Centrifuge: Evaluating full-system HLS-generated heterogeneous-accelerator SoCs using FPGA-Acceleration

Qijing Huang, Christopher Yarp, Sagar Karandikar, Nathan Pemberton, Benjamin Brock, Liang Ma†, Guohao Dai‡, Robert Quitt, Krste Asanović, John Wawrzynek

University of California, Berkeley
†Politecnico di Torino
‡Tsinghua University

Berkeley
UNIVERSITY OF CALIFORNIA

Outline

1. Motivation
2. Centrifuge Design Flow
3. Centrifuge Design Space
4. Case Studies

Motivation

What is a good methodology for designing SoCs with many accelerators?

High-Performance System
Low NRE costs
Short Time-to-market

Accelerator Design Flow

1. Workload Characterization
2. Accelerator Modeling
    - Analytical
    - Transaction-based
3. RTL Development
4. Emulation and Verification
5. Chip Tapeout
6. Software Development

Accelerator Design Flow

1. Workload Characterization
2. Accelerator Modeling
   - Analytical
   - Transaction-based
3. RTL Development
4. Emulation and Verification
5. Chip Tapeout
6. Software Development

- Profiling on existing systems

Hotspot detection is only valid for the current system

- ISA-independent IR trace analysis

Lack of detailed information for hotspot detection

Accelerator Design Flow

1. Workload Characterization
2. Accelerator Modeling
   - Analytical
   - Transaction-based
3. RTL Development
4. Emulation and Verification
5. Chip Tapeout
6. Software Development

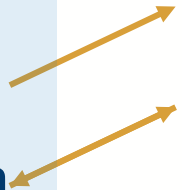
- Roofline Models
- Dynamic Data Dependence Graphs (DDDG) Analysis

Inaccurate

Lack of insights for architectural variations in real implementation

Accelerator Design Flow

1. Workload Characterization
2. Accelerator Modeling
   - Analytical
   - Transaction-based
3. RTL Development
4. Emulation and Verification
5. Chip Tapeout
6. Software Development



A. Build Software Models for Hardware Components
B. Gather Costs from RTL Synthesis
C. Verify Functionality

Engineering Intensive

Accelerator Design Flow

1. Workload Characterization
2. Accelerator Modeling
    - Analytical
    - Transaction-based
3. RTL Development
4. Emulation and Verification
5. Chip Tapeout
6. Software Development

Missing co-optimization opportunities

Current Design Flow Complexities

1. Workload Characterization
2. Hardware Modeling and Verification
3. Software Integration

Our Objectives

1. Tradeoffs informed with full-system evaluation
2. Fast development and verification cycle for both HW/SW
3. Large design space for rapid algorithm-hardware exploration

Accelerator Design Flow

1. Workload Characterization
2. Accelerator Modeling
   - Analytical
   - Transaction-based
3. RTL Development
4. Emulation and Verification
5. Chip Tapeout
6. Software Development

Proposal 1: Native Simulation

Run full-stack software with target SoC simulation

Accelerator Design Flow

1. Workload Characterization
2. Accelerator Modeling
   - Analytical
   - Transaction-based
3. RTL Development
4. Emulation and Verification
5. Chip Tapeout
6. Software Development

Proposal 1: Native Simulation

Run full-stack software with target SoC simulation

Proposal 2: Rapid Prototyping

Combine 2 and 3 with one high-level abstraction

Accelerator Design Flow

1. Workload Characterization
2. Accelerator Modeling
   - Analytical
   - Transaction-based
3. RTL Development
4. Emulation and Verification
5. Chip Tapeout
6. Software Development

Proposal 1: Native Simulation

Run full-stack software with target SoC simulation

Proposal 2: Rapid Prototyping

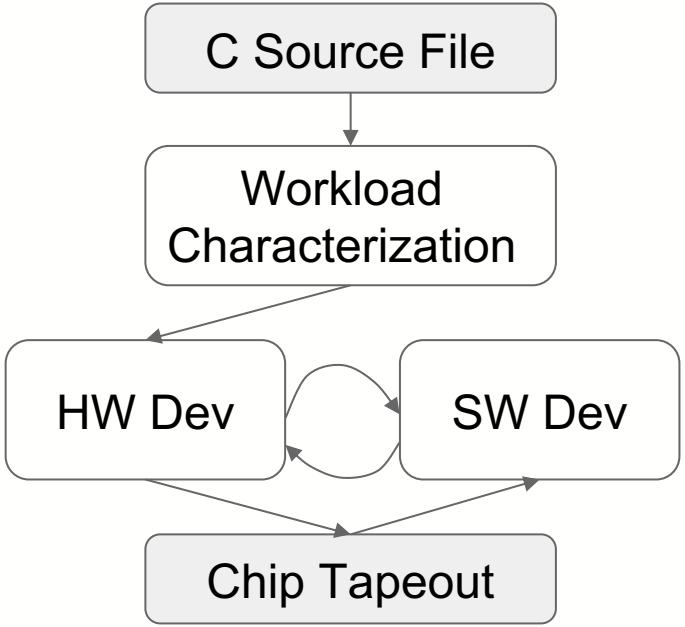
Combine 2 and 3 with one high-level abstraction

Proposal 3: Agile Development

Improve software stack concurrently with hardware development

Centrifuge Accelerator Design Flow

1. Hotspot Detection on Native System Simulation
2. Fast Accelerator SoC Generation with HLS
3. Auto Generation of SW Communication Primitives
4. Continue HW and SW Development
5. Chip Tapeout



Background: FireSim

FireSim is an open-source, FPGA-accelerated, cycle-exact, scalable hardware simulator:

- Ingests:
  - RTL Design (e.g. Rocket Chip, BOOM, NVDLA)
  - HW and/or SW IO Models (e.g. UART, Ethernet, DRAM)
  - Workload Descriptions
- Produces:
  - FPGA-accelerated Simulation (Not FPGA prototype)
  - Deployment on Cloud FPGAs

Background: High-level Synthesis

High-level Synthesis (HLS) raises the level abstraction for hardware design and verification:

- Ingests:
  - High-level Algorithmic Description in C/C++/SystemC
- Produces:
  - Hardware RTL, C and RTL Verification Models
- Commercial Tools:

Centrifuge Design Flow

Centrifuge is a unified accelerator design flow that generates the interfaces and design automation scripts that leverages existing tools:

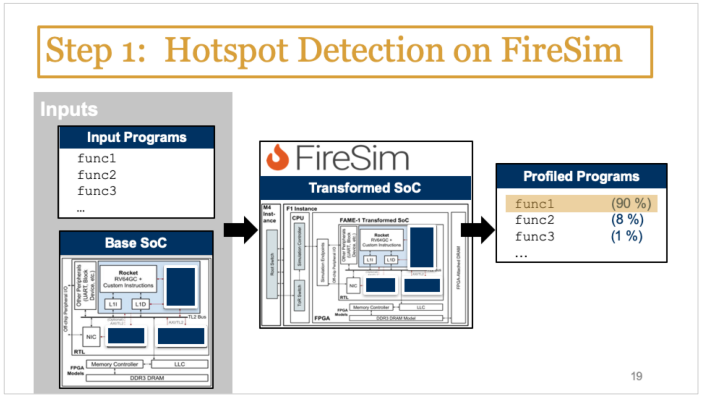
- Injects:
  - High-level Algorithmic Description in C/C++
  - Accelerator Configuration
- Produces:
  - Full Functional Accelerator SoC
  - Corresponding Software Stack
  - Tooling Scripts

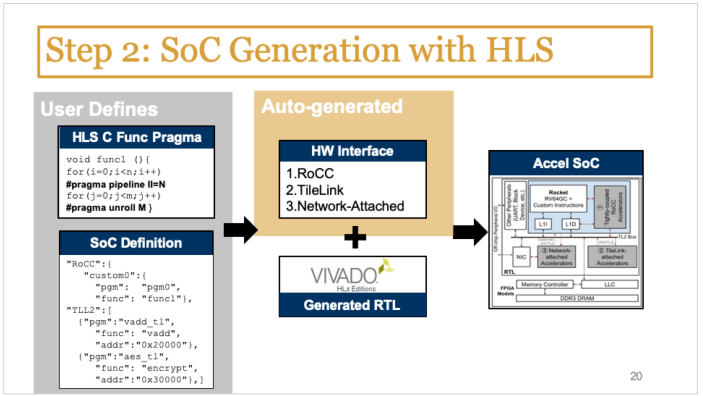
Centrifuge Design Flow

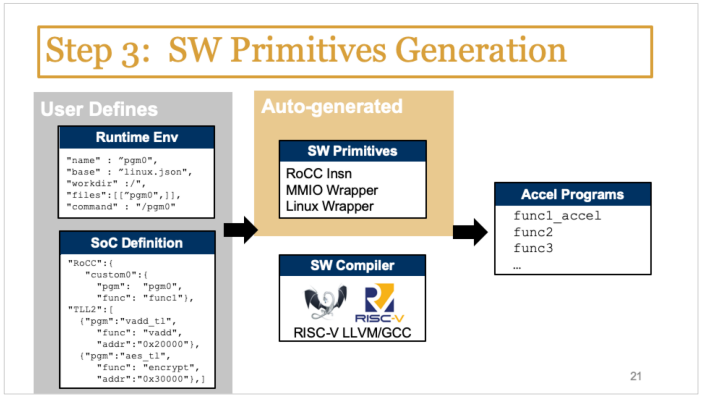
Centrifuge provides the user with:

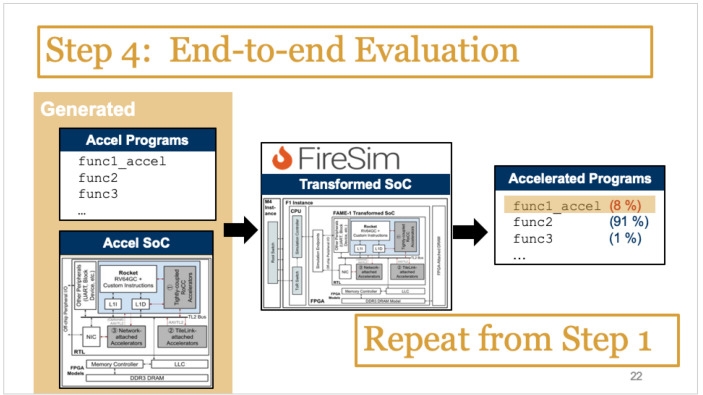
1. Full-system evaluation of the target workload
2. Fast development and verification cycle for both HW/SW
3. Large design space for rapid algorithm-hardware exploration
    A. *Hardware Integration*
    B. *Architectural Design Variation*
    C. *Software Integration*

Centrifuge Tool Flow









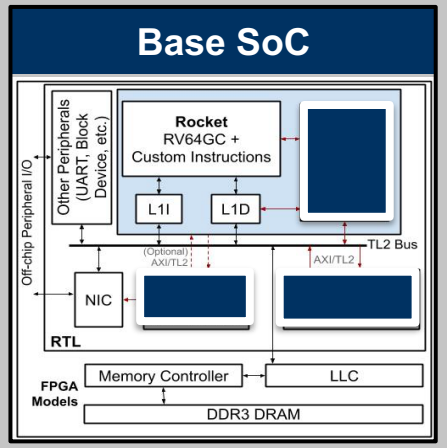
Step 1: Hotspot Detection on FireSim

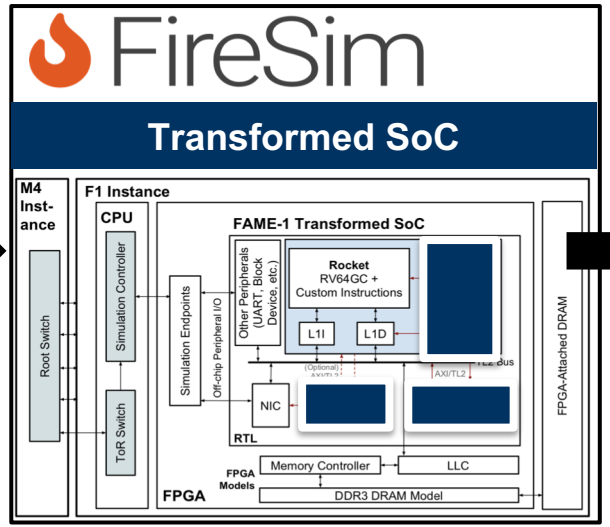
Inputs

Input Programs

```
func1
func2
func3
...
```

Base SoC





Profiled Programs

func1	(90 %)
func2	(8 %)
func3	(1 %)
...	

Step 2: SoC Generation with HLS

User Defines

HLS C Func Pragma

```
void func1 (){
for(i=0;i<n;i++)
#pragma pipeline II=N
for(j=0;j<m;j++)
#pragma unroll M }
```

SoC Definition

```
"RoCC":{
   "custom0":{
      "pgm":  "pgm0",
      "func": "func1"},
"TLL2":[
  {"pgm":"vadd_tl",
      "func": "vadd",
      "addr":"0x20000"},
  {"pgm":"aes_tl",
      "func": "encrypt",
      "addr":"0x30000"},]
```

Auto-generated

HW Interface

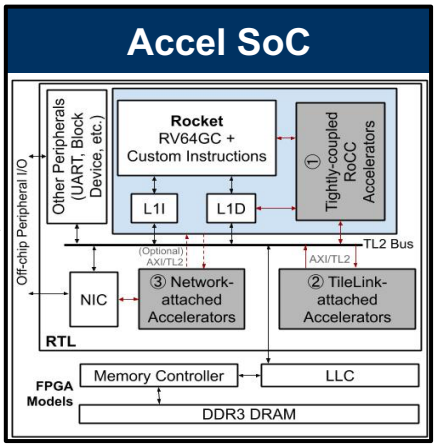
1.RoCC
2.TileLink
3.Network-Attached

+

VIVADO HLx Editions

Generated RTL

Accel SoC



Step 3: SW Primitives Generation

User Defines

Runtime Env

```
"name" : ”pgm0",
"base" : ”linux.json",
"workdir" :/",
"files":[[”pgm0",]],
"command" : "/pgm0"
```

SoC Definition

```
"RoCC":{
   "custom0":{
     "pgm":  "pgm0",
     "func": "func1"},
"TLL2":[
  {"pgm":"vadd_tl",
     "func": "vadd",
     "addr":"0x20000"},
  {"pgm":"aes_tl",
     "func": "encrypt",
     "addr":"0x30000"},]
```

Auto-generated

SW Primitives

RoCC Insn
MMIO Wrapper
Linux Wrapper

SW Compiler

RISC-V LLVM/GCC

Accel Programs

```
func1_accel
func2
func3
…
```

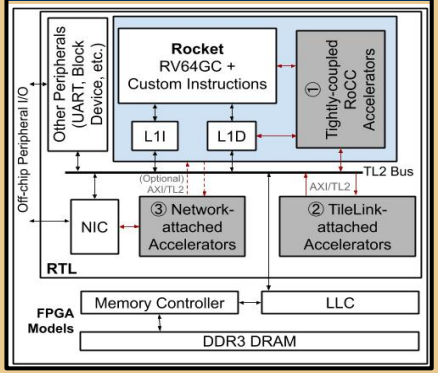
Step 4: End-to-end Evaluation

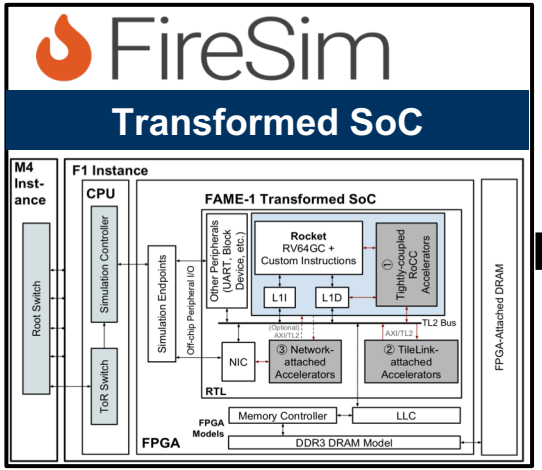
Generated

Accel Programs

```
func1_accel
func2
func3
...
```

Accel SoC





Accelerated Programs

```
func1_accel  (8 %)
func2        (91 %)
func3        (1 %)
...
```

Repeat from Step 1

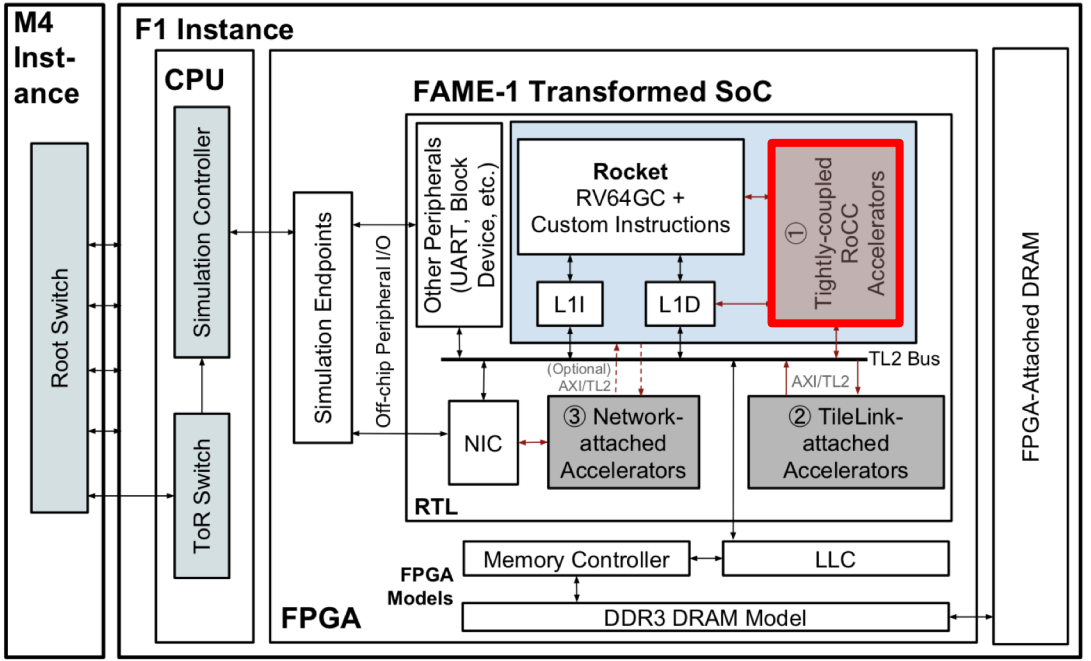
Exposed Design Space

1. Hardware Integration
2. Accelerator Design Variation
3. Software Integration

RoCC Coprocessors	TileLink Accelerators	Network-Attached Accelerators

RoCC Coprocessors

- Invoked by RoCC instruction
- Sharing L1 and LLC with the CPU
- Sharing TLB with the CPU

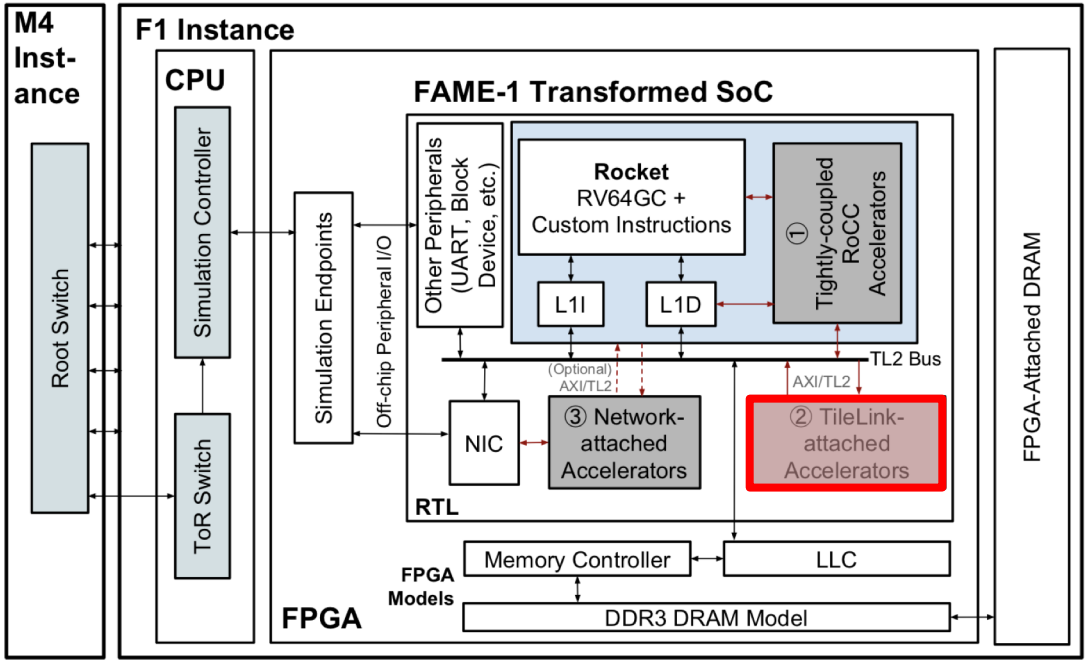


Exposed Design Space

1. Hardware Integration
2. Accelerator Design Variation
3. Software Integration

TileLink Accelerators

- Invoked by RoCC instruction or MMIO
- Sharing LLC with the CPU
- Physically-addressed

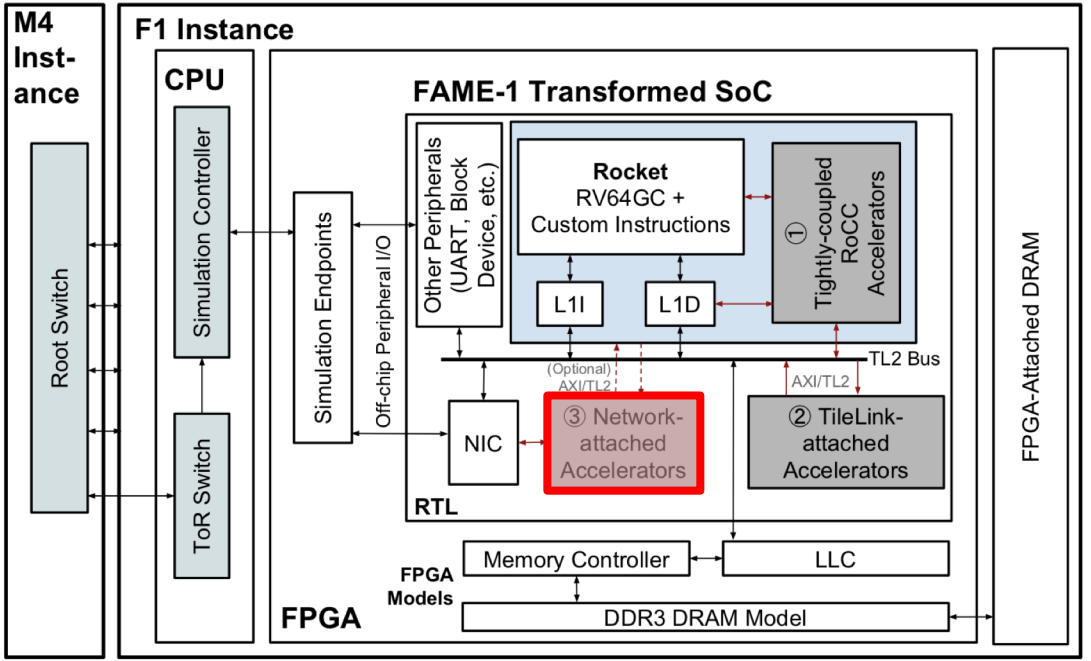


Exposed Design Space

1. Hardware Integration
2. Accelerator Design Variation
3. Software Integration

Network-attached Accelerators

- Same as TileLink Accelerators
- With Direct Access to the Ethernet



Network-Attached Accelerator

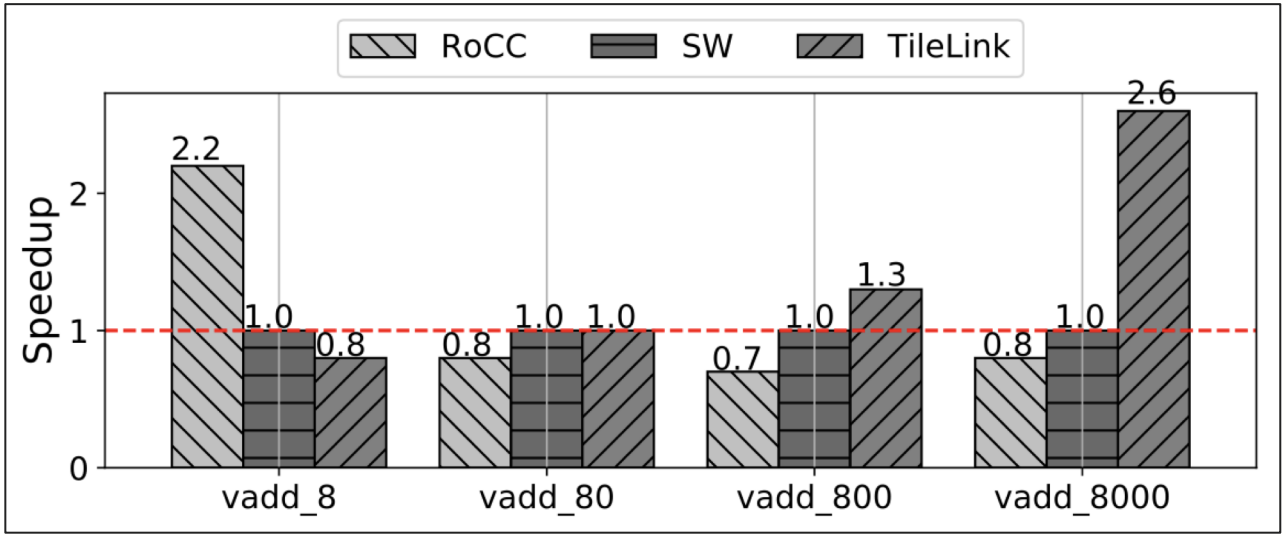
1. User includes the Centrifuge Ethernet streaming interfaces and packet parser functions in the HLS code

Centrifuge Ethernet Interfaces:

```
hls::stream<ap_uint<128> >& resp_head,
hls::stream<ap_uint<65> >& resp_data,
hls::stream<ap_uint<128> >& req_head,
hls::stream<ap_uint<65> >& req_data,
ap_uint<64>srcmac, ap_uint<64>dstmac
```

2. Centrifuge generates the interconnect between the accelerator and the Ethernet

Hardware Integration DSE



Different Coupling for *vadd* Accelerator

Three factors:

1. Cache Size
2. Cache Latency
3. Interface Bandwidth

Determine the best hardware integration strategy

Exposed Design Space

1. Hardware Integration
2. Accelerator Design Variation
3. Software Integration

Exposed Design Space

1. Hardware Integration
2. Accelerator Design Variation
3. Software Integration

A. Loop Unrolling Factors
B. Loop Pipelining Factors
C. Memory Size and Parallel Ports
D. Resource Binding

Exposed Design Space

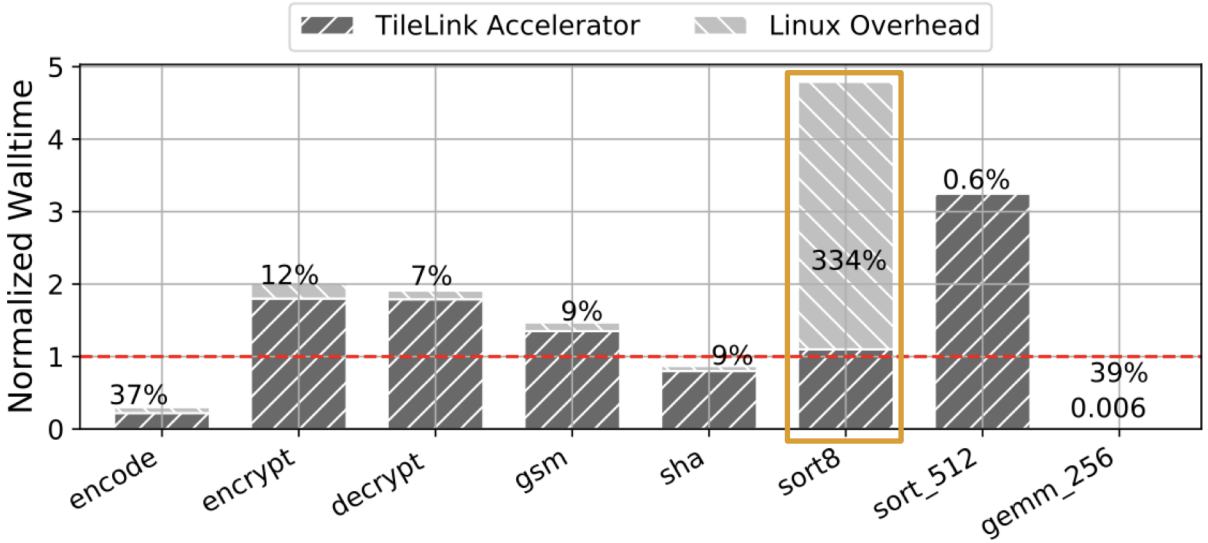
1. Hardware Integration
2. Accelerator Design Variation
3. Software Integration

A. Bare-metal
B. Linux

Linux Wrapper Generation

- Allocates contiguous physical addresses
- Virtual to physical address lookup
- Map MMIO registers to the user space

Software Integration DSE



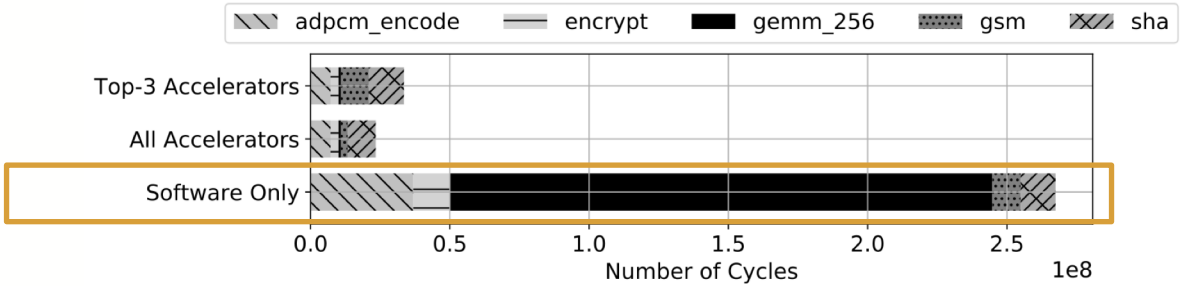
Tilelink Accelerators with Linux Driver

- In most cases, the Linux overhead is not significant
- RoCC-level integration might be better option for *sort8*

Determine whether the functions can be accelerated under Linux

Case Study 1: Smart-House Hub

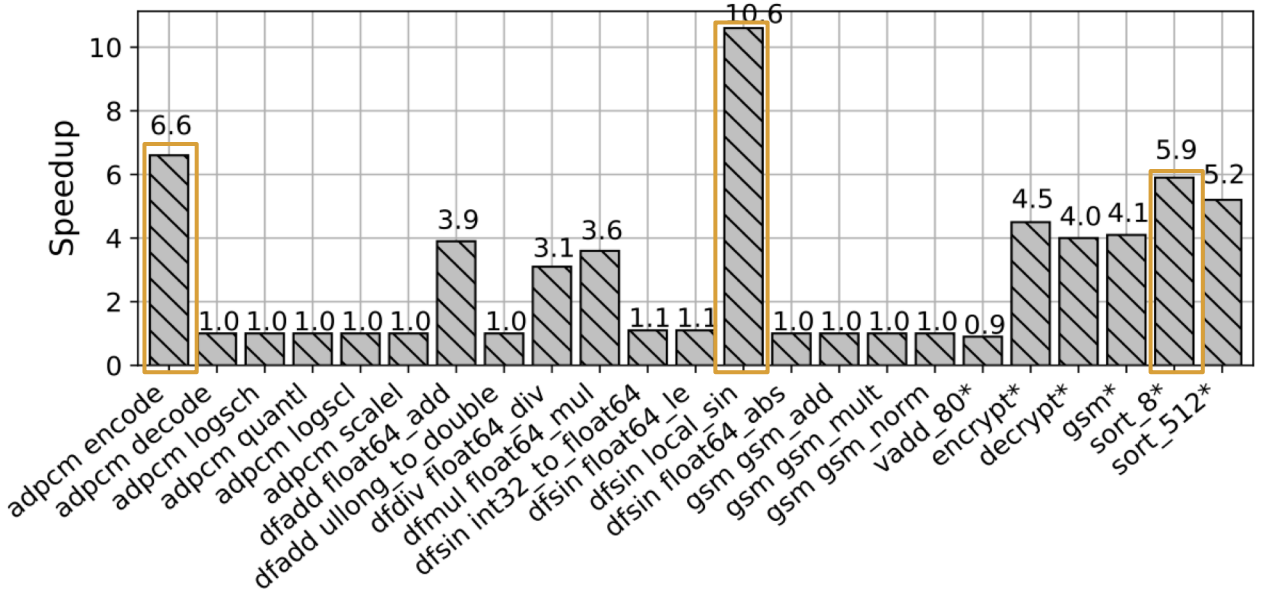
- Application: A smart-house assistant



Breakdown of key computational kernels

1. Apply Amdahl's law in hotspot detection

Acceleration Region DSE

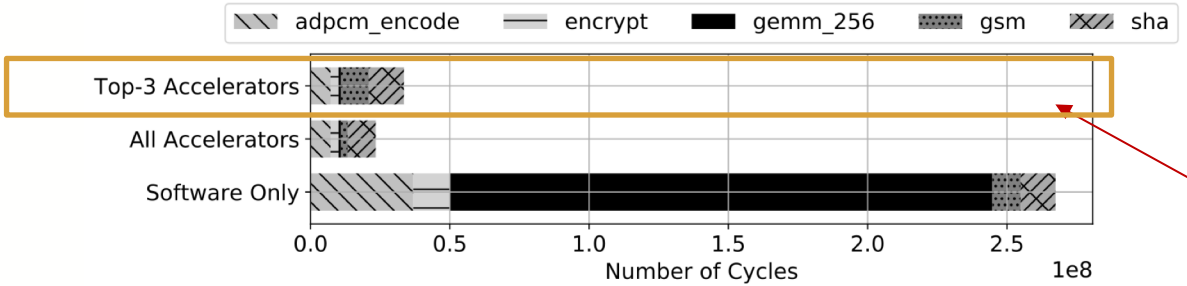


RoCC Accelerators Speedup Compared to Software

2. Determine whether the functions can be accelerated with HW

Case Study 1: Smart-House Hub

- Application: A smart-house assistant



8x Speedup

Breakdown of key computational kernels

Other Case Studies

Case Study 2: Distributed GEMM Accelerator

1. Running Full-stack Linux with MPI
2. Distributed on 16 nodes

Case Study 3: Deep Learning Accelerators

1. Different Hardware Utilization
2. Network-attached Accelerators

Conclusion

We present a methodology and flow, *Centrifuge*, that can rapidly generate and evaluate heterogeneous SoCs by combining an HLS toolchain with the open-source FireSim FPGA-accelerated simulation platform

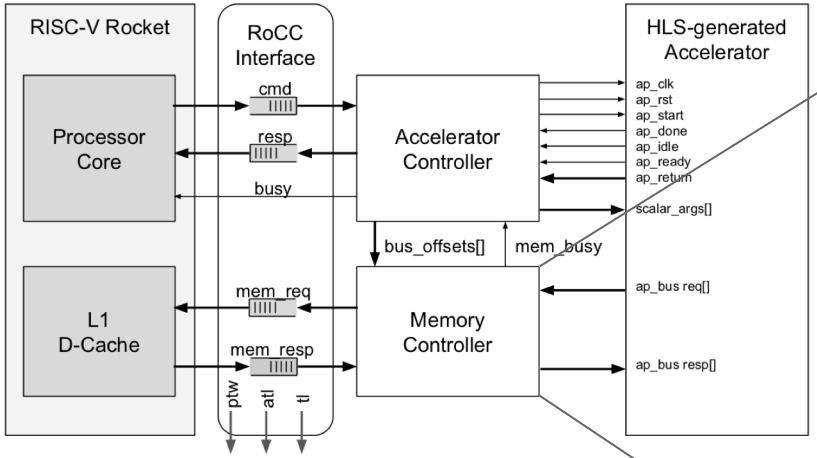
Access to code: https://github.com/hqjenny/centrifuge
Email: qijing.huang@berkeley.edu

FireSim: https://fires.im

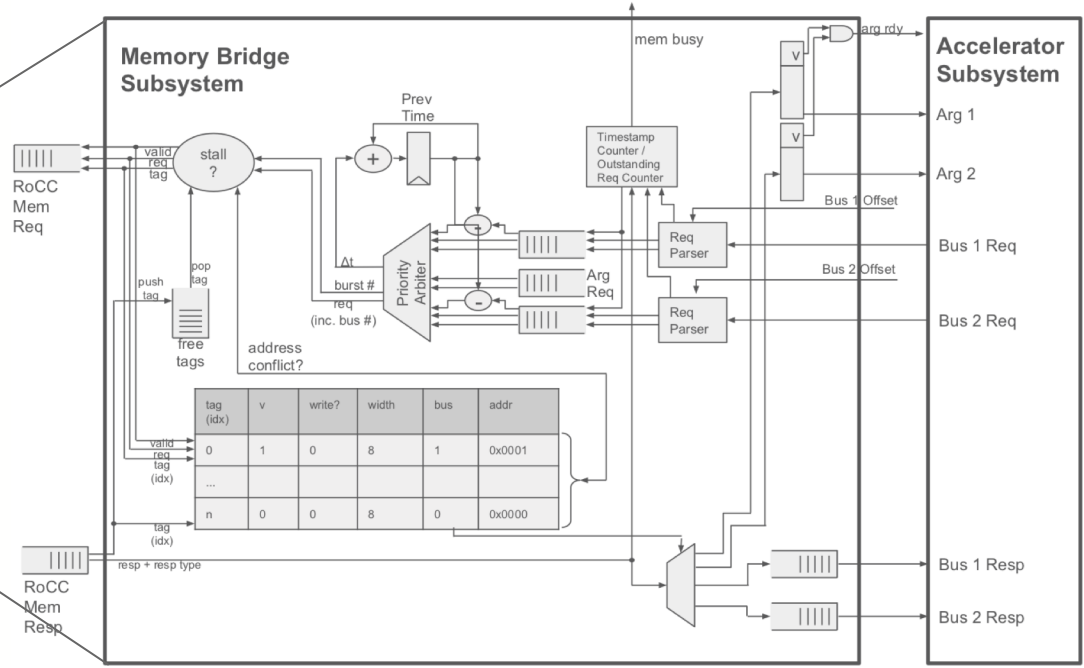
@firesimproject

Backup Slides

RoCC Coprocessor Integration

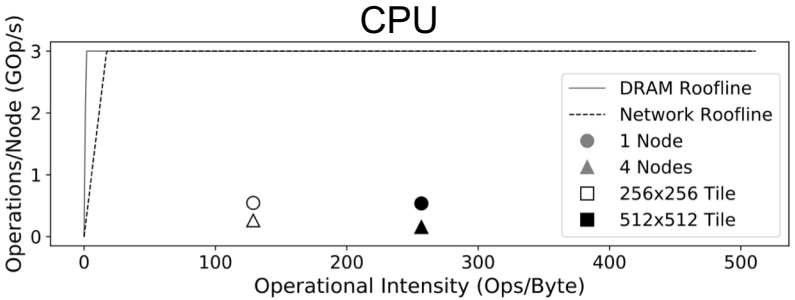


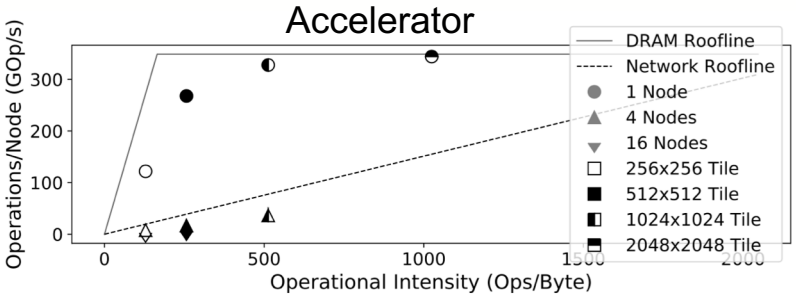
RoCC Interface



ap_bus to RoCC Decoupled Interface

Case Study 2: Distributed GEMM

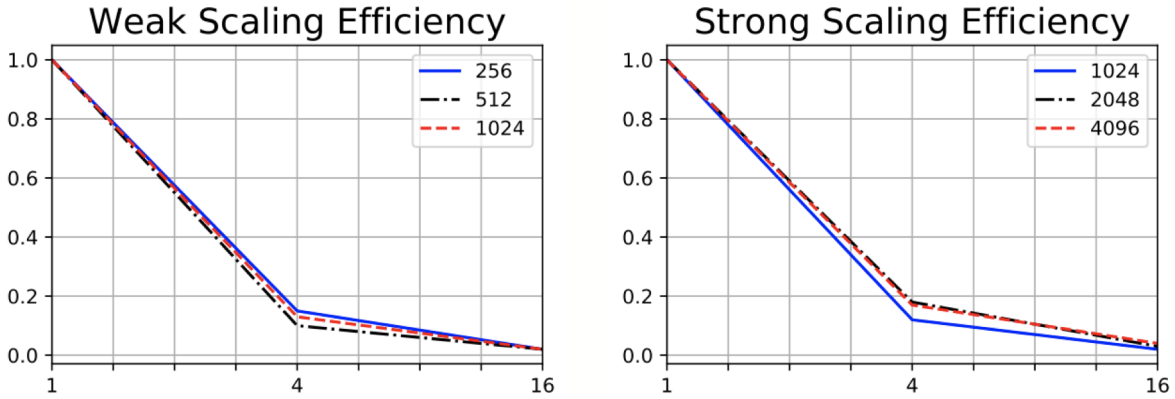




Roofline Models

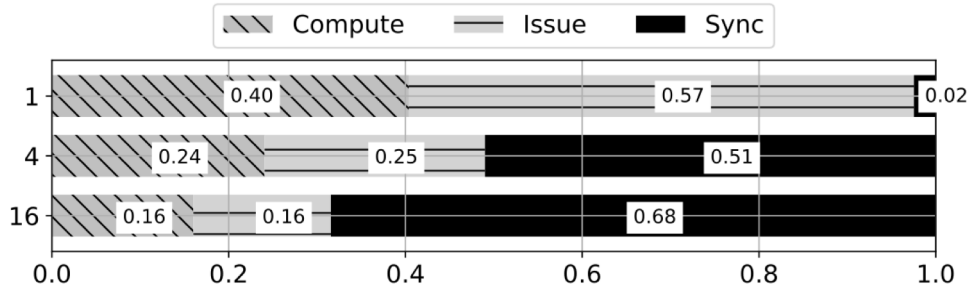
- Application: MPI-based Distributed Matrix Multiplication (DGEMM)
- Tools: STREAM and iperf are used to generate the rooflines
- Observation: With the GEMM accelerators, the distributed workload becomes more bandwidth-bound

Case Study 2: Distributed GEMM



Scaling Efficiency for DGEMM with Accelerators

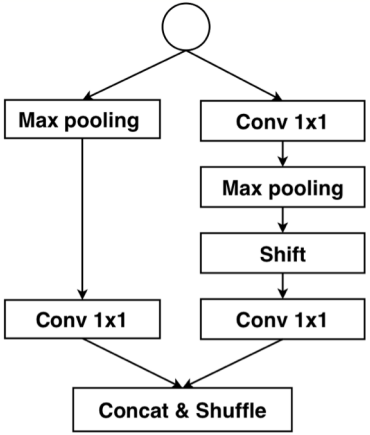
- Experiments on 1, 4, 16 nodes
- 2.0 GB/s measured DRAM bandwidth
- 1.2 Gbit/s measured network bandwidth



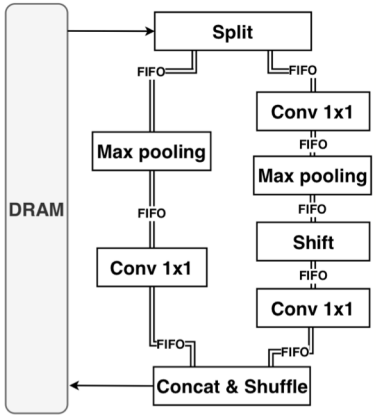
DGEMM Runtime Breakdown for 1024×1024 Tiles

Case Study 3: Deep Learning

1. HW Utilization for Different Layer Sizes



DNN Building Block



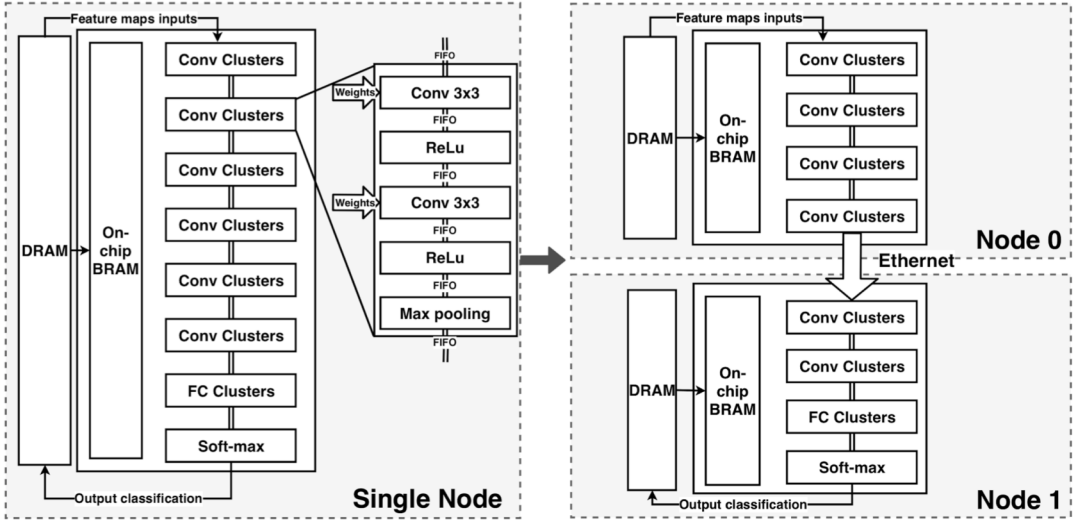
Spatial HW Design

Workload Size	Total Ops	Ops/cycle
32×16	196608	4.55
32×32	786432	15.12
32×64	3145728	20.59
16×128	3145728	21.35
8×64	196608	17.09

Ops/Cycle for Different Workload Sizes

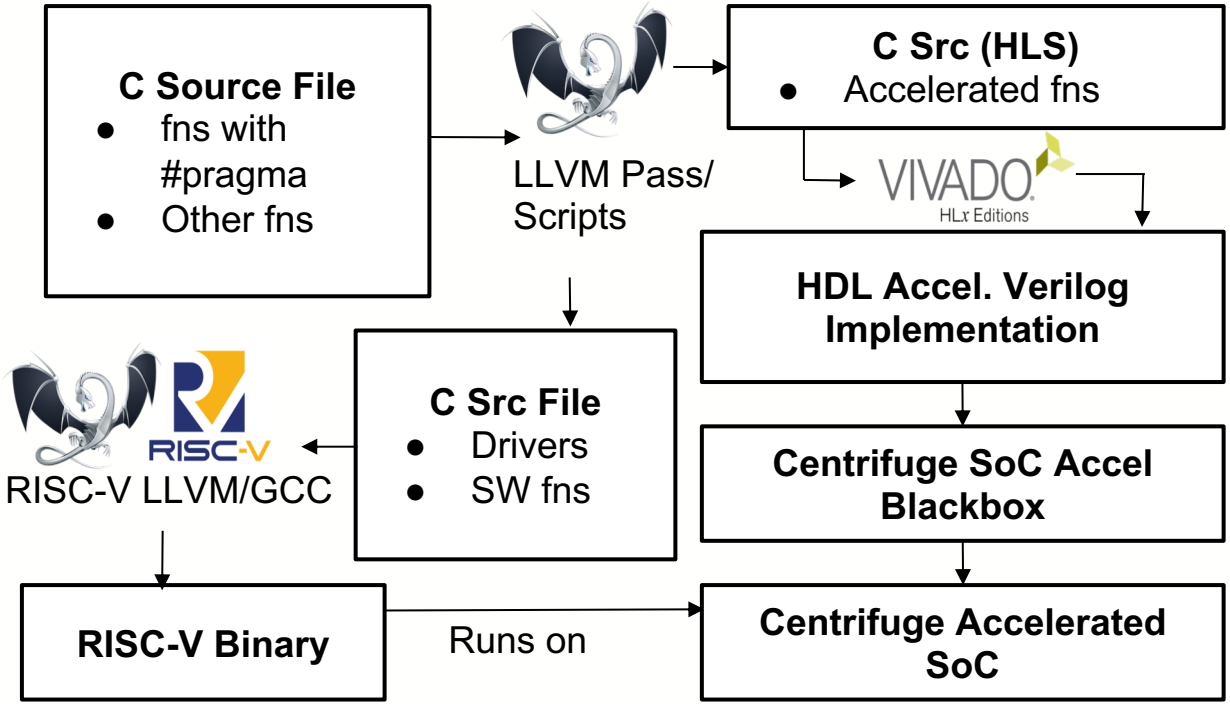
Case Study 3: Deep Learning

2. Distributed Accelerators



Multi-node accelerators, connected via Ethernet

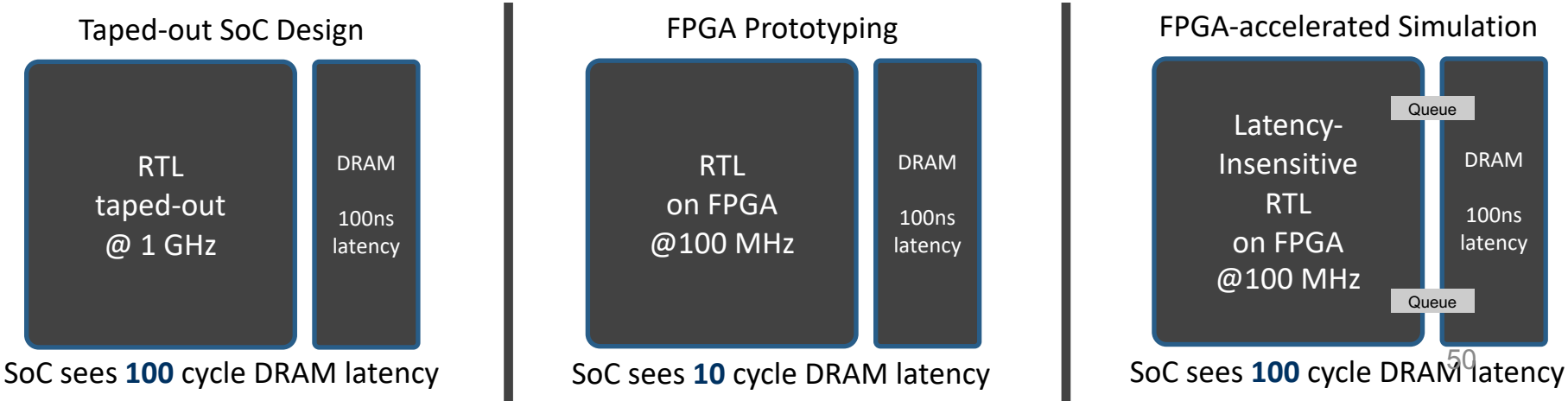
Detailed Tool Flow



FireSim

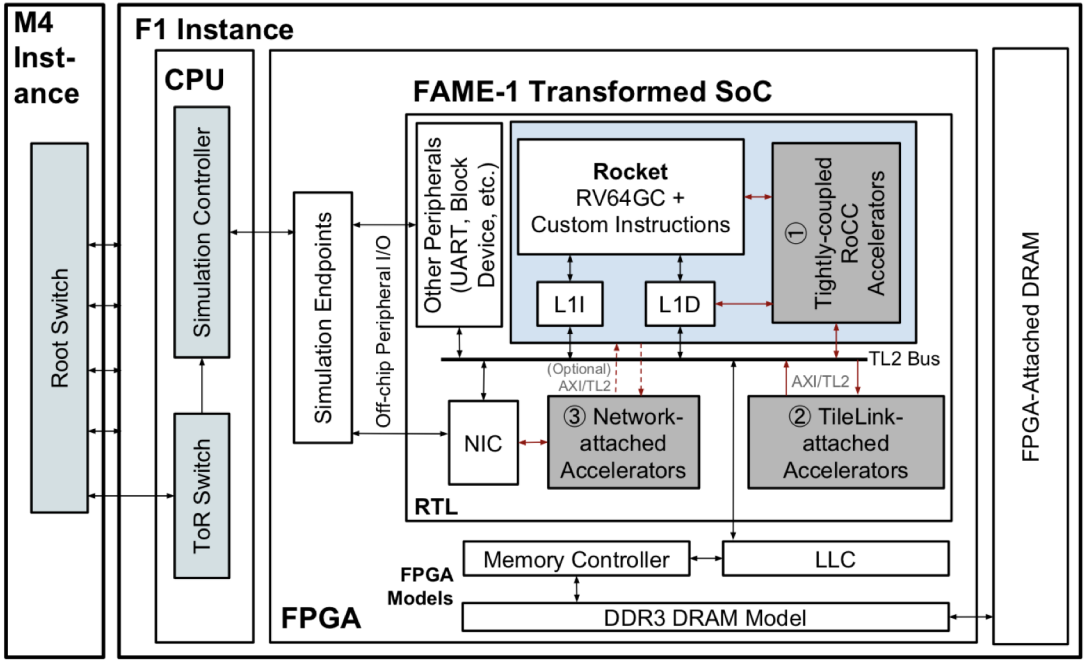
FireSim is an open-source, FPGA-accelerated, cycle-exact, scalable **hardware simulator**

- Produces:
  - FPGA-accelerated Simulation (Not FPGA prototype)
  - Deployment on Cloud FPGAs



Accelerator Coupling

1. RoCC Coprocessors
2. TileLink Accelerators
3. Network-attached Accelerators

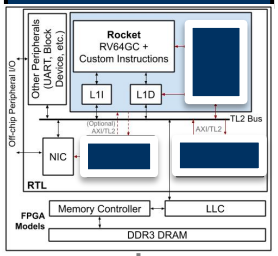


New Applications to Accelerate

- Packet Processing
- Video/Audio Compression/Decompression
- DNN Acceleration
- Graph Acceleration
- Database Systems

Centrifuge Tool Flow

Base SoC



Program

Func1
Func2
Func3
…

FireSim

Program

Func1 (90 %)
Func2 (8 %)
Func3 (1 %)
…

Hotspot

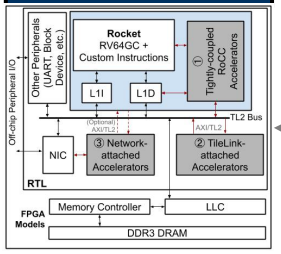
Func 1

```
for(i=0;i<n;i++)
#pragma pipeline II=N
for(j=0;j<m;j++)
#pragma unroll M
```

SW Interface

RISC-V LLVM/GCC

Accel SoC



HW Interface

1.RoCC
2.TileLink
3.Network-Attached

HLS

VIVADO HLx Editions

Automated