行政院國家科學委員會專題研究計畫成果報告

利用程式切片技術來平行執行程式(II)
Parallel Program Execution with the Help of Program Slicing

計畫編號：NSC 87-2213-E-009-024
執行期限：86 年 8 月 1 日至 87 年 7 月 31 日
主持人：楊武　副教授　　國立交通大學　資訊科學系

一、中文摘要

目前多處理器的相關軟體發展技術尚未成熟，而大多數的現有程式都是為單處理器系統設計，因此，我們提出能夠轉換現有程式，並讓程式設計師繼續使用其已有之設計程式的方法與觀念，而以軟體的技術將單處理器系統的程式，切割成平行合作的工作單位(tasks)，利用多處理器來同時執行這些工作單位。我們的切割方法將採用程式切片技術，將程式自動切割成可平行的工作單位。我們以 C++的程式為對象，將其剖析成盡可能互相獨立的單元，利用作業系統子計畫及通訊系統子計畫提供之系統服務常式，在各單元內加入適當的同步／通訊步驟，使各單元共同完成計算的目標。在本計畫的第一年裡，我們已完成了雛形的 C++程式切片系統及探討了二種切割原則。在本年度裡，我們繼續探討各種切割原則，並實作循序程式自動平行化系統。本報告介紹依據程式相依圖把循序程式自動平行化的系統。本子計畫有二個部份：第一部份是一套物件導向式的平行環境，是一個容易使用且物件導向化的程式架構，適合用來寫平行程式。此部份已在本計畫的第一年完成。第二部份是把循序程式平行化的方法，將在本報告中介紹。
關鍵詞：程式切片法、平行程式系統、系統軟體、多處理器系統

Abstract

Due to the rapid progress of computer CPU technology toward its physical limits, the feasible approach to manufacturing even faster computer systems is to coordinate many CPUs so that they cooperate to solve a common problem. This is so-called multiprocessor systems. Two main difficulties of multiprocessor systems are that (1) the design of multiprocessor architectures, and (2) the design of systems software for these multiprocessor architectures. For the second problem, we may design a new parallel programming language and ask the programmers to use these new parallel languages to write code. This approach raises the problem of personnel (re-)training and the so-called dusty deck problem. As an alternative to new parallel languages, we may design software tools that automatically transform sequential code into equivalent parallel one. In this way, the programmers are free to program in their familiar way. Our tool--a program slicing system for parallel program execution--cuts a C++ program into concurrently executable task based on the the dependencies of components in the program. In the past year, we have finished a prototype program slicing system (for C++) and studied two approaches to cutting programs. This year we investigate other cutting methods and implement the cutting and parallel execution system on the cluster of 2-CPU PCs. This report presents an automatic parallelization system for sequential programs based on program dependence graph. There are two parts in this project: the first part is an object-oriented parallel environment that provides an easy and object-oriented framework for writing parallel programs. This part has finished in the first year of this project. The second part is a method for parallelization of sequential programs. It is introduced in this report.
Keywords：program slicing, parallel program execution, systems software, multiprocessor systems

二、緣由與目的

目前電腦硬體技術突飛猛進，各式各樣的微處理器及相關系統晶片日益進步，然而目前的電腦仍多為單一處理器系統，電腦界的學者與業者無不殫精竭慮，希望製造普遍且價廉的多處理器系統，但是目前仍有二大瓶頸難以突破。

（一）如何在硬體電路板上安置多個處理器，並使它們通力合作。

（二）如何設計多處理器的作業系統核心與系統軟體。

硬體與系統軟體合作的目的，是提供一簡單而有用的應用程式設計介面(Application programming interface)。由於過去數十年程式設計及教育的經驗，一般的程式設計師比較習於為單處理器系統寫作程式，因此我們提出利用自動分割的技術，將單處理器系統模式的程式分割成平行執行的單元，以充分利用多處理器的電腦硬體。

為了能提高多處理器電腦系統的使用效率，這些平行執行的單元應該儘可能彼此獨立，也就是彼此間相依關係應減至最少。但是程式是一個嚴謹的邏輯計算整體，因此切割程式時，勢必無法切成完全獨立的單元，我們必須考慮一些因素以求得最佳的結果。

第一個因素是製造一個處理元的成本(cost of creating a new process)，如果成本高，顯然我們應採取少量處理元而集中工作的策略。反之，我們則可利用更多的處理元。

第二個因素是處理元間通訊的成本(cost of interprocess communication)，如果成本高，則處理元間的通訊量應儘量減少；欲減少處理元間的通訊數量，則需減少處理元的數目。

我們可知無論從製造處理元的成本或是從處理元間的通訊成本的觀點而言，我們均是希望減少處理元的數目，但是處理元數目過少，則系統無法有效運用其多個處理器，因此，如何決定適當數目的處理元，以充分利用多個處理器為一值得研究的課題。

分割程式基本上可以分成二大類：一是在程式執行時，視需要再加以分割，這一類稱為動態分割(dynamic partition)；另一類是在程式編譯時加以分割，這一類稱為靜態分割(static partition)，我們的研究重點是在靜態分割。在考慮靜態分割時，須考慮二類因素：一是程式執行的環境，這包括處理器的數目，硬體的架構(shared memory 或 distributed memory)，主記憶體容量，同步與通訊的方法，製造處理元的成本等，這一類的因素和程式本身無關，然而為了有效運用硬體架構，這一類因素是不可忽略的。第二類因素是程式本身的相依特性，在將程式分割成各個單元時，無論在何種程式執行環境裡，我們總是希望減少各單元之間的相依關係。因此在分割程式時，我們須特別注意程式內部各單位(entity)之間的相依關係。

我們切片系統的目的，是希望將多處理器的硬體架構儘可能隱藏起來，而讓應用程式的設計師針對一個熟習的單處理器的抽象模型來設計程式，我們的切片系統能自動利用程式內部的相依關係，將程式分散到多個處理器內同時執行，以減少程式執行所需的時間。

程式切片技術是審查程式內各單位之相依性，而將整個程式分割成若干工作單位(tasks)。各工作單位由相依關係密切的單位組成，而各工作單位之間，則毫無相依之關係，這種程式切片法必須將各工作單位共用之計算過程，在各工作單位內分別重覆計算。但是若任由各工作單位重覆進行這些冗長耗時的計算過程，有時並不合算，倒不如利用程式切片法來平行執行程式。

在分割程式時，我們首先將程式轉換成程式相依圖（program dependence graph)，在此圖中，每一個節點代表一個程式內部單元，至於單元之大小，可依不同的目的來決定。由於我們將以處理原始程式碼為目的，所以我們將各個簡單敘述（simple statements）及詢問述語（predicates）為單位。各單位之間箭號代表相依關係。相依關係有二種，一是控制相依關係（control dependencies）一是資料相依關係（data dependencies)。

有們說「a 對 b 有值（或假值）控制相依關係」我們指的是，如果 a 這個詢問述語在某次計算得到一真值（或假值）時，則 b 所代表的程式單位必定會執行一次。我們說「a 對 b 有資料相依關係」，我們指的是，a 單位計算出一個值會有可能被用在 b 單位之計算裏。除了這二種主要相依關係之外，另外也有其他的相依關係，如指定順序關係（def-order dependencies），輸出相依關係（output dependencies）等。這些相依關係代表程式內部各單位之間必要的邏輯關係，我們可以將程式作任意轉換，只要我們維持這些必要的邏輯關係，則程式執行的結困仍然相同，這些就是程式切片法的基本根據。

在完成程式相依圖之後，基本的程式切片法是依各單位之間的相依關依，逆溯而上，將所有可能會影響某變數最終值的單位包括進來，而將不相關的單位除去。因此任一程式切片，理論上應比原始程式為小，所以執行起來也會比較快些。我們可以將一個程式依此法分割成完成彼此完全獨立的切片，而這些切片各自計算結果的總合，便是原來程式計算之結果。但是由於這些切片彼此完全獨立，我可以在多處理器系統同時來執行這些切片，因而我們預期可以較短的時間，得到全部的結果。由於各切片大小不一，且切片的數目與實際參與執行的處理器數目可能相差懸殊，因此我們在必要時可以將數個小切片合成一個切片，並且在切片的工作排序（scheduling）問題，切片執行系統（run-time system）必須作適當的調配。

以上這種方法，仍然得出相當的代價。因為各切片必須完全獨立，因，此有些計算過程會同時在二個或數個切片裏重覆執行，因而造成了浪費，因而我們必須更進一步將若干切片，如切片甲與乙，共同的部分，獨立出來，自成一個工作單元（task 丙）而原來的切片甲與乙則加入適當的同步／通訊的指令，在執行甲與乙切片時，必須先等丙切片執行完畢將結果分別傳給甲與乙，然後，甲與乙切片便可同時進行剩下的工作，這個修正的方法，基本上以等待（waiting）代替重覆計算。這其間有一些互有得失的情形，如重覆的計算過多，則浪費了 CPU 的計間，如果我們要求完全無重覆，則切片系統有可能產生過多過於細碎的工作單元，則各工作單元會浪費過多的時間於同步／通訊上，因，此我們必須實際製一套系統後多作實驗，才可以決定適當的切片大小。

本「切片技術」子計畫與其他子計畫間，有密切的關係。為了提供一有效率、易使用的多處理機系統，必須先有一套易懂易用的應用程式設計介面(Application programming interface)。這一介面，最好能儘量與傳統的，單處理機系統之程式設計介面相同，以免造成程式設計師的困擾，而本「切片技術」子計畫的目的，便是提供此一易懂、易用的程式設計介面。另一方面，本子計畫須利用到許多系統內層的支援，如處理單元／處理線(process/thread)之產生與管理。處理單元之間的同步與通信、共享記憶體之管理等。這些都與「微核心」子計畫及「訊息傳遞」子計畫有密切的關係。本子計畫又與「平行編譯器」子計畫有相同的目的，並利用相關的技術。這二個子計畫均須分析程式內部各單位間的相依關係，然而此二計畫重點不相同：「平行編譯器」子計畫著重於陳列型資料給各元素個別的相依關係；而本子計畫著重於程式各單元的相依關係。

三、成果與討論

我們已經按照上一年度的計畫進度，完成一套 C++的雛形程式切片系統，此切片系統可以用來建立 C++程式相依圖及找出程式切片。在本計畫第一年裡，我們也探索過了如何利用程式切片工具來分割一整個程式成數個可以同時執行的切片程式。如何從程式相依圖區自動分出可以平行執行的程式區塊的相關文獻非常少，因此必須先研究出從程式相依圖自動找出平行程式區塊的方法。

目前，我們已經草擬出二種不同的方法，利用程式切片技術將程式自動平行化。第一種方法，將程式依其巢狀結構(nesting structure)，分為若干階層(hierarchy)，依其階層，由最外層向最內層，一次分析一層，計算在某一層敘述

(statements)、狀況(conditions)、及迴圈(for 及 while loops)之間相依關係，依其相依性，將該層的敘述、狀況與迴圈儘量分割成獨立的單位。在處理完每一層之後，我們可以得到許多小塊的獨立執行單位。由於這些獨立的執行單位可能過於細小，獨立執行的效用可能遠小於其獨立執行所需付出的代價(overhead)，因此我們再依給定的硬體與系統參數，如 CPU 數目、記憶體大小、通信之代價(communication overhead)、同步協調之時間(synchronization costs)等，合併這些小的獨立執行單位，成為較大的執行單位。

第二種方法則是從程式的相依圖找出其反向拓樸順序(inverse topological order)，再計算其可以同步執行的單位。其步驟是先建立循序程式相對應的程式相依圖，藉由分析程式相依圖內的相依關係，建立階層式平行工作圖(HPTG)。在可以忽略通訊成及緒行緒產生成本，以及處理器個數不限時，階層式平行工作圖代表最大平行度的工作圖。然而在現實世界中，通訊及產生執行緒都必須花時間，而且處理器個數都是有限的。因此，下一步是合併階層式平行工作圖產生緒行緒工作圖(TTG)，它代表一組用來平行執行工作的執行緒，每個執行緒內含有數個循序執行的工作，各執行緒之間也會通訊。這步驟有二個最重要的過程－分割與合併。分割演算法針對 HPTG 做分割，使各工作的執行時間最短。合併演算法把 TTG 內的執行緒合併以符合處理器的個數，並減少通訊的負載。依據最後的 TTG 把程式嵌入 OOPE 平行程式架構內，就形成了程式的平行版本。我們已按照上一年的進度，完成一套雛型程式切片系統，在本年度完成程式自動平行化系統。本子計畫第二年的成果，有以下諸項：

(1)找出從程式相依圖，把程式自動分割成可平行區塊的方法。

(2)製作程式自動平行化的系統。

四、計畫成果自評

在本計畫的執行過程中，我們研究了如何把傳統的循序程式自動平行化，並實做出把程式自動平行化的系統，我們亦提出了相關的研究報告。但本系統尚有一些有待加強的部份，例如加強平行程式架構，使其能夠自動平衡處理器的負載，計算時間測量的改善，以及工作排程問題等，值得繼續深入研究。

五、參考文獻

[1] Carl J. Beck Generating self-scheduling code for nonloop parallelism. Journal of Parallel and Distributed Computing. (39) 1996.

[2] J. Ferrante, K. J. Ottenstein and J. D. Warren. The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems, 1987.

[3] Goran Lj. Djordjevic and Milorad B. Tosic. Aheuristic for scheduling task graphs with communication delays onto multiprocessors. Parallel Computing, (22) 1996.

[4] Girkar, M. And Polychronopoulos, C. Automatic detection and generation of unstructured parallelism in ordinary programs. IEEE Transaction on Parallel and Distributed Systems, April 1992.

[5] S. Horwitz, T. Reps and D. Binkley. Interprocedural slicing using dependence graphs. ACM Transactions on Programming Languages and Systems, 1990.

[6] Jien-Tsai Chan, An object-oriented program slicing system. Thesis for the degree of master, CIS NCTU, 1996.

[7] Martin C. Rinard and Pedro C. Diniz. Commutativity analysis: A new analysis framework for parallelizing compilers. Proceedings of the SIGPLAN '96 Conference on Program Language Design and Implementation, PA, USA, May 1996.

[8] G.C. Sih and E.A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Transaction on Parallel and Distributed Systems, (4) 1993.

[9] T. Yang and A. Gerasouli. DSC: Scheduling parallel program tasks on an unbounded number of processors. IEEE Transaction on Parallel and Distributed Systems, (5) 1994.

[10] K. J. Ottenstein and L.M. Ottenstein. the program dependence graph in a software development environment. Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environment, 1984.