Cache

18-1 ECE 322 Computer Organization, Lab 07

Hansung Kim
2014-16824

1 Introduction

Implement cache on top of the previous pipelined CPU implementation. Evaluate the speedup achieved by using cache by comparing no cache version CPU with the new CPU.

2 Design

Several design points are:

- Write-through, write-no-allocate policy
- Separate cache for instruction and data, both 16-word, direct-mapped
- Memory with an 4 word read, 4 word write inout port (no single-word write)
- inputReady-based asynchronous I/O signal model that supports variable latency
- Parameterized module that can switch between cached/baseline model easily

2.1 Write miss handling

The multi-word write of the memory has some implication on the write-miss performance of the cache: it should read the whole cache block that contains the data to be written, before it can write to the memory. As a result, a write-miss takes 12 cycles which is a big performance hit. This can be improved if the memory supports a single-word write. (This was not done because I was unaware that the specification allows memory to have separate data size for read and write.)

After writing to the memory using a consecutive memory read and write, no further operation is needed because write-no-allocate policy is used for the write miss policy.

2.2 Stall logic

The "safe" stall logic mentioned above is devised because of the probability that a control hazard can occur during a read-miss of the instruction cache. For I-cache miss, only the IF stage is stalled and the ID and later stages is designed to carry on. However, if the hazard unit detects a wrong branch prediction during the IF stall, it fails to reset PC because `pc_write` has to remain zero in order to stall IF.

```
i1: IF ID EX MEM WB
i2:    IF IF IF  ...
       ^  ^
       control hazard of i1 may happen here, but
       IF of i2 does not respond due to read-miss stall
```

Figure 1: Illustration of control hazard during I-cache read miss

This can be easily solved by just staling the entire pipeline for a read miss and thus preserving the control hazard signals until the read miss resolves. However, because of the bug that this change uncovered in the baseline implementation in the last minute of the deadline, the "stupid" version of stall logic is retained.

3 Implementation

A new module for the cache is implemented in `cache.v`. It is connected with datapath and control units in `cpu.v`.

Several implementation details and the performance measurement are presented.

3.1 I/O signaling

To accommodate for different memory latency for baseline and cache implementation without changing logic, the input_ready signal is used to signify the master that the operation is finished and ready on the inout port. As soon as input_ready signal rises, the master resets the readM/writeM signal to zero, making the memory ready for the next operation. This is shown in `memory.v` and `cache.v`.

3.2 Baseline implementation

The implementation can be simply switched between cache-enabled or baseline by changing the parameter called `CACHE` given to the cpu module and the memory module in `cpu_TB.v`.

To minimize code change for cache and baseline implementation, the baseline is implemented by modifying the cache to simply forward cache read/write operation to the memory, and the cache data

inout port to the memory inout port. This way we can reuse the I/O signaling logic implemented in the cache (section 3.1), e.g. waiting for input_ready to be on, for the baseline as well.

3.3 Measurements

All measurements are done under the 2-bit hysteresis counter branch prediction and data forwarding from the previous design enabled.

The performance comparison between the cache and baseline implementation under the given workload is presented in Table 1.

Implementation	# of cycles	IPC
Baseline	4102	0.2392
Cache	2939	0.3338

Table 1: Performance of the cache-enabled and baseline implementation

The speedup of the cache version over the baseline is 1.396.

The reason why the baseline exhibited such a high number of cycles is due to the use of relaxed stall condition that stalls 2 more cycles for every instruction read (section 2.2). This rather extreme logic is used because a bug in the stall condition, causing memory read and write signal to 'race' under load and store instructions, could not be fixed in time. This affects both the cache and baseline implementation, but because the baseline issues memory read for every instruction, the performance impact is bigger for the baseline.

The hit/miss count and ratio for the given workload is presented in Table 2.

Cache	Hit	Miss	Total	Hit ratio
Instruction cache	1391	159	1550	0.897
Data cache	234	12	246	0.951
Total	1625	171	1796	0.905

Table 2: Hit ratio for both I-cache and D-cache

The instruction cache exhibits notably higher cache hit ratio. This can be attributed to the better spatial locality of the instruction memory over the data memory, and that the execution time of the Fibonacci loop takes the major part of the total execution time. A loop also exhibits high temporal locality, as well as high spatial locality, because it reads the same instruction repeatedly over time.

4 Discussion

The implementation was successful and speedup from the cache could be measured. However, the inefficient stall logic and the choice of multi-word write port for the memory negatively affected the performance. If the stall logic had been modified to eliminate the unconditional 2-cycle stall for every instruction read, the baseline implementation (currently taking 4 cycles per instruction) would have been sped up greatly. Considering the cache version would not see as much performance increase due to fewer instruction memory access, the speedup would not have been as high. On the other hand, if the memory was revised to support single-word write, it would make the additional memory read in each cache write miss unnecessary, speeding up the cache version. Overall, the measured speedup value may not be a good representation of the benefit coming solely from cache, for the given workload.

The measurement showed high cache hit ratio of 90% which does indicate the cache version can potentially greatly benefit performance for workloads that have better spatial and temporal locality.

5 Conclusion

Successfully implemented cache on top of the previous pipelined CPU implementation. Evaluated speedup achieved by using cache by comparing no cache version CPU with the new CPU.