行政院國家科學委員會專題研究計畫 成果報告

應用於嵌入式異質多核心平台之工作管理演算法的設計與實作（嵌入式系統軟體關鍵技術開發分項）
研究成果報告(精簡版)

計 畫 類 別 ：整合型
計 畫 編 號 ：NSC 97-2218-E-009-030-
執 行 期 間 ：97 年 08 月 01 日至 98 年 10 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：單智君

計畫參與人員：碩士班研究生-兼任助理人員：陳冠男
碩士班研究生-兼任助理人員：王惠珊
博士班研究生-兼任助理人員：吳奕緯
博士班研究生-兼任助理人員：高淑娟

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 98 年 10 月 13 日

行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告
□期中進度報告

應用於嵌入式異質多核心平台之系統軟體關鍵技術與開發工具-

應用於嵌入式異質多核心平台之工作管理演算法的設計與實作

計畫類別：□ 個別型計畫 ■ 整合型計畫
計畫編號：NSC 97－2218－E－009－030－
執行期間： 97 年 8 月 1 日至 98 年 7 月 31 日

計畫主持人：單智君 副教授
計畫參與人員：吳奕緯、陳冠男、楊惠珊、高淑娟

成果報告類型(依經費核定清單規定繳交)：□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(所)
中 華 民 國 98 年 7 月 31 日

目錄

一、中英文摘要

中文摘要

為了滿足對計算機系統日益增加的效能需求，目前處理器的架構主要藉由增加處理器核心(processor core)的方式來達到這個目的，這樣的架構通稱為多核心(multi-core)處理器。由於嵌入式(embedded)系統在功率消耗上的限制，因此 embedded 系統中的 multi-core 處理器通常採用異質多核心(heterogeneous multi-core)架構。為了要在異質多核心處理器開發與執行程式，必須有相關的軟體開發工具來配合。因此，本計畫針對異質多核心處理器開發一套程式碼到程式碼編譯器(source-to-source compiler)，用來協助程式設計者(programmer)自動找出程式中的 DOALL 迴圈(loop)，並分析哪些 DOALL loop 是適合哪一種類的處理器核心上執行。根據分析的結果，系統會將選定的 DOALL loops 提出(outline)成數個新的函式(function)，並在這些 functions 前增加必要得指導性註解(directive)，來讓後端的編譯器(compiler)可以產生相對應的程式碼。藉由本計畫開發的工具，將大幅地減少 programmer 在開發程式時的困難度。目前，本計畫所開發的工具主要針對 IBM 的 Cell 處理器來設計。此外，透過部份程式碼的修改，本計畫所開發的工具，將可以快速的支援其他類型的異質多核心處理器。

關鍵字：嵌入式系統、多核心處理器、異質多核心架構、編譯器、靜態程式碼分割/配置/排程、動態行程配置/排程

英文摘要

To meet the increasing demand for computing performance, more and more embedded systems deploy multi-core architectures to achieve this aim. Because of the diversity of architectures, multi-core architectures are usually classified into two categories: homogeneous multi-core and heterogeneous multi-core. Since heterogeneous multi-core architecture is an application-specific design, it has advantages in terms of computing performance and power consumption on many applications. These advantages are exactly what most of the embedded systems need. Nevertheless, due to the lack of software development tools, it is difficult to develop and execute programs on an embedded heterogeneous multi-core system. This motivates us to design and implement a source-to-source compiler to assist a programmer to properly assign each DOALL loop within an application on one type of processor cores.

Keyword: embedded systems, multi-core processors, heterogeneous multi-core architecture, compiler, code static partition/allocation/scheduling, process dynamic allocation/scheduling

二、報告內容

(1) 系統功能概述

IBM Cell 處理器(processor)是由一個主控處理單元(PowerPC® processing element，PPE)以及八個協同處理單元(synergistic processing element，SPE)構成的處理器。本計畫主要是設計一套 IBM Cell 處理器的程式碼到程式碼編譯器(source-to-source compiler)，用來協助程式設計者(programmer)分析程式碼中哪些 DOALL loop 是適合在 PPE 上執行，哪些部分是適合在 SPE 上執行。根據分析的結果，開發工具會將適合在 SPE 上執行的程式碼片段合併成一個新的函式(function)，並在 function 前增加必要的指導性註解(directive)來讓 Cell 處理器的後端軟體開發工具產生對應的 PPE 與 SPE 的程式碼。藉由本計畫開發的工具，我們預期可大幅地減少 programmer 在開發 Cell 程式時的困難度。

(2) 系統軟硬體平台介紹

本開發工具主要使用的硬體平臺是 IBM Cell 處理器[1]。Cell 處理器是由 SONY、TOSHIBA、與 IBM 聯合開發的處理器。它包含了九個處理器核心(processor core)，其中有一個 64 位元的 PPE(PowerPC 架構)與八個 SPEs，其架構如圖 1 所示。目前 Cell 已經被應用在 SONY 新一代的遊戲主機 PS3 上。

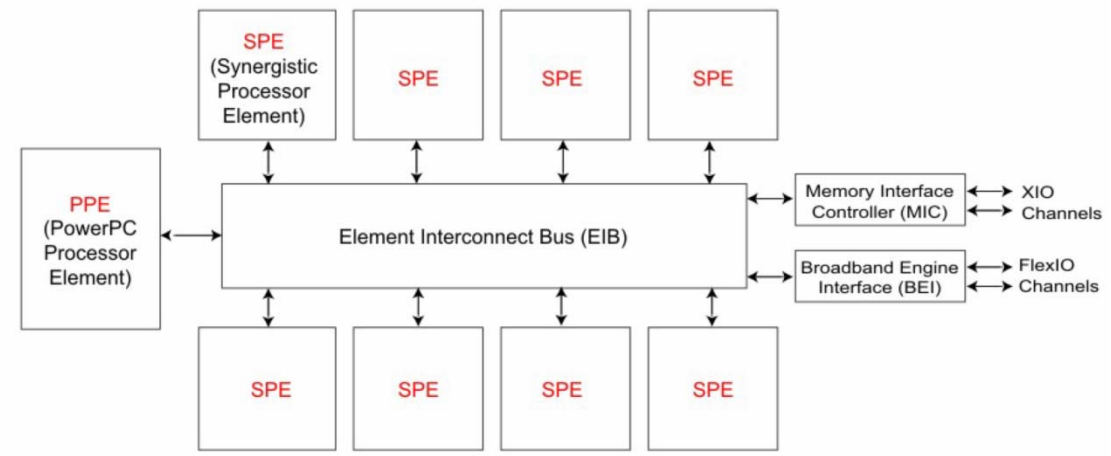


圖 1：IBM Cell 處理器的架構圖[1]

在軟體的部分主要將採用 Rose compiler [2]，以及 CellSs [3]這兩套開放原始碼的軟體，簡述如下：

- Rose compiler：Rose 是一個開放源碼的 compiler framework，主要由勞倫斯利弗莫爾國家實驗室(Lawrence Livermore National Laboratory)所開發。使用者可以根據需求使用 Rose compiler 來建構一套新的程式碼分析工具。目前 Rose 可以支援 C、C++與 Fortran 等語言。
- CellSs：CellSs 是一套 Cell 處理器的開發工具，其中包含一套程式模型(programming model)與一套程式碼分割(partition)工具，用來將使用此 programming model 開發的程式碼分成 PPE 或 SPE 可以執行的程式碼。

(3) 系統架構說明

本系統稱為異質多核心處理器靜態程式碼分割與排程系統（Heterogeneous Multi-Core Static code Partition and Scheduling system, HMCSPS）主要是針對程式碼進行靜態(static)的分析來找出程式中的

DOALL 迴圈(loop)，並針對這些 loops 的特性分析是否適合放在 SPE 上執行。如果適合，則將適合的 loops 提出成一個或數個新的 functions，並在這些新產生的 functions 前增加必要的 directive 來告知後端的 compiler 產生不同處理器核心的執行檔。據此，整套系統的主要功能可分割為三個子系統，分別敘述如下：

- CTGC（Conditional Task Graph Construction subsystem, 條件式工作圖產生子系統）

  CTGC 將輸入程式的 source code 轉成中間代碼（intermediate representation, IR），藉此找出source code的DOALL loops，並把這些DOALL loops提出(outline)成一個或數個新的函式(function)，在此稱這些新產生的functions為DOALL functions。

- TEPE（Task Execution Performance Estimation subsystem, 工作效能估算子系統）

  TEPE根據CTGC的輸入資訊與PPE與SPE的執行能力來估算每個DOALL functions在PPE與SPE上的執行時間需求，以及每個DOALL functions的資料傳遞量。TEPE將此結果儲存於一檔案中，並將此檔案傳給TPS子系統。

- TPS（Task Partition and Scheduling subsystem, 工作分割與排程子系統）

  TPS根據TEPE的結果來決定哪些DOALL functions是適合放在PPE或SPE上執行。並在這些選出的DOALL functions前增加必要的directive來告知後端的compiler產生不同處理器核心的執行檔。

圖 2 為 HMCSPS 的系統架構圖，說明了 HMCSPS 與其包含的數個子系統間的互動關係，及其與使用者和操作環境的關係。

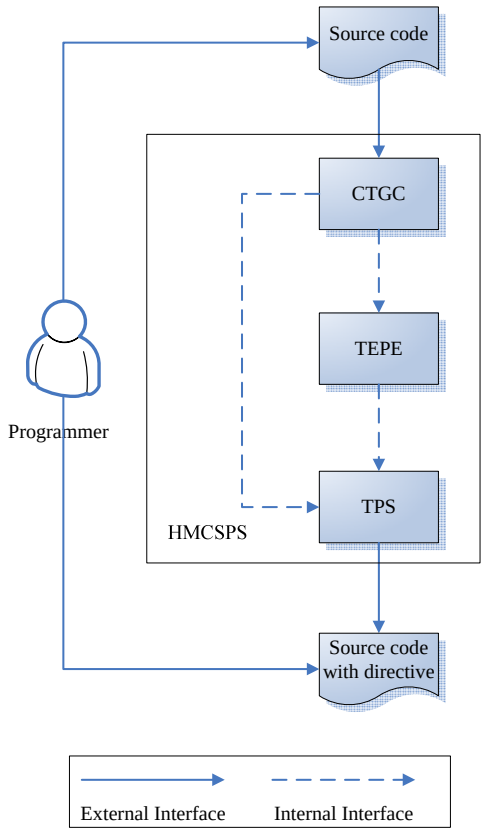
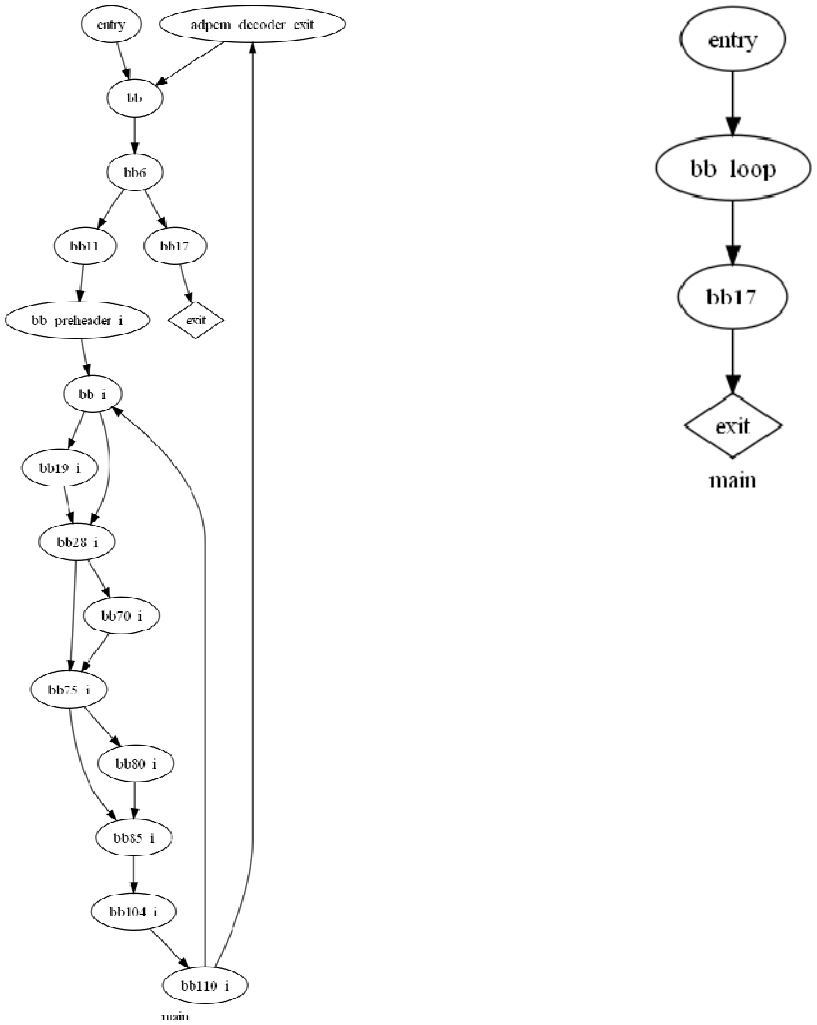


圖 2：HMCSPS 系統架構圖

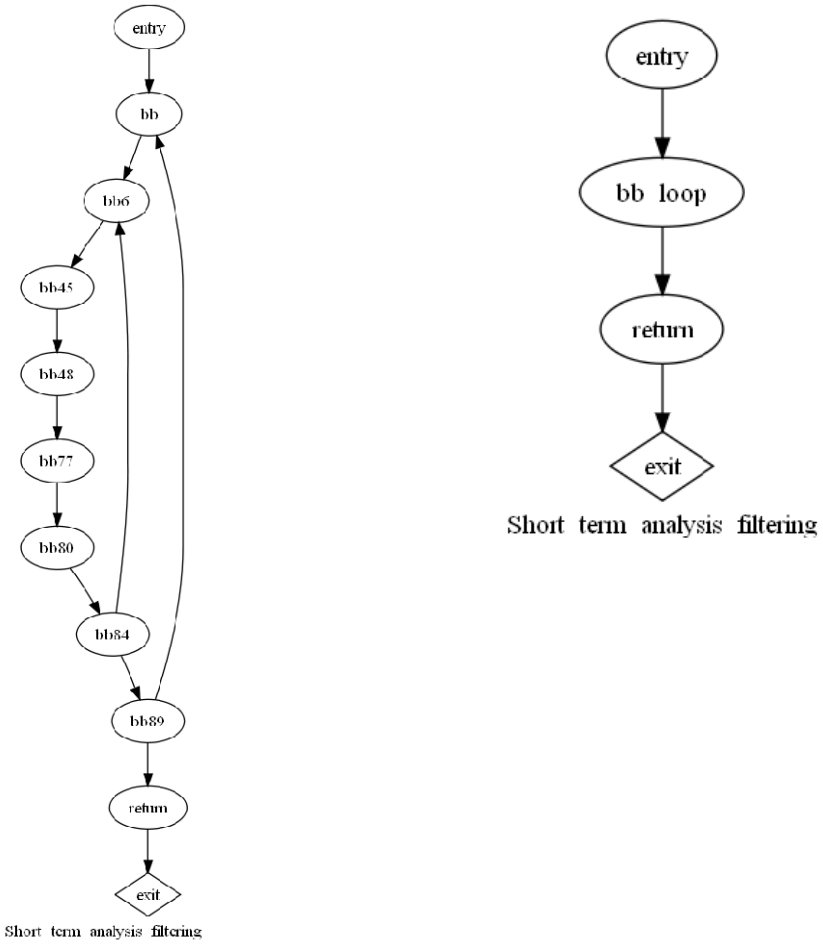
CTGC原本是設計來產生程式的conditional task graph（簡稱task graph）。為了讓系統可以對程式執行程式碼的partition與scheduling，在此每個task graph都必須是一個有向非循環圖(directed acyclic graph, DAG)。然而，當我們觀察多支嵌入式系統中常用的測試程式（如：MiBench [4]）後，發現所有的程式在轉成DAG後，每個task graph中的node數目甚少。圖3與圖4列出兩個MiBench中程式的task graph，分別是Adpcm decode與GSM decode。在圖3與圖4中，左邊的圖（(a)部份）是程式還沒轉成DAG前的圖形，圖中的每個node代表程式中的一個基本區塊(basic block)，而node間的連線則表示basic block間存在有data或control dependence。右邊的圖（(b)部份）則是程式轉成DAG後的結果。因此，倘若程式中僅可找出數個tasks時，則無需電腦的協助便可對此程式做partition與scheduling。因此，我們便將CTGC的工作由原本task graph建立轉成DOALL loop的尋找。此外，

CTGC還會將每個DOALL loop提出成一個新的function（在此稱為DOALL function），並且判斷每個DOALL function的輸入/出變數的大小。這部份的結果主要是用來產生directive中所需的資料，其用途主要是告知PPE該傳送多少的資料量給SPE。



(a) (b)

圖3：Adpcm decode的task graph



(a) (b)

圖4： GSM decode的task graph

TEPE 的工作就是分析 CTGC 所找出的 DOALL function 是否適合放到 SPE 上執行。這部份主要是根據 DOALL function 在 SPE 上預估的執行時間與傳送 DOALL function 輸入與輸出資料的時間來評估一個 DOALL function 是否適合放在 SPE 上執行。TEPE 會將此分析結果儲存於一檔案中，並將此檔案傳給 TPS 子系統。

TPS主要包含partition與scheduling這兩部份。Partition就是根據TEPE效能評估來判斷哪些DOALL function是適合放到SPE上執行。倘若一個DOALL function是適合放到SPE上執行，則TPS會在此DOALL function前加

註必要的directive。在本計畫中，我們使用CellSs所定義的directive來為每個適合放到SPE上執行的DOALL function加註。Scheduling就是排定程式中的每個task的執行次序；其中，一個task會由單一個function或是單一或數個迭代(iteration)所組成。換言之，一個DOALL function會被分割成數個tasks來平行執行。在本計畫裡，scheduling是透過CellSs提供工作的動態管理(runtime task management)來達成。CellSs的runtime task management會根據目前硬體系統的狀態與task間的相依性(dependence)來安排tasks的執行次序。此外，在本計畫中，TPS處理的對象是tasks，因此不會改變tasks內部指令（instruction）的執行次序。

圖 3 為一個含有 directive 的程式碼範例，這個範例主要是將兩個方陣（矩陣 A 與 B）做相乘，並把乘積的結果放到另一方陣（矩陣 C）中。此外，在這個範例中，矩陣會被分成 NB 個區塊來分別計算，其中每個區塊的大小就是 B。在圖 5 左上方的方框裡的程式碼是整個矩陣乘法的 main () function，而矩陣乘法主要的工作則是交由 block_addmultiply()這個 function 來處理（也就是圖 5 右下方的方框裡的程式碼）。在 block_addmultiply()有一行用#符號所標記的程式碼就是 CellSs 專屬的 directive，主要是告知 CellSs 兩件事：

I. block_addmultiply()這個 function 是需要放到 SPE 上來執行。
II. block_addmultiply()的輸出與輸入資料分別是 A, B, C 這三個矩陣，其中矩陣 A 與 B 是輸入型態，而矩陣 C 則是兼具輸出與輸入型態。輸入型態的資料是指系統需要將這些資料從 PPE 的主記憶體(main memory)傳送至 SPE 的本地記憶體(local memory)，但是無須傳回。兼具輸出與輸入型態是指系統需要將這些資料從 PPE 的 main memory 傳送至 SPE 的 local memory，且在 function 執行結束後，仍需將資料從 SPE 的 local memory 傳回 PPE 的 main memory。

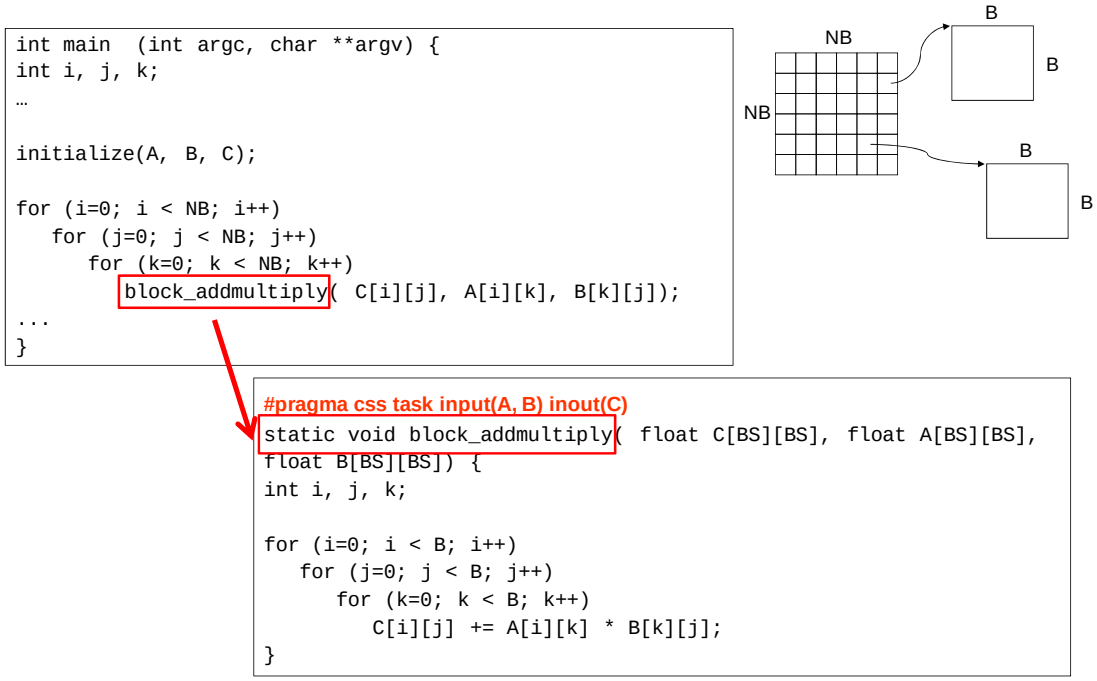


圖 5：含 directive 的程式碼

圖 6 則是 task 在 runtime 時的執行過程。其中，虛線框起來的部分是 CellSs PPE lib(簡稱 PPElib)所負責的工作範圍，有：資料相依性(data dependence)的檢查、資料更名(data renaming)、與 scheduling 等。當有新的 task(此稱 task A)要被執行時，PPElib 會根據 directive 找出 task A 與其他 tasks 間的 dependence 關係，並把這層關係記錄在 task graph 中。倘若，跟 task A 有 dependence 關係的 tasks 都已經執行完畢，而且 task A 所需的 data 都已經備齊。此時，PPElib 就會檢查是否有空閒的 SPE 可以使用。若無，則 PPElib 會將 task A 放到一個佇列(queue)中，讓 task A 等待其他執行優先權較高的 tasks 都執行完畢後才開始執行。當 task A 開始要執行時，PPElib 會使用資料同步(data synchronization)的機制將 task A 所需的 data 都搬到 SPE 的 local memory，讓 task A 可以就近存取所需的 data。當 task A 需要將 data 寫回 PPE 的 main memory 時，PPElib 則需要判斷這些要寫回的 data 是否會產生資料危障(data hazard)。若會，PPElib 就必須對這些 data 進行

data renaming 的工作，也就是把這些 data 先寫到一個暫時的 memory 位置，等到 data hazard 解除後，再將這些 data 從暫時的 memory 位置寫回 main memory 中。

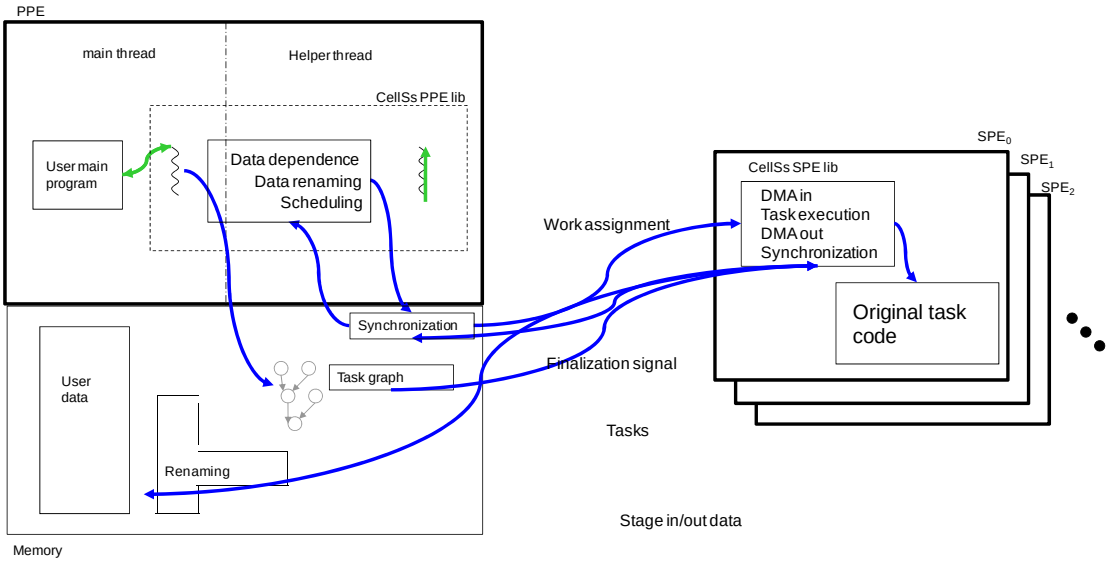


圖 6：Task 的動態執行過程 [3]

(4) 實驗結果

本計畫的實驗環境主要包含兩個部份：(一) Source-to-source compiler；(二) Cell Compiler。第一部份是使用 Rose compiler 所提供的 library 來開發完成；第二部份則是直接透過 CellSs 來達成。礙於 CellSs 的限制，目前能夠順利透過 CellSs 編譯成功的程式只有 MiBench-Basicmath 與矩陣乘法，其中 MiBench-Basicmath 包含了兩個 functions，分別是開根號與徑度和度數轉換。表 1 為在 PPE 與 PPE+SPE 的情況下，這些程式所需的執行時間比較。從表 1 中，我們看到開根號與徑度和度數轉換這兩個 functions 都可以藉由 SPE 來得到執行效能的提昇。這是因為這兩個 functions 主要的執行時間都是由一個 DOALL loop 所主宰著。因此，只要將這個 DOALL loop 平行分配到不同的 SPE 上執行，便可以得到執行效能

的改善。另一方面，在表 1 中，我們也可以看到在使用 SPE 後，矩陣乘法的執行時間反而拉長。這是因為雖然矩陣乘法的執行時間也是由 DOALL loop 所掌控，但是如果將此 DOALL loop 交由 SPE 來執行，則在執行前需要將所要計算的矩陣資料先傳送到 SPE 的 local memory 上，等待計算完成後，再將結果回傳到 PPE 的 global memory。由此一來，矩陣乘法的主要執行時間將不是耗費在此 DOALL loop 上，而是在矩陣資料的傳輸上。相較之下，開根號與徑度和度數轉換這兩個 functions 都只有傳送幾個變數到 SPE 上執行，因此在資料傳輸的時間上便小許多。這也是為什麼開根號與徑度和度數轉換這兩個 functions 可以得到執行效能改善的主因。

表 1：執行效能比較表

	PPE 執行時間 (秒)	**PPE + SPE 執行時間 (秒)**
開根號	13.554	2.747
徑度和度數轉換	4.313	1.095
矩陣乘法	1.235	4.484

三、參考文獻

[1] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, D. Shippy, “Introduction to the Cell multiprocessor”, *IBM J. Res. & Dev*. , September 2005.

[2] Rose compiler, http://www.rosecompiler.org

[3] Pieter Bellens, Josep M. Perez, Rosa M. Badia and Jesus Labarta, “CellSs: a Programming Model for the Cell BE Architecture”, *Supercomputing Conference*, 2006.

[4] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, Richard B. Brown, “MiBench: A free, commercially representative embedded benchmark suite,” I*EEE 4th Annual Workshop on Workload Characterization*, Austin, TX, December 2001.

四、計畫成果自評

本計畫目前的執行成果與當初計畫規劃時有些出入，主要的原因是，在嵌入式系統中，將程式轉成一個有向非循環圖(directed acyclic graph, DAG)後，整個 task graph 僅包含數個節點（node），其中每個 node 表示程式中的一個 task，而 node 間的連線則表示 task 間存在有 data dependence。換言之，倘若程式中僅可找出數個 tasks 時，則無須電腦的協助便可對此程式做程式碼的靜態（statically）分割（partition）與排程（scheduling）。因此，我們將計畫的執行內容由原本的程式碼的靜態分割與排程轉成 DOALL loop 的尋找與 directive 的自動產生。

目前本計畫已經完成 DOALL loop 的尋找與 directive 的自動產生這兩部份的程式設計，並且也已透過數個測試程式來證明這兩部份程式的正確性。由於目前對於大部分的 programmers 來說，開發 Cell 處理器程式或是其他異質多核心處理器程式都是一個相當大的挑戰。藉由本計畫開發的工具，我們認為可大幅地減少 programmer 在開發 Cell 程式時的困難度。此外，透過部份程式碼的修改，本計畫所開發的工具，將可以快速的支援其他類型的異質多核心處理器。