CA Lab3 实验报告

PB18071463 朱映

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

使用的硬件：xc7a100tcsg324-1

- **实验工具**：Vivado，MacBook Pro 13，Windows 10，VS CODE
- **实验方式**：Vivado自带的波形仿真

实验内容

- 阶段一：理解提供的直接映射策略的 cache 并它修改为 N 路组相连的 cache，并通过我们提供的 cache 读写测试。
- 阶段二：使用阶段一编写的N路组相连 cache，正确运行矩阵相乘和快速排序。
- 阶段三：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。

阶段一：组相联 cache

对每个原来直接映射的数组加入新的一维，作为路数。中间的这个地址就是FIFO或则LRU需要产生的地址。原有的hit也需要修改（对每一路都需要检查才能确定）。

实现 FIFO：FIFO 思想是先进先出，因此对每一组都维护和路数相同的数组存储存入的地址，并维护队头和队尾信息。初始化都设置为0，新换入的地址为队尾，新换出的作为队头。如果组内有空地址，就优先使用；否则换出队头的地址。代码如下。

```
// implement FIFO
reg [WAY_CNT-1:0] FIFOQueue [SET_SIZE][WAY_CNT];    //FIFO队列
reg [WAY_CNT-1:0] FIFO_addr;             //FIFO产生的way_addr，可以用于cache
reg [WAY_CNT-1:0] Front [SET_SIZE];      //队头信息
reg [WAY_CNT-1:0] Rear  [SET_SIZE];      //队尾信息
reg flag;
```

```
initial begin
    for (integer i = 0;i < SET_SIZE;i++) begin
        Front[i] = 0;
        Rear[i] = 0;
    end
end

always @ (*) begin
    if (cache_stat == IDLE && !cache_hit && (rd_req | wr_req)) begin // 要读或写
        flag = 0;
        for (integer i = 0;i < WAY_CNT;i++) begin
            if (!flag && !valid[set_addr][i]) begin     // 这一路有空的地址
                flag = 1;
                FIFO_addr = i;
            end
        end
        if (!flag) begin                        // 修改队头队尾，分配一个队头位
            FIFO_addr = FIFOQueue[set_addr][Front[set_addr]];   // 等于队头地址（相
当于把队头移除）
            if (Front[set_addr] != Rear[set_addr])           // 队头不是队尾，那么说
明队列里是有数据的，更新队头位置
                Front[set_addr] = (Front[set_addr] + 1) % WAY_CNT;
        end
    end
    else if (cache_stat == SWAP_IN_OK) begin // 将换入的地址加入队列，更新队尾
        FIFOQueue[mem_rd_set_addr][Rear[mem_rd_set_addr]] = mem_rd_index_addr;
        Rear[mem_rd_set_addr] = (Rear[mem_rd_set_addr] + 1) % WAY_CNT;
    end
end
```

实现LRU：LRU在队列满时应换出最少使用的那个地址，因此维护一个记录所有地址在cache中存在时间的数组。每次未使用地址，相对应的年龄就加一。如果有换入或者换出，则地址年龄设置为 0。需要换入新地址时，如果有空位则直接换入，否则换入年龄最大的地址并设置年龄为0.代码如下。

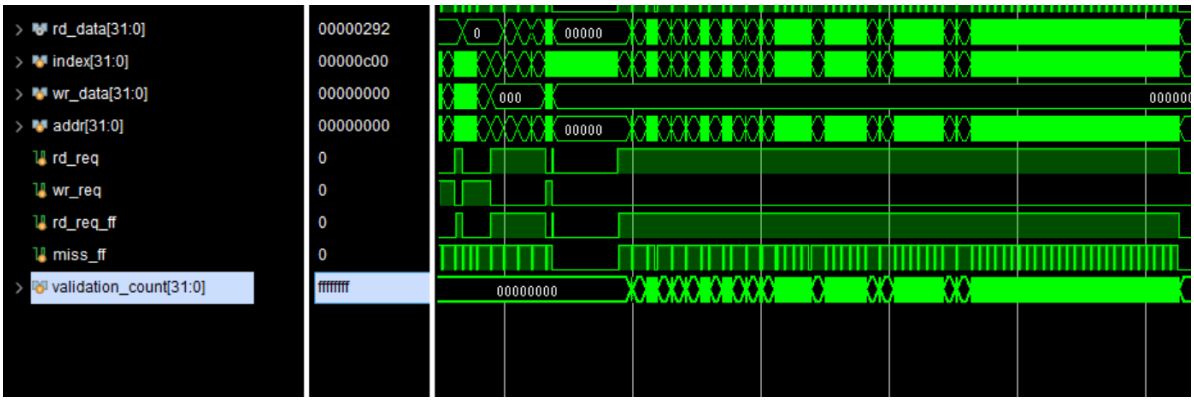
```
reg [31:0] Usage [SET_SIZE][WAY_CNT];    // 记录使用次数
reg [WAY_CNT-1:0] LRU_addr;              // LRU产生的换入地址
integer max;

initial begin
    for (integer i = 0;i < SET_SIZE;i++) begin
        for (integer j = 0;j < WAY_CNT;j++) begin
            Usage[i][j] = 0;
        end
    end
end

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++)
                Usage[i][j] = 0;
        end
    end
    else begin
        for (integer i = 0;i < SET_SIZE;i++) begin  // 开始时，对每一个数的未使用次数
+1
```

```
            for (integer j = 0;j < WAY_CNT;j++)
                Usage[i][j] = Usage[i][j] + 1;
        end
        max = 0;
        for (integer i = 0;i < WAY_CNT;i++) begin
            if (max < Usage[set_addr][i]) begin
                max = Usage[set_addr][i];       // 找到最久不用的地址将它清除
                LRU_addr = i;
            end
        end
        if (cache_stat == IDLE && cache_hit) // 命中了以后应当将该处设置0
            Usage[set_addr][way_addr] = 0;
        else if (cache_stat == SWAP_IN_OK)    // 换入的地址对应的使用年龄为0
            Usage[mem_rd_set_addr][mem_rd_index_addr] = 0;
    end
end
```

运行结果如下（两种策略结果相同，不重复展示）：



阶段二：在 CPU 上连接 cache 并运行

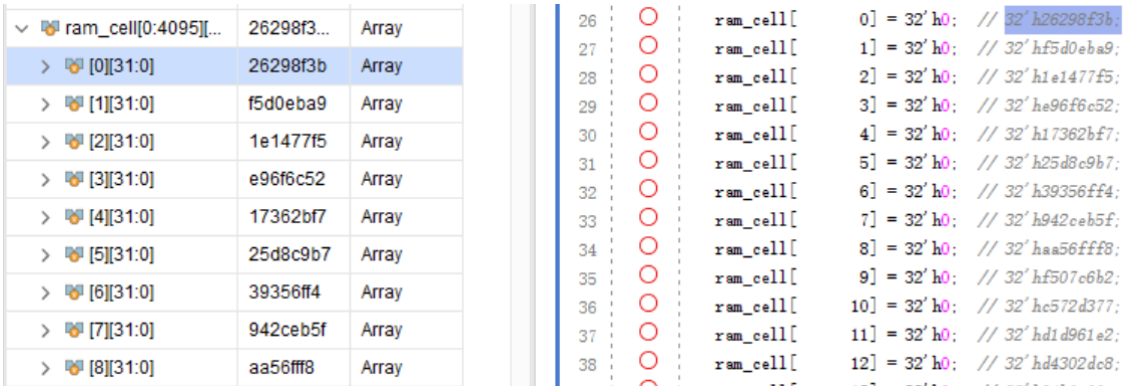
1. 修改 CPU 的数据通路，将cache替换原有的 dataram，连接到 WBSegReg 上并添加 CacheMiss信号，连接到 Hazard Unit上。
2. 并添加额外的数据通路，统计Cache缺失率。当有cache miss信号时，miss加一；当有读写请求且没有miss信号且访问的地址变化时，hit加一。

```
    cache DataCacheInstance(
        .clk(clk),
        .rst(rst),
        .miss(CacheMiss),
        .addr(A),
        .rd_req(MemToRegM),
        .rd_data(RD_raw),
        .wr_req(|WE),
        .wr_data(WD)
    );
reg [31:0] hit_count = 0, miss_count = 0, prev = 0;   // counter
    wire rd_wr = (|WE) | MemToRegM;
    always @ (posedge clk or posedge rst) begin
        if(rst) prev <= 0;
        else if( rd_wr ) prev <= A;
    end

    always @ (posedge clk or posedge rst) begin
        if(rst) begin
```

```
            hit_count  <= 0;
            miss_count <= 0;
        end else begin
            if( rd_wr & (prev!=A) ) begin
                if(CacheMiss) miss_count <= miss_count+1;
                else hit_count <= hit_count +1;
            end
        end
    end
```

MatMul运行结果如下（FIFO和LRU运行结果一致）：



ram_cell[0:4095][...	26298f3...	Array
[0][31:0]	26298f3b	Array
[1][31:0]	f5d0eba9	Array
[2][31:0]	1e1477f5	Array
[3][31:0]	e96f6c52	Array
[4][31:0]	17362bf7	Array
[5][31:0]	25d8c9b7	Array
[6][31:0]	39356ff4	Array
[7][31:0]	942ceb5f	Array
[8][31:0]	aa56fff8	Array

```
26  ram_cell[  0] = 32'h0;  // 32'h26298f3b;
27  ram_cell[  1] = 32'h0;  // 32'hf5d0eba9;
28  ram_cell[  2] = 32'h0;  // 32'h1e1477f5;
29  ram_cell[  3] = 32'h0;  // 32'he96f6c52;
30  ram_cell[  4] = 32'h0;  // 32'h17362bf7;
31  ram_cell[  5] = 32'h0;  // 32'h25d8c9b7;
32  ram_cell[  6] = 32'h0;  // 32'h39356ff4;
33  ram_cell[  7] = 32'h0;  // 32'h942ceb5f;
34  ram_cell[  8] = 32'h0;  // 32'haa56fff8;
35  ram_cell[  9] = 32'h0;  // 32'hf507c6b2;
36  ram_cell[ 10] = 32'h0;  // 32'hc572d377;
37  ram_cell[ 11] = 32'h0;  // 32'hd1d961e2;
38  ram_cell[ 12] = 32'h0;  // 32'hd4302dc8;
```

QuickSort运行如下（FIFO和LRU运行结果一致）：

ram_cell[0:4095][...	0000000...	Array
[0][31:0]	00000000	Array
[1][31:0]	00000001	Array
[2][31:0]	00000002	Array
[3][31:0]	00000003	Array
[4][31:0]	00000004	Array
[5][31:0]	00000005	Array
[6][31:0]	00000006	Array
[7][31:0]	00000007	Array
[8][31:0]	00000008	Array
[9][31:0]	00000009	Array
[10][31:0]	0000000a	Array
[11][31:0]	0000000b	Array
[12][31:0]	0000000c	Array
[13][31:0]	0000000d	Array
[14][31:0]	0000000e	Array
[15][31:0]	0000000f	Array
[16][31:0]	00000010	Array
[17][31:0]	00000011	Array
[18][31:0]	00000012	Array
[19][31:0]	00000013	Array
[20][31:0]	00000014	Array
[21][31:0]	00000015	Array
[22][31:0]	00000016	Array
[23][31:0]	00000017	Array
[24][31:0]	00000018	Array

说明运行结果是正确的。

阶段三：性能分析

使用我们提供的快速排序和矩阵乘法的benchmark进行实验，鼓励自己编写更多的汇编benchmark进行测试，体会cache size、组相连度、替换策略针对不同程序的优化效果，以及策略改变带来的电路面积的变化。针对不同程序，权衡性能和电路面积给出一个较优的cache参数和策略。其中“性能”参数使用运行仿真时的时钟周期数量进行评估。“资源占用”参数使用vivado或其它综合工具给出的综合报告进行评估。进行这一步时需要用阶段一的结果进行一些实验，不能仅仅进行理论分析，实验报告中需要给出实验结果（例如仿真波形的截图、vivado综合报告等）。
提示：为了方便进行性能评估，建议用上阶段二的缺失率统计功能

组相连度和缓存大小对性能影响

用332代表：LINE_ADDR_LEN = 3 ,SET_ADDR_LEN = 3 ,WAY_CNT = 2;以此类推。每次实验的时钟周期长度相等，因此在此次用运行时间代替时钟周期数。

注：实验的结果图保存在images文件夹中，命名：QF332.png，代表快排FIFO策略在332条件下的数据。

有的排序结果完全相同，只留一份截图。比如快速排序的434----43 10条件下FIFO策略的结果一致，等等。

1. 快速排序

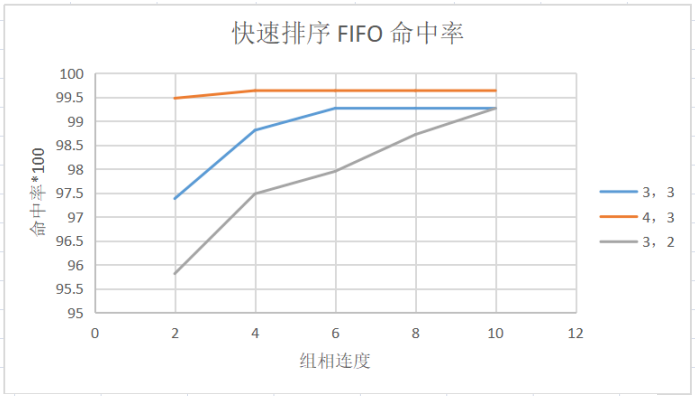
FIFO策略

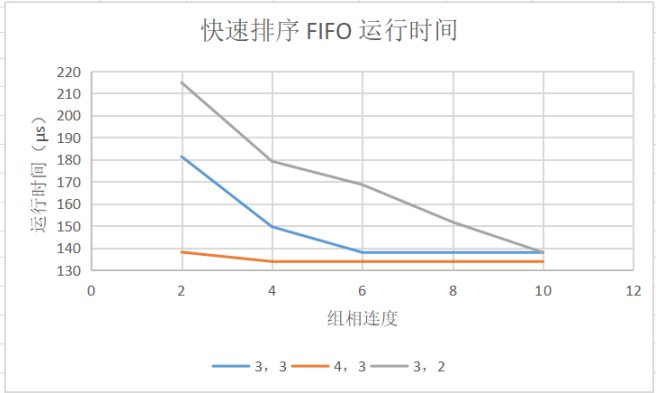
条件	hit	miss	HIT rate / %	RUN time / μs
332	*5095*	*137*	*97.38*	*181.324*
334	*5170*	*62*	*98.81*	*149.532*
336	*5194*	*38*	*99.27*	*137.9*
338	*5194*	*38*	*99.27*	*137.9*
33 10	*5194*	*38*	*99.27*	*137.9*
432	**5205**	**27**	**99.48**	**138.02**
434	**5213**	**19**	**99.64**	**133.796**
436	**5213**	**19**	**99.64**	**133.796**
438	**5213**	**19**	**99.64**	**133.796**
43 10	**5213**	**19**	**99.64**	**133.796**
322	5013	219	95.81	214.836
324	5100	132	97.48	179.2
326	5125	107	97.95	168.604
328	5165	67	98.72	151.648
32 10	5194	38	99.27	137.9

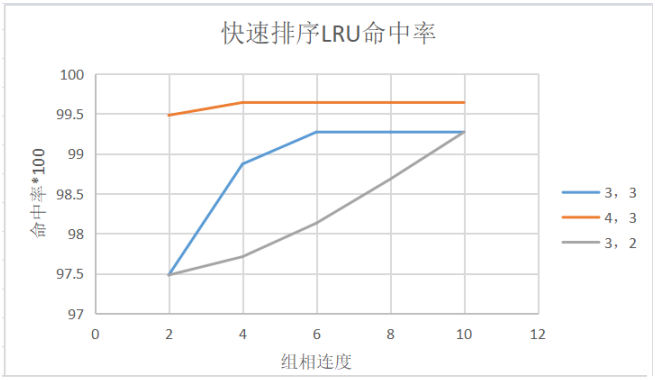
LRU策略

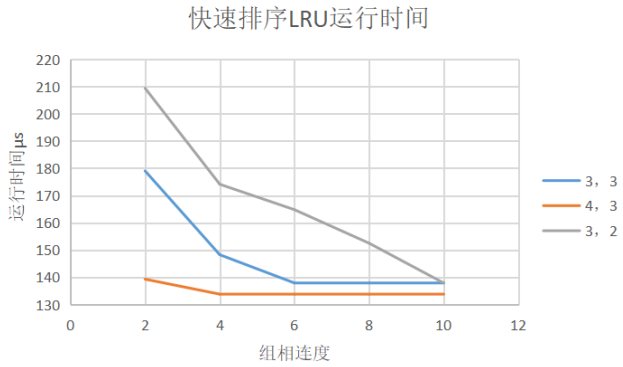
条件	hit	miss	HIT rate	RUN time / μs
332	*5100*	*132*	*97.48*	*178.992*
334	*5173*	*59*	*98.87*	*148.26*
336	*5194*	*38*	*99.27*	*137.9*
338	*5194*	*38*	*99.27*	*137.9*
33 10	*5194*	*38*	*99.27*	*137.9*
432	**5205**	**27**	**99.48**	**139.292**
434	**5213**	**19**	**99.64**	**133.796**
436	**5213**	**19**	**99.64**	**133.796**
438	**5213**	**19**	**99.64**	**133.796**
43 10	**5213**	**19**	**99.64**	**133.796**
322	5026	206	99.50	209.328
324	5112	120	97.71	174.112
326	5134	98	98.13	164.788
328	5163	69	98.68	152.496
32 10	5194	38	99.27	137.9

分析:









---

2. 矩阵乘法

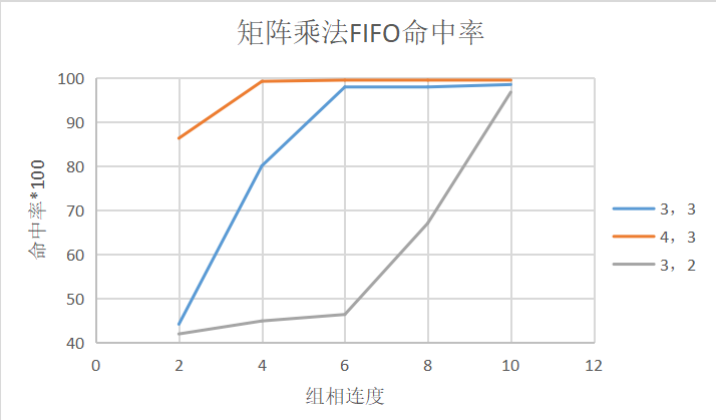
FIFO策略

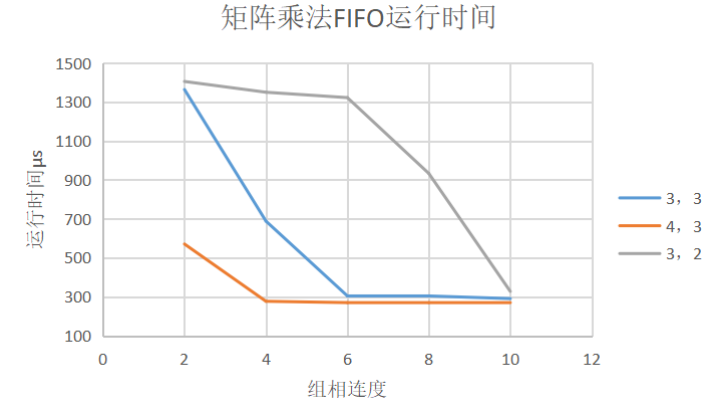
条件	hit	miss	HIT rate / %	RUN time / μs
332	*3840*	*4864*	44.12	1363.112
334	*6965*	*1739*	80.02	687.904
336	*8520*	*184*	97.89	304.032
338	*8520*	*184*	97.89	304.032
33 10	8568	136	98.44	290.336
432	**7509**	**1195**	86.27	570.336
434	**8632**	**72**	99.17	**276.512**
436	8656	48	99.45	269.672
438	8656	48	99.45	269.672
43 10	8656	48	99.45	269.672
322	3648	5056	41.91	1404.584
324	3904	4800	44.85	1349.288
326	4032	4672	46.32	1321.64
328	5833	2871	67.02	932.416
32 10	8418	286	96.71	327.312

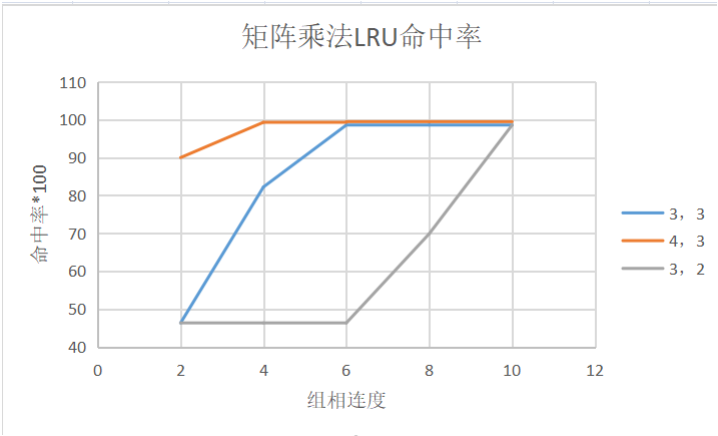
LRU策略

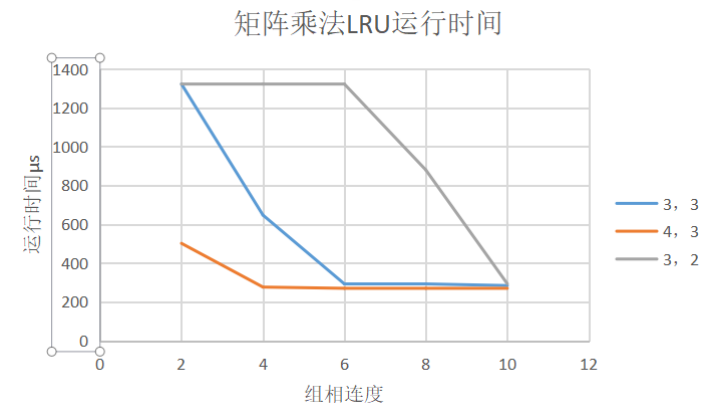
条件	hit	miss	HIT rate / %	RUN time / μs
332	*4032*	*4672*	46.32	1321.64
334	*7156*	*1548*	82.21	646.856
336	*8579*	*125*	98.56	292.332
338	*8579*	*125*	98.56	*292.332*
33 10	8599	*105*	98.79	283.852
432	**7830**	**874**	89.96	**501.208**
434	8641	**63**	99.28	276.024
436	**8656**	**48**	99.45	269.672
438	**8656**	**48**	99.45	269.672
43 10	**8656**	**48**	99.45	269.672
322	*4032*	*4672*	46.32	1321.64
324	*4032*	*4672*	46.32	1321.64
326	*4032*	*4672*	46.32	1321.64
328	6080	2624	69.85	879.272
32 10	8577	127	98.54	293.18

分析:









综上，利用控制变量的方法，可以看出**不论对于何种算法**，不管是什么策略，都有以下规律：

1. 组相连度相同的情况下，采用更大的cache 大部分时候可以提升性能，这在相连度较小时更加明显；
2. cache组和字大小相同的情况下，采取更大的组相连度在大部分时候可以提升性能；
3. 当cache大小和组相连度达到一定程度后，继续增加组相连度或者cache大小对性能的提升微乎其微。这是因为在算法刚开始运行时，缺失是不可避免的，所以继续加大cache已经不能减少这些不可避免的缺失了。
4. 访存时间和缺失率是大致成比例的，但在缺失率较低时，总时间并不是特别低，因为当缺失率较低时，计算的时间在总时间中所占比例会比较高。

对于**快速排序算法**，LRU 策略在有时会比 FIFO 性能差，这是因为快速排序算法每次运行时只考虑数组的前一半，处理完后再来考虑后一半，空间局部性较好，FIFO 策略的替换方法可能比 LRU 更适合快速排序。

对于**矩阵相乘算法**，空间局部性较差（一次读入一行、一列，算完后全部换出），可以看见当line addr len增大时，命中率和运行时间都大幅提升，这很可能是因为矩阵读入的一行内容在空间上相邻，那么在读入cache时也是相邻的，运算时读入一行的第一个数就可能把一整行都换入，所以对命中率影响很大。在达到最大值后继续提升参数带不来命中率和性能的明显提升，这是因为 cache 已经足够大到能装入几乎整个矩阵的内容，再提升也不会影响第一次访问元素时的强制缺失，所以当cache不够大时，不管什么替换策略都无法对命中率等产生提升。这是这个算法的特殊性导致的。

总结：不管何种策略，cache的set addr len=3，line addr len=3，way cnt = 6 时，性能都已经足够好。小于这个参数需要较大的组相连度，而大于这个参数对于性能影响又不大。

对电路面积影响

用（3，3，6，2）代表：LINE_ADDR_LEN = 3 ,SET_ADDR_LEN = 3, TAG_ADDR_LEN = 6 ,WAY_CNT = 2;以此类推。

FIFO策略：

1. (3, 3, 6, 6) :

Resource	Utilization	Available	Utilization %
LUT	6208	63400	9.79
FF	13981	126800	11.03
BRAM	4	135	2.96
IO	81	210	38.57

2. (3, 3, 6, 8) :

Resource	Utilization	Available	Utilization %
LUT	7744	63400	12.21
FF	18440	126800	14.54
BRAM	4	135	2.96
IO	81	210	38.57

3. (3, 4, 5, 8) :

Resource	Utilization	Available	Utilization %
LUT	13737	63400	21.67
FF	35772	126800	28.21
BRAM	4	135	2.96
IO	81	210	38.57

4. (4, 3, 5, 8) :

Resource	Utilization	Available	Utilization %
LUT	13887	63400	21.90
FF	35542	126800	28.03
BRAM	4	135	2.96
IO	81	210	38.57

LRU策略:

1. (3, 3, 6, 6) :

Resource	Utilization	Available	Utilization %
LUT	8310	63400	13.11
FF	15176	126800	11.97
BRAM	4	135	2.96
IO	81	210	38.57

2. (3, 3, 6, 8) :

Resource	Utilization	Available	Utilization %
LUT	10805	63400	17.04
FF	19913	126800	15.70
BRAM	4	135	2.96
IO	81	210	38.57

3. (3, 4, 5, 8) :

Resource	Utilization	Available	Utilization %
LUT	26148	63400	41.24
FF	38863	126800	30.65
BRAM	4	135	2.96
IO	81	210	38.57

4. （4，3，5，8）：

Resource	Utilization	Available	Utilization %
LUT	18261	63400	28.80
FF	37035	126800	29.21
BRAM	4	135	2.96
IO	81	210	38.57

综上，利用控制变量的方法，可以看出不管是什么策略，都有以下规律：

1. 组相连度增大时，电路面积也在增大；

2. 增大块大小会导致电路面积明显增加；

3. 增加组数会导致电路面积显著增加，特别是在组相连度较大时影响更大；

4. 同等条件下，LRU所用资源都多于FIFO策略，说明LRU策略比FIFO更复杂带来了相应的电路面积代价。

5. 在组大小为3和4的情况下LUT的使用情况接近，说明组大小为3时存在着LUT资源的不充分利用。

综合评价结论

综合性能表现和电路资源利用，可以得出下列结论。

1. 对于快速排序， FIFO 策略，SET_ADDR_LEN = LINE_ADDR_LEN = 3，WAY_CNT = 6应当是最好的选择。
2. 对于矩阵乘法， LRU 策略（同等大小下性能稍好），SET_ADDR_LEN = 2或3， LINE_ADDR_LEN = 3，WAY_CNT = 6应当是最好的选择。

综合上述结论，应该选择 LRU 策略，SET_ADDR_LEN = 3， LINE_ADDR_LEN = 3，WAY_CNT = 6。这样的cache通常在同等大小下能提供更好的性能，同时不会占用大量的硬件资源，是性能和电路面积的平衡之选。

收获和建议

花费时间：思考cache逻辑并打草稿2h，编程1h，debug 约2天（主要是所给CPU的自带bug）。

收获：

1. 对于 Verilog 和 system Verilog有了更加深入的了解，明白了一些编程的技巧。
2. 对于cache和CPU的工作有了更深的理解，明白了cache在现实条件下访存时间长时带来的性能提升。
3. 明白了工作策略各有千秋，在不同情况下应当灵活使用，没有绝对的优胜或者劣等。

建议：

1. 生成的文件和前一个实验的模块不能对上，重新连接电路非常麻烦，并且花费了本该使用在cache或者其他地方的时间。
2. 所给CPU文件有很严重的而且难以排查的 BUG ，导致debug时间很长，而且也没有进行说明。
3. 实验文档没有更新，有错误之处，并不够完善。