利用區域性與變動位址偏移量設計高效率的追蹤檔壓縮方法

陳青文　　黃銘吉　　顧長榮
逢甲大學資訊工程系　逢甲大學資訊工程系　逢甲大學資訊工程系
chingwen@fcu.edu.tw　m9510632@fcu.edu.tw　p9522050@fcu.edu.tw

摘要

在計算機結構的研究範疇中，追蹤檔的分析與模擬是一項不可或缺的技術，透過追蹤檔的分析不僅能夠得知一個系統複雜的執行行為，也能夠去評估一個新開發的系統所能產生的效益。然而，由於追蹤檔為一個程式執行時所有行為的記錄，因此儲存及管理追蹤檔也必須付出相當可觀的成本。為了簡化追蹤檔的管理，追蹤檔在儲存之前大多會先經過壓縮的程序。在過去壓縮方法的研究中，大部分是利用記錄位址的位移量來達到壓縮的效果，然而，由於程式執行具有區域性，此方法會儲存大量重複的位移量因而浪費空間。因此，若此空間浪費的問題得以改善，並且進一步的縮減每筆位移量的儲存空間，那麼就可提升壓縮效果。在本文章中，我們提出"利用區域性與變動位址偏移量設計高效率的追蹤檔壓縮方法" (LAAO)。主要針對非循序位址處理，由於在這些非循序位址的位移量之間也具有區域性，因此我們將位移量放入類似快取的動態參考表中，以減少儲存重複位移量的次數，並利用記錄參考表欄位索引值取代記錄整筆位移量，如此一來，能夠使用較少的記錄位元來儲存位移量，提升整體的壓縮效果。另外，在追蹤檔的分析時，我們時常只針對某個特殊需求而分析追蹤檔內的某些區段，由於追蹤檔案相當龐大，若每次分析追蹤檔都必須將整筆追蹤檔解壓縮是相當沒有效率的，並且若使用一般個人電腦進行追蹤檔分析，龐大的追蹤檔將對系統記憶體造成相當大的負擔。因此，為了解決此問題，在本篇論文的壓縮方法中也支援了即時解壓縮的功能，讓使用者能依照需求針對特定區段進行接壓縮。最後，在我們的實驗結果中顯示，我們的壓縮法可以使壓縮後的追蹤檔變成原始檔大小的3%~4%。

關鍵詞：追蹤檔，壓縮，位移量，區域性

一、 簡介

追蹤檔是記錄了程式執行時所執行過的指令以及位址，而在程式執行中，往往會有數以百萬計的指令被執行，因此儲存追蹤檔所需要的容量也相當龐大，雖然隨著科技的發展，儲存裝置的容量也大量的增加，但對於儲存如此龐大的追蹤檔而言，還是有著許多挑戰與不便。

儘管追蹤檔的管理與儲存有著某種程度的困難，但追蹤檔卻在許多計算機結構等研究中扮演著相當重要的角色。如：在近幾年受到高度重視的嵌入式系統發展上，不僅講求執行效率的提升，對於成本以及電力消耗也都有相當的要求，而這些訴求過去都有學者透過追蹤檔的分析、觀察程式執行時的特性，並用以解決問題。透過觀察追蹤檔可以得知如何重新安排指令順序以提高系統的整體執行效能，或者利用追蹤檔的分析得到常用指令，並將常用指令放入快取中以提高快取命中率而達到省電的效果等。

追蹤檔除了應用在嵌入式系統的研究上，也可以應用在桌上型電腦以及其他方面等，由於追蹤檔龐大不易儲存管理，因此，我們希望透過壓縮的方式來簡化追蹤檔的管理並縮減儲存追蹤檔縮需的空間。

追蹤檔儲存著程式執行過程的記錄，追蹤檔的記錄是就如同程式一樣具有區域性的特質，所以如果在沒考慮到區域性的特點之下使用一般的壓縮方法來壓縮追蹤檔，那麼將無法對追蹤檔做有效率的壓縮，在追蹤檔的壓縮若能夠考慮區域性，那麼勢必在壓縮率上有顯著的成長。

過去學者在壓縮的研究上已經有不錯的成果[1-12]，在壓縮的方法上大致可分成兩類，一種是失真壓縮 (Lossy) 另一種是不失真壓縮 (Lossless) 。失真壓縮是取出檔案中較具代表性的部份當作來壓縮，而其他部分忽略不記錄，此種壓縮方法包含取樣 (Sample) [1-3]和過濾 (Filtering) [4]，雖然失真壓縮可以擁有不錯的壓縮率，但經由失真壓縮的檔案解壓縮後，將無法得到與原始檔案完全相同的檔案，如 mp3、jpeg 壓縮。因此，若使用失真壓縮的方法來壓縮追蹤檔，會造成解壓縮過後得到跟原始追蹤檔有所不同，此時可能會流失我們所需要的重要資料，如果我們對此檔案做分析，那麼可能會造成在分析追蹤檔上的不準確，而影響到實驗結果，所以失真壓縮法不適合用於追蹤檔的壓縮。在不失真壓縮方面，壓縮過程中不會忽略任何資料，所以在壓縮率的表現上，不失真壓縮的壓縮率不如失真壓縮來的出色，但不失真壓縮法的方法，可保證

在解壓縮還原後的檔案與原始檔案完全相同。所以對於追蹤檔的壓縮方式，較適合使用不失真壓縮的方式來進行壓縮，在過去的研究中，Mache [5]，PDAT [6]，SBC [7]，以及 LBTC [8]等都是屬於不失真壓縮這類型的壓縮法。

過去對於追蹤檔壓縮研究，有些利用記錄位址偏移量來取代儲存完整 32 位元的位址，但在記錄位址偏移量的過程中，會因程式的區域性而記錄了許多重複的偏移量，如果能夠減少重複出現的次數，即可以節省儲存的空間。對於所要儲存的位址偏移量來說，如何決定適當的位元長度來記錄位址偏移量也必須加以取捨，若使用過多的位元來記錄位址偏移量會造成空間上大量浪費；使用變動長度的位元記錄偏移量，為了能夠辯別正確的位元數，必須額外增加標頭或識別碼，以供解壓縮引擎解碼，增加的識別碼除了會使壓縮效果下降以外，在解壓縮的過程中也會對效能造成負面的影響。

在本文章中，我們提出了“利用區域性與變動位址偏移量設計高效率的追蹤檔壓縮方法(Trace File Compression with Locality and Adaptive Address Offset, LAAO) ”來壓縮追蹤檔，有效縮減重複記錄位址偏移量的問題，為了解決偏移量記錄位元的問題，我們提出了三個模式來因應不同的程式，並且改善了標頭或識別碼對壓縮率所造成的影響。在本論文的方法中，我們將追蹤檔中的指令分為循序位址與非循序位址，對於非循序指令的位址加以運算後儲存在動態參考表當中，透過動態參考表的使用來減少位址偏移量重覆記錄造成空間浪費的問題，最後，利用記錄動態參考表的索引值取代記錄位址偏移量，來縮減儲存時所需要的空間。

過去研究指出，大部分的程式都能夠以 16 位元來表示 99%的位址偏移量，若將 32 位元的位址偏移量以 16 位元取而代之，則可再提高壓縮率將近 50%，也就是追蹤檔經過壓縮後的空間大小約為原始追蹤檔的 25%左右。此外，因為追蹤檔裡的位址偏移量具有區域性的特質，我們設計了具有類似快取特性的動態參考表來輔助壓縮，動態參考表的功能是用來暫時存放位址偏移量，透過動態參考表的存取，可以減少減少重覆儲存相同位址偏移量的次數。另外，我們記錄動態參考表欄位的索引值來減少記錄完整位址偏移量所需的空間。

在實驗模擬中得知，我們所提出的壓縮法可以將原本的追蹤檔壓縮到原始追蹤檔大小的 3%～4% 左右，比較於其他學者所提出的壓縮法後(SBC、PDAT 以及 LBTC 壓縮法)，雖然我們的方法在解壓縮時間付出些微的代價，但我們所提出的壓縮法在壓縮率以及壓縮時間上的表現都較其他方法出色。

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壓縮方法與其解壓縮流程，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。

二、 相關研究

過去許多學者在追蹤檔壓縮上有豐碩的成果，如 Mache [5]，PDAT [6]，SBC [7]，以及 LBTC [8]等方法，以下我們分別介紹 PDAT、LBTC 與 SBC 三種追蹤檔壓縮方法。

PDAT：在此壓縮法中所使用的方法主要利用標頭來做前導，透過標頭檔中的資訊來告知跟隨在標頭後有哪些資料，在每道記錄的標頭後都帶有不等量的資訊，如位址偏移量、馬上被重複使用的次數。在標頭中則記錄著位址偏移量的編碼長度，以及記錄位址偏移量是否有重覆使用的情形。此壓縮法必須透過標頭來告知應該要如何抓取指令及解碼，雖然此方法在有連續數道相同位址偏移量的指令位址下可以節省相當多的壓縮空間，不過一旦沒有馬上可重複利用的偏移量的情形時，每筆記錄還是需要透過標頭來告知在某道指令後有多少道重覆且相同的位址偏移量，以及位址偏移量是用多少位元數記錄等，如此一來，標頭的存在就會造成空間的浪費，壓縮法的效益也就不會顯現出來。

LBTC：在 LBTC 中作者利用程式的區域性，並且在壓縮方法中加入快取的架構。此壓縮法是利用快取來輔助壓縮，作法是將指令放入快取中，如果在快取中已經有想要的指令以及資料的話，那麼就可以利用記錄其快取索引值來取代整道記錄指令以及位址。雖然此壓縮法能夠動態調整記錄位移量所需的位元數，使得記錄偏移量的空間達到最精簡狀態，不過此方法在每道記錄後面必須加上識別碼，以便解壓縮時告知該如何抓取資料解碼。若資料不存在快取當中，除了要記錄完整的指令以及指令位址之外，還要加上識別碼，如此一來，反而會使得每道記錄所需的空間變大而影響整體的壓縮率，造成了解壓縮時額外的負擔。

SBC：觀察了程式的追蹤檔後作者認為，程式中大部分的執行時間都花費在執行迴圈，因此追蹤檔中迴圈的資訊也相對占了相當的空間，如果可以縮減追蹤檔中迴圈的儲存空間，那麼將可以大幅縮減追蹤檔儲存時所需要的儲存空間。此壓縮方法將程式裡不同的程式片段編成不同的串列，而每種串列只需儲存一次，如：同樣的迴圈的資訊只需儲存一次即可。在 SBC 中作者將追蹤檔分成三個部分來記錄：串列索引值、存放指令的指令參考表，以及存放資料位址的資料參考表。在解壓縮過程中串列索引表指出

目前要處理哪個串列，而指令參考表提供此串列中有哪些指令，從這些資訊將得知在從抓取指令過程中，何時該到資料參考表中抓取資料。透過這些步驟來進行解壓縮的動作，並將壓縮過的追蹤檔還原。然而在 SBC 的壓縮方法僅適用於程式中具有大量迴圈，且這些迴圈重複執行的次數極高時，在我們的實驗結果也證明 SBC 壓縮法在一般的程式中壓縮效果不甚理想。

三、 追蹤檔壓縮方法

(一) 壓縮追蹤檔的設計

過去在處理器與記憶體的研究中，通常會透過追蹤檔來做模擬，但追蹤檔的檔案往往都很大，因此造成儲存上的困難，所以為了減少追蹤檔的大小，就勢必需要做追蹤檔的壓縮。而因追蹤檔是儲存程式執行的過程，在其中會有區域性的特徵，使用一般常見的壓縮方法並不能有效的減少其儲存空間。另外，為了研究上的方便，追蹤檔的壓縮方法必需能夠支援即時的解壓縮功能，而為了有效的提升解壓縮的時間，我們在設計上將採用固定長度的編碼方式，來降低解壓縮的複雜度。

在我們的壓縮方法中，將追蹤檔分為四個部分來儲存，分別是位址與指令對照表、循序位址表、壓縮表以及記錄檔來儲。以下我們將一一說明上述四個表的設計重點與原理。

1. 位址與指令對照表：對於追蹤檔中的指令，我們只需儲存一份位址對應到指令的參考表，並利用此參考表，即可還原出該道位址所執行的指令。程式執行時，是由處理器發出一道位址，並且透過該位址到記憶體中存取指令。而追蹤檔則是記錄程式的執行行為，其中每個欄位都是由位址與該位址所對應到的指令所組成，每道位址會對應到唯一一道的指令，所以只要儲存位址與指令的對應，那麼就可以透過位址來還原指令，所以可以節省大約一半的儲存空間。並且，因為指令的區域性並沒有位址來的強烈，所以我們希望利用此區域性來對位址做壓縮，來得到更好的壓縮率。
2. 連續位址的壓縮：根據過去的研究指出，跳躍指令約佔執行指令的 20% ~ 25%，其中這些跳躍指令，要做跳躍又佔 53%，所以總體來說，追蹤檔中會跳躍的指令只佔 10% ~ 13%的比例。換句話說，大部份的指令 (90% ~ 86%) 都是執行下一道指令，即是循序執行的指令，而下一道指令的位址即為 PC = PC+4。所以對於這些循序執行的指令，若我們能夠盡可能的減少其儲存的空間，那麼便可能很有效減少追蹤檔的大小。

基於上述的動機與分析，我們可以將追蹤檔的位址分成兩類，循序位址 (PC=PC+4) 與非循序位址，針對兩種不同的類型，我們分別使用了循序位址表與位址偏移量表記錄之。在壓縮時，若為循序位址，因位址與前一道位址相差固定皆為 4，所以我們只記錄 1 個 bit 來表示之，即在循序位址表記錄為 0。然而，若相差不為 4，則記錄為 1，如圖 1 所示。

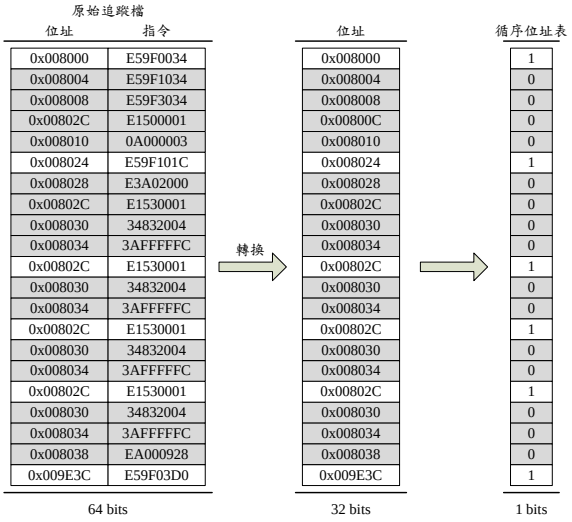

圖 1 產生循序位址表

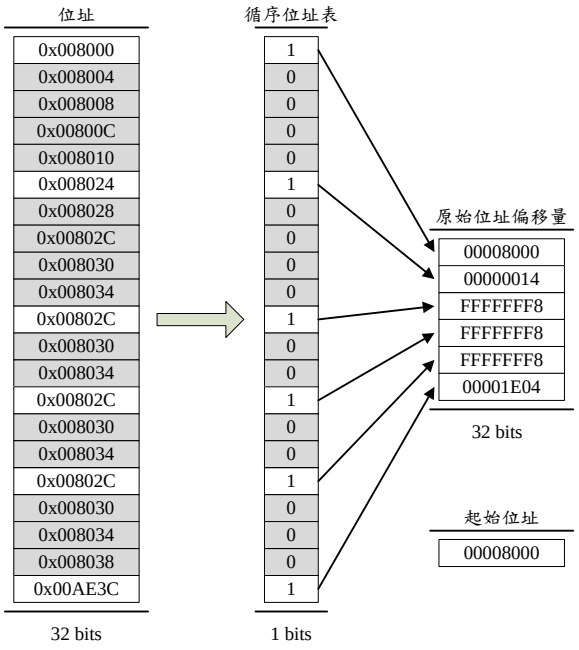

圖 2 變動位址表

對於循序位址，因與前一道位址所相差的偏移量皆為 4，所以我們可以

使用簡單的方式來表示之，然而，對於非循序位址而言，因位址偏移量相差的數值並不固定，所以必需記錄此偏移量值。在壓縮時，若為非循序指令時，將循序位址表當中記錄為 1，並且計算其位址與上一道位址的位址偏移量，再記錄到原始位址偏移量表當中，如圖二所示。而程式執行時，分支指令產生了非循序執行的情形，而分支指令可能會往前執行或往後執行，偏移值可能為負數，所以我們使用 2 補數的方式來表示正負數。

經由上兩步驟的壓縮，大約可達到的壓縮率計算方法如下：假設追蹤檔中有 C 道指令，所以追蹤檔共需要 2*C*32 bits 的空間 (2*C 為指令與位址)，假設非循序指令的有 K 道，經過我們的設計，所需的儲存空間為 (K*32+C)/2*C*32，因為 K 大約為 C 的十分之一，所以壓縮率約為 (0.1*32+1) /2*32 = 6.56%。

3. 減小位址偏移量表的大小：透過以上的分析我們發現，原始變動位址表中的容量大小佔了全部的四分之三左右 (k*32/(K*32+C))，如果可以減少原始位址偏移量表中記錄的長度，那麼將有助於壓縮率的提升。而在這一部份我們使用 16 位元的長度來取代記錄 32 位元的偏移值，並將之記錄在變動位址表中，如圖 3 所示。

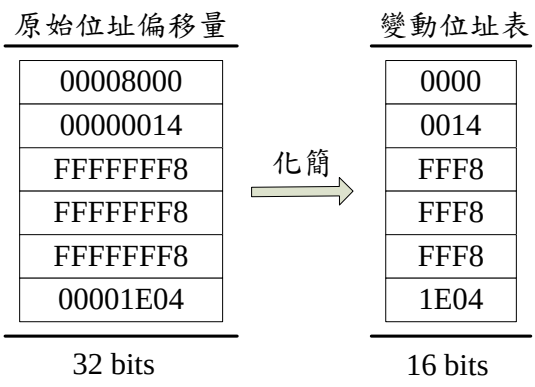

圖 3 減化位址偏移量的大小

透過過去學者研究指出，在執行 Spec92 的標竿程式，有 75%左右的指令只需要 12 位元的位址偏移量，將可以得到以下的壓縮率：(0.1*(3/4*12+1/4*12)+1)/2*32=4.218%，而將位址偏移量的記錄長度設定成 16 位元的話，則可以覆蓋到 99%的位址，並得到以下的壓縮率：(0.1*(0.99*16+0.01*32)+1)/2*32=4.088%。雖然使用 12 位元的位址偏移量可達到較好的壓縮率果，但是有 25%的位址無法使用 12 位元的長度來記錄，所以對這些位址必需額外增加標頭來區別，反而會造成壓縮效果不如預期。而使用 16 位元來記錄位址偏移量，則可包含 99%的位址偏移量，只有 1%的偏移量需要做額外的處理，經由此計算，壓縮率則可到達 2.5%。而對於無法使用 16 位元所表示的偏移值我們將在 3.4 節討論。

4. 利用動態參考表取代位址偏移量表：在第三部分我們已經將變動位址表的每個偏移量的寬度變小，但在圖 3 當中，我們可以發現，變動位址表中會記錄重複的位址偏移量，此一部分我們將要再利用其具有的區域性，來進一步的縮短位址偏移量表的大小。

位址偏移量表當中記錄著非循序位址的偏移量，若遇到迴圈，則會產生大量重複的位址偏移量。比如說如果迴圈執行 1000 次，每當執行到迴圈的最後一道指令時，會跳回迴圈中的第一道指令執行，所以產生了 1000 個同樣的位址偏移值，於是，於是在變動指令的位移表上面就會存在許多相同的位移量，雖然可以統計這些出現比較多的位移量，將之放入到字典表中，然後在變動指令表內以字典表的索引取代，然而，當不同偏移量的數目增多的時候，其所要表示的索引值將會越大。這樣的結果將會造成不佳的壓縮率。所以我們提出使用動態索引的方式，在壓縮的過程中，變更索引值所代表的位址偏移量，用以縮減索引值過大的問題。

在動態參考表的壓縮方法中，我們會將變動位址表壓縮成兩個部分來儲存，分別是壓縮表以及記錄檔。壓縮表當中的第 1 個位元用來記錄動態參考表的使用狀態，0 代表失誤、1 代表命中，第 2、3 個位元用來儲存動態參考表的欄位索引。而記錄檔則依序記錄著在動態參考表中失誤的位址偏移量。以下我們以圖 4 當範例逐步介紹我們壓縮的過程，在範列中動態參考表可放入 4 個欄位，使用完全關聯度、隨機取代的策略：

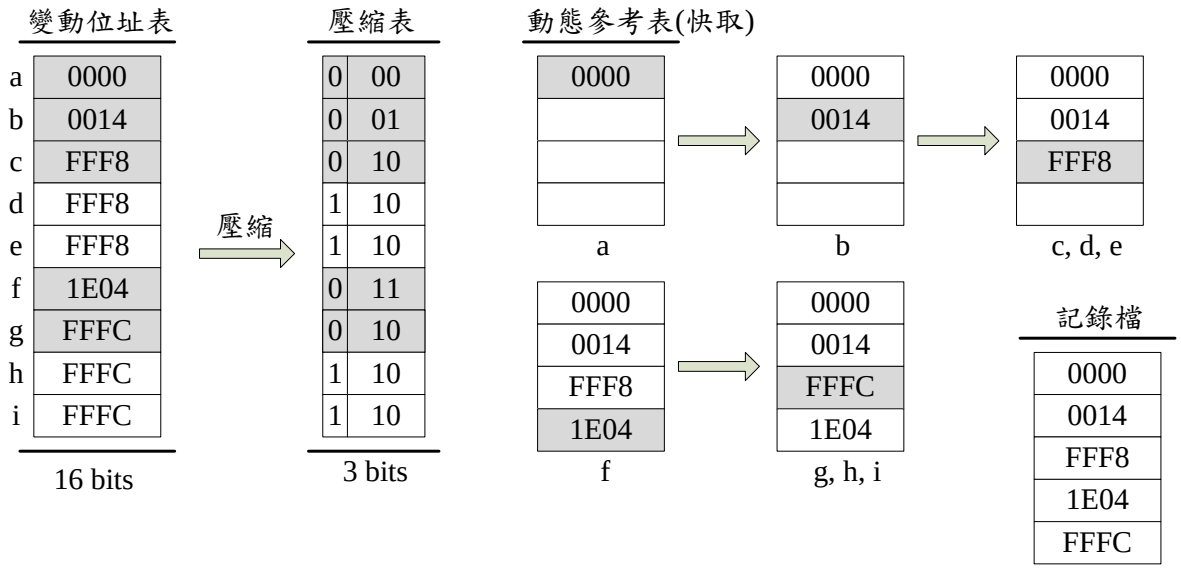

圖 4 動態參考表示意圖

- Step a、b、c：動態參考表的初始狀態為空，屬於強迫性失誤，在壓縮表記錄 0 表示為失誤，偏移值依序填入到動態參考表的 00、01、10 的欄位，並記錄在壓縮表中的第 2、3 個欄位。另外，將位址偏移量 0000、0014、FFF8 依序儲存到記錄檔中。
- Step d、e：在動態參考表中命中，在壓縮表記錄為 1，且記錄其在第 10 的欄位命中。
- Step f：屬於強迫性失誤，將偏移值填入到 11 的欄位，在壓縮表中記錄 011，並且將位址偏移量 1E04 儲存到記錄檔中。
- Step g：屬於強迫性失誤，因動快參考表已滿也引發容量性的失誤，若隨機取代策略選擇欄位 10 來取代，則會將位址偏移量 FFFC 放入，壓縮表中記錄 010，表示此偏移值為失誤且放在動態參考表 10 的欄位。另外將偏移值放入記錄檔中。
- Step h、i：在動態參考表中命中，在壓縮表記錄 110。

在上述的動作中，動態參考表欄位的多寡將影響索引的長度以及記錄檔的大小。當動態參考表欄位的數目增多時，雖然命中率可以提升而降低記錄檔的大小，但相對的索引的寬度增加，卻需要儲存較大的壓縮表，反之亦然。為了得到最佳的動態參考表欄位數目，我們會對各個欄位數進行模擬。

(二) 壓縮流程

我們提出的壓縮方法會將追蹤檔分成四個部分記錄，分別是位址與指令對照表、循序位址表、壓縮表以及記錄檔，其壓縮的流程如下：

1. 透過靜態程式碼，產生位址與指令的對照表。
2. 計算位址是否為循序指令，若為循序位址則在標示為 0，若遇到非循序位址則標示為 1，並將之儲存到循序位址表中。
3. 對於非循序位址，在儲存循序位址表的同時，計算其 16 位元的位址偏移量，並接續壓縮流程 4。
4. 尋找動態參考表中是否已有此筆偏移量，如果有，就在壓縮表中記錄 1 與動態參考表欄位的索引值。如果動態參考表中沒有這筆資料，就將此筆偏移量依照取代的策略放入動態參考表的欄位中，並且在壓縮表中記錄 0 與所放置的欄位索引，最後將此筆偏移量放入變動位址表中儲存起來。

(三) 解壓縮流程

在我們的設計當中，變動位址表 (如圖 2 所示) 記錄著每道非循序指令與其上一道指令之間的位址偏移量，動態參考表在解壓縮的過程扮演著類似快取的角色，在動態參考表當中儲存著最近被使用到的幾個位址偏移量，而壓縮表則告訴我們目前欲解壓縮的指令其位址偏移量是否存在於動態參考表當中，並清楚的指出該指令的位址偏移量存在動態參考表的第幾個欄位。除了一般的批次解壓縮外，這個解壓縮方法也支援了在執行時的動態解

壓縮，以下，我們將依序的說明本論文中所提出的解壓縮流程：

1. 參考圖 5(a)，先循序讀取壓縮表，若目前壓縮表欄位中的資料開頭為 0 表示該位址偏移量不存在動態參考表當中，此時我們依據壓縮時所產生的紀錄檔(見圖 4)依序載入一個位址偏移量至動態參考表的欄位中(壓縮表欄位中第一位元以後的值即為此該位址偏移量在動態參考表中的索引值)，並將此位址偏移量依序記錄於位址偏移量表當中。
2. 若壓縮表欄位中資料開頭為 1 表示該位址偏移量存在動態參考表當中，我們依據壓縮表欄位中的索引值至動態參考表中提取出位址偏移量並記錄於位址偏移量表中。
3. 當位址偏移量表產生完成後，就可以依據循序指令表以及位址偏移量表來還原指令位址，此時參考圖 5(b)解壓縮步驟，依序讀取循序指令表，當讀取到第一個 1 時，我們將位址偏移量表中第一個位址偏移量與起始位址相加並將結果記錄於指令位址表當中，之後，若在循序位址表中讀取到 1，則依據將位址偏移量表中的位址偏移量與指令位址表中最後一道位址相加，並且將結果記錄於指令位址表當中。
4. 若在循序位址表中讀取到 0，表示此道指令與前一道指令的位址連續，此時只需要將指令位址表中最後一道指令位址加 4 並記錄於指令位址表中即可。
5. 在還原指令位址之後，就可以透過指令位址與指令的對應來還原出原始的追蹤檔。

(四) 狀態模式

在實驗過程中我們發現，Media Bench 套件中，cjpeg、pegwit_d 及 pegwit_e 標竿程式的追蹤檔裡含有某些指令的位址偏移量需要超過 16 位元的記錄長度，對於這些指令我們發現，需要 20 位元才能夠完整正確的紀錄其位址偏移量，使得解壓縮程序成正確執行。為了區別不同長度的位址偏移量，我們提出了三種模式來處理此現象，動態調整模式 (Dynamic)、常態模式 (Normal) 與特別 (Special) 模式，在這三種模式下，壓縮方法大致雷同，其主要的差別在於使用不同的位元長度來處理位址偏移量。

對於程式中大多數的指令變動位址在 16 位元以內，我們使用動態調整模式，在動態調整模式下，對於位址偏移量仍然以 16 位元為主，若有指令其位址偏移量需要大於 16 位元表示，則在此指令之前一道位址偏移量後加入識別碼來告知，因此對於不同長度的偏移量可以使用不同的辨識碼加以區別。在解壓縮過程中，若抓取到附帶有辨識碼的位址偏移量，則表示下一道指令需要抓取 20 位元並加以解碼，反之，則直接抓取 16 位元進行解碼。

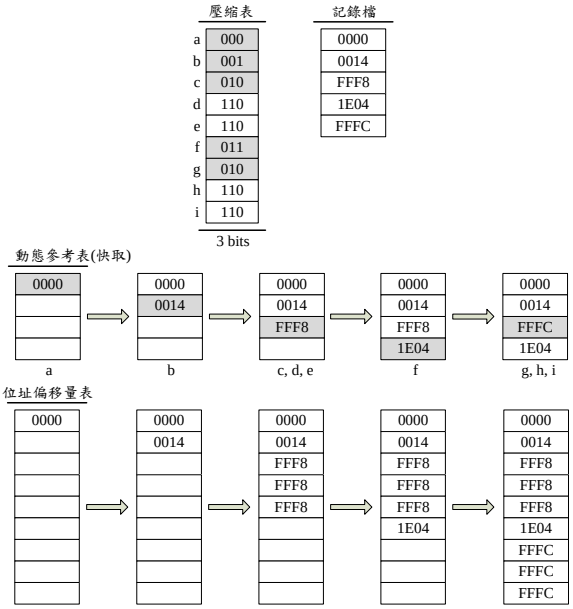

圖 5 解壓縮流程圖 (a)

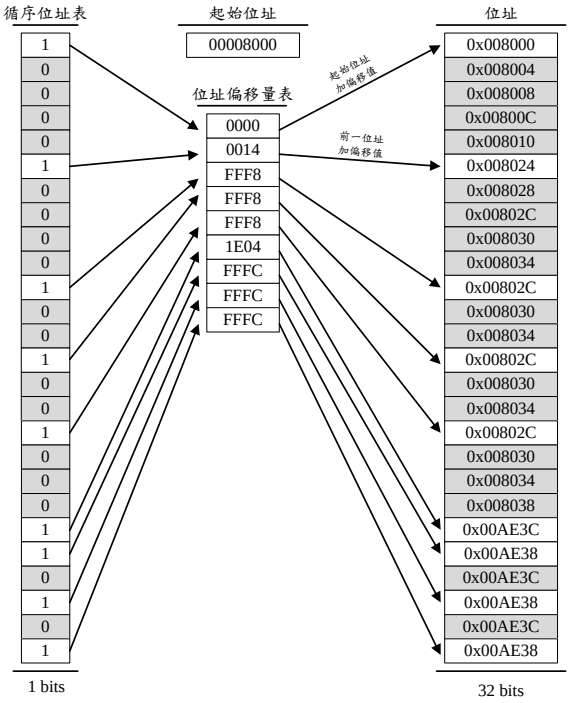

圖 5 解壓縮流程圖 (b)

當程式中指令的位址偏移量大多是超過 16 位元能表示時，則使用常態模式，在常態模式下每道指令的位址偏移量都以 20 位元表示。

在特別模式下，我們將 cjpeg，pegwit_d 以及 pegwit_e 這三個標竿程式的追蹤檔做特別處理，在

這些標竿程式的追蹤檔我們以 20 位元來記錄位址偏移量，而其他的標竿程式的追蹤檔則用 16 位元處理位址偏移量。

四、 實驗模擬

在追蹤檔壓縮上，我們主要的觀念是希望藉由記錄動態參考表索引值取代記錄位址偏移量，並且利用動態參考表區域性的特質來減少相同位址偏移量出現的次數來達到節省儲存空間，以此提高壓縮效果。以下將介紹我們的模擬環境、實驗參數以及實驗結果。在實驗結果裡包括了我們所提出的壓縮法 LAAO 在三種模式 (動態、常態、特別模式)下的壓縮率及壓縮時間的評比，另外，在實驗中也將我們所提出的壓縮方法與其他壓縮法在壓縮率、壓縮時間以及解壓縮時間進行比較。

(一) 模擬環境

我們所使用的模擬環境如下，處理單元為兩個 Intel Xeon 3.4 800MHz CPU 內含 2MB L2 快取記憶體，主記憶體容量為 3GB DDR2-400 ECC Registered，在儲存空間方面為 73G SCSI HD 並使用 Raid-0 的磁碟陣列，而本次實驗環境中的作業系統為 Windows Server 2003。

(二) 標竿程式

在本次的實驗我們採用 Media Bench 中的 cjpeg 、 g721decoder 、 g721encoder 、 pegwit_d 、 pegwit_e、sorts 以及 timing 作為標竿程式並以 ARM STD2.5 來模擬 ARM 系統的工作環境。實驗的流程如下：首先，我們將標竿程式導入 Strong ARM STD2.5 中，編譯器會將程式編譯成 ARM7 所能夠執行的靜態程式碼，利用分析此靜態程式碼可以得知整個程式未壓縮前的空間大小，我們使用 ARM STD 2.5 的 Debugger 執行前一步所產生的靜態程式碼，並產生追蹤檔。

(三) 實驗結果

以下的實驗將對我們所提出的方法中三種模式進行模擬，針對 Media Bench 中的七個標竿程式在不同的動態參考表欄位數下進行壓縮率及壓縮時間的模擬。

在動態模式下，位址偏移量仍然以 16 位元為主，若有指令其位址偏移量需要大於 16 位元表示，則在此指令之前一道位址偏移量後加入識別碼來告知，在圖 6 中，我們發現到當動態參考表欄位數為 32 時表現最為突出，平均壓縮率達 3.189%，而在此圖 5 中，可以發現又以 pegwit_e 的壓縮率最出色。在壓縮時間方面，如圖 7 所示，我們可以發現，隨著動態參考表欄位數的增加，壓縮時間也隨之增加，這是因為動態參考表欄位數增加後，比對位址偏移量的時間也會增加。另外，雖然壓縮時間會隨著動態參考表欄位數增加，綜合圖 6 以及圖 7 可以發現，當動態參考表欄位數為 32 時，整體來說不論在壓縮率或壓縮時間都有最好的表現。

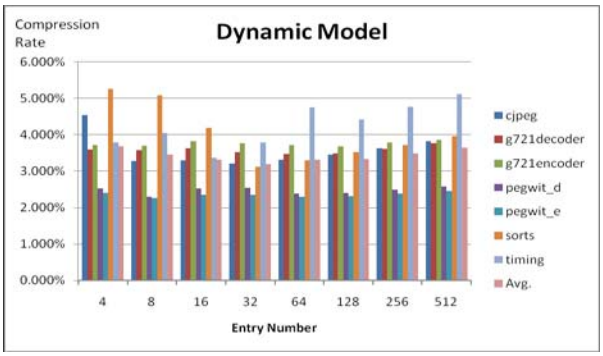

圖 6 在動態模式下的壓縮率

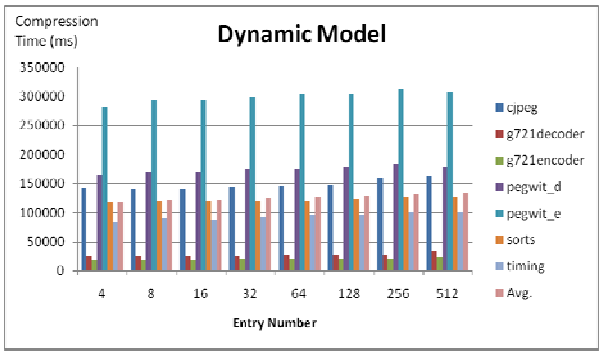

圖 7 在動態模式下的壓縮時間

常態模式下，每道指令的位址偏移量都固定以 20 位元記錄，圖 8 與圖 9 分別為本論文的壓縮方法在常態模式下對於不同的動態參考表欄位數所造成的壓縮率以及壓縮時間，與動態模式相同的，無論是在壓縮率或是壓縮時間，都是以在動態參考表欄位數為 32 時表現最為出色。

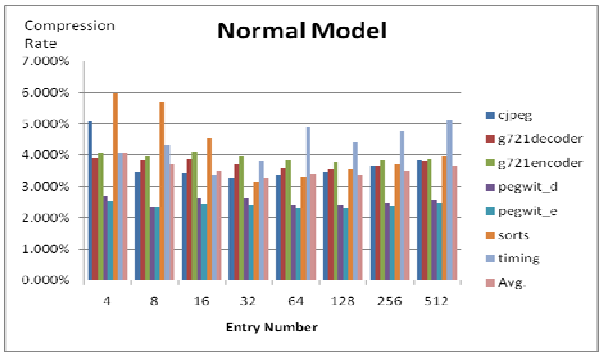

圖 8 常態模式下的壓縮率

在特別模式中，我們僅對 cjpeg，pegwit_d 以及 pegwit_e 這三個標竿程式的追蹤檔做特別處理，以 20 位元來記錄位址偏移量，而其他的標竿程式的追蹤檔則用 16 位元處理位址偏移量，圖 10 與圖 11 顯示了特別模式下所產生的壓縮率以及壓縮時所需要的時間，對於不同的標竿程式同樣是在

變動位址表欄位數為 32 時有最佳表現。

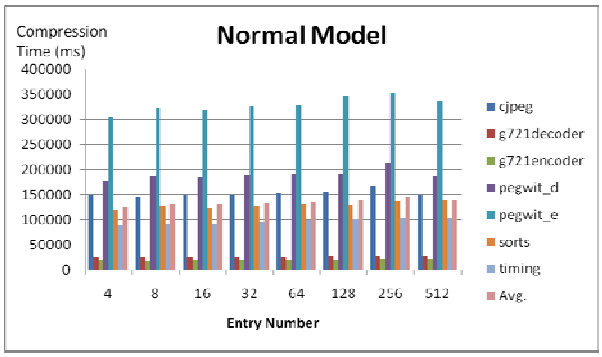

圖 9 常態模式下的壓縮時間

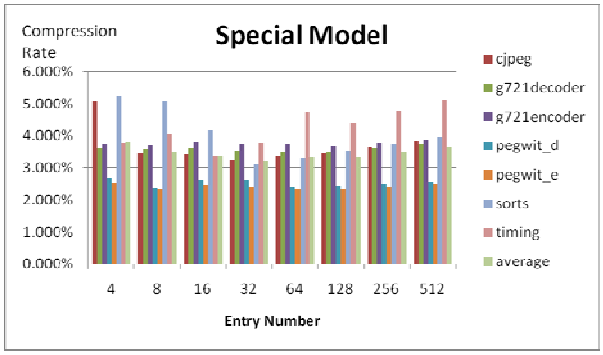

圖 10 特別模式下的壓縮率

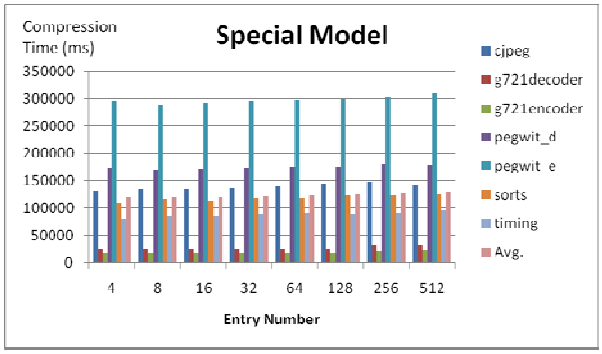

圖 11 特別模式下的壓縮時間

為了使此壓縮方法在程式中具有超過 16 位元偏移量時可以正常運行，我們提出了三種模式來解決，從以上的實驗我們可以發現在三種模式下各種動態參考表欄位數對於壓縮率以及壓縮時間的影響，以下我們將比較平均而言，在不同的模式下對於壓縮率以及壓縮時間所造成的影響。

圖 12 為三種模式壓縮率的比較圖，在動態模式下，只有在真正需要 20 位元的位址偏移量才會使用 20 位元表示，因此壓縮後可以獲得最佳的壓縮率。在特別模式中，若程式裡具有超過 16 位元偏移量則該程式中的位址偏移量完全使用 20 位元表示，其餘程式仍然維持使用 16 位元表示，因此壓縮率較動態模式差。而在常態模式下，由於任何偏移量皆由 20 位元表示，在程式中大多數的偏移量卻是小於 16 位元，因此，此三種模式中，常態模式所得到的壓縮率最高。圖 13 中比較了三個方法在壓縮時間上的差異，從圖中我們可以發現，這三種模式會造成每次存取位址偏移量的位元數不同，因此在全部位址偏移量皆為 20 位元的常態模式下，需要最長的壓縮時間。

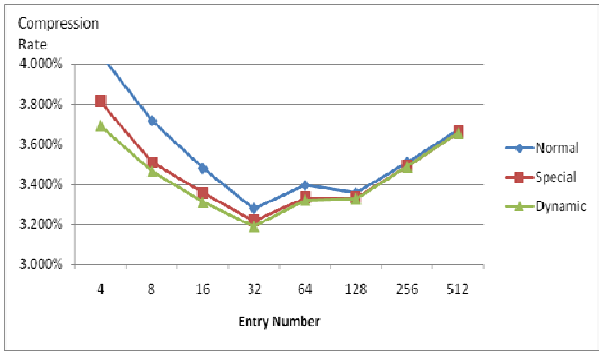

圖 12 三種模式的壓縮率比較圖

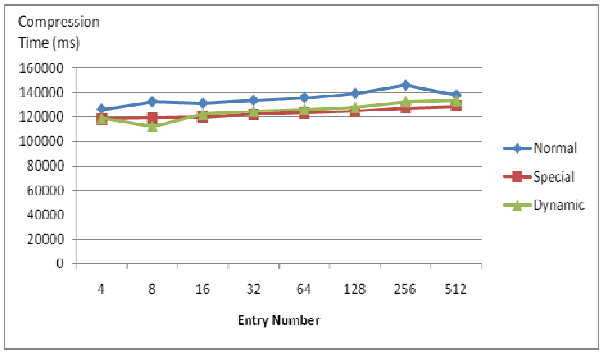

圖 13 三種模式下的壓縮時間比較圖

在比較本論文方法中的三種模式後，以下我們將就本論文所提出的壓縮方法與過去其他的壓縮方法(PDAT，LBTC，SBC)進行壓縮率、壓縮時間以及解壓縮時間的比較。

在圖 14 中，同樣的我們使用 Media Bench 中的六個標竿程式進行壓縮率方面的比較，從圖中可以發現，我們所提出來的壓縮方法在各個標竿程式中都達到最佳的壓縮率表現。而在 LBDT 的壓縮率偏高的原因，如同之前所提，因為 LBDT 需要在每道壓縮的位址後面加上識別碼，並且如果沒有在快取命中的情形下，就必須用記錄整道位址以及指令，所以此壓縮效果比較不理想。另外，SBC 壓縮法在壓縮率上整體表現皆不甚理想，其原因是此壓縮法較適合用在迴圈數量多且每個迴圈會被多次執行的程式上，所以對於迴圈數較少的標竿程式就難達到較好的壓縮率，值得注意的是，在 pegwit_d 以及 pegwit_e 這兩個標竿程式當中具有較多執行頻率較高的迴圈，因此，對於這樣的標竿程式 SBC 壓縮法就能達到較好的壓縮效果。

判斷一個壓縮方法的優劣，除了從壓縮法所能夠達到的壓縮率來判斷以外，壓縮時的效率也是非常重要的因素。然而，希望達到好的壓縮率經常需要犧牲掉壓縮時間或解壓縮時間。比較圖 14、圖 15 我們可以發現，雖然在這四個壓縮方法當

中，本論文提出的壓縮方法具有最佳的壓縮率，但在平均壓縮時間上的表現也是最佳。

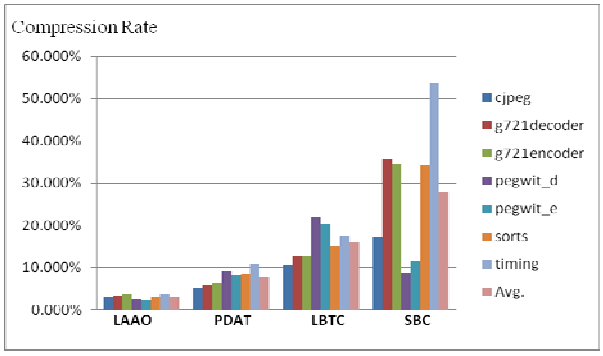

圖 14 各方法壓縮率比較圖

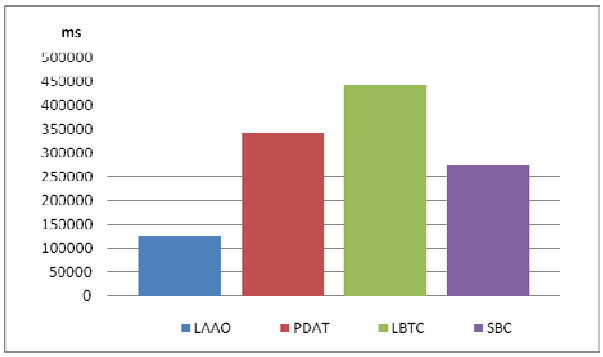

圖 15 各方法的平均壓縮時間比較圖

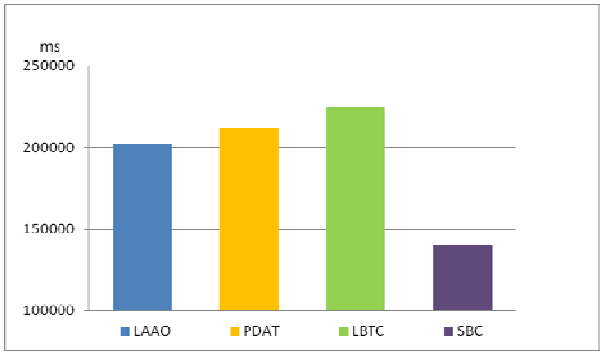

圖 16 平均解壓縮時間

在圖 16 中，雖然我們的解壓縮時間表現不及壓縮率最差的 SBC 壓縮法，但仍然較 PDAT 及 LBTC 壓縮方法來的出色，綜合圖 13、圖 14 及圖 15，在四個壓縮方法中，雖然我們犧牲一些解壓縮時間，但換來的卻是在壓縮率以及壓縮時間上都得到最佳的表現。

五、 結論

追蹤檔的壓縮跟一般壓縮不相同，因為程式執行具有區域性的關係，如果忽略區域性特質，而單純以傳統壓縮法的話，將無法得到很非常有效的壓縮效果。透過追蹤檔的分析，我們發現追蹤檔中指令的位址也是具有相當的區域性，因此，在本篇論文中我們主要透過區域性的特質來壓縮追蹤檔。在方法中我們僅記錄非循序指令的位址偏移量，並透過動態參考表的使用來減少位址偏移量的重複記錄。透過其他研究知道我們得知在大多數的成是當中 16 位元就可涵蓋 99%的位址偏移量，因此我們僅紀錄 16 位元或 20 位元來縮減位址偏移量的記錄長度。實驗結果說明，本論文中的壓縮方法約能將追蹤檔壓縮至原始檔案大小的 3.189%左右。

六、 參考文獻

[1.] A. Agarwal and M. Huffman, “Blocking: Exploiting Spatial Locality for Trace Compaction,” Proc. ACM SIGMETRICS, 1990.

[2.] E.E. Johnson and C.D. Schieber, “Real-Time Address Trace Compression Hardware for Extended trace,” Performance Evaluation Review, vol. 21, nos. 3-4, pp. 22-32, Apr. 1994.

[3.] S. Dad and E.E. Johnson , “Accuracy of filtered Trace,” Proc. IEEE Int’l Phoenix Conf. Computers and Comm., pp. 82-86, Apr. 1995.

[4.] J.W.C. Fu and J.H. Patel, “Trace Driven Simulation Using Sampled Traces,” Proc. 27th Ann. Hawaii Int’l Conf. System sciences, pp. 211-220, 1994.

[5.] A.D. Samples, “Mache: No-Loss Trace Compaction,” Proc. ACM SIGMETRICS 1989, pp. 89-97, 1989.

[6.] E. Johnson, J. Ha, and M.B. Zaidi, “Lossless Trace Compression,” IEEE Trans. Computers, vol. 50 no. 2, pp. 158-173, Feb. 2001.

[7.] A. Milenkovic and M. Milenkovic, “Stream-Based Trace Compression,” Computer Architecture Letters, vol. 2, Sep. 2003.

[8.] Yeu Luo, and Lizy Kurian John, “Locality-Based Online Trace Compression,” IEEE Transformations on Computer, vol.53, no.6, June 2004.

[9.] Martin Burtscher, “VPC3: A Fast and Effective Trace-Compression Algorithm,” SIGMETRICS/Performance 4, June12-16, 2004 ACM, New York, USA.

[10.] Martin Burtscher, and M. Jeeradit, “Compression Extended Program Traces Using Value Predictors,” International Conference on Parallel Architectures and Compilation Techniques, pp.159-169, Sep. 2003.

[11.] Martin Burtscher, and B. G. Zorn, “Exploring Last n Value Prediction,” International Conference on Parallel Architectures and Compilation Techniques, pp.66-76, Oct. 1999.

[12.] Martin Burtscher, and B. G. Zorn, “Hybird Load-Value Predictors,” IEEE Transformations on Computers, vol. 51, no.7, pp. 759-774, July 2002.