Multicore Architectures Lab Assignments

Sohan Lal, Nadjib Mammeri, Ben Juurlink

Embedded Systems Architecture

Technische Universität Berlin

Einsteinufer 17, 10587 Berlin, Germany

{sohan, b.juurlink}@tu-berlin.de

25. Oktober 2017

This document contains the lab assignments of the Multicore Systems (MCS) course. The five assignments are grouped into two parts. Part 1 contains three tasks related to cache coherence. Part 2 contains two tasks related to GPU programming. It is recommended that you do the assignments in their respective order so that the lab and lecture topics are synchronized.

Each assignments has deliverables which are further described in the assignments. The deliverables have to be submitted via the ISIS page of MCS, which has a link for each assignment in the Grading and Examination section. The assignments can be performed in groups of 2-3 students. Submit only one assignment per group.

Deadlines of the assignments are:

Part 1 Cache Design - Task 1	**14.11.2017**
Part 1 Cache Coherence - Task 2	**05.12.2017**
Part 1 Cache Coherence - Task 3	**09.01.2018**
Part 2 GPUs - Task 1	**30.01.2018**
Part 2 GPUs - Task 2	**20.02.2018**

Submitting the deliverables past the deadline reduces your maximum points for the respective assignment. You can submit the assignments until midnight on the day of the deadline.

This part contains assignments 1-3, and is based on the lab assignments of the course *Advances in Computer Architecture*[1] given by Prof. C. R. Jesshope, University of Amsterdam.

Introduction

In the Cache Coherence part of the lab, you will use SystemC to build a simulator of a Level-1 D-cache and various implementations of a cache coherency protocol and evaluate their performance. The simulator will be driven using trace files which will be provided. For information about cache coherencies and memory, please refer to the lecture notes, or see Appendix A in file doc/Appendices.pdf.

The framework for this part of the lab can be downloaded from the ISIS page. Download it to your local home directory, and extract it (tar -xvf part1_student_srcs.tar.gz). This framework contains all documentation, the helper library with supporting functions for managing tracefiles and statistics, the tracefiles, and a Makefile to automatically compile your assignments. Using the Makefile, the contents of each directory under src/ is compiled automatically as a separate target. It already includes directories and the code of the Tutorial, as well as a piece of example code for Task 1.

Trace files are loaded through the functions provided by the helper library (aca2009.h and aca2009.cpp), which is in the provided framework. Detailed documentation for these functions and classes is provided in Appendix C in file doc/Appendices.pdf, but the example code for Task 1 also contains all these functions and should make it self-explanatory. Note that, although later on simulations with up to eight processors with caches are required to be built, it is easier if you start with a single CPU organization. Then, extend it to support multiple CPUs. Make sure that the number of CPUs and caches in your simulation is not statically defined in your code, but uses the number of CPUs read from the trace file. When you are debugging, please print important events on the console or in a log file, in order to evaluate the correctness of each action in your cache system. Part of the evaluation of your work is also based on this log information when you present your work to the lab assistants.

Trace files

There are 3 kinds of trace files available to you: random, debug and FFT. All versions exist for 1, 2, 4 and 8 processors. The random (rnd_pX.trf) trace files have the processors read or write memory randomly in a window in memory that is moving. The debug (dbg_pX.trf) trace files are a short version of the random trace files, suited for debugging your simulation. The FFT (fft_16_pX.trf) trace files are the result of the execution of a FFT on a 64K array and can be used, along with the random trace files, to generate results for your reports. All reads and writes are byte-addressed, word-aligned accesses.

Submission and reports

Students can work in groups of 2-3 persons. Submitting work is through the ISIS page, and includes:

- the source code (packed in tar/gz/zip format) at the end of each task

[1] http://staff.science.uva.nl/ mwvantol/teaching/ACA2009/

- a brief report (in pdf, with cover page indicating group members and assignment name) at the end each task with your results, explanation of results. Moreover, at the end of Task 3, please also include a comparison between the protocols implemented in Tasks 2 and 3.

General guidelines:

- Your source code should compile and run without any modifications on the Linux lab machines, either using the provided Makefile or your own which has to be included.
- The first commandline argument your program accepts is the name of the tracefile. (this happens automatically when the init_tracefile function is used).
- Hit/Miss rate statistics should be gathered with the provided functions, and should be printed with the stats_print function after the simulation ends.
- Your simulations should always terminate and exit without any errors.
- Supporting a different number of CPUs in your simulation should not require recompilation.

Additional Information

Even though attending the lab sessions is not compulsory, we would suggest you to come in regularly. This is because the lab assistants will have the most time for answering your questions during the sessions. If you want to work from home or your own machine and want to be able to test if your code is working properly on the Linux lab machines, you can log in to the student systems from outside using SSH (wiki.freitagsrunde.org/SSH).

If you had any questions for the lab assistant outside the lab time, you can email him, and if necessary visit him at his office after arranging an appointment (by email too).

1.	Fachgebiet Architektur eingebetteter Systeme **Multicore Architectures**	WS17/18

Assignment start: 25.10.2017 Submission deadline: 14.11.2017

Assignment 1 - SystemC Intro and L1 Cache **25 Points**

In this task you will build a simulator for L1 D-cache. The cache size is 32kB, 8-way set-associative and cache line size of 32-Byte. The simulator must be able to be driven with the single processor trace files. Assume a Memory latency of 100 cycles, and single cycle cache latency. Use a least-recently-used, write-back replacement strategy.

Notes:

- Read doc/SystemC_tutorial.pdf file, and implement the examples mentioned there.
- You can use the example code provided for Task 1, which is based on the tutorial, as a guide and modify the Memory module so that instead of RAM, it simulates a set-associative data cache.
- As we only simulate accesses to memory, it is not necessary to simulate any actual contents inside the cache. Furthermore, you do not need to simulate the actual communications with the memory, just the delay.
- We expect that when running your code, it will generate a waveform file, which includes the important signals in your design, for example: clock, address, set number, line number in case of a hit, line number to be replaced in case of a miss, hit/miss, read/write, etc. Please make a SNAPSHOT of the wave file and include it in your report
- In addition to your source code, please also submit a brief report, not more than one page, including your calculations of the tag/index/offset bits, your results that are printed at the end of run, in addition to the total run time, and average memory access time.
- In the report, include your observations about the results such as reasons for variation in hit rates across different benchmarks.
- The SystemC library is already configured, installed and accessible on the lab machines in: `/afs/tu-berlin.de/units/Fak_IV/aes/tools/mca/systemc-2.3.0`