Lab3 实验报告

PB18051098 徐碧涵

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

实验环境

Vivado2018.3

实验过程

实现 N 路组相联 cache

需要将 cache 增加一个维度，将原来的 1 路扩展至 N 路

```
reg [31:0] cache_mem[SET_SIZE][WAY_CNT][LINE_SIZE];
reg [TAG_ADDR_LEN-1:0] cache_tags [SET_SIZE][WAY_CNT];
reg valid [SET_SIZE][WAY_CNT];
reg dirty[SET_SIZE][WAY_CNT];
reg[31 :0] last_use[SET_SIZE][WAY_CNT];
```

在对 cache 访问时发生变化，由于在使用 set 定位到对应的组后，需要的块可能在该组中的任意一个位置，故而需要遍历组内的块比较 tag 以及 valid 位信息来判断需要的块在 cache 中的位置

```
reg cache_hit = 1'b0;
always @ (*) begin
    index = 0;
    cache_hit = 0;
    for(integer    i = 0; i < WAY_CNT; i++)     //遍历组内块
    begin
        if(valid[set_addr][i] && cache_tags[set_addr][i] == tag_addr)
        begin
            cache_hit = 1'b1;    //如果组内某一块 tag 和访问地址一致并且 valid 即命中
            index = i;           //记录命中的块在组内索引
```

```
        end
    end
    if(cache_hit == 1'b0)      //miss
    begin
        for (integer i = 0; i < WAY_CNT; ++i)
        begin
            if(last_use[set_addr][index] < last_use[set_addr][i])   //选择换出的块
            begin
                index = i; //记录换出的块在组内索引
            end
        end
    end
end
```

LRU 替换策略

每个块记录其上一次被使用至当前时间所经过的 Cache 操作次数（即期间每一次对 cache 访问时该值会相应累计），可知越久没使用的块该值也会相应更大，即可以与记录上一次使用至当前所经过的时间起到相同的作用。那么在需要进行换出时选择记录值最大的块进行换出即可，该块即最久未使用的块。为了实现记录值的计数，每当 rd_req 或 wr_req 有效时表示对 cache 的一次操作，由于当 cache miss 时，rd_req 或 wr_req 会长时间有效，为了防止重复计数，可以引入 ref_signal（有效时代表对 cache 的一次访问）用于提取 rd_req 或 wr_req 的一个时钟周期用于计数。使用记录维护如下：

```
            //一次访问命中时
            if (ref_signal && miss == 1'b0)
            begin
                for(integer i = 0; i < WAY_CNT; ++i)
                begin
                    if (i == index)        //index 是当前访问 hit 的块
                    begin
                        last_use[set_addr][i] <= 32'b0;     //该块当前被使用，记录清零
                    end
                    else
                    begin
                        last_use[set_addr][i] <= last_use[set_addr][i] + 1; //其余未使用的块记录值加一，表示其从上次使用到当前时间 cache 使用的次数又增加了一次
                    end
                end
            end
            //访问 miss 时，在换入的时候，换入的块记录值也应该为 0，其他块记录值加1
            else if (cache_stat == SWAP_IN_OK)
            begin
                for(integer i = 0; i < WAY_CNT; ++i)
```

```
            begin
                if (i == mem_in_index)
                begin
                    last_use[set_addr][i] <= 32'b0;
                end
                else
                begin
                    last_use[set_addr][i] <= last_use[set_addr][i] + 1;
                end
            end
        end
```

FIFO 替换策略

FIFO 与 LRU 策略不同的是块只在换入时开始计时即使之后被访问也不会重新开始计时，所以代码如下：

```
        if (cache_stat == SWAP_IN_OK)
        begin
            for(integer i = 0; i < WAY_CNT; ++i)
            begin
                if (i == mem_in_index)
                begin
                    last_use[set_addr][i] <= 32'b0;
                end
                else
                begin
                    last_use[set_addr][i] <= last_use[set_addr][i] + 1;
                end
            end
        end
    end
```

结果分析

cache 消耗资源评估

改变组相联度，其余参数不变（LINE_ADDR_LEN = 3, SET_ADDR_LEN = 3, TAG_ADDR_LEN = 6）令 WAT_CNT 分别为 2，4，6，8，查看资源消耗

下图展示的时 WAY_CNY = 2，LRU 替换策略的资源分析

Resource	Utilization	Available	Utilization %
LUT	3581	63400	5.65
FF	8053	126800	6.35
BRAM	4	135	2.96
IO	82	210	39.05

FIFO

Resource	Utilization	Available	Utilization %
LUT	3581	63400	5.65
FF	8053	126800	6.35
BRAM	4	135	2.96
IO	82	210	39.05

由于我在实现 LRU 与 FIFO 时使用策略大致相同，同样采取 last_use 数组用于计时，故而两者使用资源理论上应该是一致的，所以之后仅查看不同 cache 的参数变化对于 LRU 策略的资源使用情况，为了更好的展示对比效果，下面对每种参数的变化直接使用一个列表整体描述使用资源的 utilization，不再展示截图。

	LUT	FF	BRAM	IO
WAY_CNT = 2	5.65%	6.35%	2.96%	39.05%
WAY_CNT = 4	8.44%	8.22%	2.96%	39.05%
WAY_CNT = 6	11.21%	11.96%	2.96%	39.05%
WAY_CNT = 8	13.07%	15.69%	2.96%	39.05%

保持 cache size 不变，改变组数和组相联度，（LINE_ADDR_LEN = 3）分别测试 8 组 1 路组相联，4 组 2 路组相联，2 组四路组相联

	LUT	FF	BRAM	IO
8 组 1 路	1.82%	2.42%	2.96%	39.05%
4 组 2 路	3.48%	2.62%	2.96%	39.05%
2 组 4 路	4.34%	2.63%	2.96%	39.05%

改变块大小（SET_ADDR_LEN = 3, TAG_ADDR_LEN = 6，WAY_CNT = 2），测试 4，8，16 个字大小的块对资源的消耗

	LUT	FF	BRAM	IO
4 字	2.30%	2.58%	2.96%	39.05%
8 字	5.65%	6.35%	2.96%	39.05%
16 字	6.85%	8.31%	2.96%	39.05%

缺失率分析

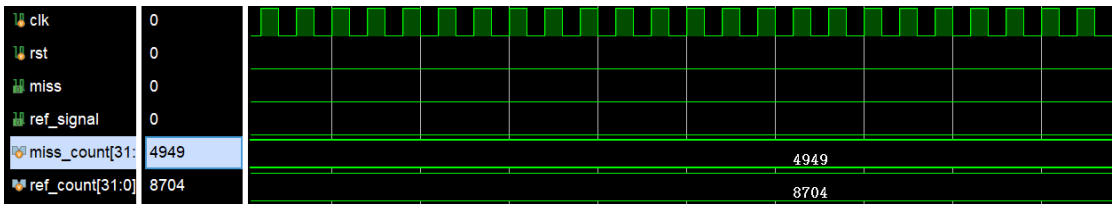
针对以上不同参数的 cache 以及替换策略执行排序算法以及矩阵乘法得到 cache 的缺失率聪儿评价不同的 cache 对于程序运行的优化。

在程序运行时对 ref_signal 以及 miss 进行计数，得到访问 cache 的次数以及缺失次数，两者相除可得到 cache 缺失率。

矩阵乘法 16*16

访存次数：8704

由于图片比较多不一一展示，将结果全部由表格展示



不同相联度使用不同替换策略的 cache 在执行矩阵乘法算法时的缺失率

	2 路	4 路	6 路	8 路
LRU	0.5686	0.2352	0.0458	0.0207
FIFO	0.5717	0.1998	0.0211	0.0168

Cache size 不变改变组数和组相联度

	8 组 1 路	4 组 2 路	2 组 4 路
LRU	0.6075	0.6034	0.5611
FIFO	0.6075	0.6002	0.5855

改变块大小

	4 字	8 字	16 字
LRU	0.5792	0.5686	0.4512
FIFO	0.5790	0.5717	0.1373

排序算法 256 个元素

访存 6472 次

不同相联度使用不同替换策略的 cache 在执行排序算法时的缺失率

	2 路	4 路	6 路	8 路
LRU	0.0692	0.0116	0.0074	0.0063
FIFO	0.0769	0.0256	0.0063	0.0063

Cache size 不变改变组数和组相联度

	8 组 1 路	4 组 2 路	2 组 4 路
LRU	0.1284	0.1225	0.1185
FIFO	0.1284	0.1145	0.1128

改变块大小

	4 字	8 字	16 字
LRU	0.1486	0.0692	0.0077
FIFO	0.1403	0.0769	0.0071

观察两个算法的执行得到的缺失率，可以看到，对于不同的组相联度，组相联度越大缺失率最小，但显然占用资源也越多，当相联度为 4 时已经有比较低的缺失率。对与相同的 cache size，相联度越高缺失率越低，对于不同的块大小，可以看到块越大缺失率越小，但显然其占用资源也越多。可以看到 LRU 以及 FIFO 替换策略在测试中缺失率相差不大，有时 FIFO 甚至表现更好，考虑到 FIFO 实现更为简单，可见选择 FIFO 更佳。

综上分析，对于矩阵乘法可以使用 8 组 4 路块大小为 4 个字替换策略为 FIFO 的 cache，此时已经有较低的缺失率。对于排序算法，可以选择使用 8 组 2 路块大小为 4 个字替换策略为 FIFO 的 cache。