使用分類迴圈出口模型的分支預測
Branch Prediction with Classified Loop-Exit Models

李 聰　羅旭堂

Tsung Lee　Shiu-Tang Lo

國立中山大學電機工程學系副教授　國立中山大學電機工程學系

摘要

在本研究中，我們分析科學工程程式的迴圈行為，並找出使用傳統分支預測方法所有的分支預測失誤來源。在這些程式中，內迴圈巢的迴圈出口大多顯示一定的規律行為。取代使用傳統分支預測於迴圈出口分支，我們發展一個硬體迴圈出口分支偵測技術與一個用於迴圈出口分支的分支預測技術。此偵測方法動態的將分支分類為迴圈出口分支與非迴圈出口分支。對迴圈出口分支，我們發展一些吻合科學工程程式中規律迴圈出口行為的迴圈預測模型，此迴圈預測方法可以學習與適應選擇合適分類的迴圈出口行為的預測模型，以達到更準確的分支預測。在實驗裡，我們展示此分類迴圈出口分支預測可以有效改進對科學工程程式的分支預測命中率。

關鍵詞：分支預測、迴圈出口分支偵測、迴圈出口預測模型、迴圈行為、管線處理器。

ABSTRACT

In this research, we analyze loop behavior of scientific and engineering programs and identify sources of branch prediction misses with traditional branch prediction methods. In such programs, loop exits of inner loop nests mostly exhibit certain regular behaviors. Instead of using traditional branch prediction for these loop-exit branches, we developed a hardware loop-exit branch detection technique and a branch prediction method for loop-exit branches. The detection method dynamically classifies branches into loop-exit branches and non-loop-exit branches. For loop-exit branches, we developed a few branch prediction models matching corresponding regular loop-exit behaviors in scientific and engineering programs. The prediction method can learn and adaptively select suitable classified prediction model of loop-exit behaviors for more accurate branch prediction. In experiments, we show that the classified loop-exit branch prediction can enhance hit ratio of branch prediction for scientific and engineering programs effectively.

Keywords： branch prediction, loop-exit branch detection, loop-exit prediction model, loop behavior, pipelined processor

1. 動機與目的

隨著處理器速度不斷的增加，且使用管線(pipeline)架構的設計，處理器可以在同一時間內抓取(fetch)跟發出(issue)多個指令，分支預測(branch prediction)的命中率(hit ratio)(Hennessy & Patterson, 2012; Hwang & Naresh, 2010; Lilja, 1988)影響處理器是否能正確的提前抓取指令，及將正確的指令送入管線，如果預測失誤會造成指令管線內的無效氣泡(bubble)的形成，因此降低處理器的效能。

在科學與工程程式中，存在大量迴圈的執行，這些迴圈的執行常反映一些規律的方式。其控制流由分支指令控制，這些分支指令的具有某種規律的分支行為，其他非迴圈控制的分支指令具有非規律分支行為，在傳統的分支預測方法中，這兩類分支指令是由相同的方法做分支預測。我們理解傳統的分支預測方法並不能針對性的預測這些迴圈控制分支指令的規律分支行為，因此會造成一定的預測失誤。此為傳統分支預測方法的不足，本研究導向於此方面的改善設計。

過去分支預測採用的方法，依特性可分成動態與靜態兩種。靜態方法中最簡單的預測方式為預測跳躍結果永遠成立或永遠不成立(Lee & Smith, 1984)，此外亦可以由指令特性來做判斷，在(Smith, 1981)的做法中，遇到一個往回跳(backward jump)的指令就預測其永遠成立(taken)，往前跳(forward jump)的指令就預測其永遠不成立，這樣的預測方式對一個迴圈程式來說有著不錯的預測準確率，但對一般不規則跳躍的程式則效果不彰。為了因應不同種類程式的分支跳躍情形，(Lee & Smith, 1984)中提出了藉由累積過去執行分支指令的資訊，來判斷跳躍結果的動態預測機制，此方法使用一個跳躍目的暫存區(branch target buffer)累計執行資訊，每當執行一次

分支指令就動態改變暫存區的內容，並靠此資訊產生預測的結果。

而在(Yeh & Patt, 1991; Yeh & Patt, 1993; Chang, Hao, Yeh, & Patt, 1994)的做法中，更提出了一種雙層式適應性訓練分支預測(Two-Level Adaptive Train Branch Prediction)方法來做更有效的動態分支預測，此方法利用兩個資料結構：分支歷史暫存器(branch history register)跟分支歷史樣式表格(branch history pattern table)來儲存程式的分支資訊及做分支判斷。分支歷史暫存器是一個移位(shift)暫存器，分支指令執行後即把結果移位進來，分支歷史樣式表格中儲存的就是對應到不同歷史暫存器的跳躍執行結果。要做分支預測時即以目前指令執行時歷史暫存器的值為索引(index)，到分支歷史樣式表格中去讀取資料，並根據裡面的資訊來做跳躍結果的判斷。

(Menezes, Sathaye, & Coate, 1997)根據(Yeh & Patt, 1991; Yeh & Patt, 1993; Chang, Hao, Yeh, & Patt, 1994) ，提出了路徑預測(path prediction)的方法，不同的跳躍結果會產生不同的路徑，根據分支歷史樣式表格的內容，就可以選出不同的執行路徑，進而對多個分支做預測。

雖然動態方法可以藉由學習機制做更準確的預測判斷，但是必須要經過一段時間的學習才能做出準確的判斷，此外亦無法根據程式的行為給定判斷結果，因此我們設計一種新的分支預測機制，以靜態方法做偵測，並進一步與動態方法結合，以期可以達到更高的預測準確率。

(Ozturk & Sendag, 2010)探討分支預測的各種困難，其中提到固定長度迴圈出口分支的準確預測為其中之一。(Seznec, 2011)使用較長的歷史做為分支預測的依據。(Ozturk, Karsli, & Sendag, 2014)提出分支預測的樣式尋找方法，以分析找出較合適的硬體分支預測的所使用的硬體因素與樣式。(Ozturk, Karsli, & Sendag, 2016)提出在迴圈最內層分支做區域歷史的預測，其對任何向回跳的分支做處理，但是如本文內所描述，我們的分析顯示迴圈出口的分支指令並不一定是向回跳。(Seznec, San Miguel, & Albericio, 2015; Seznec, San Miguel, & Albericio, 2016)針對最內層回圈內的分支指令做分支預測，以降低硬體容量的成本，其並未針對迴圈跳出行為做針對性的分支預測模型。

這些方法僅在系統結構易取得的硬體因素上做不同的硬體樣式預測方法的尋找，其並未提出動態偵測迴圈結構與迴圈出口指令的判斷，再依據其做行為分類偵測，以之作準確對應的分支預測。硬體動態迴圈結構與分支分類與相關衍生因素用於這些樣式的分支預測，尚未被研究出來。

我們分析科學工程程式的迴圈行為，並找出使用傳統分支預測方法所有的分支預測失誤來源。在這些程式中，內迴圈巢的迴圈出口大多顯示一定的規律行為。如果使用傳統狀態機的分支預測方式，對於這些規律的迴圈出口分支行為不能有效的吻合預測，因此會造成一定的分支預測失誤。由於在科學工程程式中，規律的迴圈是其所在執行上花用最主要的計算時間，因此這類增加的預測失誤會造成在此類程式上的分支預測命中率的不足。

在本研究中，我們於迴圈出口分支指令，規劃取代使用傳統分支預測方法，因此我們發展一個硬體迴圈出口分支偵測技術與一個用於迴圈出口分支的分支預測技術。此偵測方法動態的將分支分類為迴圈出口分支與非迴圈出口分支。對非迴圈出口分支指令，仍使用傳統分支預測方法。但是對迴圈出口分支指令，我們發展一些吻合科學工程程式中規律迴圈出口行為的迴圈預測模型，此迴圈預測方法可以學習與適應選擇合適的迴圈出口行為的預測模型，以達到對其更準確的分支預測。

在本論文中，第二節將敘述對迴圈出口分支預測的解法規劃，第三節將介紹迴圈出口分支偵測方法，第四節說明迴圈出口分支的分支預測模型及其整合方法，第五節中，我們將說明本研究的實驗規劃、結果與討論，最後，我們針對本研究做一結論。

2. 解法規劃

在程式應用中，不同種的分支指令存在有不同的分支行為，本研究的主旨在於找出具備有特定類型分支行為的分支指令，支援其分類判斷方式，並支援其所屬分支行為的分支預測方法，亦稱之為分支預測模型(branch prediction model)。以目前的軟硬體分割的設計趨勢，這些分支行為分類功能與分支預測功能都需要以硬體的方式實現。

我們在本研究中的考慮是針對科學工程類型的程式，支援其分支預測更為準確。因此，我們根據科學工程程式與一般程式的差別，在於其迴圈出口分支指令具有相關種類的分支行為，如能有效動態偵測分類各分支指令，並提供相對準確的分支預測模型，將可以達到更有效的分支預測成效。

因此，所需要的分支指令行為分類將如圖 1 分為兩層，第一層分支分類進行迴圈出口與非迴圈出口的分支指令動態分類，這層分類需要對分支指令在程式中的結構關係做一研究，再依據研究結果，設計硬體的結構性偵測方法，產生第一層分類的結果。

在第一層分類為非迴圈出口的分支指令，由於其通常不具規律行為，我們將採用傳統的分支預測模型做其分支預測工作。如第一層判斷為迴圈出口分支指令，我們考量其可能仍為非規律分支行為，或具備有某種規律的分支行為。我們研

究各種規律分支行為的預測模型，再將這些整合。在第二層分支分類，進行針對迴圈出口的分支指令，判斷選擇合適的分支預測模型，據此以做其分支預測。第二層的分類選擇，我們考慮一個基於狀態機的學習模型，在程式執行時，動態逐步做適應性學習各迴圈出口分支指令所合適的分支預測類型。

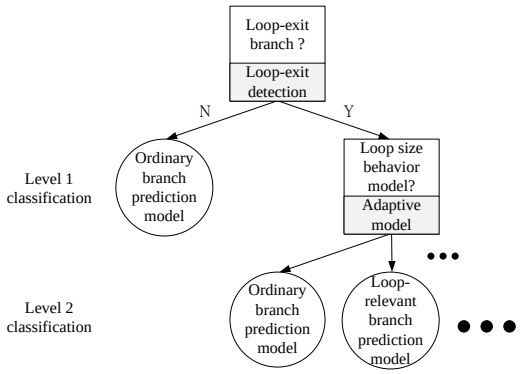


圖 1　二層分類分支預測

3. 迴圈出口分支偵測方法

3.1 整體方法

為了偵測一個分支指令是否為迴圈出口指令，我們規劃了三種方法，第一種由硬體取得迴圈樣式相關的指令，做迴圈樣式的判斷，確認是否符合迴圈出口的條件。第二種是當相關指令不在快取記憶體中，我們可以根據累積記錄在迴圈範圍硬體緩衝區(loop range buffer)的迴圈範圍做判斷，此牽涉累積迴圈範圍的方式，與緩衝區記錄與查詢的做法。第三種是根據原有判斷累積在分支歷史緩衝區(branch history buffer)中，作已判斷迴圈出口分支記錄的查詢。如三種均無法有效判斷為迴圈出口指令，則暫時會判斷為非迴圈出口指令。

要偵測一個分支指令是否為迴圈出口指令，首先我們先偵測其行為是否符合迴圈樣式(loop pattern)，之後我們再將結果紀錄在分支歷史緩衝器上，之後若需要再對同一指令做判斷，即可以以查詢方式取得判斷結果，不需要再做迴圈樣式的偵測。而當迴圈樣式偵測因所需資訊沒有查到而無法完成，我們可以透過迴圈範圍檢查，來輔助判斷其是否為迴圈出口指令。圖 2 顯示做此指令分類偵測所需之硬體設計，主要是加入迴圈範圍判斷與迴圈出口指令的偵測，結合分支對應的分支預測，可以對迴圈出口指令做特定的分支預測。

3.2 迴圈樣式預測

分支或非條件式跳躍指令由跳躍的目的地來區分可分為兩種，一種是往回跳(backward jump)的分支指令，一種是往前跳 (forward jump)的分支指令。兩者的區別為，當跳躍的目的地，其在記憶體上的位址小於目前分支指令所在的位址，則我們歸類為前者，反之則歸類為後者。

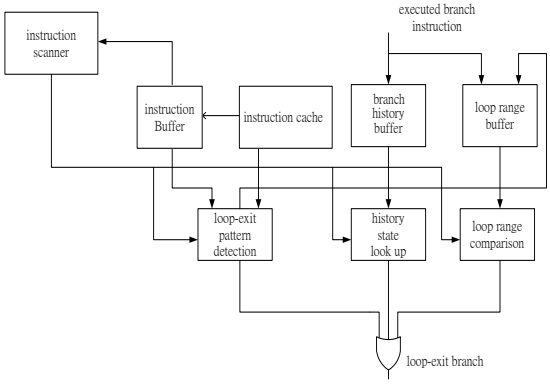


圖 2　迴圈出口指令之硬體設計

我們藉由觀察高階語言透過編譯器所編譯出來的低階語言程式，觀察其分支指令的運用方式有某種特性，以下就我們的觀察，歸納出幾種高階語言迴圈最常見的跳躍編譯樣式：

3.2.1 直接出口樣式(direct exit pattern)

分支指令的目的地(target)所在，往回一個指令為一個往回跳的分支或非條件式跳躍指令，其往回跳的範圍包含此分支指令所在，如圖 3(a)所示，其對應的低階語言描述如圖 4(a)所示，高階語言描述如圖 5(a)所示。

3.2.2 間接出口樣式(indirect exit pattern)

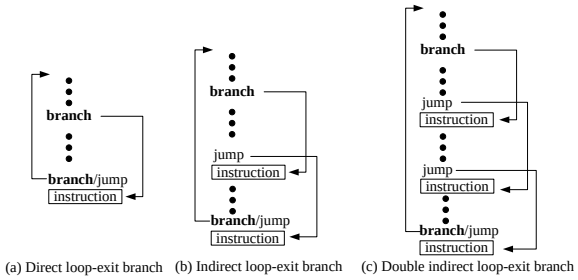
分支指令的目的地所在，其往回一個指令為一個往前跳的非條件式跳躍指令，而這個非條件式跳躍指令的目的地所在，往回一個指令為一個往回跳的分支或非條件式性跳躍指令，其往回跳的範圍包含此分支指令所在，如圖 3 (b)所示，其對應的低階語言描述如圖 4(b)所示，高階語言描述如圖 5(b)所示。

3.2.3 雙重間接出口樣式(twice indirect exit pattern)

分支指令的目的地所在，往回一個指令為一個往前跳的非條件式跳躍指令，而這個非條件式跳躍指令的目的地所在，往回一個指令為另一個往前跳的非條件式跳躍指令，而這第二個非條件式跳躍指令的目的地所在，前一個指令為一個往回跳的分支或非條件式跳躍指令，其往回跳的範圍包含此分支指令所在，如圖 3 (c)所示，其對應的低階語言描述如圖 4(c)所示，高階語言描述如圖 5(c)所示。

因此，我們對每一分支指令做迴圈樣式偵測的硬體流程如下，其流程圖如圖 6 所示。

1. 從分支指令的位移(offset)，與程式計數器(program counter)相加之後得到的記憶體位址(memory address)處，讀取跳躍目的地往回的一個指令。



(a) Direct loop-exit branch (b) Indirect loop-exit branch (c) Double indirect loop-exit branch

圖 3　迴圈出口樣式

```
$32:
  lw    $14, 4($sp)
  beq   $14, 5, $33
  lw    $15, 0($sp)
  addu  $24, $15, 1
  sw    $24, 0($sp)
  lw    $25, 4($sp)
  addu  $8, $25, 1
  sw    $8, 4($sp)
  blt   $8, 10, $32
$33:
```

```
.L3:
  lw    $2,16($fp)
  slt   $3,$2,10
  bne   $3,$0,.L6
  b     .L4
.L6:
  addu  $2,$fp,20
  .
  .
  .
  addu  $3,$2,1
  sw    $3,16($fp)
  b     .L3
.L4:
```

```
.L2:
  lw    $2,16($fp)
  addu  $3,$2,-1
  .
  .
  beq   $2,$3,.L5
  lw    $2,16($fp)
  addu  $3,$2,-2
  sw    $3,16($fp)
  b     .L6
.L5:
  sw    $0,16($fp)
  b     .L3
.L6:
  lw    $2,16($fp)
  b     .L2
.L3:
```

(a) Assembly code of loop 1 (b) Assembly code of loop 2 (c) Assembly code of loop 3

圖 4　迴圈出口的低階程式

```
for (a=0;a<10;a++) {
 if (a==5) break;
 b=b++;
}
```

```
for (a=0;a<10;a++) {
 b=b++;
}
```

```
do {
 c-=1;
 if (c!=5000)
  c-=2;
 else {
  c=0;
  break; }
} while (c>0);
```

(a) Loop 1　(b) Loop 2　(c) Loop 3

圖 5　迴圈出口的高階程式

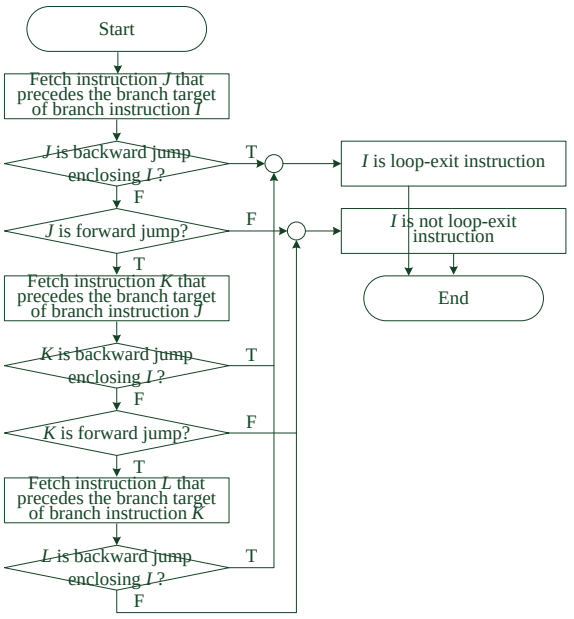


圖 6　迴圈出口判斷流程

2. 若步驟 1 讀取出來的指令為一往回跳的分支或非條件式跳躍指令，且往回跳的位址小於或等於原分支指令之位址，則判斷它為符合迴圈樣式一的迴圈出口指令。若為一往前跳的非條件式跳躍指令，則繼續做步驟 3。若兩者都不是，則判斷它為非迴圈出口指令。
3. 讀取步驟 2 所得非條件式跳躍指令的往回一個指令，若其為一往回跳的分支或非條件式跳躍指令，且往回跳的位址小於或等於原分支指令之位址，則判斷它為符合迴圈樣式二的迴圈出口指令。若為一往前跳的非條件式跳躍指令，則繼續做步驟 4。若兩者都不是，則判斷它為非迴圈出口指令。
4. 讀取步驟 3 所得非條件式跳躍指令的往回一個指令，若其為一往回跳的分支或非條件式跳躍指令，且往回跳的位址小於或等於原分支指令之位址，則判斷它為符合迴圈樣式三的迴圈出口指令。若不是，則判斷它為非迴圈出口指令。

3.3 迴圈範圍檢查

針對迴圈樣式偵測，因為硬體需要去讀取到目前執行指令的後幾個指令，而那些指令可能尚未被抓取至快取記憶體來，因此我們無法判斷目前執行的分支指令是不是迴圈出口指令。基於此點我們可以設計一個迴圈範圍檢查的機制，來協助這種情形的判斷，此外，可能有些迴圈出口樣式我們並沒有列入判斷的考慮，也可利用此方法來判斷是否為迴圈出口指令。

一個偵測的迴圈範圍包括迴圈起始地址(loop starting address (LSA))與迴圈邊界地址(loop bound address (LBA))，以及其有效性旗標(valid flag(V))，多筆偵測的迴圈範圍其可以被記錄在迴圈範圍緩衝區，以供動態查詢使用。

迴圈範圍檢查的流程如下:

1. 當執行到往回跳的條件式或非條件式跳躍指令，則以目前指令的地址搜尋迴圈範圍緩衝器中的每一個紀錄 LR_i 中的 *LBA*，如果有即做更新動作，將此指令跳躍目的地的位址值填入對應欄位的 *LSA* 中，如果沒有則新建一個紀錄，將 *LBA* 設成目前指令的地址，*LSA* 設成跳躍目的地的位址。而若這個指令是一個條件式跳躍，我們就判斷它為迴圈出口分支指令。
2. 當執行到往前跳的條件式跳躍指令，即對迴圈範圍緩衝器中的每一個紀錄 LR_i，做以下的檢查：

   $V \text{ and } (LBA \geqq IA \geqq LSA) \text{ and } ((TA1 > LBA) \text{ or } (TA2 > LBA))$

   其中 *IA* 表示目前跳躍指令的地址，*TA1* 表示目前跳躍指令的目的地，*TA2* 則是上節間接出口樣式中，第一個非條件式跳躍指令的目的地。如果有任何一個 LR_i 符合這個情況，我們就判斷這個指令為迴圈出口分支指令。

4. 迴圈出口分支預測模型與整合方法

針對已判斷為迴圈出口的分支指令，我們將以相關的迴圈出口分支預測模型對其做分支預測，以下將先說明各種規律迴圈行為所對應的迴圈出口分支預測模型，至於如何選擇相關模型，則為這些模型整合的適應性設計。

4.1 迴圈出口分支預測模型

我們分析迴圈出口分支行為影響到該次的迴圈計數(loop count)，我們根據迴圈計數的變化，整理出下列的數種行為

1. 永不出口(always-not-exit)行為，或過大迴圈計數而接近永不出口行為。
2. 固定累加計數行為，此包括累加值為負值或零。
3. 可變累加計數行為，在一定期間內，呈現累加計數行為，但之後又從新的計數重新做另外的累加過程。
4. 固定比例計數行為，每次迴圈計數均成比例性的改變。
5. 其他行為，此種顯示迴圈計數的變化無法落入上述四種行為。

針對這五種迴圈出口的分支行為，我們分別做出其分支預測模型，前四種模型在預測迴圈計數倒數到零時，則依迴圈出口方向預測分支結果，其餘時間預測不出口的分支方向。第五種不以迴圈計數為基礎，而以傳統的分支預測方法預測。

4.2 迴圈出口分支預測整合

當五種模型需要做整合的第二層分類選擇時，我們提出一個以狀態機逐步做適應性學習的過程，此狀態機顯示於圖 7。

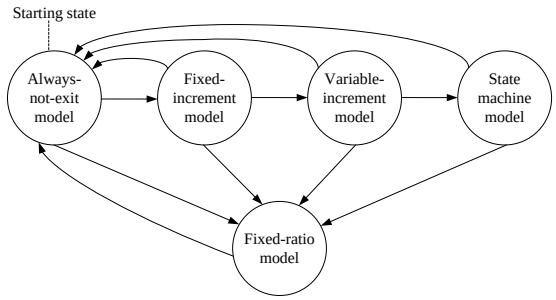


圖 7　迴圈出口分支預測模型的整合模型

初始時，迴圈出口分支指令使用永不出口的模型(always-not-exit model)做為預測，在持續過程中會記錄每次的迴圈計數。當逐次的迴圈計數產生時，會檢查是否符合固定迴圈計數累加的情況，如若成立，則轉入固定累加模型(fixed-increment model)。在固定計數模型預測時，會監測每次迴圈計數重設後，計數累加值是否會改變，此時則轉入可變累加模型(variable-increment model)。如若仍不正確可已轉入傳統的分支預測模型。在各預測模型下，均可監測是否符合固定迴圈計數的比例行為，此時則可轉入固定比例模型(fixed-ratio model)。

此基於狀態機的整合機制係針對所判斷出的各迴圈跳出的分支指令進行動態的預測模型選取，實際上的硬體行為非常詳細，我們在實驗結果上統計其整體的分支預測效應。

5. 實驗

5.1 實驗環境設計

我們的系統主要針對分支指令做分支預測，需要模擬的部分為指令從記憶體中被抓取開始，至分支指令的分支預測為止，因此我們將著重在此部份的模擬，如圖 8 所示，先萃取執行時期的資訊，並以之傳給模擬分支預測的蹤跡驅動模擬器(trace-driven simulator)，做分支預測模擬。

圖 8　效能模擬軟體流程

為了取得程式的執行資訊，我們修改 GDB 程式，使其能在使用 MIPS 指令集的 SGI 工作站上產生我們需要的程式執行蹤跡檔案。我們要得知分支預測的準確率，則必需要知道分支在何處發生及其執行結果，所以我們改寫 GDB 中 procfs.c 裡的 proc_run_process 函式，使其在每次執行指令時，都可以將分支指令在記憶體中的位址存在蹤跡檔中，接著再根據 MIPS 指令集的格式對其解碼，做迴圈相關性的檢查判斷，並輸出此資訊至蹤跡檔中。

在快取記憶體部分我們設計一個 256KB 四路集合關聯快取(4-way set associative cache)，在每一個紀錄(entry)中我們只設計有效(valid)及標籤(tag)欄位，每讀入一個指令就將標籤資訊寫入來更新快取的內容，之後要查詢樣式判斷需要的指令是否在快取上則僅需查詢其標籤是否在快取中，而不需要去做更新的動作。

在蹤跡驅動模擬器中，對於各模組的模擬，我們也依對應架構把它分成兩個部分：迴圈分類與預測判斷部分。迴圈分類部分為模擬依迴圈樣式判斷其是否為迴圈出口指令的機制，預測判斷部分為分支預測所採取的做法種類。在迴圈分類部分，我們可以選擇是否不做，光從歷史資訊來判斷分支預測的結果，藉以得知我們迴圈相關性方法可以提供多少額外的預測準確率。預測判斷部分則可分類成是否需要由多個路徑來決定判斷結果，而判斷方法又可選擇是由狀態機(state machine)變更狀態來完成預測，或是完全由歷史資訊的多數決定來做分支結果判斷。最後，我們再將數值預測模型加入我們原先的預測模型，以比

較各模型不同的組合所能達成的效果。

執行時期的資訊部分，我們藉由執行指令蹤跡萃取(instruction trace extraction)程式而產生程式執行蹤跡(program execution trace)檔案，檔案中包含了完整的分支指令執行順序及執行結果，以及與迴圈樣式偵測有關的資訊。在分支預測部分，我們以 C 語言撰寫一個蹤跡驅動模擬器，使其能讀取執行蹤跡檔，並根據我們的系統設計模擬其分支預測功能，並分析統計出實驗數據。

5.2 實驗規劃與結果

為了評估我們的設計，我們規劃了如表 1 所列測試程式的實際執行情形來做測量比較，其中 LZW、Gcc 與 Gzip 為一般的純量程式，其餘為科學工程程式中常用的程式模組。

表 1　測試程式

Program	Function
Huffman	Huffman encoding
MPEG	MPEG-1 decoding
LU	32x32 Matrix LU decomposition
Matrix Mult	48x48 Matrix Multiplication
FFT	64x64 2D-FFT
LZW	LZW encoding
Gcc	Gcc compiling
Gzip	Gzip encoding

在蹤跡驅動模擬器方面，我們因應各種不同的迴圈出口行為，而產生不同的迴圈出口分支預測模型的組合模型，其包括：

1. L 模型:代表不做迴圈出口分類，因此是使用傳統的分支預測模型，代表使用傳統分支預測方法(Chang, Hao, Yeh, & Patt, 1994)的結果。
2. LI 組合預測模型:代表組合傳統的分支預測模型、永不出口模型、迴圈固定累加模型與迴圈可變累加模型，為圖 7 省去迴圈固定比例模型的組合。
3. LIR 組合預測模型:代表組合傳統的分支預測模型、永不出口模型、迴圈固定累加模型、迴圈可變累加模型、與迴圈固定比例模型，即為圖 7 所示。

總體分支預測結果如表 2 所示，在 Huffman coding 方面，使用迴圈間距與狀態機決定的預測模型(LI)，比不做迴圈分類純粹由狀態機決定的預測模型提升了 8%的預測準確率，而在 MPEG 方面，做迴圈分類預測的也比不做迴圈分類的提升近 6%，矩陣 LU 分解方面也是有 6%的提升率。至於矩陣相乘，純粹狀態機決定或多數決定模型已提供了不錯的預測準確率，但加入迴圈分類預測模型之後，幾近辨認所有內含之規律行為的迴圈，因而提高到 99.9%以上。

表 2　分支預測命中率

Application	Prediction Model		
	L	LI	LIR
Huffman	86.41%	94.48%	94.48%
MPEG	91.00%	96.42%	96.42%
Matrix Mult	97.90%	99.98%	99.98%
LU	92.77%	98.38%	98.38%
FFT	84.33%	88.02%	88.27%
LZW	89.86%	88.93%	88.93%
Gcc	96.20%	95.36%	95.36%
Gzip	93.36%	93.33%	93.33%

在 FFT 方面，我們可以看到使用迴圈固定次數比模型(LIR)的預測準確率，比一般迴圈間距預測模型(LI)還要略高一些，這在其他程式則是相等的準確率，這代表 FFT 有一些迴圈次數是以等比改變，而我們的程式可以偵測到這部分，因而可以提供比較高的準確率，在其他程式方面，迴圈固定次數比模型跟迴圈間距預測模型則相等。

另外在 LZW、Gcc 及 Gzip 程式部分，因為程式大部分屬於純量程式，較不具有規律行為的迴圈組成，所以迴圈離開預測模型的預測準確率就和純粹狀態機的預測模型差不多，在迴圈出口分支的分支預測上，反因分支模型的適應性學習過程，造成略為損失分支預測的命中率。

本研究的方法是採用硬體支援做迴圈跳出指令的偵測與對其做分支預測模型選擇及預測，偵測電路僅牽涉減法器，各分支預測模型與狀態機均很簡單且僅需一組，主要是儲存迴圈跳出指令的分支歷史緩衝區的成本，因其僅需儲存各迴圈跳出指令，因此其容量可以比一般分支歷史緩衝區的容量小很多，因此可以以低硬體成本支援本研究的硬體分支預測方法。

6. 結論

在本論文中，我們針對科學工程程式的大量規律迴圈行為，設計相關的迴圈出口分支預測模型，再使用適應性整合方式將這些預測模型與傳統迴圈預測模型整合，成為組合的分支預測模型。我們並發展一個硬體迴圈出口分支偵測技術，以動態偵測與分類迴圈出口分支指令，以區分起動組合模型內的相對應迴圈預測模型，對應科學工程程式中規律迴圈出口分支行為，達到更準確的分支預測。在實驗裡，我們展示此分類迴圈出口分支預測可以有效改進對科學工程程式的分支預測命中率。

7. 參考文獻

[1] Chang, P.-Y., Hao, E., Yeh, T.-Y., & Patt, Y. N. (1994), "Branch classification: a new mechanism for improving branch predictor performance", *Proceedings of the 27th Annual International Symposium on Microarchitecture*, 22-31.

[2] Hennessy, J., & Patterson, D. (2012), *Computer Architecture: A Quantitative Approach*, Massachusetts: Morgan Kaufmann.

[3] Hwang, K., & Naresh, J. (2010), *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, New Delhi: Tata McGraw Hill.

[4] Lee, J., & Smith, A. J. (1984), "Branch Prediction Strategies and Branch Target Buffer Design", *IEEE Computer*, Vol. 17, No. 1, 6-22.

[5] Lilja, D. J. (1988), "Reducing the branch penalty in pipelined processors", *IEEE Computer*, Vol. 21, No. 7, 47-55.

[6] Menezes, K.N.; Sathaye, S.W.; Coate, T.M., (1997), "Path prediction for high issue-rate processors", *Proceedings of 1997 International Conference on Parallel Architectures and Compilation Techniques*, 178-188.

[7] Ozturk, C., & Sendag, R. (2010), "An Analysis of Hard to Predict Branches", *Proceedings of the 2010 IEEE International Symposium on Performance Analysis of Systems & Software*, 213-222.

[8] Ozturk, C., Karsli, I. B., & Sendag, R. (2014), "An Analysis of Address and Branch Patterns with PatternFinder", *Proceedings of the 2014 IEEE International Symposium on Workload Characterization*, 232-242.

[9] Seznec, A. (2011). 64KB ISL-TAGE branch predictor. *Championship Branch Prediction Competition*.

[10] Seznec, A., San Miguel, J., & Albericio, J. (2015), "The Inner Most Loop Iteration Counter: a New Dimension in Branch History", *Proceedings of the 48th International Symposium on Microarchitecture*, 347-357.

[11] Seznec, A., San Miguel, J., & Albericio, J. (2016), "Practical Multidimensional Branch Prediction", *IEEE Micro*, Vol. 36, No. 3, 10-19.

[12] Smith, J. E. (1981), "A Study of Branch Prediction Strategies", *Proceedings of the 8th International Symposium on Computer Architecture*, 443-458.

[13] Yeh, T.-Y., & Patt, Y. N. (1991), "Two-Level Adaptive Train Branch Prediction" *Proceedings of the 24th Annual International Symposium on Microarchitecture*, 51-61.

[14] Yeh, T.-Y., & Patt, Y. N. (1993), "A Comparison Of Dynamic Branch Predictors That Use Two Levels Of Branch History", *Proceedings of the 20th Annual International Symposium on Computer Architecture*, 51-61.