IPv4 라우팅에 적합한 SMT 아키텍처 개발

임 정 빈*, 홍 인 표, 조 정 현, 이 용 석

An Optimal SMT Processor Architecture for IPv4 Packet Routing

Lim Chung Bin*

요 약

네트웍 시스템은 빠른 트래픽 처리속도와 다양한 서비스 제공을 위한 유연성이라는 두 가지 측면을 모두 만족시키기 위하여 발전되어 왔고 이러한 결과로 네트웍 프로세서가 출현하게 되었다. 네트웍 프로세서의 성능을 향상시키는 방법으로는 고속의 외부 인터페이스와 전용 연산기 등을 이용하는 방법들이 사용되어왔다. 최근에 이르러 이러한 방법 이외에 아키텍처적인 방법을 이용한 SMT(Simultaneous MultiThreading) 네트웍 프로세서의 구조가 제안되었는데 SMT 구조는 설계의 복잡성으로 인하여 상용화에 어려움이 있다. 따라서 SMT 구조의 네트웍 프로세서를 개발하기 위해서는 SMT 구조의 하드웨어 간략화 측면의 연구가 필요하다. 본 논문에서는 SMT 구조의 각 유닛들이 네트웍 알고리즘을 실행하는데 있어서의 영향을 분석하고 그 결과를 바탕으로 최적화된 SMT 네트웍 프로세서의 구조를 제안하였다.

ABSTRACT

Network systems have been developed to meet the high performance of forwarding packets and flexibility for providing various services, so network processor emerged. In order to improve the performance of network processors, fast external interface and special functional units have been used. Recently as an architectural method of improving performance, the SMT(Simultaneous MultiThreading) architecture is proposed, but this architecture is difficult to implement due to its complexity. Therefore research for architectural optimization is needed to develop the SMT network processors. In this paper we analyze each functional units on performing network algorithms and propose an optimized SMT network processor architecture.

Ⅰ. 서 론

인터넷 사용의 급증에 따라 오늘날의 라우팅/스위치 시스템은 많은 양의 패킷처리와 다양한 서비스의 제공이라는 두 가지 목표를 향해 발전하게 되었다.

초기의 라우팅 시스템에서는 wire-speed 수준의 빠른 처리속도를 위하여 ASIC설계로 시스템을 구축하였으나 이 방식은 시장진입까지의 시간이 길어진다는 특징으로 인해 네트웍 시장에 맞지 않는 단점을 드러내며 한계에 도달하였고, 빠른 처리속도와 시장 유동성이라는 두 가지 상반되는 특성을 만족시키기 위하여 1990년대 이후 네트웍 프로세서가 등장하게 되었다. 네트웍 프로세서는 효율적인 네트웍 기능 수행을 위한 명령어 및 연산기와 고속의 데이터 처리를 위한 외부 인터페이스를 채용하므로써 고속의 데이터 처리와 확장성을 동시에 만족시키는 시스템을 구성하는 핵심코어로 사용되어왔다.

본 논문에서 제안하는 네트웍 프로세서의 기본구조는 SMT 구조를 기반으로 하고 있다. SMT 구조는 쓰레드간의 병렬성을 이용하여 성능향상을 꾀하는 구조로써 단일 프로세서로 여러 개의 패킷처리를 병렬적으로 수행하는 효과를 보이므로 인하여 활용도가 돋보였으나 설계상의 복잡성으로 인하여

* 연세대학교 전기전자공학과(biu5000@dubiki.yousei.ac.kr)
논문번호: 030426-1006, 접수일자: 2003년 10월 6일
※ 본 연구는 한국과학재단 목적기초연구(R01-2002- 000-00310-0) 지원으로 수행되었음.

그 적용에 어려움이 있는 구조이다.

본 논문에서는 SMT 구조의 네트웍 프로세서로써 패킷 라우팅 기능을 수행하는데 있어서 성능과 하드웨어자원을 최적화 시키기 위한 연구를 수행하였다. 2장에서는 기본적으로 사용되는 여러 라우팅 알고리즘을 하드웨어 관점에서 분석해 보았다. 이어 3장에서는 SMT 네트웍 프로세서의 개략적인 구조와 파이프라인, 그리고 하드웨어 간략화 측면에서의 그룹핑에 대하여 논하고, 4장에서는 연구 방법에 대하여 설명한다. 5장에서 성능평가 결과를 보이고 6장에서 결론을 맺는다.

Ⅱ. 라우팅 알고리즘과 하드웨어

초기의 라우팅 알고리즘은 소프트웨어 기반의 단순한 이진 trie 검색 구조에서 시작하였다. 이 방식은 라우팅 테이블의 구조가 간단하고 추가 및 변경이 용이한 구조를 가지고 있지만 필요이상으로 메모리의 사용이 많고 라우팅 테이블의 검색 속도가 오래 걸리는 단점을 가지고 있다.

특히 이진 trie 구조의 단점은 라우팅 테이블의 크기가 커짐에 따라 메모리의 사용량이 더욱 증가한다는 것이다. 특히 이진 trie는 수만 개의 유효 엔트리를 가지는 라우팅 테이블을 구성하는데 있어서 수십 배에 해당하는 빈 엔트리를 함께 유지해야 하는 비효율성을 가지고 있었다.

이러한 메모리 낭비를 줄이기 위해 테이블내의 빈 엔트리를 제거하는 PC(Path Compressed) trie 방식의 라우팅 알고리즘이 제안되었다. PC trie 알고리즘은 테이블 내 단일 패스를 가지는 노드들은 테이블 검색시 각 단계마다 비교할 필요가 없다는 점에 착안하여 라우팅 테이블 trie 내의 단일 패스를 압축하므로써 메모리 사용을 크게 줄인 방식이다. 그러나 실질적으로 PC trie 방식은 IP 주소의 모든 비트들을 비교하는 것이 아니기 때문에 이를 보완하기 위하여 실제 IP 주소를 비교하는 longest matching이라는 추가적인 작업이 필요하게된다. 또한 longest matching은 라우팅 테이블 내의 유효 엔트리를 이루는 IP address key 값의 길이가 대개 각 클래스별 IP 주소의 길이와 비슷하다는 통계적 특성을 이용하여 trie의 맨 끝 노드부터 역 방향으로 적용하여 검색한다.[1] 이러한 방식의 trie 구조는 BSD trie라는 이름으로 더 잘 알려져 있으며 현재에도 많이 사용되고 있다.

이러한 PC trie 구조에서는 각 노드를 지날 때마다 메모리상의 라우팅 테이블에서 특정 엔트리를 엑세스하게 되는데 이때의 메모리의 엑세스가 순서에 따라 연쇄적으로 이루어져야 하기 때문에 trie의 height는 라우팅 테이블 검색시간에 결정적인 영향을 미치게 됨을 알 수 있다. 비록 PC trie 구조가 라우팅 테이블 내의 단일 패스들을 압축시키므로써 메모리와 프로세스의 낭비를 일부 제거하였지만 PC trie와 이진 trie의 height를 비교해 보았을 때 평균 height 길이의 감소가 그다지 많지 않음을 알 수 있다.

이러한 단점을 보완하기 위하여 trie의 height를 줄여 메모리 엑세스 지연시간을 제거하므로써 성능 향상을 이루어낸 것이 LC(Level Compressed) trie 구조이다. 이진 trie 구조와 PC trie 구조에서 각 노드는 최대 2개씩의 하위 노드를 가지며 각 하위 노드로의 분기의 결정에 있어서 32 bit IP 주소 중 하나의 bit를 통해 분기되는 구조이다. 그러나 LC trie 구조에서는 각 노드는 2^n (단 n>0)개의 하위 노드를 가질 수 있으며 하위 노드로 분기하기 위하여 32bit IP 주소 중 n bit를 참조한다. 여기서 참조되는 n bit는 바로 메모리 참조에 이용되므로써 하위 노드로의 분기가 이루어진다. 따라서 한 노드를 기준으로 그 노드의 하위 노드들은 메모리 상에 연속적으로 존재해야만 하며 이러한 구조상의 장점 때문에 각 노드는 간단한 데이터 구조 형태를 가지면서도 가변적인 수의 하위 노드를 가질 수 있는 것이다. 이러한 이유로 LC trie는 노드의 height를 줄이면서도 메모리 엑세스를 효율적으로 수행 할 수 있는 특성을 지니게 된다.[1][2]

실제 LC trie에서 라우팅 테이블의 엔트리 수가 4만 여개인 경우 평균 height가 8 정도로 나타나는데 비해 비슷한 크기의 라우팅 테이블을 가지는 PC trie의 구조의 평균 height는 20정도로 나타남을 통해 이 알고리즘의 효율성을 알 수 있다.[1][3]

또한 LC trie 알고리즘의 경우 하위 노드로의 분기 시 비교 연산을 이용하는 것이 아닌 포인터를 이용한 메모리 레퍼런스를 사용하므로써 프로그램의 명령어 레벨에서 분기 명령어를 제거하고 있다. 따라서 LC trie 알고리즘을 사용할 때 분기 명령어 출현 빈도 수가 줄어듦으로 인하여 분기예측 실패에 의한 지연시간의 기대치를 줄일 수 있을 뿐 아니라 하드웨어적으로도 정교한 분기 예측기를 사용

할 필요가 없게 되므로써 하드웨어 측면의 부담을 줄일 수 있게 된다.

그러나 이러한 LC trie 역시 일반적인 응용 프로그램에 비하여 많은 메모리 레퍼런스를 포함하고 있으며, 이러한 특성은 프로세서 상에서 잦은 캐쉬 미스 등으로 인한 처리 시간 지연을 유발한다. 기존의 프로세서는 이러한 메모리 레퍼런스로 인한 시간 지연 동안에 남아 있는 ALU등의 실행 자원을 효율적인 작업에 사용하지 못하고 낭비함으로써, 실제적으로 패킷 처리 성능의 심각한 저하를 가져왔으며 대부분의 고속 라우팅 응용에서 네트워크의 선속도(wire speed)를 따라가지 못하는 결과를 낳았다. 이에 비하여 SMT는 다수의 스레드로부터 경쟁적으로 명령어를 이슈함으로써 메모리 레퍼런스 혹은 분기 명령으로 인한 지연시간을 효과적으로 이용할 수 있다. 즉, 하나의 프로세서에서 여러 개의 패킷 처리 알고리즘을 동시에 수행함으로써 전체적인 패킷 처리 스루풋(throughput)을 향상시킬 수 있는 것이다. 실제로 Patrick Crowley등은 수퍼스칼라, 단일 칩 멀티프로세서(CMP), 비순차 SMT 등의 여러 프로세서 아키텍처를 사용하여 다양한 네트워크 동작을 수행했을 때의 성능을 측정하여 SMT가 네트워크 응용에 적합한 아키텍처임을 밝힌 바 있다.[4]

Ⅲ. SMT 네트웍 프로세서의 아키텍처

3.1. SMT 프로세서의 기본 아키텍처

SMT 프로세서는 명령어간 병렬성(Instruction Level Parallelism)만을 이용한 슈퍼스칼라 방식의 프로세서와는 달리 명령어간 병렬성과 쓰레드간 병렬성(Tread Level Parallelism) 모두를 이용하므로써 자원의 활용도를 높여 전체적인 성능을 향상시킨 구조이다.

본 논문에서 제안하고 있는 SMT 네트웍 프로세서에서는 순차적 명령어 이슈 및 완료(in-order issue and in-order completion)정책을 사용하고 있다. SMT 구조는 서로 다른 쓰레드의 명령어간 병렬성을 이용하므로써 분기예측 실패와 메모리 액세스 지연에 대한 immunity를 가지기 때문에 메모리 액세스가 빈번한 trie 알고리즘에서 이러한 특성이 큰 효과를 발휘한다. 또한 SMT 구조는 기본적으로 ILP와 TLP를 동시에 효과적으로 이용하도록 되어 있으므로, 한 쓰레드 내에서는 순차적인 이슈 및 완료 방식으로 인하여 ILP를 다소 덜 이용한다고 하여도 이를 TLP로 보충하여 프로세서 전체적인 성능을 높일 수 있다. Hily의 논문에 따르면 싱글 쓰레드 환경에서 순차적 이슈 및 완료 방식과 비 순차적 방식의 퍼포먼스 갭은 46%정도이지만, 4 쓰레드 정도의 멀티쓰레딩 구조에서 순차적 방식과 비 순차적 방식의 퍼포먼스 갭은 15%정도로 크게 줄어들기 때문에 순차적 방식을 사용하는데에 따른 성능 저하가 크지 않을 뿐 아니라 파이프라인의 제어가 간단해지고 명령어가 순서대로 이슈 및 실행되므로써 분기 예측 오류로 인한 손실이 줄어들게 된다. 더욱이 실제 설계를 고려했을 때 하드웨어 설계가 단순해짐에 따라 높은 클럭주파수에서 동작할 수 있게 되므로써 실질적인 성능저하는 동일한 구조하에서 비교된 수치보다 낮게된다.[5]

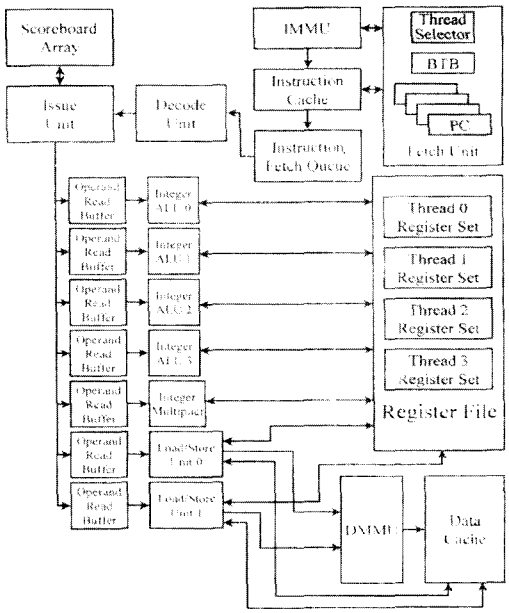

그림 5. 순차적 SMT 구조의 블록 다이어그램

그림 1은 4개의 쓰레드를 지원하는 SMT 구조의 전체적인 모습을 보여주는 블록 다이어그램이다. 이 그림에서 볼 수 있듯이 전체적인 SMT 구조는 슈퍼스칼라 구조와 별 차이가 없고 단지 페치 유닛 내에 쓰레드 선택기가 존재한다는 것과 PC와 레지스터 세트의 개수가 쓰레드 숫자만큼 추가로 존재한다는 점이 다르다. 서로 다른 프로세스 사이의 명령어 수행을 위해서 프로세스간 컨텍스트 스위칭을 해야만 하는 CMT(Coarse MultiThreading)나 FMT(Fine MultiThreading)등의 멀티쓰레딩 프로세

서 구조에 비해 SMT 프로세서는 단일 사이클 내에 여러 쓰레드내의 명령어들을 컨텍스트 스위칭 없이 동시에 수행하기 위하여 이러한 추가적인 하드웨어가 필요하다.[6]

3.2. SMT 프로세서의 파이프라인구조

SMT 프로세서 구조는 슈퍼스칼라 프로세서 구조와 거의 유사함을 보이지만 세부적으로는 좀 더 많은 차이점을 보인다. 그림 2는 SMT의 파이프라인 구조를 간략히 도시한 것이다.

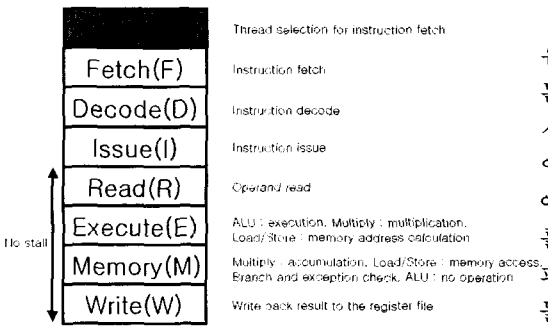

그림 6. 순차적 SMT 구조의 파이프라인

이 그림에서 알 수 있듯이 SMT 파이프라인 구조에서는 슈퍼스칼라의 구조와 달리 Fetch 단 앞에 Select 단계가 추가된다. SMT 구조에서는 다수의 쓰레드들로부터 명령어를 페치 하여 페치 큐로 보내는데 이때 매 사이클마다 어떤 쓰레드에서 명령어를 페치 할 것인지를 결정해야 한다. 물론 동시에 모든 쓰레드에서 명령어를 페치 할 수도 있지만 이러한 방법은 캐쉬의 포트 수를 쓰레드 수 만큼 늘려야 하기 때문에 증가되는 하드웨어 복잡도가 매우 커지며 그에 비해 페치 성능향상은 그다지 크지 않기 때문에 명령어 캐쉬포트 수를 제한하여 매 사이클마다 일부 쓰레드로부터만 명령어를 페치 하도록 하는 방법을 사용하므로써 하드웨어 복잡도를 줄였다. 따라서 SMT 파이프라인 구조에서는 명령어를 페치 하기 전에 어떤 쓰레드에서 명령어를 페치 해 올 것인지를 결정해야 하는데 페치 단에서 쓰레드 선택과 명령어 페치를 함께 할 경우 페치 단의 지연시간이 너무 길어지게 되므로 이를 각각의 단계로 나눈 것이다.

페치유닛은 매 사이클마다 페치 쓰레드 셀렉터에서 할당된 캐쉬 포트를 이용하여 명령어를 읽어와서 명령어 페치 큐에 넣는다. 이를 위해서 명령어 캐쉬는 non-blocking 캐쉬를 사용하였다. 기존의 프로세서 구조에서는 일반적으로 일정 수행 사이클 동안 하나의 쓰레드만을 처리하므로, 실행중인 쓰레드에서 캐쉬 미스가 발생하면 모든 명령어의 페치가 중단된다. 따라서 명령어 캐쉬는 blocking 캐쉬로 설계하는 것이 일반적이었다. 그러나 SMT에서는 동시에 여러 쓰레드가 명령어를 페치 할 뿐더러 한 쓰레드에서 캐쉬 미스가 발생하여도 다른 쓰레드에서는 명령어를 페치 할 수 있어야 하므로 다중 포트 non-blocking 캐쉬가 요구된다.

그 이외의 슈퍼스칼라 구조와의 차이점은 이슈 유닛에서 의존성을 검사할 때 같은 쓰레드에 속하는 명령어끼리만 의존성을 검사하고 예외상황 발생 시 처리를 위해서 이슈 큐 및 모든 실행 유닛의 각 엔트리는 쓰레드 식별자(thread identifier)를 포함해야 한다는 것, 그리고 동시에 여러 쓰레드의 명령어를 실행하기 위하여 실행 유닛과 각각의 레지스터 파일 셋과의 와이어링이 복잡해진다는 등의 특징을 들 수 있다.

3.3. SMT 프로세서의 그룹핑

SMT 프로세서는 슈퍼스칼라 프로세서를 기반으

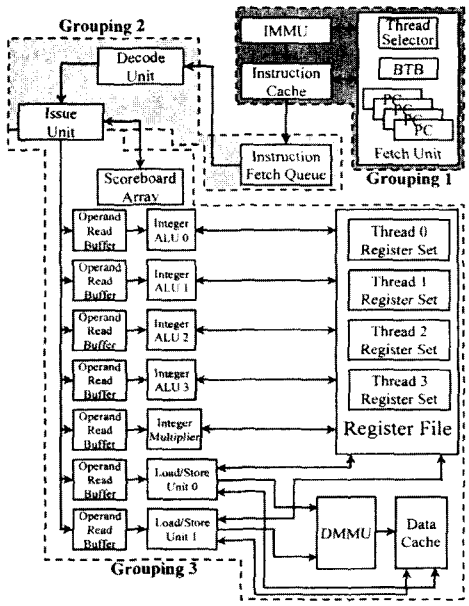

그림 7. 순차적 SMT 구조에서의 그룹핑

로 추가 및 변경을 통해 만들어 질 수 있다. 그러나 SMT 구조의 하드웨어 설계 복잡도가 네트웍 프로세서로의 실용화에 큰 장애가 된다. 따라서 이러한 약점을 보완하기 위한 방안으로 그룹핑이 제안되었다. 그룹핑은 자원과 쓰레드를 몇 개의 그룹으로 나누어 특정 자원을 같은 그룹에 속하는 쓰레드들 사이에서만 공유하는 방식이다. 그룹핑을 적용할 경우 모든 쓰레드들이 전체 자원을 공유하는 방식에 비해서는 자원의 활용도 및 성능이 낮아지지만 설계상의 복잡도를 크게 줄일 수 있다는 장점이 있다.

그림 3은 순차적 SMT 구조에서의 세 가지 그룹핑에 대한 블록다이어 그램이다. 그룹핑 1에서는 4개의 쓰레드 중 2개의 쓰레드를 선택하는 4:2 쓰레드 셀렉터의 동작을 2개의 쓰레드 중 한 개의 쓰레드를 선택하는 2:1 쓰레드 셀렉터 2개를 이용하므로써 쓰레드선택 뿐 아니라 명령어 페치 및 분기 예측까지도 단순화시킬 수 있다.

그룹핑 2는 명령어 페치 큐와 디코드 슬롯, 명령어 이슈 큐에 대한 것이며 그룹핑 2가 여러 개로 나눠지는 경우 각 쓰레드의 명령어들이 적절히 섞이지 않게 되어 슈퍼스칼라와 같이 TLP의 이득을 이용할 수 없는 일이 발생된다. 따라서 그룹핑 2에서는 각 그룹의 크기를 적절히 조절하여야 하고 정해진 크기의 그룹 내에서 TLP의 이용에 무리가 없도록 큐의 엔트리 수를 적절히 조절하여 성능저하를 막아야한다.

그룹핑 3은 실행 유닛의 공유에 관한 정책인데 3가지의 그룹핑 중 설계의 간략화 효과가 가장 커지는 부분이다. 그룹핑 3은 한 그룹에 속한 쓰레드는 같은 그룹내의 실행 유닛과 레지스터 파일만을 이용하므로써 읽기 포트 및 쓰기 포트를 그룹별로 배타적으로 사용할 수 있기 때문에 레지스터 파일의 설계를 간략화 할 수 있다. 그룹핑 3을 잘 적용시키면 레지스터 파일의 포트 수가 작아지므로 단일 레지스터 파일의 크기를 줄일 수 있을 뿐 아니라 엑세스 시간도 단축시킬 수 있게 된다.

이렇듯 그룹핑은 SMT 구조의 설계 복잡도라는 약점을 보완하기 위한 방편으로 사용되고 있다. 전체적으로 세 가지의 그룹핑이 제시되고 있고 각각의 그룹핑은 명령어 페치관련 자원, 명령어의 이슈를 위해 정렬에 이용되는 자원, 명령어의 이슈 및 실행과 관련된 자원에 대한 것이다. 이들은 서로 다른 자원에 대한 내용이기 때문에 그룹핑은 각기 독립적으로 적용시킬 수 있다.

Ⅳ. 연구 방법

본 논문에서 제안된 SMT 네트웍 프로세서 구조는 cycle-based, execution driven 시뮬레이터로 구현되며 시뮬레이터는 C 언어를 사용하여 작성되었다.

시뮬레이터는 SMT 구조 전체를 모델링 하도록 설계되어지고 여러 가지 하드웨어 옵션에 선택 사항을 두어, 이를 변화시키면서 프로세서의 성능에 이러한 옵션들이 미치는 영향을 볼 수 있도록 하였다. 이 선택 옵션들로는 쓰레드 수, 그룹핑 옵션, 명령어 페치 큐의 엔트리 수, 레지스터 포트 수, 이슈 정책, 분기 예측 옵션, 실행 유닛과 캐쉬크기와 way 등을 정하였다.

SMT 네트웍 프로세서의 명령어 셋(Instruction Set Architecture, ISA)으로써는 ARM의 ISA를 채택하였다. ARM은 소형 내장형 프로세서로서 상태 레지스터가 존재하고 모든 명령어가 조건부 실행을 지원하여 SMT 구조로 고성능화 하기에는 어려움이 있다. 상태 레지스터는 거의 모든 명령어가 목표 레지스터로 사용하므로, 명령어간의 종속관계를 예측하는 데에 있어 조건을 복잡하게 만드는 등의 문제점을 유발한다. 또한 조건부 실행은 스칼라 RISC 프로세서에서 잦은 분기를 제거하는 효과가 있었으나, SMT나 슈퍼스칼라와 같은 다중 명령어 실행 환경에서는 또 다른 명령어간의 종속관계를 만들게 되어 명령어 이슈 시에 종속관계를 해결하기 힘들게 한다. 하지만, ARM은 시장 진입이 용이하고 현재 더 많은 응용 프로그램과 개발환경이 갖추어져 있으며, 가격 경쟁력이 우수하다. 따라서 이러한 장점을 수용하여 ARM 명령어 셋을 본 연구의 SMT 네크웍 프로세서 호환 명령어 셋으로 선택하였다.[7]

LC trie 방식의 라우팅 프로그램을 ARM 프로세서에서 수행될 수 있도록 응용 프로그램으로 만들어 놓은 실행 파일을 시뮬레이터의 입력으로 사용했다. ARM 프로세서용의 응용 프로그램의 실행 파일을 얻기 위해서는 ARM Development Suit(ADS)에 포함되어 있는 컴파일러를 사용하였으며 시뮬레이터에는 ADS에서 컴파일된 라우팅 프로그램들을 각 쓰레드별 입력으로 가하여 다중 쓰레드에서의 SMT 네트웍 프로세서의 성능 평가를 실시하였다.

ADS에서 생성되는 실행 파일의 형식은 ELF이다. 본 시뮬레이터도 ELF 형식의 실행 파일에서 프로그램 명령어들을 추출하여 시뮬레이터의 메모리에

적재하는 loader 부분을 포함한다.

ADS는 별도의 OS가 존재하지 않고, ARMulator라는 프로세서 emulator에 software interrupt(SWI)를 처리하는 부분이 있어, 화면 출력, console 입력, 파일 입출력 등의 몇 가지 필수적인 OS 기능을 응용 프로그램에 제공하도록 되어 있다. 본 시뮬레이터도 같은 응용 프로그램을 사용해야 하므로 ARMulator와 호환이 되는 SWI 명령을 처리할 수 있도록 software interrupt routine을 작성하여 OS를 대신할 수 있도록 하였다.[8]

성능 평가에 사용된 LC trie 검색 방식의 라우팅 프로그램은 C language로 작성되었으며 이 프로그램이 올바른 결과를 출력하는가를 검증하기 위하여 동일한 데이터를 이용한 다양한 방법의 추가시뮬레이션을 ADS의 ARMulator와 Windows 환경의 Visual C++을 통하여 검증하였다.

라우팅 프로그램이 사용하는 데이터로써는 LC trie 방식으로 압축된 라우팅 테이블과 이 라우팅 테이블을 이용하여 패킷 포워딩을 수행할 소스 패킷이 있다. 라우팅 테이블은 PC trie 방식의 라우팅 테이블로부터 Level Compress 압축 기법을 참고하여 LC trie 방식의 라우팅 테이블로 압축하였다. 처리될 패킷은 난수 발생기에 기반한 패킷 제너레이터를 이용하지 않고 실제 패킷을 이용하였는데 이는 난수 발생기를 기반으로 한 패킷 제너레이터로는 멀티미디어 스트림이나 일부 서버에서 특정 서비스 제공을 위해 사용되는 스트림 등과 같은 최근에 사용되는 패킷의 특성들을 모두 반영할 수 없기 때문이다. 패킷의 특성이 일반적인 실제 패킷을 얻어내기 위하여 연세대학교 백본 라우터에서 3일에 걸쳐 각각 2G byte 씩의 패킷을 수집하였으며 백본 라우터와 연결된 포트를 미러링한 후 스니퍼 프로그램을 이용하여 캡춰하는 방식으로 패킷을 채집하였다. 실제 입력 패킷 데이터는 캡춰된 Ethernet layer frame에서 헤더 분석을 통하여 IP packet을 골라내어 사용하였다.

성능 측정의 기준으로는 초당 처리된 패킷 수(Packet Per Second)와 초당 처리된 비트 수(Bit Per Second)를 사용할 수 있다. 그러나 초당 처리된 비트 수를 사용하는 경우 프로세서의 성능보다는 패킷의 특성에 따라서 그 성능 변화 폭이 달라질 수 있다. 따라서 보다 객관적이고 합리적인 BPS 수치를 제시하기 위하여 패킷의 크기를 64bytes로 고정시켜 표현하였다.[9] 실제로 채집한 초기 패킷의 평균 크기는 917 bytes 이었으며 실험 시 기준으로 잡은 64 bytes에 비해 약 14배의 차이가 난다. 본 논문의 시뮬레이션 수행 시 패킷의 크기를 64bytes로 고정 시켰으며 이 과정에서 채널상의 패킷의 오류에 관련된 특성 변화를 막기 위하여 패킷 헤더와 checksum 부분을 상대적으로 변형시키므로써 TTL과 checksum fail 빈도를 패킷의 변형전과 동일한 수치로 유지되도록 조절하였다.

V. 성능 평가

5.1 쓰레드의 수

SMT 구조에서 동시에 몇 개의 쓰레드를 실행할 것인가에 대한 결정은 매우 중요하다. 적은 수의 쓰레드를 사용하는 경우 TLP를 통한 이득을 얻을 수 없게되고 너무 많은 수의 쓰레드를 사용하게 되면 TLP를 통해 얻어지는 이득에 비해 하드웨어 구조가 지나치게 복잡해지는 단점이 있다. 그림 4는 쓰레드에 따른 프로세서의 성능 변화를 나타낸다. 쓰레드의 증가에 따라 페치 width 뿐 아니라 BTB(Branch Target Buffer)와 캐쉬 그리고 각 기능 유닛의 수 등의 하드웨어 자원을 쓰레드의 수와 비례하여 변화시키면서 성능을 비교하였다.

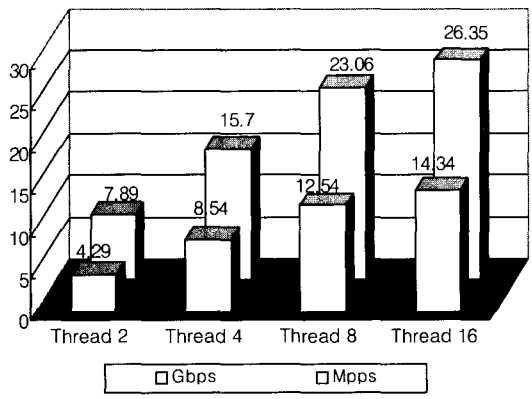

그림 8. 쓰레드 수에 따른 성능 변화

그림에서 알 수 있듯이 8개의 쓰레드를 사용하는 경우까지는 성능향상의 폭이 크지만 16개의 쓰레드를 사용하는 경우는 8 쓰레드를 사용하는 것에 비해 성능 향상 폭이 그리 크지 않다. 이는 16 쓰레드의 사용이 8 쓰레드를 사용하는 것에 비해 TLP를 통해 얻을 수 있는 이득의 폭이 그다지 증가하지 않을 뿐 아니라 쓰레드간 경쟁적인 메모리 엑세

스를 통한 캐쉬 쓰래싱 현상 등에 의해 오히려 성능이 떨어질 수 있다.

위의 시뮬레이션 결과와 하드웨어의 복잡도를 고려하였을 때 쓰레드의 수를 8로 설정하는 것이 자원 효율과 성능 면에서 유리하다.

5.2 그룹핑 옵션

그림 5는 그룹핑 옵션에 따른 성능 변화를 나타낸 것이다. 그룹핑 1은 페치 셀렉트 단을 1개, 2개, 4개의 그룹으로 각각 나누었을 때의 성능 변화를 기록한 것이다. 그룹핑 1에서는 2개의 그룹으로 나누는 것이 그룹핑을 하지 않은 것보다 더 좋은 성능을 보이는데 이것은 그룹핑을 통해 명령어 페치 자원의 사용이 특정 쓰레드들에게 집중되는 현상이 방지되기 때문이다. 또한 그룹핑 1에 의한 최대 그룹 수는 쓰레드가 아닌 명령어 캐쉬, ITLB 및 BTB의 포트 수에 의해 제한된다. 결과적으로는 그룹핑 1의 개수를 늘리는 것은 포트수의 증가를 야기시키므로써 설계 복잡도를 증가시키게 된다.

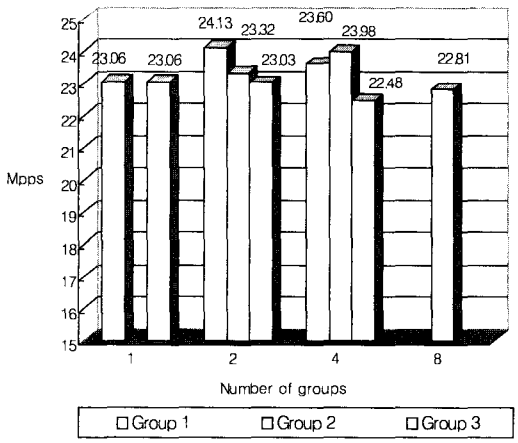

그림 9. 그룹핑에 따른 성능 변화

그룹핑 2의 경우 이슈 큐와 명령어 페치 큐에 관한 그룹설정을 옵션으로 하여 시뮬레이션 하였다. 그림 5를 통해 알 수 있듯이 2개의 그룹으로 나눈 것보다 4개의 그룹으로 나눈 것이 더 높은 성능을 보이는데 이는 그룹핑 1에서와 마찬가지로 그룹핑으로 인하여 명령어 이슈 큐와 페치 큐가 쓰레드간 자원을 균등하게 배분하므로써 그룹핑이 오히려 성능 향상에 긍정적인 영향을 준 것이다.

그룹핑 3은 하드웨어 간략화에 가장 큰 영향을 미치는 부분이다. 그림 5의 결과를 보면 그룹핑 3은 그룹핑에 의해 성능이 저하되는 것을 볼 수 있다. 이것은 각 쓰레드별 하드웨어의 공유가 제대로 이루어지지 않기 때문이다. 그룹핑 3을 8개의 그룹으로 나눈다는 것은 CMP 와 마찬가지로 서로 다른 쓰레드간의 실행 유닛을 공유하지 않게 되므로 SMT 구조의 특징을 전혀 이용하지 못하는 것이 된다. 그룹핑 3의 결과에서 1개의 그룹으로 8개의 쓰레드를 통합하는 것과 2개의 그룹으로 나누는 두 가지 옵션에 대해서 비슷한 결과를 보이면서도 하드웨어적인 측면으로는 레지스터 파일과 실행 유닛 간의 포트를 줄이므로 써 설계 간소화 및 각 블록의 실행 속도를 빠르게 만들 수 있는 장점이 있다. 따라서 그룹핑 3은 2개의 그룹으로 나누는 것이 가장 효과적이다.

최종적으로 그룹핑 옵션은 그룹핑 1/2/3에 대하여 2/4/2 가 가장 최적화된 옵션이 된다.

5.3 페치 우선 정책과 이슈 우선 정책

페치 우선 정책에 의한 성능상의 차이는 무시할 수 있는 정도이다. 이러한 이유는 본 SMT 구조에서 각 쓰레드들이 사용하는 어플리케이션이 모두 동일하게 LC trie 구조의 라우팅 테이블을 검색하는 프로그램이라는 특징 때문이다. 이러한 특징으로 말미암아 프로그램내의 특정 명령어 비율이 일정하게 유지되기 때문에 페치 우선 정책의 영향보다는 그룹화나 페치 큐의 크기에 더 큰 영향을 받게된다.

그림 6은 이슈 우선 정책에 따른 성능 변화를 나타낸 그래프이다. OLDEST란 이슈 큐 내의 가장 오래된 명령어에 대해 가장 높은 우선권을 주는 방

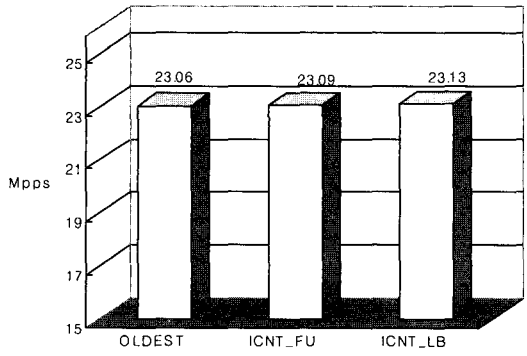

그림 10. Issue 옵션에 따른 성능 변화

식이다. ICNT_FU는 기능 유닛들을 가장 적게 점유하고 있는 쓰레드에 대하여 우선권을 주는 방식이며 ICNT_LB는 Load buffer를 점유하지 않은 쓰레드에 가장 높은 우선권을 주는 방식이다.

시뮬레이터에서 사용되고 있는 LC trie 기반의 라우팅 테이블 lookup 알고리즘이 프로세서에서 프로그램으로 실행될 때 주요 병목현상이 발생될 것으로 예측되는 연쇄적인 메모리 엑세스 부분에 대해 이슈 단에서 이러한 특징을 반영한 ICNT_LB 정책이 높은 성능을 보일 것으로 예상되었으나 세 가지 우선 정책에 대한 결과를 살펴보면 페치 우선 정책과 마찬가지로 성능차이에 그다지 큰 영향을 주지 못함을 알 수 있었다.

이러한 이유는 SMT 구조에서 서로 다른 쓰레드 간의 병렬성을 이용하는 특성으로 인해 분기예측 실패와 메모리 액세스 지연에 대한 immunity를 가진다는 특성이 반영된 것이다.

5.5 명령어 이슈 큐 및 페치 큐의 크기

그림 7은 페치 큐와 이슈 큐의 크기를 조절하여 성능을 측정한 결과이다.

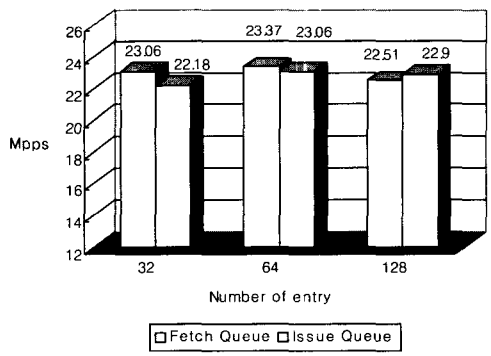

그림 11. Queue 크기에 따른 성능 변화

명령어 페치 큐의 경우 64개의 엔트리를 사용하는 것이 가장 바람직한 결과로 나왔다. 이는 페치 큐의 크기가 지나치게 커질 경우 페치 큐를 채울 때 특정 쓰레드의 명령어가 페치 큐의 많은 엔트리를 차지하는 현상이 발생하여 결과적으로 자원의 활용이 특정 쓰레드로만 몰리기 때문이다.

명령어 이슈 큐의 경우 64개의 엔트리를 이용하는 것과 128엔트리를 이용하는 두 가지 경우 사이의 성능차이가 거의 없기 때문에 마찬가지로 64개의 엔트리를 사용하는 것이 바람직하다.

5.6 분기 예측기 옵션

분기 예측기의 성능은 본 구조의 프로세서에서는 큰 차이를 보이지 않는다.

그림 8을 보면 BTB의 엔트리 수나 PHT의 크기에 따른 성능향상이 거의 보이지 않고 있다. 이러한 결과를 보이는 이유는 LC trie 알고리즘 내에 분기 명령어가 거의 쓰이지 않기 때문이다. 앞장에서 밝혔듯이 LC trie 알고리즘은 하위 노드로의 검색 시 분기 명령어를 통한 제어가 아닌 포인터를 이용한 메모리 레퍼런스를 사용하므로써 프로그램의 명령어 레벨에서 분기 명령어를 제거하였다.

실제로 수행된 프로그램에서 분기 명령어의 비율은 3% 정도로써 다른 어플리케이션에 비해 그 비율이 매우 낮았고, 시뮬레이션의 최소 옵션인 BTB 126, PHT 256 에서의 분기 예측 성공률이 95.5%로 매우 높았다. 따라서 본 SMT 네트워 프로세서에서는 루프 예측을 목적으로 하는 최소한의 분기 예측기로도 성능 저하가 없기 때문에 분기 예측기 부분의 설계는 가능한 최소한의 사양으로 간략화하는 것이 바람직하다.

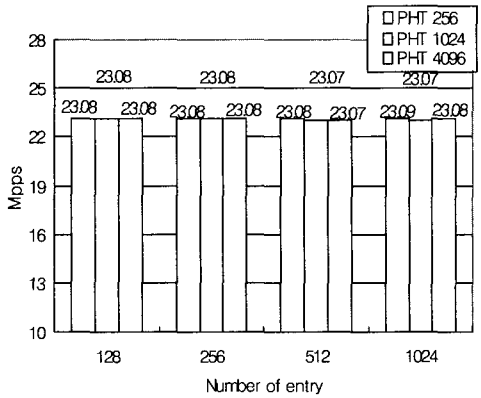

그림 12. 분기 예측기 옵션에 따른 성능 변화

5.7 실행 유닛

SMT 프로세서는 한정된 실행 유닛을 여러 쓰레드들이 균일하게 사용하도록 하여 하드웨어의 사용 효율을 높이므로 써 전체 성능향상을 이루는 구조이다. 따라서 8개의 쓰레드들이 매 사이클마다 적절히 하드웨어 자원을 이용할 수 있을 정도의 충분한

실행 유닛을 확보해야 한다. 그림 9는 실행 유닛의 수를 변경해 가며 성능 변화를 측정한 결과이다.

먼저 시뮬레이션 수행 시 실행 유닛의 개수를 짝수로 정한 것은 그룹핑 3을 2개의 그룹으로 나누는 것이 하드웨어 복잡도를 줄이면서 성능 저하를 최소한으로 할 수 있다는 5.2절의 결과를 고려하여 두 개의 그룹에 동일한 실행 유닛이 들어가도록 했기 때문이다. 그림 9의 결과를 살펴보면 ALU 4개와 LSU 4개만 존재하여도 그 이상의 하드웨어 자원을 이용하는 것에 비해 성능이 더 좋아지지 않는다는 것을 알 수 있다. 이를 통해 그룹핑 3을 기준으로 두 개로 나뉜 각 그룹마다 2개의 ALU와 2개의 LSU만으로도 성능 저하 없이 설계 복잡도를 줄일 수 있다는 것을 의미한다.

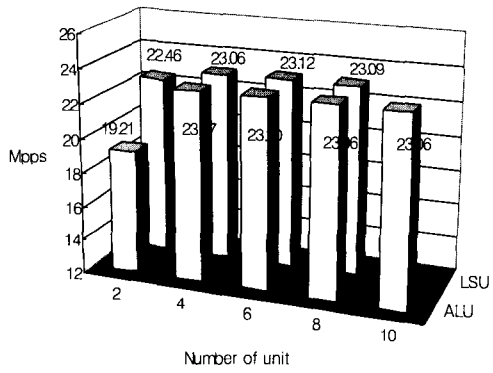

그림 13. 실행 유닛의 개수에 따른 성능 변화

5.8 캐쉬 크기 및 상관성

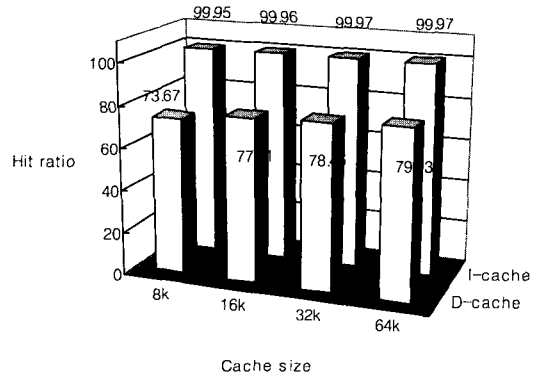

그림 14. 캐쉬의 크기에 따른 hit율 변화

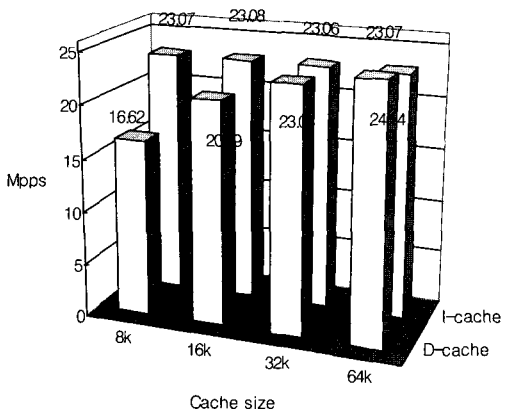

그림 15. 캐쉬 크기에 따른 성능 변화

그림 10과 11은 캐쉬의 크기에 따른 성능 변화를 나타낸 그림이다. 명령어 캐쉬의 경우 라우팅 프로그램의 크기가 그다지 크기 않고 프로그램의 플로우가 단순하기 때문에 시뮬레이션 옵션 중 가장 적은 용량인 8k byte로도 성능저하가 거의 없는 것을 관찰 할 수 있다. 데이터 캐쉬의 경우 캐쉬의 크기가 커짐에 따라 hit 율과 최종 성능이 계속 증가하는 결과를 보인다. 그 중 32k byte 와 64k byte의 성능 변화율을 비교하면 hit 율이 0.58% 증가했고 전체 성능은 4.7% 정도 증가됨을 보인다. 그러나 이는 16k byte 와 32k byte의 경우 hit율 1.3%, 전체 성능 9.7% 가 증가되는 것에 비하여 성능향상 폭이 그다지 크지 않음을 알 수 있다. 따라서 명령어 캐쉬의 경우 8K byte, 데이터 캐쉬의 경우 32k byte로 설계함이 가장 합리적인 설계가 된다.

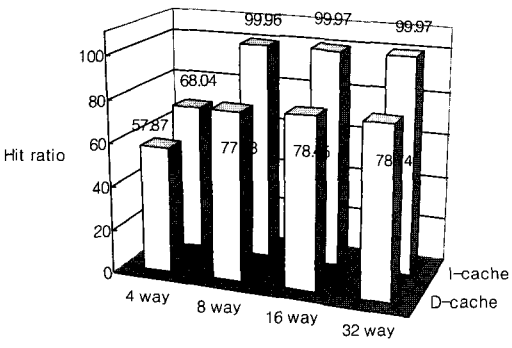

그림 16. 캐쉬의 way수에 따른 hit 율 변화

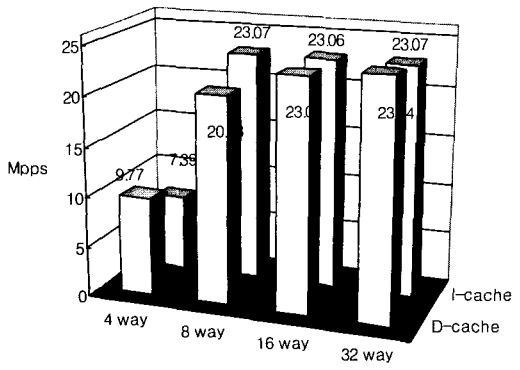

그림 17. 캐쉬 way 수에 따른 성능 변화

그림 12와 13은 캐쉬의 way 수에 따른 성능 변화를 나타내고 있다. 명령어 캐쉬의 경우 8 way부터 거의 일정한 hit율과 성능을 보이고 있다. 이는 어플리케이션의 크기가 작고 그 플로우가 단순하기 때문에 한 쓰레드마다 1 way의 명령어 캐쉬로도 그 기능을 충분히 수행할 수 있으며, 따라서 최대 쓰레드 숫자인 8 정도의 way 수 만을 확보해도 hit율에 변화를 주지 못한다. 데이터 캐쉬의 경우 명령어 캐쉬보다 더 큰 수치인 16 way 정도에서 더 이상 성능향상이 나타나지 않는 현상이 관찰된다. 이는 각 쓰레드의 동작 시 메모리로부터 라우팅 테이블과 패킷이라는 두 가지 데이터를 엑세스하기 때문에 데이터 캐쉬의 way 수가 8 way로는 부족하기 때문이다. 따라서 명령어 캐쉬의 경우 최대 쓰레드 숫자인 8 만큼의 way 수만을 확보해도 성능저하가 일어나지 않고, 데이터 캐쉬의 경우에는 최대 쓰레드의 2배에 해당하는 16 way의 캐쉬를 사용해야 한다.

5.9 최적화된 SMT 구조

다음 표는 앞서 서술한 다양한 시뮬레이션 결과를 이용하여 도출된 SMT 구조를 정리한 것이다.

항 목	결 과	분 석
수행방식	In-order	설계의 단순화
ISA	ARM	호환성 고려
쓰레드	8	TLP 이용률 극대
그룹핑	2/4/2	설계 단순화
명령어 캐쉬	8k, 8way	어플리케이션 특성이 좋으므로 최소화가능
데이터 캐쉬	32k, 16way	쓰레드 * 2 만큼의 way수 필요
실행 유닛	4 ALU, 4 LSU	8쓰레드 지원을 위한 최소한의 자원
분기 예측기	BTB 128 entry	성능에 영향을 주지 않으므로 최소화가능
패치 큐 크기	64 entry	2 쓰레드마다 8 entry
이슈 큐 크기	64 entry	2 쓰레드마다 8 entry

표 3. 최적화된 SMT구조의 옵션

위와 같이 최적화된 순차적 SMT 구조는 앞선 시뮬레이션 결과 그래프에서 확인할 수 있듯이 500MHz 클럭을 가정한 경우에 약 23Mpps의 성능을 보인다. Crowley의 연구에서 500MHz로 동작하는 비순차 SMT의 ip4 포워딩 성능이 20~25Mpps 사이에 있는 것과 비교하면 본 연구의 순차적 SMT가 성능과 하드웨어 복잡도 측면에서 충분히 경쟁력이 있음을 알 수 있다.[4]

VI. 결 론

SMT 네트웍 프로세서는 쓰레드간의 명령어 병렬성을 이용하여 하드웨어 자원을 효율적으로 이용하므로써 높은 성능을 보이는 구조이다. 그러나 SMT 구조가 가지는 설계복잡도로 인하여 상용화 단계에 이르지는 못하고 있다. 따라서 본 논문에서는 실질적인 SMT 네트웍 프로세서의 설계를 위한 하드웨어 최적화를 제시하였다.

기본적으로는 ARM의 ISA를 사용하는 in-order 방식의 SMT 구조로써 8 쓰레드의 동시 실행을 지원한다.

지원하는 쓰레드의 수가 증가하는 것은 그 만큼

의 하드웨어 복잡도로 이어지기 때문에 TLP의 이용률을 고려하여 쓰레드 수를 정해야 한다.

그룹핑 1, 2는 통하여 설계 복잡도를 줄이는 역할 뿐 아니라 특정 쓰레드의 명령어가 집중적으로 큐에 몰리게되어 발생하는 성능저하를 줄여주는 효과가 있으며 그룹핑 3은 하드웨어의 복잡도를 가장 크게 줄일 수 있는 부분이 된다.

명령어 캐쉬는 어플리케이션의 크기와 특성에 따라 작은 용량의, 그리고 쓰레드 수와 동일한 way만큼의 캐쉬로 설계하는 것이 바람직하며 데이터 캐쉬의 경우 어플리케이션의 주요 데이터가 라우팅 테이블과 패킷이라는 두 가지의 독립적인 데이터이기 때문에 지원하는 쓰레드의 2배에 해당하는 way 수가 필요한 특징이 있다.

실행 유닛은 그룹핑 3을 기준으로 하여 각 그룹마다 4개의 쓰레드를 동시에 실행 할 수 있도록 4개 이상의 실행 유닛이 필요하다.

그 외 BTB의 크기나 명령어의 페치 큐와 이슈 큐의 크기, 그리고 명령어 페치 및 이슈 정책은 전체 성능에 큰 영향을 주지 못함을 알 수 있었다.

참 고 문 헌

[1] Miguel A. Ruiz-Sanchez, Ernst W. Biersack, Walid Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", March/April 2001 IEEE Network, pp. 8-23

[2] Stefan Nilsson and Gunnar Karlsson, "IP-Address Lookup Using LC-Tries", IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 17, No. 6, JUNE 1999, pp. 1083-1092

[3] Stefan Nilsson and Gunnar Karlsson, "Fast address lookup for Internet routers", Proceedings of "Algorithms and Experiments" Trento, Italy, Feb, 1998, pp. 9-18

[4] Patrick Crowley, Marc E. Fiuczynski, Jean-Loup Baer, and Brian N. Bershad, "Characterizing Processor Architectures, for Programmable Network Interfaces", 2000 International Conference on Supercomputing, Santa Fe, N.M., May, 2000

[5] Hily S, and Seznec A, "Out-of-order execution may not be cost-effective on processors featuring simultaneous multi-threading", High-Performance Computer Architecture, 1999. Proceedings. Fifth International Symposium On , 9-13 Jan 1999, Pages: 64 -67

[6] Dean Tullsen, Susan Eggers, and Henry Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism", Proceedings of the 22nd Annual International Symposium on Computer Architecture, June 1995, pages 392-403.

[7] ARM, ARM Architecture Reference Manual, Part A CPU Architecture, 1996

[8] ARM, ARM Developer Suit version 1.1, Compiler, Linker and Utilities Guide, 1999

[9] David Meng, Eswar Eduri Manohar Castelino, "IXP 2400 Intel Network Processor IPv4 Forwarding Benchmark Full Disclosure Report for Gigabit Ethernet", Network Processing Forum, March 2003

임 정 빈(Lim Chung Bin)
2002년 2월 : 연세대학교 전자공학과 졸업
2004년 2월 : 연세대학교 전기전자공학과 석사
<주관심 분야> 마이크로프로세서, 네트워크프로세서

홍 인 표(Hong In Pyo)
1999년 2월 : 연세대학교 전자공학과 졸업
2001년 2월 : 연세대학교 전기전자공학과 석사
2001년 3월 - 현재 : 연세대학교 전기전자공학과 박사과정
<주관심 분야> 마이크로프로세서, 네트워크프로세서

조 정 현(Cho Jung Hyun)
1997년 3월 - 현재 : 연세대학교 기계전자공학부 재학
<주관심 분야> 마이크로프로세서, 네트워크프로세서

이 용 석(Lee Yong Surk)
1973년 2월 : 연세대학교 전기공학과 졸업
1977년 2월 : Univ. of Michigan, Ann Arbor 석사
1981년 2월 : Univ. of Michigan, Ann Arbor 박사
1993년 3월 - 현재 : 연세대학교 전기전자공학과 교수
<주관심 분야> 마이크로프로세서, 네트워크프로세서