行政院國家科學委員會專題研究計畫　成果報告

針對多重資料記憶體模組 DSP 架構探討指令排程法

計畫類別： 個別型計畫
計畫編號： NSC94-2213-E-009-117-
執行期間： 94 年 08 月 01 日至 95 年 07 月 31 日
執行單位： 國立交通大學資訊工程學系(所)

計畫主持人： 陳正

報告類型： 精簡報告

處理方式： 本計畫可公開查詢

中　華　民　國 95 年 10 月 25 日

行政院國家科學委員會補助專題研究計畫 þ 成 果 報 告
□期中進度報告

針對多重資料記憶體模組 DSP 架構探討指令排程法

計畫類別：þ 個別型計畫　　□ 整合型計畫
計畫編號：NSC 94－2213－E－009－117－
執行期間：　　94 年 8 月 1 日至 95 年 7 月 31 日

計畫主持人：陳正
共同主持人：
計畫參與人員：李宜軒、蔡明憲、廖哲瑩

成果報告類型(依經費核定清單規定繳交)：þ精簡報告　□完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□ 涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：國立交通大學資訊工程學系

中　華　民　國　　95 年 10 月 24 日

針對多重資料記憶體模組 DSP 架構探討指令排程法
A Study of Instruction Scheduling Algorithms for DSP Architecture with Multiple Data-memory Modules

計畫編號：94-2213-E-009-117

一、中英文摘要

在數位訊號處理器的架構設計中，為了達到高效能及低功耗，時常採用 Non-orthogonal 架構 (同時具有多重資料記憶體模組及異質性暫存器集合特性)；而要能夠確實得到此架構提供的優勢，則必須配合有效的編譯技術。去年計畫我們已提出三個包含變數分割機制的指令排程法，在本計畫中我們將其延伸，加入暫存器配置的考量，提出二個用於 Non-orthogonal 架構的指令排程法，並制定一套虛擬架構及相關數學模組評估其初步效能。根據我們的分析，新提出的方法不僅能有效改進之前方法的效能，而且具有延伸性適用於多種性質類似的架構，可用來評估不同資源對整體效能造成的影響。
關鍵詞：數位訊號處理器、多重資料記憶體模組、異質性暫存器集合、指令排程

Abstract

In order to meet demands for higher performance and lower power consumption, many high-end digital signal processors (DSPs) commonly employ non-orthogonal architecture characterized by multiple memory banks and heterogeneous register sets. To harvest the benefits provided by this architecture, effective compiler techniques are required. Last year we have proposed three variable partitioning and instruction scheduling methods. In this project, we extend them to consider the register assignment phase, and propose two instruction scheduling algorithms used for non-orthogonal DSP architecture. A hypothetical machine model and related analytic models are also defined to evaluate their preliminary performance. Based on our analyses, two new algorithms obviously outperform previous proposed methods. Moreover, the proposed algorithm is suitable for various architectures with similar features, which can be used to study the influence of differing number of resources on the scheduling result.
Keywords: *Digital Signal Processor* (*DSP*), *Multiple Data-memory Banks*, *Heterogeneous Register Sets*, *Non-orthogonal Architecture*, *Instruction Scheduling*

二、前言

隨著多媒體通訊日益劇增的需求，陸續發展出許多新的方法來對影像、音訊及圖形等做壓縮及傳輸的動作。這些訊號處理的過程通常包含大量且規則的運算，為了能即時 (real-rime) 完成運算並顧及價格/效能比 (cost/performance)，便產生了許多針對不同應用程式所設計的數位訊號處理器 [1-4]。近年來由於硬體製程技術的進步，各種嵌入式系統晶片

(embedded system chip) 及單晶片系統 (system-on-a-chip, SoC) 越來越受到廣泛重視，也間接使得數位訊號處理器的研發無論是在學界或業界，均持續佔有一席之地 [5-7]。

大部分 DSP 應用程式都有重覆性 (iterative)，主要由規則相依迴圈 (uniform loop) 組成，運算過程則以乘法和加法為主 [1-4]。為了達到高效能 (high performance) 和低功耗 (low power consumption) 的需求，DSP 常採用 non-orthogonal 架構，意即架構中同時具有多重資料記憶體模組 (multiple memory banks) 及異質性暫存器集合 (heterogeneous register sets) 二種特性 [8]。多個資料記憶體模組配合獨立匯流排，可以同時存取多筆資料，以便開發潛在的記憶體頻寬 (potential memory bandwidth) [9-11]。異質性暫存器集合則是將 DSP 中少量暫存器依功能分成數個集合，藉由限制不同動作能夠存取的暫存器，以簡化硬體設計的複雜度。從 DSP 應用程式的特性來看，這種架構確實有助於提升效能；但根據實際硬體架構的設計，並非所有資料都能被同時存取，必須符合特定的資料平行存取條件 (parallel access conditions)，並使用合適的暫存器。由此可見，在不規則的 non-orthogonal DSP 架構下，如何將資料做適當的分割 (partition)，並妥善安排指令執行順序及暫存器配置，便成為能否真正提升執行效能的關鍵 [8, 12-20]。

本實驗室多年來持續探討排程的研究議題，除了傳統多處理機上的迴圈平行化之外，近期對 DSP 架構的指令排程也有涉獵。在去年的計畫中我們已針對變數分割 (variable partition) 機制提出三個指令排程法，在本計畫中將對其做進一步延伸，並增加新的排程目標，希望能藉由指令排程的技術充分開發 non-orthogonal DSP 架構的潛在特性，達到提升執行效能的目的。

三、研究目的

無論是在何種系統架構下，排程技術的好壞往往均是影響整體效能的關鍵，相較於 general purpose CPU，non-orthogonal DSP 架構明顯具有不規則性 (irregularity)，也因此更需要適合的編譯技術，才能有效開發其潛在特性。對於 non-orthogonal DSP 架構，完整的編譯過程應包含以下步驟：uncompacted code generation、code compaction、instruction scheduling、memory bank assignment (or variable partition) 及 accumulator/register assignment [12]。這些步驟彼此相依但可獨立執行，因此有不少相關文獻針對其中數個步驟設計演算法。在去年計畫中我們已提出三個涵蓋前四個步驟的指令排程法，在本計畫中將額外考慮暫存器配置 (register assignment) 的步驟，設計完整且具延伸性的指令排程法，同時以縮短執行時間及減少指令數量為排程目標。另外我們也根據現有 non-orthogonal 架構，增加系統資源個數及平行條件定義一套虛擬架構模組 (hypothetical machine model)，配合具延伸性的指令排程法，深入探討不同系統資源對整體效能造成的影響。

四、文獻探討

在介紹相關文獻之前，我們先大致描述 DSP 應用程式的表現方式。之前提過 DSP 應用程式以迴圈為主，而迴圈又是消耗執行時間最多的部分，所以大部分的 DSP 編譯技術都致力於開發迴圈指令的平行度來縮短執行時間。一個迴圈元素 (iteration) 通常用資料流程圖

(multi-dimensional data flow graph, MDFG) 表示，其中節點代表指令，有向邊代表資料相依性，邊上的數字則代表迴圈元素間的相依距離 (inter-iteration dependence) [8, 12-20]。合理的排程結果必須在不違反資料相依性及系統資源限制下，執行所有資料流程圖中的節點各一次；另外為了進一步提升效能，常用 retiming 技巧重組迴圈元素，以開發迴圈元素間的平行度 [21]。使用 retiming 技巧之後程式會分成 prologue、repetitive pattern 及 epilogue 三部分，其中 prologue 和 epilogue 是額外產生且必須獨立執行，嚴格來說是 overhead，但只要 repetitive pattern 重複次數夠多，prologue 和 epilogue 的執行時間將可忽略不計。

在 non-orthogonal DSP 架構完整的編譯過程有以下五個步驟：uncompacted code generation、code compaction、instruction scheduling、memory bank assignment (or variable partition) 及 accumulator/register assignment [12]，以下我們根據各方法所涵蓋的步驟大致分類相關文獻。[9-11] 針對 memory bank assignment 提出不同的變數分割機制，目標是將可能會被平行存取的陣列變數分散儲存，以便執行時同時存取。[22-24] 針對 accumulator/register assignment 步驟，設計適合的演算法配合 heterogeneous register sets 的不規則性。[13-16] 提出的方法包含前四個編譯步驟，同時解決變數分割及指令排程的問題，但假設暫存器個數無限，並未考慮 accumulator/register 的配置。[8, 12, 20] 同時選擇 Motorola DSP56000 為目標架構，提出涵蓋完整五個步驟的指令排程法。[8] 提出的方法主要是先用序列排程法 (list scheduling) 做指令排程，並將 variable partition 及 accumulator/register assignment 對應成 graph coloring 問題，再設計簡單的 heuristic 來解 graph coloring，達到變數分割及暫存器配置的目的。[12] 設計的方法較為複雜，先以序列排程法得到指令排程結果後，使用類似 graph labeling 的方式，以耗時的 simulated annealing 同時解決變數分割及暫存器配置的問題。我們認為 [8, 12] 提出的方法雖然效能不錯，但它們都沒有描述如何偵測 accumulator/register spill 以及產生對應 spill codes 的動作；既然 DSP 架構暫存器個數有限，accumulator/register spill 想必時常發生，缺乏相關解決動作將使方法顯得不夠完善。另外 [8, 12] 也只針對 Motorola DSP56000 架構設計，當系統資源增加時沒有延伸性，在實用性方面亦較為受限。

五、研究方法

之前提到雖然可以針對 non-orthogonal DSP 架構的五個編譯步驟設計獨立的演算法，但因為它各步驟彼此之間有嚴格的資料相依性，若是能同時考慮多個步驟來設計編譯技術，將有助於整體效能的提升。有鑑於此，在本計畫中我們延伸去年的研究成果，提出二個涵蓋所有步驟的指令排程法和一套虛擬架構，以下分別介紹。

首先我們描述 [8, 12, 20] 提出的方法考慮 accumulator/register assignment 的方式。這三個方法在排程指令時，都先假設暫存器個數無限，並使用 symbolic accumulator/register 來存取變數；得到初步排程結果後，再考慮 accumulator/register 個數的限制，加入需要的 spill codes。這種作法承襲自 general purpose CPU 的編譯技術，但是在 heterogeneous register sets 架構下，由於暫存器個數相對較少，容易產生較多的 spill codes，而且這些 spill codes 必須獨立執行，也會使整體執行時間延長。為了解決這個問題，我們提出一個名為 rotation scheduling with spill codes predicting (RSSP) 的方法，針對 Motorola DSP56000 架構設計，以縮短執行時間為排程目標。RSSP 共有六個執行步驟，主要特色是在排程指令之前，先假

設暫存器個數無限，將輸入的 MDFG 轉成 translated data acyclic graph (TDAG)，盡量移除記憶體存取指令。接著分析 TDAG 的結構 (topology)，在可能產生 accumulator spill 的地方加入 spill codes，再根據修改後的 TDAG 來做指令排程，於過程中避免 register spill 的產生。由此執行步驟可見，既然我們已事先預測 accumulator spill 並產生對應的 spill codes，在指令排程時可以讓 spill codes 與其他指令平行執行，相較於 [8, 12, 20] 提出的方法，確實有助於得到較短的排程結果。在下章節中我們會列出 RSSP 的評估結果，相關研究內容已發表於會議論文 [17] 並整理成期刊論文 [18]。

雖然根據評估 RSSP 效能不差，但它預測 accumulator spill 的機制僅適用於一個 data ALU 及二個 accumulators 的架構 (例如 Motorola DSP56000)，實用性頗為受限。另外在 non-orthogonal DSP 架構下，除了執行間之外，如何減少指令數量也是個重要的研究議題。因此我們提出第二個名為 rotation scheduling with spill codes avoiding (RSSA) 的方法，適用於多種性質類似的架構，可以處理系統資源 (包含 data ALU、memory bank 及 accumulator/ register) 數量不同的情形，同時以縮短執行時間及減少指令數量為排程目標。RSSA 共有五個執行步驟，部分動作直接承襲 RSSP，但設計新的機制解決 accumulator spill，不再經由分析 TDAG 預測其產生。另外為了減少指令個數，當 accumulator spill 發生時，我們希望令計算結果盡量暫存於 register (須增加一個 spill code)，僅在 register 數目不足時才用 memory 暫存 (須增加二個 spill codes)。為了達到以上目的，RSSA 將指令排程分成二個步驟 (大部分演算法均直接排程所有指令)，先排「讀取記憶體」以外的指令並解決 accumulator/register spill，之後再將記憶體讀取指令排入。我們也定義數個變數，在指令排程過程中動態更新 (dynamic update)，用來偵測及解決 accumulator/register spill。這個機制可以讓 RSSA 在處理 accumulator spill 時 register 保持淨空，使計算結果優先暫存於 register，只在 register spill 發生時才用 memory 暫存。由以上執行步驟可見，RSSA 除了在縮短排程長度之外，同時避免產生 spill code，真的需要 spill code 時也盡可能減少其數量，確實能達到預計的二個排程目標。我們另外也根據 Motorola DSP56000 架構，增加系統資源個數及平行條件定義一套 hypothetical machine model，用來模擬 scalable non-orthogonal DSP 架構，配合具延伸性的 RSSA，深入探討不同系統資源對整體效能造成的影響。RSSA 詳細評估結果以及使用虛擬架構的分析留待下章節討論。關於 RSSA 的相關研究成果已發表於期刊論文 [19]。

六、結果與討論

本計畫中提出的二個用於 non-orthogonal DSP 架構的指令排程法，我們除了演算法的設計之外，也制定對應的數學模組計算所需執行時間，並選取數個表示 DSP 應用程式的資料流程圖來做初步測試評估。表格 1~2 列出在 Motorola DSP56000 架構下，各個 DSP 應用程式使用不同排程法得到的單一迴圈元素排程長度以及包含的指令個數。在此由於系統資源數量及架構固定，所以使用 RSSP 和 RSSA 會得到相同結果。由這些結果看來，無論在排程長度或是指令個數方面，RSSP、RSSA 和 [8] 提出的方法效能都明顯較佳，主要原因是它們根據 TDAG 而其他方法根據 MDFG 來做指令排程。之前已介紹 TDAG 是由 MDFG 簡化而來，在假設 accumulator/ register 個數無限的條件下盡量移除記憶體存取指令；既然 TDAG 包含的指令個數較少，根據它來排程當然可以得到較短的排程長度及較少的指令個數。至於 RSSP 和 RSSA 得到的排程長度又比 [8] 提出方法的要短原因，則是在使用 list

scheduling 得到初始排程後透過 retiming 技巧開發迴圈元素間的平行度，能有效縮短執行時間。表格 3 列出的是各個 TDAG 的相關資訊，詳細比較表格 1~3 的內容，即可發現 RSSP 和 RSSA 得到的排程長度幾乎等於 TDAG 中 ALU 指令的個數，既然 Motorola DSP56000 只有一個 data ALU，表示 RSSP 和 RSSA 已趨近最佳解，因為所有 ALU 指令都必須由唯一的 data ALU 執行。而在指令個數方面，RSSP 和 RSSA 得到的結果也和 TDAG 節點數相去不遠，表示我們提出的方法不會增加太多 spill code，符合預期的排程目標。我們另外也套用由數學模組求出的公式，繪出各個應用程式在不同迴圈大小時使用不同排程法所得到的整體執行時間，評估結果大致與表格 1~2 類似，因資料量龐大在此省略，詳細內容可見已發表論文 [17, 19]。

接著我們使用虛擬架構產生多個資源個數不同的 non-orthogonal DSP，配合 RSSA 針對前述二個評估項目，討論不同系統資源對整體效能造成的影響。表格 4~8 分別列出增加 accumulator、register、data ALU 及 memory bank 個數之後，RSSA 得到的單一迴圈元素排程長度及指令個數。其中陰影代表已達最佳解 (排程長度方面指已達對應 TDAG 的 critical path 長度或 ALU 指令個數，指令個數方面指未增加任何 spill code)，粗體代表增加該項系統資源有助於提升效能。由這些結果看來，增加 accumulator 個數最能避免產生 spill code，若欲縮短排程長度，則必須同時增加 data ALU 及 accumulator 數目。若僅以選取的這些 DSP 應用程式而言，只要系統包含 2 個 data ALUs、4 個 accumulators、6 個 registers 和 2 個 memory bank，在二個評估項目幾乎都可以得到最佳解。最後值得一提的是，雖然表格 8 看起來在二個評估項目都有不錯效能，但由於增加 memory bank 同時也必須增加獨立匯流排，硬體成本很大，在價格/效能比例上較不值得，因此我們並不推薦使用增加 memory bank 的方式來提升效能。

七、計畫結果自評

在本計畫中我們針對 non-orthogonal DSP 架構，提出二個包含完整編譯步驟的指令排程法以及一套虛擬架構，並制定對應的數學模組，選擇數個 DSP 應用程式做初步的效能評估。總體而言，本計畫大致達成以下幾點目標：

1. 提出 RSSP 指令排程法，針對 Motorola DSP56000 架構設計，使用預測 accumulator spill 的方式預先產生對應的 spill code，令它們能與其他指令平行執行，並改進去年計畫提出方法的潛在問題，有效縮短排程長度。
2. 提出 RSSA 指令排程法，為 RSSP 的延伸，適用於系統資源數量不同但性質類似的架構，使用避免產生 spill code 的機制，盡量縮短執行時間並減少指令個數。
3. 根據 Motorola DSP56000 架構特性，增加系統資源數量及平行條件，定義一套虛擬架構模擬 scalable non-orthogonal DSP。
4. 制定以上二種指令排程法對應的數學模組，並選擇數個表示 DSP 應用程式的資料流程圖，對提出的方法做初步效能評估。
5. 用定義的虛擬架構產生多個資源個數不同的 non-orthogonal DSP，配合有延伸性的 RSSA，討論不同系統資源對整體執行效能的影響。

由以上幾點可知，本計畫確實能將 non-orthogonal DSP 架構上的指令排程議題做詳細深入的探討，改善之前提出方法的潛在問題，嘗試提出新的機制，並制定數學模組評估其

初步效能。整體看來，我們提出的方法確實都能達到預期的結果，有效縮短執行時間並減少 spill code 的產生。這些研究成果均已發表於會議或期刊論文，後續發展及其他相關排程議題我們也將持續研究。

參考文獻

[1] J. Eyre and J. Bier, "The Evolution of DSP Processors", *IEEE Signal Processing Magazine*, Vol. 17, Issue 2, pp. 43-51, March 2000.

[2] S.A. Mujbata, "Trends in Digital Signal Processors", *Proc. of International Symposium on VLSI Technology, Systems, and Applications*, pp. 108-111, 1999.

[3] V.K. Madisetti, **VLSI Digital Signal Processors: An Introduction to Rapid Prototyping and Design Synthesis**, Butterworth-Heinemnn, 1995.

[4] P. Lapsley, J. Bier, A. Shoham, and E.A. Lee, **DSP Processor Fundamentals: Architectures and Features**, Berkeley Design Technology, Inc. 1994-1996.

[5] H. de Man, "System-on-Chip Design: Impact on Education and Research", *IEEE Design & Test of Computers*, Vol. 16, Issue 3, pp. 11-19, July-sep. 1999.

[6] G. Silcott, J. Wilson, N. Peterson, W. Peisel, and K.L. Kroeker, "SOCs Drive New Product Development", *IEEE Computer*, Vol. 32, Issue 6, pp. 61-66, June 1999.

[7] J. Noguera and R.M. Badia, "A HW/SW Partitioning Algorithm for Dynamically Reconfigurable Architectures", *Proc. of Conference and Exhibition on Design, Automation, and Test in Europe*, pp. 729-734, 2001.

[8] J. Cho, Y. Paek, and D. Whalley, "Efficient Register and Memory Assignment for Non-orthogonal Architectures via Graph Coloring and MST Algorithms", *Proc. of ACM Joint conference LCTES-SCOPES*, pp. 130-138, Jun. 2002.

[9] M.A.R. Saghir, P. Chow, and C.G. Lee, "Towards Better DSP Architectures and Compilers", *Proc. of International Conference on Signal Processing Applications and Technology*, pp. 658-664, Oct. 1994.

[10] R. Leupers and D. Kotte, "Variable Partitioning for Dual Memory Bank DSPs", *Proc. of International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, pp. 1121-1124, 2001.

[11] Q. Zhuge, B. Xiao, E.H.M. Sha, "Exploring Variable Partitioning in Dual Data-memory Bank Processors", *Proc. of 34th International Symposium on Micro-architecture,* pp. 42-55, Dec. 2001.

[12] A. Sudarsanam and S. Malik, "Simultaneous Reference Allocation in Code Generation for Dual Data Memory Bank ASIPs", *ACM Transactions on Design Automation of Electronic Systems*, Vol. 5, No. 2, pp. 242-264, April 2000.

[13] Q. Zhuge, E.H.M. Sha, B. Xiao, and C. Chantrapornchai, "Efficient Variable Partitioning and Scheduling for DSP Processors with Multiple Memory Modules", *IEEE Transactions on Signal Processing*, Vol. 52, No. 4, pp. 1090- 1099, April 2004.

[14] M.A.R. Saghir, P. Chow, and C.G. Lee, "Exploiting Dual-memory Banks in Digital Signal Processors", *Proc. of 7th International Conference on Architecture Support for Programming*

Language and Operating Systems, pp. 234-243, 1996.

[15] Y.H. Lee and C. Chen, "Efficient Variable Partitioning and Scheduling Methods of Multiple Memory Modules for DSP", *Proc. of 10th Workshop on Compiler Techniques for High-Performance Computing*, pp. 80-89, March 2004.

[16] Y.H. Lee and C. Chen, "An Effective Variable Partitioning and Scheduling Algorithm for DSP with Multiple Memory Modules", *Proc. of International Computer Symposium*, Dec. 2004.

[17] Y.H. Lee and C. Chen, "An Efficient Code Generation Algorithm for Non- orthogonal DSP Architecture", *Proc. of the 11th Workshop on Compiler Techniques for High-Performance Computing*, pp. 49-58, March 2005.

[18] Y.H. Lee and C. Chen, "An Efficient Code Generation Algorithm for Non- orthogonal DSP Architecture", submitted to *Journal of VLSI Signal Processing.*

[19] Y.H. Lee and C. Chen, "An Effective and Efficient Code Generation Algorithm for Uniform Loops on Non-orthogonal DSP Architecture", accept and to appear to *Journal of Systems and Software*.

[20] W.T. Shiue, "Energy-efficient Backend Compiler Design for Embedded Systems", *Proc. of 10th International Conference on Electrical and Electronic Technology*, Vol. 1, pp. 103-109, Aug. 2001.

[21] C.E. Leiserson and J.B. Saxe, "Retiming Synchronous Circuitry", *Algorithmica*, Vol. 6, No. 1, pp. 5-35, June 1991.

[22] J.M. Daveau, T. Thery, T. Lepley, and M. Santana, "A Retargetable Register Allocation Framework for Embedded Processors", *Proc. of ACM SIGPLAN/ SIGBED*, pp. 202-210, June 2004.

[23] B. Scholz and E. Eckstein, "Register Allocation for Irregular Architectures", *Proc. of ACM Joint Conference LCTES-SCOPES*, pp. 139-148, June 2002.

[24] X. Zhuang, T. Zhang, and S. Pande, "Hardware-managed Register Allocation for Embedded Processors", *Proc. of ACM SIGPLAN/SIGBED*, pp. 192-201, June 2004.

表格 1. 實驗結果 (排程長度).

Benchmarks	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
Wave Digital Filter	7	9	9	8	6	8.5	6	5
Filter	8	13	13	9	11.5	9	6	5.5
IIR Filter 2D	20	29	33	25	27.5	28	16	16
Forward-substitution	7	12	12	9	10	11.5	5	5.5
THCS	6	8	8	6	6.5	5.5	4	4
DFT	16	21	21	18	21	18.5	13	12.5
Floyd-Steinberg	20	36	37	29	32.5	32	18	17.5
Transmission Line	15	20	21	19	18	18	12	12
IIR Filter 1D	11	15	15	11	14	--	8	8
Differential Equation Solver	16	20	21	18	21.5	--	13	11.5
All-pole Lattice Filter	21	37	37	35	28	--	17	16
Elliptic Filter	42	62	66	56	69	--	36	34

[1] Cho et al., 2002
[2] Sudarsanam and Malik, 2000
[3] Shiue, 2001
[4] RSVR
[5] RSF
[6] RST
[7] RSSP & RSSA (with RSVR mechanism)
[8] RSSP & RSSA (with RSF mechanism)
[9] RSSP & RSSA (with RST mechanism)

表格 2. 實驗結果 (指令個數).

Benchmarks	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
Wave Digital Filter	14	16	16	16	15.5	16	14	13
Filter	11	16	16	16	16	16	11	10.5
IIR Filter 2D	37	64	68	64	64	64	37	37
Forward-substitution	11	20	20	18	17.5	18	11	10.5
THCS	10	16	16	14	14.5	14	10	9.5
DFT	33	48	49	44	44	44	32	30
Floyd-Steinberg	39	68	70	59	59	59.5	39	39.5
Transmission Line	28	48	48	42	42	42	29	29
IIR Filter 1D	20	32	32	30	29.5	--	18	18
Differential Equation Solver	25	44	44	37	37	--	26	25.5
All-pole Lattice Filter	37	60	60	51	51.5	--	35	34.5
Elliptic Filter	77	136	136	125	116.5	--	75	72

表格 3. TDAG 相關資訊.

Benchmarks	Number of ALU nodes in G_t	Critical path of G_t	Number of nodes in each TDAG				
			original	unfold 2	tiled 2×1	unfold 3	tiled 3×1
Wave Digital Filter	4	6	14	13	13.5	12.6	13.3
Filter	4	7	11	10.5	10.5	10.3	10.3
IIR Filter 2D	16	7	34	34	34	34	34
Forward-substitution	5	6	11	10.5	10.5	10.3	10.3
THCS	4	4	10	9.5	10	9.3	10
DFT	12	7	32	28	30	26.6	29.3
Floyd-Steinberg	17	12	38	37.5	37.5	37.3	37.3
Transmission Line	12	10	26	26	26	26	26
IIR Filter 1D	8	6	17	17	--	16.6	--
Differential Equation Solver	11	11	25	22.5	--	21.6	--
All-pole Lattice Filter	15	18	33	29	--	27.6	--
Elliptic Filter	34	19	65	58	--	55.6	--

表格 4. 實驗結果 (增加 accumulator 個數).

	1 ALU, 2 acc, 4 reg, 2 mem						1 ALU, 3 acc, 4 reg, 2 mem					
	RSVR		RSF		RST		RSVR		RSF		RST	
	len	#	*len*	#	*len*	#	*len*	#	*len*	#	*len*	#
Wave Digital Filter	6	14	5	13	5.5	13	6	14	5	13	5.5	13
Filter	6	11	5.5	10.5	5	10.5	6	11	5.5	10.5	**4.5**	10.5
IIR 2D	16	37	16	37	16	37	16	**34**	16	**34**	16	**34**
Forward-substitution	5	11	5.5	10.5	5	10.5	5	11	5.5	10.5	5	10.5
THCS	4	10	4	9.5	4	10	4	10	4	9.5	4	10
DFT	13	32	12.5	30	13	31.5	**12**	32	**12**	**28.5**	**12**	**31**
Floyd-Steinberg	18	39	17.5	39.5	17	39.5	18	**38**	17.5	**38.5**	17	**38.5**
Transmission Line	12	29	12	29	12	29	12	**27**	12	**27**	12	**27**
IIR 1D	8	18	8	18	--	--	8	**17**	8	**17**	--	--
Equation Solver	13	26	11.5	25.5	--	--	13	26	11.5	**25**	--	--
All-pole Lattice Filter	17	35	16	34.5	--	--	17	**33**	16	**31.5**	--	--
Elliptic Filter	36	75	34	72	--	--	**35**	**70**	**30.5**	**66.5**	--	--

表格 5. 實驗結果 (增加 register 個數).

	1 ALU, 2 acc, 4 reg, 2 mem						1 ALU, 2 acc, 6 reg, 2 mem					
	RSVR		RSF		RST		RSVR		RSF		RST	
	len	#	*len*	#	*len*	#	*len*	#	*len*	#	*len*	#
Wave Digital Filter	6	14	5	13	5.5	13	6	14	5	13	**5**	13
Filter	6	11	5.5	10.5	5	10.5	**5**	11	**5**	10.5	**4**	10.5
IIR 2D	16	37	16	37	16	37	16	37	16	37	16	37
Forward-substitution	5	11	5.5	10.5	5	10.5	5	11	**5**	10.5	5	10.5
THCS	4	10	4	9.5	4	10	4	10	4	9.5	4	10
DFT	13	32	12.5	30	13	31.5	13	32	12.5	**28.5**	**12**	**30.5**
Floyd-Steinberg	18	39	17.5	39.5	17	39.5	18	39	17.5	39.5	17	39.5
Transmission Line	12	29	12	29	12	29	12	29	12	29	12	29
IIR 1D	8	18	8	18	--	--	8	18	8	18	--	--
Equation Solver	13	26	11.5	25.5	--	--	13	**25**	12	**23**	--	--
All-pole Lattice Filter	17	35	16	34.5	--	--	17	35	16	**33**	--	--
Elliptic Filter	36	75	34	72	--	--	**35**	**73**	**30.5**	**69.5**	--	--

表格 6. 實驗結果 (增加 data ALU 個數).

	1 ALU, 2 acc, 4 reg, 2 mem						2 ALU, 2 acc, 4 reg, 2 mem					
	RSVR		RSF		RST		RSVR		RSF		RST	
	len	#	*len*	#	*len*	#	*len*	#	*len*	#	*len*	#
Wave Digital Filter	6	14	5	13	5.5	13	6	14	5	13	5.5	13
Filter	6	11	5.5	10.5	5	10.5	6	11	5.5	10.5	**4.5**	10.5
IIR 2D	16	37	16	37	16	37	**12**	39*	**13**	40*	**12.5**	40*
Forward-substitution	5	11	5.5	10.5	5	10.5	**4**	11	**4**	10.5	**4.5**	10.5
THCS	4	10	4	9.5	4	10	**3**	10	**3**	9.5	**3**	10
DFT	13	32	12.5	30	13	31.5	**12**	34*	**11.5**	32*	13	35*
Floyd-Steinberg	18	39	17.5	39.5	17	39.5	**15**	41*	**16.5**	45.5*	17.5	45.5*
Transmission Line	12	29	12	29	12	29	**8**	**28**	**11**	29.5*	**11**	29.5*
IIR 1D	8	18	8	18	--	--	**6**	18	**7**	18.5*	--	--
Equation Solver	13	26	11.5	25.5	--	--	**10**	**25**	11.5	26.5*	--	--
All-pole Lattice Filter	17	35	16	34.5	--	--	**16**	**33**	**14**	35.5*	--	--
Elliptic Filter	36	75	34	72	--	--	**27**	80*	**29**	80*	--	--

表格 7. 實驗結果 (增加 data ALU 和 accumulator 個數).

	1 ALU, 4 acc, 4 reg, 2 mem						2 ALU, 4 acc, 4 reg, 2 mem					
	RSVR		RSF		RST		RSVR		RSF		RST	
	len	#	*len*	#	*len*	#	*len*	#	*len*	#	*len*	#
Wave Digital Filter	6	14	5	13	5.5	13	6	14	**4.5**	13	5.5	13
Filter	6	11	5.5	10.5	4.5	10.5	6	11	5.5	10.5	4.5	10.5
IIR 2D	16	34	16	34	16	34	**10**	34	**10**	34	**9**	34
Forward-substitution	5	11	5.5	10.5	5	10.5	**4**	11	**4**	10.5	**4**	10.5
THCS	4	10	4	9.5	4	10	**3**	10	**3**	9.5	**3**	10
DFT	12	32	12	28.5	12	30	**10**	32	**9**	**28**	**9.5**	30
Floyd-Steinberg	18	38	17	37.5	17	37.5	**13**	38	**14**	39.5	**13.5**	39.5
Transmission Line	12	26	12	26	12	26	**8**	26	**8.5**	26	**8.5**	26
IIR 1D	8	17	8	17	--	--	**6**	17	**6**	17	--	--
Equation Solver	13	26	11.5	25	--	--	**10**	**25**	**10**	25.5	--	--
All-pole Lattice Filter	17	33	16	30.5	--	--	**16**	33	**13**	30.5	--	--
Elliptic Filter	35	67	30.5	63.5	--	--	**23**	70	**24**	68	--	--

表格 8. 實驗結果 (增加 memory bank 個數).

	1 ALU, 3 acc, 6 reg, 2 mem						1 ALU, 3 acc, 6 reg, 3 mem					
	RSVR		RSF		RST		RSVR		RSF		RST	
	len	#	*len*	#	*len*	#	*len*	#	*len*	#	*len*	#
Wave Digital Filter	6	14	5	13	5	13	**4**	14	**4**	**12.6**	5	13.3
Filter	6	11	5.5	10.5	4.5	10.5	6	11	**5.3**	**10.3**	**4**	**10.3**
IIR 2D	16	34	16	34	16	34	16	34	16	34	16	34
Forward-substitution	5	11	5	10.5	5	10.5	5	11	5	**10.3**	5	**10.3**
THCS	4	10	4	9.5	4	10	4	10	4	**9.3**	4	10
DFT	12	32	12	28	12	30	12	32	12	**26.6**	12	30
Floyd-Steinberg	18	38	17	38.5	17	38.5	18	38	17	**38.3**	17	**38.3**
Transmission Line	12	27	12	27	12	27	12	27	12	27	12	27
IIR 1D	8	17	8	17	--	--	8	17	8	**16.6**	--	--
Equation Solver	13	25	11.5	22.5	--	--	**12**	25	**11.3**	**21.6**	--	--
All-pole Lattice Filter	17	33	16	31.5	--	--	17	33	**15.6**	**30**	--	--
Elliptic Filter	35	68	30.5	65	--	--	35	68	35.3	**64**	--	--