FACS

FPGA-Accelerated Multiprocessor Cache Simulator

Michael Papamichael [papamix@cs.cmu.edu], **Wei Yu** [wy@andrew.cmu.edu], **Yongjun Jeon** [yongjunj@andrew.cmu.edu]
http://www.cs.cmu.edu/~mpapamic/projects/facs.html

Computer Architecture Lab at Carnegie Mellon

Summary

Current architectural-level full-system software-based simulators (e.g. Virtutech Simics) are limited in throughput, especially when simulating multiprocessor systems. The slowdown becomes even higher when attaching additional modules to the simulator, such as cache simulators.

FACS (**F**PGA-**A**ccelerated **C**ache **S**imulator) is a fully parameterizable trace-based hardware functional **piranha-based** multiprocessor cache simulator that precisely replicates the behavior of the existing software-based **TraceCMPFlex** cache model. Our results show that FACS is over 200x faster than TraceCMPFlex.

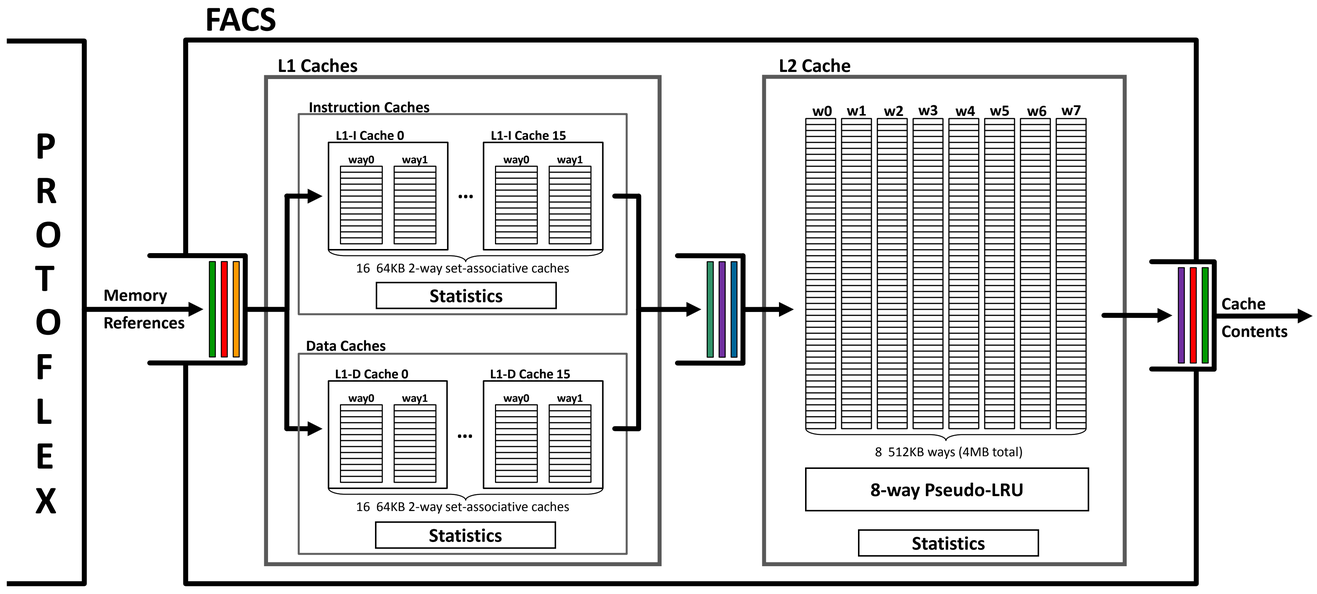
FACS in a Nutshell

- **Hardware Multiprocessor Cache Model**
  - Piranha-based 2-level Coherent Cache Hierarchy
  - 6-stage Pipeline Implementation
  - Fully Parameterizable Design
    - number of cores, L1/L2 block size, L1/L2 cache size, L2 associativity
- **Runs on Real Hardware (FPGA)**
  - Tested on BEE2 and XUP boards @ 100 MHz
  - High Throughput: 100 million references/sec
  - PowerPC Interface
    - feed references, read cache contents, read/write statistics memory
  - Traces reside on DRAM or Compact Flash Cards
- **Precise Replication of TraceCMPFlex SW model**
  - Over 200x Speedup

Implementation Details

- **2500 lines of Verilog code**
- **Functional Model**
  - Only tags and status bits stored and updated
- **L1 Caches**
  - Implemented as 2-stage pipeline
  - All 16 L1 I/D caches simultaneously accessed
- **L2 Cache**
  - Victim cache (only inserts evicted blocks from L1)
  - Each reference is processed in 2 cycles (not pipelined)

Architecture



Methodology

- **Collected memory reference traces from Apache workloads and fed to**
  - TraceCMPFlex (SW) running on Intel Xeon 5130 @ 2GHz (4MB L2) with 8GB RAM
  - FACS (HW) running on a BEE2 board @ 100MHz
- **Verifying Correctness**
  - Matched cache statistics and contents for both SW and HW cache models
- **Measuring Performance**
  - SW performance measured by actually timing the SW-based cache simulator
  - HW performance estimated by calculating best and worst case performance
    - **Best case**: less than half of the memory references miss in one of the L1 caches and have to travel to the L2 cache
    - **Worst case**: all of the references miss in the L1 caches and have to travel to the L2 cache

Performance Results

- **Processing rate of references:** FACS vs. TraceCMPFlex

	TraceCMPFlex (SW)	$FACS_{worst}$	$FACS_{best}$
Processing rate	455K refs/sec	50M refs/sec	100M refs/sec

- **Speedup:** 110x – 220x

Note: In all of our workloads the performance of FACS was very close to the best case performance. FACS performs worse than the best case only when the L1 miss rate exceeds 50%, which is very uncommon for any kind of workload.

Hardware Specs and Results

- **FPGA:** Xilinx V2P70 (BEE2 board)
- **Clock Frequency**: 100 MHz
- **Cache configuration**
  - 16 private 64KB 2-way split I&D L1 caches
  - 4MB 8-way shared L2
- **FPGA Utilization**
  - LUTs: 7602 (11%)
  - BlockRAMs: 136 (41%)

Lessons Learned & Future Work

- **Lessons Learned**
  - Simultaneous access to all of the L1 caches makes routing hard and limits scalability
  - L1 cache size should be chosen to match well with the FPGA BlockRAM dimensions
  - Verilog needs better support for writing synthesizable parameterizable modules
- **Future Work**
  - Increase L2 cache size without significantly reducing frequency
  - Take Memory Level Parallelism (MLP) effects into consideration
  - Support larger cache sizes through virtualization of the tag arrays

Acknowledgments

We would like to thank Eric Chung for his support on the BEE2 development board and Nikos Hardavellas for helping us out with the TraceCMPFlex software cache model. We would also like to thank James Hoe for providing us with previously developed pieces of Verilog code that greatly reduced the required implementation time.