

Verilog-lab4 实验报告

梁峻滔 PB19051175

Verilog-lab4 实验报告

实验目的

实验内容

阶段一: BTB实现

核心部件

功能实现逻辑

阶段二

BHT实现

核心部件

功能实现逻辑

Branch Prediction Unit

数据分析

1. 分析分支收益和分支代价
2. 统计未使用分支预测和使用分支预测的总周期数及差值
3. 统计分支指令数目、动态分支预测正确次数和错误次数
4. 对比不同策略并分析以上几点的关系

踩坑总结

实验目的

- 实现 BTB(Branch Target Buffer) 和 BHT(Branch History Table) 两种动态分支预测器
- 体会动态分支预测对流水线性能的影响

实验内容

阶段一: BTB实现

核心部件

模块内部的核心部件是一个目标地址缓存, 缓存使用指令的低位地址作为索引, 一个表项包括: 指令的高位地址、跳转目标地址、valid. 表项中各个字段的作用是:

- 指令高位地址: 缓存使用指令的低位地址作为索引, 但可能会有多条指令具有相同的低位地址, 因此需要比较高位地址确认表项对应的指令
- 跳转目标地址: 每条分支指令都会有其跳转目标地址
- valid: 标记该表项是否有效

功能实现逻辑

该模块的功能是:

1. 预测: 传入一条 IF 段指令的 PC 值, 判断该指令是否在BTB表中有缓存项, 返回是否命中结果和命中时目标地址
2. 修正: 传入一条 EX 段指令的 PC 值和实际跳转结果, 判断该指令在 IF 段的预测结果是否正确, 如果预测错误, 则需要向外传递预测错误信号, 以及修改相应表项的内容

具体逻辑:

1. 预测: 直接根据 IF 段指令的 PC 值低位查询表项, 若高位地址匹配且valid有效, 即为命中, 否则为不命中; 目标地址直接返回表项中的目标地址项
2. 修正:
 - 修改表项:
 - EX段判断实际需要跳转时, 将EX段的指令PC高位、计算出来的目标地址存入表项, valid置为1
 - EX段判断实际不需要跳转, 将相应表项valid置为0
 - 当实际跳转与预测跳转不一致时, 输出预测错误信号, 可以采用2-bit信号表示预测错误类型, 帮助CPU选择正确的下一个PC, 预测错误类型编码为

类型	编码
预测跳转, 实际不跳转	11
预测不跳转, 实际跳转	10
预测不跳转, 实际不跳转	01
预测跳转, 实际跳转	00

其中当1号位为1时表示预测出错了.

阶段二

BHT实现

核心部件

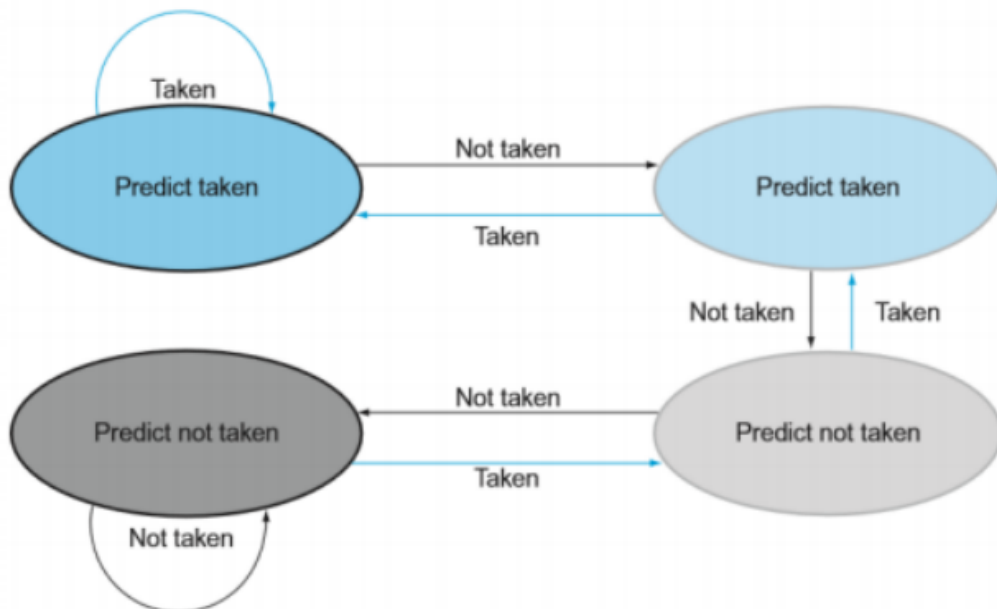
模块内部的核心部件是一个预测状态(status)表, 使用指令地址的低位作为索引, 表项为2bit, 表示该指令的预测状态:

预测状态	含义
00	STRONGLY_NOT_TAKEN, 取到该分支指令时预测不跳转
01	WEAKLY_NOT_TAKEN, 取到该分支指令时预测不跳转
10	WEAKLY_TAKEN, 取到该分支指令时预测跳转
11	STRONGLY_TAKEN, 取到该分支指令时预测跳转

功能实现逻辑

功能: 根据 IF 段的 PC 值输出预测是否跳转, 根据 EX 段的 PC 值和实际跳转结果更新状态机.

状态转移图为:



Branch Prediction Unit

使用一个顶层模块整合两种预测器. 增加统计分支指令总数和预测错误次数的电路.

分别例化BTB和BHT, 当仅使用BTB时, 使用BTB的命中结果作为预测是否跳转的输出信号; 当使用BHT时, 使用BHT的输出预测结果作为最后的预测结果. 当使用BHT的预测结果时, 使用BTB中不考虑valid的命中.

数据分析

1. 分析分支收益和分支代价

对于一个执行 N 次循环的循环体:

- 对于静态预测不跳转, 前面 $N-1$ 次都预测错误, 每次需要2个周期的代价, 共 $2(N-1)$ 个周期的代价
- 对于静态预测跳转, 前面 $N-1$ 次都预测正确, 最后一次预测错误, 共2个周期的代价
- 对于没有BHT、仅使用BTB的动态预测, 第一次和最后一次预测错误, 共4个周期的代价
- 对于带有2-bit BHT的动态预测, 前两次和最后一次预测错误, 共6个周期的代价

对于一些实际跳转情况经常变化的情形, 例如

```

while (...) {
    i = 2;
    if (i % 2 == 0) {
        ...
    }
    i++;
}
  
```













对于 if 是否跳转, 静态预测不跳转/跳转都有一半是预测错误的, 而没有BHT的动态预测则是全部预测错误, 带有2-bit BHT的动态分支预测则是一半预测错误, 一半预测正确.

而对于实际跳转情况稳定的分支, 静态分支预测的预测错误次数一般会更少(因为没有"预热"过程), 但是静态分支预测是一个二选一碰运气的策略, 如果实际跳转稳定在"不跳转", 那静态预测跳转就会导致很多预测错误. 而且"跳转情况稳定"是时间局部的, 实际应用中存在很多分支, 某一段时间内稳定不跳转, 而另一端时间内稳定跳转, 这种情况下采用无论采用哪种静态预测都会导致一半的预测错误, 但是动态分支预测就可以适应性地改变跳转预测状态以显著减少预测错误.

整体而言, 对于实际跳转变化稳定但稳定状态有时间局部性的情形, 动态分支预测的效果更好. 而在实际跳转变化频繁时, 带BHT的动态预测的效果比没有BHT的动态预测更稳定.

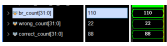
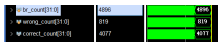
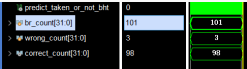
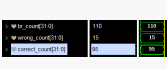
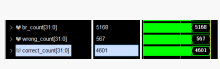
2. 统计未使用分支预测和使用分支预测的总周期数及差值

一个时钟周期持续4ns, 周期数就用执行完最后一条指令的时间(对于btb.s和bht.s)和第一次执行死循环jal指令的时间(对于QuickSort和MatMul)除以4获得. 没有预测的数据使用 Lab3 的工程来获取.

预测器	btb.s	btb 周期 数	bht.s	bht 周期 数	QuickSort	QuickSort 周期数	MatMul	MatMul 周期数
静态预测 不跳转		508		534		128741		1460485
BTB		313		381		160057		1543032
BTB+BHT		315		385		138753		1543038
差值(动 态-静态)		-195, -193		-153, -149		31316, 10012		82547, 82553

3. 统计分支指令数目、动态分支预测正确次数和错误次数

仿真波形

预测器	btb.s	bht.s	QuickSort	MatMul
BTB				
BTB+BHT				

- br_count: 分支指令总数
- wrong_count: 预测错误次数
- correct_count: 预测正确次数

数据

预测器	数据顺序	btb.s	bht.s	QuickSort	MatMul
BTB	br_count, wrong_count, correct_count, 错误率	101, 2, 99, 1.98%	110, 22, 88, 20.00%	48442, 9624, 38818, 19.87%	4896, 819, 4077, 16.73%
BTB+BHT	br_count, wrong_count, correct_count, 错误率	101, 3, 98, 2.97%	110, 15, 95, 13.64%	35506, 5522, 29984, 15.55%	5168, 567, 4601, 10.97%

4. 对比不同策略并分析以上几点的关系

- 从程序运行周期上看, 对于不同的程序, 动态分支预测的效果不一定比静态预测的效果好, 而且在复杂程序(QuickSort和MatMul)中动态分支预测的效果反而比静态预测的要差, 这与直观感觉相悖. 而动态分支预测中, 仅有BTB的预测器与带有BHT的预测器相差不大.
- 从预测正确和错误次数来看, 对于分支复杂的程序, 带有BHT的预测器的错误率比仅有BTB的预测器的错误率低. 这一点与我们的理论分析一致.

结论:

数据结果并不与理论分析一致, 理论分析中采用动态分支预测对于复杂程序应该可以带来显著的性能提升, 但是数据结果却显示相反, 这可能有多方面原因(例如程序复杂度不够, 电路逻辑设计方面有问题等等, 但是观察仿真波形发现预测器的预测行为是正确的, 所以我也不明白真正的原因).

就预测准确率而言, 对于分支复杂的程序, 带有BHT的预测器的预测效果一般比仅有BTB的预测器好.

踩坑总结

忘了将is_taken_if往后传递

没有把predict_error_type传进hazard模块

`buffer_index` 位宽一开始设为1了

区分仅有BTB的预测器中的bit_hit(考虑valid)和带有BHT的预测器中的btb_hit(不考虑valid)