Πολυτεχνείο Κρήτης

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

Τρίμας Χρήστος
Πίσκοπος Διονύσης

LAB59142109

# Contents

- Introduction
- Motivation
- CoRAM vs CoRAM++
- Background and Related Work
- Supported Data Structures and Interfaces
- Implementation of the Interfaces
- Evaluation
- Results

# Introduction

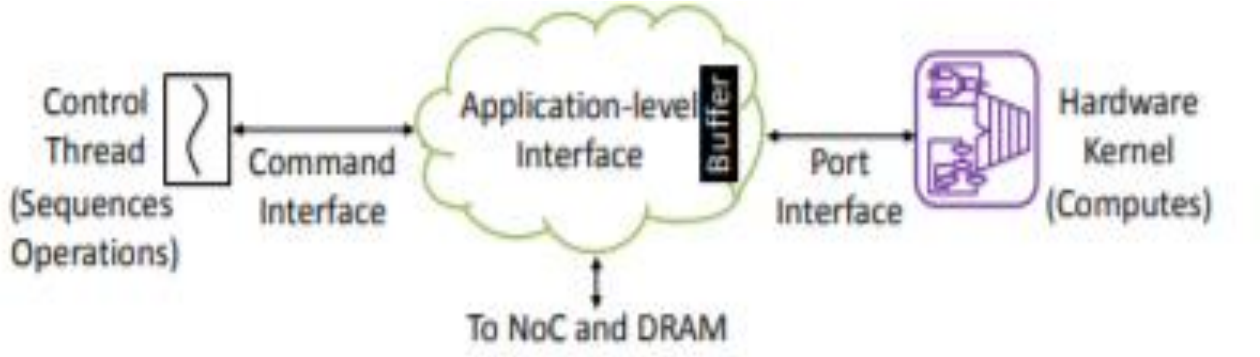
- Facilitating DRAM access is an essential part of an application programming environment for FPGA computing.
- Existing FPGA applications support only simple memory access patterns.
- Introducing CoRAM++, a programming environment for FPGA.

# Motivation

- Accessing data in external DRAM, essential for FPGA computing.
- Before CoRAM++, application level interfaces to simplify interaction with DRAM and enable application portability.
- In need for application level interface that supports complex data structures.

# CoRAM/CoRAM++

- Decomposition of an FPGA application.
- CoRAM: Protection of the kernels. How?
- Not direct communication between Hardware kernels and DRAM.



# CoRAM/CoRAM++

- CoRAM penalties in today's FPGA applications:
  Reduced performance.
  Logic resource overhead.

- How CoRAM++ handles those problems?
  Reconfigurability.
  More customized datapath to memory.
  Extensible library of data structure specific data transport modules.

- Example: To stream a block of data, provision of specialized commands, initiating, monitoring data streaming between DRAM and hardware kernels. Similarly specialized for other data structures cases.

# Background and Related Work

- Altera, Xilinx similar system builder tools for system integration.
- MaxCompiler (Maxeler), decomposed applications in to kernels and manager modules, for computation and I/O, Java-like language.
- RedSharc similar to CoRAM/CoRAM++, except for global controlling by software on embedded processor.
- LEAP, scratchpad memory environment backed up by off chip-DRAM. Multiple hardware kernels, can share the same virtualized scratchpad. Disadvantage: Generic abstractions, no specific access patterns optimization.

# Supported Data Structures

- Block Copy.
  Random access by the hardware kernel to data within a block. (CoRAM)

- Read/Write Stream.
  Sequential single data element access by the hardware.

- Multi-Dimensional Array.
  Pre-difined size dimension, tiled lay out optimization, sequential access(Read/Write data access pattern), reordering data support for permutation engines.

- Linked List.
  Compile-time definition of the data structure, allows sequential access(Read/Write data access pattern).

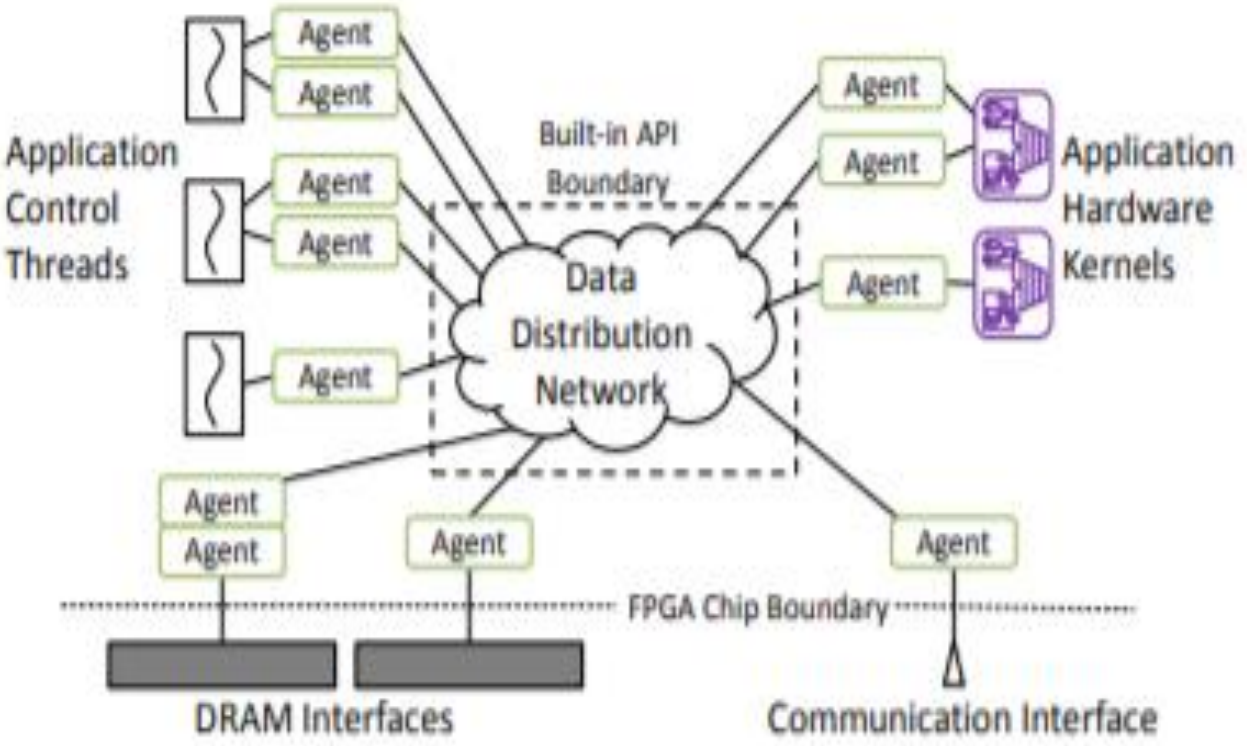
# Additional application level Interfaces

- Control thread side Scratchpad, access data from DRAM, based on data locality.
- Host Computer manages data transfer, between a host computer and FPGA's DRAM.
- Channel interface transmits messages between hardware kernels and control threads.

# Implementation of CoRAM++ Interfaces

- Agents: Hardware modules, implementing the application level interfaces.
- Connection of control threads to agents through a command interface, agent-supporting function calls, converted in to messages to hardware kernels, or transfer to another agent's control thread for sequential operations.
- Port interfaces(FIFOs, SRAMs and CAMs) for hardware kernels accordingly.
- Capability of agents to send commands to data distribution network, using fire and forget, execution model.
- Agent structure: not device specific based, instead supported by a thin hardware virtualization layer, providing the following interfaces:
  - Data Transfers.
  - Messages.
  - Direct connections for DRAM and I/O.
- Finally, combination with different programming environments such as LEAP.

# System level design of CoRAM++



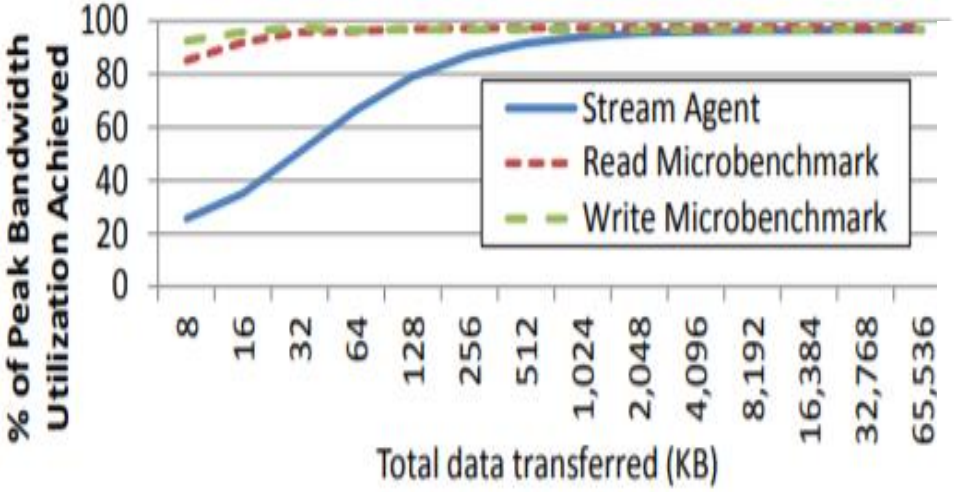
# Evaluation process

- For the evaluation process, the following boards were used.
- The main purpose of the evaluation, was investigation of bandwidth bound and latency bound scenarios(not compute bound scenarios).

| FPGA Board | Terasic DE4 | Xilinx ML605 | Xilinx ZC706 |
|---|---|---|---|
| FPGA | Altera EP4SGX530 | Xilinx LX240T | Xilinx XC7Z045 |
| Hard CPU Cores | None | None | 2×ARM A9 |
| Logic Cells | 531,200 | 241,152 | 350,000 |
| Block Memory | 3.4 MB | 1.8 MB | 2 MB |
| Hard DSP blocks | 1,024 | 768 | 900 |
| DRAM Bandwidth | 2×6.4 GB/s | 6.4 GB/s | 12.8 GB/s (Fabric only)<br>4.3 GB/s (Shared) |
| FPGA Vendor Tool | Quartus 14.0 | ISE 14.7 | ISE 14.7 |

# 1st experiment.

- Spiral generated, 64-input double precision DFT engine, 200MHz, streaming input from DRAM, streaming output to the other DRAM.




| Component | ALUT % | Register % | Block Mem % |
|---|---|---|---|
| DRAM Controllers | 11.4 | 12.3 | 15.3 |
| NoC | 6.7 | 6.2 | 0 |
| NoC Endpoints | 16.3 | 23.8 | 25.6 |
| Threads + Agents | 9.0 | 12.8 | 44.9 |
| Kernel | 56.6 | 44.9 | 14.2 |

- From the above results, streaming DFT, at 512 Kbytes has 90% peak from best read/write performance(red/green line). Maximum peak at, at least 8Mbytes of data. Limited performance at small data size sets, due to the latency of the DFT core.

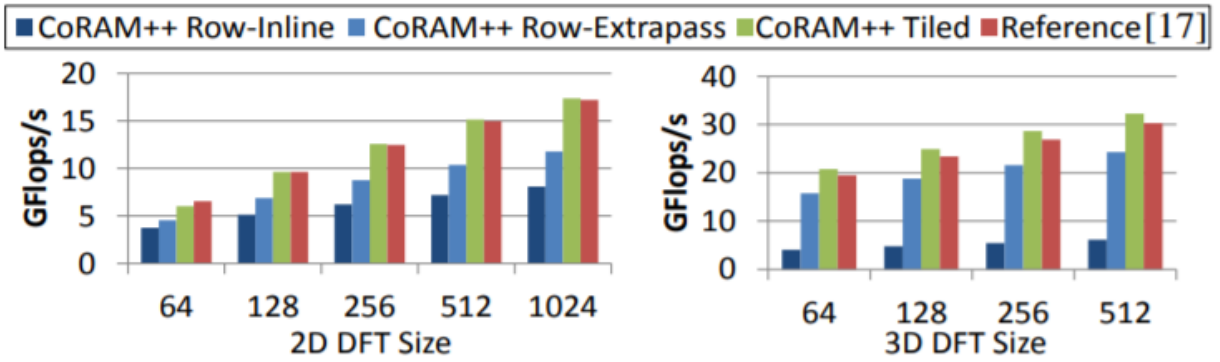
- Implementation Resources: 32% of the logic and 7% of the block memory, significant amount of resources.

# Multidimensional array accesses(2nd Experiment)

- Use permutation engines to recover rows from tiles of data and transpose tiles of data, in 2D and 3D DFT, each one decomposed in 1D DFT.

- Strided Array Accesses:
  1024x1024 double precision DFT, row major data. Ineffective, due to misses in the DRAM row buffer, caused by strided data accesses during column traversal.

- Tiled Array Accesses(3 configurations):
  CoRAM++ Row-inline, very slow, required the same strided accesses as the row major computation.
  Row-extrapass, better performance, extra pass through memory, after final phase of computation.
  Tiled, 90% matched the referenced calculations, keep data tiled throughout the computation. However, higher resource utilization than the reference DFT.



# Resource utilization for CoRAM++ and Reference DFTs

- While CoRAM++ has more overhead, provides flexibility and ease in the program environment, selection of array layout and array traversal method in DRAM, also hosts computer interface.

| DFT Type | Logic % | Block Mem % | DSP % |
|---|---|---|---|
| CoRAM++ 1024×1024 2D | 62 | 58 | 35 |
| CoRAM++ 512×512×512 3D | 52 | 51 | 20 |
| Reference [17] 1024×1024 2D | 27 | 47 | 35 |

# Linked List accesses(3rd Experiment)

- Linked List accesses(5 methods):

  **Scratchpad-1**: C like linked list, accessing DRAM using the scratchpad agent, single row data matching the DRAM interface width.

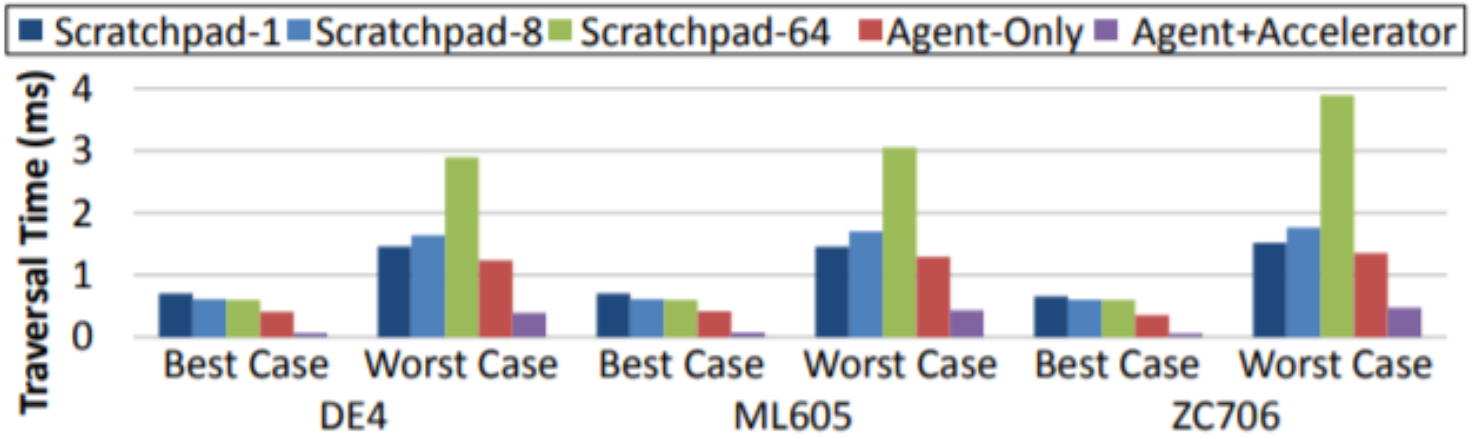
  **Scratchpad-8:** Same as scratchpad-1, 8 rows of data, same size as previous case, for locality and low cost of DRAM access.

  **Scratchpad-64**: same method with 64 rows.

  **Agent only**: Instantiates the hardware linked list engine within the linked list agent, linked list operations are executed from the linked list agent, triggered by a control thread(best performance for CoRAM).

  **Agent+Accelerator**: Usage of the linked list agent with hardware linked list engine attached to the DRAM(Accelerator).

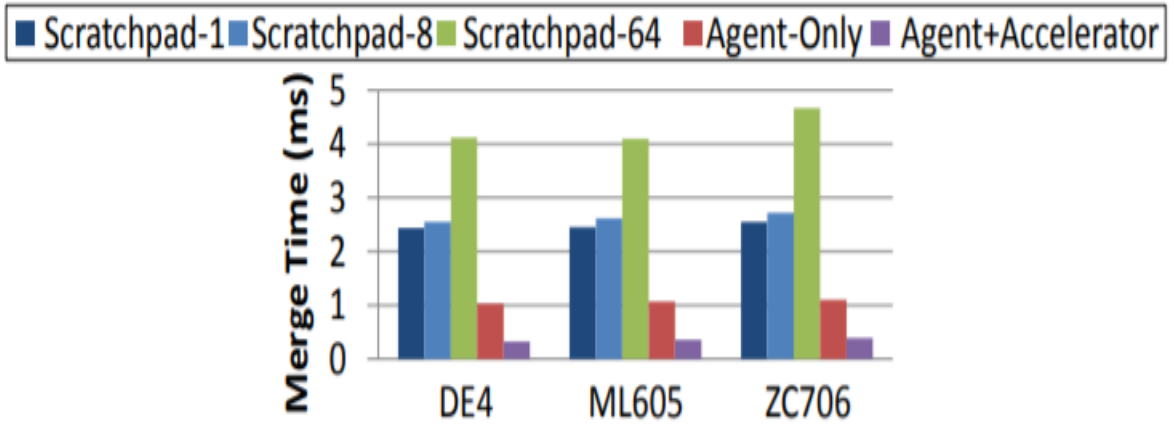
# Linked List Traversal




- Best case(packed linked list nodes and data into contiguous blocks) and worst case(linked list separated linked list nodes and data by 8 kilobytes) DRAM layouts.

- **Scratchpad**, improved performance by increasing the number of rows, however reduced performance in the worst case.
- **Agent only**, 1.7x faster than scratchpad for best case, slightly faster in the worst case.
- **Agent+Accelarator,** 9x faster than the baseline traversal on DE4, 8.5x on ML605 and 10.5x on ZC706, 5.2x faster than agent only.

# Sorted Linked List Merge




- 100 merged pairs of linked lists randomly assigned to each list pair, aligned to the DRAM interface width, 50% chance being a following node, or in a random location.

- Agent+Accelarator linked list configuration much faster than any other configuration, lower latency DRAM accesses when chasing pointers.

# Conclusions about the CoRAM++/Future Work

- Conveniently and efficiently support simple and complex data access patterns, through a library of data structure specific interfaces.
- Matches the performance of hand designs, using data structure specific modules connected to a DRAM interface achieves 5.2x better performance on data dependent operations.
- Future explorations will target in the acceleration of other data structures, such as trees and graphs.
- Investigation of hardware/software codesign of hybrid CPU-FPGA system and target FPGA programming environments, rather than devices

# END