論 文
34~8~1

Node Label에 의한 기본적 Data Flow Machine 모델

A Preliminary Architecture for a Data Flow Machine Model with Node Labelling

金 元 燮*·朴 熙 淳**
(Won-Sob Kim · Hi-Soon Park)

Abstract

The first four generations of computers are all based on a single basic design: the Von Neuman Processor, which is sequential and does one operation at a time. [3] Efforts to develop concurrent or parallel computers have been carried on for many years. Data flow approach is significant in these efforts to make high speed parallel machines and expected a great deal of parallelism.

In this paper we propose a preliminary Data Flow Machine Model operating asynchronously on the base of Node Labelling. We introduce a concept of Node Labeling for this purpose which is relevant to the Data Dependency and Parallelism. And we explain how the Node Tokens are fired in the proposed system.

1. 서 론

컴퓨터의 능력을 높이기 위하여 많은 노력들이 시도되고 있는 바 크게 나누어 두가지 방향으로 분류한다면 1) 기존의 Von Neuman 방식 컴퓨터에 대한 모든 부분에서의 성능 향상을 위한 노력과[1], 2) 이들 방식의 기본적 구조와 실행 방법을 탈피하여 더욱 큰 성능을 기대해 보려는 노력들이다.[2), 3)] 이들중 2)의 노력들은 다시 1) Data Flow Machine (이후 DFM) 과 2) Demand Driven Machine 으로 대분되며 이중 DFM에 관한 관심은 점점 높아져 가고 있다.

1970년을 전후하여 시작된 DFM 연구는 MIT, UTAH대학등에서 몇몇 사람들에 의해 지속되어 오다가 1977년 이후부터 많은 연구 논문들이 발표되기 시작되었다.

현재까지 제안된 시스템들을 정리하여 보면

1). MIT Machine[4]
2). DDM1[5]
3). LAU[6]
4). Rumbaugh's machine[7]
5). Texas Instruments[2]
6). Manchester machine[8]
7). Japan[14]

등으로 구분될 수 있으며 데이타의 유무에 의하여 비동기적으로 실행되고 Ring Type 실행 Cycle 을 갖는 유사점이 있다. 한편 Local Memory 의 유무[7] 확장성[8] structured 데이타의 처리[7] Color Code[9), 14)]의유무, subprogram[7), 18)]의 처리 능력 등에서 차이점을 발견할 수 있다.

본 논문은 Davis 의 DFM으로서의 요구 사항[10]에 될 수록 만족되게 하고 Dennis 모델과 Manchester 모델에 근거하며

1). Arbitor (Dennis) 등 Routing Network 를 단순화시키고
2). Local Memory 의 삽입
3). Sequential 제어 능력을 높임
4). 구조적 Hierarchy 의 단순화
5). Data Flow Graph (이후DFG) 와의 상응성

등에 주안하여 전체적 시스템 구성에 접근한다.

이를 위해 기존의 Von Neuman 방식에서의 Program Counter 와 기능이 유사한 Sequential Dependency Level Counter (LC) 를 설치하여 데이타

*正 會 員 : 全北大 工大 電氣工學科 敎授 · 工博
**正 會 員 : 圓光大 工大 電算工學科 助敎授
接受日字 : 1985年 4月 4日

Dependency에 의한 Sequential 관계 영역을 처리케 하므로써 시간적 제어 능력을 높이고 Processor Allocation에 있어서는 Node Token 자체에 할당된 Processor 위치를 미리 coding 하므로서 Arbitor 의 Transition 및 Contention 에 의한 시간지연을 제거토록 하였다.

우선 Sequential 및 Parallel 의미를 갖는 Node Label(이후NL)을 DFG 의 모든 Node에 부여하고 이를 사용하여 Node Token을 작성하며 구성된 시스템에 적용시켜 본다.

본론의 2.1에서 NL을 붙이는 방법과 그들의 관계를 고찰하고 2.2에서 이들 NL로 표시한 Node Token을 작성하며 2.3에서 시스템의 구조와 샘플 프로그램의 실행 과정을 설명한다. 이후 본 시스템에 대한 검토와 연구 방향등을 제시한다.

2. 본 론

2.1 Node Label(NL)

DFG[11]는 Node(또는 actuator, operator) 상호간의 직병렬 관련성 내지는 데이타의 흐름 및 그 의존성을 나타내고 산술 및 논리 연산 과정을 단순하게 표시하는데 유효하다. NL은 이 DFG상에서 해당 Node의 Sequential 순번(Si)과 Parallel 순번(Pi)을 나타내는 2차원 첨자로서

$$NL := Si, Pj$$

$$Si = 1, 2, 3, \cdots\cdots N$$

$$Pj = 1, 2, 3, \cdots\cdots M$$

로 표시한다.

Sequential 순번 Si는 데이타 의존 관계에 있어서 Dependency Depth 또는 Dependency Level을 나타내고 Parallel 순번 Pj는 동시처리 가능한 Node들중 Parallel Depth 또는 Parallel Level을 표시한다. 여기서 Dependency Level(이후DL) Si가 순서적인 의미를 갖는 것은 당연하나 Parallel Level(이후 PL) Pj가 순서적인 서열을 갖게 한 것은 별 의미 없고 단지 Concurrency를 갖는 Node들 사이에 구분을 위한 일련 번호일 뿐이며 나중에 설명되는 Processor 할당과 관련을 갖는다.

예로서 다음과 같은 일반적인 프로그램을 선택하자. 이는 Fast Fourier Transform의 Butterfly 형 계산[12]에 관한 것이다.

즉, $A' = A + C\cos\alpha + D\sin\alpha$

$B' = B - C\sin\alpha + D\cos\alpha$

$C' = A - C\cos\alpha - D\sin\alpha$

$D' = B - D\cos\alpha + C\sin\alpha$

위의 계산은 5개의 입력 변수와 4개의 출력 변수를 갖고 그림 1과 같은 DFG로 표시된다. 여기서 Dennis의 T-Gate, F-Gate, Decider (또는 Comparator) 및 Merger(또는 Selecter) 등은 포함되어 있지 않고 Node의 출력수는 System에 따라 다르지만 Copy 의 출력수를 2개, 다른 Node들의 출력수는 1개로 제한한다. 물론 Node 출력수를 많이 장치하면 처리 속도는 빨라지나 Code가 길어지고 hardware가 복잡하여져 어느쪽을 택할 것인가는 trade off 사항이다.

그림 1에서 각 Node의 오른쪽 첨자가 NL을 표시하고 Si는 Y축의 DL Number를 가르키며 Pj는 X축의 PL Number를 표시한다. 즉 DL 순번은 맨 윗 Node로부터 아래쪽으로 쓴 일련 번호이고 PL순번은 같은 DL 값을 갖는 Node에 왼쪽에서 오른쪽으로 쓴 일련 번호이다.

따라서

$$f(Si, Pj)$$

즉 $f(NL)$

은 function이 무엇이고 데이타의존도상 몇 번째 서열이며 Parallel Node 중 어디에 위치하는가를 표시한다. 그러므로 임의의 DFG 상에서 Si의 최대값 S

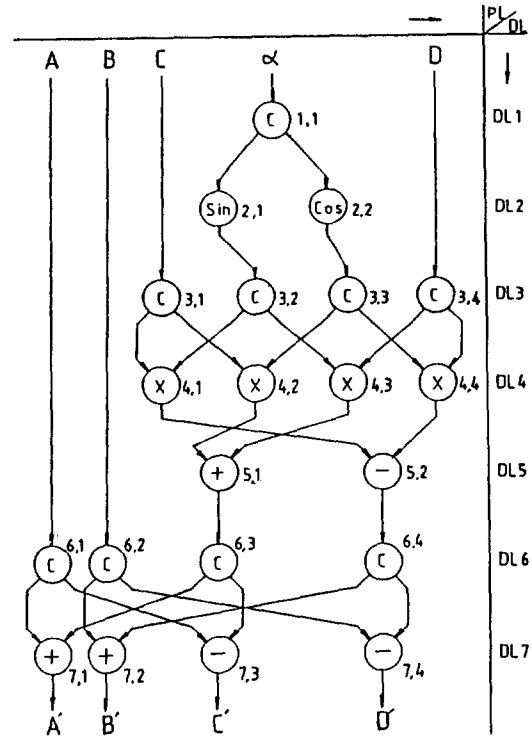

그림 1. 예제 프로그램의 데이타 흐름도

Fig. 1. Data flow graph of the example program.

max은해당 프로그램의 최대 Sequential Dependency 수를 표시하고 Pj 의 최대값 Pmax 은 최대 몇개의 Parallelism 을 갖는가를 나타낸다. 따라서 위의 예제 프로그램은 7개의 Sequential Cycle 이 필요하며 최대 4개의 병렬 처리가 요구된다.

2.2 Node Token

이제 NL을 사용하여 Node Token 을 다음과 같이 coding 한다.[12]

f(NL),(DNL 1,〔L/R〕),〔(DNL 2,〔L/R〕)〕

또는

f(Si, Pj),(Si1, Pj1 ,〔L/R〕),〔(Si2, Pj2 ,〔L/R〕)〕

f : =function (+, −, *, ÷, OR, Copy 등)

DNL 1 : = 1st Destination NL

DNL 2 : = 2nd Destination NL

L/R : = Left or Right Arc

〔 〕 : = optional

예로서 그림 1 에서 맨 윗부분 DL 1 의 copy Node는

Copy (1, 1), (2, 1), (2, 2)

로 쓸 수 있고 DL 3의 맨 왼쪽 2번째 copy 명령은

Copy (3, 2), (4, 1, R), (4, 3, L)

과 같이 Coding 할 수 있다. 이제 예제 프로그램의 모든 Node 들에 대해 coding 하면 표 1 과 같다. 표에서 DL 2의 sin 과 cos 은 편의상 단일 Node로 표시하였으나 실제로는 Function Call 또는 subroutine call 로 대체 되어야 한다.

DL 3의 4개의 copy 명령은 DL1 및 DL2의 모든 Node 실행이 끝난후 Firing 이 가능하며 이들 4개의 Token 은 동시에 처리될 수 있다. 이때 4개의processor 가 필요하며 DL 4, DL 6, DL 7, 의 경우도 마찬가지이다.

이와 같이 작성된 표 1의 DL 별 Token List 는 DFG와 그대로 상응성 (correspondency)[13] 을 가지며 Graph 의 단순한 서술에 불과하다.

이제 표 1의 List에 대해 PL별로 분류한후 DL 순으로 나열하면 표 2와 같다. 이 표를 살펴보면 동시 처리될 수 있는 Parallel Node들이 각 PL에 1개씩 분배되어 있고 각 PL들은 해당 DL들을 Level 순으로 나열시키고 있음을 알 수 있다. 여기서 PL을 processor 로 대체시키고 각 PL의 DL 순List 를 Local Memory 에 저장시켜 적당한 Algorithm의 제어 장치를 부착하면 이상의 논리에 부응하는 시스템을 구성할 수 있겠다. 이에 대해서 몇가지 구상이 가능하지만 기능 및 구조적으로 단순하고 해석이 용이하도록 다음과 같은 기본적 시스템을 구성하였다.

표 1. DL 순 Node Token

Table 1. Node Tokens in DL sequence.

DL1 :	COPY	(1,1), (2,1), (2,2)
DL2 :	SIN	(2,1), (3,2)
	COS	(2,2), (3,3)
DL3 :	COPY	(3,1), (4,1,L), (4,2,L)
	COPY	(3,2), (4,1,R), (4,3,L)
	COPY	(3,3), (4,2,R), (4,4,L)
	COPY	(3,4), (4,3,R), (4,4,R)
DL4 :	MULT	(4,1), (5,2,L)
	MULT	(4,2), (5,1,L)
	MULT	(4,3), (5,1,R)
	MULT	(4,4), (5,2,R)
DL5 :	ADD	(5,1), (6,3)
	SUBT	(5,2), (6,4)
DL6 :	COPY	(6,1), (7,1,L), (7,3,L)
	COPY	(6,2), (7,2,L), (7,4,L)
	COPY	(6,3), (7,1,R), (7,3,R)
	COPY	(6,4), (7,1,R), (7,4,R)
DL7 :	ADD	(7,1), A′
	ADD	(7,2), B′
	SUBT	(7,3), C′
	SUBT	(7,4), D′

2.3 구 조

구성된 시스템의 전체적 구조는 그림 2 와 같고 각 Unit 의 기능은 다음과 같다.

1) Level Counter (LC)

현재 실행하여야 할 Node 의 DL 값을 보관하며 실행이 끝난 후 increment 시킨다. 또한 임의의 값을 Load 할 수 있는 counting register 로서 구조상으로는 기존의 Program Counter 와 비슷하다.

Node 의 실행 완료 시각을 기준으로 상태천이(increment 또는 value change)되기 때문에 Node 의 function 종류에 따라 실행시간이 다르고 LC 의 update 시각이 달라진다.

즉 LC 는 모든 Prcessing Module 에서 현재 실행 중인 Node의 DL 값을 보관하고 이 값에 해당하는 Node를 addressing 하기 위한 register 이다.

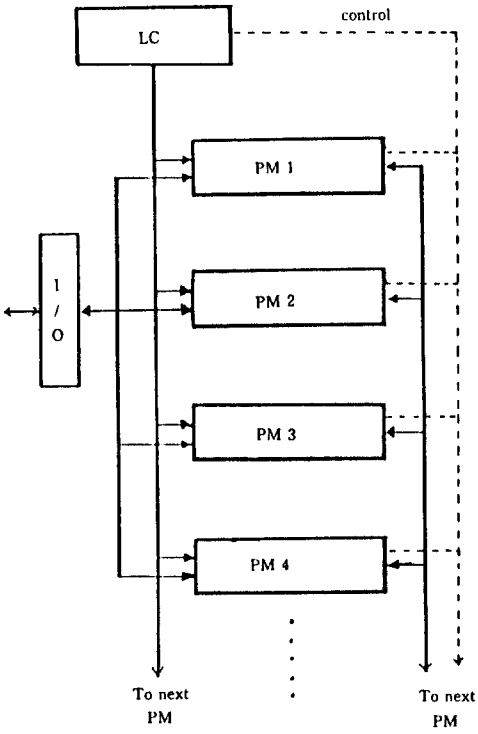

그림 2. 시스템의 기본 구조

Fig. 2. Basic system construction

2) I/O Interface

peripheral device 와의 intertace 로서 이 부분의 연구는 별도 과제로 한다.

3) Processing Module(PM)

할당된 프로그램을 데이타와 함께 보관하며 해당 Node 의 function 을 실행하는 module type processor 이다. 이는 그림 3 과 같이 6개의 Unit 로 구성되며 각각의 기능은 다음과 같다.

① Node Token Memory(NTM)

표 2의 Node Token 들을 machine code 로서 보관하는 local memory 이고 DL값을 key 로 하는 Associative memory(CAM) 이다. 최초 start 시 해당 프로그램을 각 NTM에 보관한 후 현재의 LC 값과 일치(match) 되는 DL값을 갖는 Node Token 이 fetch 되어 Processing Unit 로 보내어 진다. 표 2에서 각 Token 의 processor 할당은 이미 끝났으므로 PL값을 제외한 나머지 code, 즉

fSi, (DNL1, [L/R]), [(DNL2, [L/R])]

만 보관한다. 예제 프로그램에 대한 각 NTM의 보관 내용은 표 3 과 같다.

② Processing Unit(PU)

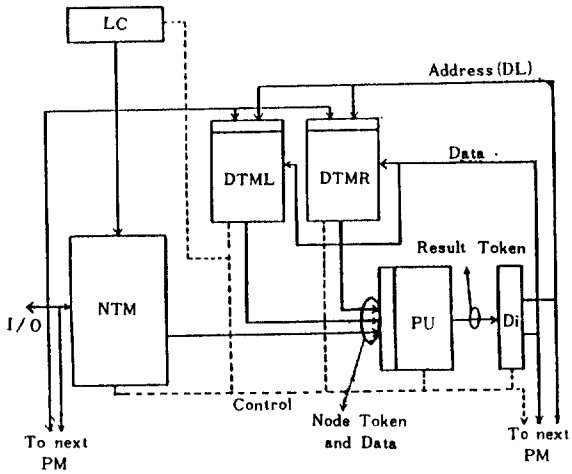

그림 3. PM의 내부구조

Fig. 3. Internal structure of PM .

표 2. PL 순 Node Token

Table 2. Node Tokens in PL sequence.

PL1 :	COPY	(1,1), (2,1), (2,2)
	SIN	(2,1), (3,2)
	COPY	(3,1), (4,1,L), (4,2,L)
	MULT	(4,1), (5,2,L)
	ADD	(5,1), (6,3)
	COPY	(6,1), (7,1,L), (7,3,L)
	ADD	(7,1), A′
PL2 :	COS	(2,2), (3,3)
	COPY	(3,2) (4,1,R), (4,3,L)
	MULT	(4,2), (5,1,L)
	SUBT	(5,2), (6,4)
	COPY	(6,2), (7,2,L), (7,4,L)
	ADD	(7,2), B′
PL3 :	COPY	(3,3), (4,2,R), (4,4,L)
	MULT	(4,3), (5,1,R)
	COPY	(6,3), (7,1,R), (7,3,R)
	SUBT	(7,3), C′
PL4 :	COPY	(3,4), (4,3,R), (4,4,R)
	MULT	(4,4), (5,2,R)
	COPY	(6,4), (7,1,R), (7,4,R)
	SUBT	(7,4), D′

표 3. NTM 보관내용
Table 3. Contents of NTM

Contents of each NTM

MTM 1

op code	DL	DNL1	DNL2
COPY	1	2,1	2,2
SIN	2	3,2	
COPY	3	4,1,L	4,2,L
MULT	4	5,2,L	
ADD	5	6,3	
COPY	6	7,1,L	7,3,L
ADD	7	A	

NTM 2

COS	2	3,3	
COPY	3	4,1,R	4,3,L
MULT	4	5,1,L	
SUBT	5	6,4	
COPY	6	7,2,L	7,4,L
ADD	7	B′	

NTM 3

COPY	3	4,2,R	4,4,L
MULT	4	5,1,R	
COPY	6	7,1,R	7,3,R
SUBT	7	C′	

NTM 4

COPY	3	4,3,R	4,4,R
MULT	4,	5,2,R	
COPY	6	7,1,R	7,3,R
SUBT	7	D′	

필요한 모든 산술 및 논리 operator(Adder, Subtracter, Multiplier, Divider, Copy, Comparator, Selecter, AND, OR 등)를 내장하고 적당한 decoder 와 address bypass 등으로 구성된 unit 로서 기존의 ALU와 비슷하다. NTM으로부터 fetch된 실행 가능(enabled) Node 를 decode 하고 동시에 Data Token Memory 로부터 operand 를 받아 명령에 따라 실행한 후 Distrbutor 를 거쳐 해당 목적지에 Result Token 을 내 보낸다.

③ Data Token Memory (DTM)[14]

DTM Left (DTML) 와 DTM Right(DTMR)로 구분되며 각각 1st 및 2nd operand 를 보관한다. addressing 을 간단히 하기 위해서 우선 DL값을 그대로 address 로 사용하도록 하였다. NTM으로부터 실행 가능 Node 가 PU로 읽혀짐과 동시에 해당 Node 가 필요로 하는 operand 데이타가 DTM에서 읽혀져 처리된다. 처리된 결과를 Distributor 가 받아 DL 값을 address 로 하여 목적지 DTML 또는 DTMR에 write 한다. 이 unit 는 범용 RAM 으로 구성된다.

④ Distributor

PU로 부터 생성된 1개 또는 2개(copy 등 output Arc가 2개인 경우)의 Result Token 을 목적지에 따라 distrtbute 및 write 한다. contention 발생이 쉬운곳으로 지금까지의 연구[4],[7],[9]에서는 Queueing 및 Multiplex에 의한 방법을 사용하고 있다.

⑤ Control

PM내부의 각 Unit 사이의 동기 및 LC update등을 제어한다. 각 Unit 는 해당 Unit에서의 최대 처리시간+알파 시간 후에 다음 Unit 의 동작이 시작되도록 동기된다.

현재 실행중인 모든 Node 들의 실행 완료 시각의 감지는 실행 완료 및 미완료 상태를 나타내는 Flag 를 PM별로 설치되고 모든 Flag 가 실행 완료된 상태에 있을때 LC를 update 시킬 수 있도록 한다. Node 의 실행 완료는 결국 result 의 해당 목적지에의 write 완료된 시각이므로 control 은 Distributor와의 communication 이 필요하다.

2.4 동 작

동작 상태를 간단히 도시하면 그림 4 와 같이 Ring Type 실행 cycle 을 이루며 점선안의 subroutine call[7],[18] 별도 연구 과제로 한다. 동작 과정을 순서적으로 설명하면 다음과 같다.

1). PM별로 분류된 machine code 를 각각의 NTM에 Load 시키고 LC 값을 1로 set 하면서 start 한다.

2). 모든 NTM에서 DL값이 1인 Node 가 fetch 되고 동시에 이 Node가 필요로 하는 데이타가 DTM으로부터 읽혀져 PU로 보내진다.

3). PU는 fetch 된 Node 의 op code 를 decode하고 결과에 따라 1개 또는 2개의 operand에 대해 execute 한다.

4). PU에서 생성된 Result 데이타와 NTM 으로부터 보내어진 destination NL을 Distributor 가 받아 해당 DTML/R에 write 한다.

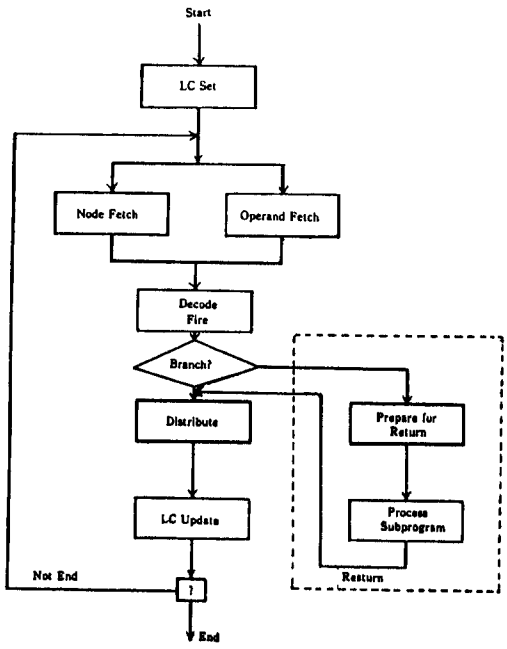

그림 4. 동작 흐름도

Fig. 4. Flow of operation.

5). 모든 PM이 현재 실행중인 Node들에 대한 실행 cycle을 완전히 끝마쳤을때(result의 DTM write까지) LC값을 increment 시킨다.

6). 다음 실행 cycle로 반복한다.

2.5 검 토

본 시스템은 다음과 같은 특징을 갖는다.

1) Dennis등의 Arbitor가 없고 모든 PM들은 그것이 처리하여야 할 Node들을 Local Memory에 보관한다. 따라서 system software의 overhead(sorting 등)가 약간 높아 지나 각 Module의 hardware는 단순하여 진다.

2) 기존의 Program Counter와 기능이 유사한 LC를 장치하였으나 clock pulse에 의하여 동기되지 않는다. 그러나 모든 processor의 실행시작이 LC에 의하여 동기되어 있다. 따라서 실행시간이 짧은 Adder의 계산이 시간이 긴 Multiplier와 동시 처리될 경우 긴쪽에 동기된다. 이점은 특별한 경우(데이타 의존도가 낮을 경우등)를 제외하고는 전체적 계산 속도에 지장을 초래치 않음을 추론할 수 있다.

3) DTML/R는 temporary scratch pad이고 DL의 depth에 따라 그 용량이 결정된다. NTM도 DL depth가 클 수록 큰 것을 사용하여야 하나 이들 모두 확장이 어렵지 않다.

4) PL은 PM의 병렬 연결로 확장 가능하다. 따라서 NTM과 DTM을 충분히 큰 것으로 장치하면 PM의 확장만으로 전체의 용량을 확장할 수 있다.[10]

5) 동시 처리 Node 수가 PM수보다 클 경우 DL값을 달리하거나 별도의 code(color code등과 유사한)를 이용하여 처리될 수 있다.

3. 결 론

DFG는 병렬 처리 가능 Node의 판별에도 유효하지만 처리 과정상 불가피한 직렬 데이타 의존 상태도 포함하고 있다.[16] 이미 제안된 DFM에서는 이들 Sequential 과정을 제어하는 기능이 없고 enabled Node는 순서없이 random(matching이 발견되기만 하면) 하게 처리하거나 적당한 routing[15]에 의하여 조정된다. 본 논문에서는 Sequential 과정을 code에 포함시키고 LC로 이를 제어하도록 하였다. 따라서 Von Neuman 방식의 Sequential 처리 방법을 복합시킨 것이라 할 수 있다.

현재까지 제안 및 실현된 DFM들은 대부분 아직 성숙된 단계에 있지 않고 신호 처리, Weather Modeling[20] 등 특수 용도의 컴퓨터로서 시험 적용시켜 보는 단계 있다. 따라서 범용 내지는 Supercomputer로서 원활한 응용이 가능하게 되기 까지는 아직 많은 연구 및 개선이 계속되어야 하리라 본다.

參 考 文 獻

1) Vasilii Zakharov, "Parallelism and Array Processing", IEEE Trans. on Computers, C-33 (1), Jan. 84, pp. 45-78.

2) Steven R. Vegdahl, "A Survey of Proposed Architectures for the Execution of Functional Languages", IEEE Trans. On Computers, C-33 (12). Dec. 84, pp.1064-1067.

3) A.L. Davis, "Computer Architecture", IEEE Spectrum, 20 (11), Nov. 83, pp. 94-99.

4) J.B. Dennis, "Data Flow Supercomputers", Computer, 13 (11), 1980, pp. 48-56.

5) A.L. Davis, "The Architecture and System Method of DDM1: A Recursively Structured Data Driven Machine", ACM Proc. of 5th Ann. Symp. on Computer Archit ecture 1978.

6) A. Plas et al., "LAU System Architecture: A Parallel Data Driven Processor Based on Single

Assignment", IEEE Proc., 1976 IC on Parallel Processing.

7) J.E. Rumbaugh, "A Data Flow Multiprocessor", IEEE Trans. on Computers, 1977, C-26 (2), pp. 138-146.
8) I. Watson, J. Gurd, "A Practical Data Flow Computer", Computer, Vol. 15, Feb. 1982, pp. 51-57.
9) Arvind, "The Tagged Token Data Flow Architecture", MIT Lab. Technical Report. Oct. 1984.
10) A.L. Davis, "A Data Flow Evaluation System Based on the Concept of Recursive Locality", AFIPS Proc., 1979, pp.1076-1086.
11) A.L. Davis, R.M. Keller, "Data Flow Program Graphs", Computer, Vol. 15 Feb. 1982, pp 26-41.
12) I. Watson, J. Gurd, "A Prototype Data Flow Computer with Token Labeling", AFIPS Proc., 1979 NCC, pp.623-628.
13) J. Backus, "Can Programming Be Liberated from the Von Neuman Style? A Fuctional Style and its Algebra of Program", ACM, Communication of 1978, 21 (8), pp.613-641.
14) Masahiro Sowa, Tadao Maruta, "A Data Flow Computer Architecture with Program and Token Memories", IEEE Trans. on Computer, Sept. 1982. pp. 820-824, C-31 (9).
15) J.B. Dennis, "Packet Communication Architecture", IEEE Proc. SCC on Parallel Processing, 1975, pp. 224-229.
16) David J. Evans, "Parallel Processing Systems", Cambridge Univ. Press. 1982. pp. 239-249.
17) Myers, "Advances in Computer Architecture", JWS, 1982, pp. 482-486.
18) Sowa, Ramos, "Subroutine Implementation in DFNDF-Data Flow Computer", 일본전자통신학회 EC 83-16, 1983년 7월
19) Dennis, Boughton, "Building Blocks for Data Flow Prototypes", IEEE Proc. 1980, Apr.
20) Dennis, Gao, Todd, "Modeling the Weather with a Data Flow Super Computer", IEEE Trans. on Computers. C-33 (7), Jul. 1984, pp.592-603.