Designing Heterogeneous Architectures

**How can we design such architectures?**
- Simulation: a way to explore the design space
- Reusing knowledge and design tools of the multi/many-core domain

**Existing works**
- Standalone accelerator: Aladdin [Shao, 2014]
- System-level simulation: gem5+Aladdin [Shao, 2016], PARADE [Cong, 2015]
- Cycle-accurate simulators are still quite slow

**What can we do?**
- Speeding up the cycle-accurate simulator
- Building performance models of accelerators

Harnessing FPGAs for Simulation not Prototyping

**Implement (parts of) the simulator on an FPGA**
- Parallelism
- FPGA-attachment technology
- FPGA is more and more powerful

**Performance model**
- Functionally equivalent and logically isomorphic
- Allowing abstractions which simplify model development, enabling modularity
Our contribution

- A methodology for generating performance models of accelerators on FPGAs

- A cycle-accurate simulator targeting CPU-FPGA platforms with the goal of speeding up the simulation

Outline

1. Accelerator modeling flow
   - Flow explanation by an example

2. System Integration
   - Simulation framework
   - Experimental study

3. Conclusion & Future work

Accelerator modeling flow

- Trace-based simulation
- Constructing a dynamic data dependence graph (DDDG)
- Optimizing and scheduling the graph
- Transforming the scheduled graph into compact instructions

Flow explanation by an example

A. C Code

```c
for(i=0; i<N; ++i)
c[i] = a[i] + b[i];
```

B. IR Trace:

```
0. r0=0  //i = 0
1. r4=load(r0 + r1) //load a[i]
2. r5=load(r0 + r2) //load b[i]
3. r6=r4 + r5
4. store(r0 + r3, r6) //store c[i]
5. r0=r0 + 1  //++i
6. r4=load(r0 + r1) //load a[i]
7. r5=load(r0 + r2) //load b[i]
8. r6=r4 + r5
9. store(r0 + r3, r6) //store c[i]
10. r0 = r0 + 1  //++i
```

B. IR Trace:

C. Un-optimized DDDG

Parameters | Configuration
---|---
Loop unrolling factor | 2
Loop pipeline | Off
Memory partitioning | Cyclic, factor-2
Accelerator Model Targeting FPGAs

- A factor of 2 loop iteration parallelism, partitioning factor 2, and without loop pipelining

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 bits: Op_code</td>
<td></td>
</tr>
<tr>
<td>1 bit: In new cycle</td>
<td></td>
</tr>
</tbody>
</table>

Address: 32 bits

op_code = load/store

E. Accelerator instructions

D. Scheduled DDDG

F. Timing model of accelerator

Simulation Framework

Framework Validation Using Blocked GEMM

A. Simulation vs Vivado HLS for Blocked GEMM

B. GEMM and Memory Structures

- The latency decreases when the loop unrolling factor increases.
- The performance is very close to Vivado HLS.
- The number of cycles increases as the complexity of memory grows.
Architecture Exploration

- Using MachSuite benchmarks
- Same level of precision as the gem5 simulator is observed.

Conclusion

- Generation of accelerator models targeting FPGAs
  - Exploited in HASim-based simulation framework
- Simulation framework allowing architecture exploration
  - Tested with single-core and accelerator models
  - Compared with other tools

Outline

1. Accelerator modeling
   - Flow explanation by an example
2. System Integration
   - Simulation framework
   - Experimental study
3. Conclusion & Future work

Future work

Multi-step greedy method
References


Questions

Intel Xeon-FPGA platform

- The Intel Xeon-FPGA tightly-coupled FPGA platform
- Minimizing communications between CPUs and FPGA

Source: https://pow.github.io/

RISC-V processor models

- Support RISC-V 32, 64 bits
- Integer Instruction Set Architecture

- Build two timing models: unpipelined and in-order pipelined
Our contribution

- Model accelerator on FPGA
- Allow architecture exploration