HW2: Branch Predictor Design

楊永琪, 109654020

***Abstract*—**本次作業為研究與實作 Two Level Branch 改良原先的 Bimodal Branch Predictor。

I. Introduction

這次作業為研究與使用 Two Level Branch 去改良原先的 Bimodal Branch Predictor。以下會使用原先 bimodal branch predictor 在 CoreMark 上針對不同 entry number 以 branch type 的 misprediction rate 進行分析, 接續會說明預計將要實作的 Two Level Branch Predictor 。

II. Analysis

A. Comparison between with and without branch predictor

由以下表格可以看出若沒有 branch preditor CoreMark 的分數會下降許多, 推測是因為他默認都會是 not taken, 而 taken 的總數占總執行的 branch 數約 57.74%, 會導致執行了許多不必要得指令。而因為原先 branch_inst_tag 的 值會跟真正需要比較的不同, 可能導致無法更新 bimodal counter , 更改後的 bimodal 比原先的分數稍微上升一點, hit 的個數相同、misprediction count 下降一點約 3.76%。

其中下表的 hit 指存在於 BHT , mispred 表示存在於 BHT 並 predict 錯誤, branch、hit 及 mispred 的算法則與 HW1 相似, 在 exe stage 後一同傳到 wbk stage 完再進行計算, 不考慮 stall 的情況。

TABLE I.

bpu	comparison with and w/o branch predictor				
	branch count	***hit count***	***mispred count***	***mispred rate***	***score***
w/o bpu	86915260	x	(taken) 50181199	x	74.259713
original	87096855	74619003	10533378	14.12%	82.433305
bimodal	87096854	74618994	7730919	10.36%	83.195714

B. Comparison Between BHT Entry

TABLE II.

BHT entry	comparison with and w/o branch predictor				
	hit	***hit rate***	***mispred***	***mispred rate***	***score***
32	74618994	85.67%	7730919	10.36%	82.195714
64	80144117	92.02%	8094798	10.10%	83.699079
128	83501200	95.87%	8392064	10.05%	84.078800
256	85517733	98.19%	8666957	10.13%	84.241876
512	86992865	99.88%	8680287	9.98%	84.458921
1024	87034153	99.93%	8685126	9.98%	84.461658

根據 Table II 會發現隨著 entry number 愈大, hit rate 愈高, score 也隨著慢慢增加, Misprediction rate 則是介於 10% 左右。

C. Different Types of Branches in CoreMark

TABLE III. Bimodal with 256 entries

function	comparison between different functions			
	hit count	***hit rate***	***mispredict count***	***mispredict rate***
core_list_find	14495788	99.98%	254098	1.75%
matrix_mul_matrix	4337849	99.58%	440440	10.15%
core_list_mergesort	3672939	98.54%	586914	15.98%
crcu8	11279746	99.76%	3570243	31.65%

在實際測試之前, 我先用 gprof 跑出來圖內的 function 於電腦上進行模擬 hit 與 misprediction rate, 發現大部分都介於 10% 左右或更少, 其中有少數的情況比較特別, 最後選擇四個 function: core_list_find, matrix_mul_matrix, core_list_merge_sort 以及 crcu8 在板子上進行分析。

其中由於 core_list_find 由 if else, 與 while 迴圈組成, 在模擬時發現他 if, else 不會交錯執行, 而 if, else 裡面的 while 迴圈按照 bimodal 的演算法只會 predict 錯一次, 因此 mispredict rate 較低為1.75%。

```
if(...) // find item by index
{ while() … }
else // find item by specific data value
{ while() … }
```

CODE I. pseudocode of core_list_find

matrix_mul_matrix 計算則為 3 層的 for 迴圈, 若為 [MxN] [NxK] 矩陣相乘, bimodal 在進行此類 branch 當 hit 時至少會有 MK+1 預測錯誤, 對於較多層的迴圈 bimodal 會因此累加錯誤的預測, misprediction rate 為 10.15%。

core_list_mergesort 會因為先將鄰近兩兩數值進行比對, 再以 2 的指數倍的 list 進行比對, 數值比對可能因為比較具有隨機性, 會比一般迴圈類的 branch 指令更難預測, 使用 bimodal 的 misprediction rate 約為 16%。

而 crcu8 function 的 misprediction rate 為 31.65%, 看其程式會發現在 for 迴圈裡面第二個 branch 其實與第一個 branch 會有關連, 如果 branch 1 執行的話, branch 2 則會執行, 但由於 bimodal 的做法無法根據 branch 1 是否有 taken

或 not taken 的歷史去做判斷, 不會考慮到 global 的 branch history, 因而較難準確預測。

```
for (i = 0; i < 8; i++)
{
    …
    if (x16 == 1) …, carry = 1; // branch 1
    else  carry = 0;
    …
    if (carry)  … // branch 2 depends on branch 1
    else …
}
```

CODE II. simplified code of crcu8

III. Two Level Branch Predictors

Two Level Branch Predictor 結合了 local branch predictor 與 global branch predictor 的優勢, 以下將說明由[1]所得知 local branch 與 global branch 的優劣以及未來將如何實現 two level branch predictor。

1. Local Branch Predictor

Local Branch Predictor 會需要兩個 table, 一個為特定 branch 的 Pattern History Table(PHT), 去記錄對應 pc_addr [n bits] 先前執行的歷史, 另一個為 Saturating Counter (bimodal), 當執行到 branch hit 時, 會先藉由 pc_addr [n bits] 去 PHT 尋找對應的 pattern [m bits], 再用此 pattern 作為 index 去 saturating counter 得到對應的 predict output。

此作法當 m 足夠大, 且不會與其他相同 pattern 的 branch 一同執行, 則可以準確的預估有固定 pattern 的 branch type, 如程式的 branch 指令依序為 [T, T, NT, T, T, NT], 若 m >= 2 , 則此 predictor 可以完整地預估下一個 branch 的 taken、not taken, 若用一般 bimodal, 則有三分之一的 branch 會預測錯誤。

其缺點為當遇到兩個一樣的 pattern 但 decision 相反的 branch 時, 若兩者交互進行, 則可能會因為共用同一個 counter entry 而互相干擾。

2. Global Branch Predictor

Global Branch Predictor 只考慮 global history pattern, 當執行到 branch 指令時會跟去先前各類 branch 的 taken, not taken history 當作 index 去 table 取得預測結果, 其壞處是當 table size 太小, 不同 branch 指令很有可能共用到同一個 table entry。

3. Two Level Branch Predictor

由於單純使用 global 準確率會比較低, 根據 [1] 在 table size 小的情況下效果會比 bimodal 還要差, 而 gselect 、gshare 的作法可以讓 branch predictor 在考慮 global history 的同時也考慮現在正在執行的 pc_address。

預計接下來將嘗試使用 gshare 的做法, 將 pc_addr [n bits] 與 branch history [n bits] 做 exclusive or 運算當作 index 於 bimodal table 取得預測結果, 同時也嘗試各種 predictor size, 看哪種效果比較好, 若還有額外的時間, 也想嘗試 Hybrid Branch Predictor 的作法。

4. Hybrid Branch Predictor

Hybrid Branch Predictor 結合了不同種 branch predictor。由於不同種 predictor 有各自的優劣, 結合不同 branch predictor 是希望能讓各類的 branch 都能用合適的 preditctor 去預測。在[1]中有提到結合 bimodal 與 gshare 的 combined branch predictor, 此種作法會需要用到 3 個 table, 一個為 bimodal counter、一個是 gshare table, 還有一個 2-bit staturating bimodal counter 用來判斷使用哪一個 predictor 的 output 比較好, 若在 exe stage 後兩邊預測結果不同則會去更新對應 selector counter 的值。

IV. Implementation

References

[1] McFarling, “Combining Branch Predictors,” WRL Technical Note TN-36, Digital Equipment Corporation, June 1993.