(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2021년01월26일
(11) 등록번호 10-2207775
(24) 등록일자 2021년01월20일

(51) 국제특허분류(Int. Cl.)
H04L 12/935 (2013.01) *H04L 12/26* (2006.01)
H04L 12/947 (2013.01) *H04L 29/06* (2006.01)
(52) CPC특허분류
H04L 49/3063 (2013.01)
H04L 43/18 (2013.01)
(21) 출원번호 10-2019-0051695
(22) 출원일자 2019년05월02일
심사청구일자 2019년05월02일
(65) 공개번호 10-2020-0061280
(43) 공개일자 2020년06월02일
(30) 우선권주장
1020180146372 2018년11월23일 대한민국(KR)
(56) 선행기술조사문헌
KR1019980086572 A*
The P4 Language Consortium, P4_16 Language Specification, The P4 Language Consortium (2017.05.22)*
Vladimir Gurevich, Programmable Data Plane at Terabit Speeds, Barefoot Networks (2017.05.16)*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
연세대학교 산학협력단
서울특별시 서대문구 연세로 50 (신촌동, 연세대학교)
(72) 발명자
김한준
서울특별시 서대문구 연세로 50, 제3공학관 C415호 (신촌동)
송승빈
서울특별시 서대문구 연세로 50, 제3공학관 C423호 (신촌동)
(74) 대리인
특허법인우인

전체 청구항 수 : 총 15 항 심사관 : 윤태섭

(54) 발명의 명칭 **네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법 및 이를 이용한 병렬화 장치**

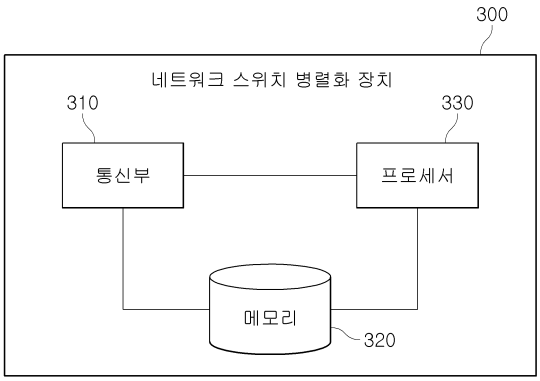
(57) 요 약

네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법 및 이를 이용한 병렬화 장치를 개시한다.

본 발명의 실시예에 따른 데이터 의존성 기반의 데이터 평면 정적 분석 방법은 네트워크 스위치 병렬화를 위하여

(뒷면에 계속)

대 표 도 - 도3



데이터 평면에 대한 정적 분석을 수행하는 방법에 있어서, 테이블의 키와 액션 연산에서 어떤 변수가 읽혀지고(use) 어떤 변수가 쓰여 지는지(def)를 찾아 상기 테이블의 사용-정의(use-def)를 분석하는 사용-정의 분석 단계; 상기 사용-정의에 대한 정보를 이용하여 상기 테이블 간 데이터 의존 관계를 분석하여 상기 테이블 간의 데이터 의존성 정보를 생성하는 데이터 의존성 분석 단계; 및 상기 데이터 의존성 정보를 기초로 키-독립(key-independent) 및 테이블-독립(table-independent) 중 적어도 하나의 독립 테이블을 찾고, 상기 독립 테이블을 룩업 및 테이블 중 하나를 병렬화하는 병렬화 처리단계를 포함할 수 있다.

(52) CPC특허분류
H04L 49/25 (2013.01)
H04L 69/22 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호	2018-11-1699
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원(한국연구재단부설)
연구사업명	정보통신방송연구개발사업
연구과제명	멀티 서비스를 지원하는 프로그래머블 스위치 제어 기술 개발
기 여 율	1/1
과제수행기관명	고려대학교산학협력단
연구기간	2019.01.01 ~ 2019.12.31

공지예외적용 : 있음

명 세 서

청구범위

청구항 1

네트워크 스위치 병렬화를 위하여 데이터 평면에 대한 정적 분석을 수행하는 방법에 있어서,

테이블의 키와 액션 연산에서 어떤 변수가 읽혀지고 어떤 변수가 쓰여 지는지를 찾아 상기 테이블의 사용-정의(use-def)를 분석하는 사용-정의 분석 단계;

상기 사용-정의에 대한 정보를 이용하여 상기 테이블 간 데이터 의존 관계를 분석하여 상기 테이블 간의 데이터 의존성 정보를 생성하는 데이터 의존성 분석 단계; 및

상기 데이터 의존성 정보를 기초로 키-독립(key-independent) 및 테이블-독립(table-independent) 중 적어도 하나의 독립 테이블을 찾고, 상기 독립 테이블을 룩업 및 테이블 중 하나를 병렬화하는 병렬화 처리단계;를 포함하되,

상기 데이터 의존성 분석 단계에서는,

상기 테이블의 키와 액션 간의 참 의존성 및 거짓 의존성을 고려하고, 상기 테이블 간의 테이블 의존성 및 액션 의존성을 정의하여 상기 데이터 의존성 분석을 수행하는 것을 특징으로 하는 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법.

청구항 2

제1항에 있어서,

상기 데이터 의존성 분석 단계는,

현재 테이블의 액션과 후속 테이블의 키 사이의 데이터 의존성(테이블-의존성)과, 상기 현재 테이블의 액션과 후속 테이블의 액션 사이의 데이터 의존성(액션-의존성)을 분석하는 것을 특징으로 하는 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법.

청구항 3

제2항에 있어서,

상기 데이터 의존성 분석 단계는,

이전 연산이 변수에 값을 기록한 후 임의의 연산이 그 변수의 값을 읽으면, 두 연산 사이에 흐름 의존성(flow dependence)이 존재하는 것으로 판단하는 것을 특징으로 하는 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법.

청구항 4

제2항에 있어서,

상기 데이터 의존성 분석 단계는,

이전 연산이 변수에서 읽은 후 임의의 연산이 그 변수에 기록하는 경우 두 연산 간에 반 의존성(anti dependence)이 존재하는 것으로 판단하는 것을 특징으로 하는 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법.

청구항 5

제2항에 있어서,

상기 데이터 의존성 분석 단계는,

이전 연산이 변수에 기록한 후 임의의 연산이 그 변수에 다시 기록하는 경우, 두 연산 사이에 출력 의존성

(output dependence)이 존재하는 것으로 판단하는 것을 특징으로 하는 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법.

청구항 6

삭제

청구항 7

제1항에 있어서,

상기 데이터 의존성 분석 단계에서는,

이전 테이블의 조치에 패킷 헤더 또는 메타 데이터가 정의로 포함되어 있고, 후속 테이블의 키에 변수가 사용으로 포함되어 있으면 두 테이블에 테이블 의존성이 있는 것으로 판단하는 것을 특징으로 하는 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법.

청구항 8

제1항에 있어서,

상기 데이터 의존성 분석 단계에서는,

이전 테이블의 액션에 패킷 헤더 또는 메타 데이터가 사용 또는 정의로 포함되어 있고, 후속 테이블의 동작에 사용 또는 정의의 변수가 들어 있으면 두 테이블이 액션 의존성을 가지는 것으로 판단하는 것을 특징으로 하는 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법.

청구항 9

제1항에 있어서,

상기 병렬화 처리단계는,

인접한 테이블 쌍이 테이블 간에 의존성이 없다면, 키와 액션을 모두 병렬화할 수 있는 것으로 판단하는 것을 특징으로 하는 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법.

청구항 10

제9항에 있어서,

상기 병렬화 처리단계는,

인접한 테이블 쌍이 액션 의존성만 가지고 있다면, 키 매칭을 병렬화하지만 두 액션 간의 데이터 의존성으로 인해 액션을 순차적으로 유지하는 것을 특징으로 하는 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법.

청구항 11

데이터 평면에 대한 정적 분석 기반의 네트워크 스위치 병렬화 장치에 있어서,

테이블의 키와 액션 연산에서 어떤 변수가 읽혀지고 어떤 변수가 쓰여 지는지를 찾아 상기 테이블의 사용-정의(use-def)를 분석하는 사용-정의 분석부;

상기 사용-정의에 대한 정보를 이용하여 현재 테이블의 액션과 후속 테이블의 키 사이의 데이터 의존성(테이블-의존성)과, 상기 현재 테이블의 액션과 상기 후속 테이블의 액션 사이의 데이터 의존성(액션-의존성)을 분석하는 데이터 의존성 분석부; 및

상기 데이터 의존성 정보를 기초로 키-독립(key-independent) 및 테이블-독립(table-independent) 중 적어도 하나의 독립 테이블을 찾고, 상기 독립 테이블을 룩업 및 테이블 중 하나를 병렬화하는 병렬화 처리부;를 포함하되,

상기 데이터 의존성 분석부는,

상기 테이블의 키와 액션 간의 참 의존성 및 거짓 의존성을 고려하고, 상기 테이블 간의 테이블 의존성 및 액션

의존성을 정의하여 상기 데이터 의존성 분석을 수행하는 것을 특징으로 하는 데이터 의존성 기반의 데이터 평면 정적 분석 방법을 이용한 네트워크 스위치 병렬화 장치.

청구항 12

제11항에 있어서,

상기 데이터 의존성 분석부는,

이전 연산이 변수에 값을 기록한 후 임의의 연산이 그 변수의 값을 읽으면, 두 연산 사이에 흐름 의존성(flow dependence)이 존재하는 것으로 판단하는 것을 특징으로 하는 데이터 의존성 기반의 데이터 평면 정적 분석 방법을 이용한 네트워크 스위치 병렬화 장치.

청구항 13

제11항에 있어서,

상기 데이터 의존성 분석부는,

이전 연산이 변수에서 읽은 후 임의의 연산이 그 변수에 기록하는 경우 두 연산 간에 반 의존성(anti dependence)이 존재하는 것으로 판단하는 것을 특징으로 하는 데이터 의존성 기반의 데이터 평면 정적 분석 방법을 이용한 네트워크 스위치 병렬화 장치.

청구항 14

제11항에 있어서,

상기 데이터 의존성 분석부는,

이전 연산이 변수에 기록한 후 임의의 연산이 그 변수에 다시 기록하는 경우, 두 연산 사이에 출력 의존성(output dependence)이 존재하는 것으로 판단하는 것을 특징으로 하는 데이터 의존성 기반의 데이터 평면 정적 분석 방법을 이용한 네트워크 스위치 병렬화 장치.

청구항 15

제11항에 있어서,

상기 병렬화 처리부는,

인접한 테이블 쌍이 테이블 간에 의존성이 없다면, 키 독립적 쌍과 테이블 독립적 쌍을 병렬화할 수 있는 것으로 판단하는 것을 특징으로 하는 데이터 의존성 기반의 데이터 평면 정적 분석 방법을 이용한 네트워크 스위치 병렬화 장치.

청구항 16

제15항에 있어서,

상기 병렬화 처리부는,

인접한 테이블 쌍이 액션 의존성만 가지고 있다면, 두 액션 간의 데이터 의존성으로 인해 액션을 순차적으로 유지하는 것을 특징으로 하는 데이터 의존성 기반의 데이터 평면 정적 분석 방법을 이용한 네트워크 스위치 병렬화 장치.

발명의 설명

기 술 분 야

[0001] 본 발명은 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법 및 이를 이용한 병렬화 장치에 관한 것이다.

배 경 기 술

[0002] 이 부분에 기술된 내용은 단순히 본 발명의 실시예에 대한 배경 정보를 제공할 뿐 종래기술을 구성하는 것은 아니다.

[0003] 네트워킹의 제타 바이트 시대 (zettabyte networking)에서, 네트워크 서비스 제공업체는 빠르고 다양한 네트워크 서비스를 제공해야 한다. 네트워크 서비스 제공업체는 고속의 고정된 기능을 갖는 네트워크 스위치로 낮은 응답 시간의 네트워킹을 구현할 수 있다.

[0004] 또한, 네트워크 서비스 제공업체는 SDN (Software-Defined Networking) 기술로 인해 설정 가능한 네트워크 스위치를 이용하여 네트워크 서비스의 다이버시티(Diversity)를 구현할 수 있다.

[0005] SDN은 네트워크 서비스의 다이버시티를 지원한다. SDN은 네트워크 스위치의 제어 평면을 스위치의 데이터 평면과 분리한다. 여기서, 데이터 평면은 네트워크 패킷을 전달하고, 제어 평면은 전달 규칙을 관리한다. 다양한 네트워크 서비스를 지원하기 위해 네트워크 서비스 제공업체는 제어 평면에서 규칙을 구성한다.

[0006] 전통적인 SDN 기술은 제어 평면에 대한 프로그래밍은 가능하나, 데이터 평면에 대한 프로그래밍은 불가능하다. 이에 따라, 네트워크 서비스 제공업체는 다양한 프로토콜을 제공하거나 고정된 기능의 네트워크 스위치로 새로운 기능을 추가 할 수 없다.

[0007] 이러한 네트워크 스위치에는 프로그래밍할 수 없는 데이터 평면이 포함되어 있기 때문에 네트워크 서비스 제공업체가 제어 평면을 통해 제공하는 실행 규칙 (또는 서비스)은 데이터 평면의 고정된 기능으로 제한된다. 네트워크 스위치에 새로운 서비스를 적용하려면 스위치의 데이터 평면을 프로그래밍할 수 있어야만 한다.

[0008] 데이터 평면의 프로그래밍 가능성을 제공하기 위해 다수의 개발자들은 데이터 평면의 프로그래머블 아키텍처를 정의하는 P4 언어를 제안하였다. P4 언어의 구조는 복수의 패킷 프로세싱 테이블로 구성된다. P4 언어를 사용하면, 네트워크 서비스 제공업체는 테이블의 기능과 테이블의 실행 순서를 프로그래밍한다.

[0009] 네트워크 스위치는 순차적으로 데이터 평면의 테이블을 실행하지만, 테이블을 병렬로 실행할 수 없다. 도 1의 (a)는 일반적인 P4 언어의 제어 흐름을 나타낸다. 일반적인 제어 흐름에서 가장 긴 경로는 테이블과 if-else 문으로 구성된 6 단계를 필요로 한다.

[0010] 도 1의 (b)에 도시된 바와 같이, 테이블 B와 테이블 C가 데이터 독립적인 경우, 스위치는 테이블 C를 True 경로로 복사하여 테이블 B와 테이블 C가 오버랩되도록 하고, 테이블 B와 테이블 C를 병렬로 실행한다. 이에 따라, 병렬화된 경로는 5 단계가 된다. 즉, 테이블을 병렬화하기 위해서는 테이블의 의존성에 대한 정보가 필요하다.

[0011] 그러나, 일반적인 P4 언어의 컴파일러는 데이터 흐름 정보를 가지고 있지 않기 때문에 병렬화를 수행할 수 없다. 종래 연구에서는 테이블 간의 의존성을 고려한 프로그래머블 데이터 평면 아키텍처를 제안하였으나, 이는 단순히 테이블을 순차적으로 실행하거나 패킷 프로세서에서 제어 순서만을 고려한 테이블의 실행을 스케줄링할 뿐이며, 테이블의 데이터 의존성을 검사하거나 테이블을 병렬화하지 않는다.

[0012] 또한, 일반적인 P4 언어의 컴파일러는 데이터 흐름을 알 수 없을 경우 최적화를 적용 할 수 없다. 컴파일러의 최적화의 일 예는 상수 전파이다. 상수 전파는 상수값이 객체 명령어에 도달하는지 여부를 결정하기 위하여 데이터 흐름 정보를 필요로 한다.

[0013] 변수의 정의가 변수를 사용하는 명령문에 도달하고, 변수의 다른 정의가 명령문에 도달하지 않으면 컴파일러는 명령문의 값을 상수값으로 대체할 수 있다. 상수 전파와 마찬가지로 많은 컴파일러 최적화에는 데이터 흐름에 대한 정보가 필요하다. 데이터 흐름 정보가 없는 경우 컴파일러는 P4 데이터 평면 프로그램에 대한 최적화를 적용할 수 없다.

발명의 내용

해결하려는 과제

[0014] 본 발명은 네트워크 스위치의 테이블에 대한 사용-정의(use-def) 분석하고, 사용-정의에 대한 정보를 이용하여 테이블 간 데이터 의존 관계를 분석하여 룩업 또는 테이블 병렬화를 수행하는 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법 및 이를 이용한 병렬화 장치를 제공하는 데 주된 목적이 있다.

과제의 해결 수단

[0015] 본 발명의 일 측면에 의하면, 상기 목적을 달성하기 위한 네트워크 스위치 병렬화를 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법은 테이블의 키와 액션 연산에서 어떤 변수가 읽혀지고(use) 어떤 변수가 쓰여지는지(def)를 찾아 상기 테이블의 사용-정의(use-def)를 분석하는 사용-정의 분석 단계; 상기 사용-정의에 대한 정보를 이용하여 상기 테이블 간 데이터 의존 관계를 분석하여 상기 테이블 간의 데이터 의존성 정보를 생성하는 데이터 의존성 분석 단계; 및 상기 데이터 의존성 정보를 기초로 키-독립(key-independent) 및 테이블-독립(table-independent) 중 적어도 하나의 독립 테이블을 찾고, 상기 독립 테이블을 룩업 및 테이블 중 하나를 병렬화하는 병렬화 처리단계를 포함할 수 있다.

[0016] 본 발명의 일 측면에 의하면, 상기 데이터 의존성 분석 단계는, 현재 테이블의 액션과 후속 테이블의 키 사이의 데이터 의존성(테이블-의존성)과, 상기 현재 테이블의 액션과 후속 테이블의 액션 사이의 데이터 의존성(액션-의존성)을 분석하는 것이 가능하다.

[0017] 본 발명의 일 측면에 의하면, 상기 데이터 의존성 분석 단계는, 이전 연산이 변수에 값을 기록한(def) 후 임의의 연산이 그 변수의 값을 읽으면(use), 두 연산 사이에 흐름 의존성(flow dependence)이 존재하는 것으로 판단하는 것이 가능하다.

[0018] 본 발명의 일 측면에 의하면, 상기 데이터 의존성 분석 단계는, 이전 연산이 변수에서 읽은(use) 후 임의의 연산이 그 변수에 기록하는(def) 경우 두 연산 간에 반 의존성(anti dependence)이 존재하는 것으로 판단하는 것이 가능하다.

[0019] 본 발명의 일 측면에 의하면, 상기 데이터 의존성 분석 단계는, 이전 연산이 변수에 기록한(def) 후 임의의 연산이 그 변수에 다시 기록하는(def) 경우, 두 연산 사이에 출력 의존성(output dependence)이 존재하는 것으로 판단한다.

[0020] 본 발명의 일 측면에 의하면, 상기 데이터 의존성 분석 단계에서는, 상기 테이블의 키와 액션 간의 참 의존성 및 거짓 의존성을 고려하고, 상기 테이블 간의 테이블 의존성 및 액션 의존성을 정의하여 상기 데이터 의존성 분석을 수행하는 것이 가능하다.

[0021] 본 발명의 일 측면에 의하면, 상기 데이터 의존성 분석 단계에서는, 이전 테이블의 조치에 패킷 헤더 또는 메타데이터가 정의(def)로 포함되어 있고, 후속 테이블의 키에 변수가 사용(use)으로 포함되어 있으면 두 테이블에 테이블 의존성이 있는 것으로 판단하는 것이 가능하다.

[0022] 본 발명의 일 측면에 의하면, 상기 데이터 의존성 분석 단계에서는, 이전 테이블의 액션에 패킷 헤더 또는 메타데이터가 사용(use) 또는 정의(def)로 포함되어 있고, 후속 테이블의 동작에 사용(use) 또는 정의(def)의 변수가 들어 있으면 두 테이블이 액션 의존성을 가지는 것으로 판단하는 것이 가능하다.

[0023] 본 발명의 일 측면에 의하면, 상기 병렬화 처리단계는, 인접한 테이블 쌍이 테이블 간에 의존성이 없다면, 상기 테이블 쌍은 독립적인 상태이므로, 키와 액션을 모두 병렬화할 수 있는 것으로 판단하는 것이 가능하다.

[0024] 본 발명의 일 측면에 의하면, 상기 병렬화 처리단계는, 키가 독립적인 테이블 쌍의 경우 즉 인접한 테이블 쌍이 액션 의존성만 가지고 있다면, 룩업 병렬화는 두 액션 간의 데이터 의존성으로 인해 액션을 순차적으로 유지하며, 상기 키 쌍에 대해서만 병렬화할 수 있는 것으로 판단하는 것이 가능하다.

[0025] 본 발명의 다른 측면에 의하면, 상기 목적을 달성하기 위한 데이터 의존성 기반의 데이터 평면 정적 분석 방법을 이용한 네트워크 스위치 병렬화 장치는 테이블의 키와 액션 연산에서 어떤 변수가 읽혀지고(use) 어떤 변수가 쓰여 지는지(def)를 찾아 상기 테이블의 사용-정의(use-def)를 분석하는 사용-정의 분석부; 상기 사용-정의에 대한 정보를 이용하여 현재 테이블의 액션과 후속 테이블의 키 사이의 데이터 의존성(테이블-의존성)과, 상기 현재 테이블의 액션과 상기 후속 테이블의 액션 사이의 데이터 의존성(액션-의존성)을 분석하는 데이터 의존성 분석부; 및 상기 데이터 의존성 정보를 기초로 키-독립(key-independent) 및 테이블-독립(table-independent) 중 적어도 하나의 독립 테이블을 찾고, 상기 독립 테이블을 룩업 및 테이블 중 하나를 병렬화하는 병렬화 처리부를 포함할 수 있다.

[0026] 본 발명의 다른 측면에 의하면, 상기 데이터 의존성 분석부는, 이전 연산이 변수에 값을 기록한(def) 후 임의의 연산이 그 변수의 값을 읽으면(use), 두 연산 사이에 흐름 의존성(flow dependence)이 존재하는 것으로 판단한다.

[0027] 본 발명의 다른 측면에 의하면, 상기 데이터 의존성 분석부는, 이전 연산이 변수에서 읽은(use) 후 임의의 연산이 그 변수에 기록하는(def) 경우 두 연산 간에 반 의존성(anti dependence)이 존재하는 것으로 판단한다.

[0028] 본 발명의 다른 측면에 의하면, 상기 데이터 의존성 분석부는, 이전 연산이 변수에 기록한(def) 후 임의의 연산이 그 변수에 다시 기록하는(def) 경우, 두 연산 사이에 출력 의존성(output dependence)이 존재하는 것으로 판단한다.

[0029] 본 발명의 다른 측면에 의하면, 상기 병렬화 처리부는, 인접한 테이블 쌍이 테이블 간에 의존성이 없다면, 상기 테이블 쌍은 독립적인 상태이므로, 키와 액션을 모두 병렬화할 수 있는 것으로 판단한다.

[0030] 본 발명의 다른 측면에 의하면, 상기 병렬화 처리부는, 상기 키가 독립적인 테이블 쌍의 경우 룩업 병렬화는 두 액션 간의 데이터 의존성으로 인해 액션을 순차적으로 유지하며, 상기 키 쌍에 대해서만 병렬화를 할 수 있는 것이 판단하는 것이 가능하다.

발명의 효과

[0031] 이상에서 설명한 바와 같이, 본 발명은 테이블에 대한 사용-정의 분석을 수행함으로써, 키 및 액션의 사용 및 정의를 찾을 수 있고, 데이터 의존성 분석을 선행한 후 데이터 흐름에 따라 컴파일러의 최적화를 수행할 수 있는 효과가 있다.

[0032] 또한, 본 발명은 데이터 평면 프로그램에 대한 데이터 의존성 분석을 수행함으로써, 테이블 간 데이터 의존성을 찾아 병렬화 가능한 쌍을 추출할 수 있는 효과가 있다.

[0033] 또한, 본 발명은 데이터 의존성 분석 및 병렬화를 수행함으로써, 파이프 라인의 스테이지 수를 줄일 수 있고, 네트워크 스위치의 응답 시간을 줄일 수 있는 효과가 있다.

도면의 간단한 설명

[0034] 도 1은 종래의 P4 언어 프로그램에 대한 제어 흐름도를 나타낸 도면이다.

도 2는 본 발명의 실시예에 따른 P4 언어의 구조를 설명하기 위한 도면이다.

도 3은 본 발명의 실시예에 따른 데이터 의존성 기반의 데이터 평면 정적 분석 방법을 이용한 네트워크 스위치 병렬화 장치를 개략적으로 나타낸 블록 구성도이다.

도 4a는 본 발명의 실시예에 따른 데이터 의존성 기반의 데이터 평면 정적 분석 방법을 이용한 네트워크 스위치 병렬화 장치의 프로세서 동작을 개략적으로 나타낸 블록 구성도이다.

도 4b는 본 발명의 실시예에 따른 프로그래머블 데이터 평면 정적 분석 방법을 설명하기 위한 순서도이다.

도 5 및 도 6은 본 발명의 실시예에 따른 데이터 의존성을 분석하는 동작을 설명하기 위한 예시도이다.

도 7은 본 발명의 실시예에 따른 병렬화 동작을 설명하기 위한 예시도이다.

도 8은 본 발명의 실시예에 따른 데이터 의존성 그래프를 나타낸 도면이다.

도 9는 본 발명의 실시예에 따른 제어 흐름 그래프 및 데이터 의존성 그래프를 결합한 도면이다.

도 10은 본 발명의 다른 실시예에 따른 데이터 의존성 기반의 데이터 평면 정적 분석 방법을 이용한 네트워크 스위치 병렬화 장치의 동작을 나타낸 예시도이다.

발명을 실시하기 위한 구체적인 내용

[0035] 이하, 본 발명의 바람직한 실시예를 첨부된 도면들을 참조하여 상세히 설명한다. 본 발명을 설명함에 있어, 관련된 공지 구성 또는 기능에 대한 구체적인 설명이 본 발명의 요지를 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명은 생략한다. 또한, 이하에서 본 발명의 바람직한 실시예를 설명할 것이나, 본 발명의 기술적 사상은 이에 한정하거나 제한되지 않고 당업자에 의해 변형되어 다양하게 실시될 수 있음은 물론이다. 이하에서는 도면들을 참조하여 본 발명에서 제안하는 네트워크 스위치 병렬화를 위한 프로그래머블 데이터 평면 정적 분석 방법 및 그를 위한 장치에 대해 자세하게 설명하기로 한다.

[0036] 고속으로 다양한 네트워크 서비스를 지원하는 네트워크 스위치에서 유연하고 병렬적인 패킷 프로세싱은 중요하다. 프로그래머블 데이터 평면은 유연한 패킷 프로세싱을 허용하지만, 데이터 평면 프로그램에 대한 데이터 의

존성 분석이 없어 패킷 프로세싱의 병렬화가 제한된다.

[0037] 본 발명에서는 프로그래머블 데이터 평면에 대한 정적 분석과 병렬화 방법을 제안한다. 본 발명에서 제안된 분석은 패킷 프로세싱 테이블 간의 데이터 의존성을 분석하고, 데이터 독립 테이블 쌍을 검색한다. 또한, 본 발명에서 제안된 병렬화는 데이터 독립 테이블을 전체 및 부분 방식으로 병렬화한다.

[0038] 본 발명의 실시예에서는 14 개의 샘플 $P4_{16}$ 프로그램에 대한 분석과 병렬 처리를 평가한다. 본 발명의 실시예에 따른 분석은 37 개의 테이블 중에서 16 개의 데이터 독립 테이블 쌍을 검색하였고, 병렬화 시뮬레이션은 파이프라인의 스테이지 개수를 15.4 % 줄일 수 있다. 본 발명은 프로그래밍 가능한 네트워크 스위치의 응답 시간을 줄이고, 병렬화 및 컴파일러 최적화의 기회를 제공하는 데 유용하다.

[0039] 도 2는 본 발명의 실시예에 따른 P4 언어의 구조를 설명하기 위한 도면이다.

[0040] 본 발명의 실시예에서는 P4 데이터 평면 프로그램에 대한 정적 분석을 제안한다. 일반적인 P4 컴파일러는 데이터 의존성 정보의 부족으로 인해 프로그램을 병렬화할 수 없다. 도 2에서는 P4언어 구조와 제한 사항을 설명하고 테이블 독립 쌍 및 키 독립 쌍에 대한 병렬화에 대해 설명하도록 한다.

[0041] 이하, P4 언어의 구조에 대해 설명하도록 한다.

[0042] P4 언어는 데이터 평면의 패킷 프로세싱을 정의하는 DSL(domain-specific language)이다. P4언어의 구조는 패킷 헤더(packet headers)와 메타 데이터(metadata)의 정의, 파서(parser), 테이블 파이프 라인(table pipeline), 디파서(deparser) 등을 포함한다. 패킷 헤더 및 메타 데이터는 P4 프로그램의 전역 변수이고, 파서는 패킷의 프로토콜 분류를 기술하고, 테이블 파이프 라인은 제어 평면이 제공하는 규칙을 사용하여 패킷 프로세싱을 기술하며, 디파서는 패킷의 패키징을 기술한다.

[0043] 패킷 헤더 및 메타 데이터는 패킷 정보를 포함하는 전역 변수이다. 패킷 정보를 기술하기 위해, 프로그래머는 먼저 헤더와 구조체를 사용하여 헤더의 유형과 메타 데이터를 정의하고 유형과 함께 변수를 인스턴스화한다. 헤더와 메타 데이터는 전역 변수이기 때문에 프로그램에서 변수를 읽거나 쓰는 연산은 데이터에 의존적 일 수 있다.

[0044] 파싱은 패킷의 프로토콜을 식별하고 프로토콜을 사용하여 패킷 헤더에서 패킷 헤더를 추출하는 작업이다. P4 프로그램의 파서는 파싱의 동작을 기술한다. 예를 들어, 이더넷 패킷을 파싱하면 패킷 프로토콜을 분류하기 위한 이더넷 유형 필드에 쓰여진 패킷의 유형이 얻어진다. 그런 다음 네트워크 스위치는 이더넷 프로토콜의 비트 마스크 필드로 패킷 헤더를 추출한다. P4 프로그램에서 파싱은 여러 종류의 프로토콜을 수용할 수 있도록 프로그램 가능하다.

[0045] 이는 네트워크 서비스 제공업체가 새로운 프로토콜을 정의하고 P4의 파서 정의로 새로운 프로토콜을 가진 패킷을 처리할 수 있음을 의미한다. 즉, 네트워크 서비스 공급자는 새 프로토콜을 정의하고, 패킷 프로세스는 P4의 파서 정의 기반의 새 프로토콜을 이용하여 수행될 수 있다.

[0046] 테이블 파이프 라인은 패킷 헤더값 또는 메타 데이터를 수정하는 동작을 포함하는 복수의 테이블로 구성된다. 각 테이블은 키 및 액션(action)으로 구성된다. 네트워크 스위치는 테이블의 키로 패킷 헤더 또는 메타 데이터를 검색하고, 패킷 헤더값 또는 메타 데이터를 수정하는 작업을 수행한다.

[0047] 제어 평면의 규칙은 키 값에 따라 실행해야 하는 액션(action)을 테이블의 동작으로 지정한다. 예를 들어 라우팅의 경우 네트워크 스위치는 소스에서 목적지로 패킷을 보낸다. 패킷을 보내려면 네트워크 스위치가 데이터 평면 테이블과 제어 평면 규칙을 사용하여 소스 (ipv4.srcAddr) 및 목적지 (ipv4.dstAddr) 주소를 변경한다.

[0048] 프로그래머는 P4 프로그램의 테이블 파이프 라인에서 테이블의 실행 순서를 정의할 수 있다. P4 언어도 조건문을 지원하기 때문에 프로그래머는 테이블 파이프 라인의 제어 흐름을 프로그래밍할 수 있다. 네트워크 스위치는 제어 흐름 순서로 테이블을 실행한다. 네트워크 스위치는 실행 순서를 변경하거나 실행을 병렬화하지 않는다.

[0049] 도 2는 P4 프로그램의 테이블 파이프 라인, 테이블 및 제어 평면 규칙 등에 대한 샘플을 도시한다. 도 2의 (a)는 파이프 라인의 수신단을 나타내고, 도 2의 (b)는 테이블 ecmp 그룹을 나타내며, 도 2의 (c)는 제어 평면의 규칙을 나타낸다.

[0050] 도 2의 (a)를 참조하면, 파이프 라인의 수신단은 제어 흐름 순서가 있는 테이블(210, 212, 214, 216, 218)이 포함되어 있다. 도 2의 (b)를 참조하면, 테이블 ecmp 그룹(220)은 키와 액션으로 구성되어 있다. 도 2의 (c)를 참조하면, 테이블 파이프 라인과 테이블은 데이터 평면 프로그램에 설명되어 있으며 규칙은 제어 평면에서 가져온

것이다.

[0051] 테이블 파이프 라인에는 제어 흐름이있는 복수의 테이블이 있다. 도 2의 (a)에서, 네트워크 스위치는 flowlet(210)을 시작으로 순차적으로 테이블을 실행한다. 테이블 파이프 라인에는 if-else 문 또는 switch 문이 포함될 수 있으며, 조건 평가에 따라 실행 경로가 다를 수 있다. 예를 들어, 테이블 파이프 라인은 변수 meta.ingress_metadata.flow_ipg가 50000보다 크면 스위치는 flowlet(210)과 ecmp 그룹(214) 사이에 추가로 신규 flowlet(212)을 실행한다.

[0052] 테이블은 키와 액션으로 구성된다. 여기서, 키는 제어 평면 규칙의 키 값과 비교되는 변수 (패킷 헤더 및 메타데이터)이다. 도 2의 (b)에서, 네트워크 스위치는 패킷의 키 hdr.ipv4.dstAddr과 제어 평면의 규칙 키 값(2320, 232)을 비교한다. 따라서, 키는 변수(사용(use))에서만 읽는다.

[0053] 반면, 액션은 패킷 헤더 또는 메타 데이터를 쓰거나, 패킷 헤더 또는 메타 데이터에서 읽는 기능을 갖는다. 도 2의 (b)에서 액션 set_ecmp_select는 변수 meta.ingress_metadata.ecmp_offset에 쓰는 해시 함수를 호출한다. 해시 함수를 호출하기 위해, set_ecmp_select는 소정의 변수를 인수로 전달한다(미도시). 따라서 액션은 패킷 헤더 또는 메타 데이터를 읽거나(사용(use)) 또는 쓰기(정의(def)) 할 수 있다.

[0054] 제어 평면의 규칙은 네트워크 스위치의 런타임 동작을 나타낸다. 네트워크 스위치는 테이블 액션이 규칙에 따라 실행되어야 하는지를 결정한다. 도 2의 (c)에서 패킷의 hdr.ipv4.dstAddr이 10.0.1.1과 같으면 네트워크 스위치는 ecmp 그룹의 set_ecmp_select를 패킷에 적용(230)한다. 한편, 패킷의 hdr.ipv4.dstAddr이 10.0.1.2이면 스위치는 패킷을 삭제(232)한다. 그러나, 컴파일러가 정적 분석을 적용할 때 이러한 규칙은 알 수 없으므로 데이터 의존성 분석은 모든 작업이 런타임에서 실행될 것이다.

[0055] 도 3은 본 발명의 실시예에 따른 네트워크 스위치 병렬화 장치를 개략적으로 나타낸 블록 구성도이다.

[0056] 본 실시예에 따른 네트워크 스위치 병렬화 장치(300)는 통신부(310), 메모리(320) 및 프로세서(330)를 포함한다.

[0057] 통신부(310)는 메모리(320) 및 프로세서(330)와 연결되어 데이터를 송수신한다. 또한, 통신부(310)는 외부의 다른 장치와 연결되어 데이터를 송수신할 수 있다. 예를 들어, 통신부(310)는 인터페이스(Interface)일 수 있으며, 내부 버스(Internal bus) 및 외부 버스(External bus), 유무선 통신모듈 등으로 구현될 수도 있다.

[0058] 통신부(310)는 외부의 장치로부터 데이터를 수신하여, 메모리(320) 및 프로세서(330)에 데이터를 전송할 수 있다.

[0059] 메모리(320)는 통신부(310)가 수신한 데이터 및 프로세서(330)가 처리한 데이터를 저장하는 동작을 수행한다. 예를 들어, 메모리(320)는 프로그램을 저장할 수 있다. 예를 들어, 메모리(320)는 하나 이상의 휘발성 메모리, 비휘발성 메모리 및 RAM(Random Access Memory), 플래시 메모리, 하드 디스크 드라이브 및 광학 디스크 드라이브를 포함할 수 있다.

[0060] 메모리(320)는 네트워크 스위치 병렬화 동작과 관련된 명령어 세트 예를 들어, 기능 소프트웨어 등을 저장할 수 있다. 네트워크 스위치 병렬화 동작과 관련된 명령어 세트는 프로세서(330)에 의해 실행될 수 있다. 프로세서(330)는 명령어 세트에 따라 네트워크 스위치 병렬화 동작을 수행할 수 있다.

[0061] 프로세서(330)는 통신부(310)가 수신한 데이터 및 메모리(320)에 저장된 데이터를 처리한다.

[0062] 프로세서(330)는 목적하는 동작들(Desired Operations)을 실행시키기 위한 물리적인 구조를 갖는 회로를 가지는 하드웨어로 구현된 데이터 처리 장치일 수 있다. 예를 들어, 목적하는 동작들은 프로그램에 포함된 코드(Code) 또는 인스트럭션들(Instructions)을 포함할 수 있다. 예를 들어, 하드웨어로 구현된 데이터 처리 장치는 마이크로프로세서(Microprocessor), 중앙 처리 장치(Central Processing Unit), 프로세서 코어(Processor Core), 멀티-코어 프로세서(Multi-Core Processor), 멀티프로세서(Multiprocessor), ASIC(Application-Specific Integrated Circuit), FPGA(Field Programmable Gate Array)를 포함할 수 있다.

[0063] 프로세서(330)는 메모리(예를 들어, 메모리(320))에 저장된 컴퓨터로 읽을 수 있는 코드(예를 들어, 소프트웨어) 및 프로세서(330)에 의해 유발된 인스트럭션들을 실행한다.

[0064] 본 실시예에 따른 네트워크 스위치 병렬화 장치(300)는 테이블의 키와 액션 연산에서 어떤 변수가 읽혀지고(use) 어떤 변수가 쓰여 지는지(def)를 찾아 상기 테이블의 사용-정의(use-def)를 분석하는 사용-정의 분석부, 상기 사용-정의에 대한 정보를 이용하여 현재 테이블의 액션과 후속 테이블의 키 사이의 데이터 의존성(테이블-

의존성)과, 상기 현재 테이블의 액션과 상기 후속 테이블의 액션 사이의 데이터 의존성(액션-의존성)을 분석하는 데이터 의존성 분석부 및 상기 데이터 의존성 정보를 기초로 키-독립(key-independent) 및 테이블-독립(table-independent) 중 적어도 하나의 독립 테이블을 찾고, 상기 독립 테이블을 룩업 및 테이블 중 하나를 병렬화하는 병렬화 처리부를 포함할 수 있다.

[0065] 네트워크 스위치 병렬화 장치(300)의 데이터 의존성 분석부는, 현재 테이블의 액션과 후속 테이블의 키 사이의 데이터 의존성(테이블-의존성)과, 상기 현재 테이블의 액션과 후속 테이블의 액션 사이의 데이터 의존성(액션-의존성)을 분석한다.

[0066] 네트워크 스위치 병렬화 장치(300)의 데이터 의존성 분석부는, 이전 연산이 변수에 값을 기록한(def) 후 임의의 연산이 그 변수의 값을 읽으면(use), 두 연산 사이에 흐름 의존성(flow dependence)이 존재하는 것으로 판단한다.

[0067] 네트워크 스위치 병렬화 장치(300)의 데이터 의존성 분석부는, 이전 연산이 변수에서 읽은(use) 후 임의의 연산이 그 변수에 기록하는(def) 경우 두 연산 간에 반 의존성(anti dependence)이 존재하는 것으로 판단한다.

[0068] 네트워크 스위치 병렬화 장치(300)의 데이터 의존성 분석부는, 이전 연산이 변수에 기록한(def) 후 임의의 연산이 그 변수에 다시 기록하는(def) 경우, 두 연산 사이에 출력 의존성(output dependence)이 존재하는 것으로 판단한다.

[0069] 네트워크 스위치 병렬화 장치(300)의 데이터 의존성 분석부는, 상기 테이블의 키와 액션 간의 참 의존성 및 거짓 의존성을 고려하고, 상기 테이블 간의 테이블 의존성 및 액션 의존성을 정의하여 상기 데이터 의존성 분석을 수행한다.

[0070] 네트워크 스위치 병렬화 장치(300)의 데이터 의존성 분석부는, 이전 테이블의 조치에 패킷 헤더 또는 메타 데이터가 정의(def)로 포함되어 있고, 후속 테이블의 키에 변수가 사용(use)으로 포함되어 있으면 두 테이블에 테이블 의존성이 있는 것으로 판단한다.

[0071] 네트워크 스위치 병렬화 장치(300)의 데이터 의존성 분석부는, 이전 테이블의 액션에 패킷 헤더 또는 메타 데이터가 사용(use) 또는 정의(def)로 포함되어 있고, 후속 테이블의 동작에 사용(use) 또는 정의(def)의 변수가 들어 있으면 두 테이블이 액션 의존성을 가지는 것으로 판단한다.

[0072] 네트워크 스위치 병렬화 장치(300)의 병렬화 처리부는, 인접한 테이블 쌍이 테이블 간에 의존성이 없다면, 상기 테이블 쌍은 독립적인 상태이므로, 키와 액션을 모두 병렬화할 수 있는 것으로 판단한다.

[0073] 네트워크 스위치 병렬화 장치(300)의 병렬화 처리부는, 키가 독립적인 테이블 쌍의 경우 두 액션 간의 데이터 의존성으로 인해 액션을 순차적으로 유지하며, 상기 키 쌍에 대해서만 병렬화하는 룩업 병렬화를 수행할 수 있는 것으로 판단한다.

[0074] 도 4a는 본 발명의 실시예에 따른 네트워크 스위치 병렬화 장치의 프로세서 동작을 개략적으로 나타낸 블록 구성도이다.

[0075] 네트워크 스위치 병렬화 장치(300)의 프로세서(330)는 정적 분석부(410) 및 병렬화 처리부(440)를 포함한다. 여기서, 정적 분석부(410)는 사용-정의 분석부(420) 및 데이터 의존성 분석부(430)를 포함한다. 도 4a의 프로세서(330)에 포함된 구성요소는 일 실시예에 따른 것으로서, 도 4a에 도시된 모든 블록이 필수 구성요소는 아니며, 다른 실시예에서 프로세서(330)에 포함된 일부 블록이 추가, 변경 또는 삭제될 수 있다.

[0076] 도 4a에서는 프로세서(330)에서 네트워크 스위치에 대한 병렬화를 수행하는 것으로 기재하고 있으나 반드시 이에 한정되는 것은 아니며, 네트워크 스위치에서 네트워크 스위치에 대한 병렬화를 수행하는 동작 예컨대, 정적 분석, 사용-정의 분석, 데이터 의존성 분석, 병렬화 처리 등을 수행할 수 있다.

[0077] 네트워크 스위치는 테이블 파이프 라인의 제어 순서에 따라 테이블을 순차적으로 실행한다. 하지만 네트워크 스위치는 테이블에 데이터 의존성이 없는 경우, 테이블을 병렬로 실행할 수 있다(도 1(b)). 컴파일러는 테이블을 병렬화하기 위해 테이블 간에 데이터 의존성을 조사해야 하지만 P4 컴파일러는 데이터 의존성을 분석하지 않는다. 따라서, 본 발명의 실시예에서는 테이블 간의 데이터 의존성을 조사하는 데이터 의존성 분석을 개시한다. 여기서, 데이터 의존성 분석은 테이블의 액션과 후속 테이블의 키(테이블-의존성) 사이의 데이터 의존성과, 테이블의 액션과 후속 테이블의 액션(액션-의존성) 사이의 데이터 의존성을 분석한다.

[0078] 테이블 간의 데이터 의존성을 확인하기 위해, 본 발명의 실시예에서는 키와 액션의 연산에서 어떤 변수의 사용

(use)과 정의(def)를 찾는 사용-정의(use-def) 분석 과정을 개시한다. 테이블에 대한 사용-정의(use-def) 분석은 데이터 의존성을 분석할 뿐만 아니라 데이터 흐름 정보가 필요한 컴파일러 최적화를 수행할 수 있다.

[0079] 본 발명의 실시예에서는 데이터 의존성 분석에 기초하여 데이터 독립 테이블을 병렬화한다. 테이블은 키 간에 독립적일 수 있지만 액션 간에 의존적 (키-독립)이거나 모든 키와 액션 (테이블-독립) 간에 독립적일 수 있다. 데이터 의존성 분석은 이러한 종류의 데이터 독립 쌍을 찾으며, 병렬화는 룩업 병렬화 및 테이블 병렬화와 같이 두 가지 방식으로 병렬화한다. 룩업 병렬화는 키 독립 테이블의 키 연산을 병렬 처리한다. 한편, 테이블 병렬화는 테이블 독립 테이블의 전체 연산을 완전히 병렬 처리한다.

[0080] 본 발명의 실시예에 따른 정적 분석 및 병렬화에 대해 설명하도록 한다.

[0081] 본 발명의 실시예에 따른 정적 분석은 사용-정의(use-def) 분석과 데이터 의존성 분석으로 구성된다. 사용-정의(use-def) 분석은 테이블의 사용 및 정의를 찾고 데이터 의존성 분석은 테이블 간의 데이터 의존 관계를 찾는다.

[0082] 본 발명의 실시예에 따른 병렬화는 데이터 의존성 정보를 이용하여 데이터 독립 테이블 쌍을 병렬 처리한다. 병렬화는 테이블 병렬화 및 룩업 병렬화로 구성된다. 테이블 병렬화는 데이터 의존성이 없는 테이블을 완전히 병렬 처리하고, 룩업 병렬화는 테이블 액션 간에만 데이터 의존성이 있는 테이블의 키 일치 연산을 병렬 처리한다.

[0083] 이하, 정적 분석부(410)의 동작에 대해 설명하도록 한다.

[0084] 본 발명은 데이터 평면 프로그램에서 테이블 파이프 라인의 테이블 간에 데이터 의존성을 찾는 정적 분석을 제안한다. 정적 분석은 사용-정의(use-def) 분석과 데이터 의존성 분석으로 구성된다. 본 발명에서는 테이블의 키와 액션을 사용하고 정의하기 위해 사용-정의(use-def) 분석을 사용한다. 데이터 의존성 분석은 사용-정의(use-def) 정보를 기반으로 키와 액션 간에 데이터 의존 관계를 찾는다. 데이터 의존성 분석에서는 사용(use) 및 정의(def) 관계에 따라 세 가지 유형의 데이터 의존성을 고려한다.

[0085] 데이터 의존성 분석은 흐름 의존성(flow dependence) 또는 쓰기 후 읽기(RAW) 의존성을 고려한다. 구체적으로, 이전 연산이 변수에 값을 기록한(def) 후 임의의 연산이 그 변수의 값을 읽으면(use), 두 연산 사이에 흐름 의존성이 존재하는 것으로 판단한다.

[0086] 또한, 데이터 의존성 분석은 반 의존성(anti dependence) 또는 읽기 후 쓰기 (WAR) 의존성을 고려한다. 구체적으로, 이전 연산이 변수에서 읽은(use) 후 작업이 변수에 기록하는(def) 경우 두 연산 간에 반 의존성(anti dependence)이 존재하는 것으로 판단한다.

[0087] 또한, 데이터 의존성 분석은 출력 의존성(output dependence) 또는 쓰기 후 쓰기 (WAW) 의존성을 고려한다. 구체적으로, 이전 연산이 변수에 기록한(def) 후 임의의 연산이 그 변수에 다시 기록하는(def) 경우, 두 연산 사이에 출력 의존성이 존재하는 것으로 판단한다.

[0088] 여기서, 흐름 의존성은 참 의존성이다. 왜냐하면 연산은 이전 연산의 결과에 의존하기 때문이다. 따라서 변수(def)에 쓰는 연산은 후속 연산이 변수 (use)에서 읽히기 전에 실행되어야 한다. 그러나, 반 의존성과 출력 의존성은 거짓 의존성이다. 두 개의 거짓 의존성 연산이 두 연산의 결과를 재정렬함으로써 병렬적으로 또는 순서 없이 실행될 수 있기 때문에 거짓 의존성이 존재할 수 있다.

[0089] 본 발명의 정적 분석은 참 의존성과 거짓 의존성을 모두 판단한다. 다만, 본 발명의 병렬화는 참 의존성과 거짓 의존성을 같은 데이터 의존성으로 판단하여 데이터 의존성이 없는 테이블에 대해서만 보수적으로 병렬화를 진행한다. 그러므로 본 발명의 병렬화는 거짓 의존성에 대한 추가적인 병렬화 기회를 탐색하지 않는다.

[0090] 이하, 사용-정의 분석부(420)의 동작에 대해 설명하도록 한다.

[0091] 테이블 간의 데이터 의존성을 확인하려면 데이터 의존성 분석에 테이블의 작업에 대한 사용-정의(use-def) 정보가 필요하다. 따라서 본 발명에서는 각 테이블의 키와 액션에 대한 사용-정의(use-def) 분석을 사용한다. 사용-정의(use-def) 분석은 어떤 변수가 입력인지 (사용), 어떤 변수가 각 연산의 출력 (def)인지를 보여준다.

[0092] 사용-정의(use-def) 분석은 테이블의 키와 액션에 대해 사용 및 정의를 계산한다. 사용은 테이블 외부에서 정의되고 테이블 내부에서 사용되는 패킷 헤더 또는 메타 데이터를 포함한다. 정의는 테이블 내부에서 정의 또는 재정의된 패킷 헤더 또는 메타 데이터를 포함한다.

[0093] 사용-정의 분석은 먼저 키에 대한 사용을 계산한다. 사용은 키가 읽는 패킷 헤더 또는 메타데이터를 포함한다. 도 2의 (b)에서, 키의 사용은 hdr.ipv4.dstAddr을 포함하며, ecmp 그룹은 하나의 키 변수를 사용하지만 테이블에서는 여러 개의 키가 있을 수 있다.

표 1

Algorithm 1 Use-def analysis on actions

for instruction ∈ action do
use ← use ∪ (In (instruction) ₩ def)
def ← def ∪ Out (instruction)
end for
In (e) : right variables of an assignment statement e , or in variables of an ex- ternal function call e
Out (e) : a left variable of an assignment statement e , or out variables of an external function call e

[0094]

[0095] 다음으로 사용-정의분석은 액션에 대한 사용과 정의를 분석한다. 액션은 복수의 명령어가 포함된 함수이다. 액션의 사용-정의를 검사하기 위한 사용-정의 분석은 위에서부터 아래까지의 모든 명령에 대해 사용-정의를 계산한다. [표 1]에서는 알고리즘 1의 동작에 대한 사용-정의 분석을 나타낸다.

[0096] 사용-정의 분석은 먼저 명령의 입력 변수를 포함하는 사용(use)을 계산하지만, 이전 정의(def)에 대한 변수는 제외한다. 일단 액션의 명령이 변수에 쓰게 되면 변수에서 읽은 명령어가 액션 바깥에서 변수의 정의를 읽지 않고 이전 명령어의 결과를 읽게 되기 때문이다. 사용은 외부에서 정의(def)에 대한 입력 변수를 참조하므로 분석은 사용(use)에서 내부적으로 변경되는 변수를 제외한다.

[0097] 사용(use)을 계산 한 후, 사용-정의 분석은 명령의 출력 변수를 정의(def)에 넣는다. 실제로 명령문은 여러 번 변수에 쓸 수 있지만, 변수의 마지막 정의만 외부에 도달한다. 그러나 사용-정의(use-def) 분석은 테이블 병렬화가 데이터 의존성 정보만 필요로 하기 때문에 정의(def)의 실제 값을 고려하지 않는다. 컴파일러가 컴파일러 최적화를 적용하려는 경우 본 발명은 사용-정의(use-def) 분석을 기반으로 도달-정의 분석을 개발한다.

[0098] 명령어의 사용 및 정의를 계산하기 위해 함수 In 및 함수 Out은 대입문 및 외부 함수 호출에서 입력 및 출력 변수를 식별한다. 대입문은 오른쪽의 변수에서 읽고 왼쪽의 변수에 쓴다. 외부 함수 호출의 경우, 사용-정의(use-def) 분석은 함수의 정의를 검사하고, in 인수와 out 인수를 찾는다. 도 2의 (b)에서 해시 함수는 첫 번째 인수를 출력 (out)으로 사용하고 다른 인수는 입력 (in)으로 사용한다.

[0099] 이하, 데이터 의존성 분석부(430)의 동작에 대해 설명하도록 한다.

[0100] 본 발명에서는 키와 액션에 대한 사용-정의(use-def) 정보를 이용하여 테이블 간의 데이터 의존성을 찾는 데이터 의존성 분석을 수행한다. 키 매칭 또는 액션이 패킷 헤더 또는 메타 데이터를 읽거나 쓰므로 테이블은 두 테이블에서 동일한 변수를 사용하여 데이터 의존성을 가질 수 있다. 데이터 의존성 분석은 키와 액션 간의 참 의존성 및 거짓 의존성을 고려하고, 테이블 간의 두 가지 데이터 의존성을 정의한다. 여기서, 두 가지 데이터 의존성은 테이블 의존성 및 액션 의존성이다. 도 5는 샘플 P4 프로그램의 테이블 중 사용-정의(use-def)를 설명하고 테이블 간의 두 가지 데이터 의존성을 나타낸다.

[0101] 테이블 의존성은 테이블의 액션과 후속 테이블의 키 간에 발생한다. 이전 테이블의 조치에 패킷 헤더 또는 메타데이터가 정의(def)로 있고 후속 테이블의 키에 변수가 사용(use)으로 포함되어 있으면 두 테이블에 테이블 의존성이 있는 것으로 판단한다. 이전 테이블의 액션에 대한 결과는 후속 테이블에서 키 매칭으로 사용된다.

[0102] 테이블 ecmp_group은 정의(def)에 대한 ecmp_offset을 포함하는 액션이 존재하며, 테이블 ecmp_nhop의 키에는 사용(use)에 대한 ecmp_offset이 포함되어 있다(도 5의 (a)). ecmp_offset은 프로그램의 제어 흐름으로 인해 ecmp_nhop보다 먼저 실행되기 때문에 ecmp_group의 ecmp_offset 정의는 ecmp_nhop의 키 매칭에 도달한다. 따라서 ecmp_group과 ecmp_nhop은 테이블 간에 테이블 의존성이 존재한다.

[0103] 반면, 이전 테이블의 액션에 패킷 헤더 또는 메타 데이터가 사용(use) 또는 정의(def)로 포함되어 있고 후속 테이블의 동작에 사용(use) 또는 정의(def)의 변수가 들어 있으면 두 테이블이 액션 의존성을 가지는 것으로 판단할 수 있다. 여기서, 흐름 의존성(def-use), 반 의존성(use-def) 및 출력 의존성(def-def)은 데이터 의존성으로 간주될 수 있지만, 동일한 변수를 사용(use)하는 두 작업(use-use) 간에는 의존 관계가 없다. 그러므로 데이터 의존성 분석은 액션 사이의 흐름 의존성, 비 의존성 및 출력 의존성을 인식함으로써 액션 의존성을 발견할 수 있다.

[0104] 테이블 flowlet은 정의(def)에 대한 flowlet_id를 가진 액션을 포함하고, 테이블 new_flowlet은 사용(use)에 대한 flowlet_id를 가진 액션을 포함한다(도 5의 (b)). 정의(def)가 있는 액션이 사용(use)에 대한 액션보다 우선하기 때문에 두 액션 간에는 흐름 의존성이 존재한다. 데이터 의존성 분석은 flowlet과 new_flowlet에 대한 액션 의존성을 확인한다.

[0105] 데이터 의존성 분석은 테이블 간의 테이블 의존성 및 액션 의존성을 찾고, 제어 흐름 순서로 테이블을 트래버싱(traversing)한다. P4 프로그램의 제어 흐름은 테이블의 실행 순서를 설명하므로 제어 흐름을 기반으로 데이터 의존성을 찾는 것이 자연스럽다. 제어 흐름이 분기되는 경우 분석은 가능한 모든 경로를 개별적으로 검사하고 싱크 테이블에서 의존성 정보를 병합한다. 도 6에서 데이터 의존성 정보는 if-else 문에서 분기되며, 분석은 ecmp_group에서 정보를 병합한다.

[0106] 데이터 의존성 분석은 조건부 분기에 대한 사용-정의도 분석한다. 조건부 분기는 변수가 조건을 충족시키는지 여부를 확인하므로 분기를 사용하면 변수가 포함된다. 변수가 패킷 헤더 또는 메타 데이터인 경우 분기는 데이터 의존성 연산의 후보가 된다. if-else 문은 flow_ipg(use)에서 읽는다. flowlet에 있는 액션 중 하나가 정의(def)로 flow_ipg를 가지고 있다면, 테이블과 if-else 문은 흐름 의존성을 갖는다.

[0107] 이하, 병렬화 처리부(440)의 동작에 대해 설명하도록 한다.

[0108] 본 발명의 실시예에서 병렬화 처리부(440)은 데이터 의존성 정보를 사용하여 병렬화를 수행한다. 본 발명의 실시예에서는 데이터 의존성 분석을 통하여 두 개의 인접한 테이블이 키 독립적인지 또는 테이블 독립적 인지를 검사하고, 각 쌍을 병렬화하는 방법을 개시한다. 병렬화의 개념은 종래의 연구에서 제안되었지만 종래의 연구의 패킷 프로세서는 데이터 의존성 분석이 없기 때문에 병렬로 테이블을 실행하지 않는다. 반면, 본 발명의 실시예에서는 데이터 의존성에 기반한 병렬화가 개시된다. 본 발명의 실시예에서는 병렬화가 프로그래밍 가능한 네트워크 스위치의 단계 및 지연 수를 줄일 수 있는 방법인 것을 보여준다.

[0109] 데이터 의존성 정보에 기반하여, 데이터 의존성 분석은 제어 흐름에 있는 두 개의 인접 테이블 쌍을 분류한다(도 7). 두 인접한 테이블 쌍이 테이블 간에 테이블 의존성을 가지면 그 쌍은 데이터 의존적이며 병렬화가 불가능하다(도 7의 (a)). 한 쌍이 테이블 간에 동작 의존성만 갖는 경우 이 쌍은 키와 무관하다 (도 7의 (b)). 한 쌍이 테이블 사이에 의존성이 없다면, 그 쌍은 테이블 독립적인 상태이다(도 7의 (c)). 이에 따라, 키와 액션을 모두 병렬화 할 수 있다.

[0110] 본 발명의 실시예에서는 독립 쌍에 대한 룩업 및 테이블 병렬화가 개시된다. 키 독립적 쌍의 경우 룩업 병렬화는 키 매칭을 병렬화하지만 두 액션 간의 데이터 의존성으로 인해 액션을 순차적으로 유지한다. 본 발명의 룩업 병렬화는 키를 병렬로 검색하여 테이블 파이프 라인의 키 스테이지 및 액션 스테이지 수를 1로 줄인다(도 7의 (b)). 데이터 의존성이 없는 테이블 쌍의 경우 테이블 병렬화가 키와 액션을 병렬 처리한다. 병렬로 두 개의 테이블을 실행함으로써 본 발명의 테이블 병렬화는 스테이지 수를 2로 줄인다 (도 7의 (c)). 여기서는 두 병렬 처리 방법이 각 테이블에서 수행되는 작업을 결정하는 키 매칭을 병렬화한다. 룩업 병렬화는 액션 의존성으로 인해 대응하는 두 액션을 순차적으로 남긴다. 그러나 테이블 병렬화는 독립적인 액션을 병렬 처리한다.

[0111] 그러나 일부 테이블에 키가 없거나 아무런 액션이 없으면, 병렬 처리로 인해 속도가 떨어진다. 두 테이블이 키 독립적 인 경우에도 키 중 하나가 누락되면 키 일치를 병렬화 할 수 없다.

[0112] 본 발명은 데이터 평면 프로그램의 정적 분석과 병렬화 방법에 대한 것으로서, 정적 분석은 키와 액션의 사용과 정의를 조사하고 테이블 간의 데이터 의존성을 찾는다. 또한, 테이블 및 룩업 병렬화는 각각 테이블 및 액션 독

립 테이블을 병렬 처리한다. 본 발명은 14 개의 샘플 $P4_{16}$ 프로그램에 대한 분석 및 병렬 처리 평가를 수행한다.

평가 결과에 따르면 37 개의 테이블에서 7 개의 테이블 독립 쌍과 9 개의 액션 독립 쌍을 찾을 수 있으며 병렬 처리 시뮬레이션을 통해 테이블 파이프 라인 단계 수가 15.4 % 줄어들게 된다. 본 발명은 프로그래밍 가능한 네트워크 스위치의 응답 시간을 줄이기 위해 데이터 평면 프로그램을 최적화한다.

[0113] 도 4b는 본 발명의 실시예에 따른 프로그래머블 데이터 평면 정적 분석 방법을 설명하기 위한 순서도이다.

[0114] 정적 분석부(410)은 사용-정의 분석 및 데이터 의존성 분석을 수행한다(S410,S420).

[0115] 사용-정의 분석부(420)은 데이터 평면의 테이블에 대한 사용-정의(use-def) 분석을 수행한다(S410).

[0116] 데이터 의존성 분석부(430)은 사용-정의에 대한 정보를 이용하여 테이블 간 데이터 의존 관계를 분석하여 데이터 의존성 정보를 생성한다(S420).

[0117] 병렬화 처리부(440)은 데이터 의존성에 근거하여 병렬화를 수행한다(S430).

[0118] 병렬화 처리부(440)은 데이터 의존성 정보를 기초로 키-독립(key-independent) 및 테이블-독립(table-independent) 중 적어도 하나의 독립 테이블을 찾고, 독립 테이블을 룩업 및 테이블 중 하나를 병렬화한다.

[0119] 도 4b에서는 각 단계를 순차적으로 실행하는 것으로 기재하고 있으나, 반드시 이에 한정되는 것은 아니다. 다시 말해, 도 4b에 기재된 단계를 변경하여 실행하거나 하나 이상의 단계를 병렬적으로 실행하는 것으로 적용 가능할 것이므로, 도 4b는 시계열적인 순서로 한정되는 것은 아니다.

[0120] 도 4b에 기재된 본 실시예에 따른 프로그래머블 데이터 평면 정적 분석 방법은 애플리케이션(또는 프로그램)으로 구현되고 단말장치(또는 컴퓨터)로 읽을 수 있는 기록매체에 기록될 수 있다. 본 실시예에 따른 프로그래머블 데이터 평면 정적 분석 방법을 구현하기 위한 애플리케이션(또는 프로그램)이 기록되고 단말장치(또는 컴퓨터)가 읽을 수 있는 기록매체는 컴퓨팅 시스템에 의하여 읽혀질 수 있는 데이터가 저장되는 모든 종류의 기록장치 또는 매체를 포함한다.

[0121] 도 5 및 도 6은 본 발명의 실시예에 따른 데이터 의존성을 분석하는 동작을 설명하기 위한 예시도이다.

[0122] 도 5의 (a)에서 테이블 ecmp_group(510)은 정의(def)에 대한 ecmp_offset(512)을 포함하는 액션이 존재하며, 테이블 ecmp_nhop(520)의 키에는 사용(use)에 대한 ecmp_offset(512)이 포함되어 있다. ecmp_group은 프로그램의 제어 흐름으로 인해 ecmp_nhop보다 먼저 실행되기 때문에 ecmp_group의 ecmp_offset 정의는 ecmp_nhop의 키 매칭에 도달한다. 따라서 ecmp_group과 ecmp_nhop은 테이블 간에 테이블 의존성이 존재한다.

[0123] 도 5의 (b)에서 테이블 flowlet(530)은 정의(def)에 대한 flowlet_id를 가진 액션(532)을 포함하고, 테이블 new_flowlet(540)은 사용(use)에 대한 flowlet_id를 가진 액션(542)을 포함한다. 정의(def)가 있는 액션이 사용(use)에 대한 액션보다 우선하기 때문에 두 액션 간에는 흐름 의존성이 존재한다. 데이터 의존성 분석은 flowlet과 new_flowlet에 대한 액션 의존성을 찾는다.

[0124] 도 6에서 if-else 문은 flow_ipg(use)에서 읽는다. flowlet에 있는 액션 중 하나가 정의(def)로 flow_ipg를 가지고 있다면, 테이블과 if-else 문(600)은 흐름 의존성을 갖는다.

[0125] 도 7은 본 발명의 실시예에 따른 병렬화 동작을 설명하기 위한 예시도이다.

[0126] 도 7을 참조하면, 데이터 의존성 정보에 기반하여, 분석은 제어 흐름에 있는 두 개의 인접 테이블 쌍을 분류한다. 테이블 쌍이 테이블 간에 테이블 의존성을 가지면 그 쌍은 데이터 의존적이며 병렬화가 불가능하다(도 7의 (a)). 한 쌍이 테이블 간에 동작 의존성만 갖는 경우 이 쌍은 키와 무관하다 (도 7의 (b)). 한 쌍이 테이블 사이에 의존성이 없다면, 그 쌍은 테이블 독립적인 상태이다(도 7의 (c)). 이에 따라, 키 독립적 쌍과 테이블 독립적 쌍을 병렬화 할 수 있다.

[0127] 본 발명의 실시예에서는 독립 쌍에 대한 룩업 및 테이블 병렬화를 제안한다. 키 독립적 쌍의 경우 룩업 병렬화는 키 매칭을 병렬화하지만 두 액션 간의 데이터 의존성으로 인해 액션을 순차적으로 유지한다. 본 발명의 룩업 병렬화는 키를 병렬화하여 테이블 파이프 라인의 키 스테이지 및 액션 스테이지 수를 1로 줄인다(도 7의 (b)). 데이터 의존성이 없는 테이블 쌍의 경우 테이블 병렬화가 키와 액션을 병렬 처리한다. 본 발명의 테이블 병렬화는 병렬로 두 개의 테이블을 실행함으로써 스테이지 수를 2로 줄인다 (도 7의 (c)). 여기서는 두 병렬 처리 방법이 각 테이블에서 수행되는 작업을 결정하는 키 매칭을 병렬화한다. 룩업 병렬화는 액션 의존성으로 인해 대응하는 두 액션을 순차적으로 남긴다. 그러나 테이블 병렬화는 독립적인 액션을 병렬 처리한다.

[0128] 도 8은 본 발명의 실시예에 따른 데이터 의존성 그래프를 나타낸 도면이다.

[0129] 제안된 데이터 의존성 그래프 표현 장치는 Boost Graph Library를 사용하여 데이터 의존 그래프를 그린다. 도 8은 flowlet switching-bmv2.p4의 데이터 의존성 그래프를 보여준다.

[0130] 도 9는 본 발명의 실시예에 따른 제어 흐름 그래프 및 데이터 의존성 그래프를 결합한 도면이다.

[0131] 도 8에서는 제어 흐름을 나타내지 않으므로 도 9와 같이 제어 흐름 그래프와 데이터 종속 그래프를 결합할 수 있다. 이 파이프 라인에는 각 테이블이 인접 테이블과의 데이터 의존성을 가지기 때문에 테이블 병렬화가 가능한 쌍이 없다. 그러나 새로운 테이블 flowlet과 테이블 ecmp 그룹은 키에 독립적이므로 룩업 병렬화를 적용 할 수 있다. 또한 분석은 조건문을 키로 간주하므로 'flow_ipg > 50000'-'new_flowlet' 및 'flow ipg > 50000'-'flowlet'가 키 독립적 쌍으로 포함된다.

[0132] 도 10은 본 발명의 다른 실시예에 따른 네트워크 스위치 병렬화 장치의 동작을 나타낸 예시도이다.

[0133] 도 9에서 신규 flowlet과 ecmp 그룹은 액션 의존성을 갖는다. 하지만, 도 10의 (a)와 같이, 룩업 병렬화는 신규 flowlet에 대한 키가 없기 때문에 단계의 수를 줄일 수 없다. 도 10의 (b)와 같이, 병렬화가 신규 flowlet의 'No Key'로 ipv4.dstAddr 키를 병렬화하더라도, 스테이지 수는 원본과 동일하다.

[0134] 이에 따라 본 발명의 다른 실시예에 따르면, 테이블을 최적화하기 위해 초기 단계에서 패킷을 제거(drop)하는 동작을 제안한다.

[0135] 액션의 제거는 패킷을 삭제하도록 표시하고, 네트워크 스위치는 패킷을 삭제한다. 도 10의 (c)를 참조하면, 테이블은 키 매칭 후 패킷을 버린다. 패킷 제거(drop)는 어떠한 액션에도 의존하지 않지만 키에 의존하기 때문에 컴파일러는 키 매칭 직후에 드롭을 이동할 수 있다.

[0136] 이상의 설명은 본 발명의 실시예의 기술 사상을 예시적으로 설명한 것에 불과한 것으로서, 본 발명의 실시예가 속하는 기술 분야에서 통상의 지식을 가진 자라면 본 발명의 실시예의 본질적인 특성에서 벗어나지 않는 범위에서 다양한 수정 및 변형이 가능할 것이다. 따라서, 본 발명의 실시예들은 본 발명의 실시예의 기술 사상을 한정하기 위한 것이 아니라 설명하기 위한 것이고, 이러한 실시예에 의하여 본 발명의 실시예의 기술 사상의 범위가 한정되는 것은 아니다. 본 발명의 실시예의 보호 범위는 아래의 청구범위에 의하여 해석되어야 하며, 그와 동등한 범위 내에 있는 모든 기술 사상은 본 발명의 실시예의 권리범위에 포함되는 것으로 해석되어야 할 것이다.

부호의 설명

[0137] 300: 네트워크 스위치 병렬화 장치

310: 통신부 320: 메모리

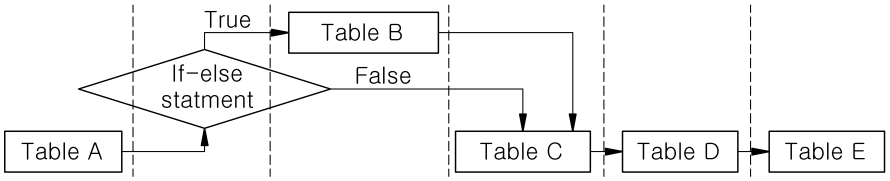
330: 프로세서

410: 정적 분석부 420: 사용-정의 분석부

430: 데이터 의존성 분석부 440: 병렬화 처리부

도면

도면1


True
Table B
If-else
statment
False
Table A
Table C
Table D
Table E

(a)

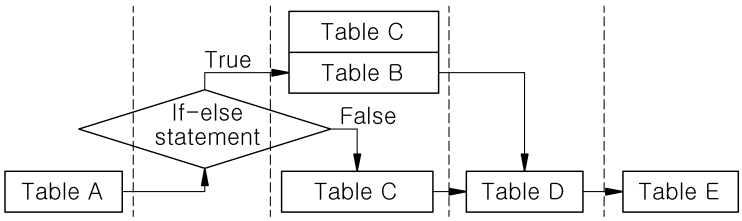
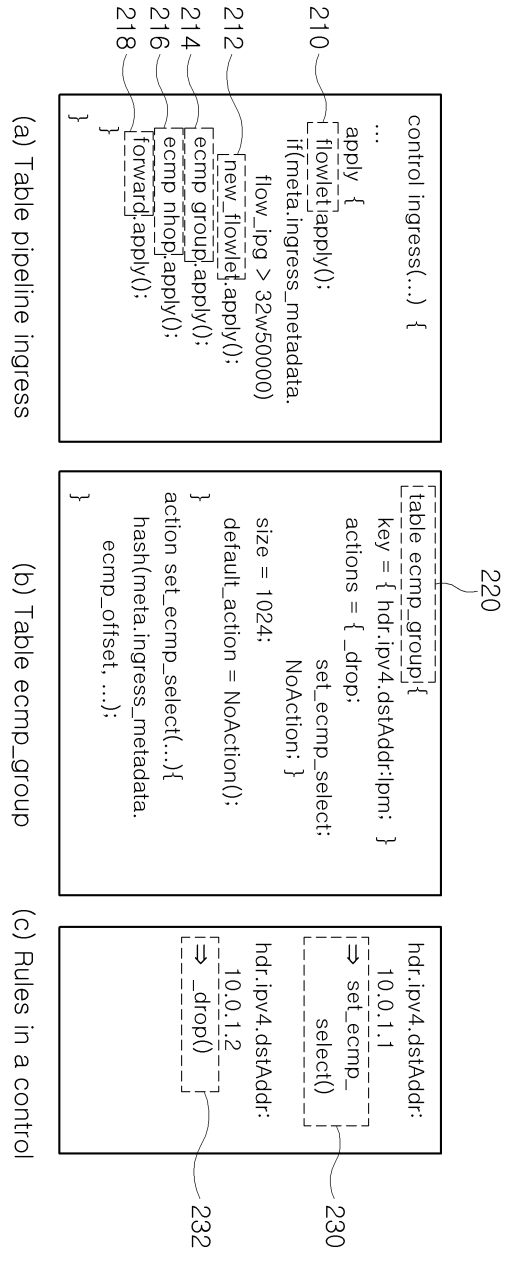


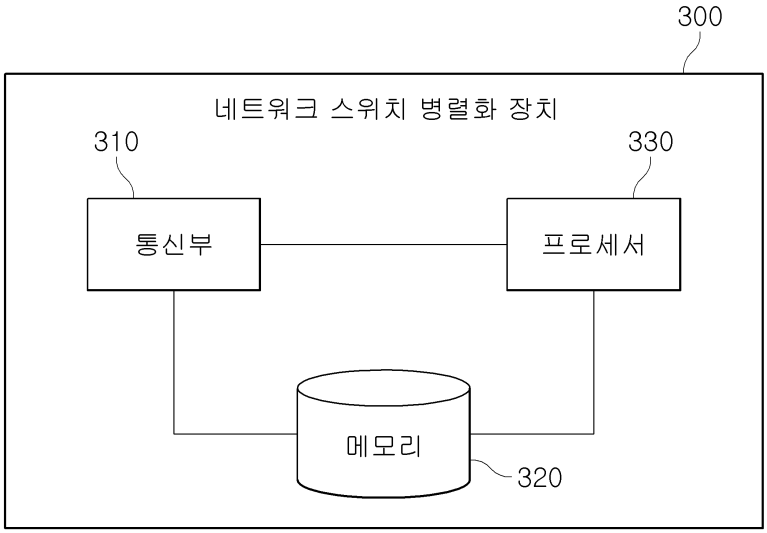
Table C
True
Table B
If-else
statement
False
Table A
Table C
Table D
Table E

(b)

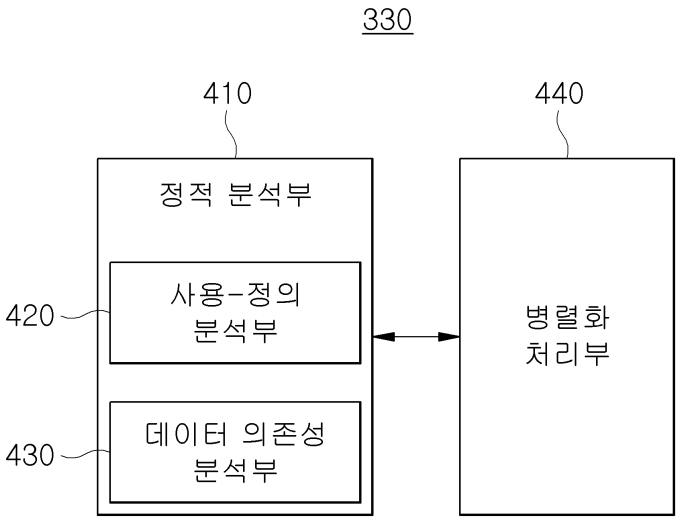
도면2


control ingress(...) {
...
apply {
flowlet.apply();
if(meta.ingress_metadata.
flow_ipg > 32w50000)
new_flowlet.apply();
ecmp_group.apply();
ecmp_nhop.apply();
forward.apply();
}
}
210
212
214
216
218
(a) Table pipeline ingress
table ecmp_group{
key = { hdr.ipv4.dstAddr:lpm; }
actions = { _drop;
set_ecmp_select;
NoAction; }
size = 1024;
default_action = NoAction();
}
action set_ecmp_select(...){
hash(meta.ingress_metadata.
ecmp_offset, ...);
}
220
(b) Table ecmp_group
hdr.ipv4.dstAddr:
10.0.1.1
⇒ set_ecmp_
select()
hdr.ipv4.dstAddr:
10.0.1.2
⇒ _drop()
230
232
(c) Rules in a control
plane

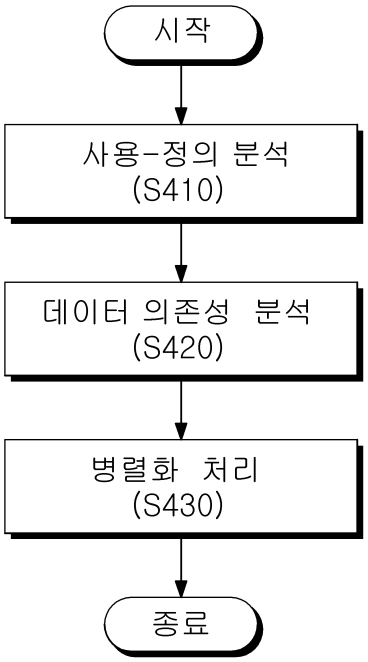
도면3


300
네트워크 스위치 병렬화 장치
310
330
통신부
프로세서
메모리
320

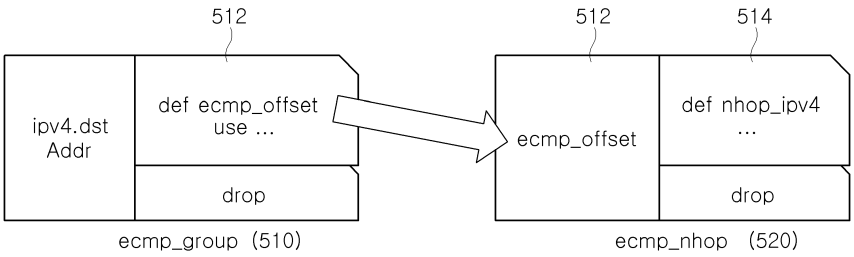
도면4a


330
410
440
정적 분석부
420
사용-정의
분석부
430
데이터 의존성
분석부
병렬화
처리부

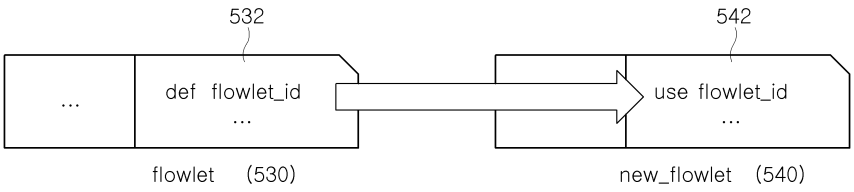
도면4b



도면5

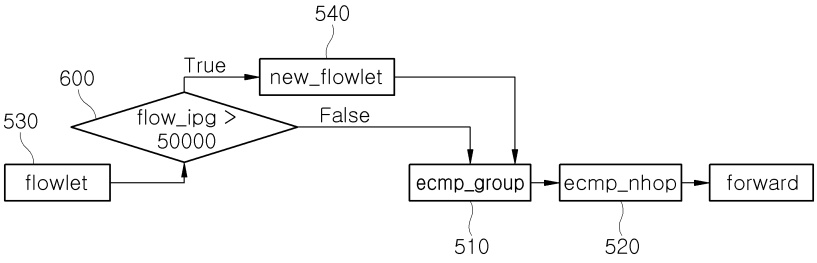


(a) Table-dependence between ecmp_group and ecmp_nhop

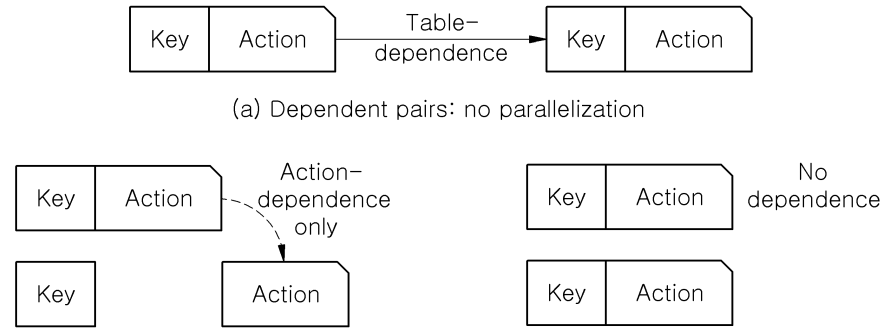


(b) Action-dependence between flowlet and new_flowlet

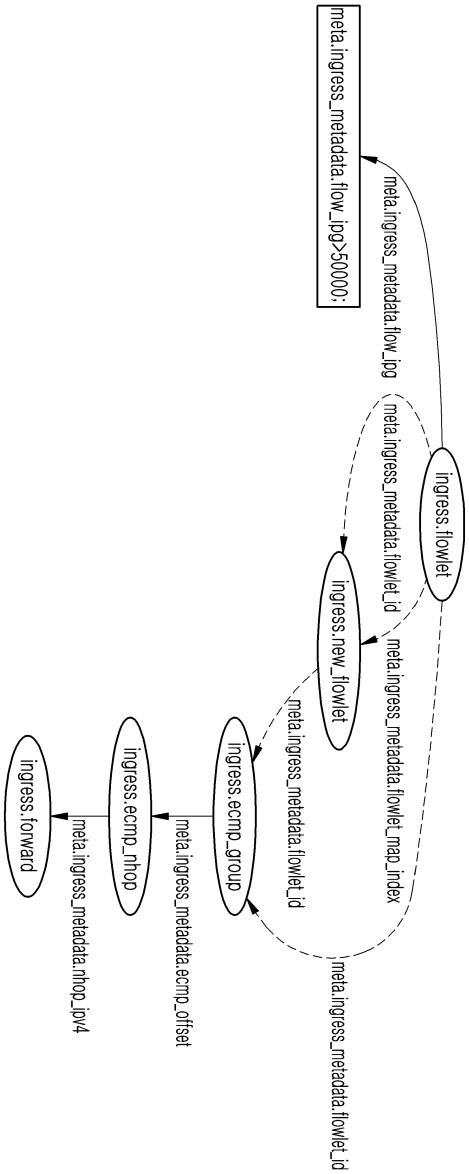
도면6



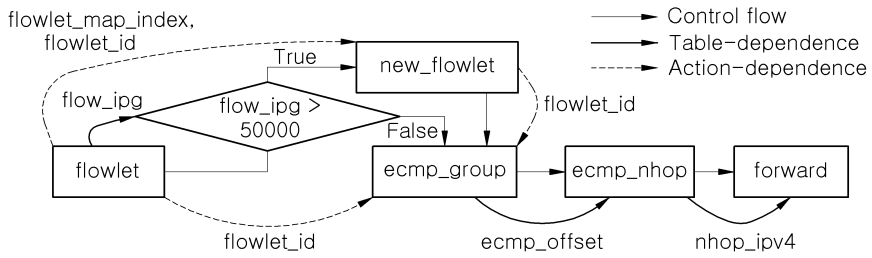
도면7


Key
Action
Table-dependence
Key
Action
(a) Dependent pairs: no parallelization
Key
Action
Action-dependence only
Key
Action
(b) Key-independent pairs: lookup parallelization
Key
Action
No dependence
Key
Action
(c) Table-independent pairs: table parallelization

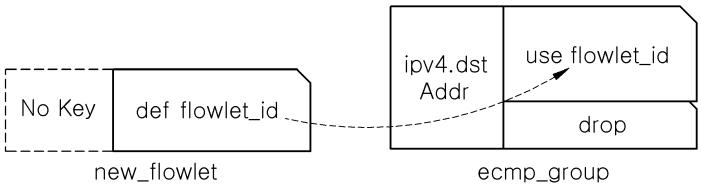
도면8


meta.ingress_metadata.flow_ipg>50000;
meta.ingress_metadata.flow_ipg
meta.ingress_metadata.flowlet_id
ingress.flowlet
ingress.new_flowlet
meta.ingress_metadata.flowlet_map_index
meta.ingress_metadata.flowlet_id
ingress.ecmp_group
meta.ingress_metadata.flowlet_id
meta.ingress_metadata.ecmp_offset
ingress.ecmp_nhop
meta.ingress_metadata.nhop_ipv4
ingress.forward

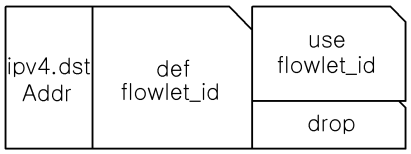
도면9



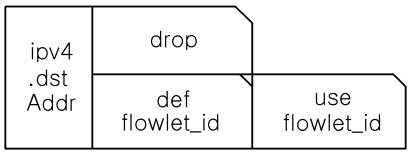
도면10



(a) Original table



(b) Lookup parallelization



(c) Packet dropping in early stage