



(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2017년04월04일
(11) 등록번호 10-1719087
(24) 등록일자 2017년03월16일

(51) 국제특허분류(Int. Cl.)
G06F 9/45 (2006.01)
(52) CPC특허분류
G06F 8/4452 (2013.01)
G06F 8/433 (2013.01)
(21) 출원번호 10-2015-7022960
(22) 출원일자(국제) 2014년03월27일
심사청구일자 2015년08월24일
(85) 번역문제출일자 2015년08월24일
(65) 공개번호 10-2015-0110742
(43) 공개일자 2015년10월02일
(86) 국제출원번호 PCT/US2014/031975
(87) 국제공개번호 WO 2014/160837
국제공개일자 2014년10월02일
(30) 우선권주장
13/853,430 2013년03월29일 미국(US)
(56) 선행기술조사문헌
US05920724 A*
Ebcioğlu, Kemal, et al. Optimizations and
oracle parallelism with dynamic translation.
Proceedings of the 32nd annual ACM/IEEE
international symposium on Microarchitecture.
IEEE Computer Society, 1999*
US20030135724 A1*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
인텔 코퍼레이션
미합중국 캘리포니아 95054 산타클라라 미션 칼리
지 블러바드 2200
(72) 발명자
롱, 홍보
미국 95116 캘리포니아주 산 호세 헤르모실라 웨
이 1637
박현철
미국 95051 캘리포니아주 산타 클라라 줄리엣 레
인 3600
우, 유형
미국 94306 캘리포니아주 팔로 알토 브라이언트
에스티. 3658
(74) 대리인
양영준, 백만기

전체 청구항 수 : 총 22 항

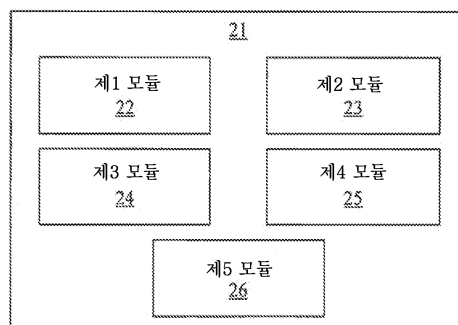
심사관 : 서광훈

(54) 발명의 명칭 런타임에서의 소프트웨어 파이프라이닝

(57) 요약

장치들과 방법들이 동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정하고 또한 규정된 시간 기간 내에 결정된 성능 레벨을 달성하는 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행하는 것을 제공할 수 있다. 한 예에서, 의존성 그래프가 하나 이상의 루프들에 대해 확립될 수 있고, 각각의 의존성 그래프는 성능 레벨에 기초하여 단들로 분할될 수 있다.

대표도 - 도1



명세서

청구범위

청구항 1

파이프라이닝을 실행하기 위한 장치로서:

동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정하는 제1 회로;

규정된 시간 기간 내에 상기 결정된 성능 레벨을 달성하는 상기 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행하는 제2 회로; 및

상기 하나 이상의 루프들에 대한 의존성 그래프를 확립하고 상기 성능 레벨에 기초하여 각각의 의존성 그래프를 단(stage)들로 분할하는 제3 회로 - 상기 성능 레벨은 상기 의존성 그래프로부터 하워드 반복 정책(Howard Iteration Policy; HIP) 알고리즘을 통해 계산되는 계산된 최소 개시 구간(minimum initiation interval; MII)에 기초한 것이고, 상기 HIP 알고리즘은 지수보다 작은(less than exponential) 복잡도를 갖도록 제한됨 - 를 포함하는 파이프라이닝 실행 장치.

청구항 2

삭제

청구항 3

제1항에 있어서,

조정된 국소적(local) 스케줄이 상기 성능 레벨을 달성하는지를 결정하는 제4 회로; 및

상기 조정된 국소적 스케줄이 상기 성능 레벨을 달성하지 못하면 상기 규정된 시간 기간이 발생했는지를 결정하는 제5 회로

를 더 포함하는 파이프라이닝 실행 장치.

청구항 4

제3항에 있어서, 상기 HIP 알고리즘은 미리 정해진 횟수보다 더 실행되지 않도록 제한되는 파이프라이닝 실행 장치.

청구항 5

제3항에 있어서, 상기 실행된 코드 최적화들은 하드웨어 리소스들을 할당하는 것 및 국소적 스케줄을 생성하기 위해 각각의 분할된 단에 포함되는 명령어들에 대한 국소적 의존성들을 분해(resolve)하는 것을 포함하는 파이프라이닝 실행 장치.

청구항 6

제5항에 있어서, 상기 실행된 코드 최적화들은 루프 전달된 의존성(loop-carried dependence)들을 분해하고 또한 상기 조정된 국소적 스케줄을 생성하기 위해 상기 국소적 스케줄을 조정하는 것을 포함하는 파이프라이닝 실행 장치.

청구항 7

제6항에 있어서, 상기 실행된 코드 최적화들은 상기 조정된 국소적 스케줄이 상기 성능 레벨을 달성하지 못하고 또한 상기 규정된 시간 기간이 발생하지 않았다면 다음 반복 스케줄을 생성하기 위해 이들의 현재 단들로부터 다음 단으로 명령어들을 이동시키는 것을 포함하는 파이프라이닝 실행 장치.

청구항 8

제7항에 있어서, 상기 실행된 코드 최적화들은, 상기 조정된 국소적 스케줄이 상기 성능 레벨을 달성할 때까지, 상기 규정된 시간 기간이 발생하지 않는 한, 상기 다음 반복 스케줄을 이용하여 새로운 조정된 국소적 스케줄을 생성하기 위해 반복되는 파이프라이닝 실행 장치.

청구항 9

하나 이상의 명령어들을 포함하는 적어도 하나의 비밀시적인 컴퓨터 판독 가능 기록 매체로서,

상기 하나 이상의 명령어들은, 컴퓨팅 디바이스상에서 실행될 때, 상기 컴퓨팅 디바이스를,

동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정하고;

규정된 시간 기간 내에 상기 결정된 성능 레벨을 달성하는 상기 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행하고;

상기 하나 이상의 루프들에 대한 의존성 그래프를 확립하고 상기 성능 레벨에 기초하여 각각의 의존성 그래프를 단들로 분할하도록 구성하는 - 상기 성능 레벨은 상기 의존성 그래프로부터 하위도 반복 정책(HIP) 알고리즘을 통해 계산되는 계산된 최소 개시 구간(MII)에 기초한 것이고, 상기 HIP 알고리즘은 지수보다 작은 복잡도를 갖도록 제한됨 -

적어도 하나의 비밀시적인 컴퓨터 판독 가능 기록 매체.

청구항 10

삭제

청구항 11

제9항에 있어서, 프로세서에 의해 실행될 때, 상기 프로세서를,

조정된 국소적 스케줄이 상기 성능 레벨을 달성하는지를 결정하고; 및

상기 조정된 국소적 스케줄이 상기 성능 레벨을 달성하지 못하면 상기 규정된 시간 기간이 발생했는지를 결정하도록 구성하는 하나 이상의 명령어들을 더 포함하는 적어도 하나의 비밀시적인 컴퓨터 판독 가능 기록 매체.

청구항 12

제11항에 있어서, 상기 HIP 알고리즘은 미리 정해진 횟수보다 더 실행되지 않도록 제한되는 적어도 하나의 비밀시적인 컴퓨터 판독 가능 기록 매체.

청구항 13

제11항에 있어서, 상기 실행된 코드 최적화들은 하드웨어 리소스들을 할당하는 것 및 국소적 스케줄을 생성하기 위해 각각의 분할된 단에 포함되는 명령어들에 대한 국소적 의존성들을 분해하는 것을 포함하는 적어도 하나의 비밀시적인 컴퓨터 판독 가능 기록 매체.

청구항 14

제13항에 있어서, 상기 실행된 코드 최적화들은 루프 전달된 의존성들을 분해하고 조정된 국소적 스케줄을 생성하기 위해 상기 국소적 스케줄을 조정하는 것을 포함하는 적어도 하나의 비밀시적인 컴퓨터 판독 가능 기록 매체.

청구항 15

제14항에 있어서, 상기 실행된 코드 최적화들은 상기 조정된 국소적 스케줄이 상기 성능 레벨을 달성하지 못하고 또한 상기 규정된 시간 기간이 발생하지 않았다면 다음 반복 스케줄을 생성하기 위해 이들의 현재 단들로부터 다음 단으로 명령어들을 이동시키는 것을 포함하는 적어도 하나의 비밀시적인 컴퓨터 판독 가능 기록 매체.

청구항 16

제15항에 있어서, 상기 실행된 코드 최적화들은, 상기 조정된 국소적 스케줄이 상기 성능 레벨을 달성할 때까지, 상기 규정된 시간 기간이 발생하지 않는 한, 상기 다음 반복 스케줄을 이용하여 새로운 조정된 국소적

스케줄을 생성하기 위해 반복되는 적어도 하나의 비밀시적인 컴퓨터 판독 가능 기록 매체.

청구항 17

파이프라이닝 실행 방법으로서:

동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정하는 단계;

규정된 시간 기간 내에 상기 결정된 성능 레벨을 달성하는 상기 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행하는 단계; 및

상기 하나 이상의 루프들에 대한 의존성 그래프를 확립하고 상기 성능 레벨에 기초하여 각각의 의존성 그래프를 단들로 분할하는 단계 - 상기 성능 레벨은 상기 의존성 그래프로부터 하워드 반복 정책(HIP) 알고리즘을 통해 계산되는 계산된 최소 개시 구간(MII)에 기초한 것이고, 상기 HIP 알고리즘은 지수보다 작은 복잡도를 갖도록 제한됨 -

를 포함하는 파이프라이닝 실행 방법.

청구항 18

삭제

청구항 19

제17항에 있어서,

조정된 국소적 스케줄이 상기 성능 레벨을 달성하는지를 결정하는 단계; 및

상기 조정된 국소적 스케줄이 상기 성능 레벨을 달성하지 못하면 상기 규정된 시간 기간이 발생했는지를 결정하는 단계

를 더 포함하는 파이프라이닝 실행 방법.

청구항 20

제19항에 있어서, 상기 HIP 알고리즘은 미리 정해진 횟수보다 더 실행되지 않도록 제한되는 파이프라이닝 실행 방법.

청구항 21

제19항에 있어서, 상기 실행된 코드 최적화들은 하드웨어 리소스들을 할당하는 것 및 국소적 스케줄을 생성하기 위해 각각의 분할된 단에 포함되는 명령어들에 대한 국소적 의존성들을 분해하는 것을 포함하는 파이프라이닝 실행 방법.

청구항 22

제21항에 있어서, 상기 실행된 코드 최적화들은 루프 전달된 의존성들을 분해하고 조정된 국소적 스케줄을 생성하기 위해 상기 국소적 스케줄을 조정하는 것을 포함하는 파이프라이닝 실행 방법.

청구항 23

제22항에 있어서, 상기 실행된 코드 최적화들은 상기 조정된 국소적 스케줄이 상기 성능 레벨을 달성하지 못하고 또한 상기 규정된 시간 기간이 발생하지 않았다면 다음 반복 스케줄을 생성하기 위해 이들의 현재 단들로부터 다음 단으로 명령어들을 이동시키는 것을 포함하는 파이프라이닝 실행 방법.

청구항 24

제23항에 있어서, 상기 실행된 코드 최적화들은, 상기 조정된 국소적 스케줄이 상기 성능 레벨을 달성할 때까지, 상기 규정된 시간 기간이 발생하지 않는 한, 상기 다음 반복 스케줄을 이용하여 새로운 조정된 국소적 스케줄을 생성하기 위해 반복되는 파이프라이닝 실행 방법.

청구항 25

파이프라이닝을 실행하기 위한 장치로서:

동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정하기 위한 수단;

규정된 시간 기간 내에 상기 결정된 성능 레벨을 달성하는 상기 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행하기 위한 수단; 및

상기 하나 이상의 루프들에 대한 의존성 그래프를 확립하고 상기 성능 레벨에 기초하여 각각의 의존성 그래프를 단들로 분할하기 위한 수단 - 상기 성능 레벨은 상기 의존성 그래프로부터 하워드 반복 정책(HIP) 알고리즘을 통해 계산되는 계산된 최소 개시 구간(MII)에 기초한 것이고, 상기 HIP 알고리즘은 지수보다 작은 복잡도를 갖도록 제한됨 -

을 포함하는 파이프라이닝 실행 장치.

발명의 설명

기술 분야

[0001] 본 명세서에서 기술되는 실시예들은 일반적으로 소프트웨어 파이프라이닝에 관련된 것인데, 보다 상세하게는 동적 컴파일러들에서 소프트웨어 파이프라이닝을 실행하는 것과 관련된다.

배경 기술

[0002] 소프트웨어 파이프라이닝은 루프들로부터의 명령어 레벨 병렬성을 이용하는 효율적 컴파일러 기술일 수 있다. 종래의 소프트웨어 파이프라이닝은 정적 컴파일러들에서의 전개에 제한될 수 있는데, 이는 하드웨어/소프트웨어 협조 설계, 동적 언어들, 클라우드 컴퓨팅 및 임베디드 시스템들에 수반되는 것들과 같은 동적 컴파일 활동들에 적합하지 않을 수 있다.

도면의 간단한 설명

[0003] 본 발명의 실시예들의 각종 장점들은 통상의 기술자가 이하의 설명 및 첨부된 청구 범위를 읽고, 이하의 도면들을 참조함으로써 명백해질 것이다.

도 1은 실시예에 따른 장치의 예의 블록도이다.

도 2a 및 도 2b는 실시예들에 따른 방법들의 예들의 흐름도들이다.

도 3a-3e는 실시예에 따른 시나리오의 그림 도해들이다.

도 4는 실시예에 따른 시스템의 예의 블록도이다.

도 5는 실시예에 따른 프로세서의 예의 블록도이다.

발명을 실시하기 위한 구체적인 내용

[0004] 도 1을 지금 참조하면, 동적 컴파일러들을 위해 소프트웨어 파이프라이닝을 실행하기 위한 장치(21)가 도해된다. 도해된 장치(21)는 제1 모듈(22), 제2 모듈(23), 제3 모듈(24), 제4 모듈(25) 및 제5 모듈(26)을 포함한다. 제1 모듈(22)은 동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정할 수 있고, 여기서 성능 레벨은 최소 개시 구간(minimum initiation interval: MII)을 계산함으로써 결정된다. 제2 모듈(23)은 규정된 시간 기간 내에 결정된 성능 레벨을 달성하는 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행할 수 있으며, 여기서 실행된 코드 최적화들은 하드웨어 리소스들을 할당하고 또한 국소적 스케줄을 생성하기 위해 각각의 분할된 단에 포함되는 명령어들을 위한 국소적 의존성(local dependency)들을 결정하는 것을 포함한다.

[0005] 덧붙여, 실행된 코드 최적화들은 조정된 국소적 스케줄을 생성하기 위해 루프 전달된 의존성(loop-carried dependence)들을 분해(resolve)하도록 국소적 스케줄을 조정하고 및 조정된 국소적 스케줄이 성능 레벨을 달성하지 못하고 규정된 시간 기간이 발생하지 않았다면 다음 반복 스케줄을 생성하기 위해 명령어들을 이들의 현재 단들로부터 다음 단으로 이동시키는 것을 포함할 수 있다. 실행된 코드 최적화들은 조정된 국소적 스케줄이 성능 레벨을 달성할 때까지 규정된 시간 기간이 발생하지 않는 한 다음 반복 스케줄을 이용하여 새로운 조정된 국소적 스케줄을 생성하도록 반복될 수 있다.

- [0006] 제3 모듈(24)은 하나 이상의 루프들에 대한 의존성 그래프를 확립할 수 있고 성능 레벨에 기초하여 각각의 의존성 그래프를 단들로 분할할 수 있다. 제4 모듈(25)은 조정된 국소적 스케줄이 성능 레벨을 달성하는지를 결정할 수 있고 제5 모듈(26)은 조정된 국소적 스케줄이 성능 레벨을 달성하지 못하면 규정된 시간 기간이 발생했는지를 결정할 수 있다. 그러므로 도해된 장치(21)는 하드웨어/소프트웨어 협조 설계, 동적 언어들, 클라우드 컴퓨팅, 및 임베디드 시스템들에 수반되는 것들과 같은 동적 컴파일 활동들을 위해 코드를 최적화하는 데에 유용할 수 있다.
- [0007] 도 2a는 동적 컴파일러들을 위해 런타임에서 소프트웨어 파이프라이닝을 실행하는 방법을 도해한다. 도해된 방법은 RAM(Random Access Memory), ROM(Read Only Memory), PROM(Programmable ROM), 플래시 메모리 등과 같은 기계 또는 컴퓨터 판독 가능 매체 내에, 예를 들면 PLA(Programmable Logic Array)들, FPGA(Field Programmable Gate Array)들, CPLD(Complex Programmable Logic Device)들과 같은 구성 가능 로직 내에, 예를 들면 ASIC(Application Specific Integrated circuit), CMOS(Complementary Metal Oxide Semiconductor) 또는 TTL(Transistor-Transistor Logic) 기술 등과 같은 어셈블리 언어 프로그래밍 및 회로 기술을 사용하는 고정 기능 하드웨어 내에, 또는 이들의 임의의 조합 내에 저장되는 로직 명령어들의 세트 및/또는 펌웨어로서 구현될 수 있다. 예를 들어, 예시된 방법에 도시된 동작들을 수행해 내기 위한 컴퓨터 프로그램 코드는 C++ 등과 같은 객체 지향 프로그래밍 언어 및 "C" 프로그래밍 언어 또는 유사한 프로그래밍 언어와 같은 전통적 절차적 프로그래밍 언어들을 포함하는 하나 이상의 프로그래밍 언어들의 임의 조합으로 작성될 수 있다. 더욱이, 방법은 전술한 회로 기술들 중 임의의 것을 이용하여 구현될 수 있다.
- [0008] 방법은 임의의 의존성 또는 리소스 제약들을 위반하지 않고서 루프 반복들의 실행을 중첩함으로써 소프트웨어 파이프라이닝 최적화들을 일반적으로 실행할 수 있다. 예시된 방법은 MII를 계산함으로써 처리 블록 11에서 시작한다. MII는 루프 반복의 시작으로부터 다음 루프 반복의 시작까지 간의 최단 시간 기간일 수 있다. 한 예에서, 개시 구간이 더 짧을수록, 루프 처리를 완료하는 데에 요구되는 주기들의 수는 더 작아지고, 소프트웨어 파이프라이닝을 이용하는 루프 처리는 더 빠르게 실행될 수 있다. 또한, MII를 계산하는 것의 부산물로서, 임계 연산들이 식별되고 후속 스케줄링 동안 우선 순위를 부여 받을 수 있다.
- [0009] 주어진 루프, 예를 들어, 도 3a에 도해된 루프에 대해, 의존성 그래프가 확립된다. 의존성 그래프는 예를 들어 도 3b에 도해된다. 그래프에서, 두 개의 루프 전달된 의존성 예지, $d \rightarrow b$ 및 $b \rightarrow a$ 가 예시되는데, 각각은 (1)의 반복 거리 값을 갖는다. 의존성 예지 $c \rightarrow d$ 는 2 주기의 대기 시간(latency)을 갖는다. 기타 의존성들은 국소적이고, 그러므로 이들은 (0)의 반복 거리 값을 갖는다. 강 연결 요소들(strongly connected components: SCC)이 예를 들어 타잔(Tarjan)의 강 연결 요소 알고리즘과 같은 알고리즘을 이용하여 의존성 그래프로부터 식별될 수 있다.
- [0010] 최소 개시 구간은, 예를 들어 하워드(Howard) 반복 정책 알고리즘과 같은 알고리즘에서 식별된 SCC를 이용하여 계산될 수 있는데, 이는 다음과 같이 구현될 수 있다:

MIICalculation(H), 여기서 H는 SCC가 하워드 반복 정책 알고리즘에서 스캐닝되는 최대 횟수임.

의존성 그래프에서 SCC들을 식별한다

$MII \leftarrow -\infty$

for each SCC do

하워드 반복 정책 알고리즘에 의해 SCC의 일반화된 최대 회로 평균을 계산하지만, 정책이 H회만큼 SCC를 스캐닝한 후 안정되지 않으면 중단한다

$MII = \max(MII, \text{the generalized maximal circuit mean})$

isCritical(a) $\leftarrow$ FALSE $\forall$ operation a

for each SCC do

if $MII = \text{SCC의 일반화된 최대 회로 평균}$ then

isCritical(a) $\leftarrow$ TRUE $\forall$ operation a $\in$ SCC의 정책 그래프에서의 임계

주기들.

[0011]

[0012]

처리 블록 12에서, 예시된 방법은 최적 스케줄에 근접한 초기 스케줄을 형성하기 위해 의존성 그래프를 초기 단들의 세트로 분할한다. 처리 블록 12는 또한 시간 기간 변수(예를 들어, "i", 하기에서 더 상세히 논의됨)를 초기화하는 것을 제공할 수 있다. 각각의 단(stage)은 MII의 미리 정해진 높이를 가질 수 있다. 예를 들어, 도 3b에 도해된 의존성 그래프가 각각의 단이 MII 주기를 갖도록 절단되면, 결과로 생기는 초기 스케줄이, MII가 두 개(2)와 동등한 경우에 도 3c에 예시된다.

[0013]

의존성 그래프는, 예를 들어 벨만-포드(Bellman-Ford) 알고리즘과 같은 알고리즘을 이용하여 분할될 수 있다. 알고리즘은 의존성 그래프에서의 의사 시작 노드(pseudo start node)로부터 각각의 노드까지의 최장 경로를 계산할 수 있다. 한 예에서, MII에 의해 나뉘셈된 노드에 대한 경로 길이는 노드에 대한 단 인덱스(stage index)가 된다. 그러한 접근법은 초기 스케줄을 만들 수 있는데, 여기서 각각의 노드에 단이 할당되었다. 이 처리에서, 각각의 에지(a→b, δ , d)는 $\delta - d * MII$ 의 길이를 가질 수 있고, 의사 시작은 길이 0를 가진 임의의 다른 노드에 연결될 수 있다. 에지 a→b, δ , d는 a → b로의 의존성을 가지며, 여기서 δ 은 대기 시간이고 d는 반복 거리이다. 계산된 최장 경로는 단들로 나누어질 수 있다.

[0014]

알고리즘은, 예를 들어, 다음과 같이 벨만-포드 알고리즘의 반복들의 횟수를 제어하기 위한 임계값 B를 이용하여 구현될 수 있다. 일단 반복들의 횟수가 B에 도달하고 알고리즘이 끝나지 않으면, 전체 방법은 중단된다.

```

len(a)  $\leftarrow$  0  $\forall$  operation a
for i  $\leftarrow$  1 to  $|V| - 1$  do
    changed  $\leftarrow$  FALSE
    for each operation b in the order in the original loop body do
        len(b)  $\leftarrow$  max(len(b), len(a) +  $\delta - d * MII$ )  $\forall (a \rightarrow b, \delta, d)$ 
        changed  $\leftarrow$  TRUE if len(b)가 이전 단계에 의해 변경되면
    if changed=FALSE then
        break
    if (i <  $|V| - 1$ ) 및 (i = B) then
        소프트웨어 파이프라이닝 중단
stage(a)  $\leftarrow \frac{\text{len(a)}}{MII} \quad \forall \text{ node (a)}$ 
offset(a)  $\leftarrow$  len(a) mod MII  $\forall$  node a
d  $\leftarrow$  d + stage(b) - stage(a)  $\forall (a \rightarrow b, \delta, d)$ 

```

[0015]

[0016]

처리 블록 13에서, 예시된 방법은 파이프라이닝된 스케줄을 생성하고 계산된 MII를 달성하기 위해 코드 최적화들을 실행하기를 시작한다. 특히, 처리 블록 13에서, 예시된 방법은 하드웨어 리소스들을 할당하고, 분할된 단들에서 명령어들에 대한 국소적 의존성들을 시행하고, 국소적 스케줄을 생성할 수 있다. 명령어 내에서의 각각의 연산은 리소스와 오프셋을 할당 받을 수 있다. 처리 블록 13은, 예를 들어 다음과 같은 알고리즘으로 구현될 수 있다:

```

height(a)  $\leftarrow -\infty \quad \forall$  operation a
for each operation a in reverse order in the current kernel do
    for each outgoing edge (a  $\rightarrow$  b,  $\delta$ , d) do
        height(a)  $\leftarrow$  max(height(a); height(b) +  $\delta - d * MII$ )
height(a)  $\leftarrow +\infty \quad \forall a, \text{isCritical(a)} = \text{TRUE}$ 

```

[0017]

[0018]

예를 들어, 국소적 의존성들상에서 및 높이 우선 순위 기능을 가진 리소스 제약들 하에서 작용하는 임의의 리스트 스케줄링을 적용한다.

[0019]

처리 블록 14에서, 예시된 방법은 루프 전달된 의존성들을 분해하기 위해 국소적 스케줄을 조정함으로써 코드 최적화들을 계속 실행할 수 있다. 국소적 스케줄로 분해되지 않은 각각의 루프 전달 의존성에 대해, 하나 이상의 비어있는 단들이 루프 전달된 의존성을 분해하도록 시도하기 위해 소스와 목적지 명령어 사이에 삽입될 수 있다(즉, 수평 커널 확장). 그러한 접근법은 2개의 연산 사이의 시간 거리가 의존성의 대기 시간보다 작지 않도록 연장한다. 도 3d는 루프 전달된 의존성들을 가진 국소적 스케줄을 예시한다. 도 3e에 예시된 바와 같이, c와 d 사이에 비어있는 단을 추가함으로써 의존성이 분해된다.

[0020]

처리 블록 13 동안 분해되지 않은 루프 전달된 의존성들을 분해하기 위해 커널 확장이 예를 들어 다음과 같이 실행될 수 있다:

[0021]

(a $\rightarrow$ b, δ , d)가 임의의 실행된 코드 최적화들 전의 최초 의존성 그래프에서의 미분해된 루프 전달된 의존성 그래프라고 하자. 제1 선택으로서, 비어있는 단들을 루프 전달된 의존성의 소스와 싱크 노드들 사이에 삽입하는 것에 의한 수평 커널 확장에 의해 시간 거리 d를 증가시키기 위한 시도가 실행된다. 그러한 접근법은 이들의 시간 차이가 의존성의 대기 시간을 만족하거나 초과하도록 강제할 수 있고, 그러므로 개시 구간을 증가시키

지 않고서 의존성을 분해할 수 있다.

[0022] 또 다른 접근법으로서, 비어있는 단들을 삽입함으로써 분해될 수 없는 임의의 루프 전달된 의존성이 있는 경우에, 개시 구간(II)이 증가될 수 있다.

[0023] 새로운 단들을 삽입할 때, 이미 충족되었던 이전 의존성들은 위반되지 않을 수 있다. 특정 의존성의 싱크 노드 단이 의존성의 소스 노드 단 전에 있다면, 이후 단을 그들 사이에 삽입하는 것은 시간 차이만을 감소시킬 수 있다. 간단히 말하면, 의존성에 대해 "금지된(forbidden)" 단들 및 "확장 가능한(expandable)" 단들이 있을 수 있다. 임의의 금지된(확장 가능한) 단의 바로 전에 비어있는 단들을 삽입하는 것은 의존성의 소스와 싱크 간의 시간 차이를 감소(증가)시킬 수 있다. 형식적으로, 예지 $e: a \rightarrow b$ 에 대해, $\text{forbidden}(e) = \{s | \text{stage}(b) < s \leq \text{stage}(a)\}$ $\text{expandable}(e) = \{s | \text{stage}(a) < s \leq \text{stage}(b)\}$. 단(o)은 연산 o의 단 인덱스일 수 있다.

[0024] 둘 모두의 세트들이 소스와 싱크 노드들의 단들 간의 단들로 구성될 수 있고, 비어있는 단들이 소스와 싱크 노드들의 단들의 둘 모두의 전에 (또는 둘 모두의 후에) 삽입되면, 시간 차이에 대한 어떤 영향도 없을 것이라는 점을 유의하라.

[0025] 이제 미분해된 의존성에 대해, 개시 구간(II)을 변경하지 않고서 이것을 분해하기 위해서, 그 확장 가능한 단들 중 하나의 단의 바로 전에 삽입될 단들의 최소 수는 이하로부터 도출될 수 있다:

[0026] $\delta \leq d * II + \text{offset}(b) - \text{offset}(a)$, 여기서 $\text{offset}(o)$ 은 커널에서의 그의 단에서의 연산 o의 시간 오프셋이다.

[0027] $\text{newStages}(e) = \delta - \text{offset}(b) + \text{offset}(a) / II - d$

[0028] 본질적으로, 금지된 단들의 전역적 세트가 계산될 수 있는데, 여기서 전역적 세트는 모든 의존성들의 금지된 단들의 합집합이다. 각각의 단 s에 대해, 레코드가 0으로 초기화되는 변수, $\text{newStagesBefore}(s)$ 에 의해 그 전에 삽입되는 새로운 단들의 수로 만들어질 수 있다. 이후 모든 루프 전달된 의존성들은 스캐닝될 수 있고, 예시된 방법은 어디에 새로운 단들을 삽입할 지와 얼마나 많이 삽입할 지를 판정한다. 임의의 루프 전달된 의존성 ($a \rightarrow b$)이 비어있는 단들을 삽입함으로써 분해될 수 없는 경우에, 개시 구간(II)이 이것을 분해하기 위해 증가된다. 이것을 분해하기 위한 최소 II는 $\text{feasibleII}(e) = [\text{offset}(b) + \text{offset}(a)]/d$ 일 수 있다.

[0029] 커널 확장을 실행하기 위한 알고리즘은, 예를 들어 다음과 같이 구현될 수 있다:

$m \leftarrow$ 현재 커널의 전체 단들

$F \leftarrow \cup \text{forbidden}(e) \quad \forall \text{ dependence } e$

$\text{newStagesBefore}(s) \leftarrow 0 \quad \forall \text{ stage } s$

for each dependence $e: (a \rightarrow b, \delta, d) \quad d \neq 0$ do

 if $\delta \leq d * II + \text{offset}(b) - \text{offset}(a)$ 이 성립하지 않으면, then

 if $\text{expandable}(e) - F \neq \{\}$ then

$n \leftarrow \text{newStages}(e)$

$m \leftarrow m + n$

 if $m > S$, 여기서 S는 커널에 허용되는 단들의 최대 수입,

then $II \leftarrow \max(II, \text{feasibleII}(e)) \quad \forall \text{ dependence } e$, 및 then 커널 확장 처리는 종료함.

$s \leftarrow \text{any stage} \in \text{expandable}(e) - F$

$\text{newStagesBefore}(x) \leftarrow \text{newStagesBefore}(x) + n$

$\forall \text{ stage } x \geq s$

$\text{stage}(a) \leftarrow \text{stage}(a) + \text{newStagesBefore}(\text{stage}(a)) \quad \forall \text{ node } a$

$d \leftarrow d + \text{newStagesBefore}(\text{stage}(b)) - \text{newStagesBefore}(\text{stage}(a))$ 및 이것 후에 $II \leftarrow$

$\max(II, \text{feasibleII}(e)), \quad \forall (a \rightarrow b, \delta, d)$ 라고 한다.

[0030]

[0031] 처리 블록 15에서, 예시된 방법은, 처리 블록 14가 실행된 후에 결과로 생기는 스케줄인 조정된 국소적 스케줄이 계산된 MII를 달성했는지를 결정할 수 있다. 만일 그렇다면, 처리는 중지되고 예시된 방법은 처리 블록 17

에서 종결된다. 만일 그렇지 않다면, 예시된 방법은 처리 블록 16에서 규정된 시간 기간이 발생했는지를 결정한다. 예를 들어, 예시된 방법은 시간 기간 변수(예를 들어, 변수 "i")가 규정된 시간 기간과 동등한지를 결정할 수 있다. 규정된 시간 기간은, 예를 들어 미리 정의된 반복들의 수와 같은 임계값일 수 있다. 예시된 방법이 처리 블록들 13 및 14를 실행할 때마다, 반복이 발생했고, 변수 (i)는 처리 블록 19에서 그에 따라서 증분된다. 실행된 코드 최적화들은 예시된 방법이 중단되기 전에 제한된 횟수만큼 실행될 수 있다. 그러므로, 예시된 방법이 규정된 시간 기간이 발생했다고 결정하면, 예시된 방법은 처리 블록 17에서 종결된다.

[0032] 처리 블록 18에서, 예시된 방법은 다음 처리 반복을 위한 더 콤팩트한 스케줄을 생성하기 위해 조정된 국소적 스케줄을 수정한다. 조정된 국소적 스케줄은 계산된 MII를 달성하는 스케줄을 생성하기 위한 노력의 일환으로 현재 스케줄에 기초하여 이들의 현재 단들로부터 이들의 다음 단들로 명령어들을 이동시킴으로써 수정될 수 있다. 그러한 접근법은 코드 최적화들의 다음 반복을 위한 다음 반복 스케줄을 생성한다. 예시된 방법은, 계산된 MII를 달성한 조정된 국소적 스케줄이 획득될 때까지 규정된 시간 기간이 발생하지 않는 한 (즉, $i \neq$ 규정된 시간 기간) 코드 최적화들을 반복한다.

[0033] 처리 블록 19에서, 예시된 방법은 다음 처리 반복을 위해 변수 (i)를 증분시킬 수 있고, 방법은 실행을 계속하기 위해 처리 블록 13으로 리턴한다.

[0034] 도 2b는 파이프라이닝을 실행하는 방법을 예시한다. 예시된 방법은, RAM, ROM, PROM, 플래시 메모리 등과 같은 기계 또는 컴퓨터 판독 가능 매체 내에, 예를 들면 PLA들, FPGA들, CPLD들과 같은 구성 가능 로직 내에, 예를 들면 ASIC, CMOS 또는 TTL 기술 등과 같은 어셈블리 언어 프로그래밍 및 회로 기술을 사용하는 고정 기능 하드웨어 내에, 또는 이들의 임의의 조합 내에 저장되는 로직 명령어들의 세트 및/또는 펌웨어로서 구현될 수 있다.

[0035] 처리 블록 32에서의 예시된 방법은 동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정하는 것을 제공하며, 여기서 성능 레벨은 MII를 계산함으로써 결정된다. 처리 블록 33에서, 예시된 방법은 처리 블록 33에서 규정된 시간 기간 내에 결정된 성능 레벨을 달성하는 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행하는 것을 제공하는데, 여기서 실행된 코드 최적화들은 하드웨어 리소스들을 할당하는 것과 국소적 스케줄을 생성하기 위해 각각의 분할된 단에 포함되는 명령어들에 대한 국소적 의존성들을 결정하는 것을 포함한다.

[0036] 또한, 실행된 코드 최적화들은, 조정된 국소적 스케줄을 생성하기 위해 루프 전달된 의존성들을 분해하기 위해 국소적 스케줄을 조정하는 것 및 조정된 국소적 스케줄이 성능 레벨을 달성하지 못하고 규정된 시간 기간이 발생하지 않았다면 다음 반복 스케줄을 생성하기 위해 이들의 현재 단들로부터 다음 단으로 명령어들을 이동시키는 것을 포함할 수 있다. 실행된 코드 최적화들은, 조정된 국소적 스케줄이 성능 레벨을 달성할 때까지 규정된 시간 기간이 발생하지 않는 한 다음 반복 스케줄을 이용하여 새롭게 조정된 국소적 스케줄을 생성하기 위해 반복될 수 있다.

[0037] 처리 블록 34에서, 하나 이상의 루프들에 대한 의존성 그래프가 확립되고 각각의 의존성 그래프는 성능 레벨에 기초하여 단들로 분할된다. 처리 블록 35에서 예시된 방법은 조정된 국소적 스케줄이 성능 레벨을 달성하는지를 결정하고, 처리 블록 36에서 예시된 방법은 조정된 국소적 스케줄이 성능 레벨을 달성하지 못하면 규정된 시간 기간이 발생했는지를 결정한다.

[0038] 도 4를 지금 참조하면, 마이크로프로세서 시스템의 다이어그램이 도해된다. 제1 처리 요소(1070) 및 제2 처리 요소(1080)를 포함할 수 있는 멀티프로세서 시스템(1000)이 도 4에 도시된다. 2개의 처리 요소(1070 및 1080)가 도시되어 있지만, 시스템(1000)의 실시예는 하나의 그러한 처리 요소만을 포함할 수 있다는 것을 이해해야 한다.

[0039] 시스템(1000)은 포인트 투 포인트 상호접속 시스템으로서 도시되어 있는데, 여기서 제1 처리 요소(1070) 및 제2 처리 요소(1080)는 포인트 투 포인트 상호접속(1050)을 통해 결합된다. 도 4에 예시된 상호접속들 중 임의의 것 또는 모든 것들이 포인트 투 포인트 상호접속이 아니라 멀티드롭 버스로서 구현될 수 있다는 것을 이해해야 한다.

[0040] 도 4에 도시된 바와 같이, 각각의 처리 요소들(1070 및 1080)은 제1 및 제2 프로세서 코어들(즉, 프로세서 코어들(1074a 및 1074b)과 프로세서 코어들(1084a 및 1084b))을 포함하는 멀티코어 프로세서들일 수 있다. 그와 같은 코어들(1074, 1074b, 1084a, 1084b)은 명령어 코드를 실행하도록 구성될 수 있다.

[0041] 각각의 처리 요소(1070, 1080)는 적어도 하나의 공유 캐시(1896)를 포함할 수 있다. 공유 캐시(1896a, 1896b)는, 제각기, 코어들(1074a, 1074b, 1084a, 및 1084b)과 같은, 프로세서의 하나 이상의 컴포넌트들에 의

해 활용되는 데이터(예를 들어, 명령어들)를 저장할 수 있다. 예를 들면, 공유 캐시는 프로세서의 컴포넌트들에 의한 보다 빠른 액세스를 위해 메모리(1032, 1034)에 저장되는 데이터를 국소적으로 캐시할 수 있다. 하나 이상의 실시예들에서, 공유 캐시는 레벨 2(L2), 레벨 3(L3), 레벨 4(L4), 또는 다른 레벨의 캐시와 같은 하나 이상의 중간 레벨 캐시들, 최종 레벨 캐시(LLC), 및/또는 이들의 조합들을 포함할 수 있다.

[0042] 2개의 처리 요소(1070, 1080)만이 도시되어 있지만, 본 발명의 범위는 이것에만 한정되지는 않는다는 것을 이해해야 한다. 기타 실시예들에서, 주어진 프로세서에는 하나 이상의 부가적 처리 요소들이 존재할 수 있다. 대안으로서, 처리 요소들(1070, 1080) 중 하나 이상은 프로세서 외의 요소, 예를 들어 가속기 또는 필드 프로그래머블 게이트 어레이일 수 있다. 예를 들어, 부가적 처리 요소(들)는 제1 프로세서(1070)와 동일한 부가적 프로세서(들), 제1 프로세서(1070)와는 이질적이거나 비대칭인 부가적 프로세서(들), (예를 들어, 그래픽 가속기들 또는 디지털 신호 프로세싱(DSP) 유닛들과 같은) 가속기들, 필드 프로그래머블 게이트 어레이들, 또는 임의의 다른 처리 요소를 포함할 수 있다. 아키텍처, 마이크로아키텍처, 열, 전력 소비 특성, 및 그와 유사한 것을 포함하는 장점의 범위를 놓고 볼 때, 처리 요소들(1070, 1080) 사이에는 다양한 차이들이 존재할 수 있다. 이들 차이들은 처리 요소들(1070, 1080) 간의 비대칭성 및 이질성으로서 자기 자신들을 실효적으로 드러낼 수 있다. 적어도 일 실시예에서, 다양한 처리 요소들(1070, 1080)은 동일 다이 패키지에 상주할 수 있다.

[0043] 제1 처리 요소(1070)는 메모리 컨트롤러 로직(MC)(1072) 및 포인트 투 포인트(P-P) 인터페이스들(1076 및 1078)을 더 포함할 수 있다. 유사하게, 제2 처리 요소(1080)는 MC(1082) 및 P-P 인터페이스들(1086 및 1088)을 포함할 수 있다. 도 4에 도시된 바와 같이, MC들(1072 및 1082)은 프로세서들을 제각기 메모리들, 즉 메모리(1032) 및 메모리(1034)에 결합하며, 이들 메모리들은 제각기 프로세서들에 국소적으로 소속되는 메인 메모리의 부분들일 수 있다. MC 로직(1072 및 1082)이 처리 요소(1070, 1080) 내에 통합된 것으로 도시되어 있지만, 대안적인 실시예들에서 MC 로직은 처리 요소들(1070, 1080) 내에 통합되는 것이 아니라 처리 요소들 외부에 있는 이산 로직일 수 있다.

[0044] 제1 처리 요소(1070) 및 제2 처리 요소(1080)는, 제각기 P-P 상호접속들(1076, 1086 및 1084)을 통해 I/O 서브시스템(1090)에 결합될 수 있다. 도 4에 도시된 바와 같이, I/O 서브시스템(1090)은 P-P 인터페이스들(1094 및 1098)을 포함할 수 있다. 게다가, I/O 서브시스템(1090)은 I/O 서브시스템(1090)을 고성능 그래픽 엔진(1038)과 결합하기 위한 인터페이스(1092)를 포함할 수 있다. 일 실시예에서, 버스가 그래픽 엔진(1038)을 I/O 서브시스템(1090)과 결합하는데 사용될 수 있다. 대안으로서, 포인트 투 포인트 상호접속(1039)이 이들 컴포넌트들을 결합할 수 있다.

[0045] 다음으로, I/O 서브시스템(1090)은 인터페이스(1096)를 통해 제1 버스(1016)에 결합될 수 있다. 일 실시예에서, 제1 버스(1016)는 PCI(Peripheral Component Interconnect) 버스일 수 있거나, 또는 PCI Express 버스 또는 또 다른 제3세대 I/O 상호접속 버스와 같은 버스일 수 있지만, 본 발명의 범위는 이것들에만 제한되지는 않는다.

[0046] 도 4에 도시된 바와 같이, 다양한 I/O 장치들(1014)이, 제1 버스(1016)를 제2 버스(1010)에 결합할 수 있는 버스 브리지(1018)와 함께, 제1 버스(1016)에 결합될 수 있다. 일 실시예에서, 제2 버스(1010)는 LPC(low pin count) 버스일 수 있다. 일 실시예에서, 예를 들어 키보드/마우스(1012), (다음 차례로 컴퓨터 네트워크(503)와 통신할 수 있는) 통신 디바이스(들)(1026), 및 코드(1030)를 포함할 수 있는 디스크 드라이브 또는 다른 대용량 저장 디바이스와 같은 데이터 스토리지 유닛(1019)을 포함하는 다양한 디바이스들이 제2 버스(1010)에 결합될 수 있다. 코드(1030)는, 예를 들어 도 2a 및/또는 2b의 방법들과 같은, 앞서 기술된 방법들 중 하나 이상의 방법들의 실시예들을 실행하기 위한 명령어들을 포함할 수 있다. 또한, 오디오 I/O(1024)가 제2 버스(1010)에 결합될 수 있다.

[0047] 기타 실시예들이 상정된다는 점을 유의하라. 예를 들어, 도 4의 포인트 투 포인트 아키텍처 대신에, 시스템은 멀티드롭 버스 또는 다른 그러한 통신 토폴로지를 구현할 수 있다. 또한, 도 4의 요소들은 도 4에 도시된 것보다 더 많은 또는 더 적은 집적된 칩들을 이용하여 대안적으로 구현될 수 있다.

[0048] 도 5는 일 실시예에 따른 프로세서 코어(200)를 예시한다. 프로세서 코어(200)는 마이크로프로세서, 임베디드 프로세서, 디지털 신호 프로세서(DSP), 네트워크 프로세서, 또는 코드를 실행하기 위한 기타 디바이스와 같은 임의의 유형의 프로세서를 위한 코어일 수 있다. 하나의 프로세서 코어(200)만이 도 5에 예시되기는 하였지만, 처리 요소는 대안적으로 도 5에 예시된 프로세서 코어(200)의 하나보다 많은 것을 포함할 수 있다. 프로세서 코어(200)는 단일 스레드형(single-threaded) 코어일 수 있거나, 또는 적어도 일 실시예에서 프로세서 코어(200)는 코어당 하나보다 많은 하드웨어 스레드 콘텍스트(또는 "논리 프로세서")를 포함할 수 있다는 점에서 멀

티스레드형(multithreaded)일 수 있다.

- [0049] 도 5는 또한 프로세서(200)에 결합되는 메모리(270)를 예시한다. 메모리(270)는 통상의 기술자에게 공지되거나 다른 방식으로 이용 가능한 (메모리 계층구조의 다양한 계층들을 포함하는) 매우 넓은 범위의 메모리들 중 임의의 것일 수 있다. 메모리(270)는 프로세서(200) 코어에 의해 실행될 하나 이상의 코드(213) 명령어(들)를 포함할 수 있으며, 여기서 코드(213)는 도 2a 및/또는 도 2b에 예시된 방법들을 구현할 수 있다. 프로세서 코어(200)는 코드(213)에 의해 표시되는 명령어들의 프로그램 시퀀스를 추종한다. 각각의 명령어는 프론트 엔드 부분(210)에 진입할 수 있고, 하나 이상의 디코더들(220)에 의해 처리될 수 있다. 디코더(220)는 그 출력으로서 사전 정의된 포맷의 고정 폭 마이크로 연산(fixed width micro operation)과 같은 마이크로 연산을 생성할 수 있거나, 또는 최초 코드 명령어를 반영하는 다른 명령어들, 마이크로명령어들, 또는 제어 신호들을 생성할 수 있다. 예시된 프론트 엔드(210)는 또한 레지스터 리네이밍 로직(225) 및 스케줄링 로직(230)을 포함할 수 있는데, 이들은 일반적으로 리소스들을 할당하고 또한 실행용 변환 명령어에 대응하는 연산을 큐잉한다.
- [0050] 프로세서(200)는 실행 유닛들의 세트(255-1 내지 255-N)를 갖는 실행 로직(250)을 포함하는 것으로 도시되어 있다. 몇몇 실시예들은 특정 평선들 또는 평선들의 세트들에 전용되는 다수의 실행 유닛을 포함할 수 있다. 기타 실시예들은 특정 평선을 실행할 수 있는 단 하나의 실행 유닛 또는 하나의 실행 유닛을 포함할 수 있다. 예시된 실행 로직(250)은 코드 명령어들에 의해 특정되는 연산들을 실행한다.
- [0051] 코드 명령어들에 의해 특정되는 연산들의 실행의 완료 후에, 백 엔드 로직(260)은 코드(213)의 명령어들을 리타이어(retire)한다. 일 실시예에서, 프로세서(200)는 비순차적 실행(out of order execution)을 허용하지만, 명령어들의 순차적 리타이어먼트를 요구한다. 리타이어먼트 로직(265)은 통상의 기술자에게 공지된 바와 같은 다양한 형태들(예를 들어, 재정렬 버퍼 또는 그와 유사한 것)을 취할 수 있다. 이러한 방식으로, 프로세서 코어(200)는, 적어도 디코더에 의해 생성되는 출력, 레지스터 리네이밍 로직(225)에 의해 활용되는 하드웨어 레지스터들과 테이블들, 및 실행 로직(250)에 의해 수정되는 임의의 레지스터들(도시되지 않음)의 관점에서 코드(213)의 실행 동안 변환된다.
- [0052] 도 5에 예시되지 않았지만, 처리 요소는 프로세서 코어(200)와 함께 칩상에 다른 요소들을 포함할 수 있다. 예를 들어, 처리 요소는 프로세서 코어(200)와 함께 메모리 제어 로직을 포함할 수 있다. 처리 요소는 I/O 제어 로직을 포함할 수 있고 및/또는 메모리 제어 로직과 통합되는 I/O 제어 로직을 포함할 수 있다. 처리 요소는 하나 이상의 캐시들을 또한 포함할 수 있다.
- [0053] 추가적 언급들 및 예들:
- [0054] 예 1은 파이프라이닝을 실행하기 위한 장치를 포함할 수 있다. 장치는 동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정하는 제1 모듈; 및 규정된 시간 기간 내에 결정된 성능 레벨을 달성하는 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행하는 제2 모듈을 포함할 수 있다.
- [0055] 예 2는 예 2의 장치를 포함할 수 있고, 추가로 하나 이상의 루프들에 대한 의존성 그래프를 확립하고 또한 성능 레벨에 기초하여 각각의 의존성 그래프를 단들로 분할하는 제3 모듈을 포함한다.
- [0056] 예 3은 제2항의 장치를 포함할 수 있고, 조정된 국소적 스케줄이 성능 레벨을 달성하는지를 결정하는 제4 모듈 및 조정된 국소적 스케줄이 성능 레벨을 달성하지 못하면 규정된 시간 기간이 발생했는지를 결정하는 제5 모듈을 추가로 포함한다.
- [0057] 예 4는 예들 1 내지 3 중 임의의 것의 장치를 포함할 수 있으며, 여기서 성능 레벨은 MII를 계산함으로써 결정된다.
- [0058] 예 5는 예 3의 장치를 포함할 수 있으며, 여기서 실행된 코드 최적화들은 하드웨어 리소스들을 할당하는 것 및 국소적 스케줄을 생성하기 위해 각각의 분할된 단에 포함되는 모든 명령어들에 대한 국소적 의존성들을 분해하는 것을 포함한다.
- [0059] 예 6은 예 5의 장치를 포함할 수 있으며, 여기서 실행된 코드 최적화들은 루프 전달된 의존성들을 분해하고 또한 조정된 국소적 스케줄을 생성하기 위해 국소적 스케줄을 조정하는 것을 포함한다.
- [0060] 예 7은 예 6의 장치를 포함할 수 있으며, 여기서 실행된 코드 최적화들은 조정된 국소적 스케줄이 성능 레벨을 달성하지 못하고 또한 규정된 시간 기간이 발생하지 않았다면 다음 반복 스케줄을 생성하기 위해 이들의 현재

단들로부터 다음 단으로 명령어들을 이동시키는 것을 포함한다.

- [0061] 예 8은 예 7의 장치를 포함할 수 있고, 여기서 실행된 코드 최적화들은, 조정된 국소적 스케줄이 성능 레벨을 달성할 때까지, 규정된 시간 기간이 발생하지 않는 한, 다음 반복 스케줄을 이용하여 새로운 조정된 국소적 스케줄을 생성하기 위해 반복된다.
- [0062] 예 9는 컴퓨팅 디바이스상에서 실행될 때 컴퓨팅 디바이스로 하여금 동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정하고; 및 규정된 시간 기간 내에 결정된 성능 레벨을 달성하는 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행하도록 구성하는 하나 이상의 명령어들을 포함하는 적어도 하나의 컴퓨터 판독 가능 매체를 포함할 수 있다.
- [0063] 예 10은 예 9의 적어도 하나의 컴퓨터 판독 가능 매체를 포함할 수 있고, 하나 이상의 루프들에 대한 의존성 그래프를 확립하는 것 및 성능 레벨에 기초하여 각각의 의존성 그래프를 단들로 분할하는 것을 추가로 포함한다.
- [0064] 예 11은 예 10의 적어도 하나의 컴퓨터 판독 가능 매체를 포함할 수 있고, 조정된 국소적 스케줄이 성능 레벨을 달성하는지를 결정하는 것; 및 조정된 국소적 스케줄이 성능 레벨을 달성하지 못하면 규정된 시간 기간이 발생했는지를 결정하는 것을 추가로 포함한다.
- [0065] 예 12는 예들 9 내지 11 중 임의의 것의 적어도 하나의 컴퓨터 판독 가능 매체를 포함할 수 있으며, 여기서 성능 레벨은 MII를 계산함으로써 결정되는 것이다.
- [0066] 예 13은 예 11의 적어도 하나의 컴퓨터 판독 가능 매체를 포함할 수 있으며, 여기서 실행된 코드 최적화들은 하드웨어 리소스들을 할당하는 것 및 국소적 스케줄을 생성하기 위해 각각의 분할된 단에 포함되는 명령어들에 대한 국소적 의존성들을 분해하는 것을 포함한다.
- [0067] 예 14는 예 13의 적어도 하나의 컴퓨터 판독 가능 매체를 포함할 수 있으며, 여기서 실행된 코드 최적화들은 루프 전달된 의존성들을 분해하고 조정된 국소적 스케줄을 생성하기 위해 국소적 스케줄을 조정하는 것을 포함한다.
- [0068] 예 15는 예 14의 적어도 하나의 컴퓨터 판독 가능 매체를 포함할 수 있으며, 여기서 실행된 코드 최적화들은 조정된 국소적 스케줄이 성능 레벨을 달성하지 못하고 또한 규정된 시간 기간이 발생하지 않았다면 다음 반복 스케줄을 생성하기 위해 이들의 현재 단들로부터 다음 단으로 명령어들을 이동시키는 것을 포함한다.
- [0069] 예 16은 제15항의 적어도 하나의 컴퓨터 판독 가능 매체를 포함할 수 있고, 여기서 실행된 코드 최적화들은, 조정된 국소적 스케줄이 성능 레벨을 달성할 때까지, 규정된 시간 기간이 발생하지 않는 한, 다음 반복 스케줄을 이용하여 새로운 조정된 국소적 스케줄을 생성하기 위해 반복된다.
- [0070] 예 17은 동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정하는 단계 및 규정된 시간 기간 내에 결정된 성능 레벨을 달성하는 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행하는 단계를 포함하는 파이프라이닝 실행 방법을 포함할 수 있다.
- [0071] 예 18 예 17의 방법을 포함할 수 있고, 하나 이상의 루프들에 대한 의존성 그래프를 확립하고 또한 성능 레벨에 기초하여 각각의 의존성 그래프를 단들로 분할하는 단계를 추가로 포함한다.
- [0072] 예 19는 예 18의 방법을 포함할 수 있고, 조정된 국소적 스케줄이 성능 레벨을 달성하는지를 결정하는 단계 및 조정된 국소적 스케줄이 성능 레벨을 달성하지 못하면 규정된 시간 기간이 발생했는지를 결정하는 단계를 추가로 포함한다.
- [0073] 예 20은 예들 17 내지 19 중 임의의 것의 방법을 포함할 수 있으며, 여기서 성능 레벨은 MII를 계산함으로써 결정되는 것이다.
- [0074] 예 21은 예 19의 방법을 포함할 수 있으며, 여기서 실행된 코드 최적화들은 하드웨어 리소스들을 할당하는 것 및 국소적 스케줄을 생성하기 위해 각각의 분할된 단에 포함되는 명령어들에 대한 국소적 의존성들을 분해하는 것을 포함한다.
- [0075] 예 22는 예 21의 방법을 포함할 수 있으며, 여기서 실행된 코드 최적화들은 루프 전달된 의존성들을 분해하고 또한 조정된 국소적 스케줄을 생성하기 위해 국소적 스케줄을 조정하는 것을 포함한다.
- [0076] 예 23은 예 22의 방법을 포함할 수 있으며, 여기서 실행된 코드 최적화들은 조정된 국소적 스케줄이 성능 레벨을 달성하지 못하고 또한 규정된 시간 기간이 발생하지 않았다면 다음 반복 스케줄을 생성하기 위해 이들의 현

재 단들로부터 다음 단으로 명령어들을 이동시키는 것을 포함한다.

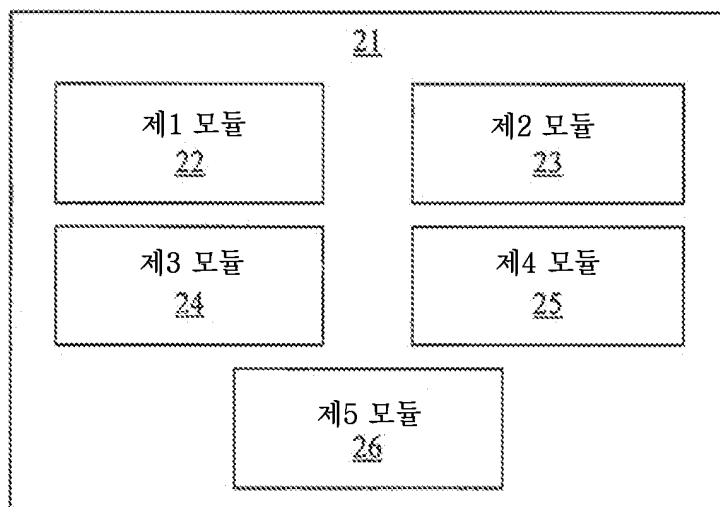
- [0077] 예 24는 예 23의 방법을 포함할 수 있고, 여기서 실행된 코드 최적화들은, 조정된 국소적 스케줄이 성능 레벨을 달성할 때까지, 규정된 시간 기간이 발생하지 않는 한, 다음 반복 스케줄을 이용하여 새로운 조정된 국소적 스케줄을 생성하기 위해 반복된다.
- [0078] 예 25는 예들 17 내지 24의 방법들 중 임의의 것을 실행하기 위한 수단을 포함하는 파이프라이닝 실행 장치를 포함할 수 있다.
- [0079] 예들은, 프로세서에 의해 실행될 때, 프로세서로 하여금 동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 성능 레벨을 결정하고 및 규정된 시간 기간 내에 결정된 성능 레벨을 달성하는 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하기 위해 코드 최적화들을 실행하도록 야기하는 명령어들의 세트를 갖는 컴퓨터 판독 가능 매체를 포함할 수 있다.
- [0080] 다른 예들이 프로세서에 의해 실행될 때 프로세서로 하여금 동적 컴파일러에 의해 하나 이상의 루프들을 처리하기 위한 MII를 계산하고 및 규정된 시간 기간 내에 계산되는 MII를 달성하는 하나 이상의 루프들을 위한 파이프라이닝된 스케줄을 생성하도록 야기하는 명령어들의 세트를 갖는 컴퓨터 판독 가능 매체를 제공할 수 있다.
- [0081] 본 발명의 실시예들은 모든 유형들의 반도체 집적 회로("IC") 칩들에서 사용되도록 적용될 수 있다. 이들 IC 칩들의 예들은 프로세서들, 컨트롤러들, 칩셋 컴포넌트들, PLA(programmable logic arrays), 메모리 칩들, 네트워크 칩들, 및 그와 유사한 것을 포함하지만, 이것들에만 한정되지는 않는다. 또한, 도면들의 몇몇에서, 신호 도전체 라인들이 라인들로 표현된다. 몇몇은, 더 많은 구성 요소 신호 경로들을 표시하기 위해, 수 라벨을 가지기 위해, 다수의 구성 요소 신호 경로들을 표시하기 위해, 및/또는 하나 이상의 단부들에서 화살표들을 가지기 위해, 주 정보 흐름 방향을 표시하기 위해 상이할 수 있다. 그러나, 이것이 제한을 위한 것으로 해석해서는 안 된다. 그보다는, 그러한 부가적 상세 사항들은 회로의 보다 쉬운 이해를 촉진하기 위해 하나 이상의 예시적 실시예들과 연계하여 이용될 수 있다. 임의의 표현된 신호 라인들은, 부가적 정보를 갖는지에 상관없이, 다중 방향으로 진행할 수 있고 또한 예를 들어 차동 쌍들, 광섬유 라인들, 및/또는 단일 종단 라인들로 구현되는 디지털 또는 아날로그 라인들인 임의의 적절한 유형의 신호 방식들로 구현될 수 있는 하나 이상의 신호들을 실제적으로 포함할 수 있다.
- [0082] 예들의 크기들/모델들/값들/범위들이 주어질 수 있지만, 본 발명의 실시예들은 이것들과 동일한 것에만 한정되지는 않는다. 시간이 흘러 제조 기술들(예를 들어, 포토리소그래피)이 성숙해짐에 따라, 더 작은 크기의 디바이스들이 제조될 수 있을 것으로 예상된다. 또한, 예시 및 논의의 단순화를 위해 그리고 본 발명의 실시예들의 소정 양태들을 모호하게 하지 않기 위해, IC 칩들 및 기타 컴포넌트들에의 공지된 전력/접지 접속들은 도면들 내에 도시될 수도 있고 도시되지 않을 수도 있다. 또한, 배열들은, 본 발명의 실시예들을 불명확하게 하지 않기 위해 그리고 또한 블록도 배열들의 구현에 관한 특정 사항들이 실시예가 그 내에서 구현되게 될 플랫폼에 크게 의존한다는 사실, 즉 그러한 특정 사항들이 통상의 기술자의 이해 범위 내에 있을 것이 틀림없다는 사실에 비추어 그러한 블록도 형태로 도시될 수 있다. 특정 상세 사항들(예를 들어, 회로들)이 본 발명의 예시적 실시예들을 기술하기 위해 제시되는 경우, 본 발명의 실시예들은 이러한 특정 상세 사항들 없이 또는 이러한 이들 특정 사항들의 변형으로 실시될 수 있음이 통상의 기술자에게 명백할 것이다. 따라서, 이 설명은 제한을 위한 것이 아니라 예시를 위한 것으로 간주되어야 한다.
- [0083] 몇몇 실시예들은, 예를 들어 기계에 의해 실행되는 경우에 기계로 하여금 실시예들에 따른 방법 및/또는 동작들을 실행하도록 야기할 수 있는 명령어 또는 명령어들의 세트를 저장할 수 있는 기계 또는 유형 컴퓨터 판독 가능 매체 또는 물품을 이용하여 구현될 수 있다. 이러한 기계는, 예를 들어, 임의의 적절한 프로세싱 플랫폼, 컴퓨팅 플랫폼, 컴퓨팅 디바이스, 프로세싱 디바이스, 컴퓨팅 시스템, 프로세싱 시스템, 컴퓨터, 프로세서, 또는 그와 유사한 것을 포함할 수 있고, 또한 하드웨어 및/또는 소프트웨어의 임의의 적절한 조합을 이용하여 구현될 수 있다.
- [0084] 기계 판독 가능 매체 또는 물품은, 예를 들어 메모리, 착탈식 또는 비착탈식 매체, 소거가능 또는 소거 불가능 매체, 기입가능 또는 재기입가능 매체, 디지털 또는 아날로그 매체, 하드 디스크, 플로피 디스크, CD-ROM(Compact Disk Read Only Memory), CD-R(Compact Disk Recordable), CD-RW(Compact Disk Rewriteable), 광학 디스크, 자기 매체, 광자기 매체, 착탈식 메모리 카드 또는 디스크, 다양한 유형의 DVD(Digital Versatile Disk), 테이프, 카세트, 또는 그와 유사한 것인, 임의의 적절한 유형의 메모리 유닛, 메모리 디바이스, 메모리 물품, 메모리 매체, 스토리지 디바이스, 스토리지 물품, 스토리지 매체 및/또는 스토리지 유닛을 예를 들어 포

함할 수 있다.

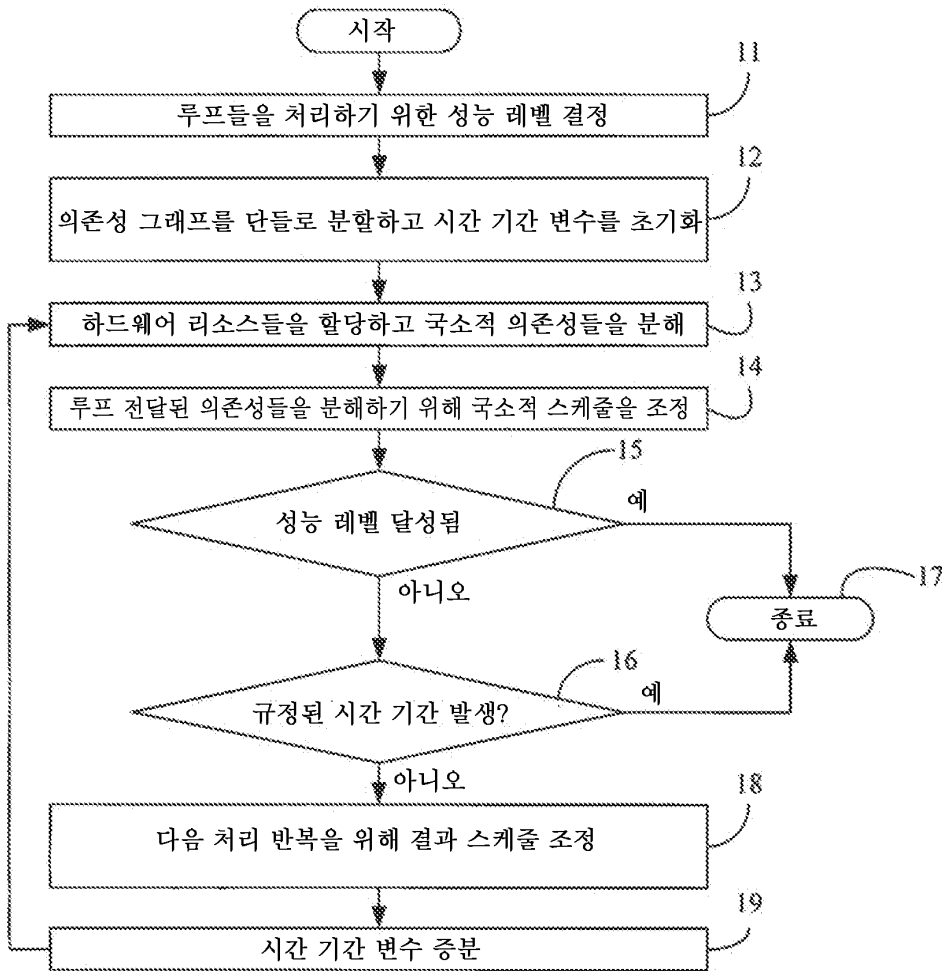
- [0085] 기계 판독 가능 매체는 기계에 의해 판독 가능한 형태로 정보를 저장, 전송, 또는 수신하는 임의의 메커니즘을 포함할 수 있고, 또한 매체는 프로그램 코드를 그를 통해 넘겨줄 수 있는 매체, 예컨대 안테나, 광 섬유, 통신 인터페이스 등을 포함할 수 있다. 프로그램 코드는 패킷들, 직렬 데이터, 병렬 데이터 등의 포맷으로 전송될 수 있으며, 압축되거나 암호화된 포맷으로 사용될 수 있다.
- [0086] 프로그램 코드 또는 명령어들은, 이를 테면, 고체 상태(solid-state) 메모리, 하드 드라이브, 플로피 디스크, 광 스토리지, 테이프, 플래시 메모리, 메모리 스틱, 디지털 비디오 디스크, DVD들, 기타 등등뿐만 아니라, 기계 접근 가능 생체 상태 보존 스토리지와 같은 더 색다른 매체를 포함하지만 이것들에만 한정되지는 않는 스토리지 디바이스들 및/또는 연관된 기계 판독 가능 또는 기계 접근 가능 매체와 같은, 예를 들어 휘발성 및/또는 비휘발성 메모리에 저장될 수 있다.
- [0087] 명령어들은 임의의 적합한 고급, 저급, 객체 지향, 비주얼, 컴파일링된 및/또는 인터프리팅된 프로그래밍 언어를 이용하여 구현되는, 소스 코드, 컴파일링된 코드, 인터프리팅된 코드, 실행가능 코드, 정적 코드, 동적 코드, 및 그와 유사한 것과 같은 임의의 적합한 유형의 코드를 포함할 수 있다.
- [0088] 구체적으로 달리 언급되지 않는 한, "처리", "컴퓨팅", "계산", "결정", 또는 그와 유사한 것과 같은 용어들은 컴퓨팅 시스템의 레지스터들 및/또는 메모리들 내에서 물리적 양들(예로서, 전자적임)로서 표현되는 데이터를 컴퓨팅 시스템의 메모리들, 레지스터들 또는 다른 그러한 정보 스토리지, 전송 또는 디스플레이 디바이스들 내에서 물리적인 양들로서 비슷하게 표현되는 다른 데이터가 되도록 조작 및/또는 변환하는 컴퓨터 또는 컴퓨팅 시스템, 또는 유사한 전자적 컴퓨팅 시스템 디바이스의 행동 및/또는 프로세스들을 지칭한다는 것을 알 수 있다. 실시예들은 이러한 맥락에만 한정되지는 않는다.
- [0089] 용어 "결합된다"는 관심 대상의 컴포넌트들 사이에서 직접적 또는 간접적인 임의 유형의 관계성을 지칭하기 위해 본 명세서에 사용될 수 있고, 전기적, 기계적, 유동적, 광학적, 전자기적, 전자기계적 또는 다른 접속들에 적용될 수 있다. 게다가, "제1", "제2" 등의 용어들은 본 명세서에서 단지 논의의 편의상 사용될 수 있으며, 달리 지적되지 않는 한 어떤 특정한 시간적 또는 연대기적 의미도 수반하지 않을 수 있다.
- [0090] 본 발명의 실시예들의 넓은 범위의 기술들이 다양한 형태들로 구현될 수 있다는 것을 상술된 설명으로부터 통상의 기술자는 알 것이다. 그러므로, 본 발명의 실시예들이 그 특정 예들과 관련하여 설명되었지만, 다른 수정들도 도면, 명세서, 및 하기 청구범위를 참조하면 통상의 기술자에게 자명하게 될 것이므로 본 발명의 실시예들의 진정한 범위는 예들에만 한정되지는 않을 것이다.

도면

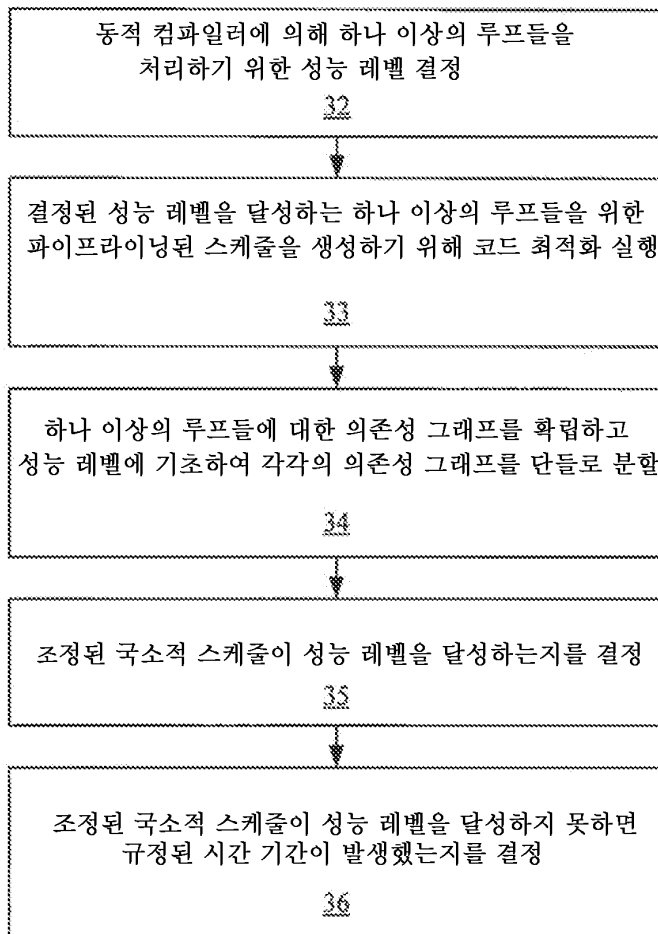
도면1



도면2a



도면2b

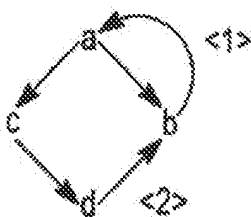


도면3a

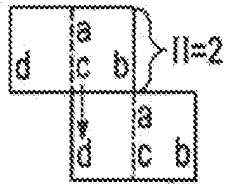
```

for (i=0; i<N; i++) {
    a
    b
    c
    d
}
    
```

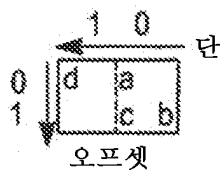
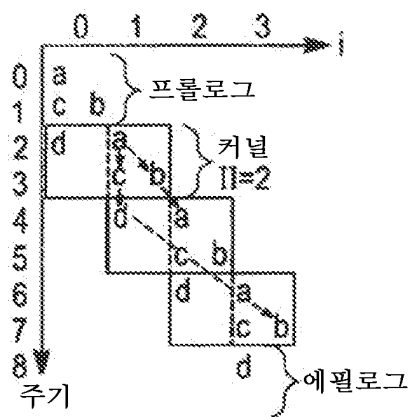
도면3b



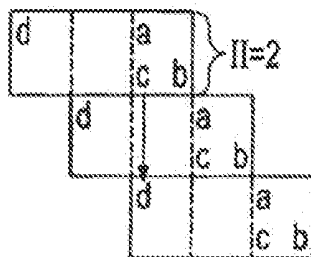
도면3c



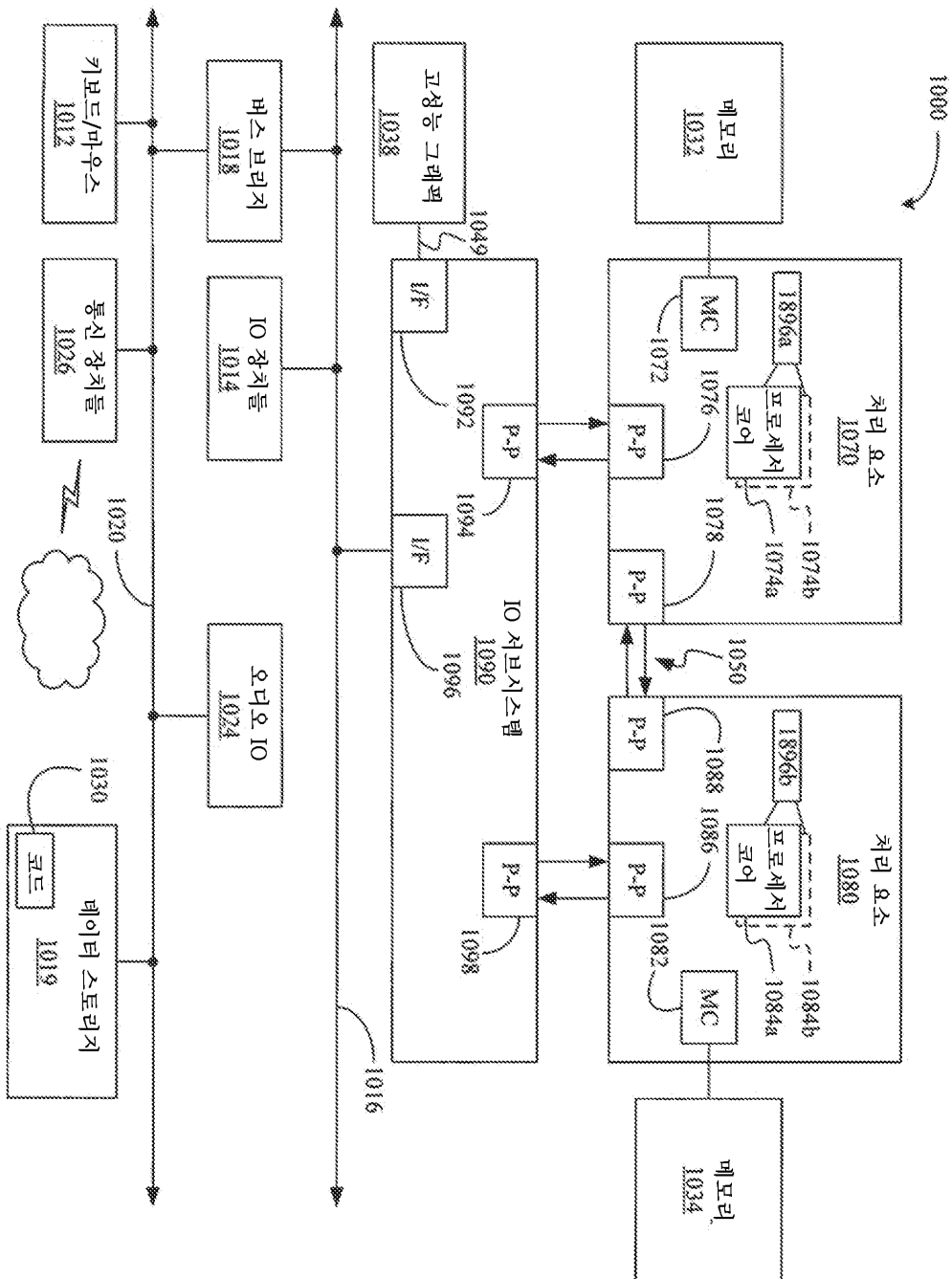
도면3d



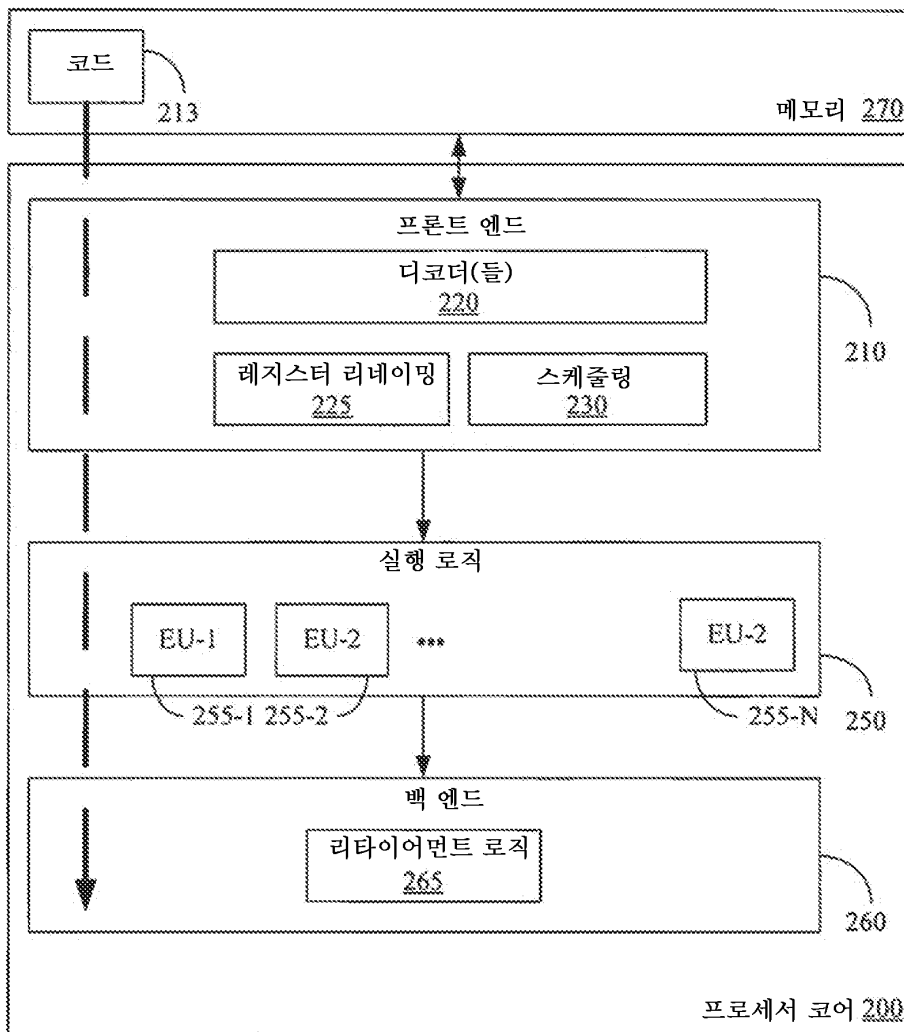
도면3e



도면4



도면5



【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 청구항 19

【변경전】

상기 조정된 국소적 스케줄이 상기 성능 레벨을 달성하는지를

【변경후】

조정된 국소적 스케줄이 상기 성능 레벨을 달성하는지를

【직권보정 2】

【보정항목】 청구범위

【보정세부항목】 청구항 11

【변경전】

상기 프로세서에 의해 실행될 때

【변경후】

프로세서에 의해 실행될 때