<td>check_list</td>
<td></td>
</tr>
</tbody>
</table>
Experimental Configuration

• Streaming, array, linked list access patterns
• Bandwidth-bound experiments on an Terasic DE4 FPGA with 2 DRAM interfaces
• Latency-bound experiments also include 2 Xilinx FPGAs:
  - Soft-logic Virtex-6
  - Zynq with 2 ARM cores (used only to bring data onto the board)
Control thread is 4 lines of code:
1. `int size=read_host_data(host_handle);`
2. `issue_read_stream(reader_handle, size);`
3. `issue_write_stream(writer_handle, size);`
4. `while (check_stream(writer_handle)) {}`
Streaming Results

This is the best the DE4 can do

256-cycle data transfer latency and 242-cycle kernel latency
Array Experiments

• 2D and 3D DFTs require array traversal along multiple directions.
• The application supports run-time selection of row-major or tiled data layout.
Array Results

CoRAM++ applications allow run-time selection of traversal order and data layout.

Reference used hand-designed data path, custom-coded address generation.

- Row-major inline conversion
- Row-major extra pass
- Fully-Tiled
- Akin, et al., FCCM 2013
Linked List Traversal Setup

Data layout in DRAM:

- **Next Pointer**
- **Data Pointer**
- **Node**
- **Payload**

**“Best Case”**
Packed Lists

**“Worst Case”**
Strided Lists

Payload size matches the DRAM interface

8k

Pointers are 4 bytes

Linked List Data

Payload data

Linked List Data

Payload data

Linked List Data

Payload data
Linked List Traversal Results

- Application Component
- Application Component
- Application Component
- Stream Interface
- Array Interface
- Linked List Interface
- Linked List Engine
- DRAM Interface
- DRAM Interface

![Traversal Time (ms)]

- Best Case
- Worst Case

- Terasic DE4
- Xilinx ML605
- Xilinx ZC706

- 5X Faster
- 2X Faster
Related Work

Leap [Yang+ FPL 2014]


RCMW [Kirchgessner+ ASAP 2013]

APMC [Hussain+ HPCS 2014]
Conclusions

• CoRAM++ offers ease of use and good memory access performance through an extensible library of data-structure-specific interfaces.

• Future work includes more data structures and pointer-chasing acceleration on tightly-coupled hard-logic processor cores.

• Thank you to:
  - NSF for funding.
  - Altera, Xilinx, and Bluespec for hardware, tools, and support.
  - Berkin Akin for help with the Spiral DFT and permutation generators.