RIKEN CGRA: Reconfigurable Data-Driven Architecture for Future HPC

Boma Adhi[1], Carlos Cortes[1], Yiyu Tan[1], Takuya Kojima[2], Artur Podobas[3], Kentaro Sano[1]

[1]RIKEN Center for Computational Science, Kobe, Japan; [2]Keio University, Tokyo, Japan; [3]KTH Royal Institute of Technology, Stockholm, Sweden

email: {boma.adhi, kentaro.sano}@riken.jp

What is a Coarse-Grained Reconfigurable Array?

- Array of tiles: Processing Elements (PE), Load/Store, Buffer, etc.
- Connected by programmable interconnect
- A loop kernel is compiled into a Data-Flow Graph then mapped onto CGRA (Place & Route)

Background

- **Limited scaling of current generation of von Neumann based CPU**
  - ✓ **Inefficient hardware:** Too much non compute related hardware (branc pred., OoO, etc.) relative to ALU
  - ✓ **Indirect data transfer between cores:** LLC and shared memory become bottleneck
  - ✓ **Non-DRAM friendly random memory access:** cache miss is costly
- Our solution: **Reconfigurable Data-driven Computing on CGRA**
  - ✓ **Efficient hardware:** can be reconfigured for each application
  - ✓ **Direct data transfer between ALU**
  - ✓ **DRAM friendly access pattern:** data is streamed and can be pipelined to hide latency
  - ✓ **CGRA**: Trade FPGA's flexibility for more efficient operation and faster compilation & configuration
- **Recently CGRA is considered for HPC-scale Deep-Learning Accelerator**
  - ✓ However CGRA traditionally only targets smaller and lower power consumption embedded device.
  - ✓ What does it take for CGRA to compete in HPC market?
  - ✓ Finding the best practice of designing CGRA for HPC as an extension of multicore CPU

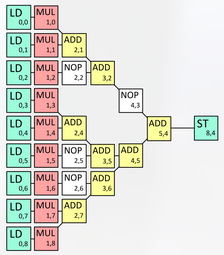
Goals

- Exploration of CGRA design space on HPC environment
- Implementing a working prototype on our FPGA cluster connected to an HPC system

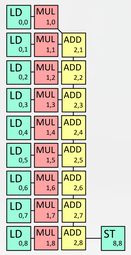
CGRA Compiler

- Front-end: Compile OpenMP annotated C code to Data-Flow Graph (DFG)
  - ✓ LLVM Based
- Back-end: Place and Route the DFG onto CGRA with specified constraints and criteria
  - ✓ Genetic-Algorithm based optimization
- Example:
  - ✓ 3 x 3 Convolution code compiled with 9 inputs and 1 output

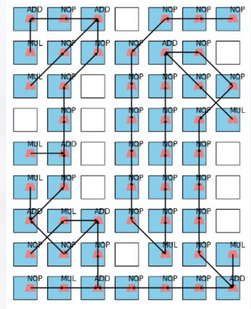
Generated by Hand **Tree**



Generated by Hand **Liner**

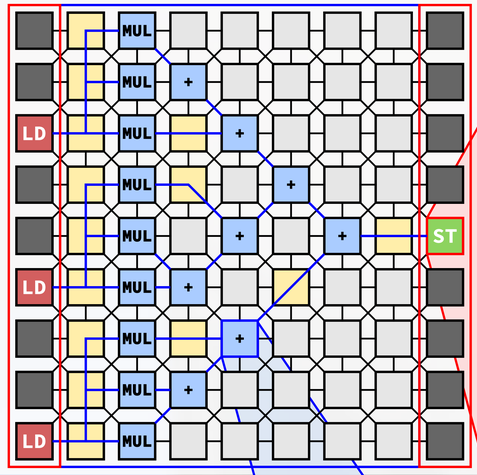


Compiler Generated **Unbalanced Tree Max**



RIKEN CGRA

- Written in SystemVerilog
- Targets Intel FPGA Platform
  - ✓ Our FPGA cluster uses Intel FPGA
- All internal connection is defined with AvalonST & AvalonMM; allowing easier integration with:
  - ✓ Intel Platform Designer
  - ✓ 3rd party IP for tiles
  - ✓ Hardened FPGA feature (network PHY, memory controller, CPU core, PCIe, etc)
- Target explorations:
  - ✓ Bypass/Torus/3D Wiring
  - ✓ Heterogenous vs Homogenous tiles
  - ✓ Composition
  - ✓ Type of tiles

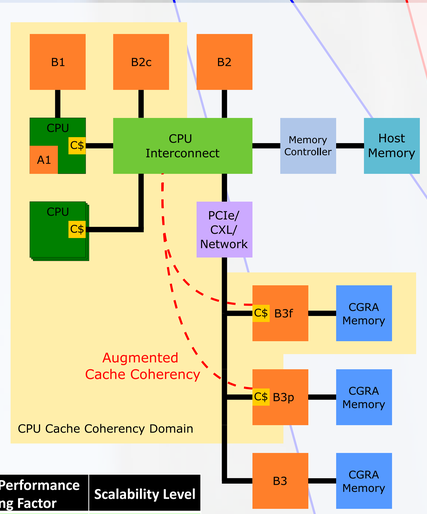


CPU - CGRA Interconnect Options

- **Internal state:**
  - ✓ (A) Stateless (100% controlled by CPU)
  - ✓ (B) Has its own state (CPU may still control the operation)
- **Memory Access:**
  - ✓ (1) Depending on CPU's LSU unit
  - ✓ Has its own LSU unit
    - ➢ (2) Within CPU's coherency domain
    - ➢ Outside CPU's coherency domain
      - ☐ (3) Direct access to main memory
      - ☐ (4) In-direct access through PCIe\Nvlink/Network/Driver (Usually has its own memory too)

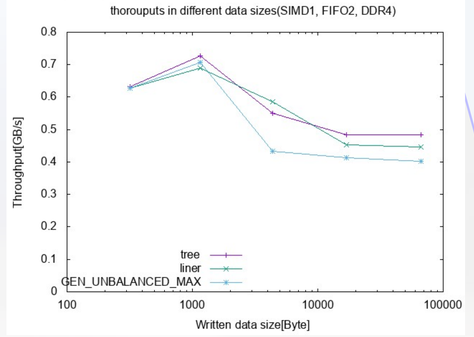
      With augmented cache coherency:
      - ✧ (4p) Partial/one way coherency
      - ✧ (4f) Full coherency through address translation

Additional consideration: virtual memory space/MMU

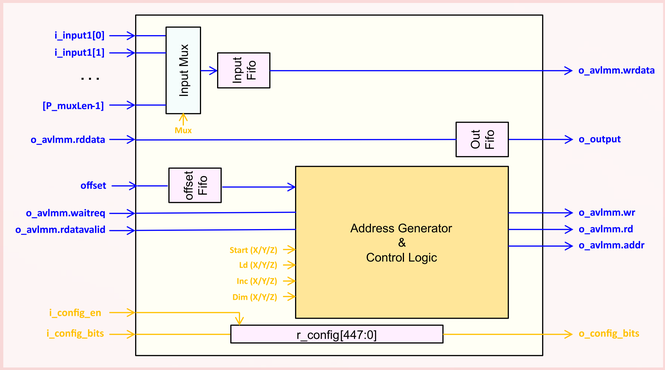


Class	Key Characteristic	Programmability	Resources	Memory Performance Limiting Factor	Scalability Level
A1	Part of CPU instruction pipeline	Easiest	Very Limited	CPU Core LSU	Chip Level
B1	Stateful, loosely coupled, shares CPU LSU	Easier	Limited	CPU Core LSU	Chip Level
B2	Outside CPU Core complex, has own LSU, direct memory access	Complex	Larger	Host Memory BW	Chip – Box Level
B3	Completely separated, Indirect memory access, can be added after design phase	More Complex	Much Larger	Accelerator Memory BW	Chip – Rack Level

*Preliminary result based on an older design with GHDL Simulator & DRAM SIM 3

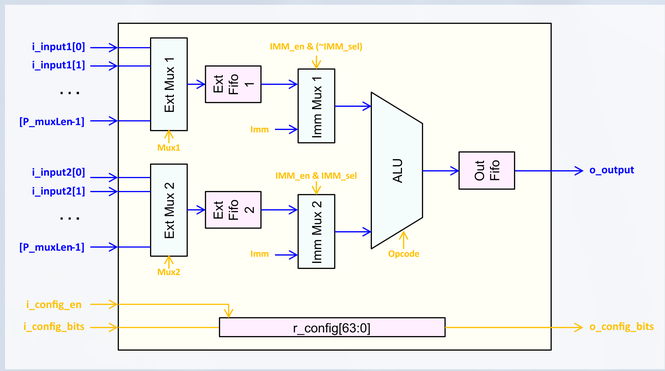


Load Store Tile



- Load/Store Tiles read and write data to and from memory then stream it to its neighboring tiles
  - ✓ Similar to DMA controller
  - ✓ Use AvalonMM for memory interface
  - ✓ Use AvalonST to talk to other tiles
- Address of the memory transaction is calculated based on the configuration bitstream sent from the host CPU
- Target Explorations:
  - ✓ DRAM Friendly Memory Access
  - ✓ Memory Coalescing

Processing Element Tile



- Processing Element Tiles stream data from selected two neighboring tiles, calculate, then send the result to neighboring tiles
  - ✓ Use AvalonST interface to communicate with other tiles
- Operations are selected by the configuration bitstream sent by host CPU
  - ✓ NOP, ADD, SUB, MUL, DIV, AND, XOR, OR, NOT, FADD, FSUB, FMUL
  - ✓ Pipelined
- Target Explorations:
  - ✓ Heterogenous Tiles
  - ✓ Buffer Memory
  - ✓ Special Function Tiles

This work is partially supported by the New Energy and Industrial Technology Development Organization (NEDO) Project JPNP16007 and Japan Society for the Promotion of Science (JSPS) KAKENHI Grant JP20H00593 and JP21H04869.