논문 2006-43SD-2-8

Content-Addressable Memory를 이용한 확장 가능한 범용 병렬 Associative Processor 설계

(Design of a scalable general-purpose parallel associative processor using content-addressable memory)

박 태 근*

(Taegeun Park)

요 약

일반 컴퓨터에서 중앙처리장치와 메모리 사이의 병목현상인 "Von Neumann Bottleneck"을 보이는데 본 논문에서는 이러한 문제점을 해소하고 검색위주의 응용분야에서 우수한 성능을 보이는 Content-addressable memory(CAM) 기반의 확장 가능한 범용 Associative Processor(AP) 구조를 제안하였다. 본 연구에서는 Associative computing을 효율적으로 수행할 수 있는 명령어 세트를 제안하였으며 다양하고 대용량 응용분야에도 적용할 수 있도록 구조를 확장 가능하게 설계함으로써 유연한 구조를 갖는다. 12 가지의 명령어가 정의되었으며 프로그램이 효율적으로 수행될 수 있도록 명령어 셋을 구성하고 연속된 명령어를 하나의 명령어로 구현함으로써 처리시간을 단축하였다. 제안된 프로세서는 bit-serial, word-parallel로 동작하며 대용량 병렬 SIMD 구조를 갖는 32 비트 범용 병렬 프로세서로 동작한다. 포괄적인 검증을 위하여 명령어 단위의 검증 뿐 아니라 최대/최소 검색, 이상/이하 검색, 병렬 덧셈 등의 기본적인 병렬 알고리즘을 검증하였으며 알고리즘은 처리 데이터의 개수와는 무관한 상수의 복잡도 O(k)를 갖으며 데이터의 비트 수만큼의 이터레이션을 갖는다.

Abstract

Von Neumann architecture suffers from the interface between the central processing unit and the memory, which is called "Von Neumann bottleneck". In this paper, we propose a scalable general-purpose associative processor (AP) based on content-addressable memory (CAM) which solves this problem and is suitable for the search-oriented applications. We propose an efficient instruction set and a structural scalability to extend for larger applications. We define twelve instructions and provide some reduced instructions to speed up which execute two instructions in a single instruction cycle. The proposed AP performs in a bit-serial, word-parallel fashion and can be considered as a 32-bit general-purpose parallel processor with a massively parallel SIMD structure. We design and simulate a maximum/minimum search, greater-than/less-than search, and parallel addition to verify the proposed architecture. The algorithms are executed in a constant time O(k) regardless of the number of input data.

Keywords : associative processor, content addressable memory, SIMD

I. 서 론

일반적인 개념에서 범용 컴퓨터는 중앙처리장치(CPU)와 메모리로 구성되는 Von Neumann 구조를 중심으로 발전하여 왔다. 최근 컴퓨터 및 반도체 기술의 발전으로 메모리의 용량은 수 백 메가바이트 범위에 이르고, 단일 프로세서 구조의 경우에 하나의 CPU가 많은 양의 메모리를 한번에 하나씩 읽어 처리해야 하기

* 정회원, 가톨릭대학교 정보통신전자공학부
(Department of Information, Communications, and Electronics Engineering, The Catholic University of Korea)

※ 본 연구는 2005년도 가톨릭대학교 교비연구비의 지원으로 이루어졌음.
접수일자 : 2005년6월3일 수정완료일 : 2006년1월3일

때문에 CPU 자체의 처리성능과 무관하게 전체 시스템의 성능이 저하될 수 있다. 즉, 전체 시스템의 처리성능은 CPU의 연산 속도보다는 메모리와 CPU간의 데이터 및 명령어 전송속도에 좌우된다. 이 때 발생하는 메모리와 CPU 사이의 병목현상을 "Von Neumann Bottleneck"이라 한다. 이러한 병목현상을 개선하기 위하여 많은 병렬 구조들이 제안되었다. 다수 개의 프로세서로 구성된 하나의 시스템에 복잡한 Interconnection network을 이용하여 데이터와 프로세서 사이의 성능 처리 개선이 모색되었는데 이때 사용된 프로세서는 대부분 범용의 고성능 CPU이며 성능 면에서는 많은 개선을 나타내지만 시스템의 복잡도와 가격 면에서 단점을 드러내고 있다.

검색위주의 응용분야에서 Content-Addressable Memory (CAM)는 효과적인 해답이 될 수 있다. 메모리에 저장된 데이터에 접근할 때에 주소를 이용하여 데이터에 접근하는 RAM과는 달리 CAM은 저장된 내용을 검색함으로써 데이터에 접근하고 처리한다. 따라서 CAM 블록에는 어드레스 디코더가 없으며 모든 워드를 대상으로 검색과정이 진행되므로 자체적으로 병렬처리의 의미가 있다. RAM에 저장된 데이터를 검색하려면 CPU는 RAM의 주소를 하나씩 증가시키면서 데이터를 모두 읽어 내어 비교함으로써 가능하지만, CAM을 이용하면 한번의 검색으로 CAM에 저장된 모든 데이터와 동시에 비교가 가능하므로 저장된 데이터의 개수에 무관하게 검색할 수 있다. 이러한 CAM을 바탕으로 각 워드에 단순한 Processing Element(PE)를 첨가하면 SIMD 형태의 병렬 Associative Processor(AP)를 구성할 수 있다. 이 때 PE는 CAM의 검색 결과를 바탕으로 다양한 부울 연산을 수행함으로써 여러 가지 복잡한 검색(query)을 처리할 수 있다. 따라서 이 구조는 수 백~수 천 개의 PE들이 네트워크로 연결된 SIMD 구조로 이해될 수 있으며 이때 내부에 설계되는 콘트롤러는 전체 시스템의 수행을 제어하게 된다. 또한 다양한 응용 분야에 적용되기 위하여 효율적인 명령어 구조가 제안되어야 하며 제안된 구조는 보조 프로세서로서 호스트와 연결되는 일반적인 SIMD 시스템과 같이 사용될 수 있다.[1]

본 연구에서는 Associative computing을 기반으로 하는 효율적인 명령어 세트를 제안하였다. 또한 다양하고 대용량 응용분야에 적용할 수 있도록 구조적인 확장이 가능하게 설계함으로써 유연한 구조를 갖는다. 전체 시스템에서 정의된 명령어 세트가 수행되기에 적합하도록 설계하여 검증하였다. 따라서 본 논문에서 제안된 프로세서는 bit-serial, word-parallel로 동작하며 대용량 병렬 SIMD 구조를 갖는 32 비트 범용 AP이다. 본 논문에서 제안하는 AP는 그 구조가 상대적으로 단순하지만 효율적이고 유연한 구조를 갖고 있으며 하나의 칩에 수천 개의 PE를 내장할 수 있다. 제안된 범용 AP의 구조 및 기능 면에서의 특징은 다음과 같다.

(1) 다양한 응용분야에 적합한 명령어 세트: 유형별로 Shift, Move, Read, Write, Match, SelectNext 관련 명령어 등, 프로그램이 효율적으로 수행될 수 있도록 명령어 셋을 구성하고 연속된 명령어를 하나의 명령어로 구현함으로써 수행시간을 단축하였다.

(2) Bit-selectable static CAM : 32 비트 데이터와 10 비트 태그(Tag) 영역이 모두 비트단위로 선택되어 처리될 수 있도록 설계하였다.

(3) 효율적인 PE 구조 : Response Registers, 부울 연산기(GPLB), MRR(Multiple Response Resolver) 트리, 쉬프트 레지스터 등을 포함한 1 비트 PE 구조를 채택하여 복잡도를 개선하였다.

(4) 확장 가능한 구조 : 수 천 개의 PE가 필요한 응용분야에도 적용할 수 있도록 Cascade 기능과 MRR 트리를 확장할 수 있도록 I/O 인터페이스 설계하였다.

(5) Binary, Quad 모드 지원 : Binary 모드뿐만 아니라 영상처리, 패턴인식, 신경망시스템 등 다양한 분야에서 필요한 Quad 모드(0, 1, *, N)도 추가의 하드웨어 없이 지원할 수 있도록 설계되었다.

본 논문의 제Ⅱ장에서는 기존에 제안되었던 AP들의 구조와 특징을 분석하고 설명한다. 제Ⅲ장에서는 제안된 확장성 있는 범용 AP 구조 및 명령어 등에 대하여 설명하고, 제Ⅳ장에서는 설계 및 모의실험을 분석하고 마지막으로 제Ⅴ장에서는 본 논문의 결론을 제시한다.

Ⅱ. Associative Processor

1. Application-specific Associative Processor

현재까지 Associative Computing을 이용하여 데이터베이스, 인공지능, 패턴인식, 전문가시스템, 영상처리, 네트워크 등 여러 가지 응용분야에서 많은 연구가 진행되어 왔다.[2~4] Storer 등은 이종 비젼 아키텍처(Heterogeneous Vision Architecture)를 위한 AP 시스템을 제안하였다.[5] GLiTCH라고 불리는 AP 칩은 8 비트 영상을 처리할 수 있도록 64 비트 CAM을 내장하고 있으며 명령어를 저장하는 ROM과 단순한 1 비트 ALU를 갖는 PE 그리고 단순한 네트워크와 제어 회로로 구성되어 있다. 초기에 DSP 칩에서 처리된 신호들은 제안된 SIMD

AP 어레이에서 영상처리 및 특징 블록이 추출되고 MIMD 네트워크로 연결된 시스템에서 추출된 데이터들이 병렬로 처리된다. CAM은 데이터 CAM과 서브셋 CAM으로 분리되어 있으며 면적을 줄이기 위하여 동적 메모리의 구조를 갖고 있다. 2048 PE, 4MB의 RAM, 그리고 32개의 데이터 라우팅 칩으로 구성된 제안된 병렬 시스템을 256×256 크기의 영상처리 및 패턴인식에 적용했을 때 기존의 방법보다 우수한 성능을 나타내었다.

Higuchi 등은 인공지능에 적합한 IXM2 병렬 AP 시스템을 제안하였다.[6] 제안된 시스템은 자연어처리를 위하여 설계되었으며 rule-based 방법은 수행속도, 확장성, 질, 규칙의 정의 등의 면에서 많은 문제를 지니고 있지만 CAM 메모리를 기본으로 하는 대용량 병렬 SIMD 방법은 실시간 응용분야에 적합하다. 제안된 시스템은 NTT에서 개발된 AM 칩을 이용하여 256K 배의 병렬효과를 얻었다. 64개의 AP와 9개의 네트워크 프로세서로 구성되어 있다. IXM2는 사실상 주컴퓨터의 제어를 받는 보조 프로세서로서 동작한다. 각 AP는 40 비트의 4K 워드와 17.5MHz IMS T800 트랜스퓨터를 포함한다. NTT AM 칩은 512×40 비트로 구성되었으며 search, write, read, garbage 등 네 가지 유형의 명령어가 수행되며 IXM2 프로그래밍은 단순하며 특별한 소프트웨어 인터페이스 루틴이 필요 없다는 장점이 있다. Louri 등은 방대한 데이터베이스를 고속으로 처리하기 위하여 Optical associative computing을 적용한 시스템을 제안하였다.[7] 제안된 시스템은 광의 장점을 이용하여 word-parallel, bit-parallel 처리 특징을 갖고 있으며 상수시간 내에 데이터베이스 검색을 수행할 수 있다.

2. General-purpose Associative Processor

위에서 설명된 것과 같이 특정한 응용분야에 적합하도록 제안된 구조들 외에 좀 더 다양한 응용분야에 적용될 수 있는 범용의 구조들도 여러 가지 제안되었다. Stormon 등이 제안한 AP 시스템은 비교적 단순한 구조를 갖지만 유연한 구조 때문에 그의 집적도가 높고 따라서 상대적으로 큰 응용문제를 다룰 수 있다는 장점이 있다[10]. 설계된 CRC32256 AP 칩 하나는 256 개의 PE를 포함하고 있으며 확장 가능하도록 설계되었다. 최근에 제안된 bit-parallel block-parallel AP는 입력 벡터와 모든 검색 대상의 벡터 간에 병렬로 연산되며 비트뿐 아니라 블록 간에도 병렬로 처리가 가능함으로써 성능이 개선되었다.[11]

Grosspietsch 등은 CAPRA라는 AP 시스템을 제안하였는데 일반적으로 데이터베이스 검색, 단순한 수리연산, 영상처리 등에 적용할 수 있으며 큰 특징은 구조의 유연성과 테스트기능과 오류내성기능으로 요약할 수 있다.[8] 사용된 CAM은 보통의 RAM처럼 선택이 가능하며 각 워드 당 4 비트의 덧셈기와 쉬프터가 구현되어 있다. 일반적인 AP의 PE에 비하여 복잡한 구조를 갖고 있으며 기능 또한 많이 구현되어 있다. 제안된 시스템은 일반적으로 데이터베이스 검색, 비교적 단순한 수리연산, 영상처리 등에 적용할 수 있다. Tavangarian이 제안한 AP 시스템은 워드단위를 기초로 하는 데이터를 플래그를 기초로 하는 형태로 변환하여 처리한다[9]. 벡터 형태의 플래그는 AP 시스템에서 병렬로 처리됨으로써 높은 효율을 나타낸다. 제안된 시스템은 IC design rule check(DRC), 복잡한 메모리 테스트, 인공 신경망 시스템에 적용되어 우수한 성능을 보여주었다.

Ⅲ. 제안된 범용 Associative Processor 구조

그림 1은 제안된 확장 가능한 범용 병렬 AP의 시스템 구조이다. 데이터 입출력은 32 비트로서 내부에 있는 5 종류의 레지스터에 저장된다. 정의된 명령어는 12 가지이며 입력된 명령어는 on-chip controller에서 디코딩되고 실행된다. 오른쪽의 CAM 블록은 42 비트 단위의 N개의 워드로 구성되어 있고 각 워드는 병렬 산술연산 등 다양한 알고리즘 적용을 위하여 비트별 선택기능을 가지고 있다. 상위 10 비트는 응용 시에 데이터의 구분을 위한 태그 비트로 사용된다.

또한 PE 블록은 CAM을 이용한 검색 결과를 이용하여 다양한 query를 처리할 수 있는 단순한 기능의 1 비트 프로세서이다. 이와 같이 구현된 AP 시스템은 문제 크기에 따라 여러 개의 AP 칩들을 직렬 연결하여 확장 가능하도록 설계되었다. 이 때 인접한 AP 칩들 간에 데이터를 주고받기 위하여 최상위와 최하위 PE의 R1 레

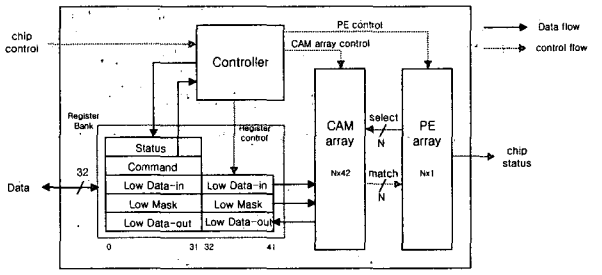

그림 1. 제안된 범용 병렬 Associative Processor 구조
Fig. 1. Proposed general-purpose parallel Associative Processor architecture

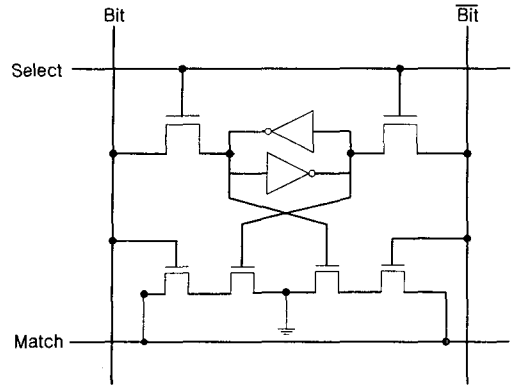

그림 2. static CAM 셀의 구조
Fig. 2. Static CAM cell structure.

지스터를 서로 연결하면 쉽게 PE 용량을 확장할 수 있다. 이 때 뒤에서 설명할 MRR 트리 구조도 확장되어야 하는데 이는 보드레벨에서 간단한 로직을 추가하여 구현할 수 있다.

가. CAM 어레이

CAM은 RAM을 확장한 형태로서 각 셀은 쓰기와 읽기 연산이 가능할 뿐만 아니라 매치연산을 할 수 있다. 아래의 그림 2에서 상위의 인버터 2개와 트랜지스터 2개는 기존의 RAM부분이고, 하위의 트랜지스터 4개가 매치연산을 위한 구조이다. 즉, 셀 내부에 저장되어 있는 값과 외부에서 주어진 값과 비교하여 상태를 나타내는 비교기 역할을 한다. 매치 라인은 매치연산을 수행하기 전에 선 충전(precharge)되며 저장된 값과 외부에서 인가된 값을 비교하여 선택적으로 방전됨으로써 매치를 수행한다. 더 적은 수의 트랜지스터를 이용한 CAM 셀 (5-transistor Dynamic CAM cell[12])이 제안되었지만 동적으로 동작하므로 주의가 필요하여 일반적으로 정적인 CAM 셀 구조가 사용된다. 정적 CAM 셀을 이용하여 하나의 칩으로 만들기에 적당한 1024 워드 x 42 비트 크기의 CAM 어레이를 구성한다.

CAM 어레이는 어드레스가 없으며 워드 선택선이 이를 대신한다. 읽기 동작 시에는 하나의 데이터만을 읽을 수 있지만 쓰기 동작 시에는 여러 위치에 같은 데이터를 동시에 쓸 수 있다. 매치 선은 wired-AND로 연결되어 있기 때문에 하나의 비트라도 매치되지 않으면 매치에 실패한 것으로 판명된다. 본 구조에서는 쓰기 명령어 수행 시에도 각각의 비트에 원하는 대로 마스크를 씌울 수 있기 때문에 하나의 워드 중 원하는 일부 비트에만 접근하는 것이 가능하다.

나. PE 어레이

각각의 CAM 워드는 데이터를 처리하기 위한 PE와 결합하여 SIMD 구조를 갖는다. 하나의 PE는 매치 결과를 저장하기 위한 세 개의 1 비트 레지스터(R1, R2, R3), 단순한 부울 연산을 할 수 있는 논리 블록(GPLB), 상하의 PE와 데이터 통신을 할 수 있는 선형 네트워크 등을 포함하는 Row logic 블록과 두 개 이상의 매치 결과가 나왔을 경우 하나의 결과를 선택할 수 있는 우선순위 인코더(MRR) 트리로 구성된다.

그림 3은 Row logic 블록의 구조를 나타낸다. 좌측의 신호들은 on-chip controller에서 만들어진 제어신호들을 나타낸다. up, down은 각각 위쪽과 아래쪽 PE의 R1 레지스터를 의미하며 R1, R2, R3 등의 1-bit 레지스터들은 내부 연산 결과들을 저장하기 위해 이용된다. Move와 Match 연산 시 발생하는 결과 값은 3 개의 레지스터 어디에나 저장될 수 있으나 데이터를 쉬프트하거나 MRR 트리에 입력으로 들어갈 수 있는 기능은 R1 레지스터만 가능하다. GPLB는 부울 연산을 담당하는 블록으로 R1, R2, R3 3개의 레지스터 값을 입력으로 8가지 minterm을 이용하여 256가지 함수의 부울 연산을 한다. PE 출력 단에 있는 멀티플렉서는 MRR 결과, GPLB 결과, Match 결과, all '1' 중의 하나를 선택하여 PE의 출력 값으로 내보낸다. Row_logic_out은 CAM 어레이의 워드 선택선으로 사용되어 읽기나 쓰기 동작 시에 해당 워드를 선택한다.

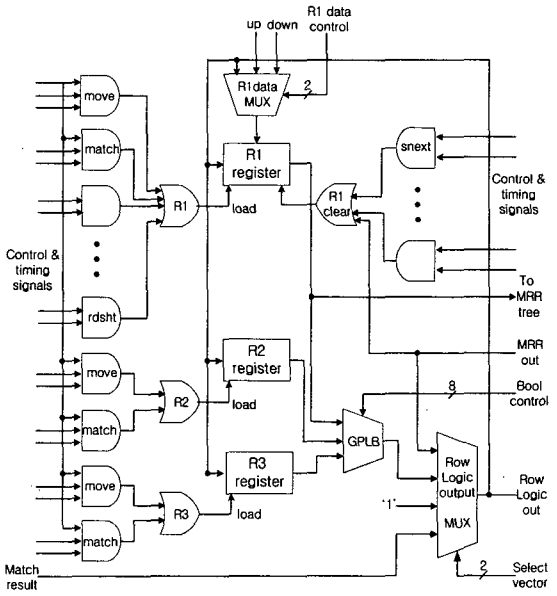

그림 3. Row logic 구조
Fig. 3. Row logic structure

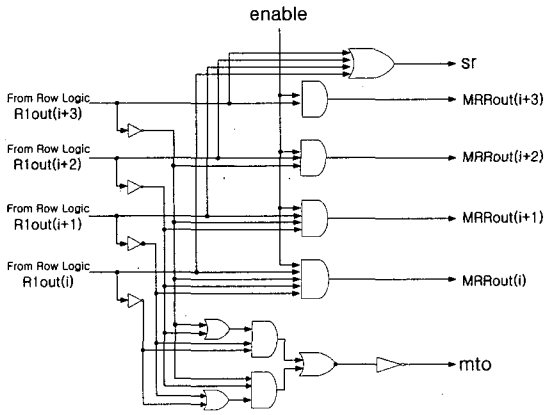

그림 4. 하나의 MRR 블록 구조
Fig. 4. Structure of one MRR block.

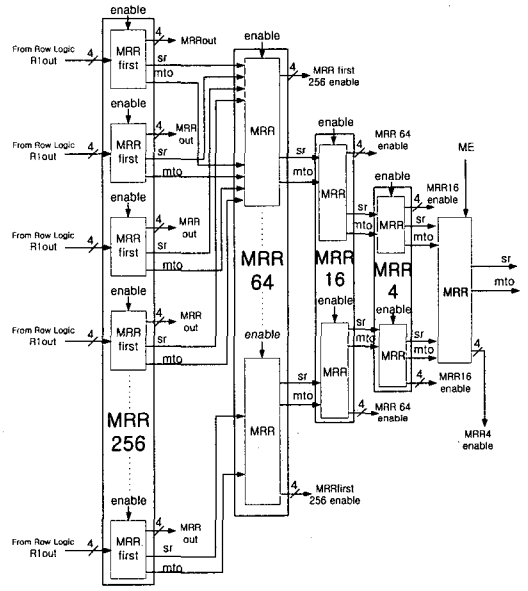

그림 5. MRR 트리 구조(1024 워드)
Fig. 5. Structure of MRR tree.(1024 words)

CAM은 그 구조적 특성상 동시에 여러 개의 워드에 쓰기 혹은 매치 연산을 실행할 수는 있으나, 읽기 연산의 수행 시에는 한 번에 단 하나의 워드만을 읽어낼 수 있다. 즉 여러 개의 1을 가진 응답벡터(response vector)에 해당하는 워드들을 읽을 경우, 복수의 응답벡터 중에서 최상위에 "1"로 세팅된 것을 선택할 수 있는 우선순위 인코더를 필요로 한다. 이로써 생성된 하나의 응답 출력을 선택선으로 하여 읽기를 수행한다. 이를 담당하는 우선순위 인코더가 MRR이며 이는 다단계 트리 구조로 되어 있다. 그림 4는 인접해 있는 네 개의 PE로부터 R1 입력을 받아 처리하는 하나의 MRR 블록 구조를 나타내며 그림 5는 1024개의 PE를 처리하기 위한 MRR 트리 구조를 나타낸다. MRR 트리는 여러 개의 "1"을 가진 입력이 들어올 경우 최상위의 "1"만 남겨두고 나머지는 "0"으로 출력한다. 그 다음 워드를 읽기 위해선 응답벡터의 최상위의 R1 값을 리셋하면 차상위의 "1"을 가진 워드를 선택할 수 있다. MRR은 출력 응답벡터를 만드는 동시에 SR(some response)와 MTO(more than one) 신호도 함께 생성한다. SR은 응답벡터 중에 "1"이 하나 이상 있는가에 대해 판단해주며, MTO는 응답벡터 중에 "1"이 2개 이상 있는가를 판단한다. 이 두 가지 신호는 읽기 연산을 언제 종료해야 할지에 대한 정보를 제공할 뿐만 아니라 현재 응답벡터의 상황을 알 수 있다.

다. 레지스터 뱅크(Register bank)

AP가 연산을 수행하기 위해서는 호스트로부터 연산에 사용될 데이터와 명령어를 전달 받아야하며, 연산을 수행한 뒤 결과 값을 저장하였다가 호스트에 전달해 주어야 한다.

그림 6은 레지스터 뱅크의 구조를 나타내며 사용되는 여러 가지 레지스터와 함께 마스크 패턴을 만들어주는 블록을 포함한다. 상태(Status) 레지스터는 현재 AP의 상태를 나타내며, 연산 수행 시 유용한 정보를 제공한다. 명령어(Command) 레지스터는 호스트로부터 입력받은 명령어를 저장한다. Data-in 레지스터는 입력 데이터를 저장하는데, 이때 CAM 어레이의 한 워드는 42 비트이기 때문에 Low와 High의 Data-in 레지스터가 필요하다. 마스크 레지스터는 호스트로부터 입력받은 마스크를 저장하며, Data-out 레지스터는 수행 결과를 저장하여 호스트에 전달한다.

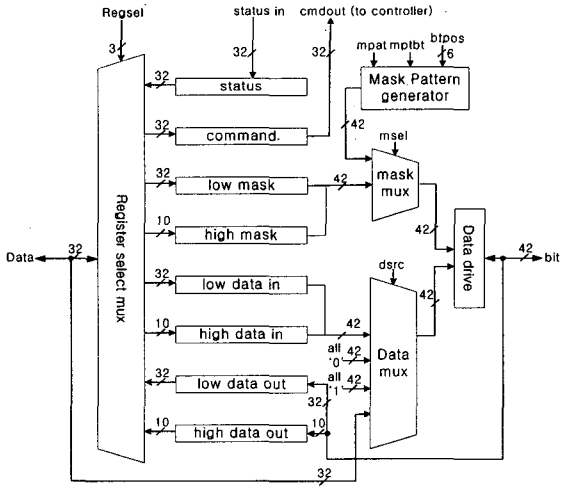

그림 6. 레지스터 뱅크의 블록 다이어그램
Fig. 6. Block diagram of the register bank.

라. 명령어 구조

제안된 AP 에 정의된 명령어에 대해서 살펴본다. 아래의 표 1, 2, 3은 정의된 명령어에 대한 형식 및 명령어의 용법을 나타낸다. 정의된 명령어는 Write, Match, Read, Move, Shift, SelectNext 계열의 명령어로 구분할 수 있으며 각각에 대한 설명은 다음과 같다.

○ Write 계열의 명령어: CAM 어레이에 데이터를 쓰는 명령어이며 병렬로 여러 워드에 같은 데이터를 쓰는 것이 가능하며 비트 단위의 선택적인 처리가 가능하다. PE에서 출력하는 값을 워드선택 선으로 이용하므로 다양한 선택 벡터가 가능하다. 어떤 값을 데이터와 마스크로 사용할지 결정할 수 있으며 마스크를 특정 패턴으로 만들어 사용하는 경우 이에 대한 정의도 필요하다.

○ Match 계열의 명령어: CAM에 저장되어 있는 데이터를 병렬로 검색하는 명령어이다. 매치 연산의 결과인 응답벡터는 PE의 세 레지스터 중의 하나에 저장한다. 어떤 값을 데이터와 마스크로 사용할지 결정할 수 있으며 마스크를 특정 패턴으로 만들어 사용하는 경우 이에 대한 정의도 필요하다. 마스크가 씌워진 부분에 대해서는 매치 연산을 수행하지 않으므로 CAM에 저장된 데이터 중 원하는 비트만 선택적으로 매치를 수행할 수 있다.

○ Read 계열의 명령어: CAM 에 저장된 데이터를 한 번에 하나 씩 읽어내는 명령어이다. CAM은 구조적 특성상 항상 MRR 트리의 출력을 워드 선택선으로 하여 데이터를 읽어낸다.

표 1. 명령어 형식
Table1. Instruction format

31 28	27	26 25	24 23	22 21	20	19 18	17	16 9	8	7	6	5 0
OP_code		rspreg	selvec	dsrc		msrc	dir	GPLB	mpat	mptbit		bitpos

표 2. 명령어의 각 영역
Table2. Instruction fields

Instruction field	bit size	Description
OP_code	4	명령어의 종류 (12 가지)
rspreg	2	match, move 수행 시 response register 선택 (R1, R2, R3)
selvec	2	CAM row를 선택하기 위한 select vector (match result, GPLB out, MRRout, all select)
dsrc	2	data source 선택 (all '0', data register, input data, all '1')
msrc	2	mask source 선택 (mask register, fill pattern, mark pattern, mask register & pattern)
dir	1	shift 수행 시 방향 선택 (up, down)
GPLB	8	R1, R2, R3을 이용한 부울 연산 (8개의 minterm on/off)
mpat	1	mask pattern 선택 (fill, mark)
mptbit	1	fill/mark pattern에서 사용될 비트 선택 ('1', '0')
bitpos	6	mask pattern 설정 시 bit position address (0 - 31)

표 3. 명령어의 수행시간 및 용법
Table3. Instruction cycles and their usage

Instruction	# of cycles	Description
nop	1	nop
write	2	write <selvec>,<dsrc>,<msrc>,[GPLB],[mpat, mptbit, bitpos]
match	3	match <rspreg>,<dsrc>,<msrc>,[mpat, mptbit, bitpos]
read	3	read
shift	2	shift <dir>
move	2	move <rspreg>,<selvec>,[GPLB]
select next	2	snext
write shift	3	wtsht <selvec>,<dsrc>,<msrc>,<dir>,[mpat, mptbit, bitpos]
write select next	3	wtsnt <selvec>,<dsrc>,<msrc>,[GPLB],[mpat, mptbit, bitpos]
write column	2	wtcol <selvec>,<bitpos>,[GPLB]
read shift	3	rdsht <dir>
read select next	3	rdsnt

○ Move 계열의 명령어: PE 내부에 존재하는 세 개의 레지스터 값을 변경시켜주는 명령어이다. PE 내부에서 수행되는 매치 결과, MRR 출력, 부울 연산 등을 저장할 수 있다.

○ Shift 계열의 명령어: PE 내부의 R1 벡터를 위 또는 아래로 쉬프트 하는 명령어이며 매치나 부울 연산의 결과에 대한 PE 간의 데이터 통신을 담당한다.

○ SelectNext 계열의 명령어: MRR의 입력인 R1 벡터가 다수의 "1"을 포함할 경우, 최상위 "1"을 갖는 R1 벡터의 해당 비트를 리셋 함으로써 차상위 반응 벡터를 선택하여 읽기나 쓰기 명령어를 수행할 경우 사용하는 명령어이다.

마. Binary/Quad 모드

영상처리, 논리함수 처리 등의 응용분야에서는 이진 값 뿐 아니라 네 가지의 값을 허용하는 Quad 모드가 유용하다. Quad 모드는 기존의 0과 1이외에 *(don't care), N(never match)의 값을 추가하여 구성된다. 기존의 Quad 모드 매치 연산은 인접한 상하의 두 워드에 Quad 모드를 지원하기 위한 하드웨어를 추가하여 구현하였다.[10] 본 논문에서는 추가의 하드웨어 없이 인접한 두 셀의 인코딩만으로 Quad 모드를 지원한다. 즉, 42 비트의 CAM 워드는 두 셀씩 묶여 21 개의 Quad 값을 갖으며 해당하는 데이터와 마스크를 적절히 인코딩하여 매치하면 Quad 모드와 같이 동작하게 된다.

표 4. CAM Quad 인코딩
Table 4. CAM Quad encoding.

*i*th CAM bit	(*i*+1)th CAM bit	Quad 값
0	0	0
1	1	1
1	0	*
0	1	N

표 5. Quad 모드 매치 테이블(M:마스크, D:데이터)
Table 5. Quad mode match table.(M:mask, D:data)

CAM value		match value: 0		1		*		N	
		M=1 D=*	M=0 D=0	M=0 D=1	M=1 D=*	M=1 D=*	M=1 D=*	M=0 D=1	M=0 D=0
0	0	yes		no		yes		no	
1	1	no		yes		yes		no	
*	1	0	yes		yes		yes		yes
N	0	1	no		no		yes		no

표 4는 인접한 두 비트를 이용한 Quad 데이터의 표현 방법이고 표 5는 Quad 매치를 수행하기 위하여 주어지는 마스크 및 데이터 값을 나타낸다. 즉, 표 5와 같이 데이터와 마스크를 설정하고 Binary 모드에서와 같은 방법으로 매치를 하면 Quad 모드에서와 동일한 결과를 얻을 수 있다는 것이다. 따라서 추가의 하드웨어 없이 Quad 매치를 수행할 수 있다.

IV. 설계 및 검증

본 논문에서 제안된 범용 AP는 VHDL로 설계되어 Modelsim 환경에서 기능이 검증되었다. 좀 더 포괄적인 검증을 위하여 명령어 단위의 검증 뿐 아니라 다음의 병렬 알고리즘의 수행을 검증하였다. 최대값/최소값 검색, 이상/이하 검색(Greater-than/Less-than search), 병렬 가감산 등의 알고리즘도 검증하였다. 모든 알고리즘은 병렬로 수행되기 때문에 저장된 데이터의 개수와는 무관하며 상수 $O(k)$의 복잡도를 가지며 처리 비트 수만큼의 이터레이션이 필요하다.

1. 최대값 검색 알고리즘

그림 7은 CAM에 저장된 데이터 중 가장 큰 값을 찾아내는 알고리즘이다. MSB부터 LSB까지 이터레이션마다 한 비트씩 '1'과 매치를 수행하면서 반응벡터의 결과 값을 이전 결과와 AND 연산하여 저장한다. 반응벡터가 단 하나의 1을 갖거나 LSB까지 검색 완료될 때까지 반복한다. 만일 AND 연산 후에 반응벡터가 모두

```
Max_srch():
  mask = 0x7fffffff;          # set mask value
  data = 0xffffffff;          # set data for match operation
  for i=0 to 31 do {
    call_Max();               # function call for minimum search
    if(no_responder) call_Elim();
    mask>>=1; }
  call_Read();

call_Max():
  R2 = R1,                    # store R1 in R2
  R3 = match(data,mask),      # find entries with '0' in i-th position
  R1 = R1 ^ R3,               # refinement

call_Elim():
  R1 = R2.                    # restore old response vector

call_Read():
  result = CAM[MRRout].       # read out result
```

그림 7. 최대값 검색 알고리즘
Fig. 7. Maximum search algorithm.

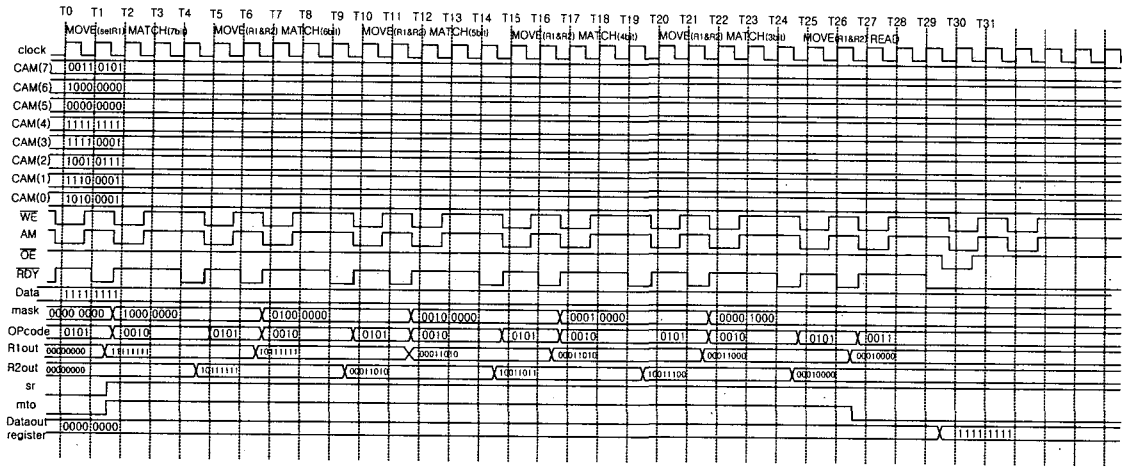

그림 8. 최대값 검색에 대한 타이밍 다이어그램
Fig. 8. Timing diagram for the maximum search.

'0'이면 해당 비트로는 경우의 수를 줄일 수 없다는 뜻이므로 원래의 반응벡터 값을 복원하여 다음 비트에 대하여 검색을 진행한다. 마지막까지 검색한 후 남는 것이 최대값이며 반응벡터 중에 '1'인 엔트리가 두 개 이상 남는다면 모두 같은 값이다.

그림 8은 최대값 검색 알고리즘을 수행한 타이밍 다이어그램이며 편의상 8 비트 데이터 8 개를 이용하여 실험하였다. T0 시간에 move 명령어가 실행되어 T1시간에 R1 레지스터가 전부 '1'로 셋 된다. T2 시간에 match 명령어가 실행되어 T4 시간에 CAM 데이터들의 7번째 비트들과 match한 결과인 10111111이 R2 레지스터에 저장된다. T5 시간에 move 명령어가 실행되어 T6 시간에 R2 레지스터 값과 R1 레지스터 값을 AND 한 결과인 10111111이 R1 레지스터에 저장된다. SR과 MTO의 값이 둘 다 '1' 이므로 다시 match와 move를 반복한다. 위의 과정을 4번 더 반복하면 T26 시간에 MTO가 0 으로 변하고 SR은 1 이므로 반응벡터에 '1'인 엔트리가 하나만 있음을 알 수 있다. T27 시간에 read 명령어를 실행하고 T29 시간에 CAM(4) 에 저장되어 있던 최대값인 11111111이 data-out 레지스터에 저장되고 OE 값이 0 으로 변하여 호스트에게 결과 값이 나왔음을 알려준다.

2. 병렬 덧셈 알고리즘

그림 9는 CAM에 저장된 데이터를 기반으로 병렬 덧셈을 수행하는 알고리즘이다. CAM의 32비트 데이터 영역에 상위 8 비트(24-31)에는 A 값이 다음 8 비트(16-23)에는 B 값이 저장되어 있고 이 두 값을 더한 결과는 하위 16 비트(0-15)에 C 값으로 저장된다. LSB부터 시작하여 a_i와 b_i 비트는 match 연산에 의해 해당 PE에 저장되고 GPLB의 연산에 의해 carry와 sum이 계산되어 CAM의 C 값의 해당 비트에 저장된다. 덧셈 연산은 bit-serial, word-parallel의 형태로 동작되어 A, B 데이터의 개수에 상관없이 병렬로 수행된다.

```
Par_addition():
  A_mask = 0xf1ffffff; # set mask value for A
  B_mask = 0xfff1ffff; # set mask value for B
  C = 0xfffffff1;      # bit position for result
  data = 0xffffffff;   # set data for match operation
  call_Clear();
  for i=0 to 7 do {
    call_Add();        # function call for parallel addition
    A_mask <<= 1;      # shift A_mask for next iteration
    B_mask <<= 1;      # shift B_mask for next iteration
    C <<= 1; }         # shift C for next iteration

call_Add():
  R2 = match(data,A_mask),        # copy a_i into R2
  R3 = match(data,B_mask),        # copy b_i into R3
  R1 = (R2^R3)+(R2^R1)+(R3^R1),   # for carry-out
  R3 = R2^~R1^~R3+R2^R1^R3+~R2^~R1^R3+~R2^R1^~R3, # for sum
  CAM[R3] = [C].      # copy R3 into c_i of CAM word

call_Clear():
  R1 = 0.             # clear carry-in
```

그림 9. 병렬 덧셈 알고리즘
Fig. 9. Parallel addition algorithm.

V. 결 론

본 논문에서는 Associative computing을 기반으로 하는 범용 병렬 AP 프로세서 구조를 제안하고 효율적인 명령어 세트를 구성하였다. 다양하고 대용량 응용분야에도 적용 가능하도록 구조를 확장 가능하게 설계함으

로써 제안된 프로세서는 bit-serial, word-parallel로 동작하는 대용량 병렬 SIMD 프로세서이다. 제안하는 AP는 그 구조가 단순하지만 효율적이고 유연한 구조를 갖고 있으며 하나의 칩에 수 천 개의 PE를 내장할 수 있다. 제안된 명령어 세트는 연속된 명령어를 하나의 명령어로 구현함으로써 수행시간을 단축하였다. 32 비트 데이터와 10 비트 태그 영역의 CAM은 모두 비트단위로 선택되어 처리될 수 있도록 설계되었다. 설계된 PE는 Response Registers, 부울함수 연산기(GPLB), MRR 트리, 쉬프트 레지스터 등을 포함하는 1 비트 구조로 복잡도를 개선하였으며 추가의 하드웨어 없이 Binary 모드뿐만 아니라 영상처리, 패턴인식, 신경망시스템 등 다양한 분야에서 필요한 Quad 모드(0, 1, *, N)도 지원할 수 있도록 설계되었다. 제안된 구조를 검증하기 위하여 제안된 명령어뿐만 아니라 기본적인 검색 및 연산 알고리즘을 구현하여 실험하였다.

감사의 글

본 저자는 본 연구를 위하여 설계 환경을 제공하여 준 IDEC(IC Design Education Center)에 감사드린다.

참 고 문 헌

[1] J. Potter, *Associative computing: A programming paradigm for massively parallel computers*, Plenum publishing, New York, 1992.

[2] A. Krikelis, "Computer vision applications with the associative string processor," *J. of parallel and distributed computing*, vol. 13, no. 2, pp. 170-184, 1991.

[3] H. Kitano, *Speech-to-speech translation: A massively parallel memory-based approach*, Kluwer academic publisher, 1994

[4] C. Weems, "The image understanding architecture," *Int'l J. computer vision*, vol. 2, no. 4, pp. 251-282, 1989.

[5] R. Storer, et al., "An associative processing module for a heterogeneous vision architecture," *IEEE Micro*, vol. 12, no. 3, pp. 42-55, 1992.

[6] T. Higuchi, et al., "The IXM2 parallel associative processor for AI," *IEEE Computer*, vol. 27, no. 11, pp. 53-63, 1994.

[7] A. Louri, et al., "An optical associative parallel processor for high-speed database processing," *IEEE Computer*, vol. 27, no. 11, pp. 65-72, 1994.

[8] K. Grosspietsch and R. Reetz, "The associative processor system CAPRA: architecture and applications," *IEEE Micro*, vol. 12, no. 3, pp. 58-67, 1992.

[9] D. Tavangarian, "Flag-oriented parallel associative architectures and applications," *IEEE Computer*, vol. 27, no. 11, pp. 41-51, 1994.

[10] C. Stormon, et al., "A general-purpose CMOS associative processor IC and system," *IEEE Micro*, vol. 12, no. 3, pp. 68-78, 1992.

[11] K. Tamaru, K. Kobayashi, and H. Onodera, "Memory based architecture and its implementation scheme named bit-parallel block-parallel functional memory type parallel processor MPMP FMPP," *Comp. and Elect. Engineering*, vol. 24, pp. 17-31, 1998.

[12] F. Herrmann and C. Sodini, "A 256-element associative parallel processor," *IEEE J. Solid-State Circuits*, pp. 365-370, vol. 30, no. 4, 1995.

저 자 소 개

박 태 근(정회원)
1985년 연세대학교 전자공학사.
1988년 Syracuse Univ. Computer 공학석사.
1993년 Syracuse Univ. Computer 공학박사.
1991년~1993년 Coherent Research Inc. VLSI 설계 엔지니어.
1994년~1998년 현대전자 System IC 연구소 책임연구원.
1998년~현재 가톨릭대학교 정보통신전자공학부 부교수.
<주관심분야 : VLSI 설계, CAD, 병렬처리>