FPGA Memory System for HPC Applications using Addressable Cache

Norihisa Fujita[1,2], Ryohei Kobayashi[1,2], Yoshiki Yamaguchi[2,1] and Taisuke Boku[1,2]
Center for Computational Sciences, University of Tsukuba[1]
Faculty of Engineering, Information and Systems, University of Tsukuba[2]

▶ Introduction

- Field Programmable Gate Array (FPGA)
  - FPGA is reconfigurable hardware and high-end FPGAs are HPC-ready
    - Fast inter-FPGA (up to 100Gbps x 4 links) communication using optical links
    - Digital Signal Processor (DSP) block for IEEE754 single-precision floating number operations
- High Level Synthesis (HLS)
  - **HLS uses languages for software (e.g. C, C++, OpenCL) for FPGA development**
  - HLS reduces programming costs on FPGA
  - **HLS allows HPC application developers to utilize FPGA technology**
- Issues for FPGA HPC
  - **memory bandwidth is a big bottleneck**
  - FPGA has only 76.8GB/s (4 channels of DDR4), whereas NVIDIA A100 has 2TB/s
  - High Bandwidth Memory 2 (HBM2) is high-bandwidth memory widely used in accelerators for HPC
  - FPGA is one of such accelerators
  - Intel Startix 10 MX FPGA has HBM2 **up to 512GB/s** of bandwidth.
  - HBM2 architecture is different from DDR4. HBM2 is aggregated architecture, as shown in Fig. 1.
  - Using all memory channels simultaneously is a big challenge for FPGAs.
- Purpose of this study
  - We introduce a new memory architecture for FPGA to utilize HBM2 memory
  - Our system uses addressable cache as a working memory for computation
  - Programmers describe data transfer between the memory and the cache explicitly
    - optimized and predictable performance
    - low resource consumption in FPGA compared to automatic cache

▶ HBM2 in Startix 10 MX FPGA

- HBM2 Spec
  - 2 stacks of HBM2 die
  - up to 16GB of capacity
  - up to 512GB/s of bandwidth
- Each HBM2 stack has
  - 8 physical memory channels
  - 16 pseudo channels
  - 256GB/s aggregated bandwidth
- Total:
  - 16 physical memory channels
  - **32 pseudo channels**
  - **512GB/s aggregated bandwidth**

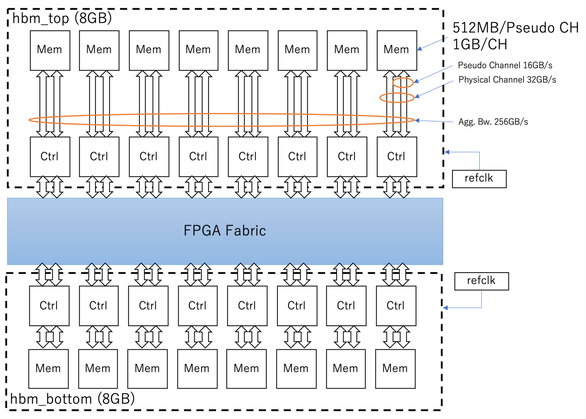

Figure 1: Internal connection between FPGA memory and FPGA fabric for Intel Stratix 10 FPGAs.

▶ Proposed Memory System

- Addressable cache
  - Scratchpad memory as data buffer on Block RAM (BRAM) inside FPGA
  - BRAM is **low-latency and high-bandwidth** memory
  - Burst data transfer between memory and cache
  - BRAM is SRAM, so random access is fast
  - Fast random access from application
  - Our cache is not automatic
    - Automatic cache consumes a lot of FPGA resource
    - We describe data transfer between memory and cache
- PCI Express (PCIe) Controller
  - data transfer between CPU and FPGA
- Crossbars
  - Maximize flexibility of memory access
- HBM Controller (HBMC) and Memory Controller (MC)
  - Packet converter (Internal packet <=> AXI)
  - Write Burst Buffer
  - Read Buffer
- Local Storage (LS)
  - 128KB scratchpad memory per LS
  - DMA controller between LS and HBM2
  - RISC-V core for kernel control

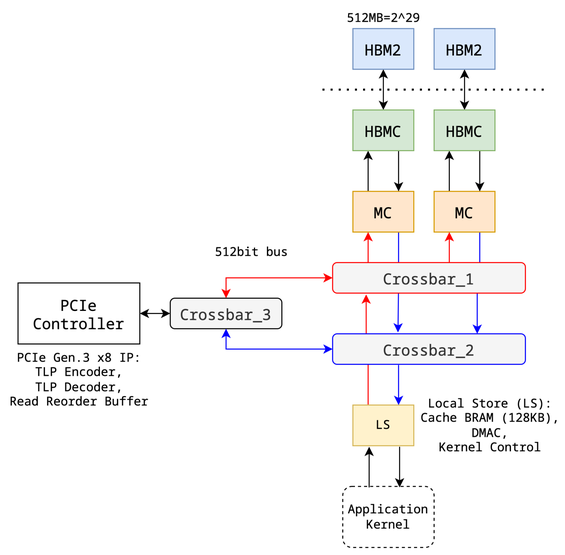

Figure 2: Overview of Proposed Memory System with two memory channels and one LS.

- Preliminary Implementation
  - This system is under development
  - We plan to implement 4 crossbars with 8 MCs and 8 LSs (total: 32 MCs and 32 LSs) in FPGA

▶ Software and API

- Same code for simulation and hardware
  - Memory Mapped IO (MMIO) wrapper to use the same code
  - MMIO is the core to control PCIe device
  - APIs are built on the wrapper MMIO API
- Stream API
  - In this system, we use small buffer for computation
    - double buffering is needed to hide memory access latency
    - we expect 10~20us for each step (depending on LS size and FPGA frequency)
    - **fine-grain FPGA control is required**
  - To address this, we introduce **Stream API** for fine-grain control
    - High-level C++ API to describe FPGA behavior
    - Sending multiple commands from host CPU at once
    - Loops on the device
  - Stream API
    - Simplified RISC-V core for each LS as a controller
    - Stream API makes **RISC-V instructions** for the controller
    - Boost.YAP[1]: to construct Expression Template (ET)
    - LibFirm[2]: to construct SSA form, optimize data flow and analyze data dependency
    - Register allocation based on Perfect Elimination Order (PEO) of dominator tree[3]
    - Up to 2 streams can be executed simultaneously

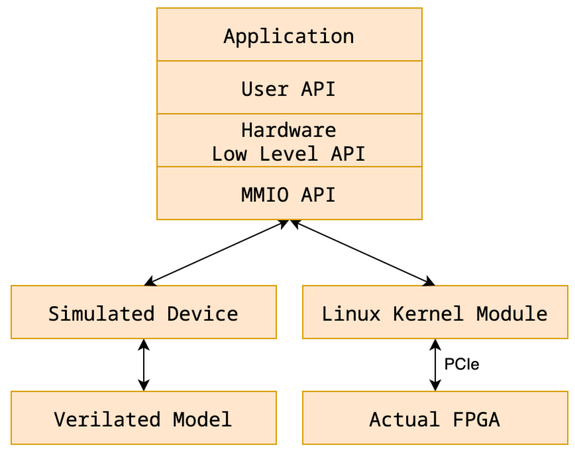

Figure 3: Software stack.

```
ctrl->begin_config();

for (int s = 0; s < 2; s++) {
  define_stream(ctrl, s) {
    var i;
    var start;
    array_view<uint32_t> buffer(
        s == 0 ? buffer1 : buffer2);
    array_view<uint32_t> data(d_data);

    i = 0;
    start =
      (c * N_STREAMS + s) * CHUNK_SIZE;

    stream_for(i = 0,
               i < N_CHUNK / N_WORKERS,
               i = i + 1) {
      buffer(0, CHUNK_SIZE) =
        data(start, start + CHUNK_SIZE);
      if (s == 0) {
        kernel(0, 0, LOOP_LEN);
      } else {
        kernel(LOOP_LEN, LOOP_LEN, LOOP_LEN);
      }
      data(start, start + CHUNK_SIZE) =
        buffer(0, CHUNK_SIZE);
      start =
       start + N_WORKERS * CHUNK_SIZE;
    }
  }
}

ctrl->stream_start_and_sync();
ctrl->end_config();
```

Figure 4: Example code of Stream API.

▶ Preliminary Evaluation

- Preliminary evaluation result on the simulator
  - two memory channels and one LS are implemented
  - Verilator is used to simulate the hardware
  - Simplified HBM2 model is used
    - Always one-cycle read and write latency
    - Reads and writes are executed simultaneously. (BW. is doubled than actual)
- Each stream runs a loop with three steps:
  - for (i = 0; i < 4; i = i + 1)
    1. DMAC reads 64KB of 32bit int array from HBM2 to LS
    2. Kernel computes bitwise-not of each element of array
    3. DMAC write back data from LS to HBM2
- Figure 5 shows that two streams can keep memory bus busy

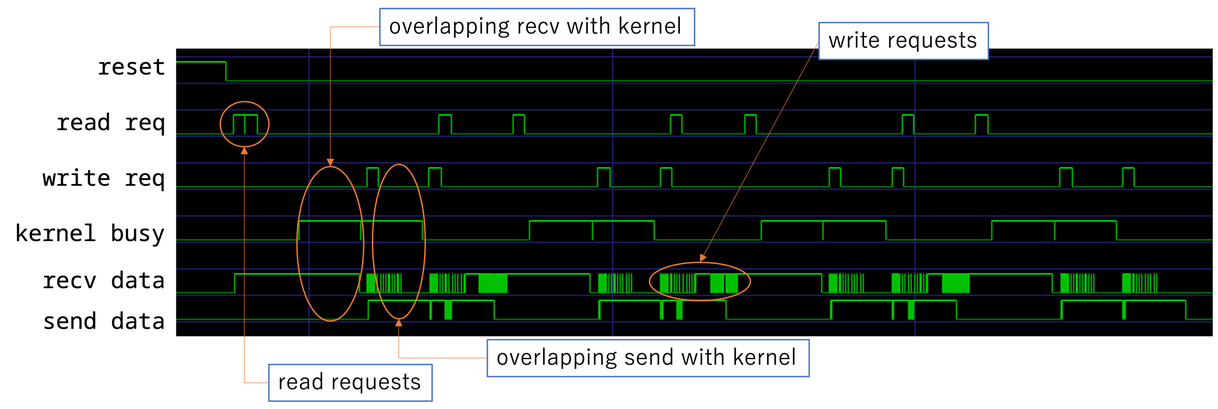

Figure 5: Waveform result of the simulation.

▶ Conclusion and Future Work

- We proposed new memory system for HBM2 and HPC
- We implemented Stream API for fine-grain FPGA control
- Preliminary results showed the API worked as intended
  - Two streams to overlap data transfer with computation
- Future Work
  - Performance evaluation on actual FPGA hardware
  - Implementing all HBM2 channels (32)
  - Implementing HPC application on this system

[1] Boost.YAP, https://www.boost.org/doc/libs/release/doc/html/yap.html [2] LibFirm, https://pp.ipd.kit.edu/firm/
[3] Sebastian Hack and Gerhard Goos, "Optimal register allocation for SSA-form programs in polynomial time", Information Processing Letters 98 (2006), pp. 150--155

Acknowledgements: This work was supported in part by the "Next Generation High-Performance Computing Infrastructures and Applications R&D Program" (Development of Computing Communication Unified Supercomputer in Next Generation) of MEXT. This research was also supported in part by the Multidisciplinary Cooperative Research Program in CCS, University of Tsukuba, and JSPS KAKENHI, Grant Number 21H04869. We also thank the Intel University Program for providing the hardware and software.