在 3SOC 上的平行移動估測

Parallel Motion Estimation on the 3SOC

林彥君 (Yen-Chun Lin) 熊全達 (Chuan-Da Hsiung)
Dept. of Computer Science and Information Engineering
National Taiwan University of Science and Technology, Taipei, Taiwan 106
Email: yclin@et.ntust.edu.tw Email: cdhsiung@giga.net.tw

摘要

移動估測是視訊壓縮的重要過程，它能大幅減少視訊的資料量，但需要龐大的計算量。本論文以 3SOC 針對移動估測演算法進行平行處理。3SOC 是一個可程式控制、可擴充、多處理器的單晶片系統。我們安排 3SOC 中不同類的處理器分別執行適合該類處理器的工作。我們的設計適用於大部分移動估測演算法，而且能讓各個處理單元保持負載平衡。實驗結果顯示，在處理單元數不超過 8 的情況下，我們能得到很好的加速。在實際以 3SOC 製作視訊壓縮器時，可以使用更多的處理單元，更快完成。我們的設計屬軟體實作法，容易與其他部分整合成完整的視訊壓縮器，也容易被修改來符合不同的視訊壓縮標準，用在不同的視訊應用。

關鍵詞：移動估測、視訊壓縮、平行處理、3SOC、單晶片系統

Abstract

Motion estimation (ME) is critical to video compression. It can greatly reduce the video data amount but requires enormous computation. This paper investigates parallel execution of ME on the 3SOC, a programmable and scalable multiprocessor system-on-a-chip. We analyze some popular fast ME algorithms and assign suitable tasks to different processors. Our design can be applied to a number of ME algorithms and can achieve load balancing. Experimental results show that we can obtain very good speedup when the number of processing units does not exceed 8. When implementing a real 3SOC video encoder, more processing units can be used to obtain greater speedup. Our design can be easily integrated into a 3SOC video encoder and modified to meet different video coding standards.

Keywords: motion estimation, video compression, parallel processing, 3SOC, system-on-a-chip

一、簡介

移動估測(motion estimation)是視訊壓縮(video compression)的關鍵部分。它貢獻了主要的壓縮率，但它的計算量龐大，約佔整個視訊壓縮器(video encoder)的 60-80% [20, 30]。速度是視訊壓縮器實作上的一個重要考量，特別是對於一些有即時(real-time)需求的應用，如視訊會議(video conferencing)、數位電視廣播(digital TV broadcasting)等。

視訊壓縮器可以用硬體或軟體來實作。硬體的實作法採用特製的架構，特別是移動估測的部分，許多人做成 ASICs 來提升速度[15, 20, 23]。它們通常以某個演算法為目標做最佳化，因此執行速度相當快。但是它們缺乏彈性，不適合用在其他的應用或是滿足未來的標準[6]，設計的技術門檻高，所需的時間也長[11]。除此之外，有些快速移動估測演算法因為缺乏規律性，不容易以硬體來實作[20]。

軟體實作法能夠克服硬體實作法的種種缺陷。但是，視訊壓縮龐大的計算量常使得單一處理器的電腦難以負荷。因此，有必要利用平行處理來減少處理時間。有人探討以平行電腦 [6, 7, 25] 或是工作站叢集 (cluster of workstations) [8, 16]為計算平台。這些平行計算系統常常所費不貲。

本論文提出了利用 Software Scalable System On a Chip (3SOC) [14]執行移動估測的方法。3SOC 是一個可程式控制(software programmable)、可擴充(scalable)、多處理器的單晶片系統(system-on-a-chip)。3SOC 可被看成是共享記憶體(shared memory)的平行架構，它採用了許多類似 RISC、DMA unit 的處理器和數位信號處理器，使得開發者能夠以軟體來實作應用，透過平行處理在高效能的晶片上執行。稍後將會對 3SOC 有更多介紹。

我們利用了視訊資料中的資料平行(data parallelism)，設計出額外負荷(overhead)很少、負載平衡(load balancing)佳的資料分配方法，分配視訊資料給各個處理單元執行。我們研究了諸多移動估測演算法，找出它們的共通性，將各種不同的工作分配給3SOC 中適合的處理

器來執行。我們的設計適用於大部分的移動估測演算法，而實驗結果顯示我們的設計能得到很好的加速(speedup)。由於 3SOC 是可程式控制的，我們的設計容易與其他部分整合成完整的視訊壓縮器，也容易被修改來符合不同的視訊壓縮標準，用在不同的視訊應用。

二、移動估測

2.1. 移動估測的介紹

移動估測意圖消去原始視訊資料(source video sequence)中時間上的多餘資料(temporal redundancy) [26]。它假設視訊資料中，相鄰畫面的差異是源自影片中物體的移動。因此，我們希望只記錄這些物體的位移，藉此減少資料量。

移動估測首先將目前畫面(current frame)切割成一群互不重疊、邊長為 B 個畫素(pixels)的正方形區塊，常用的 B 值為 16，這個尺寸的區塊被稱為 macroblock (MB)。針對每個 MB，在其參考畫面(reference frame)所對應的搜尋範圍(search window)中，找出一個最為相似的 MB，稱作 best match，並記錄這兩個 MBs 之間的位移，即移動向量(motion vector)，與兩個 MBs 對應位置畫素值的差異，即預測誤差矩陣(prediction error matrix)。參考畫面是過去或未來的畫面，經壓縮再解壓縮得來。解碼器(decoder)根據移動向量自參考畫面取得對應的 MB，再加上預測誤差，即可重建該 MB。這種以區塊比對(block matching)為基礎的方式已被許多視訊壓縮標準所採用[1-5]。

兩個 MBs 之間失真度(distortion)的檢驗被用來判斷它們相似的程度。檢驗失真度的一個簡單、有效、並被廣泛使用的方法是計算兩個 MBs 之間的 SAD (也被稱作 MAD [19])值，通常被定義如方程式(1)。式中 $I_t(m, n)$與 $I_{t-1}(m, n)$分別表示目前畫面與參考畫面位於座標(m, n)之點的畫素亮度值(intensity value)。(1)式計算了目前畫面中位於(x, y)的 MB 與參考畫面中位於$(x + u, y + v)$的 MB 之間的 SAD 值。在參考畫面中，搜尋範圍裡的每個 MBs 我們稱為候選(candidate) MBs。移動估測就是針對目前處理的 MB，在一群候選 MBs 中找出一個失真度為最小的 best match。

2.2. 移動估測演算法的介紹

移動估測最直接的作法，就是地毯式地檢驗搜尋範圍中所有的候選 MBs，稱為完全搜尋(full search)。完全搜尋保證能找到 best match，但需要最多的計算量。

相對於完全搜尋，許多快速移動估測演算法，或被稱作快速搜尋(fast searches)被發明出來。其中大部分對視訊資料作了 unimodel error surface 的假設：越接近 best match 的 MB，失真越小。這類演算法可以檢驗較少的 MBs，以逐步逼近的方式來找 best match。快速搜尋大幅地減少了視訊壓縮器的負擔。但是，unimodel error surface 的假設並非一定成立，快速搜尋可能找到 local minima，而造成畫質與壓縮率的降低。因此，這類演算法發展的重要目標，是要兼顧快速和準確度。這一類的演算法有 three-step search (TSS) [19]、new three-step search [21]、four-step search [24]、diamond search [31]、PMVFAST [27]、hexagonal search [30]等。

另外，一個 MB 的移動向量常相似於它附近的 MBs。因此，一個 MB 的移動向量可以由附近已處理過的 MBs 的移動向量來猜測，這概念稱為 motion vector prediction [9, 10, 17, 22, 27, 29]。以預測的結果作為搜尋的起點，配合一些快速搜尋來使用，常能夠較快逼近 best match，也減少找到 local minima 的機會。被用以預測移動向量的 MBs 稱為 predictors，常用的 predictors 有前一個畫面相同位置的 MB 以及目前 MB 的上面、右上、左邊三個 MBs。

為了加快移動估測的計算，有人將完全搜尋與較為簡單的 TSS [19]分別用 VLSI 製作[18, 20, 23, 28]。但是，許多快速搜尋是針對單一處理器設計的循序式(sequential)演算法，且控制流程複雜，資料流不規則，不容易以硬體來實作[20]。

三、3SOC 的介紹

圖 1 是 3SOC 架構簡圖。3SOC 是由一個或多個 Quads 所組成，每個 Quad 都是一個處理器叢集。Quads 之間是由一條高速的全域匯流排(global bus)相連。晶片中有可程式控制的 I/O 系統(programmable I/O system)，由一個特殊的 I/O Quad、兩條 I/O 匯流排與連接 I/O pins 的邏輯電路所組成，允許開發者以軟體製作大部分的 I/O 裝置，如一些高效能的 PCI、Ethernet、SCSI 介面。晶片中有一個 DRAM controller，負責 Quads 與外部記憶體(external DRAM)之間的資料傳遞。晶片中有 64 個全域號誌暫存器(global semaphore registers)，提供了 Quads 之間的同步(synchronization)機制。

$$SAD_{(x,y)}(u,v) \equiv \sum_{j=0}^{B-1}\sum_{i=0}^{B-1}\left|I_t(x+i,y+j) - I_{t-1}(x+u+i,y+v+j)\right| \quad (1)$$

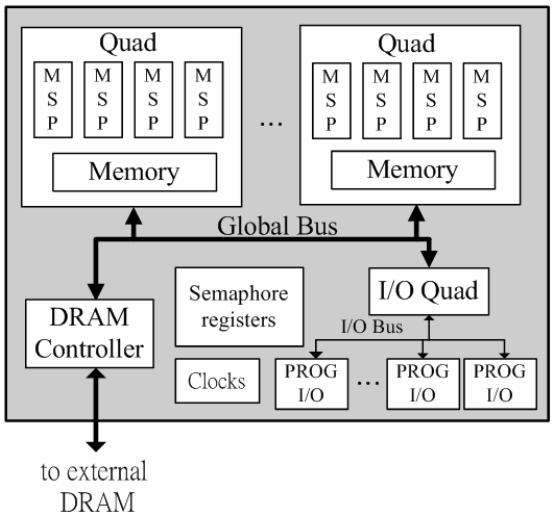


圖 1. 3SOC 架構簡圖

Quad 的架構如圖 2 所示。Quad 中有四個 Media Stream Processors (MSPs)。MSP 是由一個 Processing Engine (PE)和兩個 Digital Signal Processing Engines (DSPEs)所組成。Quad 中的共享記憶體包含了 32 KB 的程式記憶體(program memory)及 64 KB 的資料記憶體(data memory)。Quad 中有一個 Memory Transfer Engine (MTE)，它是 DMA controller，可被用來進行 Quad 共享記憶體與外部記憶體之間的資料傳輸。MTE 有 4 個程式計數器(program counters)，因此可以同時處理 4 份資料傳輸的請求。此外，Quad 中有兩條 64-bit 的 local 匯流排，分別為指令匯流排和資料匯流排；指令匯流排連接了 PEs、MTE 與程式記憶體，資料匯流排連接了 PEs、DSPEs、MTE 與資料記憶體。Quad 中有 32 個區域號誌暫存器(local semaphore registers)來負責同步。

PE 是 RISC-like 處理器，執行速度在 225 MHz 時大約是 45 MIPS。它的指令來源是 Quad 中的共享程式記憶體，每次抓取指令(instruction fetch)可以取得 4 個指令。PE 負責控制 DSPEs 的運作，也能夠透過控制 MTE 來移動資料。

DSPE 是 3SOC 中主要的計算引擎，它是 32-bit 的數位信號處理器，速度在 225MHz 時大約是 225 MIPS。它有 512 KB 的資料記憶體和可存放 512 個指令的程式記憶體。它有一個 floating point multiplier-accumulator，能夠在一個時脈週期(clock cycle)之內執行 16 個 8-bit 的運算，或 4 個 16-bit 的運算，或三個浮點數的運算(浮點數轉換、浮點數乘法、浮點數累加)。因此，在處理浮點數運算時，DSPE 能達到 675 MFLOPS。DSPE 還有兩個 16-word 佇列(FIFOs)，是 DSPE 與 Quad 共享資料記憶體之間的緩衝區(buffer)，使得資料可以被預先抓取(pre-fetch)與延後寫入(post-write)，讓 DSPE 能保持快速執行。

3SOC 上的軟體開發是透過類似 C 語言的組合語言 CLASM (C-Like Assembly) [14]。在 PE 上執行的程式，還可以用標準的 ANSI C 語言來開發。3SOC 有在 MS Windows 作業系統的模擬器，稱作 Inspector [13]，它有圖形介面的除錯環境與一些效能分析工具。

四、在 3SOC 上的平行移動估測

4.1. MSPs 之間的工作安排

正如前述，移動估測每次針對目前畫面中的一個 MB，在參考畫面所對應的搜尋範圍內，找出一個最相似的 MB。在此過程中，目前畫面中的每個 MB 都被看成彼此獨立、不具資料相依性(data dependency)的資料。我們可以利用這個特性，將各個 MB 分配給各個 MSP 來同時進行移動估測。

我們採取類似 processor farm [12]的運作方式，任取一個 MSP，命名為 MSP_0，當作 controller。MSP_0 在每個畫面開始處理之時，會針對目前畫面建立一個 job list，裡面記錄了此畫面中尚未處理的 MBs。接著，所有的 MSPs 自行從 job list 抓取尚未處理的 MB 來工作。當一個 MSP 對一個 MB 的移動估測完成之後，它會將所求得的移動向量和預測誤差矩陣存入記憶體。

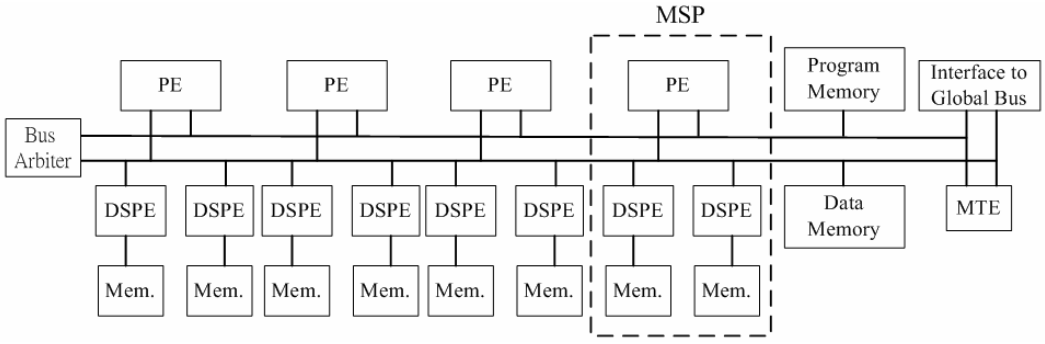


圖 2. Quad 架構圖

Job list 在實作上只需一個整數型態的變數 next_job 即可完成。舉例來說，我們可以對目前畫面中的 MBs 從 0 開始循序編號，並以 next_job 來儲存下一個待處理的 MB 的編號。在每個畫面開始處理之時，next_job 被 MSP_0 設成 0。接著，每個 MSP 將反覆地主動讀取 next_job，依編號處理對應的 MB，並更新(遞增) next_job。直到 next_job 的值等於畫面中 MB 的總數，代表這個畫面所有的 MBs 都已被處理。此時，可能有的 MSP 會較早閒置。由於一個 MSP 一次只處理一個 MB，這個閒置的時間不會超過處理一個 MB 移動估測的時間。

在處理的過程中，為了避免多個 MSPs 同時讀取 next_job，得到相同的值，我們使用號誌(semaphore)來確保一次只能有一個 MSP 可以存取 next_job。這可能造成一些 MSPs 等待號誌，不過機率相當低，等待的時間也非常短暫。

除了前述的兩個情況，還有一個情況 MSP 可能閒置。在一個畫面的處理開始之時，其他 MSPs 必須等待 MSP_0 建立 job list (僅是將 next_job 變數指定成 0)，這個時間也是非常短暫。因此，各個 MSPs 在整個過程中都大致保持忙碌，並且有很好的負載平衡度(load balance)。

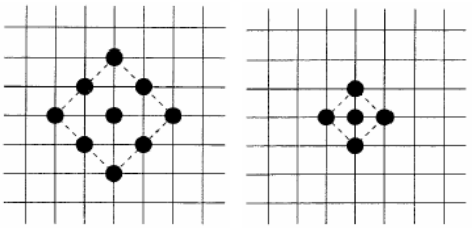
4.2. MSP 的工作

本節將探討 MSP 如何執行快速搜尋。如 2.2 節所述，許多快速搜尋的原理，是假設距離 best match 越近的 MB，失真越小。此類演算法擁有自己的搜尋樣式(search pattern)，它們透過計算搜尋樣式上各個搜尋點(search point)的 SAD 值來判斷 best match 可能的位置，逐步逼近 best match。

我們以著名的 diamond search (DS)為代表，來解釋這類演算法的運作方式。DS 有 2 個搜尋樣式，即 large diamond search pattern (LDSP) 和 small diamond search pattern (SDSP)，顯示於圖 3。

DS 的演算法如下：

(1) 將 LDSP 的中心點置於搜尋範圍的中心點，計算 LDSP 上各點的 SAD 值。若最小 SAD 值點出現在 LDSP 的中心，跳至(3)；否則，跳至(2)。
(2) 將 LDSP 的中心點置於上一步所找到的最小 SAD 值點上，計算 LDSP 上各點的 SAD 值。若最小 SAD 值點出現在 LDSP 的中心，跳至(3)；否則，重複此步。
(3) 以 SDSP 取代 LDSP，將 SDSP 的中心點置於上一步所找到的最小 SAD 值點上，計算 SDSP 上各點的 SAD 值，所得的最小 SAD 值點即為所求。



(a) (b)

圖 3. Diamond search 的兩個搜尋樣式：(a) large diamond search pattern (LDSP)，(b) small diamond search pattern (SDSP)。

計算搜尋樣式上各點的 SAD 值是彼此獨立的工作；因此，我們可以採取平行處理。也就是說，不同搜尋點的 SAD 值，我們可以交由不同的處理器同時來計算。MSP 中有兩個 DSPEs，可以同時針對兩個搜尋點計算 SAD 值。DSPE 是高速的計算引擎，而且它有一個特別針對視訊壓縮的指令，可以一次算出 4 對畫素的 SAD，所以 DSPEs 非常適合這項工作。PE 則負責執行快速搜尋的主程式，它會反覆利用搜尋樣式選出需要做 SAD 值計算的點，並命令 DSPEs 來做 SAD 值計算。DSPE 結束計算後，會把結果放在 Quad 共享資料記憶體的某個固定位址，使得 PE 能夠讀取。

我們把快速搜尋主程式和搜尋點的 SAD 值計算分開；這樣一來，若要使用不同的 ME 演算法，只需重新撰寫 PE 的 ME 程式。SAD 值的計算較為規律而繁瑣，我們以 DSPEs 來處理。

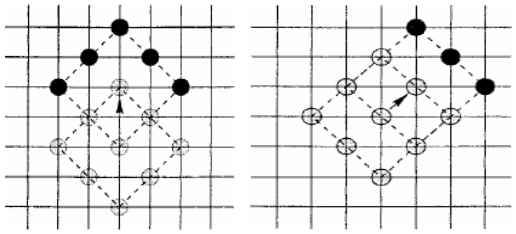
4.3. Quad 共享資料記憶體的使用

Quad 中有 64 KB 的共享資料記憶體，是 3SOC 晶片內的快取記憶體(on-chip cache)。當使用不超過 1 個 Quad，也就是 4 個 MSPs 時，4.1 節所提之 next_job 變數應該放在 Quad 共享資料記憶體。當使用超過 1 個 Quad，next_job 應該放在放在外部記憶體。另外一個可能的做法，是將 next_job 放在某個 Quad 的共享記憶體，讓其他 Quad 的 MSPs 透過全域匯流排間接來存取；但是，這樣需要額外撰寫繁雜的程式。

DSPE 在運作時，其運算元(operands)必須位在 Quad 共享資料記憶體。所以，我們必須預留空間來存放這些運算元。如 4.2 節所述，一個 MSP 包含了兩個 DSPEs，可以同時對兩個搜尋點做 SAD 值計算。因此，我們需要預留 $16 \times 16 = 256$ bytes 來存放目前 MB，與 $256 \times 2 = 512$ bytes 來存放 2 個候選 MBs。若 Quad 中四個 MSPs 同時運作，需要 $(256 + 512) \times 4 = 3072$ bytes = 3 KB。此外，計算過程中還需要一些暫時變數；由於這些記憶體需求量很小，在此不討論。

在一些快速搜尋的運作過程中，若能充

份利用 Quad 共享資料記憶體，可免去一些重複的計算。以 DS 為例，如圖 4 所示，若 LDSP 往上移動，其中 4 個點在上一步已被算出 SAD 值，我們只需再計算 5 個點，如圖中黑點；若 LDSP 往右上移動，其中 6 點在上一步已被計算，只需再計算 3 個點。因此，若能將計算過的搜尋點及其 SAD 值記錄下來，就能避免重複的計算。這個部分最簡單、快速的作法可以用一個二維陣列(2-D array)來記錄搜尋範圍中各個搜尋點的 SAD 值。我們也可以用雜湊(hashing)來完成這項工作，以各個搜尋點的座標為雜湊表(hash table)的位址。



(a) (b)

圖 4. (a) LDSP 往上移動一步，只需再計算 5 個點。(b) LDSP 往斜角方向移動一步，只需再計算 3 個點。

此外，對一個 MB 做移動估測需要好幾個候選 MBs 的資料，因而有好幾次讀取參考畫面的動作。事實上，不同的候選 MBs 常有很多重疊的畫素(參見圖 5)。因此，我們可以一次將整個搜尋範圍的資料讀入 Quad 共享記憶體，來避免重複讀取相同的畫素。這樣不只可以省去許多存取外部記憶體的時間，也減少匯流排的流量(bus traffic)。若搜尋範圍為[-7, 7]，一次讀入一個 MB 的整個搜尋範圍需要$(7 \times 2 + 16)^2 = 900$ bytes，4 個 MSPs 同時處理 4 個 MBs 共需 $900 \times 4 = 3600$ bytes ≈ 3.5 KB 的 Quad 共享記憶體空間。

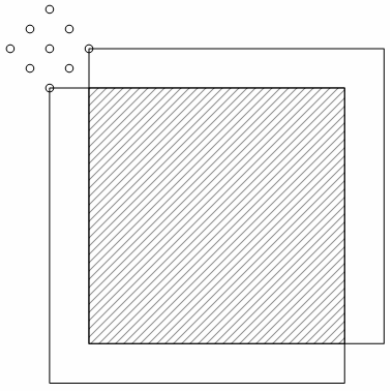
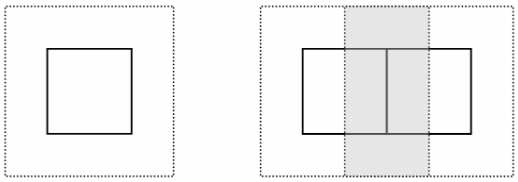


圖 5. LDSP 的右、下兩個候選 MBs 重疊的畫素用斜線表示。

另外，由於目前畫面中相鄰 MBs 的搜尋範圍也有相同的畫素(見圖 6)，我們也可以一次讀入多個 MBs 的搜尋範圍。例如，可以一次讀入好幾列 MBs 的搜尋範圍；這個動作是一次傳輸較長的、連續的資料，可以由 MTE 來執行，以減少 PE 的工作。



(a) (b)

圖 6. (a)一個目前畫面的 MB (實線)和它對應的搜尋範圍(虛線)；(b)兩個相鄰 MBs 的搜尋範圍有相同的畫素，如陰影部分。

4.4. 一些快速移動估測演算法的適用性

大部分的移動估測演算法，都是反覆地計算並比較一群候選 MBs 的 SAD 值，以找出 best match。我們所提之設計運作的基本原理，是利用各個目前 MBs 彼此資料獨立(data independent)的假設來平行處理，同時將 SAD 值的計算獨立出來讓較快的 DSPEs 執行。因此，完全搜尋以及大部分的快速搜尋都可以依照我們的設計以 3SOC 來實作。

然而，2.2 節中所提到 motion vector prediction 的概念需要修正。雖然常用的 predictors 包括了目前 MB 的上面、右上、左邊的 MBs 與前一個畫面相同位置的 MB，在我們的設計中，當目前 MB 正被某個 MSP 處理，它左邊的 MB 可能正由另一個 MSP 處理。因此，左邊的 MB 不適合被當作 predictor，而改用左上的 MB 來取代。

五、實驗結果

我們撰寫了程式，以不同的 MSP 個數來實驗我們的設計是否透過平行處理能得到好的加速。我們的程式在 Inspector 下執行，它能夠計算程式所花費的 3SOC 時脈週期數。

我們針對移動估測的部分探討所能達到的加速，而忽略視訊壓縮器的其他部分。在實際的應用時，視訊壓縮器中有輸入裝置來負責原始視訊資料的讀入，移動估測所得到的結果也必須經過進一步的壓縮才會輸出。由於沒有視訊壓縮器的其他部分，我們以 MSP_0 安排資料，來模擬真實移動估測器的運作，使得每一個畫面的處理開始之前，該畫面(即目前畫面)與其參考畫面都位在外部記憶體等待處理。

因為我們沒有視訊壓縮器的其他部分，我們直接以未經壓縮的畫面來當參考畫面。雖然一般移動估測所採用的參考畫面是壓縮後再解壓縮的畫面，由於壓縮後再解壓縮的畫面

與未經壓縮的畫面中同位置的畫素值差異不大，因此不容易影響搜尋的過程(例如搜尋樣式的移動)。所以，也不容易影響到計算時間及加速。所以，我們的實驗雖然是採用未經壓縮的畫面來當作參考畫面，仍能得到準確的加速。

我們的實驗依序對原始視訊資料的各個畫面進行移動估測，每個 MSP 執行的演算法是 diamond search [31]，搜尋範圍是[-7, 7]，每個待處理的畫面都以它的前一個畫面作為參考畫面。開始時，MSP_0 會先連續讀入兩個畫面到外部記憶體，亦即第一個待處理畫面與它的參考畫面。之後，在一個畫面的處理過程中，除了 MSP_0 之外，其他的 MSPs 負責對畫面中的 MBs 進行移動估測，並將結果寫入外部記憶體。同時，MSP_0 趁機將前一個畫面的處理結果自外部記憶體輸出到硬碟，這動作我們稱為 *output*；接著再將下一個畫面的資料自硬碟讀入記憶體，稱為 *input*。如果上述兩項工作提早完成，MSP_0 也會加入移動估測的處理。因此，在實驗裡，除了第一個與最後一個待處理的畫面外，一個畫面的處理過程中，各個 MSPs 的工作及時間關係如圖 7。在處理第一個待處理的畫面時，MSP_0 不做 *output*，因為還沒有得到任何結果需要輸出；在處理最後一個待處理的畫面時，MSP_0 不做 *input*，因為已沒有資料可以讀入。

本實驗在每個 MB 的移動估測開始之前，PE 都將這個 MB 在參考畫面的整個搜尋範圍的資料讀入 Quad 共享記憶體。如 4.3 節所述，這樣可以避免重複讀取不同候選 MBs 中相同的畫素。搜尋範圍是一個邊長 16 + 7 × 2 = 30 個畫素的正方形區塊，其中每一排(30 個)畫素是長 30 bytes 的連續資料。而 MTE 的使用時機，是在需要傳輸一段長而連續的資料的時候。若由 MTE 來做這項工作，需要啟動 MTE 30 次，一次傳輸 30 bytes。由於 30 bytes 並不長，利用 MTE 來傳輸並不划算。在實驗的其他部分，也沒有長於 30 bytes 的資料需要傳輸。所以本實驗並未使用 MTE。

我們採用了 QCIF (176 × 144)格式的原始視訊資料 Coastguard、Foreman、Salesman 的前 20 個畫面與 Salesman 的前 50 個畫面。表 1 展示了以不同的 MSP 個數處理原始視訊資料所得的加速，其中 Salesman_1 與 Salesman_2 分別代表 Salesman 的前 20 個與前 50 個畫面。以 n 個 MSPs 處理某個原始視訊資料所得的加速之計算方式如下：

$$\frac{\text{1個MSP所花費的時脈週期數}}{n\text{個MSPs所花費的時脈週期數}}$$

表 1. 以不同的 MSP 個數處理原始視訊資料所得的加速

No. of MSPs	Coast-guard	Fore-man	Sales-man$_1$	Sales-man$_2$
2	1.97	1.97	1.97	1.98
3	2.91	2.91	2.91	2.95
4	3.81	3.82	3.81	3.88
6	5.41	5.47	5.42	5.60
8	7.01	7.09	7.01	7.32
12	7.89	8.61	7.91	8.16

由表 1 來看，當使用的 MSP 個數不超過 8，即不多於兩個 Quads 時，加速表現很好；而在使用 12 個 MSPs 時，加速卻沒有顯著的成長。為了分析這個現象，我們記錄了每個 MB 是由哪個 MSP 所處理，發現當使用超過 9 個 MSPs，除了在第一個與最後一個被處理的畫面，幾乎沒有 MB 是由 MSP_0 所處理。

這顯示了當使用超過 9 個 MSPs，I/O 所需的時間就接近甚至超過移動估測的時間，使得別的 MSPs 必須等待 MSP_0 的 I/O 工作完成，才能開始處理下一個畫面。然而，在第一個畫面，MSP_0 不需做 *output*；在最後一個畫面，MSP_0 不需做 *input*。所以 MSP_0 在這兩處還能加入移動估測的處理，也因此當使用超過 9 個 MSPs 時，加速仍可稍微提升。

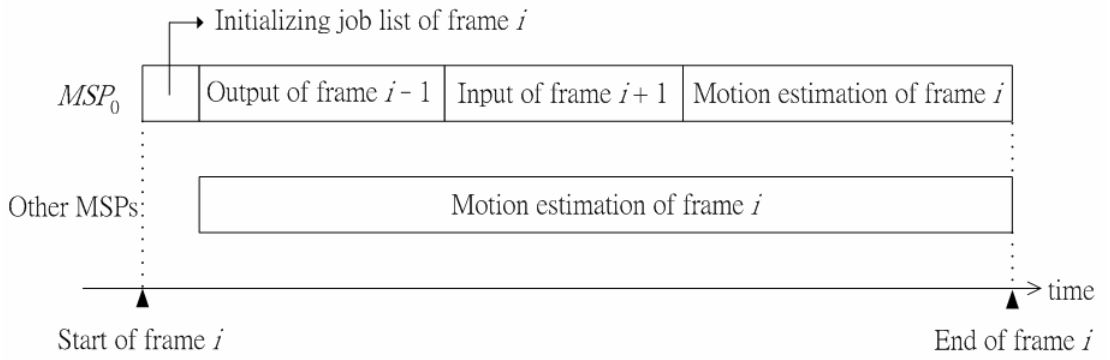


圖 7. 除了第一個與最後一個待處理的畫面外，一個畫面的處理過程中，各個 MSPs 的工作與時間關係圖。其中 MSP_0 裡各個工作的時間比例並非絕對。

雖然 MSP_0 的 I/O 工作形成了一個瓶頸，限制了加速的成長。可是，在實際利用 3SOC 製作視訊壓縮器時，MSP_0 並不需要做這些 I/O 工作。在移動估測開始之前，原始視訊資料應已由特定的輸入裝置(如攝影機)讀入外部記憶體；移動估測的結果也將留在外部記憶體，經過進一步的壓縮(如 intraframe coding)之後才會輸出。MSP_0 在設定 next_job 之後，就可以和其他的 MSPs 一樣做 ME。所以，在實際利用 3SOC 製作視訊壓縮器時，MSP_0 的 I/O 工作所造成的瓶頸並不存在。因此，可以使用更多的 MSPs，更快完成 ME。

六、結論

由於移動估測非常耗時，軟體的實作需要強大的計算能力。3SOC 是一個多處理器的高效能晶片，很適合用在多媒體領域如視訊壓縮。我們透過 3SOC 晶片以軟體來實作移動估測的平行處理。絕大部分移動估測演算法基本上都可以透過我們的設計在 3SOC 上運作。

我們利用了 MBs 之間的資料獨立性(data independency)，以一個額外負荷很小、負載平衡佳的資料分配方法，將資料分配給 MSPs 處理。我們找出適當的工作給 3SOC 中不同類的處理器來執行。

在我們的實驗中，使用的 MSP 個數不超過 8 的情況下，我們得到的加速接近 MSP 個數。當使用更多的 MSPs，加速則逐漸受到 I/O 的限制。在實際以 3SOC 製作視訊壓縮器時，可以使用更多的 MSPs，更快完成 ME。

七、參考文獻

[1] *Information Technology — Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s: Video*, ISO/IEC 11172-2 (MPEG-1 Video), 1993.

[2] *Information Technology — Generic Coding of Moving Pictures and Associated Audio Information: Video*, ISO/IEC 13818-2-ITU-T Rec. H.262 (MPEG-2 Video), 1995.

[3] *Information Technology—Generic Coding of Audio-Visual Objects: Part 2: Visual*, ISO/IEC 14496-2 (MPEG-4 Video), 1999.

[4] *Video Codec for Audiovisual Services at p × 64 kbit/s,* ITU-T Rec. H.261, Mar. 1993.

[5] *Video Coding for Low Bitrate Communication,* ITU-T Rec. H.263, Mar. 1996.

[6] I. Ahmad, S.M. Akramullah, M.L. Liou, and M. Kafil, "A scalable off-line MPEG-2 video encoding scheme using a multiprocessor system," *Parallel Computing,* vol. 27, no. 6, pp. 823-846, 2001.

[7] S.M. Akramullah, I. Ahmad, and M.L. Liou, "A data-parallel approach for real-time MPEG-2 video encoding," *Journal of Parallel and Distributed Computing,* vol. 30, no. 2, pp. 129-146, Nov. 1995.

[8] S.M. Akramullah, I. Ahmad, and M.L. Liou, "Parallel MPEG-2 encoder on ATM and Ethernet-connected workstations," in *Proc. 1999 Int. Conf. of the Austrian Center for Parallel Computation,* Salzburg, Austria, pp.572-573, 1999.

[9] D. Alfonso, F. Rovati, D. Pau, and L. Celetto, "An innovative, programmable architecture for ultra-low power motion estimation in reduced memory MPEG-4 encoder," *IEEE Trans. Consumer Electronics,* vol. 48, pp. 702-708, Aug. 2002.

[10] M. Alkanhal, D. Turaga, and T. Chen, "Correlation based search algorithm for motion estimation," in *Proc. Picture Coding Symp.,* Portland, OR, 1999.

[11] M. Bolton, F. Homewood, A. Robinson, D. Bagni, and A. Borneo, "A VLIW processor-based audio/video codec for consumer applications," *Int. Conf. on Consumer Electronics, Digest of Technical Papers,* pp. 268-269, 2002.

[12] R.S. Cok, *Parallel Programs for the Transputer.* Englewood Cliffs, NJ: Prentice Hall, 1991.

[13] Cradle Technologies, *Inspector User's Guide ver. 3.2 Beta.* Mountain View, CA: Cradle Technologies, 2002.

[14] Cradle Technologies, *The Software Scalable System On a Chip (3SOC) Architecture.* Mountain View, CA: Cradle Technologies, 2003.

[15] L. Fanucci, S. Saponara, and L. Bertini, "A parametric VLSI architecture for video motion estimation," *Integration: The VLSI Journal,* vol. 31, no. 1, pp. 79-100, Nov. 2001.

[16] Y. He, I. Ahmad, and M.L. Liou, "Modeling and scheduling for MPEG-4 based video encoder using a cluster of workstations," in *Proc. 1999 Int. Conf. of the Austrian Center for Parallel Computation,* Salzburg, Austria, pp. 306-316, 1999.

[17] P.I. Husor and K.-K. Ma, "Motion vector field adaptive fast motion estimation," in *Proc. 2nd Int. Conf. on Information, Communications and Signal Processing,* Singapore, 1999.

[18] S. Kittitornkun and Y.-H. Hu, "Frame-level

pipelined motion estimation array processor," *IEEE Trans. Circuits Syst. Video Technol.,* vol. 11, no. 2, pp. 248-251, Feb. 2001.

[19] T. Koga, K. Linuma, A. Hirano, Y. Iijima, and T. Ishigr, "Motion compensated interframe image coding for video conferencing," in *Proc. NTC81,* New Orleans, LA, pp. G5.3.1-5.3.5, 1981.

[20] P. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation.* Boston, MA: Kluwer Academic, 1999.

[21] R. Li, B. Zeng, and M.L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuit Syst. Video Technol.,* vol. 4, pp. 438-442, Aug. 1994.

[22] J.-Y. Nam, J.-S. Seo, J.-S. Kwak, M.-H. Lee, and Y.H. Ha, "New fast-search algorithm for block matching motion estimation using temporal and spatial correlation of motion vector," *IEEE Trans. Consumer Electronics,* vol. 46, no. 4, Nov. 2000.

[23] K.K. Parhi and T. Nishitani, eds., *Digital Signal Processing for Multimedia Systems.* New York, NY: Marcel Dekker, 1999.

[24] L.M. Po and W.C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuit Syst. Video Technol.,* vol. 6, pp. 313-317, June 1996.

[25] K. Shen and E.J. Delp, "A spatial-temporal parallel approach for real-time MPEG video compression," in *Proc. 1996 Int. Conf. on Parallel Processing,* Bloomingdale, IL, pp. II100-II107, 1996.

[26] Y.Q. Shi and H. Sun, *Image and Video Compression for Multimedia Engineering.* New York, NY: CRC press, 1999.

[27] A.M. Tourapis, O.C. Au, and M.L. Liou, "Fast block-matching motion estimation using predictive motion vector field adaptive search technique (PMVFAST)," *ISO/IEC JTC1/SC29/WG11 MPEG99/m5866,* Noordwijkerhout, the Netherlands, Mar. 2000.

[28] J.-C. Tuan, T.-S. Chang, and C.-W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *IEEE Trans. Circuits Syst. Video Technol.,* vol. 12, no. 1, pp. 61-72, Jan. 2002.

[29] J.-B. Xu, L.-M. Po, and C.-K. Cheung, "A new prediction model search algorithm for fast block motion estimation," in *Proc. IEEE Int. Conf. Image Processing,* Santa Barbara, CA, 1997.

[30] C. Zhu, X. Lin, and L.-P. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Trans. Circuit Syst. Video Technol.,* vol. 12, pp. 349-355, May 2002.

[31] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Trans. Image Processing,* vol. 9, pp. 287-290, Feb. 2000.