lab1 Cache

Jiacheng Ma

October 2019

1 Introduction

The lab is about adding instruction cache and data cache simulation into a simulation program of MIPS. And explores the parameter settings' influence on performance include cache size, associativity, and block size. Explores the best performing replacing policy to do cache replacement.

2 Design

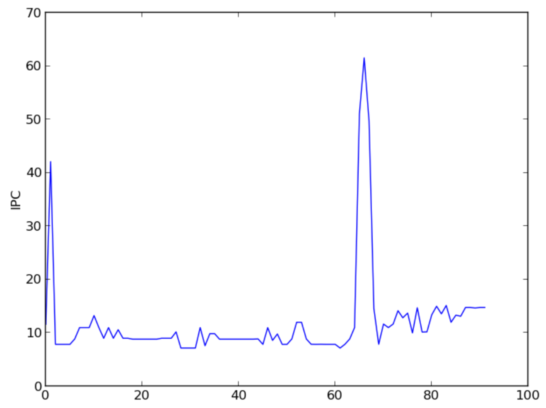
The instruction cache is 8 KB in size with 32-byte blocks, 64 sets and 4-way associativity. The data cache is initially 8-way set-associative with 32 byte blocks and 64KB total size. The replacing principle is least recent. The c program uses a struct containing a tag, dirty bit, valid bit, cache block array(8bit for each cell), recent bits and frequency bits. The last several bits of memory request address is used to address the byte as offset in the block. The second last several bits is used to address the set in cache by directly mapping without any hash. The remaining address bits is used as a tag to uniquely identify memory byte. When new block needs to be inserted into cache, if there is a vacancy in corresponding set, the block is inserted there. Otherwise, use least recent principle, replace the least recent block and updates recent bits of all other blocks in the cache. For every cache hit, increase frequency bits by 1.
For timing simulation, each memory request cost 50 cycles stall. For baseline MIPS, shell.c is modified to include 50 cycles stall for every read and write in memory, just for comparison. For cache-enabled MIPS, 50 cycles is enforced when reading or writing memory in a block. The differences is in baseline MIPS, accessing every byte cost 50 cycles stall, while in cache-enabled MIPS, accessing a block cost 50 cycles stall. But since the simulation is to explore the cache performance, the baseline MIPS is just used as a comparison.

3 Experiments and Results

3.1 Cache size, block size, associativity

Instruction cache is not changed. The following experiment is done by changing data cache. The experiments settings are summariesed in the table. Keep block size 32 byte and vary set size and associativity.
Surprisingly, those experiments show very similar results or are even identical. Thus only one result is presented in 3.1. It is possible that all the settings are ample for all the testing inputs, while program gets stuck when further lower set-associative below 4 way, leaving no space to further constrain the cache. The reason why it gets stuck is unknown. Other experiments include lower

block size are carried out as well. But result doesn't vary a lot. In extreme case where the block size is set to 1 byte, The cache-enabled MIPS reveals different result showing 4x performance degradation in terms of IPC than normal.
Overall, the experiments show many settings above provides sufficient space for cache to operate well, as the IPC increases average 10x than baseline from 3.1. Hence the the bottleneck is really the cache design. One big problem is the testing input programs didn't explore much locality reasonably as well. Due to time constrain, it is left to next step.



Experiment	block size	set size	way
1	32byte	64set	32way
2	32byte	64set	64way
3	32byte	64set	128way
4	32byte	64set	256way
5	32byte	128set	16way
6	32byte	128set	32way
7	32byte	128set	64way
8	32byte	128set	128way
9	32byte	256set	8way
10	32byte	256set	16way
11	32byte	256set	32way
12	32byte	256set	64way

Figure 1: The results from first experiment with block 32b, 64set, 32way.Y-axis is IPC ratio of MIPS with cache compared to baseline, x-axis represents different test inputs

3.2 Replacement and insertion policies

Two replacement and insertion policies are tested. One is least recent and the other is least frequent principle. The cache settings are maintained as the initial one with 8-way set-associative, 32 byte blocks and 64KB total size. The results didn't show any difference for all the testing inputs. Because the objective is to explore replacement policy efficiency, limit block size and vary set-associativity might help enlarge the difference for different policies. But even when the blocks size is lowered down to 1 byte, and vary set-associative to explore possible difference, the results didn't show any difference as all.

4 Discussion and Conclusion

The major problem of all the experiments falls on input programs. The program is not complex enough to chanllenge cache design. The future work is write good test programs to let cache performance vary. And beyond that, explore more cache design.
In short, the cache improves the performance by average 10x in terms of IPC across 90 programs.

References

[1] Wikipedia *Cache replacement policie*
`https://en.wikipedia.org/wiki/Cache`$_{r}eplacement_{p}oliciesLeast_{f}requent_{r}ecently_{u}sed_{(}LFRU)$