울산대학교 연구논문집 제18권(자연과학·공학편) 제2호 pp. 119~126. 1987
UOU Report Vol.18. (Natural Science and Engineering) No.2. pp.119~126. 1987
University of of Ulsan

Data Flow개념하에서 AND/OR Parallelism실행에 관한 연구

옥 철 영
전자계산학과
(1987. 4. 30 접수)

<要 約>

논리 프로그램에서의 AND/OR Parallelism을 Data Flow System에서 실행시키는 방법은 [10, 12] 등에서 제안되었다.

이 논문에서는 [10]에서 제안한 모형을 근간으로 하여, 논리 프로그램에서의 assertion이 data flow그래프로 해석되는 방법을 보였고, 그래프 template를 Euler Path가 아닌 Spanning Tree로 변형함으로써 AND Parallelism을 향상시킬 수 있음을 보였으며, 이러한 template 토큰들이 data flow 그래프의 각 노드로 전달되어 처리되는 과정을 제시하였다.

A Study on Execution of AND/OR Parallelism based on Data Flow Concept

Ock, Cheol-Young
Dept.of Computer Science
(Received April 30,1987)

<Abstract>

The execution models of AND/OR Parallelism in logic program are proposed in [10,12] that based on data flow concept

In this paper, based on the model that is proposed in [10], the method with that assertions in logic program are translated to data flow graphs is represented, and AND Parallelism can be increased by transforming graph templates into Spanning Tree not Euler Path, and a procedure for execution of template tokens is represented by propagating these tokens into nodes of data flow graph.

Ⅰ. 序 論

최근의 새로운 기술의 발전은 빠른 속도의 계산을 가능하게 하였으며, 또한 여러 문장을 동시에 수행하는 병행처리 방법을 가능하게 하였다.

병행처리 방법은 크게 Von Neumann방식을 근간으로 한, 다중처리 방법과 non von Neumann 방식

의 새로운 컴퓨터 구조의 연구로 나뉘어 발전되어 왔다. 그 중 다중처리 방법은 계산의 분산 처리 및 수행의 동기화 등의 어려움 때문에 제한적으로 응용 발전해 왔다. [1, 3]

Data Flow System은 비동기적으로 데이타를 전달하는 non von Neumann방식의 새로운 컴퓨터 구조로 데이타를 값 토큰으로 취급하여 임의의 명령어에 필요한 데이타 토큰이 모두 이용 가능한, 모든 명령어들을 동시에 수행함으로, 명령어(instruction)단계에서의 동시성을 구현하기에 적합하다. [14]

Data flow언어로는 side-effect가 없고, 프로그램의 명령어들 사이에 데이타 종속성을 줄일 수 있는 functional 언어가 다수 개발되었고[7, 15, 16], 이들은 컴파일러에 의해 data flow 그래프로 바뀌어 data flow system에서 수행된다. [4, 5, 6]

논리 프로그램은 high-level non-procedure언어로, 연역 추론을 필요로 하는 문제를 표현하기에 적합한 수학적인 형식 언어로[17], 병행처리용의 언어로 개발되지는 않았으나 논리가 병행처리의 interpreter를 가지고 있어, data flow system에 적용될 수 있다. [18]

논리 프로그램을 data flow개념하에서 실행시키고자 하는 방법으로, 논리 프로그램을 funtional 프로그램으로 변형하는 연구[13]와, 논리 프로그램을 data flow system에서 실행시킬 수 있는 모형을 제시한 연구[10, 11, 12]들이 진행되어 왔다. [12]에서 제안한 방법은 Eager and Lazy Evaluaton Machanism을 사용하여 제한된 자원환경에서, OR-Parallel과 AND-Pipeline처리하여 논리 프로그램을 동시에 수행하게 하였다.

[10, 11]에서는 논리 프로그램의 실행, 즉 Resolution을 패턴 일치의 과정으로 보고, 주어진 assertion 그래프 상에 그래프 template의 토큰을 비동기적으로 전달하여, 원하는 절(clause)과 일치되는 패턴을 찾는 데이타 전달 방식과 그에 필요한 모형을 제안하였다. 이 모형에서는 간단한 방법으로, 논리 프로그램을 2원 서술 논리에 한하여 제한하였고, 그래프template를 Euler Path 알고리듬을 사용하여, 선형 순서화된 토큰들을 생성하여 여러 노드로 분산시켜 AND/OR동시성을 구현하였다.

이 논문에서는 [10, 11]에서 제안된 모형을 근간으로, 보다 복잡한 그래프 template를 Euler Path가 아닌 Spanning Tree로 변형하여 AND 동시성을 보다 향상시켰으며, 논리 프로그램에서의 여러 형태의 절이 data flow 그래프의 노드에서 비동기적으로 전달되어 처리되는 절차를 제시하였다.

2. 논리 프로그램의 Data Flow 그래프로의 해석

논리 프로그램은 다음과 같은 절로 구성된다. [9, 17]

$P_0 \leftarrow P_1, P_2, \cdots\cdots, P_n$

여기서 각 P는 $P(t_1, t_2\cdots\cdots, t_m)$형태의 literal로 P는 서술문자이고, t i 는 항(term)으로 상수, 변수, 혹은 함수 값 중의 하나이다. P는 head이며, $P_1, P_2\cdots\cdots, P_n$은 절의 body로서 body가 없는 절은 assertion이라 하며 사실(fact)을 나타낸다. head와 body를 가지는 절은 추론규칙을 나타내며 내부적으로 주어진다. 또 head가 없는 절은 목표절(goal)로 목표절의 각 literal은 부목표절이 되며, 다른 절의 head와 단일화(unify)됨으로써 해결(resolve)되며, 여러 비교 흡수(resolution)방법들이 있다. [19]

그림 1은 각 개인간에 father와 mother의 관계를 나타낸, 예제 논리 프로그램이다.

(1) father (Bill, John) ←
(2) father (John, Hans) ←
(3) father (Jane, Fred) ←
(4) mother, (Bill, Jane) ←
(5) mother (John, Ann) ←

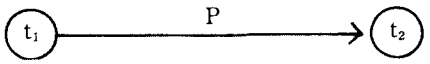
(6) parent (x, y) ← mother (x, y)
(7) parent (x, y) ← father (x, y)
(8) grandparent (x, z) ← parent (x, y), parent (y, z)

(9) ← grandparent (Bill, z)

그림 1. 예제 논리프로그램

그림1에서 (1)—(5)절은 각 개인간의 mother와 father의 관계를 나타낸 assertion이고, (6)—(8)절은 father와 mother의 관계가 parent와 grandparent로 추론되는 규칙을 나타내며, (9)절은 해결되어야 할 목표절을 나타낸다.

위의 2원 서술 논리 프로그램에서의 literal $P(t_1, t_2)$를 다음과 같이 방향성 그래프로 나타낼 수 있다.



이 방향성 그래프에서 arc의 head는 literal에서 항이 주어진 순서를 나타내며, 항은 노드로 서술문자는 arc와 같은 뜻으로 사용된다. 여러 literal들이 같은 항을 공유하는 경우는 [예, P(x, y), P(y, z)], 다음과 같이 해당 arc들을 통해 관련된 서로의 노드들이 연결된다.

x ····· y ····· z

[10]에서는 논리 프로그램의 assertion과 다른 절들을 다음 두가지의 그래프로 구분하였다. Assertion 그래프는 사실들을 나타낸 assertion절로부터 구성되며, 예제 프로그램의 assertion그래프는 그림 2와 같다.

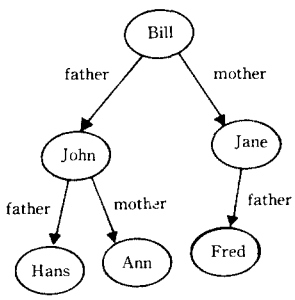


그림2. Assertion그래프

Assertion 그래프의 각 노드는, 그래프 arc를 따라 비동기적으로 전달되는 template 토큰을 전달받아, 토큰 일치 과정을 거쳐 새로운 토큰을 생성하여 다음 노드로 전달할 수 있는 active element로, 각 processor element 에 할당되는 data flow 그래프로 간주할 수 있다

목표절은 AND/OR트리의 변형으로[2], 목표절의 body와 body의 각 literal로 단일화될 수 있는 head를 가진 다른 절이, 포인트를 통해 서로 연결되어 있고, 이러한 그래프의 집합을 goal structure라 하며 그림 3과 같다.

Goal structure에서 literal L은, assertion그래프에서 assertion 중의 하나와 단일화되거나, L에 의

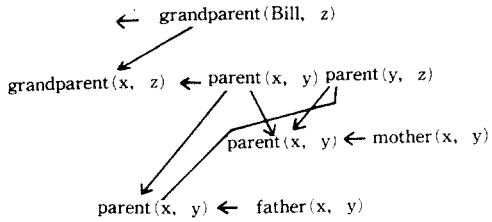


그림3. Goal Structure

해 포인터되는 부목표절을 해결함으로써 해결된다. Goal structure에서 각 절의 body는 assertion 그래프와 마찬가지로, 각 항이 arc에 의해 서로 연결되어 있는 그래프와 같으나, 이때 각 절은 변수를 가지고 있으므로 그래프 template라 하며, 그림1의 예제 논리 프로그램의 초기 목표와 부목표의 그래프 template는 그림4의 (a), (b)와 같다.

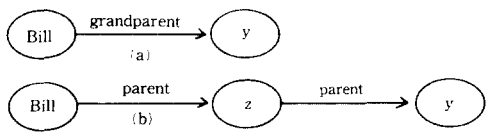


그림4. 그래프 template

그림2의 assertion그래프가 data flow그래프로 해석되기 위해서는, 각 노드와 arc 및 arc를 통해 전달되는 토큰은 다음과 같은 기능을 가진다. Arc는 순수한 data flow개념에서의 단방향이 아니라, 양방향의 path이다. 이는 arc을 통해 전달된 template 토큰의 일치 여부와 초기 목표절의 해결 여부를 알려줄 수 있는 역 path 가 필요하기 때문이다.(4절에서 보충설명)

토큰에는 그림3과 같이 다른 절로 추론될 수 있는 규칙이 저장된 위치를 가리키는, 포인터를 포함한 그래프 template 를 저장하고 있어, assertion 그래프의 해당 arc에 의해 전달되어 각 노드에서 변수의 단일화 및 그래프 일치 과정이 일어난다. 이러한 토큰은 토큰 coloring 기법[16]을 이용해서 다른 그래프 template에 속한 토큰들을 구분할 수 있고, I-structure memory[8] 을 이용하여 goal structure

를 표현할 수 있어, 포인터를 통하여 효율적으로 공유할 수 있다.

Assertion 그래프의 각 노드는 이러한 template토큰을 비동기적으로 처리하며, 그래프의 arc를 통해 다른 노드들과 commumication 할 수 있어야 한다. 또한, 논리 프로그램에서의 AND/OR 동시성을 최대한으로 구현하기 위해서, 각 노드는 순수한 data flow system 에서의 산술, 논리 연산 기능보다도 더 상위의 기능을 가지고 있어, 다음과 같은 일을 수행하게 해야 한다.

—단일화(Unification)
—AND/OR 분산처리
—토큰 생성 및 복사
—error 처리
—분산 처리된 토큰 결과의 수집

이러한 노드의 기능은 좀 더 구체적으로 정의되어야 하며, 앞으로의 연구 과제 중의 한부분이다.

3. Spanning Tree를 이용한 AND—동시성 향상

논리 프로그램의 번역 과정에서 얻어진 그래프를 이용한 논리 프로그램의 실행은, 초기 목표를 그래프 template 로 해석하여, 그래프 template 가 assertion 그래프의 일부와 일치 가능한 변수의 binding을 결정하는 비교 흡수하는 과정으로, 그래프 일치 과정과 관련된다. 이 과정에서 다른 그래프 template G를 가리키는 assertion 그래프와 일치되면 된다. 예를 들면, 목표절 grandparent(Bill,z) 가 그림2의 assertion 그래프에서 실행되는 과정은, 초기 목표를 노드 'Bill'과 'z'를 연결하는 grandparent arc 를 가진 그래프 template로 보고, 이 template 가 토큰으로 생성되어 노드 'Bill'에서 grandparent 의 arc로 전달되는 과정과, grandparent 의 포인터에 의해 지정되는 다른 절[grandparent(x,z)← parent(x,y), parent(y,z)] 에서 변수 x가 'Bill'로 binding 되어, 새로운 그래프 template : parent(Bill,y), parent(y,z) 가 토큰으로 생성되어 노드 'Bill'에서 parent의 arc로 전달되는 과정이 동시에 발생한다. 이런 두 토큰의 일치과정은 실패이나, parent(Bill y) parent(y,z) 의 새로운 그래프 template에 의해 다시 father(Bill,y)와 mother(Bill,y)의 새로운 template 토큰이 생성되어, 위의 비교일치 과정이 반복되고, 결국 변수 z에 'Hans','Ann' 혹은 'Fred' 중의 하나가 binding되어 해결된다.

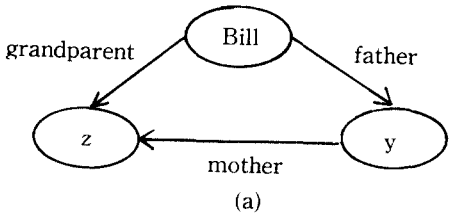
이러한 과정에서 같은 literal의 포인터에 의해 지정되는 모든 절의 그래프 template[예, mother(Bill, y), father(Bill,y)] 의 해결은, OR—동시성으로 각기 다른 토큰으로 생성되어 'John' 'Jane' 노드로 전달되어, 각각 독립적으로 수행되어 질 수 있고, parent(Bill,y), parent(y,z) 의 그래프 template같이 literal간에 공유된 변수가 있을 때는, 두 literal 의 변수 y가 같은 상수로 binding 되어야 하기 때문에 각각 동시에 수행될 수 없어 하나의 토큰으로 생성되어 해결되어져야 한다. 이와 같이, 여러 절이 하나의 변수를 공유하도록 묶여진 template 를 cluster 라 하며, cluster 간에는 공유된 변수가 없으므로, 각 cluster 는 독립적으로 assertion 그래프와 일치시킬 수 있고, 이것은 논리 프로그램 상의 AND—동시성의 실현 과정이다.

Cluster의 일치 과정은 assertion 그래프를 통해 토큰을 전달하므로써 수행되어지며, 주어진 cluster 내의 template는 임의로 연결되어진, cycle 을 포함할 수 있는 그래프이다. Data flow system은 토큰을 비동기적으로 전달하고, 중앙 집중된 control기능이 없기때문에 cycle 을 찾아내기가 쉽지 않다. 그러므로, 보다 간단한 토큰 전달 절차를 위해, 주어진 template을 다른 형태로 변형시켜야 한다.

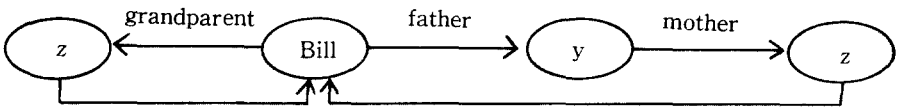
[10]에서는 template를 Euler Path 의 선형 순서화된 chain으로 변형하여, 그 path를 따라 각 노드가 토큰을 전달시킴으로써, cluster 내의 literal과 변수를 binding시켰다. 그러나, 이러한 변형은 Euler Path를 구하기 위해 literal의 중복이 필요하고, 결국 필요없는 단일화 과정이 반복 수행되며, 또한 cluster 내에서 AND—동시성의 실행을 방해한다.

그러나 template의 해결과정을, literal의 일치과정으로 해석한 Euler Path가 아닌, 각 노드에서의 변수의 binding과정으로 해석한 Spanning Tree로 변형시킴으로써, 앞서의 문제점들을 어느 정도 해결하며, AND—동시성을 향상시킬 수 있다. 예를 들어, 다음과 같은 질문에 답하는 문제를 생각해 보자. 'Bill의 아버지의 어머니인 조상은 누구인가?' 이 질문에 대한 논리 프로그램은 '←

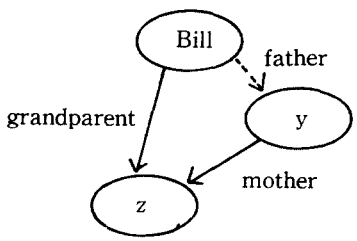
grandparent(Bill,z), father(Bill,y), mother(y,z)'이 며
이때의 그래프 template는 그림 5의 (a)와 같다.



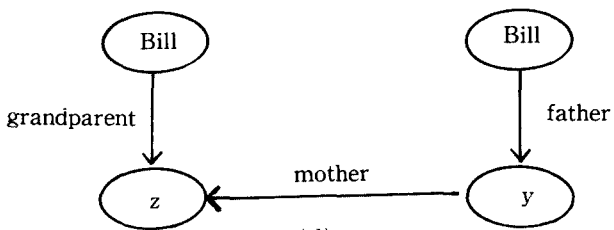
(a)



(b)



(c)



(d)

그림5.

그림 5. (a)의 그래프 template을 Euler Path로 바꾸면 그림5. (b)와 같고, 이것이 하나의 토큰으로 생성되기 때문에, AND—동시성을 구현하기가 쉽지않다. 그러나 그림5. (a)를 그림 5. (c)와 같이, Spanning Tree 를 구하여 점선의 back edge를 그림 5. (d)와 같이 복사시킴으로써, 서로 다른 두개의 template : '←grandparent(Bill,z)'와 '←father(Bill, y), mother(y,z)' 를 얻을 수 있고, 이들 두 template 는 OR—동시성으로 동시에 처리되어질 수 있다.

결국, Euler Path 로 변형됨으로써 제한되었던 AND—동시성이, 어느 정도 완화되어 OR—동시성의 다른 독립된 template로 번역하여 처리될 수 있으므로, AND—동시성을 향상시킬 수 있다.

4. 토큰 전달 방식을 이용한 목표절 해결 과정

목표절이 해결되는 과정은 논리 프로그램의 목표절에 따라서, 다음 세가지의 종류의 template 토큰이 생성되며, 각 template 토큰이 data flow 그래프의 노드로 전달되어 처리되는 과정은 다음과 같다.

(1) 다른 template 로 포인터를 갖지 않는 목표절

이는 논리 프로그램의 assertion 절과 곧 바로 일치 과정이 일어날 수 있는 template 로서 $t_1 \rightarrow t_2$의 형태로 template간에는 공유된 변수가 없으므로 이러한 template 들은 OR-동시성을 가지며, data flow 그래프의 여러 노드들에서 동시에 binding과정이 일어난다. 이러한 binding 과정은 template 가 토큰으로 생성되어, 항 t_1, t_2와 일치되는 노드로 투입되고, 그 노드에서 template 토큰은, 토큰의 서술문자와 일치되는 arc 을 따라서, 다른 노드로 전달되어 진다. 이 경우 다음 세가지 유형으로 구분되어 질 수 있다.

i) 항 t_1, t_2가 이미 상수항 T_1, T_2로 binding 되어 있는 경우.

Data flow 그래프 상에는 상수 항T_1, T_2에 해당되는 노드는 하나만 존재함으로, 둘 중의 어느 한 노드로 template토큰이 투입되고, 그 노드는, label이 P인 모든 arc를 따라 토큰을 복사하여 연결된 다른 노드로 전달하고, 복사된 노드를 전달받은 노드 중 어느 하나의 노드가 두번째 상수 항 T_2와 일치하면, template $T_1 \rightarrow T_2$는 해결되고 그렇지 않은 경우 template 토큰과 직접 일치되는 data flow 그래프는 없고, 다른 template 의 포인터도 갖지 않으므로 실패이다.

ii) 항 t_1, t_2 중 어느 하나만이 상수 T_1, T_2로 binding 된 경우.

template 토큰은 이미 binding 된 노드로 투입되어 복사되어져, label P 와 일치한 arc 를 따라 전달되고 토큰을 전달받은 노드에서는 아직 binding 되지 않은 변수를 해당 노드의 상수 항으로 binding 하며, 이런 과정을 단일화 과정이라 한다. 단일화 과정은 label 이 P인 arc 의 갯수만큼의 노드에서 동시에 발생하게 된다. 이때 data flow 그래프가 양방향을 가지므로, 항 t_1, t_2중 binding 된 어느 하나의 노드에서 시작하여도 무관하다.

iii) 항 t_1, t_2 중 어느 것도 상수 항으로 binding 되지 않은 경우.

Template 토큰을 투입할 경우 특정 노드가 결정되지 않았으므로 data flow 그래프의 모든 노드로 토큰을 수사시켜, 각 노드에서 처음 항을 binding하고 ii) 의 과정을 반복한다. 이 경우 모든 노드에서 동시에 binding 과정이 일어나나 중복된 결과를 얻게된다.

(2) 다른 template로의 포인터를 갖지 않은 목표절이 공유된 변수를 가진 경우.

이는 template가 $P_1(t_{11}, t_{12})$, $P_2(t_{21}, t_{22})$, ……, $P_n(t_{n1}, t_{n2})$처럼 연속되어 공유 변수를 가진 경우로 여기서도 다음 세가지 유형을 가진다.

i) $t_{11} = t_{i-1,2}$로 template가 여러 노드를 공유한 경우로 각 부목표절이 동시에 해결될 수 없으므로, 전체 template가 하나의 토큰으로 생성되어야 하는데, 이때 각 부목표절이 임의 순서로 해결될 수 있다면, 앞서 제시한 Spanning Tree 를 preorder 탐색하여 선형 순서화된 형태로 구성하되, [10]에서 제시한 방법처럼, 선형 순서화된 형태에서 가장 왼쪽 항이 상수항으로 binding 된 항이 오도록 하며, 공유된 변수들은 서로 연결시켜 주어 하나의 변수가 binding 되면, 연결된 모든 공유 변수를 같은 상수항으로 binding되도록 한다. 이러한 토큰은 data flow 그래프에서 토큰의 가장 왼쪽 항과 일치되는 노드로 투입되어 각 부목표절에 대해 (1) 의 ii) 와 같은 과정이 다음 조건을 만족할 때까지 진행된다.

—노드 t_{11}에서 label P 로의 arc가 없는 경우는 data flow 그래프에서 해당 path가 없는 경우로 error처리 된다.

—Template 의 마지막 항t_{n2}까지 처리된 경우는, 주어진 data flow 그래프에서 일치되는 path가 발견된 경우로 해답이 찾아진다.

ii) $t_{11} = t_{i-1,1}$로 이는 항 $t_{i-1,1}$이 상수항으로 binding 되면, 그림 5. (d)의 경우처럼 두 개의 서로 독립된 template 토큰이 생성되어 나머지 항들에 대해 (2) 의 i)과 같이 진행된다.

iii) t_{12}이 상수항으로 이미 binding 되어 있는 경우로, (1) 의 iii)에서와 비슷하게 노드 t_{12}가 가진 모든 arc로 토큰을 복사하여 분산 처리 시킨다. 이 때, 분산 처리된 토큰들이 i)의 조건을 만족할 때까지 진행된다.

(3) 다른 template에로의 포인터를 가진 목표절

예를 들면, template가

(1) $P \leftarrow P_1, \cdots\cdots P_{i-1}, P_i, P_{i+1}, \cdots\cdots P_n$

(2) $P_i \leftarrow Q_1, \cdots\cdots Q_m$의 경우로 목표절 P를 해결하기

위해서 초기의 template 토큰이 형성되어 (2) 의 경우처럼 진행되다가 부목표절 P_i를 처리하는 노드 t_{ii}에서 포인터 P_i에 해당하는 부목표절의 template 토큰이 확장되어, 새로운 토큰이 생성되어 각기 다른 path 로 전달되어 동시에 진행된다. 이 때, 노드 t_{ii}에서는 OR-분기가 새로이 발생되고 각 분기로 전달된 토큰 중 어느 하나만이 해결되어도 목표절 P는 해결된다.

위의 세가지 유형의 목표절은 논리 프로그램에서 서로 혼합된 형태로 나타날 수 있고, 이러한 토큰들은 토큰 coloring기법 [16]을 이용하여 구현함으로써, 다른 template 에 속한 토큰들을 구분할 수 있다. 또한, 각 노드 간에 전달되는 토큰들은 template 토큰 외에도 error유무와 단일화된 변수의 상수 항을 되돌려 줄 수 있는 다양한 종류의 토큰들이 필요하다.

5. 結 論

이 논문에서는 논리 프로그램이 data flow system 하에서 실행되기 위해서 논리 프로그램의 assertion 그래프가 data flow 그래프로 해석되는 방법으로 제시하였고 data flow 그래프에서 각 노드가 가져야 할 일반적인 기능을 정의 하였다. 그러나 노드의 기능은 실제로 논리 프로그램이 실행될 data flow system의 구조와 processor element들의 기능들에 따라서 다수 변형될 수 있고, 그 때의 성능 평가가 달라질 수 있다. 또한 processor 간의 network protocol 및 data flow그래프의 노드들을 processor element들로 할당하는 방법 등은 앞으로 더 연구되어져야 할 관점이다.

논문에서는 [10]에서 제안한 방법을 Euler Path가 아닌 Spanning Tree로 변형함으로써, AND-동시성을 향상시킬 수 있음을 보였고, template 토큰들이 각 노드에서 처리되는 과정을 제시하였다. 그러나 포인터를 가진 토큰의 처리 방법 및 다른 template의 토큰을 구분할 수 있는 방법 등은 좀 더 구체적으로 연구되어져야 할 것이다.

參 考 文 獻

1. D. J. Kuck, R. H. Kuhn, B. Leasure and M. Wolfe, "The Structure of an Advanced Vectorize for Pipelined Processor", Proc. of the 4'th Int'l Compt. Software & Aplli-cations Conf., pp. 709—715, Oct. 1980.
2. D. I. Moldovan, Modern Parallel Proce-ssing, Dept. of Electrical Engineering —Systems Univ. of Southern California, 1980.
3. D. J. Kuck, The Structure of Computers and Computations, Vol. 1, John Wiley & Sons, Inc., 1978.
4. A. L. Davis and R. M. Keller, "Data Flow Program Graphs", Computer, Vol. 15, No. 2, pp. 15—25, Feb. 1982.
5. W. B. Ackermann, "Data Flow Langua-ges", Computer, Vol. 15, No. 2, pp. 15—25, Feb. 1982.
6. J. B. Dennis, "First Version of a Data Flow Procedure Language", Lecture Notes in Computer Science, Vol. 19, N. Y., Springer—Verlag, pp. 362—376, 1974.
7. W. B. Ackerman and J. B. Dennis, "VAL —A Value—oriented Algorithmic Language", Preliminary Ref. Mannual, TR—218, Lab. for Comp. Science, MIT, Cambridge, Mass., 1979.
8. Irvind and R. E. Thomas, "I—Structure : An Effient Data Type for Funtional Lang-uages", Tech. Rep. TM—178, Lab. for Comp. Science, MIT, Cambridge, Mass., Sept. 1980.
9. A. Deliyanni and R. A. Kowalski, "Logic and Semantic Networks", CACM, Vol. 22, No. 3, pp. 184—192, Mar, 1979.
10. H. L. Bic, "Execution of Logic Programs on a Dataflow Architecture", the 11'th Ann. Int'l, Symp. on COPUTER ARCHITECT-URE, pp. 290—296, June 1984.
11. H. L. Bic, "Data—Driven Logic : A Basic Model", Proc. of the Int'l Workshop on High—Level Computer Architecture, pp. 1. 20—1. 25, May 1984.
12. R. Hasegawa and M. Amamiya, "Parallel Exection of Logic Programs based on Data Flow Concept", Proc. of the Int'l. Conf. on Fifth Gennertion Computer System, pp. 507—516, 1984.

13. U. S. Reddy, "Transformation of Logic Programs into Functional Programs", IEEE Trans. on Computer, pp. 187—196, 1984.
14. T. Agerwals and Arvind "Data Flow Systems", Computer, Vol. 15, No. 2, Feb. 1982.
15. J. Backus, "Can Programming be liberated from the Von—Neuman Style? A Functional Style and its Algebra of Programs", Comm. of ACM, Vol. 21, No. 8, pp. 613—641, Aug. 1978.
16. Avind and K. P Gostellow, "The Id Report : An Asynchronous Language and Computing Machine", Dept. of Computer and Information Science, TR—114, UC Irvine, Calif., Sept. 1978.
17. R. A. Kowalski, "Predicate Logic as a Programming Language", Proc. of IFIP Congress, North—Holland Pub. Company, Amsterdam, pp. 569—574, 1974.
18. T. Yokoi, S. Goto, et al, "Logic Programamming and a Dedicated High—Performance Personal Computer", Proc. of Int' l. Conf. on 5'th Gen. Comp. Systems, pp. 151—156, Oct. 1981.
19. N. Y. Nilsson, Principles of Artificial Intellgence, Springer—Verlag, N. Y., 1982.