体系结构LAB3实验报告

JL19110004 徐语林

一.实验目标

1.权衡cache size增大带来的命中率提升效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

二.实验环境和工具

语言：verilog/systemverilog

环境：vivado/windows操作系统

三.实验内容，过程及结果

实验内容及过程：

由于该实验报告的重点侧重于对于cache的一个分析和性能的评估，因此此部分的内容仅仅做一个简述。

阶段一是在从一个简单的直接映射，写回带写分配的cache代码，进一步改成一个N路组相连的代码。替换策略则是使用FIFO以及LRU两种。下面分别叙述：

FIFO:此策略是一个先进先出策略，由于最开始每组的cache访问都是缺失的，所以按照顺序换入；此后便替换出最先进入的块。这里是维护了一个buffer的头指针，它代表了此刻应该被换出的块。最主要的代码部分是在状态机的SWAP_IN_OK状态加入以下代码：

```
//对buffer进行处理
                        if(buffer[mem_rd_set_addr]<WAY_CNT)
                        begin
                            buffer[mem_rd_set_addr]<=buffer[mem_rd_set_addr]+1;
                        end
                        else
                        begin
                            buffer[mem_rd_set_addr]<=0;
                        end
```

LRU：此策略是一个最近最少使用策略，对于此策略维护了以下变量：

```
reg [31:0] used [SET_SIZE][WAY_CNT]; //为每一路都维护了一个used的计数值来实现LRU(new)
reg [4:0] swap_addr=0;//要换的块(new)
reg [31:0] max=0;
```

当cache命中的时候，命中的块的used值清零，而未命中的块的used值便增加1，在最后选择used最大的作为要替换的块即可，主要的代码是在状态机的IDLE以及SWAP_IN_OK进行改进，此外还加了求取最大值的块的地址的部分，如下：

```
always@(*)
begin
    max=used[set_addr][1];
    swap_addr=1;
    for(integer i=0;i<WAY_CNT;i=i+1)
    begin
        if(max<used[set_addr][i])
        begin
            max=used[set_addr][i];
            swap_addr=i;
        end
    end
end//做LRU处理
```

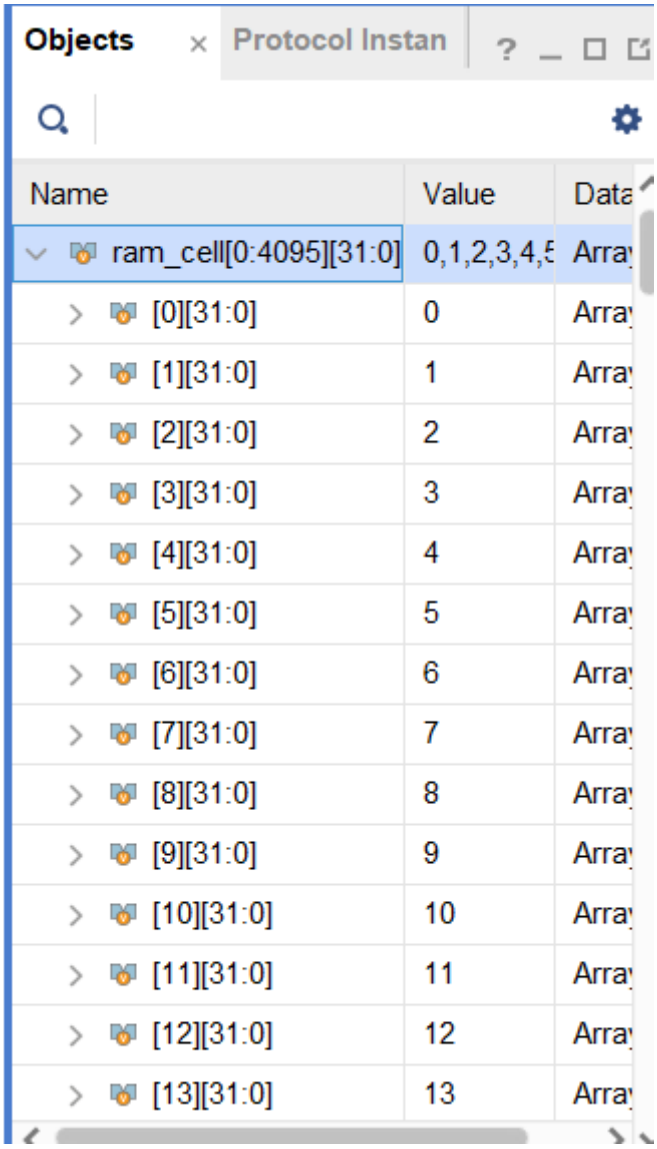
阶段二便是在实验二的cpu上用编写的组相连的cache来顶替原来的dataram，实现快速排序以及矩阵相乘算法。

注：此处踩了坑，在顶替过程中，ID和WB段段寄存器的RD_old值应在~stall_ff的条件下才可以进行更新。

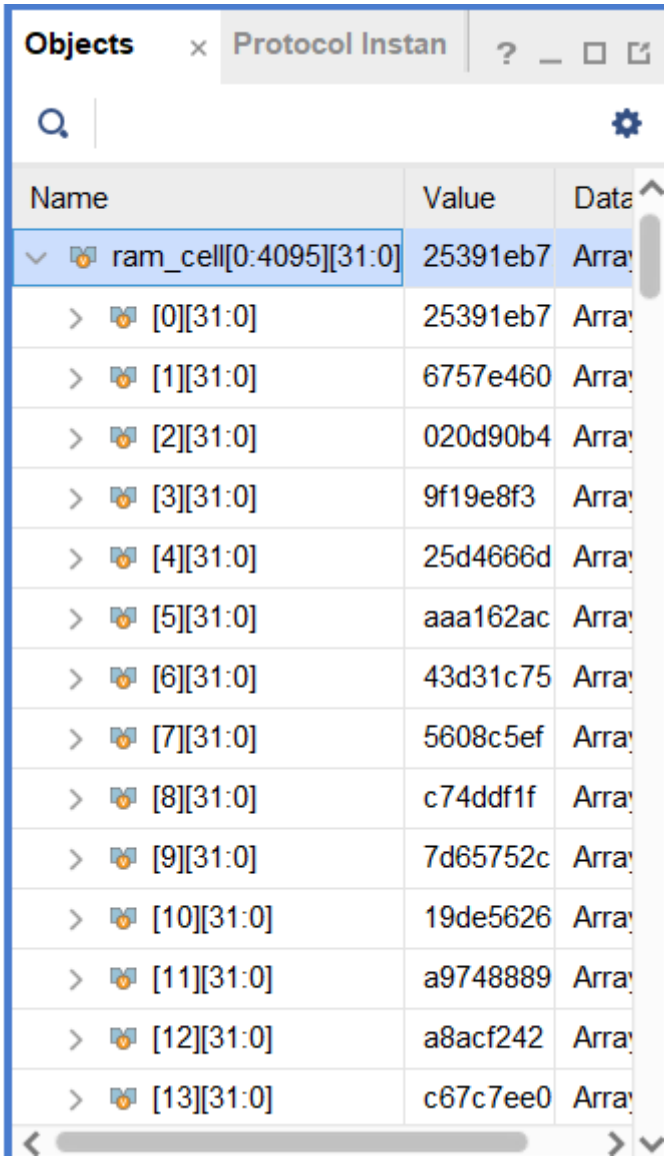
######

结果部分（在此仅展示阶段二的结果）：

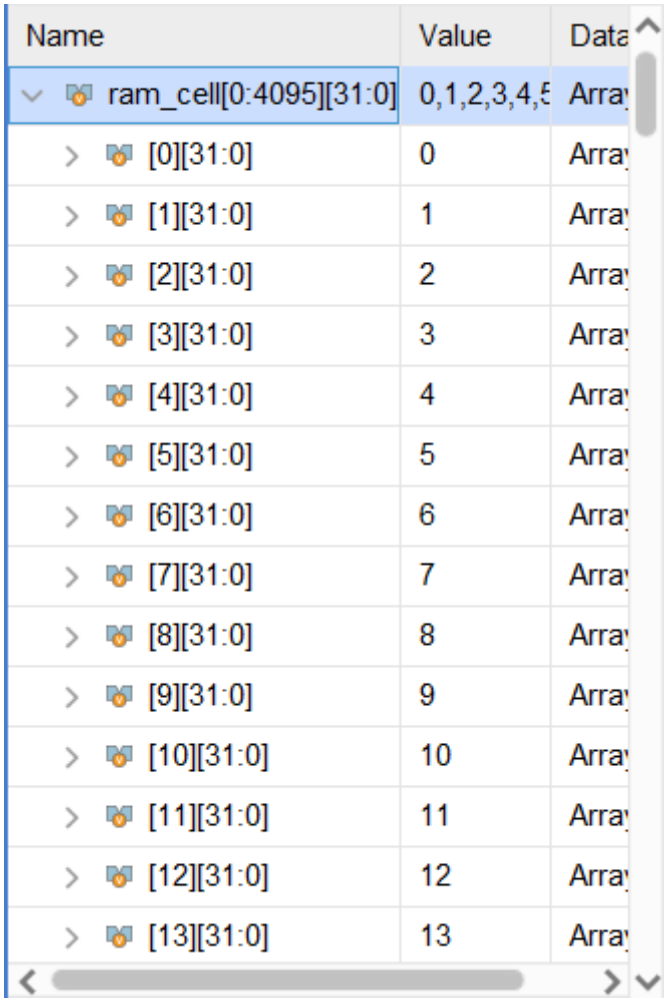
FIFO策略的quicksort:



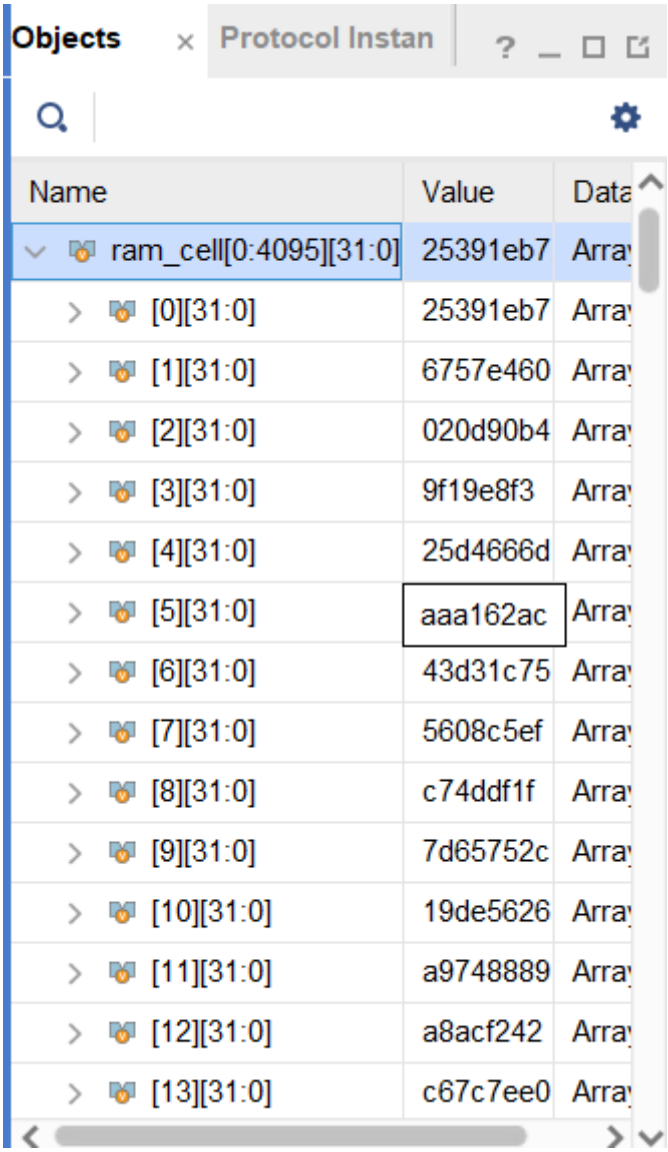
FIFO策略的Matrix:



LRU策略的quicksort：



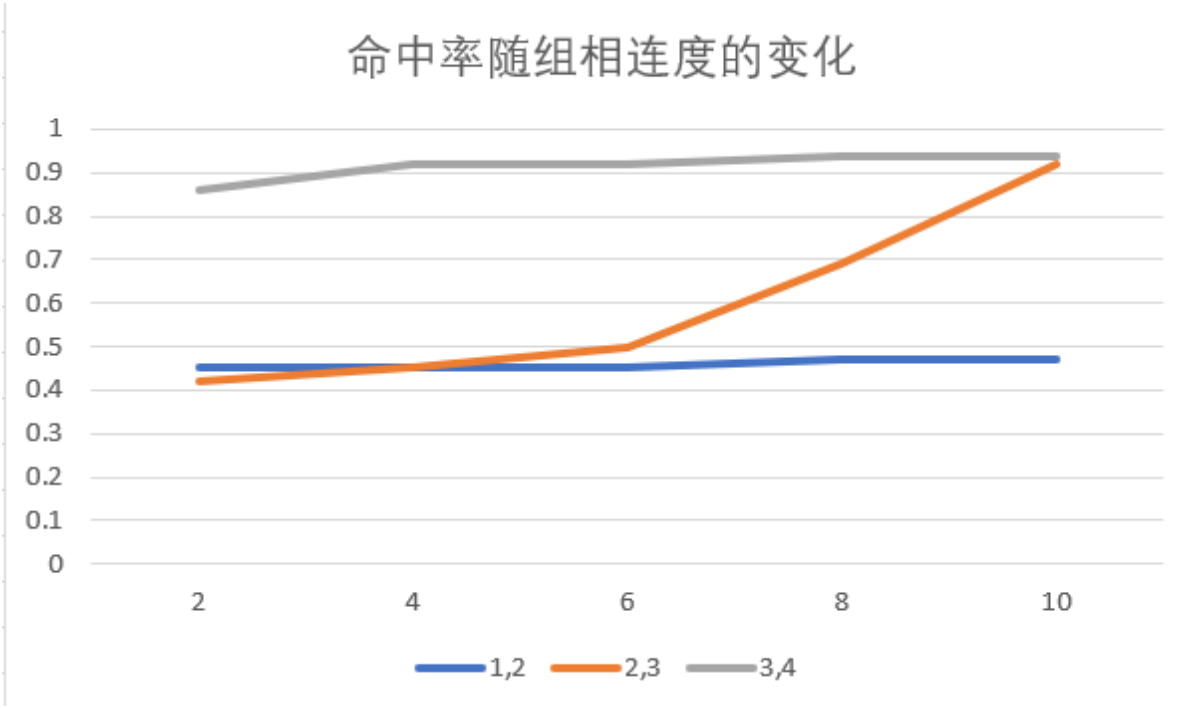
LRU策略的Matrix:



四.Cache的性能分析

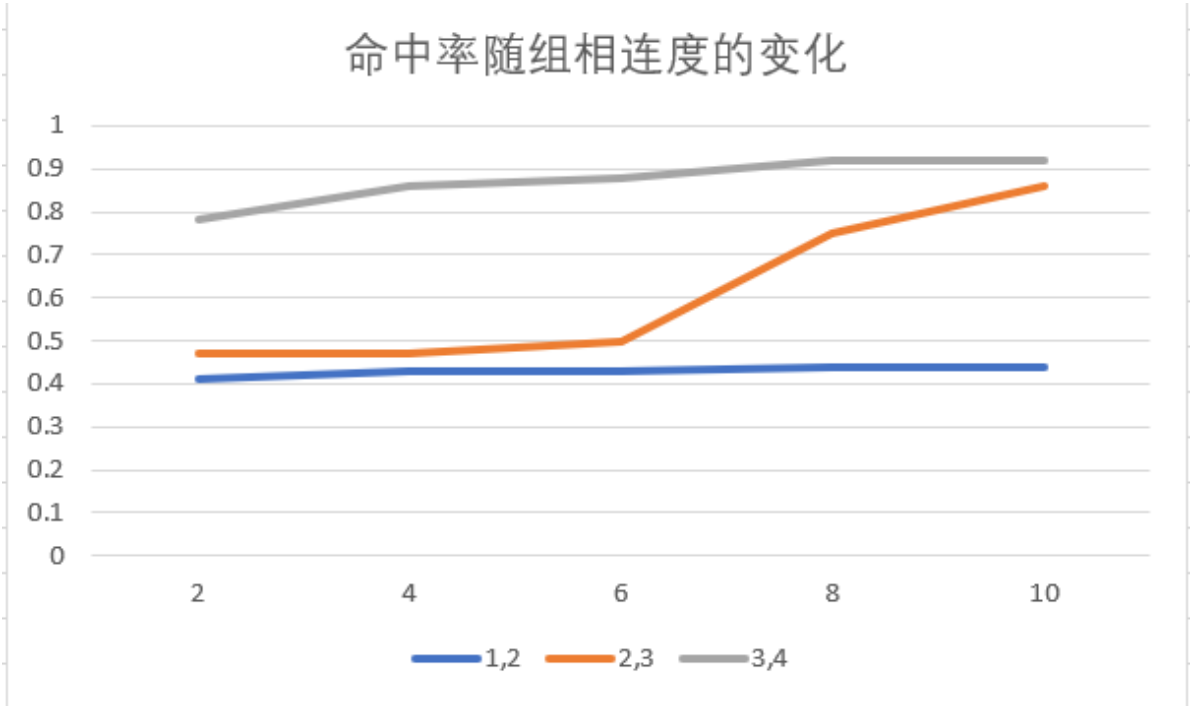
1.命中率(以矩阵算法为例):

FIFO替换法则下:



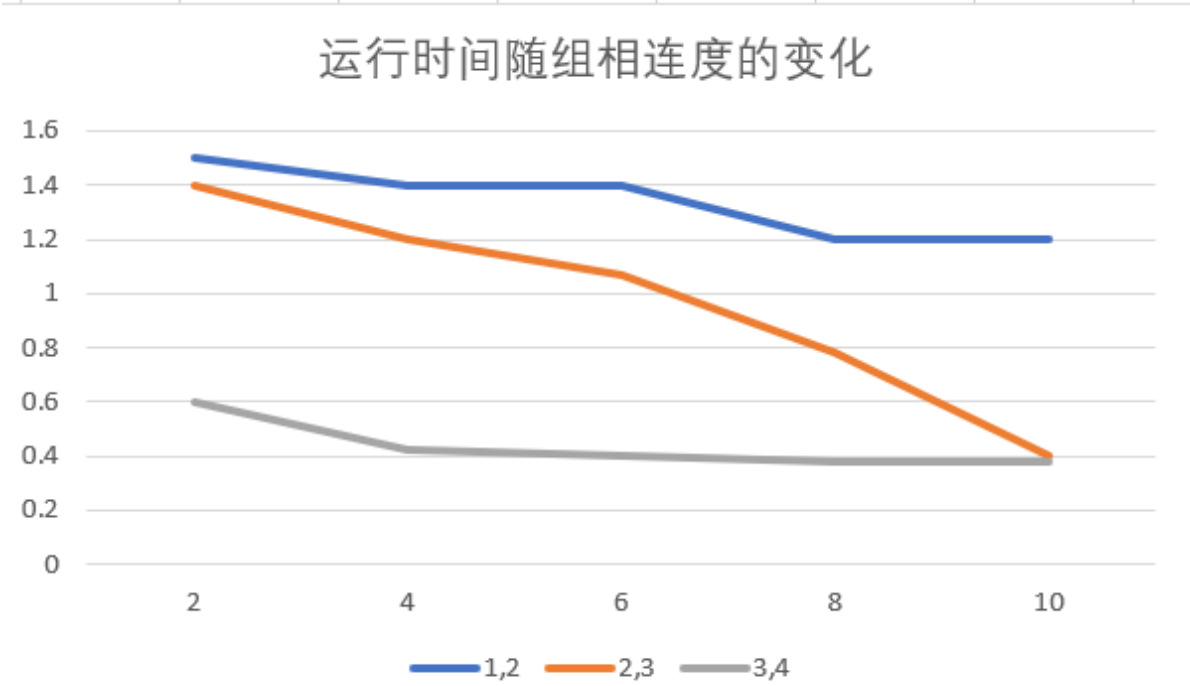
注:横坐标代表组相连度，纵坐标代表命中率。取定了几组组地址长度以及line内地址长度.前者表示组地址长度，后者表示line内地址长度

LRU替换法则下:



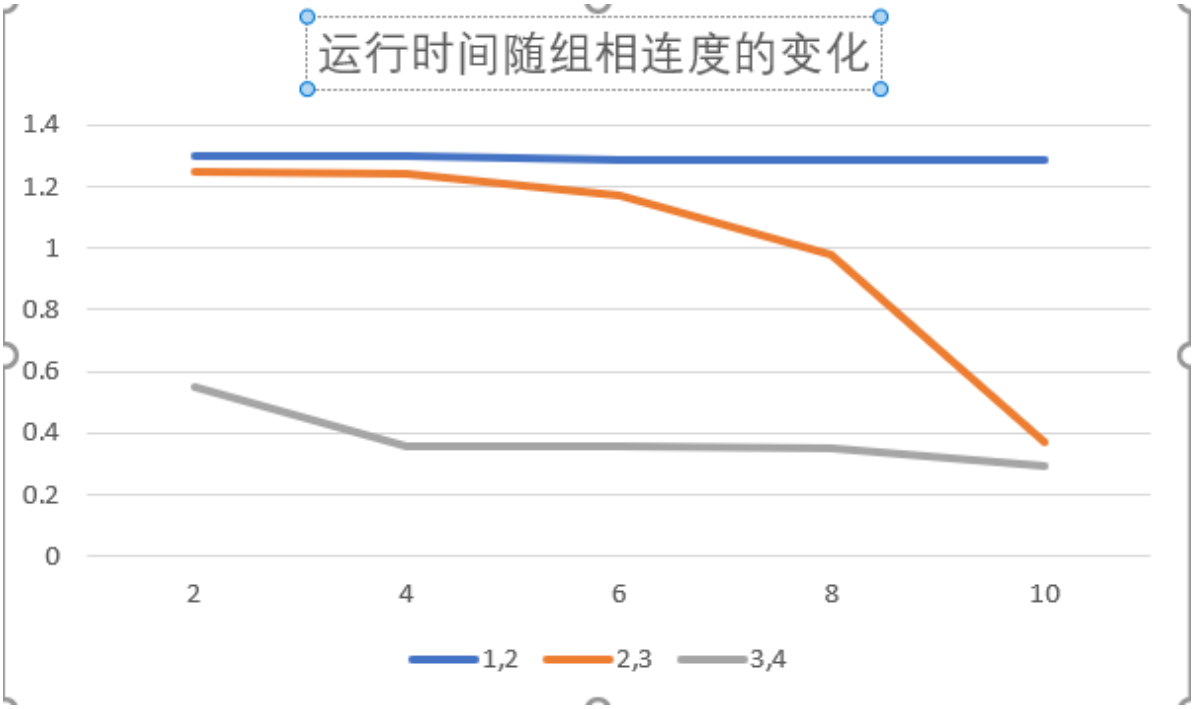
2.运行时间(以矩阵算法为例):

FIFO替换法则下:



注：纵坐标为运行时间，单位为ms;横坐标为组相连度。其余同上图.

LRU替换法则下：



由此可以看出，在给定了组地址长度及line内地址长度时，相当于cache的大小相同，但此时随着组相连度的增加运行时间在减小，命中率大致在增加，所反映出来的性能也在变好；纵向对比组地址长度为3，line内地址长度为4时的cache的性能要好于组地址长度为2，line内地址长度为3的cache,组地址长度为2,line内地址长度为3的cache的性能要好于组地址长度为1，line内地址长度为2的cache；而对于替换策略而言，LRU替换策略无论是从运行时间还是命中率都要比FIFO替换策略好。

3.Cache大小对于电路面积的影响：

使用的工具是vivado中的综合。

注：(x,y,z)中x表示组地址的长度，y表示line内地址的长度，z表示组相连度。

FIFO：

(1).(1,2,2)型：

Resource	Utilization	Available	Utilization %
LUT	638	63400	1.01
FF	1103	126800	0.87
BRAM	0.50	135	0.37
IO	78	210	37.14

(2).(2,3,2)型：

Resource	Utilization	Available	Utilization %
LUT	1922	63400	3.03
FF	3062	126800	2.41
BRAM	2	135	1.48
IO	80	210	38.10

(3).(3,4,2)型：

Resource	Utilization	Available	Utilization %
LUT	3794	63400	5.98
FF	10054	126800	7.93
BRAM	8	135	5.93
IO	82	210	39.05

LRU：

(1).(1,2,2)型：

Resource	Utilization	Available	Utilization %
LUT	867	63400	1.37
FF	1223	126800	0.96
BRAM	0.50	135	0.37
IO	78	210	37.14

(2).(2,3,2)型：

Resource	Utilization	Available	Utilization %
LUT	2297	63400	3.62
FF	3311	126800	2.61
BRAM	2	135	1.48
IO	80	210	38.10

(3).(3,4,2)型：

Resource	Utilization	Available	Utilization %
LUT	4072	63400	6.42
FF	10557	126800	8.33
BRAM	8	135	5.93
IO	82	210	39.05

综上所述，随着cache的增大，LUT和FF都在增大。而对比两种策略来看，LRU使用的资源更多。

组相连度对于电路面积的影响：

注：此处就只选用LRU策略的Cache进行综合。

(1).(3,4,2)型:

Resource	Utilization	Available	Utilization %
LUT	4072	63400	6.42
FF	10557	126800	8.33
BRAM	8	135	5.93
IO	82	210	39.05

(2).(3,4,4)型:

Resource	Utilization	Available	Utilization %
LUT	9778	63400	15.42
FF	19410	126800	15.31
BRAM	8	135	5.93
IO	82	210	39.05

(3).(3,4,6)型：

Resource	Utilization	Available	Utilization %
LUT	13162	63400	20.76
FF	28250	126800	22.28
BRAM	8	135	5.93
IO	82	210	39.05

由此可见，随着路数的减少，需要维护的信息在减少，故LUT的资源在减少。

实验结论：

综上所述，选择(3,4,6)型的cache在命中率和运行时间上具有不错的数据，同时在电路资源上消耗的也不算太大；同时对于LRU来说，命中率和运行时间都要好于FIFO，但是LRU需要的硬件资源要多于FIFO，因此在Cache比较小的时候，应该选择LRU,因为此时的cache在电路资源以及性能上都有比较好的数据，但是当cache变大的时候，便要看情况来进行权衡。

五.实验总结及建议

本次实验通过对cache的实现，使我更加深刻的理解了组相连的cache，以及顺带回顾了状态机的写法。此次实验更重要的还是实验完成后对cache进行的性能分析，从中可以更加真切的体会到不同参数的影响下的cache性能的变化。

建议：没啥，挺好。