動作估測演算法在 Cell 處理器架構上之最佳化設計

The Optimal Design for Motion Estimation Algorithm on Cell Processor Architecture

Jih-Ching Chiu (邱日清)　Ta-Li Yeh (葉大立)　Cheng-Han Liu (劉政翰)

Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung,804, Taiwan

Email: chiujihc@ee.nsysu.edu.tw; d973010017@student.nsysu.edu.tw; m963010026@student.nsysu.edu.tw

***摘要*—視訊資料編碼是視訊傳輸技術中相當重要的一環，而動作估測演算法是資料編碼過程中一個重要的動作。若能以平行化的方式進行動作估測演算法的運算將可縮短資料編碼的時間。**

本論文將會針對 Three-Step Search 以及 Diamond Search 兩種動作估測演算法進行最佳化。實現的方式是根據 Cell 處理器的特性，設計出 multiple buffering 的 DMA (Direct Memory Access) 資料存取機制、以向量計算取代原本的純量運算與以減少 branch 指令數的方式來避免 branch miss 造成的 penalty，加上設計適切的資料分割排程方式來對動作估測演算法加速，期望在多核心的環境下設計出最佳的演算法架構。依據以上的加速機制，我們設計出的動作估測演算法在使用 CIF 大小影像並且使用五張參考影像的情況下實驗結果可以達到每張影像 13.26ms 的處理速度。

***ABSTRACT*—Motion estimation algorithm is an important part of video transmission technology. If we can parallelize the calculation, the efficiency of will be raised.**

In this paper, we will optimize the Three-Step Search and the Diamond Search. The best way to optimize the algorithm is based on characteristics of Cell processor, the design of multiple buffering of DMA (Direct Memory Access) mechanisms and using vector SIMD computer mechanism to replace the original scalar computing mechanism and to reduce the number of branch instructions to avoid causing the penalty due to branch miss. Based on the acceleration of the above mechanism, we design algorithms for motion estimation in the use of CIF image size and the use of reference images of five cases the experimental results can be achieved for each image processing speed of 13.26ms.

***關鍵詞*—動作估測演算法，Cell 處理器，平行化運算，最佳化**

***Key Word*—Motion Estimation，Cell processor，parallelize calculation，optimize**

一、簡介

由於當前網路的發展以及視訊傳輸技術的演進，網路多媒體的應用已經是不可或缺的一個部份。而多媒體的應用幾乎都需要高資料流量的計算，在高資料流的情況下，如何提高多媒體應用的效能是一個相當重要的課題。

以多媒體應用中的視訊編解碼來說，動作估測(Motion Estimation)演算法是相當重要一個動作，它可以將原本的連續的影像重複的部份抽出並以一個移動向量來代替。這種動作的運算量相當的龐大，所以如何找出一個對動作估測演算法最佳化的方法就顯的十分重要。故本論文希望在多核心系統如 Cell 上以多核心平行運算的方式設計最佳化的動作估測演算法。

二、相關研究

在對 Motion Estimation 做最佳化前，首先會介紹用來實現演算法的處理器架構 Cell B.E. [1][2][3][4][5][6] ，選擇 Cell B.E.的原因是 Cell B.E. 具有高傳輸量的匯流排，可以快速的將資料傳給各協同處理器 (SPE) 計算，此外不管是虛擬平台 (IBM Cell B.E. SDK) ，或是實體平台 (SONY PlayStaion®3) 都很容易取得，所以決定採用 Cell B.E.平台來實現動作估測演算法的最佳化；然後決定使用的演算法，我們所選擇的演算法是 Three-Step Search [7][8] ，以及 Diamond Search [9][10][11] ，會選用這兩個以算法是因為 Three-Step Search 演算法中計算量是固定的，而 Diamond Search 的計算量則會視運算的影像不同而有差別，如果我們的最佳化設計在兩者上都可以得到明顯的加速，就可以證明我們的設計是有效的，而不是針對特定演算法進行最佳化；最後再介紹目前相關研究在 Cell B.E. 上所使用加速方法 [12][13][14][15][16][17]，例如：以多顆 SPE 平行計算、以 SIMD 指令取代純量指令的計算、減少 branch 指令數及使用 Multiple Buffering 等。

(一) Cell B.E. 處理器架構

Cell B.E. (Cell Broadband Engine) 處理器是由 Sony, Toshiba 和 IBM 三家公司於 2001 年開始共同研發的新一代處理器。

Cell 的硬體架構上的特點，是它具有一個或多個主要處理器 PPE (PowerPC Processing Element) 以及數顆具完整功能的協同處理器 SPE (Synergistic Processing Element) ，可以用來進行高平行度的運算，並且用 1 條高資料頻寬的 circular data bus 來連接 PPE、SPE 與 I/O 裝置，稱為 EIB (Element Interconnect Bus) ，為了使每個處理器都可以存取 Cell 的主記憶體與外部儲存裝置，所以在每個 SPE 上都有 DMA engine ； CBEA (Cell Broadband Engine Architecture) 架構如圖 1.；主要是由一顆 PPU 與八顆 SPU 所組成，PPU 與 SPU 的時脈皆為 3.2GHz，除此之外，此架構還有一個 MIC (Memory Interface Controller) 支援 Dual XDRTM 連接到記憶體，以及一個 BIC (Bus Interface Controller) ，BIC 具有兩組對外通道，可支援兩組 FlexIO™的匯流排 IO，然後將這些元件掛在 EIB 上，在這個架構下，EIB 可達到的最高傳輸速度為 96bytes/cycle (共有 12 組連接數 (PPU 一組，八顆 SPU 各一組，MFC 一組，BIC 上有兩組) ，每組連接都可以一次傳輸 16 個 byte，每次傳輸需 2cycle) ，這樣龐大的資料傳輸量就可以拿來進行多資料流的平性化運算，以提升運算效能。

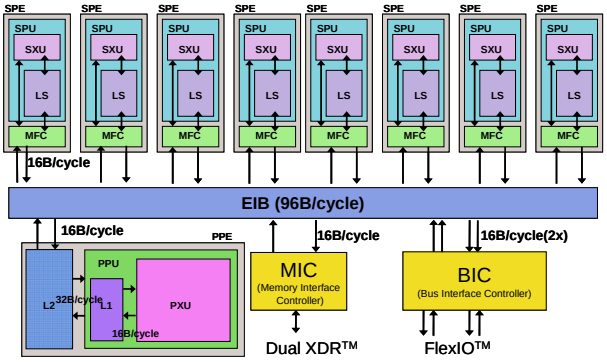

圖 1. 實際使用的 Cell B.E.架構

(二) Motion Estimation 演算法

Motion Estimation 是計算一張 2D 的影像到另一張影像之間的移動向量，這個移動向量可能影響到整張圖像或是某些特定的部份，如宏區塊 (macro block) 或是任意形狀的一個區塊，甚至是一個像素(pixel)；與 Motion Estimation 相對的動作就是 Motion Compensation，就是根據 Motion Estimation 產生出的移動向量以及原始的影像，合併出新的影像出來。

計算移動向量的方法分為兩大類，有基於像素的方法”直接法”與基於影像特徵的方法”間接法”；我們所使用的演算法是直接法，以下就只討論直接法的部份。

直接法又以計算的方式分成幾種，我們用的是 Block-matching algorithm，這種演算法的計

算方式是將目標影像 (current frame) 切成許多個區塊(block)，然後在參考圖 (reference frame) 中找出與目標區塊差異最小的區塊，判定差異大小的方式是計算 SAD (Sum of Absolute Differences) ，就是將搜索範圍 (Search Window) 內的各區塊內各像素相減之後取絕對值，然後再將所有像素算出來的絕對值加總，得到的值就是 SAD 值，而在參考圖上目標區塊的位置與 SAD 最小值的區塊之間的位移量就是最後的移動向量。

1. Three-Step Search Algorithm

Three-Step Search 是在 Motion Estimation 演算法中比較常見的，屬於 Uniform Search Algorithm，他會在整個搜索範圍內平均的尋找 SAD 最小值的位置，因為它的動作不一定是三步，所以有時也被稱為 N-Step Search；他的動作是以位移量 0 的區塊作為中心，動作如圖 2. 第一步會做中心及從中心向 X 軸方向與 Y 軸方祥延伸出距離 2N 的九個點，然後在這九個點為頂點的區塊中找出 SAD 的最小值，第二步是從第一步的最小值那個點再向 X 軸與 Y 軸的方向延伸出距離 2(N-1)的八個點，比較這八個點為頂點的區塊及前一次計算出的最小值的區塊的 SAD 值，然後就一直重複這個步驟，直到向外延伸的距離為 1 個像素才停止，每一步的動作相當固定，可以減少判斷式的使用以減少 branch 發生的機率，所以我們採用此演算法來進行最佳化運算。

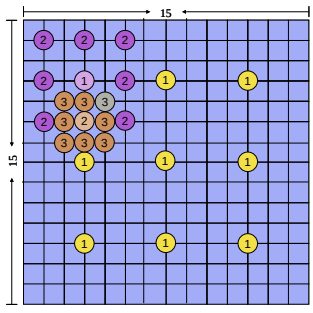

圖 2. Three-Step Search 示意圖

2. Diamond Search Algorithm

Diamond Search 也是常見的 Motion Estimation 演算法之一，與 Three-Step Search 不同，他是屬於 Center-Based Search Algorithm，這類演算法的動作特性是從位移 0 的中心點開始向外擴張出去；他的動作如圖 3.，是由位移 0 的點為中心向外擴張出去的，第一個步驟會做中心點以及從中心點向外，X 軸與 Y 軸的位移量和為 2 像素的點共九個點，再以這九個點為頂點的區塊中找出 SAD 最小值後，會依照得到 SAD 最小值點的位置不同而分成三種情況，分別是最小值在第一步的九個點所形成的菱形的邊上、頂點及中心，在邊上的情況會沿著斜邊向外延伸三個點；在頂點的情況則是會向外延伸五個點，然後會比較這些延伸出的點為頂點的區塊的 SAD 值與前一次計算的最小 SAD，找出這次的最小值；以上這兩種情況做完之後都會與步驟一相同，根據其 SAD 的最小值位置不同繼續向外擴張出去，直到第三種情況，也就是 SAD 的最小值的位置是落在中心或者是在向外擴張時最小值是落在前一次計算的最小值的位置，在這種情況下會從 SAD 最小的點向上下左右移動 1 像素的四個點，然後對這四個點作最後的一組 SAD 計算，這四個點為頂點的區塊的 SAD 值與中心的 SAD 值比較後最小值的位置就是最後的位移量。

第二個演算法會選擇 Diamond Search 的原因是因為他與 Three-Step Search 分別是屬於不同類型的 Motion Estimation 演算法，是從中心開始向外重複的進行擴散，直到找到適合的結果才會停止，會進行重複的擴散這點就是跟 Three-Step Search 最大的區別。

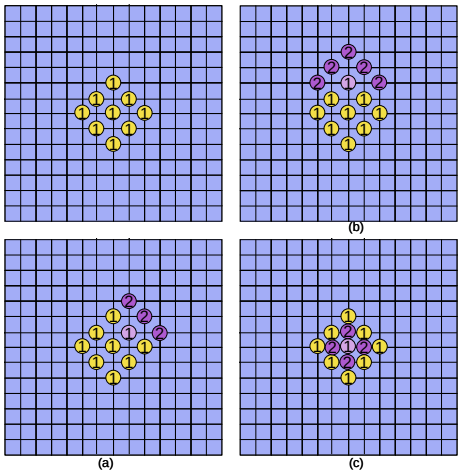

圖 3. Diamond Search 示意圖

(三) Cell B.E.平台上運算加速的方法

在 Cell B.E 平台上要進行運算加速的方法大概可以分為以下幾類，(1) 以多顆 SPE 同步處理，進行 thread-level 的平行化處理，(2) 以 SIMD 的向量運算進行 data-level 的平行化處理，(3) 以減少 branch 指令的方式降低 branch miss 時的 penalty 對程式效能的影響，(4) 以 multiple buffering 的方式減少大量資料傳輸時，運算單元等待資料就緒的時間；以下將分別介紹這些加速方法的原理。

1. 以多顆 SPE 平行運算來加速

這個方式是在 Cell B.E.平台上最常見的加速方式，將原本的演算法拆成幾個較小的動作，在將這些動作分配到 SPE 上做運算，PPE 再把 SPE 算出來的結果做整合或在進行下一步的運算。

在[16]這篇文章中，他同樣是對 Motion Estimation 做加速，他的 Motion Estimation 是針對 H.264 的影像壓縮演算法來設計的，所以會從目標影像往前參考 5 張參考影像，他的作法是將每張參考影像的移動向量的運算分配給一顆 SPE 來做，這樣的優點就是每一顆 SPE 裡面的運算較單純，每一顆 SPE 的運算量都一樣，而 PPU 就只需要接收各 SPE 做完的移動向量及最小 SAD 值，再選出其中 SAD 最小的一組移動向量就是最後結果。

2. 以 SIMD 的方式進行運算

以 SIMD 方式運算也是相當常見的加速方式，他是使用 SPE 內建的 SIMD 指令集來進行運算，以陣列的加法為例，在使用純量的加法運算時，a 與 b 兩個陣列相加就是陣列 a 的一個元素跟陣列 b 的一個元素相加，a 與 b 陣列內有幾個元素就必須經過幾次的加法運算才能算出最後 c 陣列的結果，而如果使用 SIMD 的方式運算，就可以直接使用向量的加法指令直接相加，即可得到結果。

3. Multiple Buffering

除了以上這些加速方式，改善 DMA 的動作也是可以用來加速的方法之一，在處理大量資料時無法讓 DMA 一次就完成資料抓取，而且在一次抓取大量資料的情況下等待資料抓取的時間會延遲後面運算的動作，所以就改善 DMA 的動作如圖 4.，將要抓取的資料分成多份，每抓完一次資料就發出下一次的 DMA command，把資料抓到另一個 buffer 裡，並且計算剛剛抓進來的資料，將等待 DMA 的時間與資料運算的時間重疊，然後在計算結束之後再發出下一次的 DMA command 並開始計算剛剛抓進來的資料，重複這個動作直到資料抓取結束，這樣可以減少整體運算所花的時間，效能增進最大的情況是抓取資料的時間剛剛好與計算的時間相等，設定的 buffer 大小就會與效能有明顯的關係，buffer 太大的話會變成計算結束之後還要等 DMA，buffer 太小的話則是會增加初始化 DMA 動作的次數而造成延遲，而且在 DMA 要抓的資料越多的時候加速會越明顯。

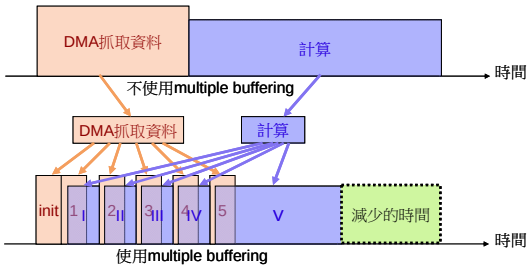

圖 4. multiple buffering 示意圖

4. 減少 branch 的指令數

在介紹 SPE 的時候已經有提到 branch miss 的 20 個 cycle 的 penalty 對 SPE 效能的傷害是很大的，所以在程式內能夠避免使用多餘的判斷式就盡可能避免，在”High-Performance Regular Expression Scanning on the Cell/B.E. Processor”這篇文章中是以 SIMD 的方式來解決判斷式的問題，他的判斷式是用來決定數筆資料的來源，他的解決的方式是將兩筆資料各存成向量變數，而判斷式的選擇訊號也是以 SIMD 的邏輯運算的方式來產生各元素的選擇訊號，根據選訊號使用 SIMD 的指令選出需要的資料，這樣就可以在不使用 branch 的方式完成這個選擇的動作。

三、Cell 平台上動作估測演算法設計

為了使動作估測演算法在 Cell 平台上進行加速，我們會對動作估測演算法的動作做一些改善，然後會將我們設計的加速方法實際應用，然後會以另一個演算法同樣改良並實際應用這些最佳化的方法，來驗證這種最佳化的方式是否對其他演算法也通用。

(一) Three-Step Search 在 Cell 平台上的實現

在將動作估測演算法實現之前，首先要設定相關的參數，我們設定每個區塊 (block) 的大小為 8x8 個像素，搜索範圍 (search window) 為 15 個像素，由於我們使用的影像是 352x288 像素的黑白影像，所以每個像素的大小為 1 個 byte，從全白到全黑是 255 到 0，參考 H.264 影像壓縮演算法中的 Motion Estimation 的作法，所以設定參考影像 (reference frame) 的數量為 5 張影像，目標影像 (current frame) 則會是連續的多張影像。

PPE 部分的程式，首先要抓取進行計算的影像，包含目標影像及參考影像，接下來設定 SPE 計算所需的參數，像是建立 SPE 的 thread，SPE 抓取影像的位址、回存移動向量的位址、區塊的大小以及搜索範圍的大小，然後啟動 SPE，讓 SPE 開始計算，接下來可以先把下一張要做 Motion Estimation 的影像抓進來，之後就是等待 SPE 計算結束的通知，然後會視有無下一張影像來決定要通知 SPE 抓取下一張影像或是結束運算，然後將移動向量寫回檔案。

然後設計 SPE 的動作，首先會接收 PPE 傳送的參數，然後根據這些參數抓取影像，在抓取影像結束之後就會依照 Three-Step Search 的動作開始尋找 SAD 最小值；完成之後就重複以上動作計尋找下一個區塊的 SAD 最小值，到整張影像結束，然後進行第二張影像的運算，直到所有影像結束後會傳回所有的移動向量，然後通知 PPE 計算結束，然後 PPE 會根據有無下一張要計算的影像來決定通知 SPE 要抓取下一張影像或是結束動作。

(二) 以 8 顆 SPE 平行運算進行加速

我們使用 8 顆 SPE 平行運算的方式與前面提到那篇的做法不同，他的方法是將每張參考影像放在不同的 SPE 裡面運算，而我們的做法則是將原本的影像拆成八分，如圖 5.(a)；與一顆 SPE 計算一張影像比較，拿 8 顆 SPE 把一張影像平分的作法在 DMA 傳輸的資料量就有差別了，在一顆算一張影像的情況下，每顆 SPE 必須取得 2 張完整的影像，而拆成八份的作法則只需要抓取目標影像的其中一份，以及從 5 張參考影像中抓取各一份及多出來的搜尋範圍的資料，這個部份是因為在分割後在邊緣的區塊的搜尋範圍並非不存在，而是與前一個分割區共用，所以會有重疊的部份，這樣可以減少許多的 DMA 傳輸量，而且 Motion Estimation 演算法在不同目標區塊之間沒有相依性的存在，所以這要切割得當，計算上是不會有任何的影響；除此之外一顆算一張影像的做法還有一個缺點，就是他真正的移動向量必須等所有 SPE 都計算結束，將各自的移動向量傳回給 PPE 之後才可以由 PPE 從這些值中找出真正最適合的移動向量，而我的做法則可以在 SPE 就直接算出最後的移動向量。

為了因應演算法的改變，PPE 與 SPE 的動作都需要跟著做改變，PPE 在設定 SPE 參數的時候要多設定每顆 SPE 要計算的影像寬度以及各 SPE 在抓取影像以及回存移動向量時需增加的偏移量；SPE 則需要在抓取影像及回存移動向量時將偏移量算進去才能做抓取及回存的動作。

此外這樣的分割是將影像切在較短邊，這樣不同分割區重疊的部分會較多，但如果把影像翻轉，改成如圖 5.(b)的切割法，可以減少重疊部分的總量，這樣就可以更加減少 DMA 需要抓取的資料量。

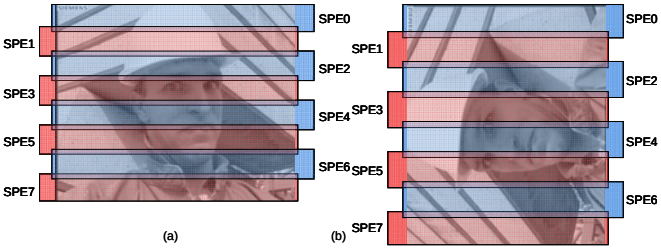

圖 5. 以兩種不同方式切割影像

(三) 以 SIMD 運算進行加速

使用 SIMD 的方式來加速主要是用在計算 SAD 的過程，原本的純量計算方式是將參考影像中區塊的每一個像素與目標影像中區塊對應的像素相減，然後取絕對值，再累加起來，這樣的運算過於繁複且效率低，所以改用 SIMD 進行向量的運算以進行加速，由於設定的區塊大小為 8x8 個像素，總共為 64 筆 1 位元的資料，剛好可以存進 4 個向量變數裡面，然後向量變數再依 圖 6.的方式，將每個參考影像的向量變數與目標影像的向量變數使用 spu_absd 指令將變數中所有的元素相減取絕對值，然後在將其中任兩組的結果使用 spu_sumb 指令做加總的動作，這個指令是將向量變數中 4 筆 1 個 byte 的元素加總變成 1 筆 2byte 的資料，然後兩個變數各自的 4 筆結果交錯放置，然後原本 4 組原本內容為 16 筆 1 個 byte 的資料的向量變數變成 2 組內容為 8 筆 2 個 byte 的資料的向量變數，然後在用 spu_add 指令將兩組向量變數相加，變成 1 組內容為 8 筆 2 個 byte 的資料的向量變數，最後再將這 8 筆資料兩兩相加，最後得到的結果就是 SAD 值。

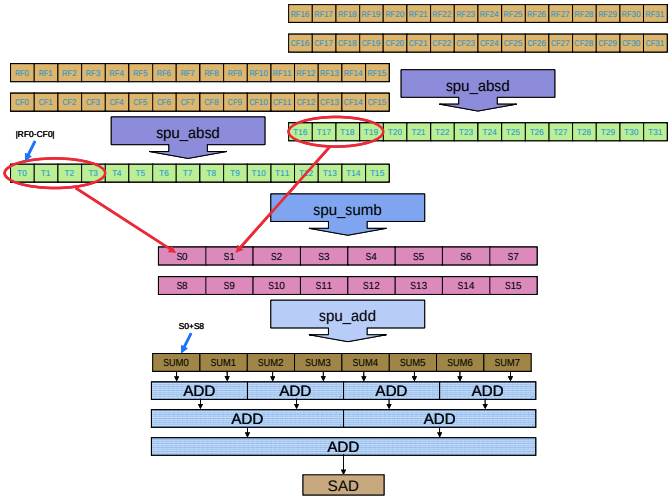

圖 6. 以向量 SIMD 方式計算 SAD

(四) 以 Multiple Buffering 運算進行加速

Multiple Buffering 主要是用來減少 DMA 存取資料的時候造成的延遲，作法是將原本 DMA 連續存取大量資料的動作改成單次存取較小量的資料，然後將存取資料的時間與計算的時間重疊來增進效能，作法是在 SPE 開始的時候先將各自負責的目標影像與參考第一列的區塊計算需要的資料先抓進來，然後在每次 SAD 的計算之前使用一個 buffer 抓取目標影像的其中一段資料，比較最小值之前使用另一個 buffer 抓取一段參考影像的資料，在計算第一張參考影像的時候參考影像與目標影像都要抓取，而在計算第二張之後參考影像時，因為需要使用的參考影像會在前一次的 Motion Estimation 計算時就會被抓取進來，所以不需要再抓取；另外回存移動向量的時候也是設計成每作完一列的區塊之後就會將結果回存一次，也可以減少最後回存資料的延遲。

設定的 buffer 大小會對整體運算的效能有不小的影響，而且必須在一列的區塊運算結束前必須把下一列的區塊的資料與其搜索範圍內的資料抓取完畢，由於正在計算的那一列區塊

的搜尋範圍會與下一列區塊重疊所以如圖 7.，所以在計算下一列之前只要一張影像再抓超過 8*288 個像素就可以確保下一次計算資料的正確性；一列總共有 36 個區塊，計算一個區塊可以對一張影像抓取 3 份 buffer 大小的資料 (在三個步驟各抓取一份) ，而 buffer 的大小只要 buffer 的大小大於 32 個 byte 就可以確保所有運算資料的正確性。

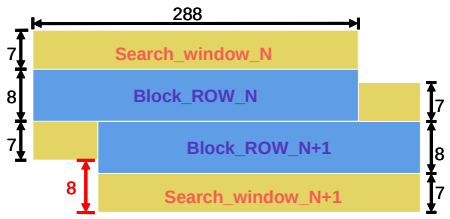

圖 7. 區塊重疊部分示意圖

(五) 減少 branch 指令數進行加速

減少 branch 指令數的方法其中一項就是將迴圈 unroll，前面使用 SIMD 的方式計算 SAD 值的動作也有 unroll 迴圈的動作，所以同時也會減少 branch 的指令數，除了這部分之外，還有 SAD 最小值的判斷也可以使用減少 branch 指令數的方式進行改良，以不使用 if 的判斷式來選出最小的 SAD 值。

改良方式如圖 8.，是在各點的 SAD 值計算出來之後先不進行比較，SAD 計算出來的結果是存成 4 個 byte 的 unsigned integer，然後用兩個向量變數來儲存其中的 SAD 值，然後使用比較指令產生一個比較的結果，這個結果也是一個向量變數內容為 4 組 4 個 byte 的比較結果，其中在結果為大於的欄位會是 4 個 byte 全部都是 1 的值，反之在結果為小於或等於的欄位會是全 0；然後在用這個比較結果對原本的兩個向量變數進行選擇，選出來的結果就是 4 筆較小的 SAD 值，然後再重複以下面的算式計算及可選出最小值。

$$SAD_{\min} = \begin{matrix} (SAD_1 \,\&\, ((SAD_1 \geq SAD_2) - 1)) \mid \\ (SAD_2 \,\&\, ((SAD_1 < SAD_2) - 1)) \end{matrix} \quad (1)$$

這個原理是在 SAD1 與 SAD2 這兩個值經過大於等於或是小於的運算之後會依照比較結果產生 0 或 1 的值，兩個比較式中只會有一個是 1，另一個則是 0；在減 1 之後則變成 0 或是 -1，-1 也就是所有的 bit 皆為 1，所以這個值就變成了一個遮罩，與 SAD 值作 and 運算之後，比較結果為 0 的那一項因為減 1 的關係變成-1，所以會把該 SAD 值保留，比較結果為 1 的那項因為減 1 之後為 0，所以在 and 之後值就為 0，然後用 or 的方式將兩者合併，也就可以將 SAD 較小的值保留下來，這樣就達成了不使用 if 的判斷式而單純使用算術運算及邏輯運算就得到 SAD 最小值的目的。

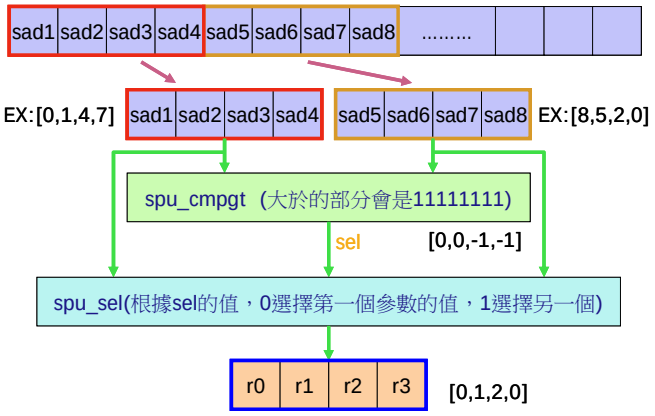

圖 8. 以向量 SIMD 方式選擇 SAD 較小值範例

但是這個方法會遇到一個問題，雖然已找出最小值，卻無法知道這個最小值是由哪一個點所產生的，這樣就沒辦法算出位移量，所以必須再加入一個 SAD 來源的判斷，作法就是將每次計算 SAD 值的點標上一個序號，然後建立一個陣列存放由小到大的序號，算出的 SAD 值的排列也是依照序號由小到大排列然後與前面的步驟相同，以 spu_cmpgt 與 spu_sel 這兩個指

令先做出部分的比較結果，並且以相同的選擇訊號對序號做出選擇，作到這裡可以得到序號以及其對應的 SAD 值，然後將上面的式子稍作改良如下：

$$\begin{aligned} GT &= (SAD_1 < SAD_2)\text{-}1 \\ LE &= (SAD_1 \geq SAD_2)\text{-}1 \\ SEQ_{\min} &= (SEQ_1 \,\&\, LE) \mid (SEQ_2 \,\&\, GT) \\ SAD_{\min} &= (SAD_1 \,\&\, LE) \mid (SAD_2 \,\&\, GT) \end{aligned} \quad (2)$$

使用這樣的公式將得到的結果兩兩計算之後即可得到最小的 SAD 值以及他的序號，這樣就可以用序號來反推他的位移量，這樣一來就可以徹底完成尋找 SAD 最小值及其移動向量的計算了。

(六) Diamond Search 在 Cell 平台上的實現

在 Cell 平台上實現第二種演算法是為了證明以上這些加速的方式並不是只針對 Three-Step Search 的加速，而是不管演算法是使用哪一種都可以以上述的加速方式進行加速，這樣的加速法才能算是最佳化。

在 PPE 的部份動作與做 Three-Step Search 時相同，一樣是先讀取參考影像與目標影像，然後設定 SPE 所需的參數，再啟動 SPE 開始運算，並且在 SPE 運算的時候讀取下一張影像，然後等待 SPE 結束傳回移動向量，最後根據是否需要再計算下一張影像來決定市要結束程式還是通知 SPE 抓取下一張影像。

SPE 部分的動作則是分為兩個部份，第一個部份是位移 0 的位置以及距離他 X 軸與 Y 軸距離和為 2 的 8 個點的 SAD 最小值，第二個部份則是根據第一步的 SAD 最小值位置而向不同方向擴散的動作。

首先將這九種不同的擴散動作記錄在一個陣列裡，然後根據第一步的結果的最小值對應的序號來查詢擴散的動作，擴散的動作又分為三類，第一類是 SAD 最小值位在菱形的頂點的情況，在這個狀態會以該頂點為中心向外擴散 5 個點，第二類狀態則是 SAD 最小值落在菱形的邊上，在這個情況會沿著邊向外擴散 3 個點，第三個狀態則是 SAD 最小值落在中心，會擴散向上下左右一個像素的位置；第二部份的動作會不斷的重複直到落入第三種狀態，或是在達到搜尋範圍的邊界時會強制進入第三種狀態才會停止。

為了實現這種動作我們將第三種狀態的序號設定成 0，第一種狀態的序號設定成奇數值，第二種狀態的序號設定成偶數值，這樣一來要知道目前的狀態下需要計算幾個點的 SAD 值只需要套用下面的公式即可算出，狀態為奇數的時候計算結果就是 5，在偶數且非 0 的時候則是 3，在狀態為 0 的時候結果是 4，符合計算需求，而且這樣的作法不需使用 if 判斷式，可以避免 branch 指令的產生以增加效能。其他的計算如 SAD 值的計算與 SAD 最小值的取得的作法都與前面改良後的 Three-Step Search 相同。

四、模擬與分析

在本章將介紹使用的模擬環境及模擬的平台，然後說明各個模擬的設定以及模擬之後的結果，最後再針對模擬結果做出分析，判斷演算法最佳化的結果是否合理。

(一) 模擬平台介紹

我們的模擬平台是建立在 IBM Cell B.E. SDK (IBM Cell Broadband Engine Software Development Kit)，這是一套包含了編譯器與模擬器的程式開發軟體，由編譯器產生執行檔並由模擬器來模擬執行結果。模擬器會在原本的 OS 上以虛擬系統的方式模擬出 Cell B.E.的架構，在模擬結果中可以得到總執行 cycle time、各類型指令的指令數及該類指令的總執行 cycle time。

(二) 各加速機制之模擬結果

以下將針對各種不同加速方式的模擬結果做分析，主要會針對完成計算參考 5 張影像的

移動向量的總 cycle、總指令數、CPI (cycle per instruction)、branch 造成的 stall 的 cycle 數、資料相依所造成的 stall 的 cycle 數，branch 的指令數等，使用的影像為 Foreman。

1. 以單顆 SPE 來執行程式

這是在完全不進行加速的情況下的模擬結果，是最基礎的數據。這個結果所代表的意義是在不加上任何加速機制的時候，以單顆 SPE 執行參考 5 張圖的 Motion Estimation 所需花的時間。

表 1. 以單顆 SPE 執行動作估測演算法之效能

cycle	Ins. count	CPI	Branch ins.
788568289	163670439	4.82	3201809

2. 以 8 顆 SPE 來執行程式

以 8 顆 SPU 將影像分割來平行處理，並且比較縱切與橫切在效能上的差別。

這兩組數據的差別是在兩個做法需要抓取的資料量不同，切割的時候交錯的範圍是一半的 Search Window 的大小乘上影像的寬度，縱向切割與橫向切割差別就在影像寬度，在橫向切割的時候寬度是 352 個像素，縱向切割時寬度則是 288 個像素，所以在以縱向切割的時候總傳輸量會比較小。

表 2. 以 8 顆 SPE 使用不同分割方法執行動作估測演算法之效能

	cycle	Ins. count	CPI	Branch ins.
8SPE(→)	99849193	68734436	1.45	433653
8SPE(↓)	98933062	68661246	1.44	397006

3. 使用 8 顆 SPU 並以 SIMD 方式來執行程式

這部份的結果是讓 8 顆 SPU 將影像分割後，在計算 SAD 最小值的時候以 SIMD 的方式來計算，影像以縱向分割的方式分割。

在加入以 SIMD 方式計算 SAD 最小值的做法之後運算的模式出現了改變，除了總運算時間的加快之外，在 SIMD 的機制也同時將 branch 指令數大量的減少，因為原本純量運算中計算 SAD 值的部份是以兩層的 for 迴圈來進行區塊內每個像素的相減取絕對值的動作，在使用 SIMD 的加速機制之後，因為每個向量變數可以存 16 個像素的值，只要使用 4 個向量變數來計算就可以 unroll 這兩層 for 迴圈，所以可以大量減少 branch 指令。

表 3. 以 SIMD 方式執行動作估測演算法之效能

cycle	Ins. count	CPI	Branch ins.
44193412	28383761	1.56	36062

4. 加上 multiple buffering 的 DMA 傳輸

在不同 buffer size 的模擬結果如圖 9.，在剛開始的時候因為 buffer size 太小，DMA 太快就結束資料抓取，且由於 DMA 結構的關係，若要 DMA 以較高效率的方式工作必須將 buffer size 設定為 128bytes 的倍數，所以沒辦法與計算的動作平行處理，而且設定 DMA 的動作次數太頻繁所以反而會把效能降低；在中間區段因為可以讓 DMA 抓取資料的時間配合上計算資料的時間，所以會逐漸加速，再來又因為 DMA 抓取資料的時間超過計算資料的時間，所以又開始減慢；在後段則是因為 buffer size 太大，在 DMA 抓取一次資料之後剩餘的資料量不足以填滿 buffer，必須另外抓取，所以效能就不會有較大幅度的變化。

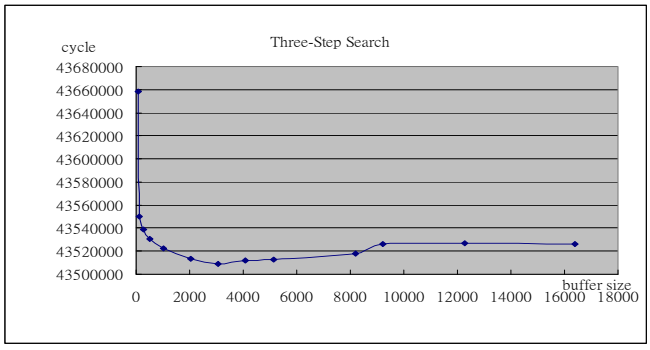

圖 9. 不同 buffer size 下的效能

5. 加上以 SIMD 方式來計算並減少 branch 指令

因為 branch 指令在 branch miss 時會有巨大的 penalty，所以用減少 branch 指令數的方式來加速；使用的 buffer size 經測試最高效能情況還是 buffer size=3072 bytes。

在加上減少 branch 指令的機制之後，branch 指令比單純使用 SIMD 的機制約可再減少 62.9% 的 branch 指令，由於 branch 指令減少，所以 branch miss 發生的機率也就跟著降低，所造成 stall 的 cycle 也就可以跟著減少。

表 4. 減少 branch 指令數後的動作估測演算法執行效能

cycle	Ins. count	CPI	Branch ins.
42432465	29228320	1.45	6904

6. 效能分析

以各加速機制加速之後的效能列出表 5.，以此表來分析效能的改進，在加入以 8 顆 SPE 平行運算的機制後，效能增進約為 7.97 倍。加入 SIMD 資料平行運算的機制後的效能增進為 2.24 倍，然後使用 multiple buffer 的機制抓取資料的效能增進為 1.04 倍，減少 branch 指令數的方法則可以增進 1.03 倍。累積的效能增進如圖 10.。

表 5. 經過不同程度的加速後的運算效能

single SPE	8 SPE	SIMD	multiple buffering	less branch
788568289	98933062	44193412	43509151	42432465

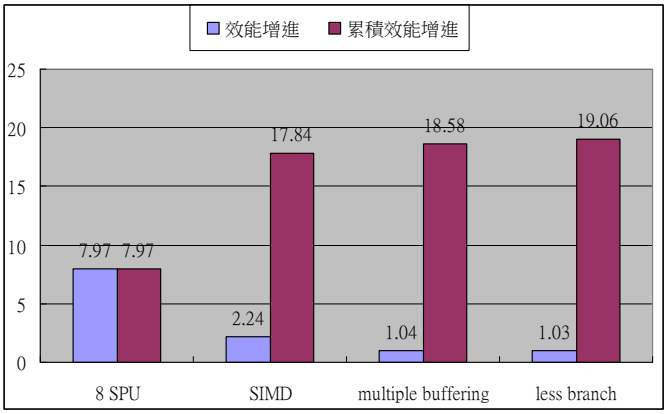

圖 10. 增進的效能與累積的效能增進

減少 branch 的機制之所以看起來效果不好的原因是因為 branch 在前面的機制像以 8 顆 SPE 分割影像以及以 SIMD 向量化運算都已經將 branch 的數量大幅減少了，所以到這個步驟的時候能減少的 branch 指令數已經不多了，所以加速的效能較不明顯。

而 multiple buffering 的加速效果不好的原因是因為總資料量不夠大，以目前的動作而言，一次最多也只會傳送 17280 個 byte，在以最大的 buffer size 的情況下只夠傳輸一次，所以加速效能不明顯，但若是使用在高解析度影像的時候，因為資料傳輸量大，所以就可以看出它真正的加速效果。

7. Diamond Search 效能分析

Diamond Search 在實現最佳化運算之後效能及與 Three-Step Search 的比較如表 6.，DS 由於其從中心擴散的工作原理的關係，且使用的影像是屬於動作量較小的，所以使用 Diamond Search 會有較高的效能。

表 6. Diamond Search 在經過不同程度的加速後的運算效能

	single SPU	8 SPU	SIMD	multiple buffering	less branch
DS (cycle)	465806955	58225869	24882850	24158107	23228949
TSS (cycle)	788568289	98933062	44193412	43509151	42432465
比較	1.69	1.70	1.78	1.80	1.82

而圖 11 則是 Diamond Search 的效能增進與 Three-Step Search 的比較圖，在加入 multiple buffering 機制時 Diamond Search 的效能增進較少有可能是因為依照我的設計，Diamond Search 在計算的時候會有擴散的動作，因為無法確認向外擴散的次數，這樣會變成無法估計資料抓取的次數，勢必要使用判斷式來檢查資料是否抓完，所以在我的設計上 Diamond Search 在進行擴散的時候就不會抓取資料，可能這就是影響效能的原因。

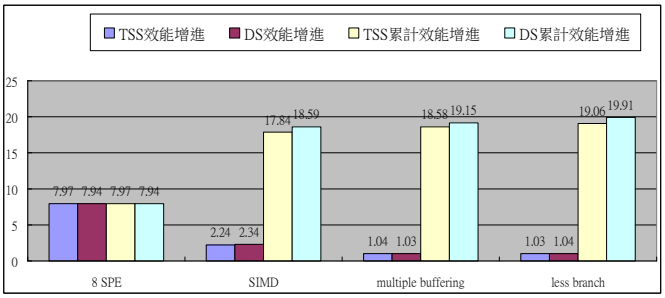

圖 11. Diamond Search 的效能增進與 Three-Step Search 的比較圖

8. 使用不同來源影像的效能分析

使用的第二組影像為 Stefan，這組影像與前面所使用的影像相比，移動變化量會比較大。使用兩種演算法運算結果如表 7。由結果可以看出在移動向量較大的情況下，使用 Diamond Search 演算法會有較差的效能，而在相同的情況下 Three-Step Search 的效能則不會有太大的改變。

表 7. 兩組影像使用不同演算法的運算效能

	Foreman	Stefan
DS (cycle)	23228949	26905246
TSS (cycle)	42432465	42403161
比較	1.82	1.58

五、結論

本論文的目的為設計動作估測演算法在 Cell B.E.處理器架構上進行運算最佳化。在最佳化的演算法架構中，我們採用了以下幾種機制：

(1) 以多顆 SPE 同步處理，進行 thread-level 的平行化處理

(2) 以 SIMD 的向量運算進行 data-level 的平行化處理

(3) 以減少 branch 指令的方式降低 branch miss 時的 penalty 對程式效能的影響

(4) 以 multiple buffering 的方式減少大量資料傳輸時，運算單元等待資料就緒的時間

經過模擬驗證之後，以上列四種機制對動作估測演算法確實可以讓 Three-Step Search 與 Diamond Search 兩種演算法在最佳化後可以得到 19~20 倍的效能，可以達到每張影像的處理時間為 13.26ms。

在未來會將 H.264/AVC 的編解碼流程同樣的進行最佳化，這樣一來 Cell 就可以符合完整的視訊編解碼的標準，期望可以實現即時編解碼的功能。

六、參考文獻

[1] Jim Kahle, “Cell Architecture”, International Symposium on Microarchitecture archive Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture, 2005

[2] C. R. Johns, D. Shippy, H. P. Hofstee, J. A. Kahle, M. N. Day, T. R. Maeurer, "Introduction to the Cell Multiprocessor," IBM Journal of Research and Development, Vol. 49, No. 4/5, July/Sept. 2005.

[3] Peter Hofstee, “Cell Today and Tomorrow”, Ph.D., Cell Chief Scientist and Chief Architect

[4] IBM , “Synergistic Processor Unit Instruction Set Architecture” Version 1.2

[5] Redbooks, “Programming the Cell Broadband Engine™ Architecture: Examples and Best

Practices", published on 8 August 2008

[6] Brian Flachs, H. Peter Hofstee, IBM, Michael Gschwind, Martin Hopkins, Sony Computer Entertainment, Toshiba ,Takeshi Yamazaki, Yukio Watanabe, "SYNERGISTIC PROCESSING IN CELL'S MULTICORE ARCHITECTURE" , published by the IEEE Computer Society

[7] A. Hirano, K. Iinurna, T. Koga, T. Ishiguro Y.Iijima, "Motion-compensated interframe coding for video conferencing," in Proc. NTC 81, New Orleans, LA, Nov./Dec. 1981.

[8] Bing Zeng, Ming L. Liou, Renxiang Li," A New Three-Step Search Algorithm for Block Motion Estimation", IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 4, NO. 4, AUGUST 1994

[9] Ashraf Ali Kassim, Jo Yew Tham, Maitreya Ranganath, Surendra Ranganath, "A Novel Unrestricted Center-Biased Diamond Search Algorithm for Block Motion Estimation", IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 8, NO. 4, AUGUST 1998

[10] Kai-Kuang Ma, Shan Zhu, " A New Diamond Search Algorithm for Fast Block-MatchingMotion Estimation", IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 9, NO. 2, FEBRUARY 2000

[11] Chun Ho Cheung, Chi-Wai Lam , Lai-Man Po," A NEW CROSS-DIAMOND SEARCH ALGORITHM FOR FAST BLOCK MATCHING MOTION ESTIMATION"

[12] Artemakis Artemiou, Despo Othonos, Pedro Trancoso, " Data Parallel Acceleration of Decision Support QueriesUsing Cell/BE and GPUs ", CF'09, May 18–20, 2009, Ischia, Italy.

[13] Haibo Lin, John Kevin O'Brien, Ling Shao, Tao Liu, Tong Chen, "DBDB: optimizing DMA transfer for the Cell BE Architecture", ICS '09

[14] Christos D. Antonopoulos, Konstantis Daloukas, Nikolaos Bellas, "Implementation of a wide-angle lens distortion correction algorithm on the cell broadband engine ", ICS '09

[15] Daniele Paolo Scarpazza, Gregory F. Russell, " High-performance regular expression scanning on the Cell/B.E. processor", ICS '09.

[16] Leonel Sousa, Svetislav Momcilovic, "A PARALLEL ALGORITHM FOR ADVANCED VIDEO MOTION ESTIMATION ON MULTICORE ARCHITECTURES", International Conference on Complex, Intelligent and Software Intensive Systems 2008.

[17] Leonel Sousa, Svetislav Momcilovic," PARALLEL ADVANCED VIDEO CODING: MOTION ESTIMATION ON MULTI-CORES", 2008 SCPE.