Verilog 实验 lab2 实验报告

PB19071405 王昊元

2022 年 05 月 01 日

1 实验目的

1. 权衡 cache size 增大带来的命中率提升收益和存储资源电路面积的开销
2. 权衡选择合适的组相连度（相连度增大 cache size 也会增大，但是冲突 miss 会减低）
3. 体会使用复杂电路实现复杂替换策略带来的收益和简单替换策略的优势（有时候简单策略比复杂策略效果不差很多甚至可能更好）
4. 理解写回法的优劣

2 实验要求

1. 理解提供的直接映射策略的 cache，将它修改为 N 路组相连的 cache，并通过 cache 读写测试。
2. 使用编写的 N 路组相连 cache 正确运行提供的几个程序。
3. 对不同 cache 策略和参数进行性能和资源的测试评估，编写实验报告。

3 实验环境

- vlab
- Vivado 2016.3

4 实验核心实现

1. 在直接映射策略的 cache 上进行一些改动，例如改变 `cache_mem`、`cache_tags` 等的数组维数，修改命中规则（即改成对组内每一个 line 进行判断，并记录 line 在组内的索引），此外还需要修改状态机的状态跳转过程，这是由 cache 相关数组维数改变所引起的。
2. 在基本的改动之后，需要实现 N 路组相连 cache 的换出策略，FIFO 和 LRU 两种策略，我们通过维护一个 `info` 变量来实现。对于 FIFO 策略，`info` 变量记录该 line 从进入 cache 到现在的时间，对于 LRU 则记录从上次被命中到现在的时间，而两种策略都取各自时间最久的来换出，这也是使用一个变量可以维护两种策略的原因。此时只需要在未命中的部分添加选择要换出的 line 索引

即可（实现中命中 line 的索引和该处未命中要换出 line 的索引用同一变量维护）。此外，由于只有时间的绝对大小并没有实际作用，所以我们对于 FIFO 策略，只在换入换出时更新 `info`，因为命中时该组每个 line 对应的数值都加 1，相对大小不变，不影响换出。下为该部分代码实现：

```
// maintain info variable
always @ (posedge clk or posedge rst)
begin
    if(rst)
    begin
        for(integer i = 0; i < SET_SIZE; i++)
        begin
            for(integer j = 0; j < WAY_CNT; j++)
            begin
                info[i][j] <= 32'b0;
            end
        end
    end
    else
    begin
        if(mode == FIFO)
        begin
            // 考虑到在没有换入换出的情况下，所有info都+1不会改变它们的大小关系，故只在有换入换出时维护info
            if(cache_stat == SWAP_IN_OK)
            begin
                for(integer i = 0; i < WAY_CNT; i++)
                begin
                    if(i == mem_line_idx)
                    begin
                        info[set_addr][i] <= 32'b0;
                    end
                    else
                    begin
                        info[set_addr][i] <= info[set_addr][i] + 1;
                    end
                end
            end
        end
        else if(mode == LRU)
        begin
            if(signal && miss == 1'b0)  // if hit
            begin
                for(integer i = 0; i < WAY_CNT; i++)
                begin
                    if(i == line_idx)
                    begin
                        info[set_addr][i] <= 32'b0;
```

```
                    end
                    else
                    begin
                        info[set_addr][i] <= info[set_addr][i] + 1;
                    end
                end
            end
            else if(cache_stat == SWAP_IN_OK)  // if not hit, then swap, and only
                update info when swap finish.
            begin
                for(integer i = 0; i < WAY_CNT; i++)
                begin
                    if(i == mem_line_idx)
                    begin
                        info[set_addr][i] <= 32'b0;
                    end
                    else
                    begin
                        info[set_addr][i] <= info[set_addr][i] + 1;
                    end
                end
            end
        end
    end
end
```

3. 除此之外，需要对 CPU 的 `Hazard` 部分进行修改，在 cache miss 时，CPU stall。

5 实验结果及分析

5.1 阶段一结果验证

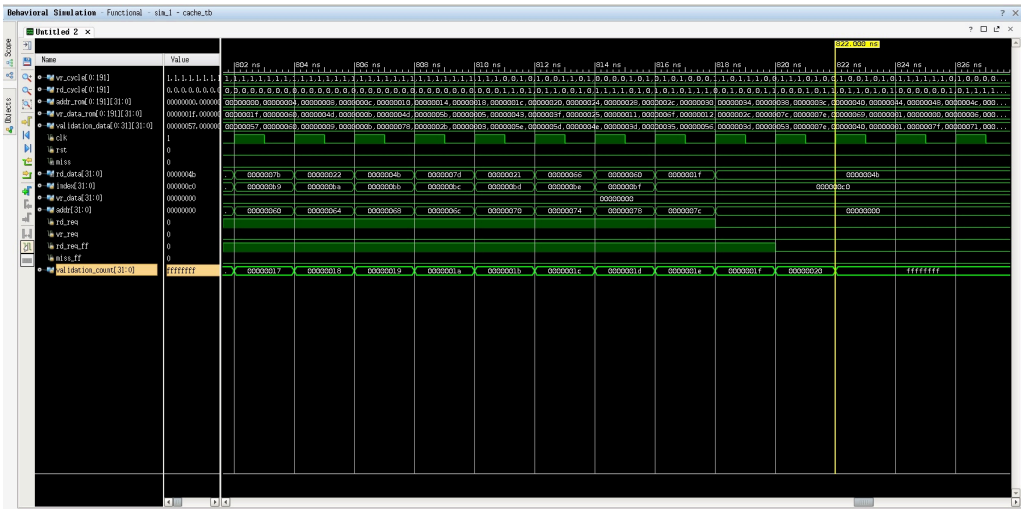


图 1: 阶段一结果截图

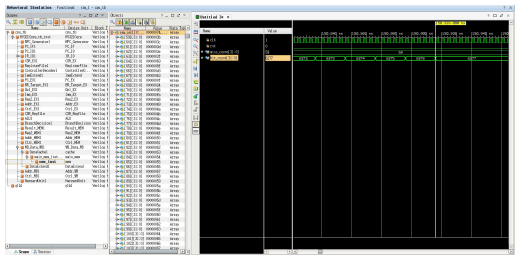
从图1可以看出，`validation_count` 不断增加，最后变为 `ffffffff`，通过了 cache 读写测试。

5.2 阶段二结果验证

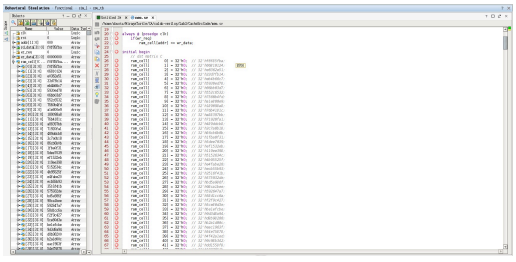
因为个人没有 windows 系统，不方便生成太多不同规模的程序，故以下结果验证及分析阶段只使用规模为 256 的快排及规模为 16 的矩阵乘法进行。

下面结果验证部分的 cache 参数为 `LINE_ADDR_LEN = 3 SET_ADDR_LEN = 3 TAG_ADDR_LEN = 6 WAY_CNT = 4`。

(1) FIFO 策略

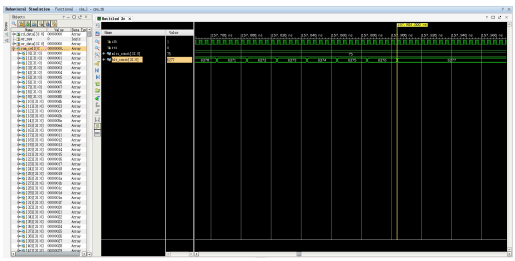


(a) 快排

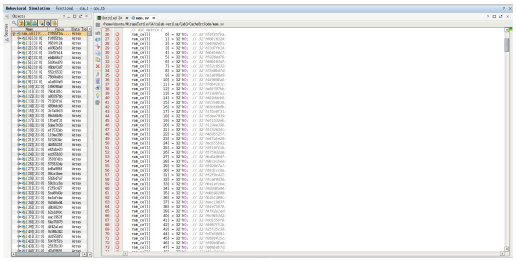


(b) 矩阵乘法

(2) LRU 策略



(c) 快排



(d) 矩阵乘法

可以看出矩阵乘法的结果与预期结果一致，快排结果大部分都是有序的，其中少部分乱序是由于 cache 的存在，部分结果没有写回内存中。

5.3 性能分析

表 1: FIFO 策略下进行快排序

LINE_ADDR_LEN	SET_ADDR_LEN	TAG_ADDR_LEN	WAY_CNT	Time(ns)	miss	hit	miss_rate
3	3	6	4	150912	59	6293	0.93%
3	3	6	2	188224	148	6204	2.33%
3	3	6	3	169760	102	6250	1.61%
2	3	7	4	224440	256	6096	4.03%
4	3	5	4	136872	20	6332	0.31%
3	2	7	4	184188	138	6216	2.17%
3	4	5	4	140976	39	6313	0.61%

表 2: LRU 策略下进行快排

LINE_ADDR_LEN	SET_ADDR_LEN	TAG_ADDR_LEN	WAY_CNT	Time(ns)	miss	hit	miss_rate
3	3	6	4	157904	75	6277	1.18%
3	3	6	2	186328	144	6208	2.27%
3	3	6	3	167444	98	6254	1.54%
2	3	7	4	227236	266	6086	4.18%
4	3	5	4	137720	22	6330	0.35%
3	2	7	4	185260	161	6191	2.53%
3	4	5	4	141400	40	6314	0.63%

表 3: FIFO 策略下进行矩阵乘法

LINE_ADDR_LEN	SET_ADDR_LEN	TAG_ADDR_LEN	WAY_CNT	Time(ns)	miss	hit	miss_rate
3	3	6	4	150912	1739	8900	16.35%
3	3	6	2	1363116	4864	5775	45.72%
3	3	6	3	1321644	4672	5967	43.91%
2	3	7	4	1340684	4760	5881	44.73%
4	3	5	4	276516	72	10567	0.68%
3	2	7	4	1349292	4800	5839	45.12%
3	4	5	4	293732	144	10495	1.35%

表 4: LRU 策略下进行矩阵乘法

LINE_ADDR_LEN	SET_ADDR_LEN	TAG_ADDR_LEN	WAY_CNT	Time(ns)	miss	hit	miss_rate
3	3	6	4	723800	1935	8704	18.19%
3	3	6	2	1345188	4781	5858	44.94%
3	3	6	3	1191188	4089	6550	38.43%
2	3	7	4	1299804	4590	6049	43.14%
4	3	5	4	295660	150	10489	1.41%
3	2	7	4	1260868	4407	6232	41.42%
3	4	5	4	307956	205	10434	1.93%

通过以上数据分析对比，可以得到以下结论：

- 随着 LINE_ADDR_LEN、SET_ADDR_LEN、WAY_CNT 的增大，丢失率逐渐降低，即当增大 cache 容量时，cache 存储数据量增加，命中率提高。
- 对于快排问题，LRU 的效果与 FIFO 相差无几，且各有胜负，对于矩阵乘法问题，LRU 效果略好于 FIFO，但效果也不显著。
- 对于快排问题，计算量较小，cache 容量增大产生的命中率提高并不明显（相较之下），但对于矩阵乘法问题，增大 cache 容量，能看到很明显的命中率提高，在某些情况下可能引起数量级上的改变。

5.4 资源分析

在 cache 参数为 LINE_ADDR_LEN = 3 SET_ADDR_LEN = 3 TAG_ADDR_LEN = 6 WAY_CNT = 4，采用 FIFO 策略时，资源占用如下：

Utilization - Post-Synthesis

Resource	Estimation	Available	Utilization %
LUT	3541	63400	5.59
FF	8047	126800	6.35
BRAM	4	135	2.96
IO	81	210	38.57
BUFG	1	32	3.13

由于篇幅原因，以下数据用表格形式展示。

表 5: FIFO 策略下的资源数量

LINE_ADDR_LEN	SET_ADDR_LEN	TAG_ADDR_LEN	WAY_CNT	LUT	FF
3	3	6	4	4270	10415
3	3	6	2	2152	5678
3	3	6	3	3541	8047
2	3	7	4	2738	5970
4	3	5	4	7741	19340
3	2	7	4	2629	5695
3	4	5	4	7477	19823

表 6: LRU 策略下的资源数量

LINE_ADDR_LEN	SET_ADDR_LEN	TAG_ADDR_LEN	WAY_CNT	LUT	FF
3	3	6	4	4339	10417
3	3	6	2	2458	5680
3	3	6	3	3598	8042
2	3	7	4	2809	5972
4	3	5	4	7795	19342
3	2	7	4	2696	5697
3	4	5	4	7615	19825

从以上数据可以看出：

- 可以看出，cache 容量的增加，会导致电路资源消耗更多，并且这种增长很快。
- 对照两种策略可以看出，FIFO 和 LRU 的资源消耗并没有明显差异，原因可能是因为我 FIFO 和 LRU 采用了几乎相同的设计，区别较大的可能是 FIFO 在命中时不更新（都加 1 不影响相对大小），LRU 在命中时更新。

6 实验总结

1. 通过本次 Cache 相关实验，我对 cache 有了更加清晰的认识，包括分类、工作原理、性能相关等等。
2. 了解了 Vivado 的综合功能，也利用该功能进行了综合分析，也更加熟悉 Verilog 的编程。
3. 同时也增加了一些 cache 设计、Vivado 设计相关的经验。