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
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
   o Analytical
   o 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
1. Workload Characterization

2. Accelerator Modeling
   o Analytical
   o 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
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
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

Transformed SoC

Profilled Programs
- func1 (90 %)
- func2 (8 %)
- func3 (1 %)

Step 2: SoC Generation with HLS

User Defines
- HLS C Func Pragma
  - void func1([]):
  - func2():
  - func3():

SoC Definition
- "soc":
  - "type": "soc":
    - "ramcap": "pipelined":
    - "func": "func1":

Auto-generated
- HW Interface
  - 1: RoCC
  - 2: TileLink
  - 3: Network-Attached

Accel RTL

Repeat from Step 1

Step 3: SW Primitives Generation

User Defines
- Runtime Env
  - "base": "pippin"
  - "base": "lime: green"
  - "iexec": "pippin"
  - "swexec": "pippin"

SW Primitives
- RoCC Primitives
- MMO Wrapper
- Linux Wrapper

SoC Definition
- "soc":
  - "soc": "soc":

Auto-generated
- SW Primitives
  - func1
  - func2
  - func3

SW Compiler
- RISC-V LLVM GCC

Generated
- Accel Programs
  - func1
  - func2
  - func3

Step 4: End-to-end Evaluation

Accelerated Programs
- func1 (81 %)
- func2 (91 %)
- func3 (1 %)
Step 1: Hotspot Detection on FireSim

Inputs

Input Programs
func1
func2
func3

Base SoC

Profiled Programs
func1 (90 %)
func2 (8 %)
func3 (1 %)

*Translated from the image as follows:*

```
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**

```c
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

**Accel SoC**

- Generated RTL
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

Transformed 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

- RoCC Coprocessors
- TileLink Accelerators
- Network-Attached Accelerators
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:
   ```cpp
   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

Three factors:
1. Cache Size
2. Cache Latency
3. Interface Bandwidth

Different Coupling for \textit{vadd} Accelerator

Determine the best hardware integration strategy
Exposed Design Space

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

- RoCC Coprocessors
- TileLink Accelerators
- Network-Attached Accelerators
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

- In most cases, the Linux overhead is not significant
- RoCC-level integration might be better option for `sort8`
Case Study 1: Smart-House Hub

- Application: A smart-house assistant

![Graph showing breakdown of key computational kernels]

1. Apply Amdahl's law in hotspot detection
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

Breakdown of key computational kernels

8x Speedup
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](https://github.com/hqjenny/centrifuge)
Email: qijing.huang@berkeley.edu

FireSim: [https://fires.im](https://fires.im)
Backup Slides
RoCC Coprocessor Integration

RoCC Interface

ap_bus to RoCC Decoupled Interface
Case Study 2: Distributed GEMM

- 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

<table>
<thead>
<tr>
<th>Workload Size</th>
<th>Total Ops</th>
<th>Ops/cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>32x16</td>
<td>196608</td>
<td>4.55</td>
</tr>
<tr>
<td>32x32</td>
<td>786432</td>
<td>15.12</td>
</tr>
<tr>
<td>32x64</td>
<td>3145728</td>
<td>20.59</td>
</tr>
<tr>
<td>16x128</td>
<td>3145728</td>
<td>21.35</td>
</tr>
<tr>
<td>8x64</td>
<td>196608</td>
<td>17.09</td>
</tr>
</tbody>
</table>
Case Study 3: Deep Learning

2. Distributed Accelerators

Multi-node accelerators, connected via Ethernet
Detailed Tool Flow

C Source File
- fns with `#pragma`
- Other fns

LLVM Pass/Scripts

C Src (HLS)
- Accelerated fns

HDL Accel. Verilog Implementation

Centrifuge SoC Accel Blackbox

Centrifuge Accelerated SoC

RISC-V Binary

RISC-V LLVM/GCC

Drivers
- SW fns

Runs on Centrifuge Accelerated SoC
FireSim is an open-source, FPGA-accelerated, cycle-exact, scalable hardware simulator

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

### FPGA Prototyping

- RTL taped-out on FPGA @100 MHz
- DRAM 100ns latency
- SoC sees 10 cycle DRAM latency

### Taped-out SoC Design

- RTL taped-out @ 1 GHz
- DRAM 100ns latency
- SoC sees 100 cycle DRAM latency

### FPGA-accelerated Simulation

- Latency-Insensitive RTL on FPGA @100 MHz
- DRAM 100ns latency
- SoC sees 100 cycle DRAM latency
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 (90 %)
Func2 (8 %)
Func3 (1 %)
...

FireSim

Program

Func1
Func2
Func3
...

Hotspot

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

Func 1

SW Interface

RISC-V LLVM/GCC

HW Interface

1.RoCC
2.TileLink
3.Network-Attached

HLS

VIVADO

Automated