Carnegie Mellon University
School of Computer Science
Computer Science Department

CS740: Computer Architecture

Project Proposal

FACS: FPGA Accelerated Multiprocessor Cache Simulator

Michael Papamichael [papamix@cs.cmu.edu]
Wei Yu [wy@andrew.cmu.edu]
Yongjun Jeon [yongjunj@andrew.cmu.edu]

Pittsburgh, PA, October 2007

Group Info:

- **Michael Papamichael** [papamix@cs.cmu.edu]
- **Wei Yu** [wy@andrew.cmu.edu]
- **Yongjun Jeon** [yongjunj@andrew.cmu.edu]

Project Web Page:

http://www.cs.cmu.edu/~mpapamic/projects/facs.html

Project Description:

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. Recent research in the field of hybrid simulation has greatly accelerated simulation using FPGAs (e.g. Protoflex [1,2] - http://www.ece.cmu.edu/~simflex/protoflex.html). This corresponds to arrow 1 in figure 1. However, attaching software-based modules (e.g. cache simulators) to FPGA-accelerated simulators greatly slows down the over-all simulation speed. The goal of FACS is to develop a functional simulator of a multiprocessor cache using FPGAs, which corresponds to arrow 2 in figure 1. Our ultimate goal is to integrate our cache module with Protoflex on a BEE2 board [2]. We will evaluate our success based on the speedup we get when compared to a purely software-based simulation.

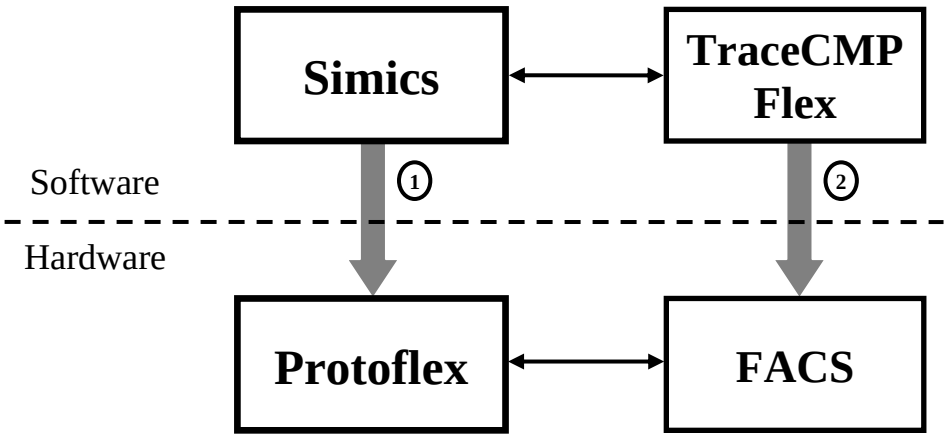

Figure 1: Software vs Hardware simulation

At this point there are two software-based cache simulators that were developed in the scope of the SimFlex project [3], which both implement a piranha-based cache hierarchy, similar to the one seen in figure 2. “CMPFlex” implements a detailed cycle-accurate model of a piranha-based cache. “TraceCMPFlex” is a simplified version of “CMPFlex”, which simulates a piranha-based cache at a functional level. We chose to replicate the behavior of “TraceCMPFlex”. This considerably lowers the design complexity and hopefully the implementation effort as well. Moreover, given the project timeframe, this decision makes reaching our goal in a timely manner more feasible.

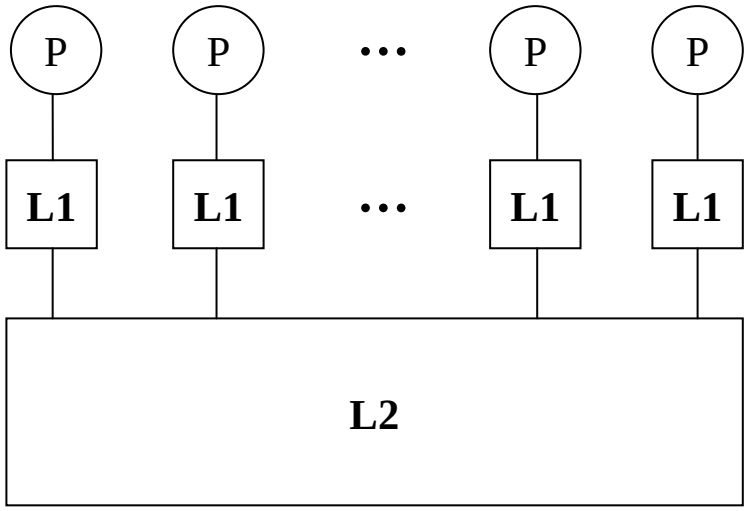

Figure 2: Piranha-based cache hierarchy

Logistics

Plan of Attack and Schedule

Week 1 (Oct 23 - Oct 30)

- Set up and get acquainted with the necessary development board and tools such as Simics and the XUP (Xilinx University Program) FPGA development boards
- Study and analyze available literature and TraceCMPFlex software model

Week 2 (Oct 30 - Nov 6)

- Study Flexus, Simics, TraceCMPFlex
- Implement L1 cache
- Implement a simple reference generator that reads from on-chip memory

Week 3 (Nov 6 - Nov 13)

- Debug, verify and reach 75% goal
- Add coherence support to L1

Week 4 (Nov 13 - Nov 20)

- Implement a reference generator that reads from off-chip DRAM
- Milestone presentation
- Implement L2 with coherence support

Week 5 (Nov 20 - Nov 27)

- Debug, verify and reach 100% goal
- Attempt 125% and 150% goals

Week 6 (Nov 27 - Dec 4)

- Finalize, evaluate, writeup and present

Milestone

By November 20th, we will have a L1 cache that supports coherence. To verify the correct behavior of our cache we will feed our L1 cache with references from traces that reside on on-chip memory.

Project Goals

If, at the end, we fall short of the 100% goal, we will modify the software-based TraceCMPFlex to match the behavior of our final implementation of FACS for more accurate performance comparison and evaluation.

75% Goal

- Study and analyze TraceCMPFlex, and confirm that it can be implemented in FPGA
- Implement a simpler uniprocessor L1 direct-mapped cache model with coherence support in hardware, as an intermediate step for extension to multiple processors
- Implement a simple reference generator that reads from on-chip memory

100% Goal

- Replicate TraceCMPFlex in hardware
- Feed TraceCMPFlex with references from traces that reside on off-chip DRAM

125% Goal

- Verify that FACS replicates the exact behavior of TraceCMPFlex
- Integrate FACS with Protoflex on the BEE2 board along with Protoflex
- Obtain results from running big traces and evaluate

150% Goal

- Support checkpointing, for better integration with the existing Protoflex project, which already supports checkpointing

Task assignment

Yongjun Jeon:

- Simics & Flexus
- TraceCMPFlex
- Other software-related tasks

Wei Yu & Michael Papamichael:

- L1 Cache model development
- L2 Cache model development
- Coherence verification
- Reference Generator

Resources Needed

- Simics simulator
- Flexus simulator source code
- TraceCMPFlex source code
- Modelsim (for verilog simulation
- Xilinx ISE (for FPGA synthesis)
- Xilinx EDK (for reference generator)
- Xilinx Chipscope (for hardware verification)

Getting Started

Up to the writing of this proposal we have been mostly studying the TraceCMPFlex source code, as well as setting up the FPGA-related tools that we are going to be using.

Bibliography

[1] ProtoFlex: FPGA-accelerated Hybrid Functional Simulation
Eric S. Chung, Eriko Nurvitadhi, James C. Hoe, Babak Falsafi, and Ken Mai.
CALCM Technical Report 2007-2, February 2007

[2] ProtoFlex: Co-Simulation for Component-wise FPGA Emulator Development
Eric S. Chung, James C. Hoe, and Babak Falsafi
In the *2nd Workshop on Architecture Research using FPGA Platforms (WARFP 2006)*, February 2006

[3] BEE2 Board: http://bee2.eecs.berkeley.edu/

[4] SIMFLEX: A Fast, Accurate, Flexible Full-System Simulation Framework for Performance Evaluation of Server Architecture
Nikolaos Hardavellas, Stephen Somogyi, Thomas F. Wenisch, Roland E. Wunderlich, Shelley Chen, Jangwoo Kim, Babak Falsafi, James C. Hoe, and Andreas G. Nowatzyk.
In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 31, No. 4, pp. 31-35, March 2004