Data Flow 시스템에서 구조체 분산 처리 방식

°맹 성 열, 현 운 봉, 하 영 호, 임 인 칠
한 양 대 학 교

A Structure Distributed Processing Method in Data Flow Systems

S.Y.Maeng, W.M.Hyun, Y.H.Ha, I.C.Lim
Hanyang University

ABSTRACT

This paper proposes a method which distributes the structure data represented by a tree and handles it. To distribute and handle the structure data, this method partitions a structure data and distributes the partitioned structure in multiple processing element and allocates the partitioned structure.

Each processing element includes the structure memory to store the partitioned structure and the structure controller to handle efficiently the distributed structure.

As the structure is distributed and is stored in the structure memory and is handled by the structure controller, the processing time is reduced.

I. 서 론

현대 사회가 정보화 시대로 변모함에 따라 많은 처리 시간을 요하는 기상예보, 신호처리, 화상처리, 행렬계산 등의 분야에서 더욱 신속한 처리 능력을 갖는 특수한 목적의 컴퓨터 시스템 개발이 요구되어 왔다. 컴퓨터의 처리 능력을 증대시키기 위하여 컴퓨터 구조는 그대로 유지하면서 프로세서의 속도를 증대시키는 기술을 개발하여 왔으나 기술의 한계성 때문에 현저한 처리 속도의 증대를 이룩하지 못하였다.

반면에, VLSI 기술이 발전함에 따라 동일하고 복잡한 많은 양의 Device를 적은 비용으로 생산할 수 있게 되었다. 따라서 비동기적으로 동작하는 수 천개의 프로세서를 사용하여 계산을 병렬 처리(Parallel Processing)할 수 있게 되므로써 처리 시간의 감소가 이루어졌다.[1-2]

그러나, 병렬 처리를 하는 컴퓨터들이 프로그램 카운터와 공유 메모리를 근간으로 하는 Von Neumann 방식을 채택하고 있어서, 명령 실행이 순서적으로 이루어지고 메모리를 억세스할 때에 충돌이 발생하는 단점을 가지고 있다.[3] 이러한 문제점을 해결하기 위하여 데이타 구동(Data-Driven) 방식과 요구 구동(Demand-Driven)방식이 제안되었다.[5] 데이타 구동 방식으로는 Data Flow기계가 있고, 요구 구동 방식으로는 reduction기계가 있다. 그러나, Data Flow 기계는 계산 방식의 특성(함수성) 때문에 구조체 데이터(Structure Data)를 처리하는데 문제점이 있다.[8]

본 논문에서는 이러한 문제점을 해결하기 위하여 처리요소(Processing Element)에 구조체 메모리와 구조체 제어기(Structure Controller)를 부가하여 구조체를 전담하여 처리하게하고, tree로 표현된 구조체 데이타의 순서성(Sequentiality)을 개선하기 위한 분산 처리 방식을 제안한다. 분산 처리를 하기 위하여 tree를 분할(partition)하고, 분할된 subtree들을 여러 개의 처리 요소에 분산 저장한다. 또한 분산된 구조체를 효율적으로 처리하기 위하여 구조체 처리를 전담하는 구조체 제어기를 구성한다. 구조체를 분산 처리함으로서 처리시간이 감소됨을 보인다.

II. 구조체 처리에서의 문제점

Data Flow에서 정수, 실수, boolean, 문자열 등과 같은 elementary 데이타는 메모리에 저장되지 않고 토큰에 수반되어 사용된 후에 소멸된다. 그런데 elementary 데이타의 집합체인 구조체 데이타(Structure Data)를 그대로 토큰에 실어서 사용하면 토큰크기가 매우 커지고 토큰을 전송하는데 많은 시간이 소비되어 시스템 성능을 저하시키는 요인이 된다. 따라서 구조체를 구조체 메모리에 저장하고 토큰은 그 구조체를 지정하는 지정자(pointer)만을 수반한다.

그리고 Data Flow에서 출력은 실행시 기계의 상태에 상관없이 단지 입력의 함수이기 때문에 한번 생성된 구조체는 프로그램 수행이 끝날 때까지 변경없이 존재하여야 한다. 따라서 구조체를 변경하려면 새로운 데이타를 포함하는 새로운 구조체가 생성되어져야 한다. 특히, 구조체 중에서 한두개의 데이타를 변경하고자 할때 그 데이타를 제외한 모든 데이타들은 복사되어야 하기 때문에 복사 시간과 메모리 영역의 낭비는 심한 overhead가 되어 시스템 성능을

저하시키는 요인이 된다. 따라서, 과다 복사를 방지하기 위하여 구조체를 공유시키고 공유 여부를 나타내기 위하여 참조 카운터(reference counter)를 사용한다.

Ⅲ. 구조체 데이타 처리

본 논문에서는 구조체 데이타를 효율적으로 관리하기 위하여 처리 요소에 구조체 데이타를 격납하는 구조체 메모리와 구조체를 전담하여 처리하는 구조체 제어기를 부가한다. 이러한 처리 요소의 구성도를 그림 1에 나타내었다.

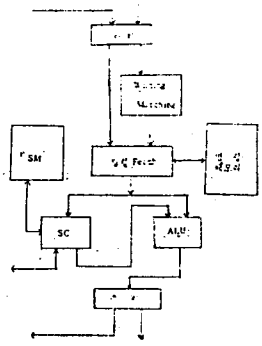


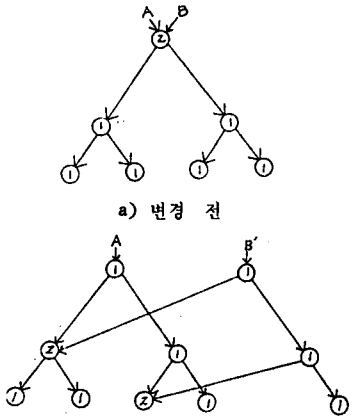
그림 1. 처리 요소 구성도

또한, 일반적인 경우의 구조체를 간단하게 처리하기 위하여 구조체를 tree로 표현한다. Tree에서 각 노드는 고정된 수의 가지를 가지며 가지의 순서는 고정되어 있다. 그리고 tree는 지정자를 갖는 가지 노드와 데이타 값을 갖는 leaf 노드들로 구성된다. 구조체에서 데이타의 index는 각 레벨에서 적당한 가지를 선택하는데 필요한 선택자(selector)로 구성된다. 따라서 tree형태로 저장된 데이타에 대한 동작은 선택자를 이용하여 실행된다. 예를들어 데이타 수가 8이고 가지 수가 2인 구조체는 깊이가 3인 2진 tree로 표현할 수 있다. 이 경우 선택자는 3비트의 비트열로 표현될 수 있으며, 이 때 최상위 비트는 root를 나타내고 최하위 비트는 leaf를 나타낸다. 그리고, 각 노드는 그 노드를 지정하는 지정자의 총수를 나타내는 참조 카운터를 갖는다. 데이터를 변경할 때 참조 카운터가 1인 노드는 복사없이 직접 변경하고 참조 카운터가 1이상인 노드는 직접 변경할 수 없으므로 새로운 노드를 하나 생성하여 변경된 값을 저장하고 참조 카운터를 1로 한다. A, B의 두 지정자에 의하여 공유되어 있는 구조체에 대한 데이타 변경 예를 그림 2에 나타내었다. Tree에서 한 노드는 구조체 메모리내의 한 단어(word)에 mapping된다. Tree에서 가지 노드는 한 단어가 참조 카운터와 지정자 쌍으로 구성되고 leaf 노드는 참조 카운터와 값으로 구성된다. 2진 tree에서 메모리 내의 한 단어의 형태는 다음과 같다.

가지 노드 : 〈rc, rp, lp〉

Leaf 노드 : 〈rc, data, value, "\"〉

여기서 rc는 참조 카운터를 나타내고 rp는 오른쪽 지정자를 그리고 lp는 왼쪽 지정자를 나타낸다.



a) 변경 전

b) 변경 후

그림 2. 공유된 데이타 변경

Ⅲ-1. 구조체 데이터의 형태

본 논문에서는 구조체를 두개로 분류하여 처리한다.

(1) 고정 구조체

구조체를 한 처리 요소에 저장하고 그 처리 요소가 구조체를 전담하여 처리할 수 있도록 한 구조체로 데이타 수가 작은 구조체가 여기에 속한다.

(2) 분산 구조체

데이타 수가 매우 많고 동시에 처리할 수 있을 때, tree 구조의 순서성 때문에 하나의 처리 요소가 한 구조체를 전담하여 처리하면 효율이 떨어진다. 이를 해결하기 위하여 tree를 4개의 subtree로 분할하여 각 처리 요소에 할당한다. 따라서 분할 저장된 분산 구조체를 동시에 처리하는 것이 가능하다. 이때 분산에 관한 정보는 구조체 표현 코드(Structure Description Code : SDC)에 저장되어 있다. 구조체의 분산 여부는 데이터의 수, 동시 처리할 수 있는 데이터 수, 메모리 부하의 균등성에 따라 결정된다. 즉, 데이터 수가 적고 메모리 참조가 빈번하지 않은 구조체는 한 처리요소 내에서 처리하고 데이터 수가 많고 동시처리 가능한 데이터 수가 큰 구조체는 여러 처리 요소에 분할하여 처리한다.

구조체를 효율적으로 처리하기 위하여 구조체 메모리를 두 부분으로 나누었다. 그 중 한 부분은 고정 구조체를 격납하고 다른 한 부분은 분산 구조체를 저장한다. 그리고 구조체에 대한 동작을 수행하는 제어기도 두 부분으로 나누어 각 구조체를 제어하게 하였다. 구조체 메모리와 구조체를 처리하기 위한 구조체 제어기의 구성을 그림 3에 나타내었다.

여기서 구조체 제어기가 구조체를 처리하는데 있어서 필

요한 동작 패킷은 다음의 두가지로 나타낼 수 있다.

1) SELECT

Selector에 의해 지정된 데이타를 읽는다. SELECT 패킷의 형은 다음과 같다.

〈f, SEL, Array, Selector, Destination〉

여기서, f는 고정 구조체와 분산 구조체를 구분하기 위한 flag이고, SEL은 op코드를 나타내고, Array는 지정된 구조체의 head 주소를 나타낸다.

2) APPEND

Selector에 의해 지정된 곳에 값을 저장시킨다. APPEND 패킷의 형은 다음과 같다.

〈f, APP, Array, Selector, Value〉

구조체 패킷이 제어기에 입력되면 입력부는 패킷내의 flag를 검색하여 입력 패킷이 고정 구조체를 처리하기 위한 패킷이면 고정 구조체 제어기(Fixed Structure Controller : FSC)로 보내고, 분산 구조체를 처리하기 위한 패킷이면 분산에 관한 정보를 가지고 있는 구조체 표현 코드(SDC)를 검색하여 유효 주소를 계산한 후에 구조체 패킷 내의 구조체 주소영역을 재구성하여 분산 구조체 제어기(Distribute Structure Controller : DSC)로 보낸다. 고정 구조체 제어부는 고정 구조체에 대한 동작을 수행하고, 분산 구조체 제어기는 분산 구조체에 대한 동작을 수행한다. 출력부는 실행 결과 패킷을 순서대로 출력시킨다.

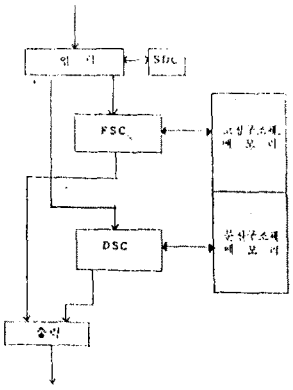


그림 3. 구조체 메모리와 구조체 제어기의 구성도

Ⅲ-2. 구조체 동작

구조체 제어기는 동작 패킷을 받아 다음과 같은 메모리동작을 행한다.

1) FET(addr)

주소에 의해 지정된 단어를 읽어들인다. 이때, 귀환되는 패킷은 아래와 같은 형을 갖는다.

귀환 패킷 : RTR(addr, data, value, rc)

2) FET+(addr)

지정된 주소의 참조 카운터(reference counter : rc)를 증가시키고 지정된 단어를 읽는다.

귀환 패킷 : RTR+(addr, data, rc+)

3) FET−(addr)

참조 카운터를 감소시켜 지정된 단어를 읽는다.

귀환 패킷 : RTR−(addr, data, rc−)

4) UPD(addr, data, rc)

지정된 단어에 데이타 값과 참조 카운터 값을 write한다. 이때, 응답 패킷은 필요치 않고 다만 인지 신호만 보내면 된다.

또한, 구조체 제어기는 메모리 셀을 효율적으로 사용하기 위하여 garbage collection을 행한다. 즉, 참조 카운터 값이 0이 되면 그 셀은 더 이상 필요가 없다. 따라서, 그 셀의 지정자가 free list에 등록되어 나중에 셀이 필요한 구조체를 위해 쓰인다. 그리고, 새로 절점이 생성되므로써 셀이 필요한 경우, 구조체 제어기는 free list에서 셀 지정자를 하나 취하여 새로운 절점에 할당한다.

구조체의 데이타를 읽어들이거나 변경하는 알고리즘은 다음과 같다.

1) SELECT 알고리즘

단계 1 : Selector string을 검색한다.

단계 2 : Selector bit에 따라 FET명령을 보낸다.

단계 3 : 응답이 오면 마지막 leaf인가를 검색하여 leaf에 도달했으면 단계 4로 가고 아니면 단계 1로 간다.

단계 4 : FET+를 보내 그 절점의 참조 카운터를 증가시키고 FET−를 보내 본래의 구조체의 참조 카운터를 감소시킨다.

2) APPEND 알고리즘

단계 1 : APPEND해야할 셀에 FET명령을 보내고 귀환 패킷의 참조 카운터(rc)를 검색한다.

rc = 1이면 단계 2로 가고

rc > 1이면 단계 3으로 간다.

단계 2 : UPD명령을 보내 데이타 값을 변경한다.

단계 3 : free list에서 셀 주소를 하나 취하여 그 주소에 UPD명령을 보내 데이타 값을 써넣고 새로운 구조체를 구성한다.

Ⅲ-3. 분산 처리의 적용 예

다음의 프로그램을 고정 구조체와 분산 구조체에 각각 적용하였다.

for (i = 1 ; 1 = < 16 ; i++)

Y[i] = X[i] + (3 * i)

고정 구조체로 처리할 경우, 배열 X를 하나의 처리요소에 저장하고 그 처리 요소내의 구조체 제어기가 그 구조체를 전담하여 처리하게 하고, 분산 구조체로 처리할 경우, 배열 X를 4개씩 4부분으로 분할하여, 4개의 처리 요소에 분산 저장시키고 각 구조체 제어기가 분산 표현 코드

를 참조하여 처리하게 하였다.

배열 X를 한 처리 요소 내에 저장하여 그 처리 요소가 전담하여 처리하는 경우와 배열 X를 균등하게 4부분으로 분할한 후에 분할된 배열들을 여러 처리 요소에 분산하여 처리하는 경우의 처리 시간을 표 1에 나타내었다.

	시 간(unit time)
고정 구조체	16
분산 구조체	4

표 1. 적용 예의 결과

결과에 나타나 있듯이, 동시에 처리할 수 있는 구조체를 분산하여 처리함으로서 처리 시간의 감소가 이루어 졌다.

Ⅳ. 결 론

본 논문에서는 구조체 데이타를 효율적으로 처리하기 위하여 처리 요소에 구조체 메모리와 구조체 제어기를 부가하였고 구조체 데이타를 고정 구조체와 분산 구조체로 나누어 처리하는 방법을 제안하였다. 즉, 구조체 데이타의 element 수가 작은 경우에는 고정 구조체로 처리하였고 동시에 처리할 수 있는 element 수가 많은 경우에는 분산 구조체로 처리하였다. 그리고 구조체에 대한 동작을 수행하는 구조체 제어기를 구성하였다.

동시 처리가 가능한 구조체 데이타들을 분산하여 처리함으로서 처리시간이 감소되었다.

참 고 문 헌

1. K. Hwang and F.A. Briggs, Computer Architecture and Parallel Processing, New York : McGraw-Hill, 1984.
2. R.W. Hockney and C.R. Jesshope, Parallel Computers, Bristol, Great Britain : Adam Hilger, 1981.
3. Arvind and R.A. Iannucci, "A Critique of Multiprocessing Von Neuman Style", Proc. 10th Ann. Symp. Computer Architecture, pp. 426-436, June 1983.
4. J.B. Dennis, "Data Flow Supercomputers", IEEE Computer, Vol. 13, pp. 48-56, Nov. 1980.
5. P.C. Treleaven, D.R. Brownbridge and R.P. Hopkins, "Data Driven and Demmand Driven Computer Architecture", ACM Computing Surveys, Vol. 14, No. 1, pp. 93-143, Mar. 1982.
6. J. Rumbaugh, "A Data Flow Multiprocessor", IEEE Trans. on Computer, Vol. C-26, No. 2, pp. 138-146, Feb. 1977.
7. K. P. Gostelow and R. E. Thomas, "Performance of a Simulated Data Flow Computer", IEEE Trans. on Computer, Vol. C-29, pp. 905-919, Oct. 1980.
8. J. L. Gaudiot, "Structure Handling in Data Flow System", IEEE Trans. on Computer, Vol. C-35, No. 6, pp. 489-502, June. 1986.
9. W. B. Ackerman, "A Structure Processing Facility for Data Flow Computers", Proc. 1978 Int. Conf. Parallel Processing, pp. 166-172, Aug. 1978.