行政院國家科學委員會專題研究計畫成果報告

設計產生全括號串之平行演算法

Design of Parallel Algorithms for Generating Well-Formed Parentheses Strings

計畫類別：個別型計畫
計畫編號：NSC 89-2213-E-009-012
執行期限：88 **年** 8 **月** 1 **日至** 89 **年** 7 **月** 31 **日**

計畫主持人：蔡中川
共同主持人：

執行單位：交通大學資訊工程系

中華民國 89年 8月 日

行政院國家科學委員會專題研究計畫成果報告
設計產生全括號串之平行演算法

Design of Parallel Algorithms for Generating Well-Formed Parentheses Strings

計畫編號：NSC 89-2213-E-009-012
執行期限： 88 **年** 8 **月** 1 **日至** 89 **年** 7 **月** 31 **日**
主持人：蔡中川　交通大學資訊工程系
共同主持人：
計畫參與人員：黃為霖、蔡慶鴻　交通大學資訊工程系

一、中文摘要

在本計劃中，我們針對產生所有全括號串之平行演算法的設計問題加以研究。我們的研究結果為設計出一種產生所有全括號串之平行演算法，此平行演算法為一種最低計算花費之韻律演算法，執行此演算法之線性韻律陣列每三個計算步驟可產生一全括號串。

關鍵詞：組合體，線性陣列，平行演算法，韻律演算法，韻律陣列，全括號串

Abstract

In this project, The problem of designing a parallel algorithm to generate all the well-formed parentheses strings has been studied. We have designed a cost-optimal systolic algorithm to generate all the well-formed parentheses strings in this research project. The systolic algorithm is run on a linear systolic array that can generate one well-formed parentheses string each three time steps.

Keywords: combinatorial object, linear array, parallel algorithm, systolic algorithm, systolic array, well-formed parentheses string

二 、緣由與目的

離散演算法(或離散計算)學科所討論的問題，其演算(或計算)對象皆為非連續的數學結構[1-2]**。離散演算法的應用廣泛，尤其是在科學及工程方面。在離散演算法內我們經常需要產生或檢視所有合乎某條件的一些基本組合體**(Combinatorial Objects)**，常遇到的組合體有排列、組合、子集合、集合分割、全括號串、. . .等等。在文獻方面，已有許多論文提出演算法以產生這些基本組合體。早期的論文中大都提供順序演算法來解決此問題(例如排列**[1, 3-7]**、組合**[1,8]**、子集合**[1-2, 8-9]**、集合分割**[10-12]**、全括號串**[13-17])**。然而，近來由於平行計算的研究蓬勃發展，已有許多的平行演算法被陸續的提出(例如排列**[18-25]**、組合**[26-29]**、子集合**[30-31]**、集合分割**[30]**、全括號串**[18, 32])**。**

在本計劃中，我們針對設計產生所有固定長度全括號串組合體(簡稱全括號串或 well-formed parentheses string**)之平行演算法的設計問題加以研究。所謂括號串仍為多個左右括號所組成的文字串，而全括號串則為一種左右括號互相平衡的括號串，全括號串可由** $S \to (S) \mid SS \mid a$ **文法規則產生，其中** λ **為空字串，例如** ((((()(()))(()(())) **即為一全括號串。多年前就有多位學者研究如何產生所有全括號串：有些學者提出了產生所有全括號串之順序演算法**[13-17]**，近來則**

有學者提出了平行演算法[18,32]。Akl 及 Stojmenovic 所提出的雖可在一簡單的線性陣列上執行[18]，但是其陣列中每一處理器(PE)所須執行的工作仍嫌複雜(線性陣列須決定一分界點，在產生下一全括號串時，分界點左方的 PEs 所產生的數值保持不變，而分界點右方的 PEs 所產生的數值則須以複雜的計數方式算出其所產生的新數值)；至於最近 1998 年 Vajnovszki 及 Pallo 所提出的平行演算法雖簡單很多[32]，但是此演算法須在較複雜且不實際的計算模式 CREW PRAM (Concurrent-Read Exclusive-Write Parallel Random Access Machine)上執行。

上述平行演算法，雖然皆為最低計算成本(cost-optimal)設計，但是仍嫌複雜不夠理想，應該還有改進的空間。我們希望能採用設計韻律陣列或演算法(systolic array and systolic algorithm)的一些設計理念設計出較簡單的平行演算法以解此問題。韻律陣列最早由孔祥重博士所提出[33-34]，其後有很多學者提出了設計韻律陣列的一些觀念如 semisystolic array and systolization[35-36]、space-time mapping[36-38]等。韻律陣列的設計觀念曾經被應用到設計產生某些基本組合體(Combinatorial Object)的平行演算法上，且得到不錯的成果[25, 27, 31]。我們希望這些設計觀念亦可應用到產生全括號串的問題上，進而研究設計出更簡單、更適合製作成超大型積體電路的平行演算法。

首先，我們研究全括號串各種代表方法，再由多種全括號串代表方法中選擇較易由韻律陣列產生的代表方法；接著，研究設計韻律陣列以產生此種代表方法下之所有全括號串；然後，在此平行演算法設計好後，我們撰寫程式以模擬和驗証其正確性；最後，評估此韻律演算法的效能(含使用的處理器數、產生所有全括號串的時間)，並與現有的平行演算法互相比較。

三、結果與討論

3.1 全括號串之代表方式

代表全括號串之方式有很多種，舉例如下(以全括號串()((()))(())為例)：

(1). Bitstring Sequence[32]: 101110001100；此為最簡單的之代表方式，其以1代表(及0代表)。

(2). Weight Sequence[32]: 112312；每一數字代表全括號子串中之第幾個)。

(3). z-Sequence[18]: 13459,10；每一數字代表(在全括號串中之位置。

(4). Left-Distance Method[39]: 011122；首先我們以一 binary tree 代表全括號串，再以 inoder 之次序標示各節點後，則每一數字代表各節點左旋到根節點所須的次數。

(5) Tree Permutation Sequence[40]: 021010；同(4)方法標示各節點後，透過 preorder traversal 可得一序列，此序列之 inversion table 即為所求。

(6) Zerling's Code[13]: 011012；類似(4)採用左旋節點之方法，只是左旋之節點的次序不同。

在上述各代表方式中，明顯的，第一種 bitstring 代表方法為最簡單且最易理解。基於此，我們研究設計韻律陣列以產生代表所有全括號串之 bitstrings。如 $2n$ 為每一 bitstring 之長度則總共有 $B_n = \frac{(2n)!}{n!(n+1)!}$ 個 bitstrings $b_{2n}b_{2n-1}\ldots b_1$。我們產生 bitstrings 的次序為由小到大，即第一個產生的 bitstring 為 $(10)^n$，而最後一個產生的 bitstring;則為 1^n0^n。

3.2 韻律演算法

本節討論產生所有全括號串 bitstrings 之韻律演算法。

首先，我們研究設計出相依圖(dependence graph)如 Figure 1 及 Figure 2 所示[36]。各節點的函數如 Figure 3 所示，其中 i 為節點所在的 column(最

右設為第 1 column)，λ為 null 符號。節點函數除了 Figure 3 所示外，還包含下述之 bitstring data：

if $v = {}^{a}$ then $d' = d$;

if $v \neq {}^{a}$ then

$d' = d$;

if $(2z+3 \le i \le v+z+2)$

or $(2z+2 \ge i \ge 1$ and $i =$ odd)

then $d' = 0$;

if $(v+z+3 = i)$

or $(2z+2 \ge i \ge 1$ and $i =$ even)

then $d' = 1$;

有了相依圖後，我們可採用下述時空映射矩陣(space-time mapping matrix)，將其映射成韻律陣列(systolic array)[36]：

$$T = \begin{bmatrix} 3 & 1 \\ 0 & 1 \end{bmatrix}$$

Figure 4 為最後所設計出的韻律陣列。

3.3 正確性之驗証及效能

最後我們撰寫一程式以模擬和驗証所研製之韻律演算法之正確性。我們在最左處理器的(v, z)依次輸入(n+1, n-1) ((λ, λ) (λ, λ) (λ, −1))*；在其它輸入端點輸入λ*；且在時間 0 時將所有暫存器全設定為λ，則模擬程式產生如 Figure 5 之輸出。所研製的韻律陣列含有有 2n 個處理器；每三個簡單的計算步驟可產生一全括號串；在其上執行的韻律演算法為最低計算花費之平行演算法。和 Akl and Stojmenovic[18]及 Vajnovszki and Pallo 所提出者[32]相比，我們所設計的平行演算法為韻律演算法，所有處理器皆執行簡單且相同的工作，韻律陣列具有可擴充性，極適合製作成超大型積體電路。

四、計劃成果自評

本計劃之研究成果已達預期目標。

五、參考文獻

[1] Reingold, E. M. and Nievergelt, J. and Deo, N. 1977. *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall. Englewood, Cliffs, N. J..

[2] Nijenhuis, A. and Wilf, H. S. 1975. *Combinatorial Algorithms.* Academic Press. New York.

[3] Sedgewick, R. 1977. Permutation Generation Methods. *Computing Surreys*, 19(2), pages 137-164.

[4] Ord-Smith, R. J. 1970. Generation of Permutation Sequence: Part 1. *The Computer Journal*, 13(3), pages 152-155.

[5] Ord-Smith, R. J. 1971. Generation of Permutation Sequence: Part 2. *The Computer Journal*, 14(2), pages 136-139.

[6] Schrak, G. F. and Shimrat, M. 1962. Permutation in Lexicographic Order (Algorithm 102). *Communications of the ACM*, 5(6), pages 346.

[7] Johnson, S. M. 1963. Generation of Permutations by Adjacent Transposition. *Mathematics of Computations*, 17(83), pages 282-285.

[8] Semba, I. 1984. An efficient algorithm for generating all k-subsets ($n \ge m \ge k \ge 1$)of the set {1, 2, ..., n} in lexicographic order. *Journal of Algorithms*, 5, pages 281-283.

[9] Stojmenovic, I. and Miyakawa, M. 1988. Applications of subset generating algorithm to base enumeration, knapsack and minimal covering problems. *The Computer Journal*, 31(1), pages 65-70.

[10] Djokic, B., Miyakawa, M., Semba, I., Sekiguchi, S., and Stojmenovic, I. 1989. A fast iterative algorithm for generating set partitions. *The Computer Journal*, 32(3), pages 281-282.

[11] Er, M. C. 1988. Fast algorithm for generating set partitions. *The Computer Journal*, 31(3), pages 283-284.

[12] Semba, I. 1984. An efficient algorithm for generating all partitions of the set {1, ..., n}. *Journal of Information Processing*, 7, pages 41-42.

[13] Zerling D. 1985 (July). Generating Binary Trees Using Rotations. *Journal of the ACM*, 32(3), pages 694-701.

[14] Er, M.C. 1983. A Note on Generating Well-formed Parenthesis Strings Lexicographically. *The Computer Journal*, 26(3), pages 205-207.

[15] Zaks, S. 1980. Lexicographic Generation of Ordered Trees. *Theoretical Computer Science*, 10, pages 63-82.

[16] Pallo, J. M. 1986. Enumerating, Ranking, and Unranking Binary Trees. *The Computer Journal*, 29, pages 171-175.

[17] Ruskey, F. and Proskruowski, A. 1990. Generating Binary Trees by Transpositions. *Journal of Algorithms*, 11, pages 68-84.

[18] Akl, S. G. and Stojmenovic, I. 1996. Generating Combinatorial Objects on a Linear Array of Processors, from *Parallel*

Computing: Paradigms and Applications., ed. Zomaya, A. Y. International Thomson Computer Press. Pages 639-670.

[19] Akl, S. G. 1987. Adaptive and Optimal Parallel Algorithms for Enumerating Permutations and Combinations. *The Computer Journal*, 30(5), pages 433-436.

[20] Chen, G. H. and Chern M. S. 1986. Parallel Generation of Permutations and Combinations. *BIT*, 26(3), pages 277-283.

[21] Lin, C. J. 1990. Parallel Generation of Permutations on Systolic Arrays. *Parallel Computing*, 15(3), pages 267-276.

[22] Wu, B. Y. and Tang, C. Y. 1990. An Optimal Parallel Algorithm for Generating Permutations on Linear Array, from *Proc. First Workshop on Parallel Processing.* Hsinchu, Taiwan, Republic of China. Pages 106-110.

[23] Tsay, J. C. and Lee, W. P. 1994. An Optimal Parallel Algorithm for Generating Permutations in Minimum Change Order. *Parallel Computing*, 20, pages 353-361.

[24] Akl, S. G. 1992. A Simple Systolic Algorithm for Generating Permutation. *Parallel Processing Letters*, 2(2), pages 231-239.

[25] Lee, W. P. and Tsay, J. C. 1994. A Systolic Design for Generating Permutations inLexicographical Order. *Parallel Computing*, 20, pages 775-785.

[26] Chan, B. and Akl, S. G. 1986. Generating Combinations in Parallel. *BIT*, 26, pages 277-283.

[27] Tsay, J. C. and Lin, C. J. 1990. A systolic design for generating combinations in lexicographic order. *Parallel Computing*, 13, pages 119-125.

[28] Lin, C. J. and Tsay, J. C. 1989. A Systolic Generation of Combinations. *BIT*, 29, pages 23-36.

[29] Lin, C. J. 1989. A Parallel Algorithm for Generating Combination. *Computers and Mathematics with Applications*, 17(12), pages 1523-1533.

[30] Djokic, B, Miyakawa, M., Sekiguchi, S., Semba, I., and Stojmenovic, I. 1990. Parallel Algorithms for Generating Subsets and Sets Partitions, from *Lecture Notes in Computer Science, Vol. 450: SIGAL'90 Algorithms.*, ed. Asano et. al., T. Springer-Verlag. Pages 76-85.

[31] Tsay, J. C. and Lee, W. P. 1994. A Cost-Optimal Systolic Algorithm for Generating Subsets. *International Journal of Computer Mathematics*, 50, pages 1-10.

[32] Vajnovszki, V. and Pallo, J. 1998. Parallel Algorithms for Listing Well-Formed Parentheses Strings. *Parallel Processing Letters*, 8(1), pages 19-28.

[33] Kung, H. T. and Leiserson, C. E. 1979. Systolic arrays for VLSI, from *Proc. 1978 Society for Industrial and Applied Mathematics.* Pages 256-282.

[34] Kung, H. T. 1982. Why systolic architectures? *Computer*, 15(1), pages 37-46.

[35] Leiserson, C. E. and Saxe, J. B. 1983. Optimizing synchronous systems. *J. of VLSI and Computer Systems*, 1(1), pages 41-67.

[36] Kung, S. Y. 1988. *VLSI Array Processor.* Prentice-Hall Int. , Englewood Cliffs, NJ.

[37] Moldovan, D. I. 1983 (Jan). On the design of algorithms for VLSI arrays. *Proc. of the IEEE*, 71(1), pages 113-120.

[38] Moldovan, D. I. 1982 (Nov). On the analysis and synthesis of VLSI algorithms. *IEEE Trans. Comput.*, C-31(11), pages 1121-1126.

[39] Makinan, E. 1987. Left distance binary tree representations. *BIT*, 27, pages 163-169.

[40] Knott, G. D. 1977(Feb). A numbering system for binary trees. *Communications of the ACM*, 20(2), pages 113-115.

da	*d*	*db*	*ua*	*yb*	*pa*	*z*	*i*	*v'*	*z'*	*yb'*	*pa'*	*ua'*
1	1	0	$\neq a$	a	a					0	$2\times ua$	
1	0		$\neq a$	a	a					0	$2\times ua+1$	
1	1			$\neq a$	>0							
0	1	1			>0 and even