

Assignment P2 – Cache Simulator



Universiteit Utrecht

Formal assignment description for P2 - INFOMOV

Jacco Bikker, 2024

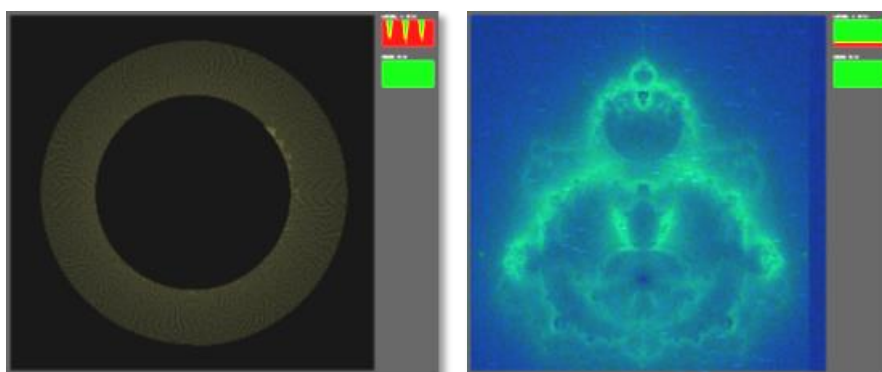
Introduction

This document describes the requirements for the second assignment for the INFOMOV course. For this assignment, you will extend a simple cache simulator, which currently implements a fully associative cache.

Base Code

The base code in `game.cpp` renders a spiral. The contents of a simulated two-level memory system (L1 cache + RAM) are visualized in real-time: bright colors are cached; darker ones reside in 'DRAM'. The provided *fully associative* cache uses a random eviction policy, so pixels of the spiral will randomly leave the cache, resulting in a pretty sparkly trail.

An alternative chunk of code 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](https://en.wikipedia.org/wiki/Buddhabrot)). The actual implementation is as mysterious as the previous sentence, on purpose. Luckily, only one part of that code matters: the two lines that read and write data to iteratively update the image.



Memory access

The two code paths have very different memory access patterns. The Buddhabrot randomly skips over the screen, while the spiral has a more predictable pattern.

The application has been augmented with an interface to a 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 `mem.WriteUint`, 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.

Cache

The starting point for this assignment is a working but basic cache simulator, which implements the *fully associative cache* scheme. You can find the implementation in `cache.h` and `cache.cpp`.

The implemented cache uses a memory hierarchy, with one cache level (derived from abstract base class `Level1`), and a simulated DRAM memory (class `Memory`, also derived from `Level1`). Communication between DRAM, the single cache level and the `MemHierarchy` object always happens in full cache lines (class `CacheLine`). `MemHierarchy` is the access point for the 'CPU', i.e. our code in `game.cpp`. The CPU code may access memory in bytes or `uints`, although only `uints` are used in this case.

Details

For this assignment, you will implement a correct **N-way set associative cache** and expand the single-level cache to a **three-level hierarchy**. Building a minimal but correct 3-layer set associative cache yields the first 5 points for this assignment. You may use a simple eviction policy for this, and you may use hardcoded cache parameters if you wish.

Additional points beyond the base 5 may be obtained by completing additional challenges:

1. Besides the spiral and the fractal, add a third algorithm to assess cache behaviour / performance. Try to come up with an access pattern that reveals other characteristics than the existing two algorithms (up to 1pt).
2. Add support to the code for setting per-level associativity, cache size and cache line width. These may be compile-time parameters. Use this functionality to answer the following research question: What is the optimal set of parameters for the spiral, and what parameters are optimal for the fractal? Which set of parameters do you recommend for general use and why? (up to 1pt)
3. Implementing LRU, LFU and random eviction for your cache levels and comparing these using the provided access patterns (spiral + fractal), as well as your own (up to 1pt).
4. Implement a comparison of the implemented eviction policies against the *Clairvoyant* algorithm (Bélády's algorithm) (+1pt).
5. Add the option to make the cache inclusive or exclusive. Add a comparison of the efficiency of both schemes, or include this in the experiments for challenge 2 (up to 1pt).

Note that **points may 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.

For this assignment you may assume a single core. It is thus not necessary to simulate the effects of false sharing, nor of 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**, **source code** and **project files**.

Make sure the code compiles out-of-the-box in Visual Studio. 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 challenges").

Deadline

The deadline for this assignment is **Friday May 31, 17:00**. If you fail to meet this deadline, you may submit one day later. One point will be subtracted from your grade in this case. Please submit your work by mail.

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



INFOMOV 2024