Computer Organization: Programming Assignment #2

Vicente Adolfo Bolea Sánchez

June 18, 2016

Abstract

In this assignment, I implemented a n-ways associative cache using C programming language. The cache data structure is a dynamic allocated array of blocks. The cache supports *LRU* eviction and different levels of associativity, block sizes and cache sizes.

Approach

I structured the source code into six different functions: four of them implement the low-level logic while the remaining of them are dependent on those four *low-level()* functions. Both *index_of_addr()* and *tag_of_addr()* computes the respective index and tag of the given virtual address.

cache_access() calls *index_of_addr()* and *tag_of_addr()* and with the computed tag and index call *insert_block()* with those two parameters. *insert_block()* will then try to find the given block in the cache (using *get_block()*) and in case that it does not exist it will check if the corresponding set in the cache is already full (using *is_full()*). In case the it is not full it will just locate that block in that free slot. Alternatively, if it is full it will evict the LRU element in that set(using *evict_block()*). After each insertion in each set we would increment the time attribute of each block in that set using *increment_all()*.
There are two parameters which can be adjusted in compile time: *WORD_SIZE* by default 32bits in MIPS, the size of a word; *WORD_WIDTH* by default 0, value 0 assumes that the virtual addresses specify a whole 32bits word. Normally, if we want to access an specific byte of the block we would change *WORD_WIDTH* to 2.

The source code is free of memory errors and leaks, it passed valgrind extensive memory test.

Rationale

The architecture of the logic was developed using a *top-bottom* approach. Specifically, in a *rolling-wave* manner, this is, at first I wrote the high level functions such as *cache_access()* and *insert_block()*. After that, I implemented the rest of the *low_level*

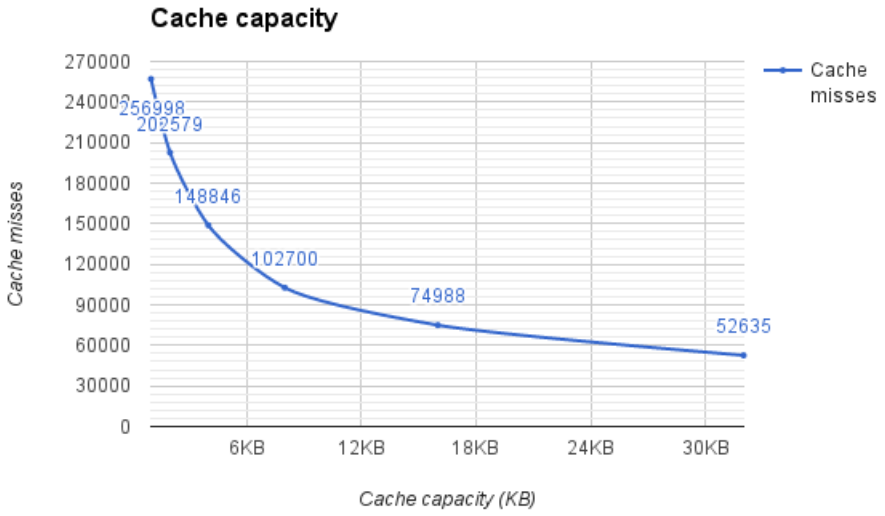
functions. Once the source code had some consistency I attempted it to compile it and debug it. Later, I refactored the code and repeated the same compiling and debuging cycle.

Experiments

For the experiments I create a small bash script called launcher which launch and filter the output of cache_sim.

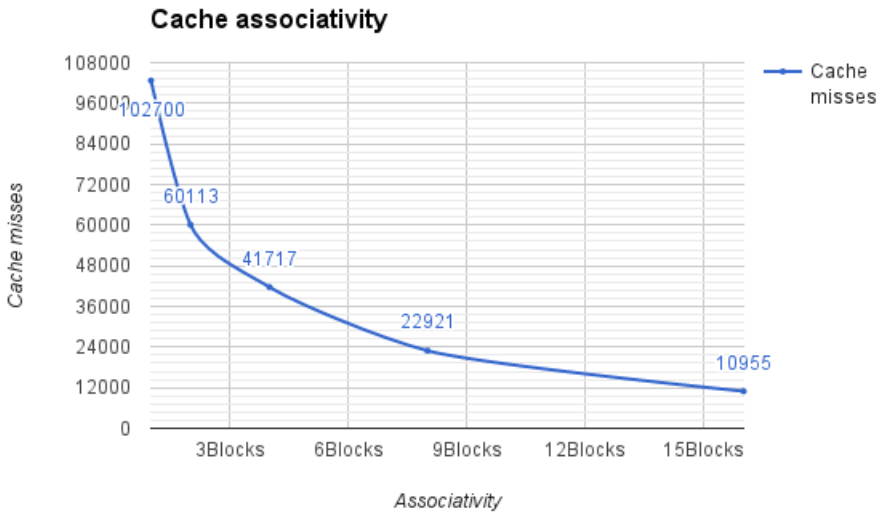
0.1 Cache capacity vs. cache misses

In this setup we will measure the cache misses using the following cache capacities: 1KB, 2KB, 4KB, 8KB, 16KB, 32KB. The associativity is 1 and the block size is 32 bytes.



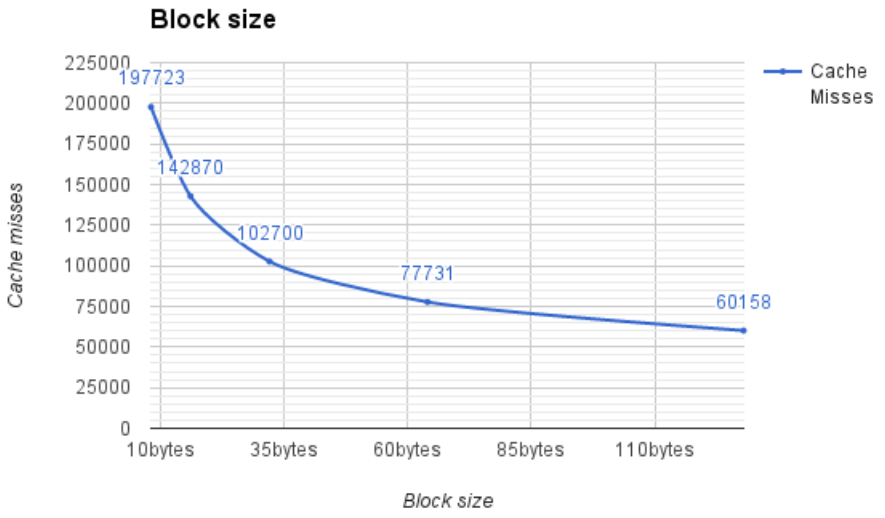
0.2 Cache associativity vs. cache misses

In this setup we will measure the cache misses using the following set associativies: 1, 2, 4, 8, 16. The cache size is 8KB and the block size is 32 bytes.



0.3 Cache block size vs. cache misses

In this setup we will measure the cache misses using the following block sizes: 8B, 16B, 32B, 64B, 128B. The cache size is 8KB and the associativity is 1.



Conclusion

After analizing those three experiments I conclude that increasing associativity leads to an higher increase in cache hits compared to increasing the block size or the number of sets (cache size). One of the possible explanations for the poor performance when increasing the cache size is that the given address were relatively near to each other (high spacial locality).

Due to that high spacial locality increasing the associativity or block size allows the cache to allocate more nearby addresses. However, if we just increase the block size the cache would easily populate without evicting not frequently or recently used elements. Thus, increasing associativity would have a similar effect, however, it will evict those blocks which are not frequently or recently used dramatically increasing the performance of the cache.

Tools and libraries

- `assert.h` library to keep the invariant in some functions.
- `stdbool.h` to enable bool types.
- `GNU toolchain: gdb, gcc, and valgrind` for compiling, debuging and memory tests.
- `Bash shell` to create the launcher script.
- `Google spreadsheet` for the charts.
- `Latex` to create this report.
- `Vim` to edit all the source codes.

Changes from the skeleton code

```
diff cache_sim.c ../pa/cache_sim.c | diffstat
unknown |  244 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++----
1 file changed, 191 insertions(+), 53 deletions(-)
```