구조성 데이터의 입체식 계수기법에 의한 벡터 처리개념의 설계

조 영 일† 박 장 춘††

요 약

스칼라 처리지향의 기계에서 벡터 처리를 위해서는 스칼라 처리가 벡터 요소 수 만큼 수행되어야 한다. 소위 von Neumann 원리에 의한 벡터 처리기법이다. 메모리를 악세스하는 장치로는 명령어의 순차적 계수를 위한 프로그램 계수기 뿐이기 때문에 벡터 데이터의 악세스는 명령어의 지시나 또는 ALU 의 주소계산에 의해 수행되어야 한다. 여기서는 재래식 개념의 하드웨어적 결함을 보충하기 위해 벡터 요소들을 입체적으로 악세스하기 위한 악세스 장치의 설계를 제안한다. 벡터의 구조처리를 위한 필요성은 명령어군에 포함되었고 그를 명령어들은 데이터 처리와 동시에 데이터 악세스 동안에 처리되도록 한다.

An Architecture of Vector Processor Concept using Dimensional Counting Mechanism of Structured Data

Young Ill Cho† Jang Chun Park††

ABSTRACT

In the scalar processing oriented machine scalar operations must be performed for the vector processing as many as the number of vector components. So called a vector processing mechanism by the von Neumann operational principle. Accessing vector data has to be performed by the every pointing of the instruction or by the address calculation of the ALU, because there is only a program counter(PC) for the sequential counting of the instructions as a memory accessing device. It should be here proposed that an access unit dimensionally to address components has to be designed for the compensation of the organizational hardware defect of the conventional concept. The necessity for the vector structuring has to be implemented in the instruction set and be performed in the mid of the accessing data memory overlapped externally to the data processing unit at the same time.

1. 서 론

스칼라 처리에서 데이터는 일정한 양(量)을 뜻하는 수치 외에는 다른 의미를 갖고 있지 않은 것이다.

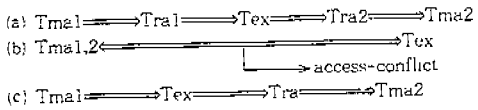
그러나 벡터는 처리의 대상인 양을 뜻하는 것 외에 숫자가 하나 이상이므로 구조적 크기에 대한 외형적인 의미도 갖는다. 다시 말해, 사용되는 변수는 담겨진 숫자 외에 그 숫자들의 외형적인 크기를 포함해야 하며, 또 그 숫자들의 처리에 따라서 그 외형의 크기 조차도 처리되어야 한다. 스칼라 처리에서는 하나의 명령어가 한 쌍의 데이터를 처리하기 때문에 수행에서 데이터 자체는 한번의 변형만으로 오퍼레이션 수행완성이 가능하지만 벡터 처리에서는 변수에 포함되어 있는 벡터 요소들의 외형적 크기에 따라서 하나의 명령어가 같은 오퍼레이션을 여러번 수행해야 한다.

* 본 논문은 1993년도 한림대학교 학술연구조성 교비지원에 의해 연구되었음.
† 정 회 원 : 한림대학교 컴퓨터공학과 교수
†† 정 회 원 : 건국대학교 전자계산학과 교수
논문접수 : 1995년 6월 14일, 심사완료 : 1995년 12월 11일

벡터 처리를 위해 설계된 컴퓨터는 데이터의 요소들이 기본적으로 스칼라 단위로 처리된다는 점에서는 스칼라 처리와 차이는 없으나, 데이터 단위 자체가 구조성을 갖는 벡터 단위로 사용된다는 점에서 설계 개념상 스칼라 처리 컴퓨터와 달라야 하는 것이다. 하나의 데이터 단위에서 벡터 요소들은 서로의 연관성과 개개의 주소로 메모리 내의 일정한 수의 인접한 저장장소를 갖는 것으로 정의될 수 있다. 이러한 데이터 요소들은 같은 유형의 데이터들이며 서로의 구조적 연관성 때문에 하나의 명령어 지시에 의해서 함께 처리되어야 하는 것이다(SIMD). 그러므로 벡터 처리를 할 수 있다는 것은 명령어처리 파이프라인 설계와는 필수의 관계를 갖는 것이 아니며, 벡터의 구조에 대한 처리가 먼저 수행되어야 하기 때문에 실제로 현대의 벡터 컴퓨터 설계이론에서는 벡터 처리를 위해 기존의 프로세서 개념(von Neumann type)에 부가적으로 벡터의 구조에 따라 악세스 처리와 제어를 위한 하드웨어 장치의 조직구조를 갖는다고 보아야 할 것이다[1, 2, 3, 4, 5].

지금까지 개발된 벡터 계산기로는 수행 방법에서 두 가지 설계개념이 있는데, 그 하나는 vector-register 개념과 또 다른 하나는 memory-memory vector 개념이다. 벡터-레지스터 계산기에서는 load와 store를 제외한 모든 벡터 오퍼레이션이 레지스터와 레지스터 사이에서만 이루어 진다[6]. 다시 말해, 전자의 경우는 오퍼레이션이 수행되기 위해서는 데이터가 먼저 벡터 레지스터(buffer)까지 들어와 있어야 하며(그림 1.a), 후자의 경우에는 메모리에서 악세스 되면서 바로 오퍼레이션이 수행된 다음에 결과는 바로 메모리로 들어가는 형식이다(그림 1.b). 그것은 메모리와 연산처리부 사이에 연산처리 이전 또는 이후에 오퍼랜드가 임시 저장되어 대기시키는 버퍼 저장장치가 따로 없는 구조이다. 여기에서 벡터가 수행중에 나오고, 또 그 수행 중에 결과 벡터가 메모리로 들어갈 때에 충돌(access-conflict)을 피하기 위해서는 수행 중에 긴 시간이 지연되는 원인이 있으므로 결과적으로 전자의 설계개념이 빠른 처리개념으로 Cray-1, NEC SX/2, Fujitsu VP-200 등이 있다. 그러나 CDC-STAR, Cyber 205의 경우 높은 메모리 대역폭(400 MW/s)으로 memory to memory 오퍼레이션이 되고 있기도 하다[2, 7].

본 연구에서는 위에 기술된 두가지 설계개념과는 다르게 오퍼랜드가 메모리에서 스트림으로 나오면서 바로 파이프라인 처리되며 처리되는 결과는 메모리 출력과 충돌을 일으키지 않게 하기 위하여 그 오퍼레이션이 완전히 끝날때까지 대기할 수 있는 레지스터버퍼(그림 1.c)가 설계된다.



Tma : 메모리 악세스 시간
Tra : 레지스터 악세스 시간
Tex : 프로세서 수행 시간

(그림 1) 벡터 처리개념의 처리시간 비교
(Fig. 1) Processing time for vector machines

본 논문의 구성은 2장에서는 오퍼레이션의 원리에 대하여, 3장에서는 사용되는 명령어와 데이터의 형식에 대하여, 4장에서는 구조성 데이터, 즉, 벡터의 구조처리에 대하여, 5장에서는 벡터 자체의 처리, 6장에서는 벡터 처리를 목적으로 설계되는 계산기 조직의 역할, 7장의 결론으로 구성되었다.

2. 오퍼레이션 원리

일반적으로 von Neumann 개념에서 하나의 변수(variable)가 의미함은 하나의 이름(name)에 하나의 수치(numerical value)가 부여될 수 있음을 뜻한다. 다시 말해, 하나의 이름(name)은 하나의 스칼라(scalar) 값을 뜻하는 변수(scalar variable)라 할 수 있다. 그러나 그것에 반해 벡터 변수의 개념은 여러개의 스칼라 값들로 구성되는 하나의 단위가 필요하므로 단위 자체의 크기(구조)의 표현이 필요하게 된다. 이러한 관계를 정리해 보면, 스칼라 변수는

von Neumann variable :: = (name, value)
value :: = scalar

이다. 그러나 벡터 변수의 경우에는

vector variable :: = (name, value)
value :: = (dimension, data)

로 표현 해야 한다. 이러한 벡터 변수의 처리에는 데이터 구조의 처리와 데이터 자체의 처리로 수행되어야 한다. 전자의 처리는 후자의 처리에 따라서 일어나기도 하지만, 경우에 따라서는 전자의 처리가 후자보다 선행되어야 하는것이 벡터 처리의 특징이기도 하다.

하나의 벡터를 이루는 요소들, 즉, 데이터의 집합은 일반적으로 정돈되지 않은 집합이므로 어떤 형태로든 저장부에 저장되어야 한다. 이런 데이터의 정돈은 임의로 일어날 수도 있으며, 가장 편리한 방법으로는 메모리 내에서 국소적으로 인접한 주소에 선형적으로 저장되어야 하는 것이다. 즉,벡터화된 데이터(vectored data) 또는 벡터 데이터이다. 데이터의 집합 S의 요소들로 구성된 길이 N의 벡터 데이터는 다음과 같이 악세스 함수(α : access function)인 식(1)로 매핑된다.

$$\alpha : M \supset [b : b-1+N] \Longrightarrow S \qquad (1)$$

여기서 M 은 메모리 주소의 집합이고, $b \in M$은 첫 번째 벡터 요소의 주소, 즉, 기본주소 이다. S는 메모리 내에서 길이 N을 이루어 선형적으로 저장되기 위한 전체 벡터 요소들을 뜻한다.

하나의 선형을 이루는 길이 N의 벡터 요소들이 메모리 내의 실주소 M에 이르기 위해서는 리스트 함수(τ)인 식(2)로 매핑이 필요하게 된다.

$$\tau : N \Longrightarrow M \qquad (2)$$

r-차원의 입체구조를 갖는 N개의 벡터 요소를 N^r이라 하면, 선형적 벡터 데이터로 전환시키는 인덱스 함수(ζ:index function)가 다음의 식(3)과 같은 매핑으로 세워진다.

$$\zeta : N^r \Longrightarrow N,\ N=\text{자연수의 집합} \qquad (3)$$

α

	β					β			
	1	2	3	4		13	14	15	16
λ	5	6	7	8	λ	17	18	19	20
	9	10	11	12		21	22	23	24

r-차원의 벡터$(n_0, n_1, n_2, \cdots, n_{r-1}) \in N^r$ 요소의 위치를 좌표 값으로 하고, r는 그 데이터의 차원이라 하면, $1 \le ni \le di$, $0 \le i \le r-1$의 관계가 성립될 수 있다. 여기서 최대값 d_i는 i-번째 필드의 최대값이며, 매트릭스의 좌표값이라 하고, $(d_0, d_1, d_2, \cdots, d_{r-1})$은 필드를 뜻하는 벡터이다. 다음과 같은 하나의 3 차원 벡터, 즉, 입체 데이터에서 $[\alpha\ \beta\ \lambda]$는 데이터의 입체형을 뜻하는 벡터로서 3차원으로 구성된 벡터 데이터를 뜻한다.

벡터의 구조는 이러한 차원 벡터를 줌으로서 완전히 표현되는 것이다. 이러한 매트릭스는 array, string, list, queue 등 에서도 1-차원의 매트릭스로 표기되기 때문에 마찬가지 이다.

위의 매핑 (1), (2), (3)에서 악세스 함수(α)는 S로 표현되는 데이터의 메모리 내에서 실제 구조인 반면에, 리스트 함수(τ)는 N-개의 요소들을 M으로 표현하는 메모리 내에서 서로 인접한 벡터를 가르키게 된다. 인덱스 함수(ζ)는 r-차원 N-개의 요소로 구성된 벡터의 외형적인 논리적 구조를 뜻하게 된다. 외형적인 논리구조$(n_0, n_1, n_2, \cdots, n_{r-1}) \in N^r$로부터 메모리 내의 선형구조 $s_i \in S$에 도달하기 위해서는 또 하나의 매핑이 필요하다.

지금까지의 모든 전위과정($N^r \Longrightarrow N \Longrightarrow M \Longrightarrow S$)을 주소지정 함수($\theta$:addressing function)라 하는 매핑(4)으로 유도할 수 있다.

$$\theta : N^r \Longrightarrow S \qquad (4)$$

이 함수는 앞에서 검토된 악세스 함수(α), 리스트 함수(τ), 인덱스 함수(ζ)를 통해 메모리에 선형으로 저장되어있는 r-차원 구조의 벡터데이터에 악세스가 가능한 것이다.

인덱스 함수(ζ)와 리스트 함수(τ)를 합성한 메모리 저장 매핑($\zeta o \tau : N^r \Longrightarrow M$)은 구조 N^r 을 갖는 데이터가[b : b−1+N]의 벡터 데이터로 되는 매핑이 된다. 즉, 벡터 데이터는 행(row) 또는 열(column)로 저장됨을 뜻한다. 이러한 인덱스 함수(ζ)에서 인덱스의 구조처리를 위해 수행될 수 있는 구조처리 함수(δ)는 식(5)의 매핑으로 표현될 수 있다.

$$\delta : N^r \Longrightarrow N^t \qquad (5)$$

여기서 구조처리 함수(δ)는 벡터 $A[k_0 ; k_1 ; \cdots ; k_{t-1}]$를 뜻하는 인덱스 t의 구조$(k_0 ; k_1 ; \cdots ; k_{t-1})N^t$가 벡터 $B[n_0 ; n_1 ; \cdots ; n_{r-1}]$를 뜻하는 인덱스 r의 구조$(n_0 ; n_1 ; \cdots ; n_{r-1}) \in N^t$로 대응되는 것을 뜻하는 것이다. 매트릭스의 벡터 데이터는 구조를 바꾸는 오퍼레이션에 의해 요소들이 이루고 있는 구조는 변형될 수는 있으나, 구조처리의 결과로 남아 있는 각각 요소들의 값은 불변인 것이다. 그 이유는 N^t의 요소들과 데이터 벡터 요소들 사이의 관계를 나타내기 위해서 구조처리 함수(δ)와 인덱스 함수(ζ)의 합성($\zeta_t \circ \delta$)으로 해당 인덱스 구조처리 함수(ζ_t)를 세우는 것으로 충분하기 때문이다.

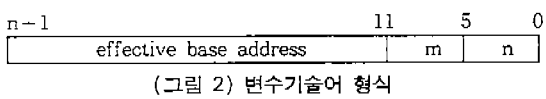
이와 같이 합성된 구조처리 오퍼레이션은 벡터 좌표 내에서 벡터 요소의 순서처리로 전개시켜 벡터의 순서가 새로운 벡터로 오퍼레이션 될 수 있다. 이런 오퍼레이션은 계수를 정수로 줌으로서 가능하다. 요소들의 순서처리를 구조처리 오퍼레이션 함수($\delta : N^r \Longrightarrow N^t$)에 삽입하는 것은 r-좌표 내에서 벡터 요소들을 치환하는 것에 해당한다. 이로써 벡터 처리에는 1). 벡터 데이터 처리 오퍼레이션과 2). 벡터 구조처리 오퍼레이션이 수행될 수 있음을 이해할 수 있다.

앞에서 언급하였듯이 복합적인 구조를 갖는 구조성 데이터의 구조처리가 사용자의 프로그램 작성영역에서 다루어지고 있다. 이러한 기법은 von Neumann 개념에 특징으로 내재해 있는 "하나의 변수에 메모리 하나의 주소 관계"에 의한 것이라 할 수 있다. 다시 말해, von Neumann 변수는 하나의 단순한 값(scalar)만을 위해 사용될 수 있다는 것이다[8, 9, 10]. 그러나 하나의 변수로 복합적인 값을 가질 수 있다고 전제할 때에 그 값들은 메모리 내에서 인접한 각각의 주소에 저장되어야만 하므로, 메모리 내적으로는 선형적인 위치를 갖는 벡터 영역을 필요로 한다. 역으로 보면, 메모리에서 벡터를 이루는 이러한 단위 데이터는 외형적 구조를 갖는다. 구조성 변수들의 악세스는 개개 주소의 연산을 전제로 한다. 이 변수들의 처리에는 두 가지, 즉, 구조처리와 데이터 처리가 수행되고 있으나, 그 이면에는 제어 데이터의 흐름이 숨겨져 있으며, 그 흐름은 데이터 요소들의 흐름에서 기인하고 프로그램 제어 흐름에 영향을 주는 값이라 할 수 있다[11].

벡터를 위한 변수는 그 벡터의 요소들 뿐만 아니라 그 요소들에 의한 구조(크기)까지도 포함해야 한다. 그러나 변수는 명령어에 동반될 수 있어야 하는 조건이 전제되어야 하며, 명령어 형식의 제한성으로 인하여 명령어상에 벡터의 구조, 기본주소 등을 동반시킬 수 없는 것이다. 그러므로 이러한 속성의 집합을 또 하나의 정보단위로 하여 변수기술어라 한다. 이것은 일종의 벡터 데이터를 수용할 수 있는 벡터 변수라 할 수 있다. 스칼라 변수가 단순한 "하나의 수치"를 수용하는 "그릇"이라 할 수 있음에 반해, 변수기술어는 벡터의 크기, 기본주소 등을 수용하며, 이러한 구조적 속성 데이터를 "구조적 관계성을 포함한 데이터의 테이블" 이라 한다[12].

3. 명령어와 데이터 정보의 구조

벡터 처리를 위한 컴퓨터에서 재래식 컴퓨터 명령어와 다른 점은 크게 명령어와 그것에 수반되는 변수 외에 변수기술어가 한가지 더 있어야 한다는 것이다. 이 변수기술어는 벡터 데이터 오퍼레이션만을 위한 것으로 벡터 데이터 요소들이 갖는 메모리 내의 실주소, 구조 등을 갖는다. 명령어들은 재래식 컴퓨터의 명령어, 즉, 스칼라 처리 명령어 외에 벡터 처리를 위한 명령어가 구분되어 있어야 한다. 그 이유로는 벡터 처리는 같은 명령어라도 처리될 벡터 데이터의 크기(m, n)에 따라서 명령어 오퍼레이션의 시간이 달라지기 때문이다. 전체적으로 변수는 재래식 명령어(von Neumann-type)처리에 사용되는 데이터의 "그릇" 이라 할 수 있고, 여기서 의미하는 변수기술어는 벡터 처리명령어에 사용되는 데이터의 "그릇" 이라 할 수 있으며 (그림 2)와 같은 형식을 갖는다.

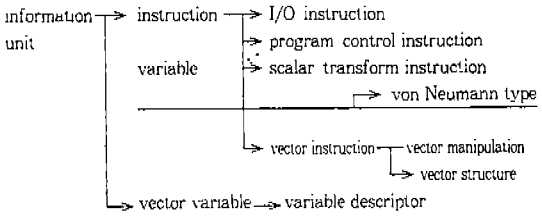


(그림 2) 변수기술어 형식
(Fig. 2) Variable descriptor format

3.1 정보단위의 기능과 형식

본 설계에서는 처리대상의 정보단위는 기계명

령어와 데이터이며, 데이터는 스칼라와 벡터 단위로 구분 처리됨을 전제로 한다. 재래식 von Neumann 기계의 명령어가 입/출-명령어, 프로그램 제어명령어, 데이터 처리명령어의 세 가지로 분류됨에 반해, 본 설계 개념에서는 벡터 처리를 위한 데이터 구조처리 제어명령어를 포함하여 다음과 같이 네 가지로 분류될 수 있다.



데이터 처리 정보: 데이터의 수치변환을 위해 4칙 연산(+, −, x, /)과 여러개의 논리연산(NOT, OR, AND)을 기본으로 하여 스칼라 오퍼레이션과 벡터 처리 오퍼레이션으로 구분된다. 스칼라 오퍼레이션의 논리연산에서는 오퍼랜드의 비트 수준에서 수행되며 벡터 처리를 제외한 스칼라 오퍼레이션을 위한 명령어는 von Neumann기계명령어군에 준한다.

벡터 처리 정보: 벡터 단위에서 요소들을 데이터로 처리하기 위해서는 벡터 오퍼레이션 코드(Vopc)와 소스벡터 오퍼랜드를 최고 2개와 결과 값 벡터를 위한 하나의 오퍼랜드를 지시하는 전체 4개의 필드로 구성된다. 그러나 변수단위로 하나의 벡터를 필요로 하는 모나딕 오퍼레이션에서는 하나의 소스오퍼랜드만 사용되므로 한 필드는 사용되지 않는다. 벡터 처리 명령어를 감축 오퍼레이션(reductive operation), 요소별 오퍼레이션(element by element operation), 내적 처리 오퍼레이션(inner product operation)등 으로 분류할 수 있다. 벡터 처리 오퍼레이션에서 A,B,C를 임의의 [m, n]-행렬이라 할 때에 다음의 정리가 성립되어야 한다[13].

(a) $A+B=B+A$

(b) $A+(B+C)=(A+B)+C$

(c) $s(A+B)=s\cdot A+s\cdot B=(A+B)\cdot s$, $s=$scalar

(d) $\exists D: A+D=B$

(e) $A\cdot(B+C)=A\cdot B+A\cdot C$

(f) $(A+B)C=A\cdot C+B\cdot C$

(g) $A\cdot(B\cdot C)=(A\cdot B)\cdot C$

(h) $A\cdot B\neq B\cdot A$

(i) $A\cdot B=0$ doesn't necessarily imply $A=0$ or $B=0$

(j) $A\cdot B=A\cdot C$ doesn't necessarily imply $B=C$

요소별 오퍼레이션은 위의 정리 (a), (b), (c)에 의한 같은 구조[m, n]를 갖는 벡터에서 같은 좌표상(i, j)의 요소별 오퍼레이션이고, 감축 오퍼레이션과 내적 오퍼레이션은 (e), (f), (g)에 의한 명령어로 구현되어야 한다.

벡터 구조처리 정보: 구조를 갖는 벡터에서 데이터 수치의 전환이 아니고, 변수단위의 벡터에서 구조의 변형을 위한 기능을 갖는다. 이 명령어는 벡터 처리언어 APL의 선택 명령어(selective operation)와 같은 것으로 rotate, take, drop, select, transpose, identity 등이다. 이들 명령어는 벡터 악세스 제어부(VAC)의 주소생성과정, 즉, 구조처리 함수에 의해서 수행되는 오퍼레이션 들이다. 이 구조처리명령어 오퍼레이션은 벡터 데이터를 악세스 제어하는 과정에서 수행되는 명령어이므로 벡터 명령처리부(VIPU)의 벡터 악세스 제어부(VAC)에서 처리 된다.

위의 벡터 처리명령어와 벡터 구조처리명령어의 수행에는 소스 변수(Vd1)가 필요하며, 명령어 수행에서 결과변수를 위한 변수기술어가 생성되어야 하는 것이다. 이러한 변수기술어들은 명령어 형식과 같은 길이를 가지며, 다시 하나의 변수기술어 형식은 세 개의 필드로 나누어져 첫 번째 벡터 요소가 갖는 메모리 내의 주소를 실기본주소(eba), 벡터의 구조(m, n)등을 포함한다. 이러한 변수기술어들은 입/출력 프로세서에서 입력 데이터의 구조에 따라서 생성되어 변수기술어 테이블에 기록되기도 하고, 또 한편으로는 벡터 악세스 제어부(VAC)에서 구조처리 명령어의 수행시 또는 데이터 처리부(DPU)의 데이터 처리명령어 수행에 앞서서 벡터 명령처리부(VIPU)에서 결과변수의 변수기술어로

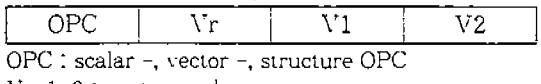
생성되기도 한다. 입/출력부가 스칼라 데이터를 받으면, 변수기술어는 크기(m=0,n=0)가 없이 기록되므로 데이터 자체는 하나의 주소에 기록되게 된다.

지금까지 위에서 검토된 명령어는 오퍼레이션 테이블 〈표 1〉에 정리 되었다. 벡터 처리명령어에서는 오퍼레이션 소스 변수(Vd1)가 하나만 필요하므로 모나딕 명령어, 소스 변수가 두개(Vd1, -2 : 필요시 하나는 스칼라) 필요하면 다이애딕 명령어라 분류하며, 두 가지 모두 파이프라인

〈표 1〉 오퍼레이션 테이블
(Table 1) Operation table

		instruction	operation
벡터처리	모나딕	reductive addition	scalar={…((a₀+a₁)+a₂)+…}+aₙ₋₁
		reductive subtraction	scalar={…((a₀−a₁)−a₂)−…}−aₙ₋₁
		reductive multiplication	scalar={…((a₀*a₁)*a₂)*…}*aₙ₋₁
		reductive division	scalar={…((a₀/a₁)/a₂)/…}/aₙ₋₁
		vector square root	vector=sqr(a₀, a₁, a₂, …,aₙ₋₁)
		vector complement	vector=comp(a₀, a₁, a₂,…,aₙ₋₁)
	다이애딕	vector vector addition	C[i, j]=A[i, j]+B[i, j]
		vector vector multiplication	C[i, j]=A[i, j]*B[i, j]
		vector vector ANDing	C[i, j]=A[i, j]∧B[i, j]
		vector vector ORing	C[i, j]=A[i, j]∨B[i, j]
		vector scalar multiplication	C[i, j]=A[i, j]*scalar
		inner product	C[i, j]=A[i, j]•B[k, l]
구조처리	벡터구조처리	rotate in row	(a₀, …, aₙ₋₁, a₀, a₁, a₂, …)=
		rotate in column	i o(a₀, a₁, a₂, …, aₙ₋₁, a₀, …, aₙ₋₁)
		take in row	(a₀, a₁, a₂, …)=
		take in column	(a₀, …, aₙ₋₁)=
		drop in row	i o(a₀, a₁, a₂, …,aₙ₋₁, a₀, …,aₙ₋₁)
		drop in column	i o(a₀, a₁, a₂,…, aₙ₋₁, a₀, …, aₙ₋₁)
		select in row	a=
		select in column	i o(a₀, a₁, a₂, …, aₙ₋₁,a₀, …,aₙ₋₁)
		transpose	[n, m]=T[m, n]
			i : row-or column parameter
			o:operation in row or column

데이터 처리부(DPU)에서 수행처리 된다. 벡터 구조처리 명령어는 오퍼레이션 자체가 데이터 처리가 아니므로 벡터 악세스 제어부(VAC)의 메모리 악세스 과정에서 수행된다. 이들 벡터 처리ㅡ, 벡터 구조처리 명령어 형식은 하나의 같은 크기를 가지며, (그림 3)과 같다.



OPC : scalar -, vector -, structure OPC
Vr, 1, 2 : vector, scalar,
V2 : row -, column parameter

(그림 3) 명령어 형식
(Fig. 3) Machine instruction format

3.2 입체 계수 기법

데이터 악세스는 소프트웨어적으로는 데이터 주소지정이며, 하드웨어적으로는 오퍼랜드를 가져오기 위한 메모리 주소지정이라 할 수 있다.

외형적 구조[m, n]을 갖는 벡터가 메모리에 저장되어 있을 때에 각각의 요소들은 하나씩 개개의 주소를 연속적, 국소적 집단으로 인접하여 저장되어 있기 때문에 외형적 논리구조에 따르는 악세스 알고리즘이 사용되어야 한다. 이러한 벡터 데이터는 입/출력부를 통해서 저장될 때에 입력부에 의해서 생성되고 변수기술어 테이블에 기록되어 있는 변수기술어 내의 구조계수에 의해서 먼저 m개의 행과 n개의 열로 구성된 벡터 데이터를 계수하는 데는 그 계산목적에 따라서 행순으로 하는 방법 또는 열순으로 하는 방법이 있다. 크기[m, n]의 벡터 악세스를 스칼라 처리 컴퓨터에서 고급언어의 스칼라 명령으로 수행하려면 프로그램 제어명령어를 사용해야 한다. 이것은 엄격한 의미에서 매트릭스 내지는 벡터의 구조제어로 분류되어야 한다. 그러나 이러한 구조제어 수행 마져도 von Neumann 기계의 ALU에서 한번씩 오퍼랜드 처리수행이 일어날 때마다 먼저 메모리로부터 오퍼랜드를 악세스 해오기 위해서 주소계산과 더불어 수행해야 하는 부담이 ALU에 이어져야 한다.

여기서 벡터를 i, 0 i, 1 i, 2 …i, j…i, (n-1)으로 읽는 것을 행순이라 하고, 0, j 1, j 2, j… i, j…(m-1), j로 읽는 것을 열순이라 하며, 데이터의 변수단위를 2-차원 벡터, 즉, 매트릭스로 하는 것에는 다음의 이유에서 이다. 첫째로 매트릭스는 가장 기본적인 구조성 데이터이기 때문이다. 즉, array, string 등 도 하나의 행 내지는 하나의 열로 구성되는 매트릭스 이다. 둘째로 고차원적인 array(hyper matrice)도 매트릭스 단위로 나누어질 수 있기 때문이다. 매트릭스의 차원 벡터는 두개의 요소를 갖는다. 첫 번째는 열의 크기(m)이고, 두 번째는 행의 크기(n) 이다. 매트릭스는 단지 두개의 좌표 값을 갖기 때문에 행과 열의 기본 구성을 위해 분리된 오퍼레이션을 수행할 수도 있다. 매트릭스의 구조설정은 매트릭스 데이터 벡터의 참조형식은 물론 또 다른 실제 값들

을 기계의 하드웨어 구현으로 가능하게 할 수 있다.

이런 목적에서 벡터 데이터에 대한 모든 것(기본 주소, 크기)을 포함하는 특수한 정보단위로 변수기술어를 필요로 한다. 이러한 변수기술어의 개념은 데이터 악세스에서 잘 알려진 von Neumann 개념의 주소 연속 할당의 원리에 앞서는 기술어 참조할당이라는 새로운 원리를 뜻하게 된다. von Neumann개념 에서는 데이터의 악세스에 앞서 연속주소할당을 위해서는 ALU의 증가기능이 1-차원적으로 수행되어야 하는 것이지만, 이 원리에서는 변수기술어정보에 의해 데이터의 외적구조에 따르는 2-차원적 계수가 일어날 수 있게 설계되는 것이다. 그러므로 기계명령어 형식에는 단순한 스칼라 값의 변수대신에 결과 값 변수가 있어야 하며, 이 변수가 지시하는 변수기술어가 있어야 한다. 매트릭스 구조의 데이터가 메모리에 저장되었을 때에 연속주소로 벡터를 이루며, 이와 같은 벡터 요소는 매트릭스의 외형적인 구조성으로 인하여 앞 단원에서 검토된 주소지정 함수(θ : addressing function)는 인덱스 함수(ζ), 인덱스 구조처리 함수(δ), 리스트 함수($\tau\cdot$), 악세스 함수(α)의 순서로 구성되어 다음과 같이 검토 될 수 있다.

3.2.1 인덱스 함수(ζ : indexing function)

인덱스 함수(ζ)는 외형적인 논리구조의 벡터를 메모리 내의 선형적 구조로 전환시켜 주는 기능을 갖는다. 이러한 수행을 위해서는 외형적인 조건, 즉, 벡터의 필수조건이라 할 수 있는 크기에 의해 다음의 식(6)과 같이 전개시킬 수 있다[14].

$$\begin{aligned} addr[i, j] &= addr+((i-mf)\ (nl-nf+1)+(j-nf)) \\ &= addr-((mf\cdot(nl-nf+1)+nf)+(i\cdot(nl-nf+1)+j)) \end{aligned} \quad (6)$$

mf : 열순의 첫째 인덱스
nf : 행순의 첫째 인덱스
nl : 행순의 마지막 인덱스

여기서 addr-((mf·(nl−nf+1)+nf)는 실기본주소(eba)이므로 mf와 nf를 제로(0)로, nl은 (n-1)로 대입하면 식(3)에 의한 인덱스 함수(ζ)는 다음 식(7)으로 유도된다.

$$addr[i, j]=eba+i\cdot n+j, \quad (7)$$

$$i=0, 1, 2, \cdots, m-1$$
$$j=0, 1, 2, \cdots, n-1$$

인덱스 계수 : 기본적인 구조성(rectangular) 데이터를 크기[m, n]의 2-차원적 구조로 정하고, 첫째로 하나의 행을 읽으려면 상대적으로 열의 계수가 정지상태에서 행의 계수가 일어나고 (i0, i1, i2,···, i(n−1), i∈[0 : m−1]), 둘째로 하나의 열을 읽으려면 상대적으로 행의 계수가 정지상태에서 열의 계수가 일어나야 하고(0j, 1j, 2j,···, (m-1)j, j∈[0 : n-1]), 세째로 대각선읽기는 상대적으로 같은 순서의 계수로 진행되어야 하고 (00, 11, 22,···, (m-1) (n-1)), 네째로 벡터에서 한 요소만을 읽기 위해서는 행과 열에서 각각 하나만의 계수만 이루어지면(i j, i∈[0 : m-1], j∈[0 : n-1])된다. 그 관계식은 다음과 같다.

1. $\{i\in[0 : m-1]\}\wedge\{j \mid n\}$
2. $\{j\in[0 : n-1]\}\wedge\{i \mid n\}$
3. $\{j \mid n\}\wedge\{i \mid m\}$, $m=n$
4. $\{i\in[0 : m-1]\}\wedge\{j\in[0 : n-1]\}$

위의 계수식으로 하나의 2-차원 구조를 갖는 벡터를 읽어내는 기법으로는 모두 4가지가 있으므로, 그 각각에 대한 제어구조가 수반되어야 한다. 매트릭스 요소들을 행으로 읽기 위해서는 j가 n-1까지 한번의 계수가 끝날 때마다 i는 1씩 증가하여, i와 j가 각각 m-1, n-1 일 때 모두 끝나며, 열로 읽기 위해서는 i가 m-1 까지 읽었을 때마다 j가 1씩 증가하여 모두 m-1, n-1 에서 끝나게 된다. 세 번째는 대각선 읽기로 i와 j가 함께 같은 클럭으로 1씩 증가하며 읽는 방법이다. 그 외에 네 번째로 i와 j가 증가하지 않고 단순한 좌표 값(i, j)에 의해 매트릭스 요소 하나를 스칼라 값으로 읽어 내는 기법이다.

3.2.2 구조처리 함수(δ : structuring function)

인덱스 i와 j의 구조처리는 매핑(5)에 해당하는 것으로 인덱스 계수(i, j)가 리스트 함수(τ)의 수행에 앞서서 다음과 같은 행과 열의 인덱스의 순서처리로 수행된다. 행의 구조처리 함수(π)에 행구조계수 π을, 열의 구조처리 함수(π)에 열구조계수 π를 대입하므로서 유도된다.

이 구조처리 함수(π, σ)들은 하나의 행 또는 열 벡터의 인덱스에서 명령어에 주어지는 구조계수(ρr, ρc)에 따라서 벡터 요소들의 인덱스를 선택적으로 구조처리하는 기능이다.

인덱스 구조처리 : 행과 열의 인덱스 i와 j의 전환은 행 구조처리 함수(π)에 행구조계수 ρr를, 열 구조처리 함수(σ)에 열구조계수 ρc을 대입 하므로서 유도된다. 즉, $i\hat{}=\pi(\rho c, c(i, j))$, $j\hat{}=\sigma(\rho r, r(j, i))$으로 표현된다. i와 j는 명령어에 있는 벡터 변수의 행과 열의 인덱스를 의미한다. 행 구조처리 함수 σ에서는 열의 계수 ρr를 줌으로서 하나의 행에서 요소들의 순서(인덱스)를 선택적으로 사용할 수 있게 하기 위함이다. 마찬가지로 열 구조처리 함수 π에서는 행의 계수 ρc을 줌으로서 하나의 열에서 요소들의 순서를 선택적으로 사용할 수 있게 하기 위함이다. 그 선택 함수는 벡터 요소들의 인덱스 주소를 변형 시키므로서 요소의 순서를 바꾸어 악세스 할 수 있는 오퍼레이션들로 구분된다. i와 j가 순서의 변형처리없이 그대로 주소생성에 사용되기 위해서는, 즉, 구조처리 함수(δ)가 아니면, 동일처리(identity) 오퍼레이션으로 간주된다. 여기서 생성되는 인덱스(i, j)가 구조계수(ρr, ρc)[1]와 오퍼레이션(ol)된 값을 sio, sjo라 할 때에,

$$sio = r(j, i)\ ol\ pc, \qquad sio = c(i, j)\ ol\ pr \tag{8}$$

이며, 변형된 행과 열의 인덱스 주소는

$$i\hat{} = \pi(i, j) = sio \bmod m, \qquad j\hat{} = \sigma(j, i) = sio \bmod n \tag{9}$$

으로 된다.

인덱스 주소생성은 크기[m, n]을 갖는 행과 열의 구조변경을 위한 오퍼레이션(8), (9)을 함께 할 수 있어야 한다. 이 벡터 구조 변경을 위한 오퍼레이션은 고급언어 APL의 폴드 오퍼레이션 rotate, take, drop,select, transpose에 해당하는 것으로 하나의 벡터에서 부분적 요소들이 처리 또는 sparse matrice처리 등을 할 수 있게하기 위함이다. 그 외에 벡터의 구조를 변경하지 않으면서 이 과정을 거치는 기능으로는 하나의 벡터에서 행과 열을 그대로 통과시켜 주는 기능으로 identity가 있다.

1) ρr : row parameter, ρc : column parameter.

3.2.3 리스트 함수(τ : listing function)

리스트 함수(τ)는 인덱스 함수(ζ)와 인덱스 구조처리 함수(δ)의 수행에 의해서 생성되는 행과 열의 인덱스를 행 벡터의 크기(n-1), 즉, 행의 수 n과 함께 오퍼레이션하여 단위 벡터의 외형적 구조로부터 선형적인 주소 벡터로 전환해 주는 함수이다.

상대주소 생성 : 매트릭스 구조 벡터[m, n]에서 각각 요소들이 갖는 상대주소(rel-addr)는 i+j에 의한 벡터 데이터 내에서 일련의 위치번호(0, 1, 2, …(m·n-1)) 이다. 구조처리된 인덱스 i^와 j^는 행순, 열순, 대각선순 등의 읽는 방법에 따라서 계수 순위가 달라진다. 행순의 경우 j가 0→1→2→…→n-1로 한번 모듈레이션 하면 i+1 되어 (m-1)까지 계수하고, 열순의 경우 i가 0→1→2→…→m-1로 한번 모듈레이션 할 때마다 j는 j-1되어 (n-1)까지 계수 한다. 대각선순의 경우는 i와 j가 동시 같은 룰럭으로 계수한다. 행순과 열순의 계수는 (i=m-1)∧(j=n-1)에서 끝나고, 후자의 방법에서는 (i=m-1)≡(j=n-1)에서 끝난다. 여기서 인덱스 i^와 j^는 구조처리 함수 π, σ에 의해 변경된 주소로 생성되는 인덱스이다.

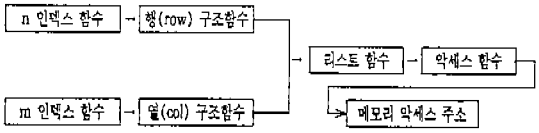
3.2.4 악세스 함수(α : accessing function)

악세스 함수(α)는 리스트 함수(τ)에 의해 벡터 데이터의 실구조인 선형구조, 즉, 상대주소로 생성된 출력에 실기본주소(eba) 또는 벡터의 첫번째 요소의 주소를 가산해 악세스되는 순서로 메모리의 내적 절대주소(abs-addr)를 생성하는 과정이다.

절대주소 생성 : 매트릭스 벡터 A[i, j]에서 요소들이 메모리 내에서 갖는 절대주소는 벡터의 첫 요소가 메모리 내에서 갖는 주소(addr-(mf·(nl-nf+1)+nf))에 해당하는 실기본주소(eba)를 eba+in+j처럼 가산해 줌으로서 얻어질 수 있다.

본 단원의 데이터 악세스 기법, 다시 말해, 메모리 주소지정 기법은 앞에서 검토된 대로 인덱

스 함수(ζ),리스트 함수(τ), 악세스 함수(α)에 의해 구성되는 식(7)의 주소지정 함수($\theta : \zeta \circ \tau \circ \alpha$)의 하드웨어 구현에 의한 주소생성장치의 주소생성에 의해 이루어 진다. 이 악세스 기법은 von Neumann개념에서는 사용자 프로그램 상의 소프트웨어에 의해서 처리되는 것을 입체 계수 원리에 의해 벡터 데이터 주소생성장치(Ag : Address generator)라는 하드웨어의 구현에 의해서 하는 입체 계수 기법이라 할 수 있으며, 벡터 요소의 주소생성전환 순서는 (그림 4)와 같이 표현된다.



(그림 4) 입체 계수 주소생성 순서도
(Fig. 4) Ag flow for dimensional counting

4. 벡터의 구조처리

데이터의 구조처리는 기계명령어에서 변수의 단위가 2-차원적 벡터이기 때문에 행과 열의 인덱스 계수과정에서 일어나야 한다. 이것은 벡터의 구조처리 함수(δ)로서 식(8)에 의해 표현되었다. 이러한 주소계산은 종래의 기계에서는 데이터의 변형처리를 하는 ALU에 의해서 수행되었으며, 시간적으로는 데이터 처리와 병행적 또는 동시적으로 수행될 수 없기 때문에 데이터 처리 시간에 부담이 되고 ALU에도 부담이 되는 것이다.본 개념에서 벡터 요소의 주소계산이란 벡터 악세스 제어부(VAC)의 주소생성과정에서 이루어질 수 있기 때문에 ALU의 부담은 그 만큼 제거시킬 수 있는 것이다.

이러한 식(8), (9)에 의한 벡터 구조처리는 기계명령어에 rotate, take, drop, select, transpose, identity등의 명령어로 존재한다. 그 형식은 오퍼레이션 코드, 결과 변수, 소스변수, 구조계수 등의 4필드로 나누어지며, 전체적으로는 일반 데이터 처리명령어와 같은 길이로 구성된다. 구조 명령어의 해독은 벡터 처리제어부(VPC)에서, 수행은 벡터 악세스 제어부(VAC)에서 처리된다. 여기서 계수의 부호(+, −)에 따라서 앞(−)과 뒤(+)로 구분된다. 식(8)과 (9)에 의하면 r(i, j)는 인덱스 스트림이며, 구조계수 pr 또는 pc는 ol에 의해서 각각의 인덱스에 오퍼레이션 한다. 이 값들은 벡터의 크기 값(m, n)에 의해 모듈레이션되어 인덱스 구조처리를 끝내게 된다.

rotate : 명령어에 제시된 변수(Vd1)에서 인덱스 계수기가 생성하는 행 또는 열의 스트림 인덱스(0, 1, 2,…, n-1)에서 구조계수(pr∨pc) 만큼의 인덱스를 전체 인덱스 뒤 또는 앞으로 옮긴다.

take : 오퍼랜드 변수의 스트림 인덱스(0, 1, 2,…, n-1)에서 명령어에 제시된 구조계수(pr∨pc) 만큼의 인덱스만 계수 한다.

drop : 오퍼랜드 변수의 스트림 인덱스(0, 1, 2,…, n-1)에서 명령어에 제시된 구조계수(pr∨pc) 만큼의 인덱스를 제외하고 계수 한다.

select : 오퍼랜드 변수의 스트림 인덱스(0, 1, 2,… ,n-1)에서 명령어에 제시된 구조계수(pr∨pc)에 해당하는 인덱스만 계수 한다.

transpose : 오퍼랜드 변수의 변수기술어에서 구조의 크기[m, n]이 서로 대치되는 인덱스 생성을 한다.

identity : 스트림 인덱스(0, 1, 2,…, n-1)에서 아무런 변화 없이 그대로 출력시키는 오퍼레이션이다. 이 오퍼레이션은 별도의 명령어 코드가 존재하지 않으며, 벡터의 구조처리가 없을 때에는 항상 사용된다.

위에 언급된 구조처리 오퍼레이션들은 기계명령어에 의해 벡터 악세스 제어부에서 수행되는 것들이다. 스트림 인덱스는 오퍼랜드 벡터의 행 또는 열이며, 명령어에 변수와 함께 주어지는 구조계수(pr∨pc)는 부호(+, −)로서 행 벡터에서 전, 후를, 열 벡터에서는 상,하를 뜻하게 된다.

5. 벡터 데이터 처리 오퍼레이션

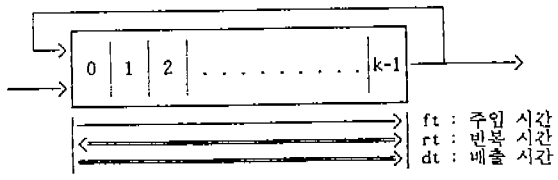
벡터 데이터 처리는 먼저 처리될 오퍼랜드가

스트림이라는 조건이 전제되어야 한다. 이 스트림 상의 벡터 요소들은 같은 유형으로서 하나의 기계명령어에 의해서 처리되는 것이다[10, 15, 16]. 다시 말해, 하나의 오퍼레이션이 필요수 대로 반복수행을 거듭해야 하는 것이다. 벡터 데이터는 메모리에서 스트림으로 버스를 흐르므로 파이프라인 클럭 시간(Pct)은 버스 사이클시간(Bct)과 같아야(Pct=Bct)하며, 전용 파이프라인에 따라 파이프라인 주입시간(Tp : Setup time)이 다음과 같이 다르게 소요된다. 이러한 벡터 처리 오퍼레이션은 그 데이터 처리명령어 오퍼레이션에 소용되는 벡터 단위의 소스 변수의 숫자로 분류하면 다음과 같다.

감축 오퍼레이션(reductive operation) : 1-변수 오퍼레이션으로 하나의 단위 벡터에 내재하는 요소들을 처리하여 스칼라 값으로 결과를 내는 오퍼레이션이다. 다시 말해, 입력으로 들어가는 벡터 요소들이 최종에는 하나의 결과 값으로 나온다는 의미에서 오퍼랜드의 감축을 뜻한다. 이 오퍼레이션에서는 파이프라인 처리의 결과로 먼저 산출된 결과 값들이 스칼라 결과를 내기위해 다시 파이프라인 입력으로 들어가야 하는데서 파이프라인 처리시간의 지연문제가 발생한다. 다시 말해, 반복지연상수 이다. 오퍼레이션 구조는 다음과 같이 표현된다.

1-변수 오퍼레이션 : scalar value=F(a_0, a_1, a_2,…, an-1)

먼저, 결과 값이 스칼라인 경우, 감축 오퍼레이션($A_m=a_0\,oa_1\,oa_2\,o,\cdots,\,oa_{n-1}$)에서 그 수행 시간은 3단계로 나누어 검토 되어야 한다. 스테이지(k)들이 반복수행 직전까지 모두 k개의 오퍼랜드로 채워지기 위해서 처음 k개의 수치값을 변화시키지 않는 중성 오퍼레이션[2]이 수행되고 하며(ft=k·Pct), 다음은 다이애딕 파이프라인에서와 마찬가지로 (n-k)개의 데이터가 감축 오퍼레이션하여 k개로 파이프라인의 각 스테이지에 하나씩 모두 차지 할 때까지의 시간(rt=(n-k)·Pct)과 마지막 단계로 파이프라인의 각 스테이지들에 하나씩 들어 있는 k개의 임시중간 결과 값들이 한 쌍식 파이프라인으로 들어가서 최종 스칼라 값으로 나오는 배출 시간(dt)으로 계산되어야 한다. 여기서 주입 시간과 감축 오퍼레이션 시간은 다이애딕 파이프라인 오퍼레이션 시간(ft+rt=n-k+k)이고, 나머지 k개의 스테이지에 하나씩 들어있는 벡터가 최종 스칼라로 나오도록 회전 배출시간(dt)을 계산하면 된다. 그러므로 길이 n 의 벡터 데이터가 처리[3]될 경우[17, 18, 19, 20], 구성된 스테이지 수(k)와 수행 시간의 관계는 (그림 5)로 표현되며, 그 하드웨어의 절대 수행 시간은 Tpm=ft+rt+dt로 표현된다.



(그림 5) 모나딕 스칼라 파이프라인
(Fig. 5) Time flow of reductive pipeline

이때에 스테이지, k에 각각 하나씩 데이터 k개가 중성 오퍼레이션을 거쳐 들어 있는 상태라고 전제하고, k개의 스테이지로 구성된 파이프라인에서 스칼라 값을 반복 배출 할 때에 발생하는 지연시간은 반복지연상수(dt)와 그 값들의 차이(dd)는 다음의 관계로 나타낸다.

```
k : 2 3 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 ...
dt: 1 4 7 11 15 19 23 28 33 38 43 48 53 58 63 69 75 81 87 93 99 ...
     \ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\ /\
dd: 3 3 4  4  4  4  5  5  5  5  5  5  5  5  6  6  6  6  6  6  6...
     \2/ \ 2  / '       2        / \       2
```

여기서 k=2x+m,(0<m<2·x)으로 표현될 수 있으며 반복지연상수(dt)와 의 관계는 m=0일 때에 식(10)으로 계산된다.

$$dt'=x\cdot 2^{x}-1 \tag{10}$$

여기서 m>0일 때는 반복지연상수의 차, dd=(x+2)·m를 더해 주어 파이프라인의 스테이지 수가 k일 때, 배출 수행에서 지연되는 파이프라인 클럭수, dt는 n>k 인 조건을 전제로 하면 식(11)으로 계산된다.

$$\begin{aligned} dt&=x\cdot(2^{x}+m)+2\cdot m-1\\ &=x\cdot k+2\cdot m-1 \end{aligned} \tag{11}$$

2) neutral operation, 곱셈 a=a×1, 덧셈 a=a+0

3) iterative operation.

그러므로 길이 n의 벡터가 모나딕 오퍼레이션(iterative reduction)에서 파이프라인 스테이지 수(k)에 따르는 시간은 식(12)으로 계산된다.

$$Tpm = ft + rt + dt = (n + x \cdot k + 2 \cdot m - 1) \cdot Pct \quad (12)$$

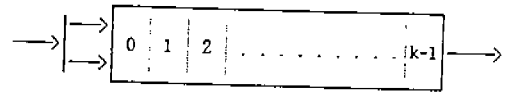
요소별 결과 벡터 오퍼레이션(element results vector operation) : 1-변수 오퍼레이션으로 하나의 소스벡터 단위가 파이프라인 처리되어 소스벡터의 크기와 같은 하나의 결과 벡터를 내는 오퍼레이션 이다. 파이프라인 입력은 오로지 한 방향으로만 들어가며, 파이프라인 상에 반복수행을 위한 되돌림(feedback)이 없다. 오퍼레이션 구조는 다음과 같이 표현된다.

1-변수 오퍼레이션 : vector value $= Fa_0, Fa_1, Fa_2, \ldots, Fa_{n-1}$

요소별 오퍼레이션(element byelement operation) : 2-변수 오퍼레이션으로 2개의 단위 벡터에서 같은 좌표상의 요소들이 차례로 번갈아 스트림으로 공급되어 한 쌍으로 처리되어야 한다. 그러므로 2개의 단위 벡터에 내재하는 요소들의 수는 상대적으로 같아야 한다. 더욱이 처리결과로 나오는 결과 벡터의 요소숫자도 처리되는 한 단위의 소스 벡터의 숫자와 같아야 한다. 오퍼랜드가 처리되는 파이프라인에서는 한 벡터 단위의 요소 수 만큼의 클럭 수로 처리를 끝낼 수 있는 것이 특징이다. 다이애딕(dyadic)기계명령어들의 오퍼레이션이 여기에 속한다. 이 오퍼레이션에는 두 개의 단위 벡터에서 같은 좌표상의 요소들끼리 처리되는 경우와 또 다른 경우는 첫 번째 단위 벡터의 행순 요소와 두 번째 단위 벡터의 열순 요소가 처리되는 경우로 구분된다. 그 외에 하나의 단위 벡터 요소들에 하나의 하나의 스칼라가 오퍼레이션 처리되는 계산이 있다. 이런 경우들은 명령어에서 구분되고 벡터 처리제어부(VPC)에서 제어하게 된다. 오퍼레이션 구조는 다음과 같이 표현된다.

2-변수 오퍼레이션 : $C[i] = F(A[i], B[i])$, $i = 0, 1, 2, \ldots, n-1$ or B : scalar

다이애딕 오퍼레이션($Ci = Ai \circ Bi$, $i = 0, 1, 2, \cdots, n-1$)에서 수행 처리될 데이터는 앞장에서 검토된 벡터 악세스 제어 유니트(VAC)에서 두 개의 주소 생성 장치의 상호 교환적 주소 생성에 의해서 $a_{10}, b_{20}, a_{11}, b_{21}, \cdots, a_{1n-1}, b_{2n-1}$의 순서로 데이터 버스를 통해 버스 사이클(Bct)의 속도로 파이프라인 처리부에 오게 된다. 이렇게 버스를 통해 오는 벡터 요소들은 스트림을 이루어 오지만 파이프라인 처리부에서 다이애딕으로 처리되기 위해서는 디멀티플랙서(DEMUX)에 의해서 $a_{00}, a_{01}, \cdots, a_{m-1\ n-1}$과 $b_{00}, b_{10}, \cdots, b_{k-11-1}$로 분리되어야 하기 때문에 파이프라인 (그림 6)의 사이클(Pct)은 버스 사이클(Bct)의 2 배로 늘어나야 된다($Pct = 2.Bct$).



(그림 6) 다이애딕 파이프라인
(Fig. 6) Dyadic operation pipeline

n쌍 개의 데이터가 k개의 스테이지로 구성된 파이프라인(그림 7)에서 처리될 경우($n \gg k$), 두 개의 벡터가 디멀티플랙스(DEMUX)에서 완전 분류되어, 스테이지 수가 k인 파이프라인에 완전 주입되기까지는 k클럭이 소요되고, 그 다음부터 n개의 데이터가 파이프라인 스테이지의 오퍼레이션을 완전히 끝내기까지는 n클럭이 소요되어야 한다. 따라서 n쌍의 데이터가 k스테이지의 선형 파이프라인에서 완전히 빠져나가기까지는 $(k+n) \cdot Pct$ 가 소요되어야 하므로 식(13)이 성립됨을 알 수 있다.

$$Tpd = (k+n) \cdot Pct \quad (13)$$

여기서 특징은 하나의 데이타 버스에 의해서 두개의 벡터 단위의 혼합된 요소들이 연속 스트림으로 오기 때문에 디멀티프랙스(DEMUX)의 분류시간은 1/2 Pct 이어야 된다. 파이프라인 설계에서 원리적으로 스테이지 수(k)가 많다는 것은 각 스테이지의 단위 수행 시간을 짧게 하는 것으로 데이터를 공급하는 속도와 파이프라인 스테이지 처리 속도에는 최소한 이 조건($Pct \leq 2 \cdot Bct$)이 전제되어 설계되어야 한다.

연계형 오퍼레이션(chained operation) : 2-변수 오퍼레이션으로 요소별 오퍼레이션과 감축

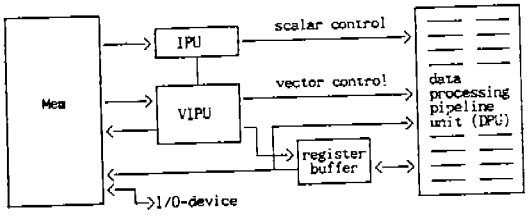
오퍼레이션의 순서적 연결로 처리하는 오퍼레이션이다. 이런 것에는 매트릭스의 내적 계산(inner product)이 있다. 내적 계산의 경우 두 가지 오퍼레이션(C[k,m]=A[k, l]ㅁ.o B[l, m])을 순서적으로 하여야 한다. 여기서는 내적 계산을 위한 데이터 처리장치가 따로 있는 것이 아니고, 2가지 오퍼레이션중 처음 것은 요소별 처리장치(o)를 거쳐서 처리하고, 두 번째는 감축 처리장치(ㅁ)를 거쳐서 수행 처리된다. 오퍼레이션 구조는 다음과 같이 표현된다.

2-변수 오퍼레이션 : $C[k, m]=F_2(F_1(A[k, l], B[l, m]))$

이와 같이 데이터 처리장치(DPU)는 사용되는 단위 벡터의 수로 보면 크게 4 종류로 구분될 수 있으나, 파이프라인 스테이지의 수행기법에 의하면 크게 두 가지로 분류되며, 같은 스테이지 k와 같은 벡터 요소수 n이 처리될때에라도 오퍼레이션 Tpm과 Tpd에서 처럼 주입시간이 달라진다.

6. 벡터 계산기의 조직

본 벡터 컴퓨터는 벡터 처리개념에 의한 벡터 처리지향 설계이다. (그림 7)에서 볼 수 있듯이 [15, 21], 명령어 처리부(IPU)와 데이터 처리부(DPU)만으로는 재래식 스칼라 처리 컴퓨터의 개념(von Neumann machine)이라 할 수 있다. 벡터 처리지향 개념의 벡터 처리부는 스트림 데이터 처리를 위해서 파이프라인 처리 유니트로 설계된다. 파이프라인의 스트림 처리를 위해서는 메모리의 벡터 요소들도 스트림으로 악세스 되어야 한다. 이것을 위해 벡터 악세스 제어부(VIPU)가 부가되는 것이다. 그러나 여기서 벡터



(그림 7) 벡터 컴퓨터의 조직구조
(Fig. 7) Organization of vector computer

처리를 위한 제어 유니트(VIPU)와 von Neumann 기계(IPU 와 파이프라인)는 back-end와 front-end의 관계가 아니고, 벡터와 스칼라 처리를 조직의 기능면에서 겸비한 하나의 stand-alone 형의 설계라 할 수 있다. 하나의 시스템으로서 유니트 별 기능들은 다음과 같다.

Sahni의 설계[20]에 의하면 벡터 처리제어부(VPC)와 벡터 악세스 제어부(VAC)로 분리되어 벡터 제어부는 벡터 명령어처리를 하고, 벡터 악세스 제어부는 단순한 벡터데이터의 악세스를 하는 것으로 기술되었다. 그러나 오퍼레이션 원리에 기술한 것과 같이 벡터 처리에는 1). 벡터 데이터 처리 오퍼레이션과 2). 벡터 구조처리 오퍼레이션이 수행될 수 있음을 알 수 있다. 이러한 데이터의 구조처리는 이미 APL의 필드오퍼레이션으로 소프트웨어적으로 수행되고 있으나, 본 설계에서는 데이터 악세스를 위한 벡터 주소생성과정에서 벡터의 구조처리가 수행되도록 (그림 5)의 하드웨어 구현하는 것이다.

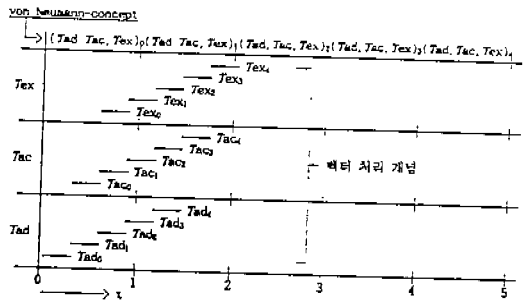
Star-100에서는 메모리와 프로세서 사이의 Stream Unit에 register file의 수학적 처리에 의해서 모든 소스오퍼랜드와 결과를 위한 주소지정이 수행되고, TI-ASC에서는 프로세서 내에 있는 MBU(memory buffer unit)가 메모리와 수학처리 파이프라인 사이의 인터페이스로서 입출력 X, Y, Z로 각각 8개의 레지스터를 갖고 고작 8까지의 증가로만 할 뿐이다. IBM/3838에서는 프로세서와 Bulk memory사이에 있는 데이터 전송제어부(DTC)가 제어 프로세서로부터 받은 벡터 주소계수에 따라서 Bulk memory와 작업 메모리를 악세스하며 소스오퍼랜드를 작업 메모리로 옮겨준다. 여기서 작업 메모리는 데이터 처리를 위한 대기장소, 즉, 버퍼레지스터의 역활이다. Cray-1에서는 8개의 벡터레지스터뱅크가 각각 64개의 레지스터로 구성되어 벡터오퍼레이션은 벡터레지스터로부터 벡터레지스터까지로 수행되기 때문에 메모리악세스는 64개 단위로만 수행되는 특징이 있다[2]. 이상에서 검토되었듯이 레지스터간 오퍼레이션 기계에서 레지스터 용량의 제한성과 메모리간 오퍼레이션 기계의 악세스 충돌로 인하여 벡터의 구조처리 오퍼레이션이 불가능한 것이다. 이런 구조처리는 벡터의 크기에 따

라서, 레지스터 용량의 제한성없이 소스데이터가 스트림으로 나올때에 가능한 것이다.

벡터 명령처리부(VIPU)는 벡터 처리제어부와 벡터 악세스 제어부로 구성된다. 벡터 처리제어부는 벡터 오퍼레이션의 시간제어와 처리결과의 새로운 변수기술어와 그 외의 벡터 악세스 제어부와 데이터 처리부에 지시를 보낸다. 벡터 악세스 제어부는 벡터 처리제어부로부터 받은 변수기술어에 제시된 변수의 크기와 메모리의 벡터 기본주소에 의해 악세스 주소생성을 한다. 이 때에 명령어가 구조처리 명령어이면 인덱스 생성과정에서 구조처리 한다. 벡터 주소의 생성은 벡터의 요소들을 메모리로부터 악세스하여 벡터 데이터 처리부(DPU)로 스트림을 이루어 보내준다. 더불어 결과 변수기술어의 생성에 의해 결과 벡터들이 메모리에 저장 될 결과주소생성도 한다.

7. 결 론

벡터 처리개념은 하나의 명령어에 의해서 여러 개의 오퍼랜드가 같은 오퍼레이션으로 반복수행 처리 되어야 하는 것이 특징이다. 다시 말해, SIMD 처리개념 이다. 이러한 수행처리는 재래식 기계(von Neumann type)에서는 프로그램 상의 반복수행 알고리즘과 중앙처리장치 내의 인덱스 레지스터 값의 증감(+, −)에 의한 데이터 주소계산에 의해 이루어 졌다. 이러한 프로그램 상의 반복수행 알고리즘과 데이터의 메모리 주소지정을 위한 인덱스 레지스터 값의 계산은 반드시 그 데이터 처리에 앞서서 수행되어야 하며, 그러한 이유로 그 계산의 전체시간은 처리되어야 하는 단위 벡터 요소의 숫자에 비례한다고 볼 수 있다. 예를 들어 n개의 요소들로 구성된 벡터가 처리되기 위해서 필요한 시간은 요소의 메모리 주소계산 시간 Tad(addressing time), 그 벡터 요소를 가져오는 시간 Tac(accessing time), ALU에서 그 요소들의 처리시간 Tex(execution time)의 시간이 필요하며, 전체적으로 n개를 위해 (Tad+Tac+Tex)·n 의 시간이 소요되어야 하는 것이다. von Neumann개념 기계와 본 설계개념의 벡터 처리 수행시간 관계는 (그림 8)에서 표현 되듯이 현격한 시차를 볼 수 있다.



(그림 8) 두 설계 개념의 벡터 처리 시간 소요 비교
(Fig. 8) Processing time figure

이러한 세 가지 수행시간대가 반드시 시간대 순서로 이어져야 하며[4], 거기에 데이터 요소의 배수 시간으로 수행되어야 한다는 것은 전체 수행의 장시간을 요하는 것이다. 본 설계에서는 (Tad·n)시간대와 (Tac·n)시간대, (Tex·n)시간대를 다중 중복시켜 수행하므로서 벡터 처리의 전체 수행시간을 단축시키려는 것이다. 이러한 연속적이고 순차적인 세 가지 시간대에서 Tad와 Tex가 von Neumann개념의 기계에서는 데이터 처리부(ALU)에서 매 오퍼레이션 마다 이루어지므로 시간적으로 수행의 부담이되는 것이다. 이러한 ALU부담을 제거하기 위한 기법으로 지금까지의 프로세서 내에서 이루어지는 데이터 주소계산 개념을 프로세서 외부에 설계하여 프로세서의 ALU에서 수행되는 수행시간 Tac와 중복되도록 하면 벡터 수행에서 벡터 데이터 주소계산 시간은 감면될 수 있는 것이다.

4) sequential

참 고 문 헌

[1] G. S. Almasi & A.Gottlieb, Highly Parallel Computing, The Benjamin/Cummings Publishing Company, Inc. 1989. pp.98-99.

[2] K.Hwang and F.A.Briggs, Computer Architecture and Parallel Processing, McGraw-Hill. 1984. pp.233-301.

[3] D.J.Kuck, and R.A.Stokes,"The Burroughs Scientific Processors(BSP)," IEEE Trans. on Comput.vol.c-31,NO.5,May,1982.

[4] S. P. Su and K.Hwang, "Multiple Pipeline Scheduling in Vector Super-computers." in Proc.1982, Int.Conf.on Parallel Processing. IEEE C. S. P.

[5] S. Sahni, "Scheduling Multipipeline and Multiprocessor Computers," IEEE Trans. on Comput. , vol. c-33, NO.7, July, 1984.

[6] R. M. Russel, "The Cray-I Computer System,"Cray Research, Inc. , Tutorial Advanced Computer Architecture, IEEE Computer Society, Order NO.667. 1986.pp. 15-24.

[7] L. Hennessy & D. A. Patterson. Computer Architecture A Quantitative Approach, Morgan Kaufmann Publishers Inc. 1990. pp. 250-400, 403-431.

[8] C. Ghezzi and M.Jazayeri, Programming Language Concepts, John Wiley & Sons INC. 1982, pp. 36-42.

[9] E. Horowitz and S. Sahni, Fundamentals of Data Structures, PITTMAN, 1976, pp. 40-66.

[10] G. J. Myers, Advances in Computer Architecture, 2ND EDITION. John Wiley & Sons, 1982. PP.29-32.

[11] G. Wiederhold, Database Design, 2nd, McGraw-Hill, 1983. pp. 16-17.

[12] D. E. Knuth, The Art of Computer Programming, vol.1.,Chapt.2,Addison-Wesley, Reading/Mass. 1975(2nd. ed).

[13] F. Ayres, Theory and Problems of Matrices, schaum's outline series, McGraw-Hill, Inc. 1974. pp. 1-3.

[14] J. F. Wakerly, Microcomputer Architecture and Programming, John Wiley & Sons, 1981, pp.195-227.

[15] P. M. Kogge, The architecture of pipeline computer, McGraw-Hill,1981, pp.134-173.

[16] D. J. Kuck, The Structure of Computers and Computations, vol. 1. New York : Wiley, 1978. pp. 257-258.

[17] G. S. Sohi, "Instruction Issue Logic for High-Performance, Interruptable,Multiple Functional Unit, Pipelined Computers," IEEE Trans. on Comput. vol. c-39, NO. 3, Mar. 1990.

[18] L. M. Ni & K. Hwang, "Vector-Reduction Techniques for Arithmetic-pipelines," IEEE Trans. on Comput. , vol. c-34, NO.5, May, 1985.

[19] H. V. Jagadish, R. G. Mathews, T. Kailath, and J. A. Newkirk, "A Study of Pipelining in Computing Arrays," IEEE Trans. on Comput. vol. c-35, NO. 5. May, 1986.

[20] H. J. Sips and H. Lin, "An improved Vector-Reduction Method,"IEEE Trans. on Comput. vol. c-40. NO. 2, Feb. 1991.

[21] S. Sahni, "Scheduling Multipipeline and Multiprocessor Computers", IEEE Trans. on Comput. , vol. c-33, No. 7. July 1984.

조 영 일

1970년 고려대학교 졸업
1980년 서 백림 공대 컴퓨터 공학과 졸업
1985년~현재 한림대학교 컴퓨터 공학과 부교수
관심분야 : 컴퓨터 구조설계, 병렬처리 시스템.

박 장 춘

1965년 연세대학교 전기공학과 졸업
1982년 캘리포니아 주립대 석사
1990년 연세대학교 공학박사
1988년~현재 건국대학교 전자계산학과 교수
관심분야 : 컴퓨터 구조설계, 이미지 프로세싱, 병렬처리 시스템, 패턴 인식.