微核心作業系統多處理機上系統軟體之研製(二)

The Implementation of Micro-Kernel-Based System Software for Multiprocessor Systems (II)

計畫編號：NSC87-2213-E009-022
執行期限：86年8月1日至87年7月31日

主持人：　曾憲雄　　國立交通大學資訊科學系教授
共同主持人：張瑞川　　國立交通大學資訊科學系教授
袁賢銘　　國立交通大學資訊科學系教授
楊　武　　國立交通大學資訊科學系副教授

一、中文摘要

關鍵詞：多處理機系統、微核心作業系統、訊息傳遞系統、平行編譯器、程式切割、平行程式除錯器

在多處理機系統上，平行度的利用是非常重要的課題。因此，有許多研究是關於發展微核心作業系統與相關系統軟體，這也正是我們研究與探討的方向。本計畫製作出適用於SMP及SMP cluster的微核心作業系統，以及相關作業系統性能改進及製作出適用於微核心作業系統的訊息傳遞系統，並以Knowledge-based技術為基礎的平行編譯器原型和製作出平行程式切割系統。系統的研究雖可勉強分割為微核心作業系統、訊息傳遞系統、平行編譯器、平行程式切割系統和平行程式除錯系統等等，其運作卻是緊密連結，不可分割的；研究若能有整體，可更加掌握系統之瓶頸及改進之道。本計畫的研究成果,對未來發展微核心作業系統與相關系統軟體，在學術或技術上都可提供重要的參考。

二、英文摘要

Keywords：multiprocessor systems, parallelization, micro-kernel operating system, parallelizing compiler, message passing system

The parallel utilization for shared memory multiprocessor systems is a very important issue; hence, there are many researches concentrated on designing and implementing micro-kernel operating systems and system software for multiprocessor systems. Our investigations also focus on it. The investigation consists of the efforts of five research projects. It aims at achieving high speedup on multiprocessor systems by using micro-kernel operating systems and system software. The system software consists of message passing system, parallelizing compiler, and parallel program slicing. In the study of high performance operating system and system software, results of this project will be able to deliver theoretical and technical contributions.

三、計畫緣由與目的

由於VLSI技術快速發展使的微處理器的計算效能大幅提昇，目前的個人電腦以能有效應付一般的文書處理及簡單的計算工作。但是對於一些需要大量計算的工作，例如 Grand Challenge 等問題，單一微處理機的計算能力又顯得不足。倘若一部平行電腦之處理器個數可由數個擴充到數千個，使用者可依目前的經費，暫時採購具較少處理器個數的平行電腦，往後依經費狀況及需要再加以擴充。而一個具延展性的平行程式，可依處理器數目的多寡，發揮其最佳的執行效能。在處理器數目改變時，程式並不需要隨著變動，節省了程式再發展的成本。因此可研展性 Scaleable Computing 的研究，可以使整個平行電腦發揮其最佳功能，亦能保障已發展之平行程式的適用性。

可延展性的平行電腦系統大致可分成兩類：1、私用記憶體多處理系統(Private Memory Multiprocessor System)，2、共用記憶體多處理系統(Shared Memory Multiprocessor Systems)。前者每個處理機只有私用記憶體，處理機之間並不存在共用記憶空間，所以必須以信息傳送(Message Passing)方式相互溝通。而後者的處理機間除了有私用記憶體外，尚存在一塊共用記憶空間，此空間可能採硬體實現也可能利用軟體實現，處理機之間可採共用變數(Shared Variable)溝通。至於在硬體上如何組織這些微處理器，則有相當多種方法。一種較為公認的分類原則是，處理器間有實體共享記憶體的機器稱為 Tightly Coupled MIMD，沒有實體共享式記憶體的機器稱為 Loosely Coupled MIMD。

在 Tightly Coupled MIMD 上，微處理器間的溝通可直接經由共享記憶體，這和原本的程式設計習慣相似，並沒有增加程式設計師太多的負擔；但在 Loosely Coupled MIMD 上，微處理器間必須靠額外的訊息傳遞機制彼此溝通，這就和原本的程式設計習慣相當不同。不過 Tightly Coupled MIMD 也有缺點，因為記憶體匯流排頻寬的限制，所以在一部機器裡無法支援太多微處理器，這也是市面上大部分的 SMP 機器微處理器通常不超過四顆的原因；反觀 Loosely Coupled MIMD，因為處理機不會使用同一個記憶體匯流排，所以就會有比較好的擴充性，可以同時支援很多微處理器。所以，如果程式裡的平行性不大，平行編譯器的開發者會使用

Tightly Coupled MIMD。但若程式有相當多的平行性可以利用，無可避免的，就一定要使用 Loosely Coupled MIMD 來執行平行編譯器所產生的程式。因此，平行編譯器的設計者必須想辦法同時支援這兩種不同的硬體平台，不過，由於兩者間的差異性甚大，如果不靠其他系統軟體的幫助，想讓平行編譯器產生出來的程式碼，可以同時支援這些硬體，幾乎是不可能的事。

對平行編譯器的設計者來說，最理想的狀況是，無論在哪一種硬體平台上，都可以不必改變平行編譯器產生的程式碼，硬體上的差異，交給系統負責處理。為了支援在本整合行計畫裡的兩個和平行編譯器相關的子計畫，即子計畫三與子計畫四(Parallelizing compiler 和 Parallel program slicing)，我們有子計畫一，負責移植能利用目前最新的 SMP 和 server Clustering 技術的微核心作業系統(OSF/1) 到 Intel 的 SMP 平台上，但這並不夠，因為 OSF/1 只支援以訊息傳遞為基礎的行程間通訊機制(interprocess communication mechanism)，並不能完全滿足平行編譯器的需求。所以，才有子計畫二的出現，提供一套能讓平行編譯器設計者很方便使用的行程間和行程內的通訊機制，以簡化他們的工作。

四、研究方法與成果討論

在子計畫一中，本計劃今年完成的工作如下: 利用 SCSI 及 Myrinet 技術建構 SMP cluster。我們自行設計了一個可靠的 MCP(Myrinet control program) 以及驅動程式，提供了一個 high bandwidth, low latency 且 reliable 的連結。我們也修改了 OSF1/AD 的 DIPC、KKT 等部份，Myrinet 移植到 OSF/1 AD 上，使其能有效的應用我們所發展的快速訊息傳遞技術。

我們實驗的設備及環境如下：

- OSF/1 AD3.0
- 4-ports Myrinet 交換器(M2F-SW4)
- Myrinet 網路介面卡 (M2F-PCI32-10556 with 256KB memory on-board) *4
- Intel Pentium-133*2 、Intel Pentium-133 SMP、Pentium II 233

在子計畫二中，我們曾在去年的國科會計畫 - 行程內與行程間的通訊同步機制 (II) 裡，設計並實作出一個兩層式 (thread level/process level) 的行程內同步機制 [1]，我們今年的研究工作，就是試圖在原有的研究成果上，繼續擴充，在原本的 thread level/process level 同步機制上，再加上一層 network level 的同步機制。我們提出了幾種可能的架構，經過分析後，我們決定採用圖 1 的系統架構。

這種架構雖然在設計及實作上的複雜度較高，但是，因為它在每部機器上，多了一個 host agent，因此，在網路上傳遞的訊息數量，會因此而減少，進而增加了系統的 throughput。而且，使用這個方法，可以在完全不更動我們之前的 thread level/process level 的同步機制的情況下，network level 的同步機制，建置在原有的系統上。

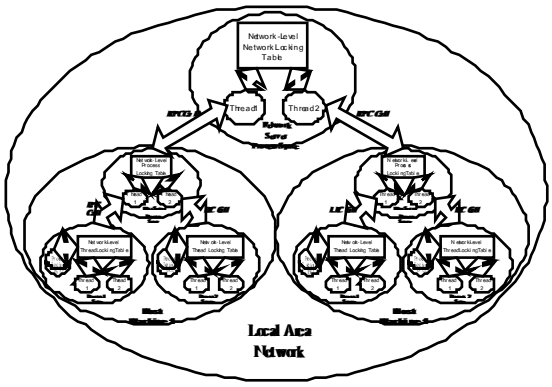

圖 1：Network-Level Lock Facility 的系統架構

至於在使用者界面上，還是沿用類似 thread level/process level 同步機制的使用者界面。下面是關於這個界面的簡單說明：

1. int *open_nlock*(char **nname*, int *lock_type*);

   開啟一個 network level lock。

2. **BOOL** *set_nlock*(int *descriptor*, int *op*);

   要求某個已被開啟的 network level lock。

3. **BOOL** *reset_nlock*(int *descriptor*);

   釋放某個已被要求的 network level lock。

4. **BOOL** *close_nlock*(int *descriptor*);

   關閉某個已被開啟的 network level lock。

5. **BOOL get_nlock_status**(int *descriptor*);

   取得某個已被開啟的 network level lock 的狀態。

至於這些 API 詳細的使用方法，請參考結果報告裡的完整說明。

在支援容錯的功能上，我們採用了 modular redundancy [7] 的容錯方式。這種方式雖然有 failure recovery 比較快，以及可以避免 Byzantine failure 等好處，但是，當被複製的 server 是 concurrent server 時，會因為每個被複製的 server 上，concurrency control 的執行結果不確定性，而導致每個 server 的副本，可能會產生不同的執行結果的情況出現。在這裡，我們為了提高系統的 throughput，而決定採用 concurrent server，但我們也想出一個方法，來解決這樣的問題。至於進一步的細節，請參考我們結果報告上的完整說明。

在子計畫三中，我們承襲專家系統精神，再沿用於平行迴圈的事前轉換處理。就一般的 Program 而言，並非所有的程式均可以平行化，但在針對某些特定的 Domain 時，有些程式在經過適當的處理時，平行化卻可以提高。所以為了增加平行度，這前置作業是必須的，且在做過適切的轉換後，更有利於接著要做的資料相依性的測試、平行迴圈的排程和平行程式碼的轉換。未來我們應用於各種不同的 Domain，做平行迴圈的轉換(Parallel Loop Transformation)，根據迴圈的資料特性和問題的範籌來建立知識庫。因為他們之間的變因太大，無法針對各種情況做最佳的安排，這也就是為什麼我們要採用專家系統，來完成所要的資料判定。此一研究，已在我們發展的可移植式平行編譯 PFPC 中實際製作出來，實驗結果並顯示可使編譯器產生更有效率的執行碼。

我們在研究中發現，迴圈在一程式中存在有大量的平行度，為了此程式平行化，平行編譯器利用靜態資料相依性分析來獲得迴圈的平行度。然而，有些迴圈則無法於編譯時期取得資料相依性的資訊。例如，在稀疏矩陣計算上，陣列述語內通常包含了間接陣列或函式，便無法完成靜態資料相依性分析。故便保守的程式循序的執行，而犧牲了潛在的平行度。因此，在本計畫中，我們提出一個兩階段(偵測階段及執行階段)的執行時期平行化方法於執行時期擷取出迴圈中潛在的平行度。偵測階段經由建立一 DEF-USE 表而決定出可平行執行的迴圈輪替集合-波前。此外，此偵測階段本身可以被完全的平行化而減少因決定波前所造成的額外負擔。這種兩階段的方法，也在我們已發展的可移植式平行編譯(PFPC)中實際製作出來，提昇編譯器產生有效率的執行碼。

在子計畫四中，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，利用程式切片技術程式自動平行化。第一種方法，程式依其巢狀結構(netsting 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，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)。階層式平行工作圖代表最大平行度的工作圖。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。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。

未來電腦系統的發展趨勢預估是以 Symmetric Multiprocessor Systems(SMP)為基本的構成單元，而伺服器則是使用 SMP 為單元加上適當的連接架構(Inter-connection)所構成的 Clustering。本計畫根據此一電腦發展趨勢研發相關的微核心作業系統(Micro-Kernel Operating System)、訊息傳遞系統(Message Passing System)、平行程式切割系統、平

行程式編譯器(Parallelizing Compiler)。所以我們有子計畫一，負責移植能利用目前最新的 SMP 和 Server Clustering 技術的微核心作業系統 (OSF/1) 到 Intel 的 SMP 平台上並利用 Mach 作業系統的支援程式庫。但這並不夠，因為 OSF/1 只支援以訊息傳遞為基礎的行程間通訊機制(Interprocess communication mechanism)，並不能完全滿足平行編譯器的需求。所以，又有子計畫二的出現，提供一套能讓平行編譯器設計者很方便使用的行程間和行程內的通訊機制，以簡化他們的工作。而子計畫四，實作一程式切片系統，可以整合於子計畫三的 Run-time Parallelization Method 以獲得程式切片後的結果 Clustering Parallelizing Compiler Design 其中最重要的因素包括(1)硬體及系統軟體的影響加入此平行化系統之中，(2)加入適當的同步與通訊指令，(3)分析重複計算(duplicate computation)與通訊(synchronized communication)之利弊，以導引平行化編譯工作。(4)產生適當的資料，協助作業系統來對這些程式迴圈作排程(Parallel Loop Scheduling)的工作。本總計畫的實作部份，在所購置的 2-CPU 個人電腦叢集(PC Clustering)上實施，成為一套完整的系統軟體。

五、計畫成果自評

未來的平行機器走向，將是採用不同的機器上之間的相互合作關係，也就是透過網路上資料的傳輸，來達成平行處理、分散計算的目的，也就是架構在非對稱的平行架構(NUMA)系統上，因此我們的 Parallelizing Compiler 也將朝向這個方向發展。我們會將現有的 Compiler System 從原來的對稱式機器(UMA)移植至平行式機器架構。在 NUMA 的系統下，因為是透過網路的傳輸，其 Overhead 是相當大的，為了減低 Message Passing 的 Cost，先前有關平行迴圈排程的問題就顯得相當重要了，我們利用資料之間的同資性及再使用性(Reusable)，來減少網路上存取的次數，並提高平行化程度。系統的研究雖可勉強分割為微核心作業系統、訊息傳遞系統、平行編譯器、平行程式切割系統和平行程式除錯系統等等，其運作卻是緊密連結，不可分割的；研究若能有整體，可更加掌握系統之瓶頸及改進之道。本計畫已發表共七篇論文，其研究成果，對未來發展微核心作業系統與相關系統軟體，在學術或技術上都可提供重要的參考。

參考文獻

[1] W. Blume, R. Eigenmann, K. Faigin, J. Grout, J. Hoeflinger, D. A. Padua, P. Petersen, B. Pottenger, L. Rauchwerger, P. Tu, and S. Weatherford, "Polaris: The next generation in parallelizing compilers," in Proc. 7th Int'l Workshop Languages and Compilers for Parallel Computing, Lecture Notes in Computer Science, 892:141-154, Springer-Verlag, Ithaca, New York, Aug. 1994.

[2] W. Blume, R. Eigenmann, J. Hoeflinger, and D. Padua, P. Petersen, L. Rauchwerger, P. Tu, "Automatic detection of parallelism: A grand challenge for high-performance computing," IEEE Parallel & Distributed Technology, 2(3):37-47, Fall 1994.

[3] U. Banerjee, R. Eigenmann, A. Nicolau, and D. A. Padua, "Automatic program parallelization," Proc. IEEE, 81(2):211-243, Feb. 1993.

[4] D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler transformations for high-performance computing," ACM Computing Surveys, 26(4):345-420, Dec. 1994.

[5] J. Boykin, D. Kirschen, A. Langerman, and S. LoVerso, Programming under Mach, Addison Wesley, 1993.

[6] R. Eigenmann, J. Hoeflinger, Z. Li, and D. Padua, "Experience in the automatic parallelization of four perfect benchmark programs," in Proc. 4th Annual Workshop Language and Compilers for Parallel Computing, Lecture Notes in Computer Science, 586:65-83, Springer-Verlag, Aug. 1991.

[7] G. J. Hwang and S. S. Tseng, "EMCUD: A knowledge acquisition method which captures embedded meanings under uncertainty," Int'l J. Man-Machine Studies, 33:431-451, 1990.

[8] M. C. Hsiao, S. S. Tseng, C. T. Yang, and C. S. Chen, "Implementation of a portable parallelizing compiler with loop partition," in Proc. Int'l Conf. Parallel and Distributed Systems, ICPADS'94, Hsinchu, Taiwan, R.O.C., pp. 333-338, Dec. 1994.

[9] K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, MIT Press and McGRAW-Hill Inc., 1993.

[10] S. H. Kao, C. T. Yang, and S. S. Tseng, "Run-time parallelization for loops," in Proc. 29th 1996 Hawaii International Conference on System Sciences, HICSS-29, vol. 1, pp. 233-242, Hawaii, Jan. 1996.

[11] D. J. Lilja, "Exploiting the parallelism available in loops," Computer 27(2):13-26, Feb. 1994.

[12] K. Loepere, Mach 3 Kernel Principles, Open Software Foundation and Carnegie Mellon University, 1992.

[13] K. Loepere, Mach 3 Server Writer's Guide, Open Software Foundation and Carnegie Mellon University, 1992.

[14] S. Leung and J. Zahorjan, Extending the Applicability and Improving the Performance of Runtime Parallelization, Technical Report UW-CSE-95-01-08, Department of Computer Science and Engineering, University of Washington, Jan. 1995.

[15] O'Reilly & Associates, Inc., Guide to OSF1: A Technical Synopsis, O'Reilly & Associates, Inc., Sebastopol, CA, June 1991.

[16] C. D. Polychronopoulos, Parallel Programming and Compilers, Kluwer Academic Publishers, MA, 1988.

[17] W. C. Shih, C. T. Yang, and S. S. Tseng, "Knowledge-based data dependence testing on loops," in Proc. 1994 International Computer Symposium, Hsinchu, Taiwan, R.O.C., pp. 961-966, Dec. 1994.

[18] M. Wolfe, High-Performance Compilers for Parallel Computing, pp. 137-162, Addison-Wesley Publishing, New York,

1995.

[19] C. T. Yang, S. S. Tseng and C. S. Chen, "The anatomy of parafrase-2," Proceedings of the National Science Council Republic of China (Part A), 18(5):450-462, Sep. 1994.

[20] C. T. Yang, S. S. Tseng, C. D. Chuang, C. T. Wu, S. H. Kao, and M. C. Hsiao, "A portable FORTRAN parallelizing compiler on shared-memory multiprocessor systems," in Proc. 1995 the First Workshop on High Performance Multiprocessor Systems, Hsinchu, Taiwan, R.O.C., pp. 56-62, July 1995.

[21] C. T. Yang, S. S. Tseng, C. D. Chuang, and W. C. Shih, "Using knowledge-based techniques for parallelization on parallelizing compilers," in Proc. the 1995 EURO-PAR Int'l. Conf. Parallel Processing, EURO-PAR'95, Lecture Notes in Computer Science, 966:417-428, Springer-Verlag, Stockholm, Sweden, Aug. 1995.

[22] C. T. Yang, S. S. Tseng, and M. C. Hsiao, "A model of parallelizing compiler on multithreaded operating systems," in Proc. the 1995 Int'l. Conf. High Performance Computing, HPC-ASIA'95, Taipei, Taiwan, R.O.C., Sep. 1995.

[23] H. P. Zima and B. Chapman, Supercompilers for Parallel and Vector Computers, Addison-Wesley Publishing and ACM Press, New York, 1990.

[24] Charles E. McDowell, David P. Helmbold, and Anil K. Sahai, "A Survey of Debugging Tools for Concurrent Programs", Baskin Center for Computer Engineering and Information Sciences, University of California, UCSC-CRL-87-22, Santa Cruz, California, Dec., 1987.

[25] Charles E. McDowell and David P. Helmbold, "Debugging Concurrent Programs", ACMCS, 21(4):593-622, Dec., 1989.

[26] Cherri M. Pancake and Robert H. B. Netzer, "A Bibliography of Parallel Debuggers, 1993 Edition", Proceedings of the ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, pp. 169-186, ACM SIGPLAN Notices, 28(12), Dec., 1993.

[27] Stallman, Richard M., GDB Manual, third edition, GDB version 4.x, Free Software Foundation, Cambridge, Mass., oct, 1994.

[28] Stallman, Richard M., "Using and Porting GNU CC, version 2.x, Free Software Foundation, Cambridge, Massachusetts, Jan., 1995.

[29] Mach kernel source code.

[30] Joseph Boykin; David Kirschen; Alan Langerman; Susan LoVerso; Programming under Mach, Addison-Wesley, 1993.

[31] Mach 3 Kernel Principles, Open Software Foundation and Carnegie Mellon University, 1991.

[32] Mach 3 Kernel Interface, Open Software Foundation and Carnegie Mellon University, 1991.

[33] The Design of the OSF/1 Operating System - PRELIMINARY, Open Software Foundation, 1993.

[34] Intel MP spec. V1.4 1995.

[35] P.A. Alaberg ang J. D. Day , " A principle for resilient sharing of distributed resources," in *proc. of 2nd Int. Conf. on Soft.*, Oct 1976, pp. 562-570.

[36] A. Avizienis, "The n-version approach to fault-tolerant software," in IEEE *Tansection on Software Engineering*, Vol. SE-11, No. 12, Dec 1985, pp. 1491-1501.

[37] J. S. Banino et al, "Some fault-tolerant aspects of the CHORUS distributed system," in *Proc. of 5th International Conference on Distributed Computing Systems*, May 1985, pp. 430-437.

[38] K. P. Birman, T. A. Joseph, T. Raeuejle and A. E. Abbdai, "Implementing fault-tolerant distributed objects," in *Proc. of 4th Symp. on Reliability in Distributed Softwave and Database*, 1984, pp. 124-133.

[39] A. Borg, J. Baumbach and S. Glazer, "A message system supporting fault tolerance," in *Proc. of 9th ACM Symp. on Operating System Principles*, Oct. 1983.

[40] Bulter W, Lampson and David D. Redell, "Experience with processes and monitors in Mesa," in *Comm. of ACM*, Vol. 23, No. 2, Feb. 1980, pp. 105-117.

[41] Douglas B. Comer, "Internetworking with TCP/IP," Prentice-Hall, 1991.

[42] E. C.. Cooper, "Replicated Distributed Programs," in *Proc. of 10th Symp. on Operation Systems Principles*, 1985, pp. 63-78.

[43] B. D. Fleish, "Distributed System V IPC in Locus : A design and implementation retrospective, " in Proc. of ACM SIGCOMM'. 86, Aug 1986, pp. 386-396.

[44] P. Gunningberg, "Voting and redundancy management implemented by protocols in distributed systems," in Proc. of FTCS-19 Jun 1983, pp. 182-185.

[45] Larsen, L.D. and M.J. Harrold , "Slicing Object-Oriented Software," TR 95-114, Clemson Univ., Computer Science Dept., November 1995.

[46] Reps, T. and T. Turnidge, "Program specialization via program slicing," TR-1296, Computer Sciences Department, Univ. of Wisconsin, Madison, WI., December 1995.

[47] Larsen, L.D. and M.J. Harrold , "Slicing Object-Oriented Software," TR 95-114, Clemson Univ., Computer Science Dept., November 1995.

[48] Reps, T. and T. Turnidge, "Program specialization via program slicing," TR-1296, Computer Sciences Department, Univ. of Wisconsin, Madison, WI., December 1995.

[49] Larsen, L.D. and M.J. Harrold , "Slicing Object-Oriented Software," TR 95-114, Clemson Univ., Computer Science Dept., November 1995.

[50] Reps, T. and T. Turnidge, "Program specialization via program slicing," TR-1296, Computer Sciences Department, Univ. of Wisconsin, Madison, WI., December 1995.