Assignment P2 – Cache Simulator

Formal assignment description for P1 - INFOMOV
Jacco Bikker, 2018

Introduction

This document describes the requirements for the second assignment for the INFOMOV course. For this assignment, you will implement a cache simulator. You can test the simulator with the included test application, which renders fractals.

Fractals

Unmodified, the application renders a *Buddhabrot* fractal (note: historically *Ganesh* is more accurate). This is a fractal similar to the Mandelbrot fractal, and consists of the set of points in the complex plane for which the sequence $z_{n+1} = z_n^2 + c$ does *not* tend to infinity for $z_0 = 0$ (as described by Wikipedia). The actual implementation is as mysterious as the previous sentence. One part of the code matters: the two lines that read and write data to iteratively update the image buffer. More on that in a second.

With a small modification, the application can be made to render a *Barnsley fern* fractal. Details on this fractal can also be found on Wikipedia, but again, the only relevant part is the code that interacts with the image buffer.

Memory access

The two fractals have very different memory access patterns. The Buddhabrot randomly skips over the screen, while the fern has a more deliberate pattern.

The application has been augmented with an interface to the cache simulator. Every read from and write to the image buffer is replaced by a function call. E.g., reading a uint is now a call to `LOADUINT`, which takes the address of the uint and returns the value at that address.

The function calls allow us to intercept the memory operations and guide them through the simulator. The implementation of this cache simulator is the goal of this assignment.

Note that only the image buffer operates via the cache simulator, to limit the scope of this assignment.

Cache

The provided implementation of the cache is obviously incomplete. Every memory operation triggers the fetch or store of a full cache line: in actual hardware, this is the only interface between cache and memory system, and you may thus not alter this behaviour. To simulate the latency involved with RAM access, an artificial delay is created whenever RAM is touched.

Your task is to prevent memory access. The idea is that in most cases data can be retrieved from a cache line stored in the cache, or written to a cache line that is already in the cache. In these cases, RAM is not accessed and the artificial delay is prevented. In some cases a write may require eviction of data already in the cache, which obviously requires a memory operation, with the associated cost.

Details

For this assignment, you will implement a correct **set associative cache**, with a reasonably accurate eviction policy. You can integrate this with the supplied test application. If you wish to work in a different language than C/C++, you may also provide your own test application. Building a correct single layer cache yields the first 6 points for this assignment.

Additional points may be obtained by:

- implementing a cache hierarchy (L1-L2-L3) (+1pt);
- implementing and comparing at least three realistic eviction policies. Realistic means that the simulator accurately mimics an existing hardware implementation (+1pt);
- implementing a way to gather data on cache performance over time (+1pt).

The final point is reserved for exceptional work.

Note that points will be subtracted if the cache is not correctly implemented. One symptom of this could be changed application functionality. Other symptoms include cache performance (hit/miss ratio) that differs significantly from what could be expected; compare with peers to verify your results.

Note that the performance of your simulator is irrelevant. The simulator could very well reduce the efficiency of the application beyond the current slow speed caused by the artificial delay.

For this assignment you may assume a single core. It is thus not necessary to simulate the effects of false sharing, nor any inter-core synchronization.

Team

You may work on this assignment alone, or with one partner. You may team with one partner for all assignments, but it is also allowed to change teams per assignment. You cannot change your team halfway an assignment; if for whatever reason you don't want to finish the project with your partner, both of you will work alone. Both team members may continue working with the code that was produced up till the split.

You may exchange information about the project with other students, online or in real life. Do not share code snippets, limit the exchange to ideas, hints, and concepts.

Deliverables

Your submission will consist of a **report** plus **project files**. Make sure the code compiles out-of-the-box in VS2017. If any other tools are required to produce the intended executable, please add a `readme.txt` that contains build instructions. The report should describe your cache architecture, an analysis of cache performance, a statement on work division, references to sources used in the process, and an overview of implemented functionality (especially for the "additional points").

Deadline

The deadline for this assignment is **Thursday October 4, 23:59**. Please submit your work using the SUBMIT system. If you fail to meet this deadline, you may submit one day later. One point will be subtracted from your grade in this case.

Academic Conduct

The work you hand in must be your own original work, or properly referenced. If you used materials from other sources, please specify this clearly in the report.

Do not store your work in a publicly accessible location (this includes github!). If other students use your work (now or in the future), you may be reported along with the perpetrators.

Purpose

The purpose of this assignment is to gain insight in the caching system of the CPU. The practical work partially replaces a literature study, as hands-on experience typically yields a better understanding of this important topic.

The End

Questions and comments:

bikker.j@gmail.com or room 4.24.

INFOMOV 2018