# 파이프라인 아키텍처 기반의 네트워크 프로세서를 이용한 고속 패킷 처리에 관한 연구

손경덕, 진현정, 김화종
강원대학교 전자공학과

# A Study on Fast Packet Processing Using Pipeline Architecture-Based Network Processors

Kyoung-Duk Son, Hyun-Jung Jin, Hwa-Jong Kim
Dept. of Electronics Engn., Kangwon National University

**Abstract** - 본 논문에서는 파이프라인 아키텍처 기반의 네트워크 프로세서를 이용한 네트워크 시스템 개발에 대해 다룬다. 파이프라인 아키텍처는 구조상 Hazards 문제가 발생할 수 있으며 이는 시스템의 성능에 중요한 영향을 주게 된다. 또한 네트워크 프로세서는 고수준의 프로그래밍 모델을 제공하므로 고속의 패킷 처리를 위한 코드 작성이 수월하다. 따라서 파이프라인 아키텍처 기반의 네트워크 프로세서를 이용한 시스템 개발시 Hazards 문제를 피할 수 있는 방법과 효율적인 패킷 처리를 위한 코드 작성에 대한 지침을 제시하고 그 방법이 일반적인 방법보다 효율적임을 확인하였다.

## 1. 서 론

인터넷은 모든 네트워크에 영향을 주게 되었으며 새로운 네트워크 아키텍처와 기술은 인터넷을 기반으로 발전하고 있다. 급속한 인터넷의 확장은 다양하고 새로운 네트워크 시스템과 기술의 발전을 야기시켰다. 인터넷은 패킷 스위칭 기술을 기반으로 발전되었기 때문에 패킷 처리는 모든 네트워크 시스템에서 기초가 된다. 하나 또는 그 이상의 입력에서 동시에 도착하는 패킷을 모두 처리해야 하는 네트워크 시스템의 대부분은 소프트웨어 또는 하드웨어 기반으로 개발 되었으며 이는 각각 성능과 확장성 면에서 장·단점을 가지게 되었다. 또한 10Gbit 환경을 넘어서면서 네트워크 시스템의 성능은 중요한 이슈로 떠오르고 있다.

본 논문에서는 패킷 처리의 성능과 확장성을 높이기 위한 하나의 방안으로 떠오르고 있는 네트워크 프로세서 중 파이프라인 아키텍처를 가진 EZchip을 이용하여 네트워크 시스템 개발시 주의할 점과 패킷 처리의 성능을 향상 시키기 위한 최적화된 코드 작성에 대한 지침을 제시하고자 한다.

본 논문의 구성은 2장에서 네트워크 프로세서의 일반적인 특징과 파이프라인 아키텍처에 대해서 설명하고 EZchip의 주요 구성 컴포넌트와 하드웨어 아키텍처와 함께 프로그래밍 모델에 대해서 소개한다. 3장에서는 EZChip에서의 패킷 처리 모델을 설명하며 코드 작성시 유의할 점과 성능 향상을 위한 최적화된 코드 작성에 대한 지침을 제시하고 4장에서 성능을 측정하고 마지막으로 5장에서 결론을 맺는다.

## 2 네트워크 프로세서 아키텍처

### 2.1 네트워크 프로세서의 특징

네트워크 프로세서는 여러 개의 컴포넌트가 복잡하게 구성되어 있으며 네트워크 프로세서의 공통적인 특징은 아래와 같다[2,5].

- o 프로세서 구조
- o 메모리 구조
- o 내부 데이터 전송 메커니즘
- o 외부 인터페이스와의 통신 메커니즘
- o 프로그래밍 모델

• 프로세서의 구조

다양한 패킷 처리를 수행하는 하드웨어로 정해진 프로세서와 프로그래밍 기능을 포함하고 있으며 이 구조는 네트워크 시스템을 저수준에서 고수준으로 확장시키며 일반적인 CPU처럼 하나의 작업을 수행한다. 하지만 네트워크 프로세서는 패킷 처리를 위한 모든 요소를 포함하고 있지 않다. 대신 네트워크 프로세서는 다른 레벨의 패킷을 처리하기 위해 프로세서 내부에 다른 하드웨어를 포함할 수도 있으며 외부 하드웨어와 함께 동작할 수도 있다.

• 메모리 구조

메모리는 저비용으로 높은 성능을 구현하는데 중요한 요소이다. 이는 네트워크 프로세서를 이용하여 시스템을 구현할 때 전체 비용을 감소시킬 수 있다는 점에서 필수적인 보완물이라 할 수 있다.

• 내부 데이터 전송 메커니즘

네트워크 프로세서와 내부의 다른 하드웨어들과의 데이터 전송을 담당한다. 패킷은 메모리에 위치하고 내부 데이터 전송 메커니즘은 패킷 헤더로부터 정보를 추출하거나 패킷의 포인터 값을 전송하는데 사용된다. 대부분의 내부 데이터 전송 메커니즘은 충분한 전송 대역폭을 제공하기 위해 다중 전송 메커니즘을 포함하고 있으며 대표적으로 internal bus, hardware FIFO, transfer register, onboard shared memory를 사용하고 있다.

• 외부 인터페이스와 통신 메커니즘

네트워크 프로세서는 외부 하드웨어와의 통신 방법을 제공하고 있으며 이는 크게 표준/특수 bus 인터페이스, 메모리 인터페이스, 다이렉트 I/O 인터페이스 그리고 switching fabric 인터페이스 등을 가진다. 이러한 bus들은 PCI와 같은 표준 방식과 Network Processor Forum(NPF)에서 명시한 LA-2 버스와 같은 특수한 형태의 bus 인터페이스를 가진다.

메모리 인터페이스는 외부 메모리의 접근에 사용되며 대부분의 경우 bus를 이용하여 연결된다. 다이렉트 I/O 인터페이스는 bus를 이용하여 고속의 I/O 디바이스와 연결될 수 있으며 Switching Fabric 인터페

이스는 CSIX 표준과 같은 특수한 타입의 switching fabric과 연결할 수 있다.

• 프로그래밍 모델

네트워크 프로세서는 프로그래밍 모델을 지원하고 있으며 이는 확장성을 크게 향상시켰다고 할 수 있다. 네트워크 프로세서는 크게 '비동기 이벤트 핸들러'와 '통신 스레드'의 두 가지 프로그래밍 모델을 제공한다. 비동기 이벤트 핸들러 모델은 프로그래머가 각 이벤트마다 이와 관련된 핸들러를 구현하는 방식이며 통신 스레드 모델은 무한 루프를 수행하면서 처리할 내용이 있는지를 확인하는 전형적인 절차적 프로그래밍 방식이다.

## 2.2 네트워크 프로세서 아키텍처와 패킷 흐름

[그림 1], [그림 2], [그림 3]은 네트워크 프로세서의 아키텍처와 이에 따른 패킷의 흐름을 나타내고 있다.

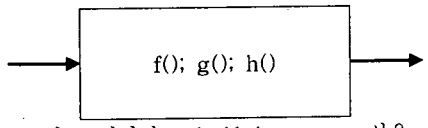


그림 1. 하나의 embedded processor 사용

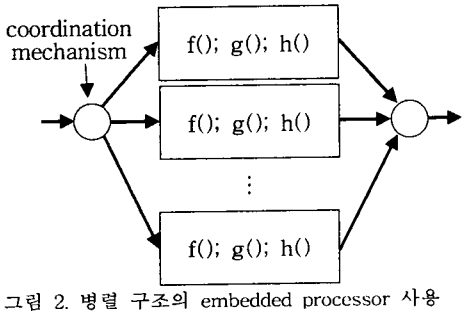


그림 2. 병렬 구조의 embedded processor 사용

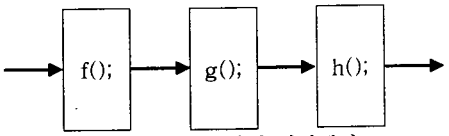


그림 3. 파이프라인 아키텍처

[그림 1]은 하나의 임베디드 프로세서를 가지고 있는 네트워크 프로세서에서의 패킷 흐름을 보여주고 있으며 [그림 2]는 N개의 임베디드 프로세서를 사용하는 네트워크 프로세서에서의 패킷 흐름을 보여주고 있다. [그림 2]에서 각 프로세서는 약 1/N개의 패킷을 처리하며 또한 각 프로세서에 패킷을 균등하게 배분해 주기위한 추가적인 메커니즘을 필요로 한다. [그림 3]은 파이프라인 기반의 네트워크 프로세서를 보여주고 있다. 파이프라인 기반의 네트워크 프로세서는 각 스테이지마다 할당된 고유의 작업이 있으며 그 작업을 수행한 후에 다음 스테이지로 패킷을 전달한다.

## 2.3 EZchip 아키텍처

파이프라인 아키텍처 기반의 EZchip은 [그림 4]에서와 같이 고유의 패킷 처리 작업을 수행하는 다중 스테이지로 구성되어 있다[3].

### 2.3.1 EZchip 하드웨어 아키텍처(컴포넌트)

EZchip은 고유한 패킷 처리 작업을 수행하기 위한 Task Optimized Processor(TOP)의 배열 구조를 가지며 각 스테이지의 작업을 위한 고유의 instruction set을 가진다.

TOPs는 parse, search, resolve, modify 4개로 구성되고 search는 다시 search I과 search II로 세분화되어 있으며 search II는 필수요소로 정의되어 있지 않다.

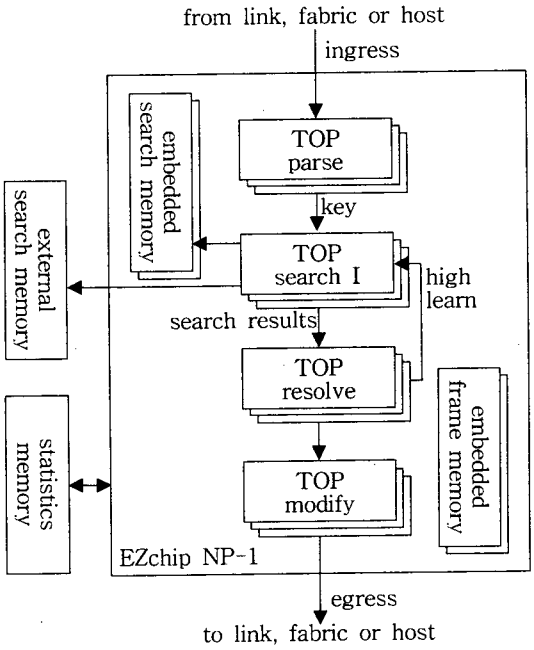


그림 4. EZChip 아키텍처

• TOPparse

TOPparse는 MAC 주소나 VLAN 또는 MPLS 태그들과 같이 패킷의 7 Layer와 관련된 필드나 프레임 헤더를 decode하고 분석하여 key를 생성한다. TOPparse가 얻은 key는 TOPsearch로 전달된다.

• TOPsearch

TOPparse로부터 전달받은 key를 이용하여 table look-up을 수행하며 radix tree, hash table, direct access table 등의 데이터 구조를 지원한다. 이 table은 3개의 내부 메모리에 저장되고 TOPsearch는 microcode를 통해서 look-up의 chaining을 지원하며 이 microcode는 TOPsearch 엔진에 의해서 실행된다. look-up 수행의 모든 결과는 TOPresolve로 전달된다.

• TOPresolve

TOPresolve는 TOPsearch에서 전달받은 결과를 분석하여 패킷의 포워딩 또는 수정을 결정하며 'high-learn'이라는 인터페이스를 가진다. 'high-learn'은 table에 들어있는 패킷의 entry가 생성, 수정, 삭제되도록 하며 이 table은 TOPsearch가 host의 개입 없이 table look-up을 수행하도록 한다.

• TOPmodify

TOPmodify는 TOPresolve의 의사결정에 따라서 새로운 데이터의 추가, 이전 데이터의 제거, 현재 contents의 수정 작업을 수행한다. 예를들어, 패킷이 수정되면 checksum을 다시 계산하고 전송을 위해 패킷을 egress queue에 넣는 작업을 수행한다.

### 2.3.2 EZchip 프로그래밍 모델

파이프라인 기반의 EZchip은 TOPs에서는 어셈블리 기반의 프로그래밍 모델을 지원하며 NP-1을 초기화하고 구동시키기 위한 NP Script Language(NPsl)를 지원하고 있다. 또한 EZdriver를 이용하여 관리를 목적으로 하는 호스트와의 통신을 지원한다.

• TOPs 프로그래밍

각 컴포넌트는 고유의 엔진을 가지고 있으며 엔진 내부의 요소들은 버스를 통해 연결되어 있다. 엔진 내부의 각 요소는 고유의 이름이 부여되어 있으며 이 이름이 프로그램 작성에 이용된다.

• NPsl 프로그래밍

NPsl은 NP-1 네트워크 프로세서의 초기화를 담당하는 스크립트 작성에 사용된다. NPsl은 직접 읽을 수 있는 텍스트 형식으로 작성되며 드라이버를 위한 파라미터와 함께 API를 호출한다.

• EZdriver 프로그래밍

EZchip Driver API는 [그림 5]와 같이 PIC bus를 이용하여 NP-1 네트워크 프로세서와 관리 호스트간의 인터페이스를 제공한다. EZdriver는 NP-1 chip의 환경설정, microcode의 로딩과 lookup을 위한 구조체의 생성과 관리를 담당하며 네트워크 프로세서와 패킷 정보 등을 주고 받는다.

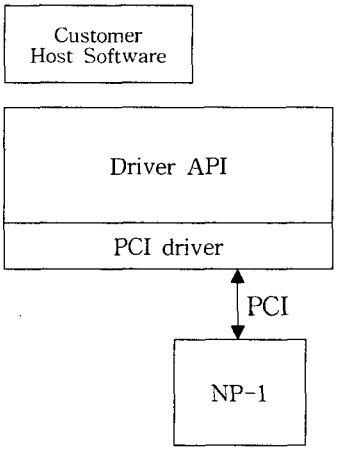


그림 5. EZchip Driver API

## 3. 효율적인 네트워크 프로세서의 코드 작성

EZchip의 프로그래밍 언어인 microcode는 C 스타일의 코드와 어셈블리 스타일의 코드가 혼합된 형식을 보이며 대부분의 명령은 어셈블리 명령어를 통해서 이루어지고 각 요소의 이름은 예약어로 정의되어 있다.

### 3.1 Data Hazards 제어

파이프라인 기반의 EZchip은 4개의 스테이지로 구성되며 각 스테이지는 1 클럭 사이클을 갖는다. 스테이지 1은 메모리에서 명령을 불러오거나 저장하고 스테이지 2는 불러온 명령을 번역한다. 스테이지 3은 번역한 명령을 보고 패킷 처리 방법을 결정하며 스테이지 4에서 해당 명령을 수행한다. [그림 6]은 4개의 명령 수행 과정을 보여주고 있다.

각 클럭마다 서로 다른 명령어를 불러옴으로써 4 사이클의 수행을 시작한다. 이전 스테이지(F)의 수행이 완료되기 전에는 다음 스테이지(D)가 수행을 시작할 수 없으며 이러한 현상을 hazards라 한다. 이는 delay를 발생시킬 수 있으며 시스템 성능의 중요한 요소이다.

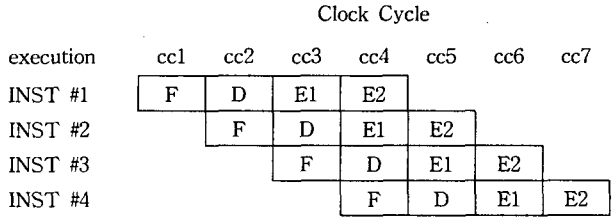


그림 6. 명령어 실행 과정

명령어는 E1, E2에서 실행되며 E1은 조건을 검사하여 다음에 실행될 코드를 결정한다. 그러나 이 결과는 INST #2, INST #3이 수행이 시작되었음에도 불구하고 clock cycle 4(cc4)까지는 유효한 데이터라고 할 수 없다. 결국 INST #4의 F가 INST#1의 E1에서 결정한 결과를 읽을 수 있으며 INST #2, #3에는 데이터 처리 코드가 수행되지 않도록 해야 다음 스테이지에서 실행될 코드를 결정하는 과정에서 오류를 제거할 수 있다([그림 7]).

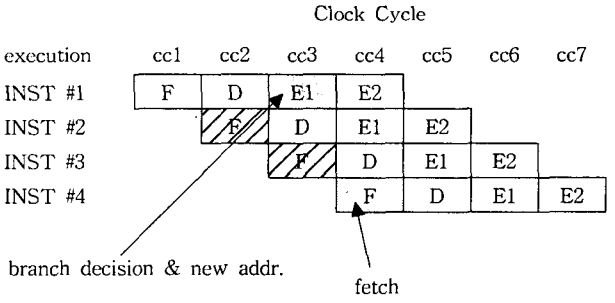


그림 7. 코드 분기의 실행

INST #2, #3의 F는 실행되지 않도록 해야만 파이프라인 아키텍처 기반의 Data Hazards 문제를 피할 수 있으며 이는 _NOP 관련 상수를 이용하여 INST #2, #3의 스테이지 F는 실행을 하지 못하게 함으로써 해결할 수 있다([그림 8]).

```
Mov UREG[1], 2, 2;
if (UREG[0].BIT[3])
    jmp DISCARD | _NOP0
mov UREG[2], UREG[3], 4;
mov UREG[3], UREG[1], 4;
...
DISCARD :
    halt;
```

그림 8. _NOP을 이용한 microcode 예제

### 3.2 효율적인 패킷 처리

#### 3.2.1 필터링 룰의 순서 조정

웹 트래픽을 측정할 경우 프레임의 type 필드는 2-octet으로 0x0800의 값을 가지고 IP 데이터그램은 1-octet으로 6 그리고 TCP 세그먼트의 포트 필드는 2-octet으로 80의 값을 가진다. 이의 경우 C 스타일의 pseudo 코드는 [그림 9]와 같다[1,2,4].

```
if ((frame type == 0x0800) &&
   (IP type == 6) && (TCP port == 80))
    declare the packet matches the classification
else
    declare the packet does not match
```

그림 9. 웹 트래픽 측정 코드

이와 같은 경우 시스템의 목적에 맞게 룰의 순서를 조정하여 최적화 코드를 작성할 수 있는데 예를 들어 프레임의 95%가 IP이고 IP 데이터그램의 92%가 TCP이며 TCP 세그먼트의 85%가 가지는 목적지 포트가 80이라면 [그림 9]의 필터링 룰은 비효율적이며 [그림 10]과 같이 웹 트래픽 측정을 최적화 할 수 있다.

```
if ((TCP port == 80) &&
   (IP type == 6) && (frame type == 0x0800))
    declare the packet matches the classification
else
    declare the packet does not match the classification
```

그림 10. 최적화된 웹 트래픽 측정 코드

최적화된 코드의 수행을 보면 제일 먼저 포트 번호를 검사한다. [그림 9]의 코드에서 첫 번째 비교 대상을 살펴보면 5%만이 필터링되는 반면 [그림 10]의 코드는 80의 목적지 포트 번호를 가진 TCP 세그먼트가 85%이므로 한번의 비교를 수행한 후에는 모든 프레임의 15%를 필터링할 수 있게 된다.

## 4. 성능 평가

### 4.1 평가 모델

파이프라인 아키텍처 기반의 EZchip은 TOPparse에서 패킷 처리를 시작하므로 본 논문에서는 패킷을 분류하는 TOPparse에서 성능 평가를 수행하였다.

TOPparse에서 소비되는 클럭수를 비교하기 위하여 동일한 패킷을 전송하는 프로그램을 작성하고 이 패킷을 처리하는 일반적인 코드와 최적화된 코드를 작성하여 각각의 경우에 소비되는 클럭수를 측정하였다.

패킷은 100,000개의 패킷을 전송하도록 하였고 패킷 처리는 parse, resolve, modify의 3 단계로 처리되도록 프로그램을 작성하였으며 parse에서의 정확한 측정을 위하여 resolve, modify는 동일한 코드를 사용하였다.

### 4.2 측정 결과

[그림 10]은 측정 결과를 나타낸 것이다. 가로축은 EZchip의 각 스테이지를 의미하여, 세로축은 각 스테이지에서 소비한 클럭수를 의미한다.

실험결과에 의하면 일반 코드와 최적화 코드는 약 17%의 성능 개선 효과가 있는 것으로 나타났으며 이 결과에서 우리는 패킷을 처리하는 네트워크 시스템에 효과적으로 사용될 수 있음을 확인하였다.

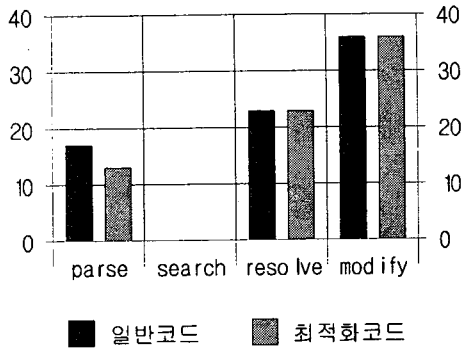


그림 10. 트래픽 룰에 따른 할당 클럭수

## 5. 결 론

본 연구에서는 고속의 패킷 처리를 위한 방법으로 네트워크 프로세서를 이용한 네트워크 시스템 개발시 발생할 수 있는 문제점과 성능을 고려하지 않은 코드의 문제점을 살펴보고, 이들의 해결방법에 관하여 논의하였다.

본 논문에서는 먼저 네트워크 시스템의 특징과 패킷 처리 과정과 파이프라인 아키텍처 기반의 네트워크 프로세서에 대하여 기술하였고, 성능과 확장성을 만족시키기 위해 네트워크 프로세서를 이용한 패킷 처리의 코드 작성법을 제시하고, 성능을 평가하였다. 결과는 일반적인 패킷처리 방법에 비해 최적화된 패킷처리 방법은 약 17%의 성능 향상을 보였다.

인터넷 전송 기술의 발전 과정을 살펴보면 앞으로는 더 높은 성능과 확장성을 요구함과 동시에 여러 분야에서 응용될 수 있는 기술이 요구될 것이다. 네트워크 프로세서는 모든 Layer에서 동작할 수 있으므로 L2, L3 switch, MPLS 트래픽 엔지니어링, 네트워크 모니터링 및 분석, 서버 로드 밸런싱 등 그 응용 분야가 매우 넓다고 할 수 있고 하드웨어 기반의 성능과 소프트웨어 기반의 확장성을 모두 제공하고 있으므로 네트워크 프로세서는 앞으로 발생되는 요구를 수용할 수 있는 기술이라 할 수 있다.

### [참 고 문 헌]

[1] W. Richard Stevens, "TCP/IP Illustrated, Volumn 1", 1994

[2] Douglas E. Commer, "Network Systems Design using Network Processors", 2003

[3] "NP-1 Network Processor Manual", 2002

[4] Wen Xu and Larry Peterson, "Support for Software Performance Tuning on Network Processors", IEEE Network, July/August, 40-45, 2003

[5] Mohammad Peyravian, Gordon Davis, and Jean Calvignac, "Search Engine Implications for Network Processor Efficiency", IEEE Network, July/August, 12-20, 2003