國 立 中 山 大 學 電 機 工 程 學 系

碩士論文

指導教授：黃宗傳 博士

在記憶體處理器系統上改善工作負載平衡與程式最佳化

Improving Workload Balance and Code Optimization on Processor-in-Memory Systems

研究生：李藍基 撰

中華民國九十年五月

國立中山大學電機工程學系碩士班

李藍基君所撰之論文

在計憶體處理器系統上改善工作負載平衡與程式最佳化

係完成碩士學位資格之一部份，業經下列委員口試及審查通過，特此證明：

口試委員：

系主任：

Improving Workload Balance and Code Optimization on Processor-in-Memory Systems

By

Lan-Chi Lee

A Thesis Submitted to the Graduate Division in Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
Republic of China
May 29, 2001

Approved by :

Department Chairman :

致 謝

在我漫長的求學路程，首要感謝的是我父母全力的支持，他們重來未曾強迫我走我不想走的路，尤其我的母親，她總是默默的為我不斷的付出，卻從未要我的回報，這分至高的情感，帶者我度過重重的困難，讓我有了今天的這份成就，如果，我在這份論文中有得到任何的榮譽，我希望能夠將這些榮譽加於我的父母。

此外，對於那些愛護我的人，授業恩師 黃宗傳先生、朱守禮學長、同學與實驗室的各位伙伴，我也深深的感謝他們給予我的幫助；另外，我要感謝陪我走過漫長人生的女友-廖釗儷小姐，感謝她體諒了我這九年來的任性與無知，並總是給與我最適度的鼓勵，她的陪伴讓我的生命有了另一層意義。

最後，再一次的感謝幫助過我的各位，謝謝你們！

在記憶體處理器系統上改善工作負載平衡與程式最佳化

研究生:李藍基　　　指導教授:黃宗傳 博士

國立中山大學電機研究所

中文摘要

記憶體處理器（Processor-In-Memory）是在最近幾年中被提出，它主要的目的在於減少記憶體與處理器間效能的差距，在過去，我們為了能利用記憶體處理器潛在的優點，提出了一種以陳述(Statement)作為分析模型的平行系統-SAGE[1,2]。在這篇論文中，為了使記憶體處理器中的各個處理器能有最適量的工作，以輪次（Iteration）為目標的分析方法為另一個重要的研究概念，我們設計出幾種最佳化技巧去擴展原有系統的效能，這些技巧包含了智慧型記憶體操作辨識法（IMOP; Intelligent Memory Operation Recognition）、PIM式的分塊法（Tiling）與一個能取得精確工作分配(Workload Balance)的執行流程；最後，我們將提出我們的實驗結果，並且對這些結果做一個討論。

Improving Workload Balance and Code Optimization on Processor-in-Memory Systems

Student：Lan-Chi Lee   Advisor：Tsung-Chuan Huang

Department of Electrical Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, R.O.C.

Abstract

PIM (Processor-In-Memory) architectures have been proposed in recent years. One major objective of PIM is to reduce the performance gap between the CPU and memory. To exploit the potential benefits of PIM, we designed a statement base parallelizing system –SAGE in [1, 2]. In order to make all processors take the best-fit workload in PIM, iteration base analysis is another research issue in this paper. We extend this system to achieve better performance by devising several comprehensive optimizing techniques, which include IMOP (Intelligent Memory Operation) recognition, tiling for PIM, and a precise mechanism to get workload balance execution schedule. The experimental results are also presented and discussed.

目錄

中文摘要……………………………………………………………………I

英文摘要…………………………………………………………………...II

目錄…………………………………………………………….…………III

圖目錄…………………………………………………………...………V

表格與演算法目錄………………………………………………..……VII

第一章 介紹………………………………………………………...… 1

第二章 智慧型記憶體之架構……………………………………………6

第 2.1 節 智慧型記憶體(IRAM) ……………………………...…6

第 2.2 節 動態頁(Active Page) ……………………………………7

第 2.3 節 資料集中式架構(DIVA) ……………………………...9

第 2.4 節 FlexRAM 架構描述…………………………………………10

第 2.5 節 基本參數……..………………………………………11

第三章 平行的方法…………………………………………………13

第 3.1 節 加權分割圖的建立…………………………………..14

第 3.2 節 加權時間自我修補法…………………………………..16

第 3.3 節 PIM式的分塊法……………………………...…21

第 3.4 節 在記憶體處理器架構下的迴圈分割……………...…23

第 3.5 **節 智慧型記憶體操作辨識法**……………………………..…25

第 3.6 **節方法整合**………………..…………………………………..31

第四章 實驗結果……………………………………………………32

第五章 結論…………………………………………………………36

圖目錄

圖 1-1. 在 SAGE系統上的編譯步驟……………………………………...…3

圖 2-1. IRAM 的架構…………………………………………………...7

圖 2-2. Active Page 的架構....…………………………………………...8

圖 2-3. DIVA 的架構..…………………………………………………....9

圖 2-4. FlexRAM的組織圖..……………………………………………...11

圖 3-1. 由圖 3-9 程式導出之加權分割圖 ..……………………………15

圖 3-2. 加權值表的資料結構與相關操作函式..………………………...17

圖 3-3. 修補未知運算子之方法…………………………………….....20

圖 3-4. 輸入相依關係示意圖……………………………………………24

圖 3-5. IMOP 之程式範例……………………………………………...26

圖 3-6. IMOP 程式執行時間關係圖…………………………………....27

圖 3-7. 說明區域性與 IMOP 關係之簡單的程式範例…………………28

圖 3-8. 程式執行結果……………………………………………….....28

圖 3-9. 整合後之 SAGE組織圖……………….……………………….....31

圖 4-1. 數值取代之表示程式……………………………………………33

圖 4-2. **五個測試程式的執行時間柱狀圖**……………………………..35

表格與演算法目錄

表一. FlexRAM **架構的相關參數** 12

表二. **實驗結果**..33

演算法一. Self-Patching for Weight Evaluation 18

演算法二. PIM's tiling..22

演算法三. Loop Splitting..25

第 一 章
介　紹

將記憶體與處理器做結合的構想可以推至 1995　1997 年，當時所提出的架構稱為智慧型記憶體（IRAM：Intelligent RAM）或是記憶體處理器（PIM：Processor-In-Memory），這些架構中可以看出它們已結合了記憶體與簡單的處理器（類似 ALU），而我們的實驗平台-FlexRAM[12]則算是眾多 PIM 架構中的一個類別，在這些類別中另有一些有名的系統，如 IRAM, Active Page, DIVA[18,8,9]等等，在第二章的開頭將可以看到其概略架構。

記憶體處理器有一些優點值得我們去注意，像是可以降低處理器與記憶體間效能差距的影響、比較經濟、功率的消耗較傳統的系統低…等等，但是，它也有一些不好的缺點，如晶片溫度較高、晶片測試與製程不易等，這些困難的問題尚需要一些時間解決，不過，隨著科技的進步，這些問題將會被逐一的克服；接下來，我們將指出 PIM 與傳統多處理器的差異。

記憶體處理器是一種特殊的多處理器架構，它和以往的多處理器電腦（Multiprocessor Computer）最大的不同有下列幾點：

1. 傳統的多處理器電腦其處理器為相似（Homogeneous）的，而記憶體處理器則不一定為相似的。
2. 處理器需透過主機板（Main Board）上所提供的系統匯流排與記憶體做連結，而記憶體處理器則是將處理器與記憶體做在同一個晶片上。
3. 由於記憶體能夠處理一些簡單的運算，故能有效的降低處理器與記憶體之間傳遞資料的次數，相對的增加了頻寬。
4. 將電路微型化，使得整個處理系統的耗電量降低。

在眾多的架構當中，我們選擇了 UIUC 的 FlexRAM 做為整個系統研究的基礎，對於這種系統的描述我們會於第二章的第四節加以說明。

在記憶體處理器的機器上，我們需要使用與過去不同的編譯器，像這一方面有 VIRAM 這種被新提出的 PIM 編譯器，不過 VIRAM 是屬於專屬的編譯器（專門處理向量語言），這會使的此編譯器會產生機器相依性，而我們的想法

是希望能夠建立一個不單只適用於某專屬機器架構的編譯器，我們期望能透過一些高階的轉換技巧，去得到一個較好的可攜性（Portability）與效能（Performance）。

FlexRAM 為一種處理器階層式的架構，為了能夠有效的利用這種特殊的架構，需要有一個與以往不同的編譯機制，能夠將一個程式依據處理器能力的不同，切分出不同工作之區塊，為了達到這個目的，我們提出了 SAGE（Statement Analysis Group Evaluation）如圖 1-1：

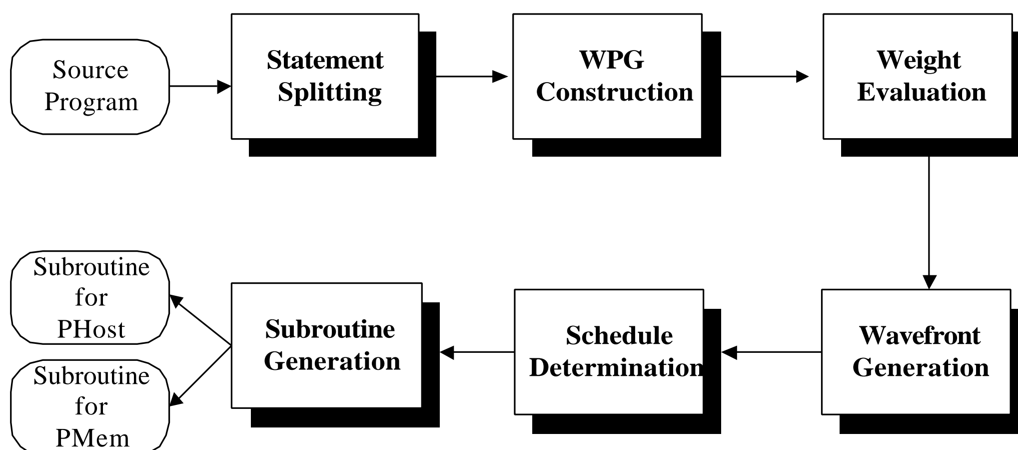


圖 1-1. 在 SAGE 系統上的編譯步驟

在 SAGE 的環境下，我們分析不同的陳述（Statements），看看彼此之間是否有相依關係（Dependence Relations），沒有相依關係的陳述可以被同時的執行，當然，我們也需要慎重的選擇可同時執行的陳述，以取得更好的效能；由圖 1-

1 中可以看出 SAGE 不僅止於工作的分配而已，同時它還結合了加權分割圖（Weight Partition Graph）建構、加權值的取得（Weight Evaluation）、波前的產生（Wavefront Generation）以及工作的排程等等，對於一個來源程式（Source Program）的最佳化轉換，SAGE 可以算是一個完整的系統了；但是 SAGE 本身還有一些地方可以再做進一步的加強，所以，我們又提出了一個修改過的 SAGE，在此我們先將這種 SAGE 稱做「現行的 SAGE」（Current SAGE）以作為區分，並且在這篇論文的第三章，可以看到其組織圖。

傳統的多處理器之工作平衡（Workload Balance）是建立於平均分配法則（Round-Robin）的技巧去達成的，例如平行語言中的「Do All」宣告；其實不論是哪一種平行化的處理[3、10]，都是針對各個處理器的能力皆相同的的情況（包含計算能力與記憶體的存取時間），去達到所謂的工作平衡分配；因此對於不同能力的處理器，為了充分發揮其個別的效能，我們須要針對其計算能力、記憶體的存取速度與架構差異作一個整體性的考量，並將程式依照其特性，分配適當的工作量給適當的處理器，使得處理器間的同步時間（Synchronization Time）降到最低，進而藉由工作平

衡的情況達到效能的提升，所以，我們需要在記憶體處理器上進行工作平衡的分析。我們的方法是結合迴圈分散法（Loop Distribution）與迴圈切割法（Loop Splitting），使得工作的切割同時兼顧了中等工作量（Medium-Grain）與少量工作量（Fine-Grain），這將使得我們所提出的方法對於工作量的分配具有更大的彈性，並且配合具有學習能力的預先的測試（Profiling）機制，考量記憶體處理器架構上特有的智慧型記憶體操作（IMOP; Intelligent Memory OPeration），為程式切割提供前置的分析。

在我們進入主題前，會先在第二章的部份對前人所做的重要研究作一番簡單的介紹。第三章，則會針對各個轉換機制如加權時間自我修補法（Self-Patch Weight Evaluation）、迴圈切割法、迴圈分散法、分塊法與智慧型記憶體操作等做一個詳盡的說明，在第四章，我們將對實驗的結果作討論。

第 二 章
智慧型記憶體之架構

第一章中我們有提到幾種新的 PIM 架構，它們設計的目的皆是為了降低去記憶體存取的次數或是平衡處理器與記憶體間效能的差異，綜觀各種架構的 PIM 系統，我們會發現處理器與記憶體被整合到同一個晶片上，在下面各節中，我們將看到不同的 PIM 系統，並對各個系統作一個簡單的介紹。

2.1 智慧型記憶體（IRAM）

IRAM 的系統是由 David Patterson 所題出的，它的中央處理器由 Alpha 21164 擔任，這顆處理器上有 8KB 的指令快取計憶體, 8KB 的資料快取計憶體, 96KB 的第二層快取與 4MB 的第三層快取，像這種由向量電腦構建出的系統通常有幾種好處，第一，因為一個指令就包含了許多的工作，所以指令的提取次數，自然較純量電腦（Scalar Computer）為少，第二，它的資料存取形式多為規則式的，並且，通常都是同時存取一整個區塊的資料，這使的資料的存取時間較不

會影響整個系統的效能，但不可誨言的，由於此處理器有三層且較大的快取，這也意味著當快取失敗時，其花費在重載資料的代價也就越高。

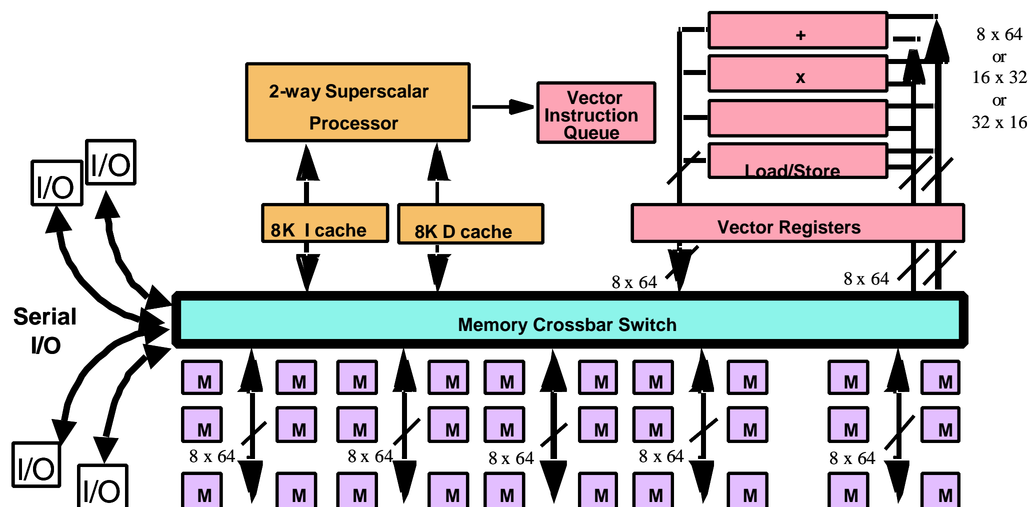


圖 2-1. IRAM 的架構

2.2 動態頁（Active Page）

Active Page 其硬體架構的重點為可重新組態邏輯（Reconfigurable Logic），配合由資料頁（Data Page）與集合關連函數（Set Associated Function）構建成 PIM 的系統，透過一些基礎的介面函式（如下所示），為程式選擇出適當的處理函數，並將其嵌入（embedded）系統當中。

✍ 標準記憶體函式：專門用來存取記憶體用的函式，如讀取與寫入。

✍ 特定功能之計算函式：用在處理某些特定用算用途之函式。

✍ 記憶體配置函式：負責處理記憶體分配的問題。

✍ 結合函式：選出適合原程式的特定函式並將其結合。

舉例來說，就好像一個由陣列作為主體的程式，我們可以將一些常被用來處理陣列的函式，如插入、刪除與尋找等函式整合（燒錄）到可程式邏輯中，如此將可以獲致可觀的效能。

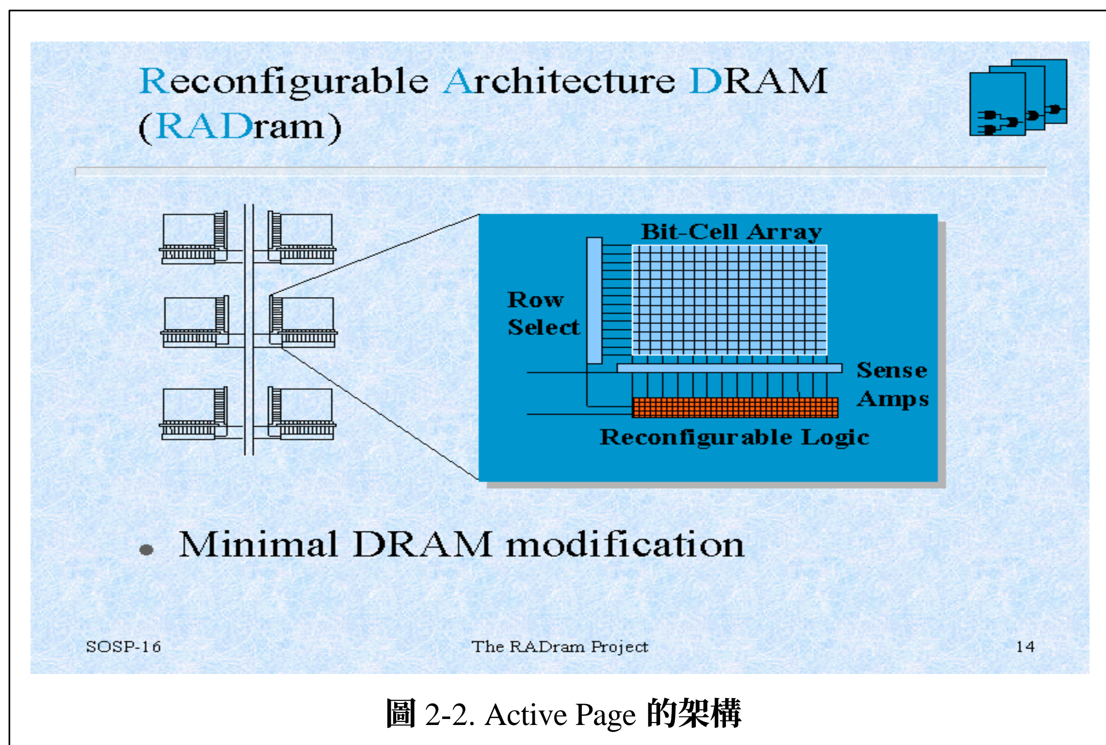


圖 2-2. Active Page 的架構

不過這個系統有一個缺點，就是每當處理器要執行一新的程式，新的程式其需用到的函式與上一個程式不同時，我們需要付出等待函式重新燒錄的時間，此外，如此頻繁的燒錄、抹除動作，有可能會降低整個處理器的使用壽命。

2.3 資料密集式架構（DIVA）

這個系統是由美國的南加大所提出，它有一個被稱為「寬字元邏輯」（Wide-Word Logic）的特殊架構，具有多個 128 512 位元的列暫存器（Row Register File），可以執行純單一指令多重資料（SIMD）或是利用罩蓋（Masking）的方式執行有條件的單一指令多重資料命令，PIM 晶片間的資料傳送，可以不用透過主處理器來維護。

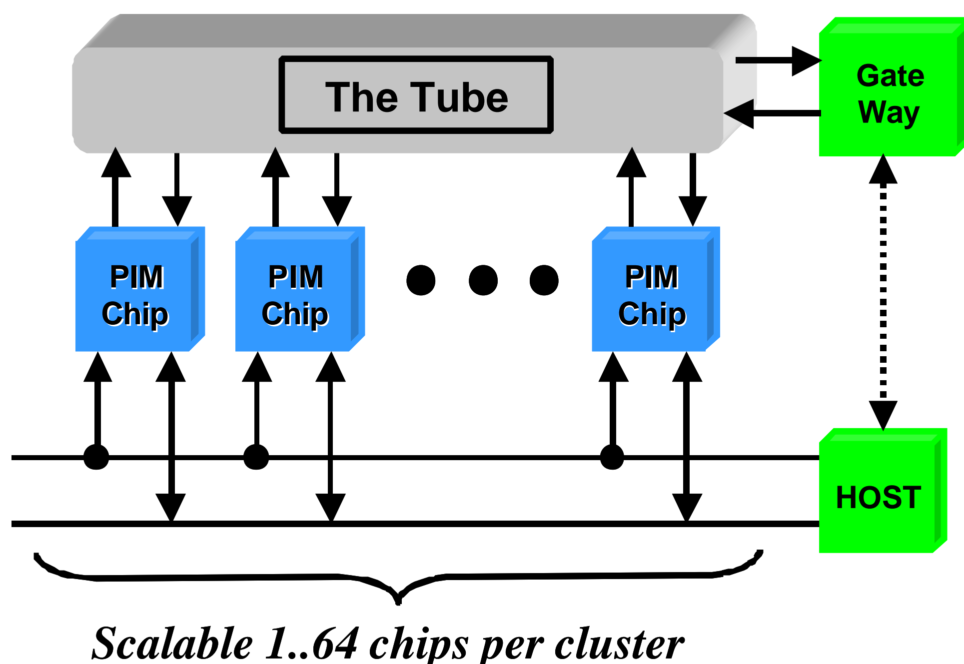


圖 2-3. DIVA 的架構

上圖為 DIVA 這個系統中的一個叢集之架構圖，若有需要可以透過閘道器(Gateway)再與其它相同架構的 DIVA 做串接，但由於這個系統的資料獲得不易，故一些相關的細節，我們無從得知。

2.4 FlexRAM 架構描述

FlexRAM 的架構如下圖[7]，它為三層的處理器架構，分別為 P.Host、 P.Mem、P.Array，每一個 FlexRAM 的晶片（chip）上有一個 P.Mem 處理器與 64MB 的記憶體空間；P.Host 可以喚起一至數個 P.Mem 去執行工作，而 P.Mem 與 P.Mem 之間的溝通是透過晶片間網路(Inter-chip Network)，每一個 P.Mem 又下轄 64 個 P.Array，而 P.Array 只能與相鄰的另外二個 P.Array, 這是為了節省複雜的溝通網路成本，所以 P.Array 被連成一個環狀的網路，若要存取遠端的 P.Array 則需要透過 P.Mem 來做溝通；這三種處理器分別有不同的處理效能，對於計算能力而言，P.Host > P.Mem > P.Array，但對於記憶體存取的速度則是 P.Host < P.Mem < P.Array，這是因為 P.Array 是最接近記憶體單元（cell）的處理器，其傳輸資料的延遲時間也就相對的大幅縮短。

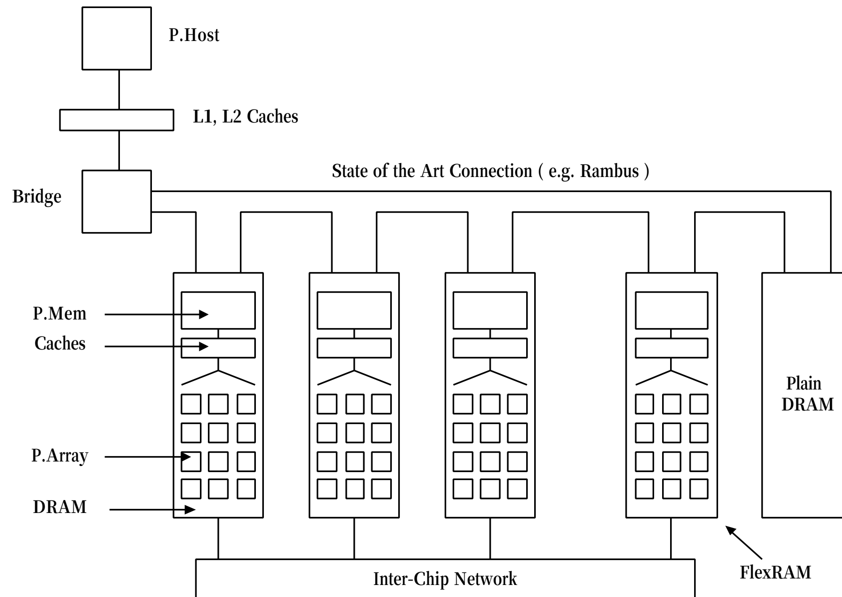


圖 2-4. FlexRAM 的組織圖

2.5 基本參數

在瞭解其初步的架構後，接下來就是要介紹這個架構的一些重要參數了，我們將 P.Host 的時脈設定為 800MHz，它是同一個時間可以發出 6 個指令（Issue）的超純量（Superscalar）機器，而且指令的執行方式允許不按照順序的執行（Out-of-Order），它有二個儲存/載入單元（Load/Store Units），執行一次儲存/載入各要花 8 個時脈週期（Cycle）；而 P.Mem 的時脈速度則定為 400MHz，為同一個時間可以發出 2 個指令的的超純量機器，可是指令的執行必須為按照順序的，在這個處理器上亦有二個儲存/載入單元，其執

行一次記憶體之儲存/載入時間花費較 P.Host 短；P.Host 具有 2 層的快取記憶體，而 P.Mem 則只有一層的快取，至於 P.Array 則無快取的存在。

另外，我們可以從表一中看出 P.Host 傳資料到記憶體的延遲時間較 P.Mem 傳資料到記憶體的時間長，這個原因已在先前說明過，其餘的參數可參考表一：

表一. FlexRAM 架構的相關參數

P.Host	P.Mem	Bus & Memory
Freq: 800 MHz	Freq: 400 MHz	Bus Freq: 100 MHz
Dyn. issue Width: 6	Static issue Width: 2	PHost Mem RT: 262. 5 ns
Integer unit num: 6	Integer unit num: 2	PMem Mem RT: 50. 5 ns
Floating unit num: 4	Floating unit num: 2	Bus Width: 16 B
FLC_Type: WT	FLC_Type: WT	Mem_Data_Xfer: 16
FLC_Size: 32 KB	L1 Size: 16 KB	Mem_Row_Width: 4K
FLC_Line: 64 B	L1 Line: 32 B	
Replace policy: LRU	Replace policy: LRU	
SLC_Type: WB	L2 Cache: N/A	
SLC_Size: 256 KB	Branch penalty: 2	
SLC_Line: 64 B	PMem_Mem_Delay: 17	
Replace policy: LRU		
Branch penalty: 4		
PHost_Mem_Delay: 88		

第 三 章
平行的方法

在過去多年的平行研究當中，有許多的方法可以用在平行處理上，比較常看到的方法有：分塊法、迴圈融合法（Loop Fusion）、迴圈切割法、迴圈分散法和展開與塞入法（Unroll-and-Jam）等等，當然，平行的方法不是只有這麼一些而已，還有許多不同的研究處理方式，但在 PIM 這種系統被提出前，處理平行資料的思考著眼點通常僅針對於對稱式處理器，雖然在過去也有過非對稱式的的系統出現過，如叢集式電腦（Cluster Computer），而資料的分配（散）與排程對於這種系統亦是一種重要的課題，但這種系統卻存在一個重要的瓶頸-資料的傳輸延遲時間長，這而對於 PIM 的系統來說，幾乎可以忽略這個問題。

所以，本章的各小節所提出的方法多為處理資料分配方面的問題，至於資料傳輸延遲的問題，我們則使用一些可增加資料區域性（Data Locality）的方法，來降低資料傳遞的次數。

3.1 加權分割圖的建立

在介紹我們的演算法之前，有一個重要的資料結構必須在這為各位做一個簡要的說明，那就是加權分割圖的建立。在建立這個分割圖前，我們需要針對節點分割作定義，在此引用了 D.J. Kuck（Node Partition）的說法[10]：

Definition 1 (Node Partition ?)

On the dependence graph G, for a given loop L, we define a node partition ? of $\{S_1, S_2, \ldots\ldots S_d\}$ in such a way that S_k and S_l, $k ? l$, are in the same subset if and only if $S_k ? S_l$ and $S_l ? S_k$, where ? is an indirect data dependent relation. On the partition ? $=\{ ?_1, ?_2 \}$, we define partial ordering relations $?$, $\overline{?}$, and $?^o$ as follows.

For i ? j:

1) $?_i ? ?_j$ iff there exist $S_k ? ?_i$ and $S_l ? ?_j$ such that $S_k ? S_l$, where $?$ is the true dependence relation .
2) $?_i \overline{?} ?_j$ iff there exist $S_k ? ?_i$ and $S_l ? ?_j$ such that $S_k \overline{?} S_l$, where $\overline{?}$ is the anti dependence relation.
3) $?_i ?^o ?_j$ iff there exist $S_k ? ?_i$ and $S_l ? ?_j$ such that $S_k ?^o S_l$, where $?^o$ is the output dependence relation.

在瞭解節點分割的定義後，我們需要建立出各個陳述間的資料相依圖（Data Dependence Graph），並且依據定義一建構出符合定義的分割集。

Definition 2 (Weighted Partition Dependence Graph)

For a given node partition ? as in Definition 1, we define a weighted partition dependence graph *WPG(P,E).* For each $?_i$? ? , there is a node $b_i < I_i, S_i, W_i, O_i > ? P$, where I_i is the loop index, S_i is the body

statement , W_i is the weight of node i in the form of $W_i\,(PH,PM)$ where PH and PM are the weights to P.Host and P.Mem respectively, and O_i is the execution order of this node. There is an edge e_{ij} ? E from b_i to b_j, if b_i and b_j have dependence relations ? , $\overline{?}$, or $?^{o}$ as in Definition 1, and which are respectively denoted by ?? ? , ? $?^{anti}$? , and $??^{O}$? .

上面所提到的加權分割圖，我們將會利用它作為資料分配與資料排程的主要核心，其長相如下圖：

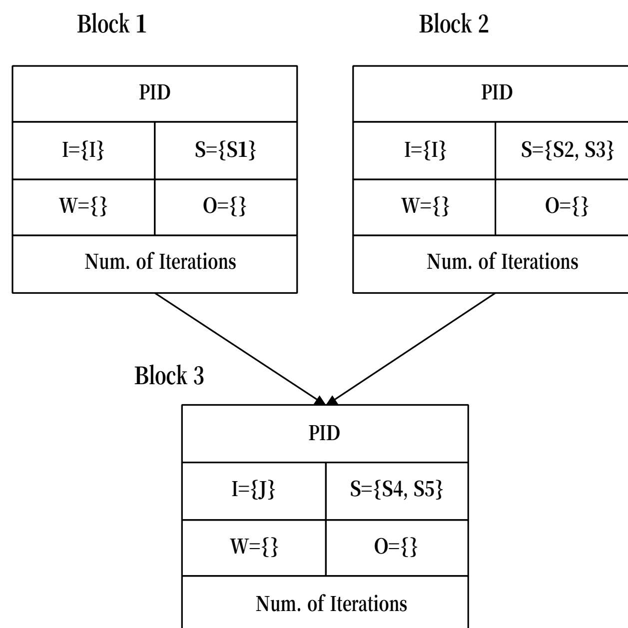


圖 3-1. **由圖** 3-9 **程式導出之加權分割圖**

另外，我們藉由定義二為它加上基本的屬性，這些屬性將分別記錄各個分割集的總加權值與它的執行順序（Wavefront Order**），這些屬性相關的資訊在定義二已有說明。**

3.2 加權時間自我修補法

為了達到一個好的工作分配，我們需要有一個評估機制作為工作切分的依據，在過去，曾有研究者利用預測法去執行程式中幾個重要片段，並利用所得的資訊作為分析的依據[6]，這種機制的優點是可以評估程式的一些動態行為與精確，但是，它沒有保存過去辛苦所得到的資訊，以致於每次執行同一個程式時，還是需要重做相同的事，如此大大的浪費了編譯時間；而在 SAGE 系統中則是使用一種靜態的查表法，如此將會導致程式無法動態的依據環境不同做一個適當的變化，為此，我們在 Current SAGE 中修正了這個問題，並提出了加權時間自我修補法，此方法乃是結合了預測法與靜態查表法，利用預測法去建立初步加權表（Weight Table）中的各項參數值，同時將這些有用的資訊加以保留，使用加權值的好處在於往後再度進行工作分配時，不需要再次使用預測法，使的編譯的時間增加。

在討論加權時間自我修補法前，有一點值得我們注意的，就是需要先知道同一個運算子（Operator）或是記憶體參考（Memory Reference）有可能會因為處理器的種類或能力不同而有不同的加權值，其後，我們需要瞭解加權表中

應該要包含那幾種資訊，如第一，它必須要記錄不同的運算子的加權值，第二，它必須具有記錄記憶體存取時間的能力，第三，它要能記錄程式動態產生的行為（如 cache miss、branch penalty etc.**）；依照上述我們描述出加權表的長相，下圖除了有加權表的資料結構，還包含了其存取加權表應有的基本操作：**

```
Data Structure of Weight Table
Weight_Table
{
    Table PH_Optr ( operator, weight );
    Table PM_Optr ( operator, weight );
    Table PH_Mem ( Mem_ref, weight );
    Table PM_Mem ( Mem_ref, weight );
     /* operator is the machine code of the operation;
        Mem_ref denotes the layer of referenced memory hierarchy. */
}

Basic Subroutine Set for Manipulating Weight Table


  Add_weight_value ( pid, Weight_Table, Op_Mem, weight_value );
     /* Adding weights into the Weight_Table. */
  Find_weight_value ( pid, Weight_Table, Op_Mem, weight_value );
     /* Finding the weight value of the operator in the Weight_Table. */
  Get_weight_value ( pid, Weight_Table, Op_Mem, operation)
     /* Finding the weight value of the operator/mem_ref in the
     Weight_Table. */
     /*Op_Mem : the value '0" and "1" represent for access operator or
       memory reference table respectively. */
```

圖 3-2. 加權值表的資料結構與相關操作函式

有了這些基本的操作之後，接下來我們就將利用它們製作一個智慧性的預測法，下圖為其演算法：

Algorithm 1. (Self-Patching for Weight Evaluation)

[Input]
Weight_Table and WPG(P, E)

[Algorithm]

For each b_i ? WPG
For each S_i ? b_i /* S_i is a statement in loop i. */
For each Op_i ? S_i /* Op_i is an operator in S_i . */
OP_find_out = Find_weight_value (pid, Weight_Table, 0, Op_i);
If (! OP_find_out) /* Can’t find the operator’s weight. */
Call Patch (Weight_Table, S_i, Op_i);
End if
PH_OSum = PH_Osum + Get_weight_value (P.Host, Weight_Table, 0, Op_i);
PM_OSum = PM_Osum + Get_weight_value (P.Mem, Weight_Table, 0, Op_i);
End for
For each Ref_i ? S_i /* Ref_i is the memory reference type. */
PH_MSum = PH_MSum + Get_weight_value (P.Host, Weight_Table, 1, Ref_i);
PM_MSum = PM_MSum + Get_weight_value (P.Mem, Weight_Table, 1, Ref_i);
End for
End for
$W_i(PH,PM)$ = { PH_OSum + PH_MSum + dynamic behavior cost, PM_OSum + PM_MSum + dynamic behavior cost };
/* Dynamic behavior cost is the cost of cache miss, condition or branch penalty. */
End for
End

為了以後排程的需要，每一個區塊必須有一個專屬的識別

編號（ID），這個編號是為了讓我們知道這個區塊應該要交給 P.Host 或是 P.Mem，不過這個部份的資訊將會是在迴圈分割後才能有一個確定的答案。

在演算法一中，我們有呼叫一個副程式進行修補（Patch）的工作，它的目的是用來將尚未被記之運算子加入加權表中，其作法如下：

```
Subroutine    (Patch)
[ Input ]
  Weight_Table, Statement, and Operator
[ Intermediate ]
  existed_op, existed_op_weight; /* A known operator and its
  weight */
  Exec_in_ph(S_i); /* Execute statement S_i in P.Host. */
  Exec_in_pm(S_i); /* Execute statement S_i in P.Mem. */
  Replace_optr(S_i, Op_i, existed_op); /* Replace Op_i by existed_op.
  */
[Subroutine]
/* Execute S_i in P.Host. */
  original = Exec_in_ph ( S_i );
  temp = Exec_in_ph ( Replace_optr( S_i, existed_op ) );
  ph_opw = original – temp + existed_op_weight;
  call Add_weight_value ( "P.Host", Weight_Table, 0, ph_opw )

/* Execute S_i in P.Mem. */
  original = Exec_in_pm ( S_i );
  temp = Exec_in_pm ( Replace_optr( S_i, existed_op) );
  pm_opw = original – temp + existed_op_weight;
  call Add_weight_value ( "P.Mem", Weight_Table, 0, pm_opw );
End
```

雖然，上面的演算法已陳述了加權值的測量法，但為避免

過於抽像，特別舉一個例子如下圖，作更進一步的說明：

假設”+”為一已知加權值的運算子，且其值為 2

若現在有一陳述如下：

A=B+C%D ✍ “%”為未知的運算子

執行過後，得到的執行時間為 6

此時，將原陳述的%運算子以+代換

A=B+C+D

並從新執行一次，可得到執行時間為 4

由此，我們就利用下列算式得知：

“%”的加權值 = 6-4+2(”+”的加權值)

∴“%”的加權值 = 4

圖 3-3. 修補未知運算子之方法

我們利用修補的機制，使得預測法具有修正的能力，藉由不斷的訓練，能夠降低執行預測法所花的編譯時間，理想的狀況是可以直接使用先前所建立的加權表，並依據此表給與每個處理器一個適合它們能力的工作，使其達到更高的工作效率。

3.3 PIM 式的分塊法

在 FlexRAM 這種異質型的機器上，我們可以利用各種不同的迴圈轉換（Loop Transformation）技巧去改善程式的效能，像是之前提過可增加區域性（Locality）轉換技巧的分塊法、迴圈融合法、展開與塞入法…等等[3、4]，但是在這篇論文裏，我們將只注重於增加資料區域性的方法，使得參考快取記憶體失敗（Cache Miss）的情況降到最低。

P.Host 這顆處理器擁有 16KB 的 L1 cache 與 256KB 的 L2 快取記憶體（參閱表一），而在 P.Mem 這顆處理器上則只有 32K 的 L1 快取記憶體，而沒有 L2 快取記憶體，對於這種特殊的記憶體架構，我們可以針對不同的記億體階層，將區塊執行不同大小的分塊法，以增加資料的區域性作為一個關鍵性的考量。

為了使分塊法的行為更有效率，我們參考了 Marta Jiménez 所寫的同時在多種記憶體架構下的分塊法（SMT; Simultaneous Multi-level Tiling）[11]，這種分塊演算法的好處除了能夠得到一個精確的迴圈邊界外，還能夠同時對多層的迴圈依其適合的記憶體架構做相應的切塊，除了一般

的分塊法外，我們亦引進了展開與塞入法這種最佳化技巧，它可以配合數值取代法（Scalar Replacement），有效的降低程式對記憶體的存取次數，下面為 PIM 式分塊的演算法。

```
Algorithm 2. (Tiling for PIM)

[ Input ]
WPG( P, E ) and pid    /* it could be P.Host or P.Mem*/
[ Output ]
WPG( P, E ); WPG( P, E ) after be applied SMT and unroll_and_jam.
[ Intermediate ]
HL1_cache; /* P.Host Level one cache. */
HL2_cache; /* P.Host Level two cache. */
ML1_cache; /* P.Mem level one cache. */
unroll_and_jam ( block, register file );
{
For each b_i  ?  WPG
  If ( pid = P.Host ) then
        If ( the outmost thorugh the innermost loop are fully
        permutable )
           b_i = SMT(b_i, HL1_cache, HL2_cache );
           Unroll_and_jam ( I_i, PH_reg );
        End if
  Else
        If ( the outmost thorugh the innermost loop are fully
        permutable )
           b_i = SMT(b_i, ML1_cache );
           Unroll_and_jam (I_i, PM_reg );
        End if
  End if
End for
End
```

利用此演算法可以有效使的快取記憶體的失敗降低，由於

快取記憶體的存取時間通常遠小於記憶體存取的時間，降低了快取記憶體失敗等於減少了去記憶體存取資料的機會，一方面增加了暫存器與快取的使用率，另一方面也間接的增加了系統匯流排的頻寬。

由圖 3-11 可看出我們將分塊法放在迴圈分割法之後，這是因為如果太早引進分塊法這種轉換技巧，那些經過慎重處理的迴圈邊界（Loop Bound），可能會因隨後的迴圈切割步驟被破壞掉，使得迴圈邊界無法精確的對應到記憶體架構上。

3.4 在記憶體處理器架構下的迴圈分割(Loop Splitting)

如同前言所提到的，要將一個工作切割成適合 PIM 這種處理器能力不同的架構下，必須審慎的考慮其能力不同之處；對於大量存取記憶體的工作，可以透過一開始 IMOP 的辨認，將具有 IMOP 性質的工作標記起來，以做為未來工作排程時的依據；標記起來的意義，不是指它只能被 P.Mem 所執行，而是要讓排程控制器（Scheduler）知道這個工作由 P.Mem 來執行會比 P.Host 更有效力，因此 P.Mem 會有較高的優先權去執行這個工作；在這一節，我們要介紹與

之前 SAGE 不同的一個工作切割方法，也就是迴圈切割法。

SAGE 的環境下，其工作切分的著眼點在於使用特殊的排程機制去達到良好的工作效能，但是，在 SAGE 的處理模式當中，有可能會導致原程式的區域性被破壞，這是因為一般的相依性分析，通常不會去考慮輸入相依性（Input Dependence)，如圖：

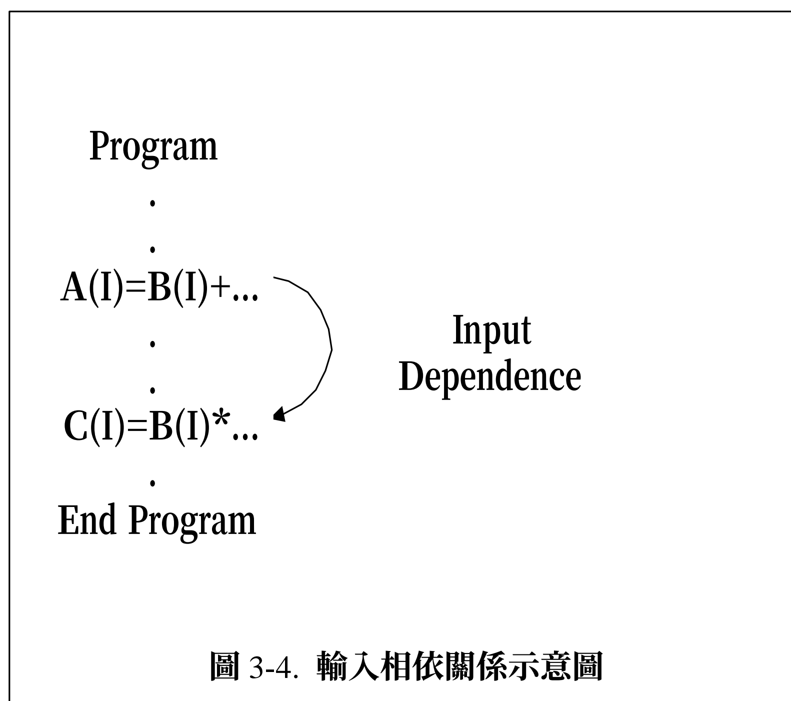


圖 3-4. 輸入相依關係示意圖

而這種相依關係若能有效的利用，則有可能得到空間區域性（Spatial Locality）的好處，但是，SAGE 卻有可能因為不考慮這種相依關係而破壞了原來存在的區域性，使的資料的使用率下降。此外，對於某些可以平行處理的關鍵區

塊，SAGE 無法利用排程法，去同時使用 P.Host 與 P.Mem，所以，在這裡使用迴圈切割法就會有至少二個好處，第一，能充分考慮到整個迴圈中的區域性，進而使用資料區域性所帶來的好處，第二，切割的單位較小，能取得較好的工作平衡。

我們利用 3.3 節的方法求出加權值的比值，並依此將工作分成 2 塊（或是 2 塊以上），分別交由 P.Host 與 P.Mem 來執行，下面為其演算法：

Algorithm 3. (Loop Splitting)
[Input]
WPG(P, E)
[Output]
WPG(P, E)
[Algorithm]
{
Step 1. Identify a block from the WPG to be split.
Step 2. Compute the workload ratio by the weights of P.Host over P.Mem. (The weights can be obtained from the WPG.
Step 3. Split the iteration space of the block by this ratio into two blocks, and modify the WPG according to their dependence relations.
Step 4. If there are blocks to be split, go to Step 1.

}

3.5 智慧型記憶體操作辨識法 (IMOP)

IMOP 具有大量資料運算的特性，故又有一些學者將

IMOP 稱為資料密集式運算（Data Intensive Operation）；在一般的多處理器系統中，各個處理器之間往往不會有能力上的差異，所以並不需要去思考有哪些情況是屬於計算密集式（Computation Intensive）或是資料密集式，但在異質型的多處理器上，我們就有必要去考慮哪些處理器比較適合處理哪些資料，如果我們能夠正確的分配工作，我們將可以充分發揮其架構的優勢。

那到底 IMOP 能給我們至少二個好處，我們將以下面的例子作一個解釋：

```
Program
.
Do I=1 to N
   .
   A(I)=B(I)+C(I)+…
   .
End Do
.
End Program
```

圖 3-5. IMOP 之程式範例

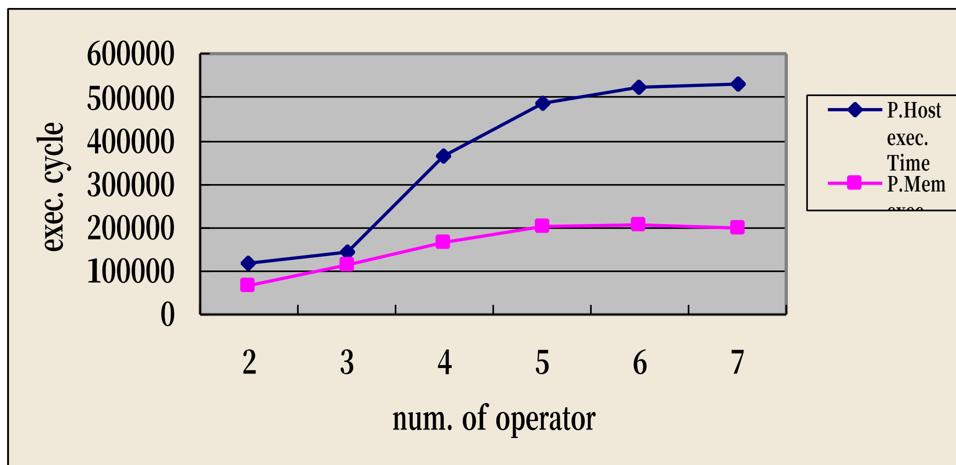


圖 3-6. IMOP 程式執行時間關係圖

圖 3-5 為一個簡單的程式，在這個程式當中有一個迴圈結構，圖 3-6 為其執行結果，它包含有一個記憶體參考的陳述，我們可以清楚的看到，當這個陳述所包含的陣列（記憶體參考）越多時，P.Host 與 P.Mem 的執行時間比的差異就越大，這告訴我們一件重要的事，那就是當一個程式所包含的資料量越大時，將資料交由 P.Mem 執行所獲得的好處也就會越大。

在 Steve Carr 所提的論文中[5]，曾提到可以利用 β_M 與 β_L 二個參數去識別迴圈的行為到底是屬於計算密集式或是資料密集式，依據 Steve Carr.的公式，若 $\beta_L > \beta_M$ 則此迴圈為資料密集式的迴圈。換句話說，若某一個巢狀迴圈（Loop

Nest）其記憶體參考數目相同，則此二個迴圈將為同一種形式。下面的例子將說明上述的說法是有問題的。

```
Program 1                  Program 2

Do I =1 to N               Do I=1 to N
     Statements                 Do J=1 to N
End Do                               Statements
                                End Do
                           End Do
```

圖 3-7. 說明區域性與 IMOP 關係之簡單的程式範例

L1=1024, L2=8192					
level	data size	PH exec time	read times	PM exec time	read times
1	1024	12923	1.05K	12588	1.484K
2	32*32	10783	1.05K	12531	1.484K

圖 3-8. 程式執行結果

由例子中發現同樣的記憶體參考個數的資料集，卻出現不同的執行時間情況，其實，這個原因很簡單，這是因為資料的區域性不同，導致資料集出現不同的形式，也就是說 ，一個 IMOP 的辨識，不單單取決於記憶體參考的個數，最重要的是考慮記憶體的大小，所以在下面我們將對於 IMOP 有一個初步的定義：

Definition 3. (**IMOP; Intelligent Memory Operation**)

For a given block b_i ? *WPG(P,E)*. If b_i conforms to following equation, it can be classified into IMOP.

$$PH1_AC+PH1_MT * (PH2_AC + PH2_MT * PH2_MC) + PH2_MT * (PH_MEMAC + PH_MEMMT * PH_MEMMC) + OPW(b_i) > PM1_AC + PM1_MT *PM1_MC + PM1_MT * (PM_MEMAC + PM_MEMMT * PM_MEMMC) + OPW(b_i)$$

where

PX1_AC is P.Host/P.Mem L1 cache access cost of b_i.
PH2_AC is P.Host L2 cache access cost of b_i.
PM1_MC is P.Mem L1 cache miss cost of b_i.
PX1_MT is P.Host/P.Mem L1cache miss times of b_i.
PH2_MC is P.Host L2 cache miss cost of b_i.
PH2_MT is P.Host L2 cache miss times of b_i.
OPW(b_i *)* is operation weight of block b_i.
PX_MEMAC and *PX_MEMMT* is P.Host/P.MEM memory access cost and miss times.

當程式發生符合上列數學式之情況時，我們說此程式具有 IMOP 的行為。

我們可以參考圖 3-6 來說明定義三，我們發現當 P.Host 的資料大於其快取記憶體時，P.Mem 的執行速度就有可能會比 P.Host 要來的快（這是因為參考記憶體的次數變多)，當然，這並不是絕對的，我們還要進一步的考慮其陳述的行為為何，因為在實際的情況中，P.Host 的快取記憶體的大小有可能會大於 P.Mem，那麼這會使的 P.Mem 的存取記

憶體比 P.Host 快的好處略收影響。

透過 IMOP 的辨認，我們可以得到至少二種的好處，第一，可以有效的利用各種處理器潛在的效能，第二，可以省掉其他企圖達到最佳工作分配所花的轉換時間。對於第二個好處，在此稍做解釋，當某一個區塊（Block）被辨識出具有 IMOP 的行為時，我們將不會對這個區塊再做迴圈切割，因為這有可能會使得原來的 IMOP 行為被破壞掉，這意味著我們可以省去執行工作切割的步驟。

3.6 方法整合

將上述的方法與先前所提的 SAGE 系統結合起來，可以有一個不錯的效能提升，而且同時兼顧了 coarse grain 與 fine grain 的平行，下圖為其整合後之組織圖：

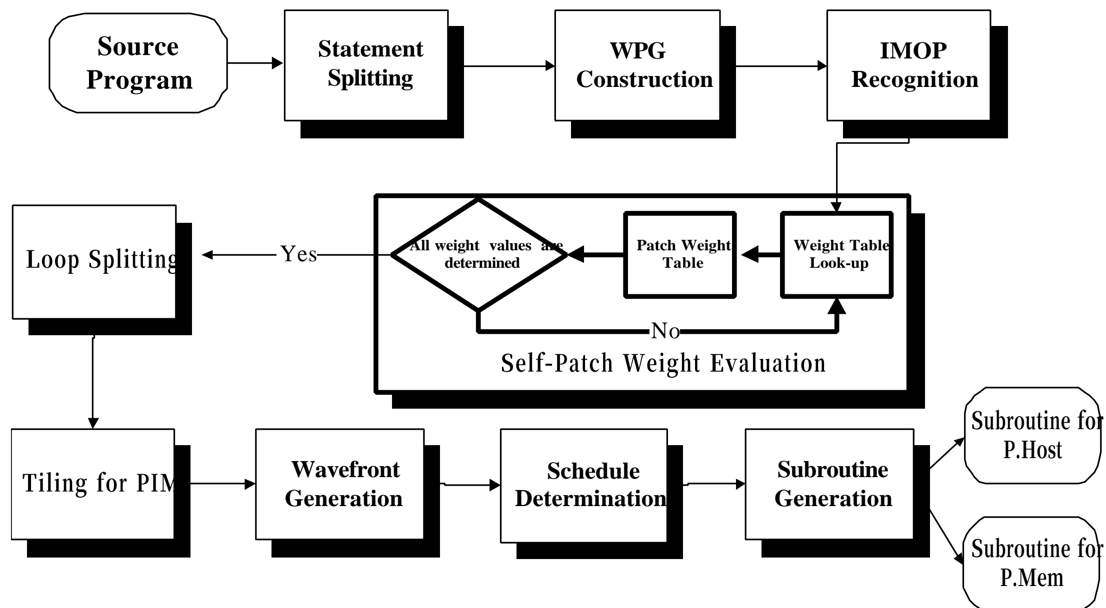


圖 3-9. 整合後之 SAGE 組織圖

在這個圖形上，我們將上述的各項轉換技巧加以修整與結合，透過這樣的修正可以有效的改善過去無法處理的某些情形，讓記體處理器的系統能有一個好的工作分配，同時，有效的發揮出其特有之長處。關於這一點我們可以由第四章的實驗結果看出這一點。

第 四 章
實驗結果

本實驗我們以 UIUC 的 FlexRAM 模擬器為實驗平台，這個模擬器會將執行的時間以週期為單位表現出來，此外，我們利用圖 3-11，將原始程式轉換成我們想要的樣子。

為了讓實驗單純，我們只使用了一個 P.Host 與一個 P.Mem，並用此驗證上述的方法，我們利用了 BLAS3 的 strmm、SPEC 95 的 swim 與 tomcatv、NAS 的 ep 與 FFT 數個不同的測試程式（Benchmarks）在 SGI Origin200 的機器上來做驗證，同時使用-O2 的參數（F77 Compiler）對程式做初步的最佳化，我們定義「standard」為 P.Host 單獨執行全部工作所花的時間，而「original」為 P.Mem 單獨執行全部工作所花的時間，「SAGE Mode」為二個工作重疊執行，工作的分配方法為則採用 SAGE 的方式，「Current SAGE Mode」亦為重疊二個工作執行，但工作的分配方法則改用我們此篇論文所提的方法，由表二的實驗數據中，我們可知道改良後的 SAGE 其效能確實有提升。

表二. 實驗結果

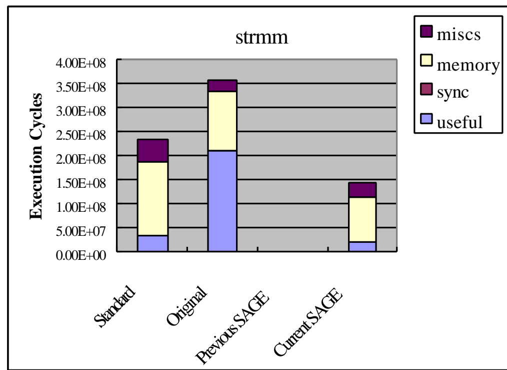
benchmarks	standard	original	SAGE Mode	Current SAGE Mode	Speedup	
					SAGE Mode	Current SAGE Mode
strmm	234331341	337237336	n/a	140942002	n/a	1.66
swim	188295086	258533896	142900464	116670133	1.32	1.61
tomcatv	380279967	455768117	391225016	297241276	0.97	1.27
ep	103006081	194402248	n/a	84813319	n/a	1.21
fft	4720700	16158182	5352920	2278611	0.88	2.07

由表二中可以看到 strmm 與 ep 無法由原先的「SAGE」去最佳化，這是因為這些程式其迴圈內部的陳述多為變數形式（如圖 4-1），如果 SAGE 要強行將其切分的話，需要將這些變數作數值展開（Scalar Expansion），這樣一來會很浪費記憶體的空間，進而增加了程式的執行時間。此外，我們可以看到 tomcatv 與 ep 這二個測試程式的速度提昇有限，這是因為它們程式本身的平行度就不高，在這樣的環境限制下，自然使得程式的效能也會受到一部份的影響。

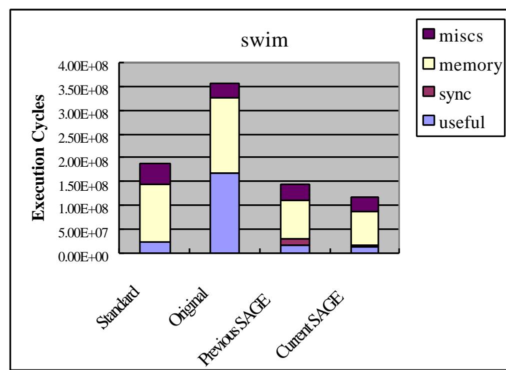
```
Loop1
    :
  Loop2
    :
    a = …
    b = …
    :
  End Loop1
End Loop2
```

圖 4-1. 數值取代之表示程式

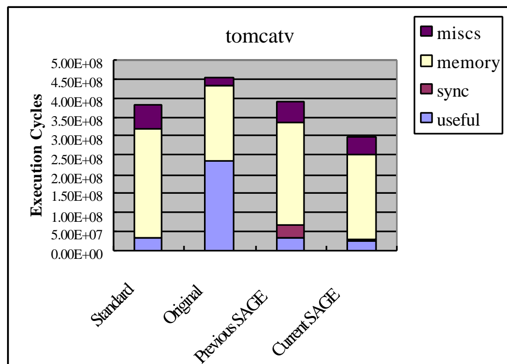
圖 4-2 反應出在 PIM 的環境下，若某個處理器它的工作量不適合其能力，那麼就會有其它的處理器需要等候其執行結果，也就是說花費在同步的時間會變的較為壅長；另外我們也可以看到在 Current SAGE 修正過後的程式，其花費在記憶體存取的時間也減少了，這種情況說明了二件事，第一，藉由新的 SAGE 架構，使得 P.Mem 的執行效率變的更好，第二，就是在 PIM 的系統下使用增加區域性的技巧，確實有其存在的意義。



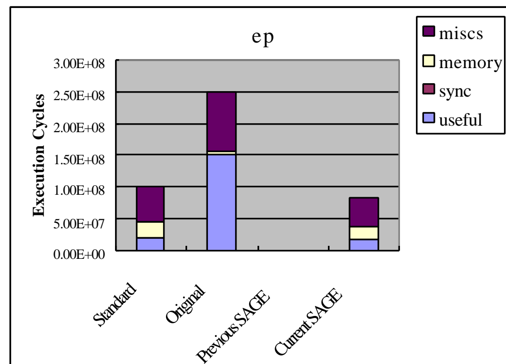
(a)



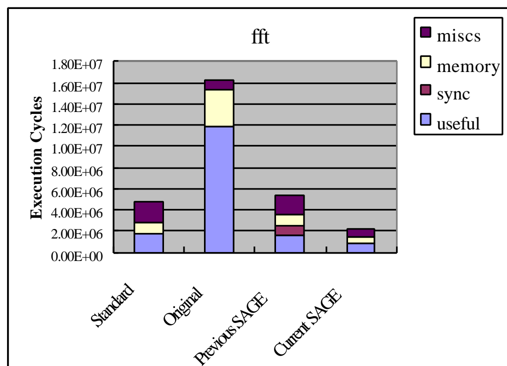
(b)



(c)



(d)



(e)

圖 4-2. 五個測試程式的執行時間柱狀圖，(a), (b), (c), (d), (e)**分別代表** strmm,swim,tomcatv,ep,fft**，圖中的**"useful"**為執行指令所花的時間，**"sync"**為等待其它處理器的時間，**"memory"**為花費在記憶體存取的時間，**"miscs"**為其它的執行錯亂。**

第五章
結　論

記體處理器這種特殊的架構提出已有一段時間了，其主要的目的在於有效的降低記憶體存取能力與處理器計算能力的差距，在過去的研究中，主要是偏重於提出新架構的智慧型記憶體，這些是針對其不同的硬體架構企圖作一個最佳的選擇；若從編譯器的角度來看，我們將需要一些轉換技巧去對程式做最佳化的編譯，但在過去幾年，一些高階的轉換技巧較少被提及，所以在這篇論文中，我們應用了幾種適當的轉換技巧，包括有分塊法、迴圈融合法、迴圈切割法和迴圈分散法，以及由我們所提出的智慧型記憶體操作與加權時間自我修補法。

由第四章的數據，確知了利用上述的轉換技巧，可以達到一個不錯的速度提昇；這一篇論文的重點是放在如何達到工作的平衡，所以對於排程的機制僅作一個簡單的說明，將來多顆處理器的工作排程會是我們研究的一個重點之一，透過一個有效的排程讓 PIM 能發揮出更好的效能。另一個值得我們研究的方向是在整個 PIM 的環境中再加入

P.Array 的變化，這樣將會使的整個系統更形完整，同時可以讓系統的效能有更大幅度的提升，不過這也會使得系統的複雜性增加，所以我們將利用這一篇論文作為一個前置的基礎分析，好為了未來的研究鋪路，藉由這一連串的實驗中證明了精確的工作分配確實能使 PIM 系統的效能有所提昇。

參考資料

1. Huang, T. C., and Chu, S. L.: SAGE: A New Analysis and Optimization System for FlexRAM Architecture. In proceedings of 2nd Workshop on Intelligent Memory Systems, Cambridge, MA, Nov. 12, (2000).
2. Huang, T. C., and Chu, S. L.: A New Analysis Approach for Intelligent Memory Systems. Will appear in proceedings of CATA 2001.
3. Wolf, M. E., Maydan, D. E., and Chen, D. K.: Combining Loop Transformations Considering Caches and Scheduling. Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture, (1996), pp. 274 –286.
4. Manjikian, N., and Abdelrahman, T. S.: Fusion of Loops for Parallelism and Locality., IEEE Transactions on Parallel and Distributed Systems, Vol. 8 Issue 2 , Feb. (1997), pp. 193 –209.
5. Carr, S.: Combining Optimization for Cache and Instruction-Level Parallelism. Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques, (1996), pp. 238 –247.
6. Wang, K. Y.: Precise compile-time performance prediction for superscalar-based computers. Proceedings of the ACM SIGPLAN '94 conference on Programming Language Design and Implementation,

(1994), pp. 73 – 84.

7. Kang, Y., Huang, W., Yoo, S., Keen, D., Ge, Z., Lam, V., Pattnaik, P., and Torrellas, J.: FlexRAM: Toward an Advanced Intelligent Memory System. International Conference on Computer Design (ICCD), Austin, Texas, Oct. (1999).
8. Oskin, M., Chong, F. T., and Sherwood, T.: Active Page: A Computation Model for Intelligent Memory. Computer Architecture. In Proceedings of the 25th Annual International Symposium on Computer Architecture, (1998), pp. 192 –203.
9. Granacki, J. *et al.* Data Intensive Architecture: DIVA. http://www.isi.edu/asd/diva/, (1998).
10. Kuck, D. J.: A survey of parallel machine organization and programming. ACM Comput. Surv. 9, 1, Mar. (1977), pp. 29-59.
11. Jiménez, M.: Multilevel Tiling for Non-Rectangular Iteration Spaces. Ph.D. Thesis, Departamento de Arquittectura de Computadores, Universitat Politécniac de Catalunya, May (1999).
12. Yoo, S. M., Renau, J., Huang, M., and Torrellas, J.: FlexRAM Architecture Design Parameters. Technical Report 1584, Oct. (2000).
13. Veenstra, J., and Fowler, R.: MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In MAS-COTS' 94, Jan. (1994), pp. 201-207.
14. Judd, D., and Yelick, K.: Exploiting On-Chip Memory Bandwidth in

the VIRAM Compiler. In proceedings of 2nd Workshop on Intelligent Memory Systems, Cambridge, MA, Nov. 12, (2000).

15. Moritz, C. A., Frank, M., and Amarasinghe, S.: FlexCache: A Framework for Flexible Compiler Generated Data Caching. In proceedings of 2nd Workshop on Intelligent Memory Systems, Cambridge, MA, Nov. 12, (2000).

16. Veidenbaum, A. V., Tang, W., Gupta, R., Nicolau, A., and Ji, X.: Adapting cache line size to application behavior. In Proceedings ICS'99, Jun. (1999).

17. Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P.: Numerical Recipes in Fortran 77. Cambridge University Press, (1992).

18. Patterson, D., Anderson, T., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C., Tomas, R., and Yelick, K.: A Case for Intelligent DRAM. IEEE Micro, Mar./Apr., (1997), pp. 33-44.

博碩士論文授權書

（國科會科學技術資料中心版本）

本授權書所授權之論文為本人在＿國立中山＿大學(學院)＿電機工程＿系所＿丙＿組＿八十九＿學年度第＿二＿學期取得＿碩＿士學位之論文。

論文名稱：在記憶體處理器系統上改善工作負載平衡與程式最佳化

☐同意　☐不同意　**※非博士暨教育類碩士論文本項免鉤選**

本人具有著作財產權之論文全文資料，授予行政院國家科學委員會科學技術資料中心、國家圖書館及本人畢業學校圖書館，得不限地域、時間與次數以微縮、光碟或數位化等各種方式重製後散布發行或上載網路。

本論文為本人向經濟部智慧財產局申請專利的附件之一，申請文號為：＿＿＿＿＿＿，註明文號者請將全文資料延後半年再公開。

☑同意　☐不同意

本人具有著作財產權之論文全文資料，授予教育部指定送繳之圖書館及本人畢業學校圖書館，為學術研究之目的以各種方法重製，或為上述目的再授權他人以各種方法重製，不限地域與時間，惟每人以一份為限。

上述授權內容均無須訂立讓與及授權契約書。依本授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。上述同意與不同意之欄位若未鉤選（第一項若非博士暨教育類碩士論文可不鉤選），本人同意視同授權。

指導教授姓名：黃宗傳

研究生簽名：李藍基　　學號：8831676.
（親筆正楷）　　（務必填寫）

日期：民國 90 年 6 月 13 日

1. 本授權書請以黑筆撰寫並影印裝訂於書名頁之次頁。
2. 授權第一項者，請確認學校是否代收，若無者，請個別再寄論文一本至台北市106-36和平東路二段106號1702室 國科會科學技術資料中心 王淑貞。(本授權書諮詢電話:02-27377746)
3. 本授權書於民國85年4月10日送請內政部著作權委員會(現為經濟部智慧財產局)修正定稿，89.11.21部份修正。
4. 本案依據教育部國家圖書館85.4.19台(85)圖編字第712號函辦理。