Assignment 1: a single L1d writeback cache

Sam van Kampen

January 20, 2022

1 Introduction

In this assignment, we were asked to implement a single level 1 data cache, which was 8-way set-associative, performed writeback, and used a least-recently-used replacement policy. In this report, the implementation and results are described and analyzed.

2 Implementation

The implementation is fairly straightforward. Given the size of the cache (32 kibibytes), the amount of ways (8), and the size of a cacheline (32 bytes) we can calculate the amount of cache lines and sets. The cache is then represented as an array of `CACHE_SETS` sets containing `CACHE_WAYS` lines containing cache metadata. In this simple case, the required metadata is a tag, a dirty bit, a valid bit and an access time (or some other way like a counter to indicate which line is least recently used). Each address consists of a 20-bit tag, a 7-bit set number and a 5-bit offset within a cache line, for a total of 32 bits.

When a memory operation is issued, the cache is checked by taking the set from the address and comparing each line in the set for being valid and having the correct tag. If there is a match, we have a cache hit, and after updating the access time we can perform the memory operation. In case it is a write, the line is marked as dirty so that it is written back to memory when it is evicted.

If there is no match, we need to load the cacheline from memory. We select a new cacheline from the set (an unused line, or, if none are available, we evict the least recently used line) and fill it.

3 Results

The cache statistics can be seen in table 1.

	Reads	Read hits	Read misses	Writes	Write hits	Write misses	Hitrate
dbg_p1.trf	40	19	21	60	26	34	45.000000
fft_16_p1.trf	86298	7682	78616	43195	10882	32313	14.335910
rnd_p1.trf	33031	18659	14372	32505	18306	14199	56.404114

Table 1: Cache statistics for the single L1d cache

4 Discussion

Perhaps surprising is the performance of the FFT trace; a hitrate of 14 percent is pretty abysmal, and the other hitrates aren't great either. This indicates that there are a lot of conflicts within a cache set, causing a large number of evictions and memory loads. Ways of mapping addresses to cache sets which reduce these conflicts are used in practical caches, for instance using functions that XOR different address bits to determine the cache set.