設計無標籤比對快取之省電嵌入式系統

陳青文
逢甲大學資訊工程系
chingwen@fcu.edu.tw

夏安
逢甲大學資訊工程系
m9702769@fcu.edu.tw

林俊宏
逢甲大學資訊工程系
m9619205@fcu.edu.tw

顧長榮
逢甲大學資訊工程系
p9522050@fcu.edu.tw

快取消耗電量在整體系統電量中佔了很大的比例。在快取元件中，標籤比對是造成快取耗電的主要原因之一。因此，如何設計一個不需比對標籤且能正確存取指令的省電快取便成為重要的問題。在本篇論文中，我們針對基本區塊來做分析，進入基本區塊中執行時將循序執行指令，此時是不用比對標籤。透過分析程式追蹤檔將常用基本區塊放置到額外小快取中。透過類似 BTB 的硬體架構紀錄基本區塊在小快取中的起始位址。進入小快取中存取時便能不用比對標籤來達到省電的目的。再者，對於不能透過分析追蹤檔找出常用基本區塊的程式。為了也能透過不用比對標籤以達到省電效果，我們提出動態抓取常用基本區塊的方法。當程式進入迴圈中執行時，我們將迴圈中常用的基本區塊放置到小快取中且透過不用比對標籤的方式來達到省電的目的。

***關鍵詞*—低耗電、快取、嵌入式系統、標籤比對**

1. 簡介

『嵌入式系統』是為了解決某些固定需求而特別設計出來的一種微電腦系統。這類產品在現今生活中愈來愈普及，例如：手機、衛星導航等...。設計嵌入式系統時，我們主要會考量系統的電力消耗與硬體成本並在兩者中做一個平衡以達到最大的系統效能。且這類產品通常被設計為行動裝置，因此其電源供應通常來自於電池，而電池供應的功率是有限的，所以如何設計一個省電的嵌入式系統，使產品的續航力能更長，便成了一個重要的議題。

隨著科技的進步，處理器效能提昇愈來愈快。然而在記憶體系統效能的提昇卻是相當有限。在計算機結構學中提到，在 1986 年之前，每年在處理器效能上提升 35%。但在 1987 年以後，每年提昇可達到 55%。處理器與記憶體之間效能的差距逐年增加。造成系統在運作時，處理器花費了大部分的時間在等待記憶體的回應。為了解決此問題，便在處理器與記憶體之間加入速度快的儲存單元，稱之為快取。讓大部分的存取行為都發生在快取之上，以利提升系統的效能。

當有快取存在於系統當中時，大部分的存取行為會在快取完成，造成決大多數的存取電量消耗在快取之上。過去研究指出，快取約佔處理器整體電量的 23% - 43% [1]-[2]。若能降低存取快取時的耗電量[1]-[21]，對於降低電量將會有很大的改善。過去學者降低快取耗電量可分為兩大類：(1)在處理器與快取之間放入小快取。(2)減少存取快取時不必要的元件耗電。

快取容量愈大其耗電量相對也就愈大，因此若能使用容量較小的快取來取代容量較大的快取且達到相同的命中率的話便能有效縮減快取耗電的問題。學者提出 Filter Cache [4]-[7]的概念，透過在快取前面加上一個額外的小快取，稱 Filter Cache。當在 Filter Cache 命中時消耗電量遠小於主快取的電量。加入 Filter Cache 後會造成效能的下降，由於 Filter Cache 是採取階層式快取的架構，當 Filter Cache 發生失誤時便會造成效能與電量上的損失。過去研究中，學者便提出了如何使用 Filter Cache 並且能改善效能下降的問題[5]-[7]。

快取內部包含的元件可分為：decoder、wordlines、bitlines、sense amplifiers、comparator、multiplexor drivers、output drivers 七種。根據查詢 cacti 電量表後發現，標籤比對電路佔了整體電量中很大的比例，且隨著關聯度的增加比較器電路所佔的電量比例也隨著升高。在直接映射快取中，大約有 55%耗電量是耗費在比對標籤的比較器上。當使用 2-way set-associative 的快取，標籤比對的耗電量大概佔了 65%左右。當 4-way set-associative 時，標籤比對的耗電量甚至會高達 75%左右。由此可見，快取中標籤電路也是快取耗電主要原因之一，如果能減少標籤比對次數便能有效降低快取存取電量。

Efthymiou and Garside [8] 提出減少標籤比對的位元數來達到省電的新快取架構，這方法中透過將標籤分段比對並且在比對後若不符合便停止剩餘位元的比對動作，如此一來每經過一次比對後將會有多數個比較器可以停止比對，便能達到省電的效果。這個方法中雖然可以透過較短的標籤比對進而達到省電的效果但由於將標籤

比對分成數個階段以此存取時間也就會變長進而導致系統效能下降。類似於調整標籤長度的方法中，P. Petrov, and A. Orailoglu [9] 也提出一個能動態調整標籤長短的架構，在此架構中系統會根據程式目前執行的情況來調整快取關聯度的高低，由於不同關聯度中標籤長度也就不同藉由使用較短的標籤位元數來達到省電效果。

除上述兩個藉由縮減標籤比對位元數來達到省電效果外，學者提出預測的方式來減低存取次數以達到省電的效果 [10]-[13]。由於程式在執行時會有規律性，再這類方法中系統會將這些規律性紀錄下來當作以後存取時的歷史資訊。利用這些歷史資訊去猜測資料是存放在集合內哪一個 way 中，若猜中便可以直接存取此時便可以省下剩餘大多數的比較器的耗電量。假設在 k-way set-associative 的快取中，在預測正確的情況下只需耗費原本 1/k 的電量因此再使用此方法後可以大幅降低快取耗電量。但預測錯誤時須付出很大的失誤代價。也就是說在預測錯誤時系統需額外花一個回合到剩餘 k-1 個區塊中找出資料所在。此時會使存取時間變為原本兩倍而造成系統效能嚴重下降。因此在此類方法中，如何維持預測的準確率便成為效能好壞的關鍵。

另外，有學者提出新的快取架構能在指令執行前便知道指令位於快取中哪個 way 中，如此一來系統只需去存取其中一個 way 便可抓取到此道指令。藉由這樣的方式，系統可以省下大多數的快取存取進而達到省電效果。過去學者將這類的方法稱之為 Way-determine [14]-[16]。在這類的方法中，為了紀錄指令位置資訊，系統會加入額外的硬體來紀錄這些歷史資訊，當指令下次再被執行到時便能透過這些資訊來存取指令，但也因為如此系統必須要去維護且確認這些資訊的正確性，因此系統必須付出額外代價，例如：cycle time，硬體成本等。在上述方法中，雖然可以透過減少標籤比對次數來達到省電的效果，但還是至少一次的標籤比對且在這類的方法中通常會有失誤代價而造成系統效能下降的問題。

在本篇論文中，我們的目標是希望設計一個透過避免不必要標籤比對以達到省電的嵌入式快取架構。當處理器到快取抓取指令時，處理器所發出的位址會被分成 Tag、Index 與 Offset 三部份。藉由 Index 找出相對應的集合。再透過 Tag 比對來確保能抓取到正確的指令。然而，程式執行處理器大部份時間都在執行少部份的常用指令。當常用指令在某段時間內頻繁的被存取，此時每次存取指令無論命中或失誤都還是要標籤比對後才能被存取，這些頻繁的標籤比對造成了快取不必要的耗電。由於快取的命中率通常都很高，假設有一快取的命中率有 90%，那表示將近 90%的指令都能在快取中正確抓取。在抓取這些指令時，雖然我們知道指令已經在快取內且沒有被替換，但還是需要比對標籤後才能存取，也就表示這段時間中的標籤比對都是浪費的。因此我們希望能設計出一個新的快取架構，透過減少或是避免這些不必要的標籤比對來達到省電效果。

標籤比對的目的是為了能確認抓取指令的正確性，因此我們希望能在能夠確保指令可以命中的前提下可以不用標籤比對。我們在原本架構中加入一小快取，稱之 Tag-Less Instruction Cache (TL-IC)。經由我們提出的方法後，在存取 TL-IC 時是不用標籤比對的。為了使 TL-IC 的使用率能夠很高以達到較好的省電效果。在本篇中我們提出了兩個方式來配置指令到 TL-IC 中：(1) 利用 profiling 的方式配置指令。(2) 利用動態配置指令的方式。首先，我們透過分析程式追蹤檔後，將程式中指令依照『執行次數』多寡依序放置到無標籤比對快取 TL-IC 中。如此一來 TL-IC 中存放的是程式中最常用的指令便能大幅的降低快取的耗電。再來，我們也提出了一個可以動態配置指令的方式，由於我們知道程式中大部分的時間都在執行迴圈中的指令，也就是說迴圈中的指令通常為程式中最常用的指令，我們提出一個機制能在程式進入迴圈中時將指令動態的配置到 TL-IC 中，透過這個方式便能在大部分時間中皆透過 TL-IC 來達到省電效果。

以下是本篇文章的文章架構，第二部分介紹與本篇文章相關的研究工作，在第三部分我們介紹在本篇論文中所提出的避免標籤比對的方法與存取流程，實驗部分我們將介紹本次實驗的模擬環境與實驗數據，最後則是本篇文章的結論。

2. 相關研究

過去學者提出一個有條件不比對標籤的方法[17]-[20]。作者在快取前面加上一個 cache line 大小的緩衝器，每次到快取存取後便把抓取到的 cache line 放到此 Linebuffer 中。當處理器發出連

續兩道指令位址且兩道指令位於同一個 block 時，此時可以不用比對標籤而直接到 Linebuffer 中抓取 cache line 中的下一道指令。假設快取 block 大小為 16 bytes 且指令長度為 32 bits，因此每個 block 可以存放四道指令，假設這四道指令皆為循序執行。在傳統快取作法下每存取一道指令都要存取一次快取，所以執行這四道連續指令要存取快取四次。但在使用 Linebuffer 後執行第一次後便將此line抓到其中且在這之後的三道指令便只要到 Linebuffer 中直接抓取即可，也就是說在執行這四道指令時只須存取一次快取即可。但當兩道指令位於不同 line 時，Linebuffer 便還是要一次的標籤比對與替換。此現象在程式執行到迴圈時會特別的明顯，由於 Linebuffer 的容量不足而造成頻繁替換的問題。此時 Linebuffer 不但省電效果有限反而會造成系統效能的下降。舉個例子，假設迴圈要執行 10,000 次，迴圈最內層的指令區塊大小為 5 個 block 大小且 Linebuffer 大小只有一個 block 大小。所以執行到此迴圈時，Linebuffer 要在短時間內卻要替換 50,000 次才能完成指令存取。頻繁搬取與替換 Linebuffer 對系統耗電量與延遲都是負擔。

Stephen Hines [21] 提出 Tagless-Hit Instruction cache 的架構，稱 TH-IC，其作法是在快取與處理器中間再加入一個較小的快取，並在此快取內部加上許多辨識位元。之後存取時系統便會利用這些辨識位元來告知指令是否在 TH-IC 中且在 TH-IC 中的位址，因此在此方法中作者指出 TH-IC 中能保證每次存取時皆命中。此方法中雖然可以在 TH-IC 中保證命中且不用標籤比對，但考量硬體成本後 TH-IC 快取大小不會太大，再扣除用來維護此機制的硬體成本後 TH-IC 的可用空間是很有限的。另外，當 TH-IC 中發生替換行為時便會將 TH-IC 所有的辨識位元清除掉，因此在這段時間 TH-IC 是不能使用的直到歷史資訊再次被建立起來，因此 TH-IC 在指令被替換後會有一段時間的使用率是相對低，此段中的省電效果也是偏低的。

3. 提出的方法

本篇論文提出一個透過避免多餘標籤比對來達到省電的快取架構。處理器每次到快取中抓取指令時，為了確保指令的正確性無論命中或失誤都還是需要一次標籤比對才能抓取。而快取命中率通常很高，也就是說大多數快取命中時的標籤比對其實都是可以避免，這些多餘標籤比對電耗便造成快取耗電的主要原因之一。因此我們希望能減少這些標籤比對次數來達到省電的目的。

3.1 利用 profiling 的方式配置指令

由於嵌入式系統執行的程式通常是固定的。對於此類的嵌入式系統，我們透過分析其特定的應用程式來達到無標籤比對的目的。透過分析程式追蹤檔，我們將常用指令區塊放置到額外增加的小快取中，並且在之後存取時可以不用標籤比對以達到快取省電的目的。另外，對於某些所執行應用程式較為廣泛的嵌入式系統，我們將在下個章節來介紹使用動態抓取方式達到無標籤比對效果。在動態抓取的方式下，我們透過偵測迴圈啟動機制來抓取指令到 TL-IC 中，在不用分析情況下便可以抓取到不同迴圈中的常用指令，並且存取 TL-IC 中的無標籤比對來達到省電效果。

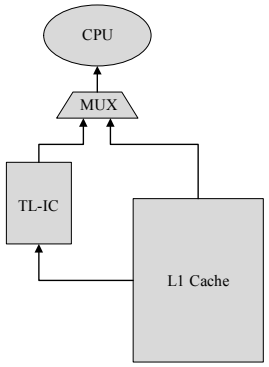


圖 1、加入 TL-IC 後系統架構圖

基於這個動機，我們在處理器中加入一個額外的小快取，稱之 TL-IC(圖 1)。在我們的架構中處理器可以直接到 TL-IC 中抓取指令且不用標籤比對。處理器抓取指令前會先經由一個多工器決定到 TL-IC 或原本 L1 快取來抓取此道指令。當處理器在 TL-IC 中抓取指令時，此時指令可以直接抓取且不必標籤比對，藉由減少標籤比對所消耗電量便可以達到省電的目的。TL-IC 的使用率愈高便可以降低愈多的標籤比對次數省電效果也就愈好。我們知道程式在執行時具有區域性，經由分析程式的指令追蹤檔來找出程式中的 basic block 並且根據其執行次數多寡依序配置到 TL-IC 內。透過這樣的方式可以找出程式中最常用的指令區塊，並將這些常用區塊放置到 TL-IC 中，並且藉由 TL-IC 中的無標籤比對設計來達到

省電的效果。

處理器到快取中存取資料是以區塊(block)為基本單位，而一個區塊通常會存放數道指令。因此當處理器連續抓取位於同一區塊的連續兩道指令時，為了確保指令的正確性處理器必須經過兩次標籤比對後才可以抓取指令。但此時兩次抓取的區塊其實是同一區塊，卻還是需要兩次標籤比對才能抓取。

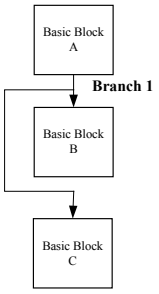
標籤比對是為了確保所抓取指令的正確性，無論是否為跳躍指令每次抓取前都會先經過標籤比對後才存取。但是一般來說，程式中指令只有少數的跳躍指令，大多數指令都是循序執行。為了降低標籤比對的次數，學者[7]提出當連續兩道指令為循序執行且位於同一區塊時，第二道指令定為此區塊的下一道指令，在這種情況下指令抓取可以不用標籤比對。藉由這樣的方式可以減少標籤比對次數來達到省電的效果。

當連續指令位於相同區塊時可以有條件的不用標籤比對，但在連續指令位於兩個不同的區塊時還是要標籤比對後才能存取。在一般電腦架構中，區塊內的指令數通常不多，也就是說在上面方法中每幾道連續指令抓取後便需要一次標籤比對。因此，在本篇論文中我們希望能在循序指令且位於不同區塊的情況下也可以不用額外標籤比對而直接存取。藉由這樣的方式可以更進一步的降低標籤比對次數。

我們希望在 TL-IC 中抓取循序指令時，無論指令是否位於同一個區塊都可以不用標籤比對。基於這樣動機，我們觀察程式流程後發現程式可以切成數個『basic block』。所謂 basic block 是一個循序執行的指令區塊，在 basic block 中的指令除了最後離開跳躍指令外，其餘指令皆為循序執行不會有跳躍的行為。也就是說，在進入 basic block 中執行後定不會執行到跳躍指令，且此段時間中指令皆為循序執行直到離開此 basic block 為止。因此，在本篇方法中我們將以 basic block 為單位來將指令放置到放到 TL-IC 中，之後再存取此 basic block 時便循序抓取指令直到離開 basic block 為止，且在這段時間中指令抓取皆不用標籤比對。

前面提到我們將部份常用指令以 basic block 為單位放置到 TL-IC 中，當處理器再執行到此 basic block 時，處理器必須知道目前所執行的 basic block 是否位於 TL-IC 中。因此在將 basic block 放置到 TL-IC 中時我們必須將指令放置的位址記錄下來，之後再執行到此 basic block 時處理器才能轉換到 TL-IC 中來抓取指令。我們知道 basic block 中除了離開的跳躍指令外其餘皆為循序指令，也就是說 basic block 之間是由跳躍指令所串連起來且 basic block 的首道指令是另一跳躍指令的目標位址。而這些跳躍指令的目標位址是紀錄在 Branch Target Buffer (BTB)中。因此在本篇方法中，我們便透過 BTB 來幫助我們紀錄 basic block 所在的位址。

BTB 是一種用來幫助完成分支預測的硬體架構。BTB 會在指令編譯或執行初期蒐集跳躍指令的目標位址以便程式執行時用來預測下一道目標指令的執行方向。BTB 目前已經被廣泛的使用在高效能或嵌入式電腦架構中，利用事先查詢 BTB 中紀錄的跳躍指令資料可以使處理器的效能提高。由於 basic block 的首道指令為另一道跳躍指令的目標位址，因此我們可透過 BTB 來紀錄 TL-IC 中 basic block 位址與其大小。



Branch Target Buffer

Branch Instruction	Address of Taken	Index of TL-IC	Size of Block	Address of Nontaken	Index of TL-IC	Size of Block
Branch1	T1	**Index of B_c**	**Size of B_c**	NT1	**Index of B_b**	**Size of B_b**
Branch2	T2	NULL	NULL	NT2	NULL	NULL
Jump1	T3	NULL	NULL	NT3	NULL	NULL
Branch3	T4	NULL	NULL	NT4	NULL	NULL
...						
$Branch_n$	Tn	NULL	NULL	NTn	NULL	NULL

圖 2、BTB 架構圖與 basic block 流程圖

我們在原來 BTB 架構中加入兩個欄位(圖 2 右)分別用來紀錄被放置 TL-IC 中 basic block 的位址與大小。當 basic block 被放進 TL-IC 的同時便在 BTB 對應的跳躍指令欄位中填入 TL-IC 中的起始位址與此區塊大小。之後再存取到此 basic block 時，處理器在到 BTB 查詢是否為跳躍指令與其目標位址的同時也會去判斷新增的兩個欄位是否為 NULL。若為 NULL 則表示指令不在 TL-IC 中，處理器在下回合還是到傳統 L1 快取中抓取指令。若不為 NUL 則表示此跳躍指令的目標位址是一個常用 basic block 的開頭且已經被放置到 TL-IC 中。處理器在下回合便會選擇到 TL-IC 中存取此 basic block。首先，處理器會根據 BTB 對應欄位中的 TL-IC 起始位址開始往下抓取指令，並且根據另一欄位中紀錄的 basic

block 大小 n，在抓取首道指令後繼續往下抓取 n 道指令，如此一來便能在無標籤比對的前提下完整的抓取 basic block。

舉例來說，圖 2 左是程式流程示意圖，Branch1 是一道條件式跳躍指令，當條件滿足時會跳到 basic block C 中執行，當條件不成立時則會到 basic block B 中執行。因此，當在將 basic block C 放到 TL-IC 的同時，我們也會將 basic block C 放置的起始位址以其指令數紀錄到 BTB 中。之後再存取到 Branch1 且發生跳躍的同時，處理器便會在下回合到 TL-IC 中來存取 basic block C，存取過程中根據 Branch1 欄位中紀錄的起始位址開始循序往下抓取 n 道指令便可以在沒有標籤比對的情況下抓取 basic block C。

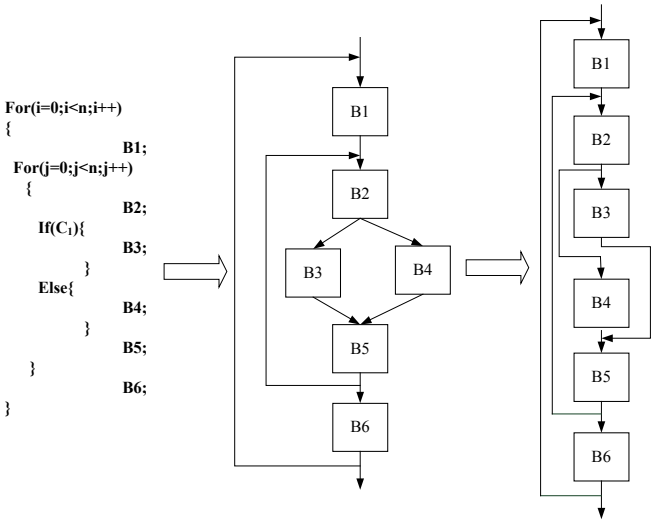


圖 3、 範例程式化簡成 Control Flow Graph

在這個部份我們舉一個例子來說明。我們以程式碼中的片段來做說明。圖 3 左為一個程式碼片段，根據程式碼流程可以進一步將程式碼化成程式區塊流程圖(Control Flow Graph)，透過程式區塊流程圖便可以清楚知道程式中各 basic block 之間的相對關係，我們根據程式流程將其切成六個 basic block 並化簡。上圖 3 右為化簡後的程式區塊流程圖。再來，我們透過分析程式追蹤檔後並統計出程式中 basic block 的執行次數並且根據執行次數多寡放置到 TL-IC 中。

圖 4 是本篇方法所提出的系統架構圖。在這個例子中，假設最常用的 basic block 分別為 B2、B5、B4、B3，也就是巢狀迴圈中內層迴圈中的 basic block。我們將這四個 basic block 配置到 TL-IC 中。在放置的同時我們將放置起始位址與大小紀錄在對應的欄位中以便下次抓取時可以知道指令所在位置。例如：當 B2 被放到 TL-IC 的同時，我們在 BTB 中 branch1 指令欄位後面加上 B2 在 TL-IC 中的起始位址 00 與其大小為 15。當下次執行到 branch1 且跳躍條件不成立時，處理器便會到 TL-IC 中的位址 00 開始連續抓取 15 道指令且中間皆不用標籤比對。

以此類推，若是執行到 branch 3 且跳躍條件不成立時，處理器同樣會到 BTB 中存取且此時 Index of TL-IC 與 Size of block 兩個欄位，此時這兩個欄位中的值皆為 NULL，這表示此對應的 basic block，B6 並沒有被放置到 TL-IC 中。處理器下一回合便會選擇到 L1 快取中抓取指令。因此透過 BTB 的幫忙我們可以將程式中常用區塊放置到無標籤比對 TL-IC 中，如此一來在程式執行時大多數的指令都會到無標籤比對的 TL-IC 中來抓取，藉由此方式便可以降低標籤比對的電量以降低整體快取系統的電耗。

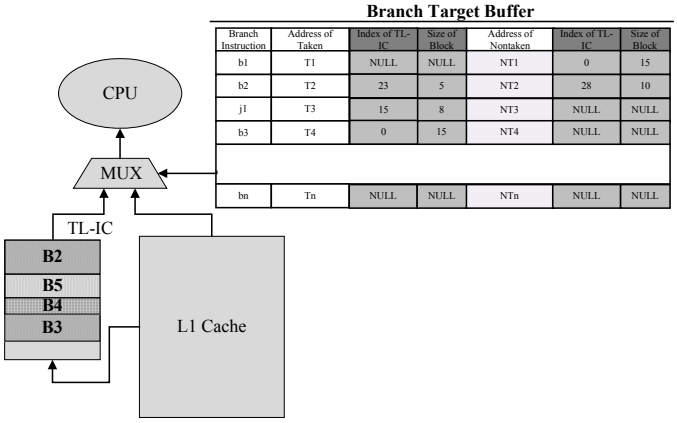


圖 4、透過 profiling 方式來配置指令的範例圖

分支預測器通常會使用 BTB 來預測指令下一步可能的執行方向，因此處理器每發出一道指令都會先存取 BTB 以判別指令是否為跳躍指令並將 BTB 中紀錄的目標位址傳送給 NPC 作為下一道執行指令位址。頻繁的存取造成 BTB 的電力消耗是系統中耗電的元件之一，因此我們可以藉由降低 BTB 的存取次數來達到省電的目的。

在本篇論文中我們提出一個透過減少 BTB 存取次數來降低整體電量的機制。我們的方法中，TL-IC 中存放的指令是以 basic block 為單位，且當進入 basic block 中執行時，除了跳離此 basic block 的跳躍指令外，其餘的皆為循序指令，此段時間內執行的指令都是循序指令，而 BTB 中紀錄的全都是跳躍指令的欄位，因此這段

時間內處理器到 BTB 中必定會發生失誤，而這些 BTB 存取所消耗的電量便都浪費且不必要的。我們提出一個機制當處理器切換到 TL-IC 中存取時可以避免存取 BTB 直到離開 basic block。

前面方法中提到我們在 BTB 中新增了兩個欄位：起始位址與大小用來幫忙配置常用 basic block 到 TL-IC 中。我們在系統內增加一個計數器 (圖 5)，當處理器到 TL-IC 內抓取第一道指令的同時將對應欄位中 block 大小放到計數器中，每抓取一道指令後將計數器的值減一。當計數器內的值大於零時，表示尚在存取此 basic block，所以下回合時處理器便不會存取 BTB。以此類推，當計數器等於零時，這時候表示此 basic block 已經抓取完畢，並在下回合回到傳統 BTB 存取的模式。由於 TL-IC 中存放的皆為常用的 basic block，因此透過這樣的機制便能大幅降低 BTB 的存取次數。因此在本篇論文中，大多數指令都可在 TL-IC 中命中且由於不用標籤比對且不用 BTB 的電力消耗，因此對於整體省電效果也會較原本的架構來得好。

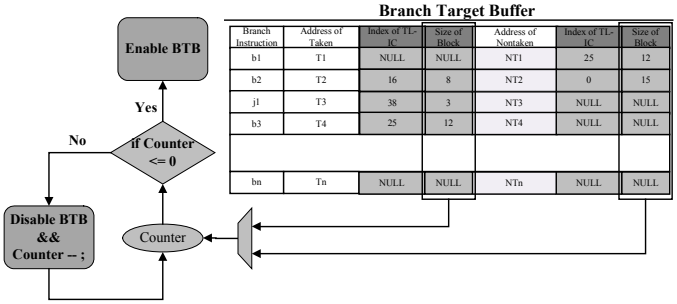


圖 5、透過計數器使有條件可以不存取 BTB

在本篇方法中，我們將程式中的常用指令以 basic block 為單位放到 TL-IC 中，由於程式區域性大部分時間都可在 TL-IC 中命中。TL-IC 存取一次所需消耗的電量是遠小於 L1 快取，所以相對於傳統快取我們對於常用指令部份可以省下很多電量。但是當程式執行到那些相對較不常用指令時便會回到 L1 快取中存取，然而此時對省電是完全沒有幫助的。我們希望當執行到這些較不常用指令時同樣可以減少標籤比對次數來更進一步達到省電的效果，我們在 L1 快取前面加上了一個 Linebuffer (圖 6)，在 Linebuffer 中可以有條件不比對標籤來達到進一步省電效果。

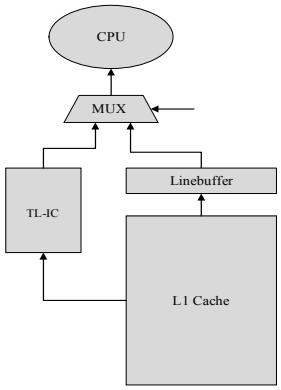


圖 6、加入 Linebuffer 之後的架構圖

雖然前面提到 Linebuffer 方法下會有頻繁被替換的問題，尤其是在執行到迴圈中的 basic block 時，由於 Linebuffer 的空間不足而產生頻繁替換。但在本篇方法中，大多常用指令皆已經被放置到 TL-IC 中此時執行的指令大多已不是關鍵迴圈中的常用指令區塊，這些指令大多是在程式離開迴圈後的指令，且通常不會有頻繁存取的指令區塊。

3.2 利用動態配置指令的方式

前面方法中提到我們透過分析程式追蹤檔的方式來抓取全部程式碼(Global)中的最常用指令區塊並放置到 TL-IC 中且在考量硬體成本下 TL-IC 容量通常不能太大，因此我們根據指令執行次數多寡依序擺放到 TL-IC 中。由於程式執行時具有區域性，也就是說不同的程式碼片段中(Local)中的最常用指令區塊也就不同，若能透過動態的方式來抓取此段時間中的常用的指令便能抓取到更多的真正常用指令。

我們提出透過動態配置的方式將常用指令放置到無標籤比對快取 TL-IC 中。經由動態配置的方式可以大幅縮減 TL-IC 容量並達到不錯的效果。在先前方法中是透過 profiling 的方式來抓取常用的指令，因此 TL-IC 容量大小會與其使用率成正比。但在動態抓取的方法中，我們只需抓取某一段時間內真正的常用指令而不用抓取全部時間中的常用指令，因此在動態抓取方式中只須使用較小容量的 TL-IC 便能同樣達到省電的效果。

下面章節中我們將說明如何使用動態配置的方式來抓取不同的常用指令，我們將此方法分成下面三部分來討論: (1) 啟動動態抓取機制 (2) 利用 BTB 來紀錄指令區塊位址 (3) 停止動態抓取機制。最後，我們同樣會舉一個例子來說明動

態配置指令的流程。

3.2.1 啟動動態抓取機制

在動態配置方法中，我們所使用的系統架構與前面提到的架構是相同的。我們希望找出不同時間中的常用指令，且將這些指令以 basic block 為單位擺放到無標籤比對的 TL-IC 中。我們透過分析程式追蹤檔找出常用指令的分佈情形。在分析後我們發現程式區域性主要集中在程式執行到迴圈的時候。迴圈中的常用指令很有可能在某一段時間中頻繁的被存取。若能在執行到此迴圈時將指令配置到快取中，則之後執行時快取命中率將會大幅提昇直到離開迴圈為止。因此，我們希望能透過抓取不同迴圈中的 basic block 到 TL-IC 中，並且透過 TL-IC 的無標籤比對來達到省電的效果。

為了有效找出不同時間的迴圈指令，我們根據跳躍指令跳躍的方向分成兩類：往前跳躍或往後跳躍。根據程式流程的特性發現當跳躍目標位址是往回跳躍且頻繁的發生跳躍的情況下很有可能是執行到迴圈的條件跳躍指令，在我們方法中我們透過 BTB 幫忙判斷跳躍指令跳躍情況與其跳躍目標位址，當執行到這類的跳躍指令時我們便假設將進入迴圈中執行並且啟動動態抓取機制並在下一回合開始將指令放置到 TL-IC 中。

3.2.2 利用 BTB 來紀錄指令區

我們藉由判斷迴圈跳躍指令來啟動動態抓取機制，且在啟動的下一個回合開始依序將指令填入到 TL-IC 中直到離開此 basic block 為止。當處理器再抓取到跳躍指令時表示完成了此 basic block 的抓取，並且在離開同時將此 basic block 在 TL-IC 中的起始位址記錄到對應的 BTB 欄位中。我們同樣在 BTB 中新增兩個欄位分別用來紀錄 basic block 起始位址與大小，當下次再執行到此 basic block 時，處理器便會切換到 TL-IC 中抓取常用指令區塊。

迴圈中的指令可能被切成數個 basic block，也就是說迴圈中可能會包含數個 basic block。例如：最常見的像是條件式 if、else 或 switch 等這類指令。這類條件式指令中通常只會成立一個，也就是說迴圈中包含的許多 basic block 在同一時間只會執行其中一個。以圖 7 為例子來說，當 if 條件成立時便會執行 B1 且在這回合中 B2 將不會被執行到。且在接下來幾個回合中，若 if 條件式一直成立時則 B2 便會一直沒有被配置 TL-IC 中。當某次執行到 if 條件式沒成立時此次執行到的便是另一個的 basic block，例如 B2。此時系統會判別到此指令尚位於迴圈中但 BTB 中對應欄位卻無記錄，在遇到這種情況時系統會再重新啟動 TL-IC 抓取機制來將此 basic block 也抓取到 TL-IC 中。因此，在我們提出的方法中系統必須在啟動 TL-IC 抓取機制時會將目前迴圈跳躍指令與其跳躍目標位址記錄下來並檢測目前指令是否位於迴圈中。藉由這些機制，我們便可以透過 BTB 的幫忙來將迴圈中的各個 basic block 根據執行先後順序來配置到 TL-IC 中。

```
if (C1) {                    switch (C1) {
        B1;                  case1:
                }                    B1;
else {                       case2:
        B2;                          B2
                }                                }
```

圖 7、 條件程式流程範例圖

3.2.3 停止動態抓取機制

程式在執行到迴圈時會具有很高的區域性與規律性。因此，當程式進入迴圈執行時，我們藉由偵測迴圈跳躍指令判別程式是否進入迴圈中執行，並且將這些指令抓取到無標籤比對的 TL-IC 中來達到省電效果。但當程式離開迴圈時，此時區域性與規律性會大幅降低，也就是說這些指令在接下來時間中可能不再是常用指令。因此，當偵測到程式離開迴圈時也必須有一個停止機制來切換回 L1 快取並且準備抓取下一個迴圈時的常用指令。

我們考量到程式執行時可能會有巢狀迴圈的情況發生。在我們方法中，當程式執行到巢狀迴圈時，我們仍然可以確保抓取到最內層的迴圈。這是由於內層迴圈的跳躍指令會較外層迴圈的跳躍指令先被執行到，在執行到內層迴圈的跳躍指令時，系統便會啟動抓取機制並且開始抓取內層迴圈中的常用 basic block 到 TL-IC 中。在接下來的時間中，處理器便都會到 TL-IC 中來存取。假設接下來時間中皆在執行內層迴圈中的常用 basic block，藉由 TL-IC 中無標籤比對設計便能達到很好的省電效果。

當程式跳離內層迴圈時，系統會切換回 L1 快取中來抓取指令直到偵測到另一個迴圈。在接下來執行到的迴圈跳躍指令很有可能便是外層

迴圈的跳躍指令，此時系統必須再次將此外層迴圈中的 basic block 抓取到 LT-IC。在這種情況下，如果外層迴圈所包含的指令數過多便很有可能因為 TL-IC 的空間不足而無法將全部 basic block 抓取到 TL-IC 中。但對於此段時間來說，內層迴圈中的指令執行次數定多於外層迴圈的指令，也就是說內層迴圈中的指令區塊才是目前最常用的指令區塊。而此時在 TL-IC 中指令卻為外層迴圈的指令。

為了避免這類情況發生，當系統偵測到指令離開迴圈時，我們會先保留 TL-IC 中的 basic block。當之後再偵測到迴圈跳躍指令時，藉由比對先後兩道跳躍指令的目標位置來判斷是否為巢狀迴圈。若不為巢狀迴圈，則表示接下來存取到的指令將會是一個完全不同的常用指令區塊。在這樣的情況下系統將會清空 TL-IC 與 BTB 中紀錄的所有指令與資訊並且重複前面的指令抓取步驟將目前執行到的迴圈填入 TL-IC。但若偵測為巢狀迴圈時，此時系統便要選擇保留 TL-IC 中的內層迴圈並且啟動抓取機制將外層迴圈接著填入 TL-IC 中直到完成或 TL-IC 容量不足為止。透過這個方式，我們可以在各種不同的情況下抓取到不同時間中真正的常用 basic block，且能在 TL-IC 容量大小有限的情況下保留最內層也是最常用 basic block。

最後，我們同樣舉一個例子來說明如何透過動態配置指令到 TL-IC 中，如圖 8 所示。我們以圖 3 的程式片段作為例子。以這個例子來說，在迴圈指令 branch 3 條件成立且往回跳躍時，我們便會將 B2 依序填入 TL-IC 中，並且在 BTB 中 branch 3 與 branch 1 對應的欄位中填入 B2 的位址 00 與其大小 15。之後在存取到 B2 時，處理器便會切換到無標籤比對 TL-IC 中來存取。

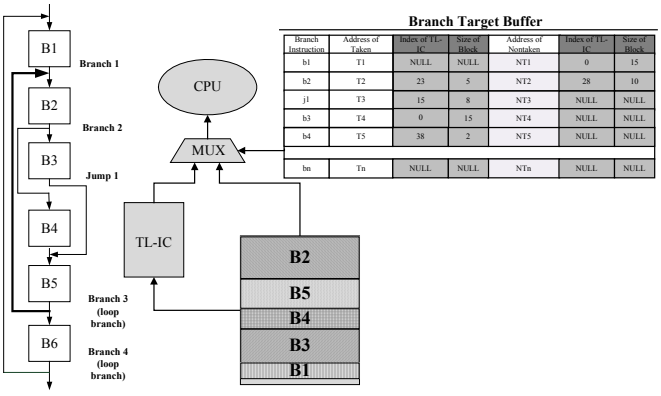


圖 8、動態配置指令的範例圖

接下來，根據程式流程系統會依序將 B4、B5 填入到 TL-IC 中。值得注意的是 B3 與 B4 分別為 if、else 條件成立時所執行的 basic block，這兩個區塊在同一個回合只會執行其中一個。因此，當 else 區塊第一次被執行到時會檢查 branch2 指令是位於迴圈中但 BTB 中的對應欄位卻無紀錄，此時系統會在次啟動指令區塊抓取來將 B3 也放置入 TL-IC 中。

當常用指令填入後，接下來時間中處理器只需到無標籤比對的 TL-IC 中存取來達到省電效果。然而一旦偵測到迴圈跳躍指令(branch 3)跳離此迴圈時，系統便會啟動 TL-IC 停止機制且在下一回合切換回 L1 快取中來抓取直到下一次再執行到迴圈跳躍指令才會再次啟動 TL-IC 機制。在我們方法中，為了確保抓取到最內層迴圈中的常用指令，我們會先保留 TL-IC 中的資料直到下一個迴圈跳躍指令。在這個例子中，當程式離開 branch3 的迴圈時，系統會先保留 TL-IC 中的資料。接下來在執行到 branch 6 的迴圈跳躍指令時會先藉由跳躍位置與上一個迴圈的目標位置比較。一旦比較後發現目前執行到的兩個迴圈關係為巢狀迴圈時，系統將會保留 TL-IC 中內層迴圈的資料，並且接續在後面開始抓取外層迴圈指令直到快取空間滿或是抓取完畢。藉由這樣方式，我們便可以確保在大小有限的 TL-IC 中存放著此段時間中最常用指令，如此一來便可以使快取使用率更好進而達到更好得省電效果。

4. 模擬結果

本章節中我們將針對本篇提出的方法來做實驗模擬並分析之。本篇方法中，我們分別透過分析指令追蹤檔與動態抓取的方式來將指令以 basic block 為單位放置到 TL-IC 中，透過降低標籤比對次數來達到省電的目的。首先，我們將針對在不同容量大小 TL-IC 中加入 Linebuffer 後對整體電量的影響來做分析。再來，我們將實驗在我們方法之下快取的整體電力消耗與其他方法來做比較。我們根據相關研究中與我們議題較相近的三個方法來做分析比較。此三個方法分別為傳統 L0 快取、Linebuffer 與 Stephen Hines 所提出的 TH-IC 方法來做比較。接下來我們會針對使用不同 TL-IC 大小時，快取電力消耗多寡來做比較與分析。最後，在動態配置指令的方法中，我們同樣會在省電效果與命中率部份來與傳統 L0

快取、Linebuffer 與 TH-IC 快取做比較與分析。

我們實驗環境是使用 ARM STD2.5 來模擬 ARM 系統的工作環境。標竿程式為 Media Bench 中的六個不同類型來當作我們得標竿程式。透過將挑選的標竿程式丟入 Strong ARM STD2.5 模擬器中，並由其編譯器將所填入的標竿程式指令編譯成 ARM 7 所能夠執行的靜態程式碼。再由 ARM STD 2.5 的 Debugger 執行靜態程式產生追蹤檔。此追蹤檔中會紀錄著程式執行時的所有指令及流程，所以我們可以利用分析追蹤檔來得到整個標竿程式執行時指令的流程與執行次數。

透過 CACTI 4.0 [3] 的電量模擬工具我們可以得知快取在不同關聯度、不同大小時各個元件單次執行所需的電力消耗。而之後方法中提到的 BTB 或 linbuffer 等得消耗電量也是由 CACTI 4.0 的模擬工具所取得，其中製程是使用參數 0.18μm。表 1 說明了存取不同大小快取標籤比對所需耗費的電量。

表 1、存取不同快取大小標籤比對耗費的電量

CacheSize (Bytes)	Energy (nJ)
256	0.019
512	0.021
1024	0.024
2048	0.026
4096	0.028
8192	0.032

4.1 利用 profiling 的方式配置指令

如圖 9 所示，我們針對不同大小的 TL-IC 中加入 Linebuffer 後對整體電量的影響。我們可以看到在 TL-IC 大小為 256 Bytes 時增加一個 Linebuffer 在 L1-IC 前可以省下 12%的電量。在大小為 512 Bytes 時亦可以省下 9%的電量。另外，我們可以看到當 TL-IC 容量愈來愈大時，Linebuffer 對整體電量的影響也就愈來於小。這個現象是由於當 TL-IC 夠大時能放置的常用指令區塊數也就愈來愈多，因此有一部份指令區塊便從原本 Linebuffer 中轉換到 TL-IC 中來存取。

再來，我們將本篇所提出的方法來與相同硬體下的傳統 L0 快取來比較。L0 快取與本篇提的方法的架構差別在於我們方法可以避免大多的標籤比對。從圖 10 的實驗結果中，我們可以看到在同樣大小 256 Bytes 時我們的方法 TL-IC 大約可以省下 64%左右的電量。由此可見無標籤比對對於快取整體電量的改善效果是很好的。

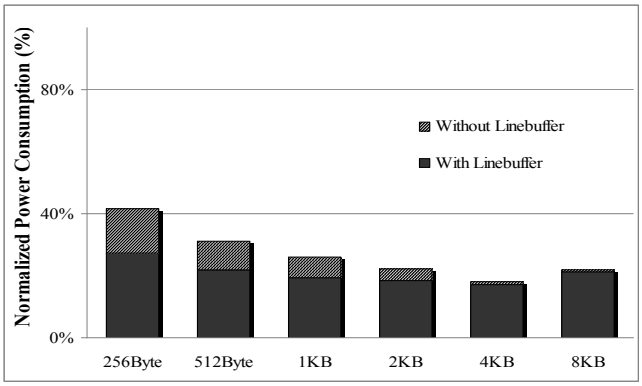


圖 9、不同 TL-IC 大小下電力消耗比較

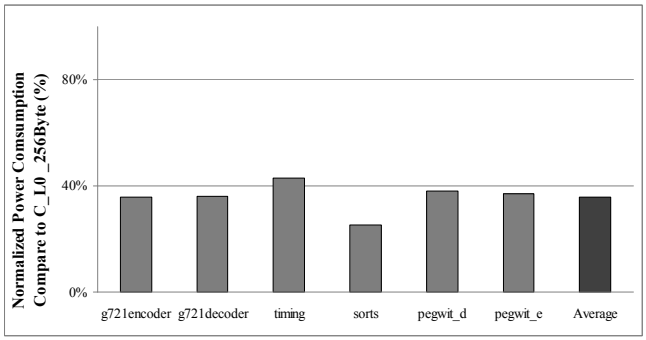


圖 10、TL-IC 與傳統 L0 快取電力消耗比較

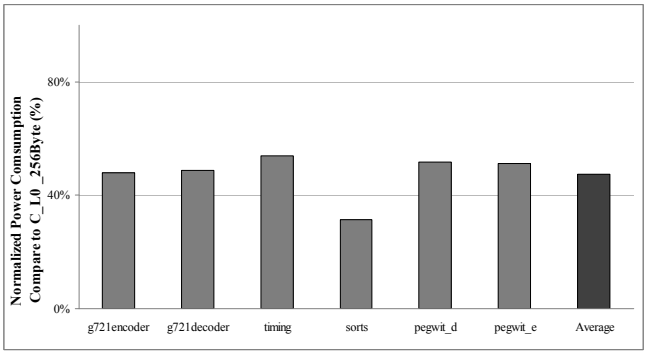


圖 11、TL-IC 與 Linebuffer 方法電力消耗比較

在前面相關研究中提到 Linebuffer 是一種有條件可以不用標籤比對的快取設計，透過在快取前增加一個與 cache line 同大的緩衝器，並透過在連續兩道指令且位於同一個 block 時可以不用標籤比對且只需耗費 Linebuffer 較小的電量來達到省電的效果。且前面也提到 Linebuffer 在執行到指令區塊時將很容易因為空間的不足而造成頻繁的替換，尤其在執行關鍵迴圈的指令區塊時頻繁替換的現象會更為明顯。在我們的方法中，透過將指令以區塊來放置到 TL-IC 存取，這樣一來便沒不 Linebuffer 頻繁替換的問題存在。從圖 11 的實驗結果中可以看到在與 Linebuffer 比較時我們的方法可以省下大約 52%左右的電量。這說明了當我們以 basic block 為區塊來存取可以避

免掉快取頻繁替換而耗電的問題。

在透過無標籤比對以達到省電效果的相關研究中，Stephen Hines 提出 TH-IC 的架構，其作法主要也是透過新增加一個小快取，透過在此快取中存取可以不用標籤比對以達到省電效果。在此方法中，TH-IC 是透過紀錄程式過去執行的歷史資訊來達到無標籤比對的目的。但是 TH-IC 中指令被替換時 TH-IC 則必須清掉所有的歷史資訊，並且再接下來的時間中 TH-IC 的使用率將會大幅降低直到歷史資訊再次被建立起來。再替換時常發生的情況下，TH-IC 省電效果便會顯得較低。在我們方法中，指令是透過 profiling 方式找出程式中最常用的指令區塊，因此不會有指令替換的問題產生，。我們在這個實驗中將本篇所提出的方法與 TH-IC 的方法再省電效果上來比較，從圖 12 的實驗結果中我們看到本篇所提出的方法可以省下大約 45%左右的電量。

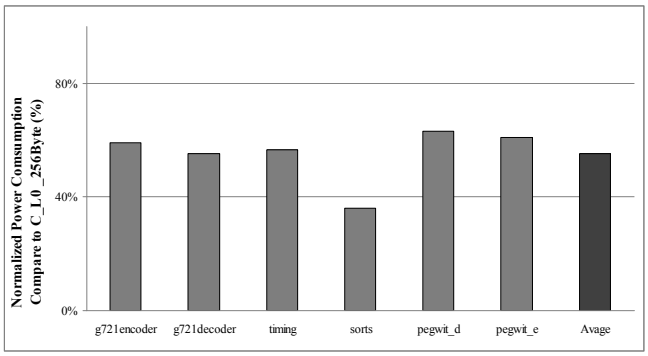


圖 12、TL-IC 與 TH-IC 方法電力消耗比較

我們知道 TL-IC 的大小會影響能放置的常用指令區塊多寡，放入愈多的常用指令區塊省電的效果理論上變會愈好。但實際上 TL-IC 的大小也影響著單次存取 TL-IC 的所需耗費電量，也就是說當 TL-IC 愈來愈大每次所存取的電量也就需要愈多。我們針對不同大小的快取在執行不同標竿程式時需要的總電量統計出來，圖 13 的實驗結果中便是各個標竿程式對於不同 TL-IC 大小時執行一次所需消耗的電量，可以看到隨著快取容量增加所需消耗的電量也就降低，但當快取增大到一個程度時，消耗電量反而會上升。尤其在某幾支標竿程式中此現象會特別的明顯。這是由於當快取大小增加到一定程度時其單次存取所需的電量便會超過無標籤比對所降低的電量，因此才會造成整體電量上升的結果。

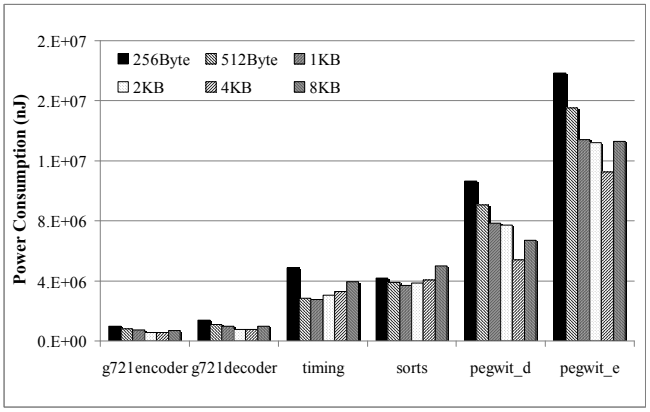


圖 13、TL-IC 不同大小下電力消耗比較

4.2 利用動態配置指令的方式

我們比較動態指令配置與 profiling 配置兩個方法之間所能達到的省電效果。圖 14 中我們可以看到在動態配置指令的方法下，在 128Bytes 的 TL-IC 時可以省下約 15%的電量，當 TL-IC 容量大小加大到 256 Bytes 的時可以省下約 18%的電量。我們看到使用動態配置指令的方式時，我們只需要使用容量更小的 TL-IC 便可以達到省電效果，這是由於程式執行時具有區域性，不同時間時常用指令也不一樣，透過動態配置指令的方式我們可以抓取程式執行時各個區段中真正的常用指令，也就是說從全域程式碼的角度來看我們可以抓取到更多更多的真正常用指令。

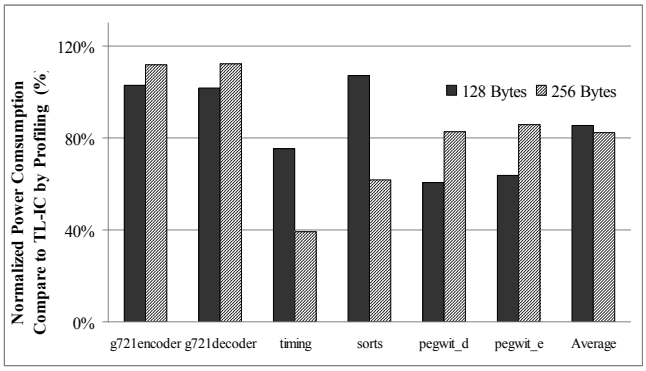


圖 14、動態配置與靜態配置方法電力消耗比較

動態配置指令的方法中，我們是透過進入迴圈後開始抓取指令，但當迴圈過大時，例如迴圈中包含副函式時，此時我們便無法完整將迴圈中所有常有指令都放置到 TL-IC 中。在 profiling 配置的方法中不會有這樣的問題發生，profiling 方法中是根據分析程式追蹤檔後挑選執行次數最多的 basic block 來放置到 TL-IC。我們發現在某些標竿程式中便會有上述迴圈過大的現象導致動態抓取的效果比透過 profiling 方法抓取的效果

來的差。但整體來說，對於某些無法透過分析追蹤檔的嵌入式系統中，我們必須使用動態配置指令的方法。且根據實驗數據顯示在動態抓取方法下整體來說確實可以在縮小 TL-IC 容量的前提下且達到不錯的省電效果。

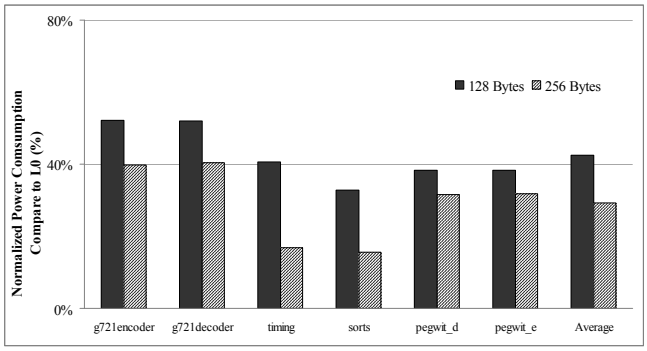


圖 15、動態配置與傳統 L0 快取電力消耗比較

如圖 15 所示、我們可以看到透過將迴圈中常用指令動態配置到 TL-IC 中且不用比對標籤的方法下，在快取大小為 128 Bytes 時可以比使用 L0 快取省下 58%的電量。在 256 Bytes 的 TL-IC 中，省下的電量大約有 71%。

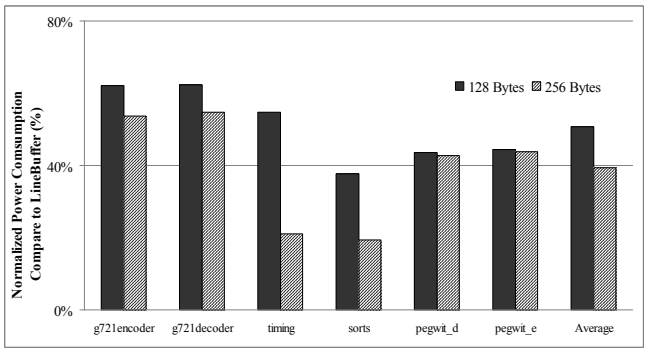


圖 16、動態配置與 Linebuffer 電力消耗比較

我們同樣與 Linebuffer 的方法在快取電力消耗上來做比較。從圖 16 的實驗結果中可以看到在使用 128 Bytes 的 TL-IC 時我們的方法便可以較 Linebuffer 省下大約 49%左右的電量。當快取加大至 256 Bytes 時更可以省下約 61%的電量。

最後，我們同樣與 Stephen Hines 所提出的 TH-IC 方法來做比較。TH-IC 同樣也是透過避免標籤比對來達到省電的目的。不同的是，我們在動態配置指令方法中只有在執行到迴圈跳躍指令才會切換到 TL-IC 中存取。一般來說，當程式進入迴圈中執行時將會有相當高的區域性與規律性。因此，這段時間中 TL-IC 的使用率會很高。藉由執行迴圈時指令的規律性以及無標籤比對的機制來達到更高更好得省電效果。從圖 17 的實驗結果中我們看到本篇所提出的方法在 TL-IC 大小為 128 Bytes 時大約可以省下 45%的電量。當快取加大到 256 Bytes 時，TL-IC 可以省下大約 54%左右的電量。

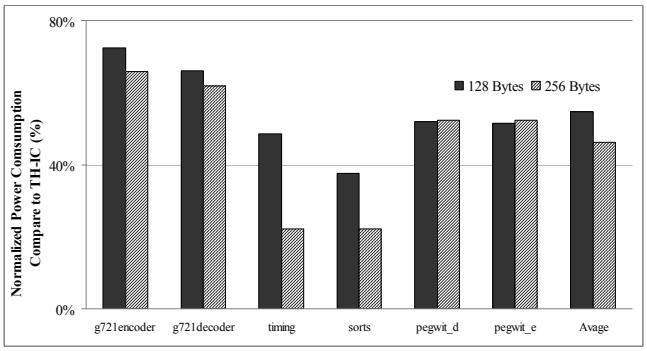


圖 17、動態配置與 TH-IC 快取電力消耗比較

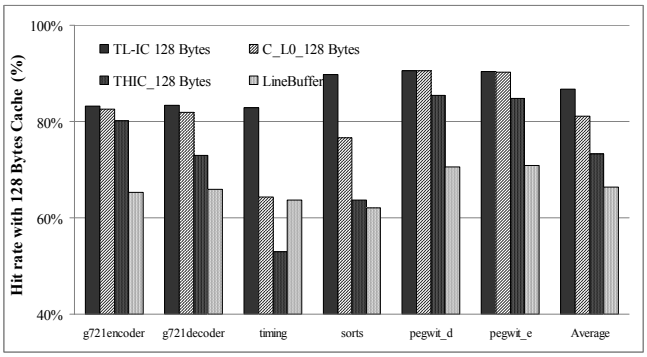


圖 18、動態配置 Hit rate 比較

接下來章節中，我們針對不同方法中快取的 hit rate 做分析比較。在使用 128 Bytes 的 TL-IC 下，系統 hitrate 大約可以達到 83%左右。在圖 18 中，我們看到由於傳統 L0 快取採垂直快取架構因此 hit rate 最高，但由於 L0 快取中每次存取皆要標籤比對，因此在上圖中我們也看到整體電量來說 L0 快取是最耗電的。再來，Linebuffer 方法中，由於加入與快取區塊同大小的緩衝器，Linebuffer 命中率大約可以達到 66%左右但由於其單次存取電量很小，因此整體電量卻較傳統 L0 快取來的好。在 TH-IC 方法中，主要是根據紀錄過去指令執行的歷史資訊，因此其 hit rate 大約有 73%左右。對於設計一個省電的嵌入式系統快取來說，快取的命中率固然重要，但快取單次存取所需消耗的電量來的更為重要。在考量單次存取電量的因素後，若還能達到較高的命中率才能有更好的省電效果。因此本篇方法所提出的 TL-IC 是可以大幅降低整體快取電耗。

5. 結論

在本篇論文中，我們提出一個新的快取架構

能有效改善嵌入式系統耗電的問題。我們在 L1 快取旁上一個額外的快取，且分別透過靜態與動態的方式來將程式中常用的 basic block 放到 TL-IC 中。在靜態的方法中，我們透過分析程式的指令追蹤檔來統計出程式中最常用的指令區塊並且透過 BTB 紀錄這些區塊的資訊，之後再存取到這些常用區塊時便可以不比對標籤而直接到 TL-IC 中存取。由於在存取 TL-IC 時保證是循序執行，因此這段期間可以不用存取 BTB。對於較不常用的指令區塊，我們在 L1 前面再加上一個 LineBuffer 進一步避免掉部份的標籤比對。

由於程式執行時具有區域性，不同時間中執行到的常用指令也就不同，因此，我們提出透過動態配置指令的方式來抓取不同時間中真正的常用指令。在動態配置指令的方法下，我們只需使用較容量較小的 TL-IC 便能達到同樣的省電效果。我們針對迴圈中的指令通常具有很高區域性的特性，當程式進入迴圈中執行時我們便開始將迴圈中常用指令以 basic block 為單位放到 TL-IC 中。且同樣透過 BTB 來紀錄這些區塊的資訊。在程式離開迴圈時，我們將 TL-IC 中的指令先保留直到執行到下一個迴圈跳躍指令。且若執行的迴圈跳躍指令為 TL-IC 中的外層迴圈我們會保留 TL-IC 中內層迴圈中的 basic block，並且接著將外層迴圈的指令循序填入，透過這個方式我們可以抓取到這段時間中真正的常用指令。

最後，在實驗結果中我們分別比較傳統 L0、LineBuffer 和 TH-IC 相關研究的電量消耗。我們提出的 profiling 方法對於傳統 L0 與 LineBuffer 分別可以省下 66%與 62%的電量。對於 Stephen Hines 所提出的 TH-IC，由於使用率的關係，我們的方法則可以省下 45%左右的電量。在動態指令配置方法下，我們在 128Bytes 與 256Bytes 的 TL-IC 大小下分別可以省下 22%與 18%的電量。

6. 誌謝

C. W. Chen's research was supported by Nation Science Council (NSC 96-2221-E-035-004-MY3).

7. 參考文獻

[1] Biju K Raveendran, T S B Sudarshan, Avinash Patil, Komal Randive, S Gurunarayanan, "Predictive Placement Scheme In Set-Associative Cache For Energy Efficient Embedded Systems," IEEE-International Conference on Signal processing, Communications and Networking, pp. 152-157, Jan. 2008.

[2] Kugan Vivekanandarajah, Thambipillai Srikanthan and Saurav Bhattacharyya, "Dynamic Filter Cache for Low Power Instruction Memory Hierarchy", In proceedings of the EUROMICRO Systems on Digital System Design (DSD'04), IEEE 2004.

[3] S. J. E. Wilton, and N. P. Jouppi, "CACTI: An enhanced Cache Access and Cycle Time Model," In IEEE Journal of Solid-State Circuits, Volume 31, Issue 5, pp.677 -688, May 1996.

[4] Lea Hwang Lee, William Moyer and John Arends, "Instruction Fetch Energy Reduction Using Loop Caches for Embedded Applications with Small Tight Loops", In Proceedings of the International Symposium on Low Power Electronics and Design.

[5] Kashif Ali, Mokhtar Aboelaze and Suprakash Datta. "Modified Hotspot Cache Architecture: A Low Energy Fast Cache for Embedded Processors" Proceedings of 2006 International Conference on Embedded Computer Systems, pp. 35-42, July, IEEE 2006.

[6] J. Kin, M. Gupta and W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," In Proc. of 30th International Symposium on Microarchitecture, pp. 184-193, Dec. 1997.

[7] Kugan Vivekanandarajah and Thambipillai Srikanthan, "Custom Instruction filter Cache Synthesis for Low-Power Embedded Systems", In proceedings of the 16th International Workshop on Rapid System Prototyping (RSP'05), 2005 IEEE.

[8] A. Efthymiou, and J. D. Garside, "An Adaptive Serial-parallel CAM Architecture for Low Power Cache Blocks," In ISLPED'02: In proceedings of the Design, pp.136-141,2002.

[9] P. Petrov, and A. Orailoglu, "Data Cache Energy Minimizations Through Programmable Tag Size Matching to the Applications," ISSS'01: International Symposium on System Synthesis, Montreal, pp. 113-117, October 2001.

[10] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting Set-Associative Cache for High Performance and Low Energy Consumption," In Proceedings of the 1999 International Symposium on Low Power Electronics and Design, pp. 273-375, August 1999.

[11] Michael D. Powell, Amit Agarwal, T. N. Vijaykumar, Babak Falsafi and Kaushik Roy, "Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping" In the Proceedings of the 34th International Symposium on Micro architecture , 2001.

[12] Kevin Skadron, Pritpal S. Ahuja, Margaret Martonosi and Douglas W. Clark, "Improving Prediction for Procedure Returns with Return-Address-Stack Repair Mechanisms", In proceedings of the 31st Annual ACM/IEEE International Symposium on Micro architecture, pp. 259 - 271, Nov.30 – Dec. 2, 1998.

[13] Stkphan Jourdant, Tse-Hao Hsing, Jared Stark, Yale N. Patt, "The Effects of Mispredicted-Path Execution on Branch Prediction Structures", In proceedings of the PACT '96, 1996 IEEE, pp.58-67.

[14] In proceedings of the te and ions for Superscalar Processors Eui-Young Chung, Cheol Hong Kim and Sung Woo Chung, "An Accurate and Energy-Efficient Way Determination Technique for Instruction Caches by Using Early Tag Matching" 2008 4th IEEE International Symposium on Electronic Design, Test & Applications, pp. 190-195.

[15] Zhang Mingming, Chang Xiaotao and Zhang Ge, "Reducing Cache Energy Consumption by Tag Encoding in Embedded Processors", In proceedings of the ISLPED'07, ACM 2007, pp. 367-370, August,2007.

[16] Ming Yang and Lixin Yu, "Tag compression for low power in instruction caches", In proceedings of Electron Devices and Solid-State Circuits, IEEE 2007, Page(s):837 – 840, Dec. 2007.

[17] Ramesh Panwar and David Rennels, "Reducing the frequency of tag compares for low power I-cache design," International Symposium on Low Power Electronics and Design, pp. 57-62 , 1995.

[18] Kanad Ghose and Milind B.Kamble, "Energy efficient cache organizations for superscalar processors." In Power Driven Microarchitecture Workshop, held in conjunction with ISCA 98, June 1998.

[19] K. Ghose and M. B. Kamble. "Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation." In Proceedings of the 1999 International Symposium on Low Power Elec- tronics and Design, pages 70–75, NY, USA, 1999.

[20] Ching-Long Su and Alvin. M. Despain, "Cache designs for energy efficiency", In Proceedings of the 28th Annual Hawaii International Conference on System Sciences, pp. 306 - 315 vol.1, Jan, 1995.

[21] Stephen Hines, David Whalley, and Gary Tyson, "Guaranteeing Hits to Improve the Efficiency of a Small Instruction Cache", 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), pp. 434 - 444, Dec.