(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2015-0112017
(43) 공개일자 2015년10월06일

(51) 국제특허분류(Int. Cl.)
G06F 9/30 (2006.01)
(52) CPC특허분류
G06F 9/30 (2013.01)
(21) 출원번호 10-2015-7023380
(22) 출원일자(국제) 2014년01월28일
심사청구일자 없음
(85) 번역문제출일자 2015년08월27일
(86) 국제출원번호 PCT/US2014/013455
(87) 국제공개번호 WO 2014/120690
국제공개일자 2014년08월07일
(30) 우선권주장
13/753,098 2013년01월29일 미국(US)

(71) 출원인
어드밴스드 마이크로 디바이시즈, 인코포레이티드
미국 캘리포니아 94088-3453 서니베일 피.오.박스 3453 원 에이엠디 플레이스
(72) 발명자
야즈다니 레자
미국 캘리포니아 94024 로스 알토스 매츠 코트 911
(74) 대리인
박장원

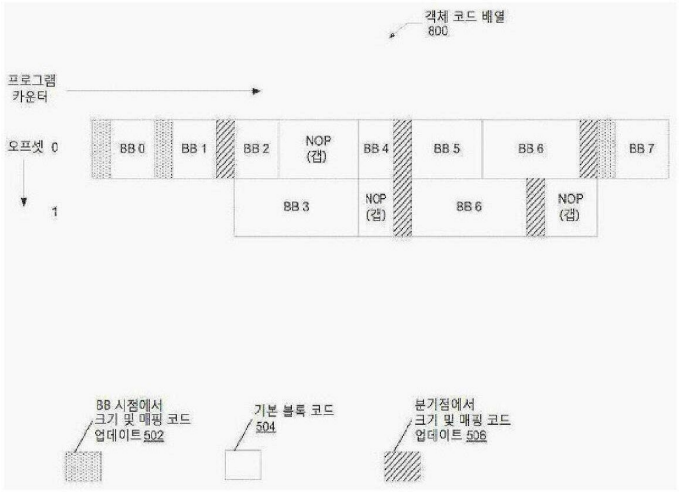
전체 청구항 수 : 총 20 항

(54) 발명의 명칭 **병렬 파이프라인에서의 분기에 대한 하드웨어 및 소프트웨어 해법**

(57) 요 약

프로세서 내의 하드웨어 병렬 실행 레인에서 명령어의 효율적 처리를 위한 시스템 및 방법이 개시된다. 식별되는 루프 내 주어진 분기점에 응답하여, 컴파일러는, 식별되는 루프 내 명령어들을 VLIW(very large instruction world)로 배열한다. 적어도 하나의 VLIW는 주어진 분기점과, 대응하는 집중점 사이에서 서로 다른 기본 블록으로부터 섞인 명령어들을 가리킨다. 컴파일러는 실행될 때, 주어진 VLIW 내의 명령어를 런타임 시에 목표 프로세서 내의 복수의 병렬 실행 레인에 할당하는 코드를 발생시킨다. 목표 프로세서는 SIMD(single instruction multiple word) 마이크로구조를 포함한다. 주어진 레인에 대한 할당은 주어진 분기점에서 주어진 레인에 대해 런타임 시에 발견되는 분기 방향에 기초한다. 목표 프로세서는 연관된 레인이 실행할, 인출되는 VLIW 내의 주어진 명령어를 표시하는 표시사항을 저장하기 위한 벡터 레지스터를 포함한다.

대 표 도



명 세 서

청구범위

청구항 1

컴퓨터 시스템의 적어도 하나의 프로세서에 의해 실행되도록 구성되는 적어도 하나의 프로그램을 저장하는 비-일시적 컴퓨터 판독가능 저장 매체에 있어서, 상기 적어도 하나의 프로그램은 명령어를 포함하고, 상기 명령어는,

복수의 프로그램 명령어 내의 루프 및 대응하는 기본 블록을 식별하도록 실행가능한 명령어와,

식별되는 루프 내의 주어진 분기점에 응답하여, 상기 식별되는 루프 내의 복수의 명령어를 복수의 VLIW(very large instruction words)로 배열하도록 실행가능한 명령어 - 상기 적어도 하나의 VLIW는 상기 주어진 분기점과 대응하는 집중점 사이에서 서로 다른 기본 블록으로부터 섞인 명령어들을 포함함 - 를 포함하는,

비-일시적 컴퓨터 판독가능 저장 매체.

청구항 2

제 1 항에 있어서, 상기 식별되는 루프 내의 상기 주어진 분기점에 응답하여, 상기 SIMD(single instruction multiple data) 마이크로구조를 포함하는 목표 프로세서 내 복수의 병렬 실행 레인에, 주어진 VLIW 내 명령어를 런타임 시에 할당하도록 실행가능한 명령어를 더 포함하는

비-일시적 컴퓨터 판독가능 저장 매체.

청구항 3

제 2 항에 있어서, 주어진 레인을 위한 할당은, 상기 주어진 분기점에서 상기 주어진 레인에 대해 런타임 시에 발견되는 분기 방향에 기초하는,

비-일시적 컴퓨터 판독가능 저장 매체.

청구항 4

제 2 항에 있어서, 다음 프로그램 카운터(PC)에 대응하는 VLIW의 저장된 크기를 업데이트하도록 실행가능한 명령어를 더 포함하는

비-일시적 컴퓨터 판독가능 저장 매체.

청구항 5

제 4 항에 있어서, 검출되는 주어진 분기점과, 대응하는 집중점 사이의 제 2 트레이스 경로보다 작은 제 1 트레이스 경로에 응답하여, VLIW의 제 2 트레이스 경로에 대응하는 명령어와 함께 nop를 그룹형성하도록 실행가능한 명령어를 더 포함하는,

비-일시적 컴퓨터 판독가능 저장 매체.

청구항 6

제 4 항에 있어서, 상기 주어진 VLIW 내의 명령어를 런타임시에 상기 복수의 병렬 실행 레인에 할당하기 위해, 상기 복수의 실행 레인 중 대응 레인과 연관된 벡터 레지스터 내의 특정 비트 범위로 오프셋을 기록하도록 실행가능한 명령어 - 상기 오프셋은 실행할 연관된 레인에 대한 인출되는 주어진 VLIW 내의 주어진 명령어를 식별함 - 를 더 포함하는

비-일시적 컴퓨터 판독가능 저장 매체.

청구항 7

제 6 항에 있어서,

주어진 트레이스가 식별되는 루프의 종점에 도달함에 응답하여,

런타임 시에 상기 주어진 트레이스가 상기 식별되는 루프의 시점으로 다시 분기되도록 스케줄링됨을 결정함에 따라, 상기 벡터 레지스터 내 대응하는 비트 범위의 슬립 상태를 기록하도록 실행가능한 명령어와,

런타임 시에 상기 주어진 트레이스가 상기 식별되는 루프로부터 분기하여 나오도록 스케줄링됨을 결정함에 따라, 상기 벡터 레지스터 내 대응하는 비트 범위의 엑시트 상태를 기록하도록 실행가능한 명령어를 더 포함하는

비-일시적 컴퓨터 판독가능 저장 매체.

청구항 8

제 7 항에 있어서, 상기 주어진 트레이스가 슬립 상태 또는 엑시트 상태에 있음에 응답하여,

주어진 트레이스의 실행을 중지하도록 실행가능한 명령어와,

상기 주어진 트레이스에 대해 적어도 다음 프로그램 카운터(PC) 및 작업 아이템 식별자(ID)를 저장하도록 실행가능한 명령어를 더 포함하는

비-일시적 컴퓨터 판독가능 저장 매체.

청구항 9

제 8 항에 있어서, 각각의 작업 아이템이 슬립 상태에 있거나 각각의 작업 아이템이 엑시트 상태에 있음에 응답하여, 각자 저장된 다음 PC에서의 각각의 작업 아이템에 대한 실행을 재개하도록 실행가능한 명령어를 더 포함하는

비-일시적 컴퓨터 판독가능 저장 매체.

청구항 10

제 8 항에 있어서, 각각의 작업 아이템이 중단되고 적어도 하나의 작업 아이템이 다른 작업 아이템과 다른 상태에 있음에 응답하여, 각자 저장된 다음 PC에서의 슬립 상태의 작업 아이템에만 실행을 재개하도록 실행가능한 명령어를 더 포함하는

비-일시적 컴퓨터 판독가능 저장 매체.

청구항 11

SIMD(single instruction multiple data) 마이크로구조 내의 복수의 병렬 실행 레인과,

VLIW(very large instruction word)의 크기를 저장하도록 구성되는 사이즈 레지스터와,

저장되는 크기와 동일한 주어진 VLIW 내의 다수의 명령어를 각자의 사이클로 인출 및 디코딩하도록 구성되는 제어 로직을 포함하며,

상기 복수의 실행 레인은 상기 주어진 VLIW 내 다수의 명령어들을 동시에 실행하도록 구성되는, 프로세서.

청구항 12

제 11 항에 있어서,

상기 프로세서는 상기 복수의 실행 레인 중 대응 레인과 연관된 특정 비트 범위의 오프셋을 저장하도록 구성되는 벡터 레지스터를 더 포함하고, 상기 오프셋은 연관된 레인이 실행할, 인출되는 주어진 VLIW 내의 주어진 명령어를 식별하는

프로세서.

청구항 13

제 12 항에 있어서, 상기 벡터 레지스터에 저장되는 다수의 유효 오프셋은 상기 사이즈 레지스터에 저장되는 크기와 동일한, 프로세서.

청구항 14

제 12 항에 있어서,

VLIW 내의 복수의 리소스-독립적 명령어에 대응하는 오프셋을 검출함에 응답하여, 상기 복수의 실행 레인은, 연관된 레인 내의 복수의 명령어를 동시에 실행하도록 또한 구성되는, 프로세서.

청구항 15

제 12 항에 있어서, 주어진 트레이스가 식별되는 루프의 종점에 도달하였고 주어진 트레이스를 표시하는 벡터 레지스터 내 대응하는 비트 범위가 슬립 상태 또는 엑시트 상태에 있음에 응답하여, 상기 제어 로직은,

상기 주어진 트레이스의 실행을 중단시키도록 또한 구성되고,

상기 주어진 트레이스에 대한 다음 프로그램 카운터(PC) 및 레인 식별자(ID)를 적어도 저장하도록 또한 구성되는, 프로세서.

청구항 16

제 15 항에 있어서, 상기 복수의 실행 레인 중 각각의 레인이 슬립 상태 또는 엑시트 상태에 있음에 응답하여, 상기 제어 로직은 각자의 저장된 다음 PC로 분기함으로써 각각의 레인에 대한 실행을 재개하도록 또한 구성되는, 프로세서.

청구항 17

제 15 항에 있어서, 상기 복수의 실행 레인 중 각각의 레인이 슬립 상태 또는 엑시트 상태에 있음에 응답하여, 상기 제어 로직은 각자의 저장된 다음 PC로 분기함으로써 슬립 상태의 레인의 경우에만 실행을 재개하도록 또한 구성되는, 프로세서.

청구항 18

복수의 프로그램 명령어 내의 루프 및 대응하는 기본 블록을 식별하는 단계와,

식별되는 루프 내의 주어진 분기점에 응답하여, 식별되는 루프 내의 복수의 명령어를 복수의 VLIW(very large instruction words)로 배열하는 단계 - 적어도 하나의 VLIW는 상기 주어진 분기점과 대응 집중점 사이에서 서로 다른 기본 블록으로부터 섞인 명령어를 포함함 - 를 포함하는

방법.

청구항 19

제 17 항에 있어서, 식별되는 루프 내의 주어진 분기점에 응답하여, 상기 방법은, 상기 주어진 분기점의 주어진 레인에 대해 런타임 시에 발견되는 분기 방향에 기초하여, SIMD(single instruction multiple data) 마이크로 구조를 포함하는 목표 프로세서 내 복수의 병렬 실행 레인에, 주어진 VLIW 내의 명령어를 런타임 시에 할당하는 단계를 더 포함하는, 방법.

청구항 20

제 19 항에 있어서, 상기 주어진 VLIW 내 명령어를 런타임 시에 상기 복수의 병렬 실행 레인에 할당하기 위하여, 상기 방법은, 상기 복수의 실행 레인 중 대응 레인과 연관된 표시사항을 저장하는 단계를 더 포함하고, 상기 표시사항은 연관된 레인이 실행할, 상기 주어진 VLIW 내의 주어진 명령어를 식별하는, 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 컴퓨팅 시스템에 관한 것으로서, 특히, 프로세서 내에 하드웨어 병렬 실행 레인에서 효율적 명령어 처리에 관한 것이다.

배 경 기 술

[0002] 작업의 병렬화를 이용하여, 컴퓨터 시스템의 처리량을 높일 수 있다. 이를 위해, 컴파일러는 시스템 하드웨어 상에서 병렬로 실행될 프로그램 코드로부터 병렬화된 작업을 추출할 수 있다. 하드웨어 상의 병렬 실행을 증가시키기 위해, 프로세서는 SIMD(single instruction multiple word) 마이크로구조(microarchitecture)와 같은, 복수의 병렬 실행 레인을 포함할 수 있다. 이러한 타입의 마이크로구조는 단일-레인 마이크로구조 또는 범용 마이크로구조보다 특정 소프트웨어 애플리케이션용으로 높은 명령어 처리량을 제공할 수 있다. SIMD 마이크로구조로부터의 장점을 가진 일부 예의 작업들은 비디오 그래픽 렌더링, 암호 기법, 및 가비지 컬렉션을 포함한다.

[0003] 많은 경우에, 특정 소프트웨어 애플리케이션은, 각각의 작업 아이템의 실행 또는 병렬 함수 콜이 자체 내의 데이터 의존적인, 데이터 병렬화를 가진다. 예를 들어, 제 1 작업 아이템은 제 2 작업 아이템으로부터 데이터 독립적일 수 있고, 제 1 및 제 2 작업 아이템 각각은 SIMD 마이크로구조 내의 별도의 병렬 실행 레인 상에 동시에 스케쥴링된다. 그러나, 제 1 및 제 2 작업 아이템 각각 내에서 실행되는 소정 양의 명령어들이 데이터 의존적일 수 있다. 분기 명령어로 구현되는 조건부 검사는 제 1 작업 아이템에 대해 통과될 수 있으나, 각각의 작업 아이템의 데이터에 대해 의존적인 제 2 작업 아이템의 경우 실패할 수 있다.

[0004] 병렬 실행 효율은, 제 2 작업 아이템이 실행 정지됨에 따라 그리고 제 1 작업 아이템이 진행중인 실행을 계속함에 따라, 감소될 수 있다. 검사 통과로 인해 몇개의 작업 아이템만이 실행을 계속하고 대부분의 작업 아이템은 검사 불합격으로 인해 아이들 상태일 때 비효율성이 증가한다.

발명의 내용

과제의 해결 수단

[0005] 프로세서 내의 하드웨어 병렬 실행 레인에서 명령어의 효율적 처리를 위한 시스템 및 방법이 고려된다. 다양한 실시예에서, 백엔드 컴파일러는 명령어를 배열하도록, 그리고, 목표 프로세서 상에서 명령어를 효율적으로 처리하기 위한 코드를 발생시키도록, 소프트웨어 애플리케이션의 프로그램 명령어를 검사한다. 목표 프로세서는 SIMD(single instruction multiple word) 마이크로구조 내에 복수의 병렬 실행 레인을 포함한다. 컴파일러는 루프 및 대응하는 기본 블록을 식별할 수 있다. 루프 내의 분기점은 분기 명령어를 포함할 수 있다. 예를 들어, if-else-else 구조, if-else 구조, case 구조, 등이, 식별되는 루프 내 프로그램 명령어에 사용될 수 있다. 분기점(divergent point)과 대응하는 집중점(convergent point) 사이에서 변환된 그리고 컴파일된 프로그램 명령어의 실행 중, 복수의 트레이스 경로를 거칠 수 있다.

[0006] 컴파일 중, 식별된 루프 내 주어진 분기점에 응답하여, 컴파일러는 식별된 루프 내의 명령어들을 하나 이상의 VLIW 내에 배열할 수 있다. 적어도 하나의 VLIW는 주어진 분기점과, 대응하는 집중점 사이에서 서로 다른 기본 블록으로부터 섞인 명령어들을 가리킬 수 있다. 예를 들어, 주어진 분기점과 대응하는 집중점 사이에 4개의 명령어를 가진 기본 블록 A와, 6개의 명령어를 가진 기본 블록 B가 존재할 경우, 컴파일러는 명령어를 6개의 VLIW 내로 배열할 수 있다. 첫번째 4개의 VLIW는 기본 블록 A 및 기본 블록 B 각각으로부터 제 1 명령어를 포함할 수 있다. 제 1 VLIW는 기본 블록 A 및 기본 블록 B 각각으로부터 제 1 명령어를 포함할 수 있다. 제 2 VLIW는 기본 블록 A 및 기본 블록 B 각각으로부터 제 2 명령어를 포함할 수 있으며, 등등이다. 마지막 2개의 VLIW는 nop로 그룹형성된 기본 블록 B로부터의 명령어를 포함할 수 있다. 컴파일러는 각각의 VLIW를 가리키는 프로그램 카운터(PC) 값을 추적할 수 있다.

[0007] 컴파일러는 변환 및 컴파일된 프로그램 명령어와 함께 삽입할 코드를 발생시킬 수 있다. 삽입되는 코드는, 실행될 때, 주어진 VLIW 내의 명령어들을 런타임 시에 목표 프로세서 내 복수의 병렬 실행 레인에 할당할 수 있다. 주어진 레인에 대한 할당은 주어진 분기점에서 주어진 레인에 대해 런타임시에 발견되는 분기 방향에 기초할 수 있다. 위 예를 계속하면, VLIW가 기본 블록 A 및 기본 블록 B로부터 생성된 제 2 VLIW이고 주어진 레인에 대한 분기 명령어가 선택되면, 주어진 레인은 제 2 VLIW 내 기본 블록 A 내 제 2 명령어를 할당받을 수 있다. 분기 명령어가 선택되지 않을 경우, 주어진 레인은 제 2 VLIW 내 기본 블록 B 내 제 2 명령어를 할당받을 수 있다. 다양한 실시예에서, VLIW는 가변 길이를 가진다. 삽입되는 코드는, 실행될 때, 다음 PC에 대응하는 VLIW의 크기를 업데이트할 수 있다.

[0008] 일부 실시예에서, 프로세서는 SIMD 마이크로구조 내에 복수의 병렬 실행 레인을 포함한다. 프로세서는 가변 길이 VLIW의 크기를 저장하기 위한 사이즈 레지스터를 포함할 수 있다. 프로세서 내의 제어 로직은 저장된 크기와 동일한 주어진 VLIW 내의 다수의 명령어를 각자의 사이클로 인출 및 디코딩할 수 있다. 복수의 실행 레인은 주어진 VLIW 내 다수의 명령어들을 동시에 실행할 수 있다. 프로세서는 복수의 실행 레인 중 대응 레인들과 연관된 비트 범위를 가진 벡터 레지스터를 추가로 포함할 수 있다. 이러한 비트 범위는 오프셋을 저장할 수 있다.

주어진 오프셋은 연관된 레인이 실행할, 인출되는 VLIW 내의 주어진 명령어를 식별할 수 있다.

[0009] 이러한 실시예 및 다른 실시예들은 다음 설명 및 도면을 참조하여 더 잘 이해될 것이다.

도면의 간단한 설명

[0010] 도 1은 SIMD(single instruction multiple word) 파이프라인 실행 흐름의 일 실시예의 일반화된 블록도다.

도 2는 제어 흐름 그래프의 일 실시예의 일반화된 블록도다.

도 3은 제어 흐름 그래프의 실행 시퀀스의 일 실시예의 일반화된 블록도다.

도 4는 프로세서용 SIMD 마이크로구조의 로직 레이아웃의 일 실시예의 일반화된 블록도다.

도 5는 객체 코드 배열의 일 실시예의 일반화된 블록도다.

도 6은 컴파일러 기술을 이용하여 프로세서 내 복수 작업 아이템들의 병렬 실행을 최적화하기 위한 방법의 일 실시예의 일반화된 흐름도다.

도 7은 하드웨어 기술을 이용하여 프로세서 내 복수 작업 아이템들의 병렬 실행을 최적화하기 위한 방법의 일 실시예의 일반화된 흐름도다.

도 8은 객체 코드 배열의 다른 실시예의 일반화된 블록도다.

실시예들이 다양한 변형예 및 대안의 형태를 취할 수 있으나, 구체적인 실시예가 도면에서 예를 들어 도시되며, 여기서 세부적으로 설명된다. 그러나, 도면 및 도면에 대한 상세한 설명은 개시되는 특정 형태로 실시예를 제한하고자 하는 것이 아니며, 반대로, 실시예는 첨부 청구범위에 의해 규정되는 실시예의 사상 및 범위 내에 있는 모든 변형예, 등가물, 및 대안들을 커버한다.

발명을 실시하기 위한 구체적인 내용

[0011] 다음의 설명에서, 실시예의 완전한 이해를 제공하기 위해 수많은 구체적 세부사항들이 제시된다. 그러나, 당 업자는 실시예들이 이러한 구체적 세부사항없이 실시될 수 있음을 인식할 것이다. 일부 예에서, 잘 알려진 회로, 구조, 및 기술들은 실시예의 본질을 흐리지 않기 위해 세부적으로 도시되지 않았다.

[0012] 도 1을 참조하면, SIMD(single instruction multiple word) 파이프라인 실행 흐름(100)의 일 실시예를 예시하는 일반화된 블록도가 도시된다. 명령어(102-108)가 인출되어 연관 데이터와 함께 SIMD 파이프라인에 전달된다. 병렬 수직 실행 레인 내 복수 연산 유닛이 도시된다. 연산 유닛 중 일부는 액티브 연산 유닛(110)이다. 다른 연산 유닛은 주어진 파이프 스테이지 동안 작동정지됨으로 인한 인액티브 연산 유닛(112)이다. 제어 로직 및 저장 요소, 가령, 파이프라인 레지스터는 설명을 쉽게 하기 위해 도시되지 않는다.

[0013] 하드웨어 연산 유닛은 연관 데이터와 함께 주어진 작업 아이템의 주어진 명령어의 실행을 수행하는 하드웨어를 포함한다. 이러한 하드웨어는 가산, 승산, 제로 결함, 비트-와이즈 시프트, 나누기, 비디오 그래픽, 및 멀티미디어 명령어, 또는, 프로세서 설계 분야의 당 업자에게 알려진 그외 다른 작동을 수행하도록 구성되는 산술 로직 유닛을 포함할 수 있다. SIMD 파이프라인 내의 병렬 실행 레인을 가진 프로세서의 예는, 그래픽 프로세싱 유닛(GPU), 디지털 신호 프로세싱(DSP), 등을 포함한다. 일 실시예에서, SIMD 파이프라인은 비디오 카드 상에 위치할 수 있다. 다른 실시예에서, SIMD 파이프라인은 마더보드 상에 통합될 수 있다.

[0014] SIMD 파이프라인은 게임, 엔터테인먼트, 과학, 및 의료 분야에 사용되는 폭넓고 다양한 데이터-병렬 애플리케이션의 컴퓨팅 성능을 개선시킬 수 있다. 이러한 애플리케이션들은 일반적으로, 많은 수의 객체 상에서 동일 프로그램의 실행을 수반한다. 각각의 객체가 다른 객체에 독립적으로 처리되지만 동일한 시퀀스의 작동들이 사용되기 때문에, SIMD 마이크로구조는 상당한 성능 개선을 제공한다. GPU는 논-그래픽 연산을 위해 또한 고려되고 있다.

[0015] 소프트웨어 애플리케이션은 함수 콜, 또는 컴퓨트 커널과, 내부 함수의 집합을 포함할 수 있다. 소프트웨어 프로그래머는 함수 콜을 규정할 수 있고, 반면에 내부 함수는 주어진 라이브러리에서 규정될 수 있다. 예를 들어, 소프트웨어 애플리케이션은 이미지 파일과 같은, 2차원(2D) 어레이의 데이터에 대한 데이터 프로세싱을 수행할 수 있다. 소프트웨어 애플리케이션은 2D 이미지의 화소 단위로, 또는, 2차원 매트릭스의 요소 단위로 소프트웨어 프로그래머에 의해 개발된 알고리즘을 수행할 수 있다. 주어진 함수 콜은 인덱스 공간을 통해 호출될 수 있다. 인덱스 공간은 차원 공간으로도 불릴 수 있다. 데이터-병렬 소프트웨어 애플리케이션의 경우에, N-차원 연

산 도메인은 1차원, 2차원, 또는 3차원 공간 또는 인덱스 공간을 형성할 수 있다. 한 예는 2D 이미지 내의 화소들이다.

[0016] 함수 콜은 연산의 하나 이상의 작업 아이템을 생성하기 위해 데이터의 하나 이상의 레코드와 매칭될 수 있다. 따라서, 2개 이상의 작업 아이템이 단일 함수 콜의 동일 명령어를 이용할 수 있으나, 서로 다른 레코드의 데이터에 대해 작동한다. 함수 콜은 포크(fork)를 생성하는 제어 흐름 전달 명령어를 포함할 수 있으나, 컴퓨터 시스템의 포크는 공통 규정에 의해, 소프트웨어 스레드(software threads)를 통상적으로 생성한다. 인덱스 공간 내 주어진 지점에서 함수 콜의 주어진 인스턴스는 "작업 아이템"으로 불릴 수 있다. 작업 아이템은 작업 유닛으로도 불릴 수 있다. 위 예를 계속 진행하면, 작업 아이템은 2D 이미지의 주어진 화소(주어진 인덱스)에 대응하는 데이터의 레코드에 대한 함수 콜의 하나 이상의 명령어로 작동할 수 있다. 통상적으로, 작업 아이템은 관련 고유 식별자(ID)를 가진다.

[0017] 인덱스 공간은 하드웨어 지원이 충분할 경우, 병렬로 실행되는 작업 아이템의 총 개수를 규정할 수 있다. 예를 들어, 인덱스 공간은 280개의 작업 아이템을 규정할 수 있으나, GPU는 임의의 주어진 시간에 64개의 작업 아이템의 동시 실행을 지원할 수 있다. 작업 아이템의 총 수는 전역 작업 크기를 규정할 수 있다. 작업 아이템은 작업 그룹으로 더 그룹화될 수 있다. 각각의 작업 그룹은 고유 식별자(ID)를 가질 수 있다. 주어진 작업 그룹 내의 작업 아이템은 서로 통신할 수 있고, 실행을 동기화할 수 있으며, 메모리 액세스를 조율할 수 있다. 다수의 작업 아이템이 SIMD 방식으로 GPU 상에서 동시적 실행을 위해 하나의 웨이브 프론트로 클러스터화될 수 있다. 총 280개의 작업 아이템에 대한 위 예와 관련하여, 웨이브 프론트는 64개의 작업 아이템을 포함할 수 있다.

[0018] 명령어(102-108)가 인출되어 연관 데이터와 함께 SIMD 파이프라인에 들어간다. 명령어(104)는 조건부 분기와 같은, 제어 흐름 전달 명령어일 수 있다. 명령어(106)는 조건이 참일 때 실행되는 경로의 제 1 명령어일 수 있다. 명령어(108)는 조건이 거짓일 때 실행되는 경로 내 제 1 명령어일 수 있다. 예를 들어, 분기 명령어(104)는 하이 레벨 언어 프로그램의 IF 문과 연관될 수 있다. 명령어(106)는 하이 레벨 언어 프로그램의 THEN 문과 연관될 수 있다. 명령어(108)는 하이 레벨 언어 프로그램의 ELSE 문과 연관될 수 있다.

[0019] 주어진 로우(row) 내 각각의 연산 유닛은 동일 연산 유닛일 수 있다. 이러한 연산 유닛 각각은 동일 명령어 상에서 작동할 수 있으나, 다른 작업 아이템과 연관된 다른 데이터 상에서 작동할 수 있다. 도시되는 바와 같이, 작업 아이템 중 일부는 조건부 분기 명령어(104)에 의해 제공되는 검사에 합격할 수 있고, 다른 작업 아이템은 검사에 불합격할 수 있다. SIMD 파이프라인 내의 제어 로직은 각각의 가용 경로를 실행할 수 있고, 현 경로를 선택하지 않은 작업 아이템에 대응하는, 연산 유닛과 같은, 실행 유닛을 선택적으로 작동정지시킬 수 있다. 예를 들어, IF-THEN-ELSE 구성문의 실행 중, SIMD 아키텍처의 각각의 칼럼 내에, "Then"(경로 A) 및 "Else"(경로 B) 경로를 실행하도록 구성되는 실행 유닛이 위치한다.

[0020] 병렬 실행 효율은, 제 1 및 제 2 작업 아이템의 실행이 일시정지되어 제 3 작업 아이템이 그 진행중인 실행을 계속할 것을 기다림에 따라 저하될 수 있다. 따라서, 분기 명령어(104) 실행 후 주어진 로우 내 모든 연산 유닛이 액티브 연산 유닛(110)인 것은 아니다. 도시되는 바와 같이, 하나 이상의 연산 유닛은 실행 중지된 인액티브 연산 유닛(112)이다. 다수의 연산 유닛이 주어진 파이프 스테이지 중 인액티브 상태일 경우, SIMD 코어의 효율 및 처리량이 저하된다. 일 실시예에서, "Else" 경로는 함수 콜에 대한 리턴이다. 함수 콜의 실행이 종료되고 대응하는 작업 아이템이 아이들 상태가 된다. 그러나, SIMD 코어 내 이웃 작업 아이템들은 실행을 계속할 수 있다.

[0021] 이제 도 2를 살펴보면, 제어 흐름 그래프(200)의 일 실시예를 예시하는 일반화된 블록도가 도시된다. 일반적으로 말해서, 제어 흐름 그래프는 컴파일러 옵티마이저 및 정적 분석 툴에 의해 사용될 수 있다. 제어 흐름 그래프(200)는 실행 중 프로그램 또는 프로그램의 일부분을 통해 가로지를 수 있는 모든 경로를 나타낼 수 있다. 제어 흐름 그래프에서, 그래프 내 각각의 노드는 기본 블록을 나타낸다. 대부분의 표현은 입구 블록 및 출구 블록을 포함하며, 입구 블록을 통해 제어가 제어 흐름 그래프 내로 들어가고, 출구 블록을 통해 제어가 제어 흐름 그래프를 빠져나온다.

[0022] 컴파일 중, 소프트웨어 애플리케이션은 기본 블록 0(BB 0)으로부터 기본 블록 7(BB 7)까지 번호를 갖는 8개의 기본 블록(BB)을 제어 흐름 그래프(200)에 제공할 수 있다. 8개의 기본 블록이 도시되지만, 다른 예에서, 다른 개수의 기본 블록이 사용될 수 있다. 제어 흐름 그래프(200)의 경우, 기본 블록(1)은 입구 블록이고 기본 블록(6)은 출구 블록이다. 기본 블록(0-7) 각각은 하나의 입구점 및 하나의 출구점을 가진 명령어들의 직선 시퀀스다. 제어 흐름 그래프(200)는 루프를 나타낼 수 있다. 루프 내에서, 제어 흐름 그래프(200)는 기본 블록(1-4)을 가진 IF-THEN-ELSE 구성과, 기본 블록(4-6)을 가진 IF 구성을 나타낼 수 있다.

[0023] 이제 도 3을 참조하면, 제어 흐름 그래프의 실행 시퀀스(300)의 일 실시예를 예시하는 일반화된 블록도가 도시된다. 실행 시퀀스(310)는 SIMD 파이프라인에 할당될 때 앞서 도시된 제어 흐름 그래프(200)에 대한 전형적인 실행 시퀀스를 나타낸다. 단일 루프 반복의 실행 시간은 BB 1 으로부터 BB 6 까지와 같이, 루프 내 각각의 기본 블록의 실행 시간의 합이다. 그러나, 주어진 작업 아이템 및 대응하는 하드웨어 실행 레인의 경우에, BB 2 및 BB 3 중 하나만이 실행된다. 마찬가지로, BB5는 주어진 작업 아이템에 대해 건너뛸 수 있다. 특정 기본 블록들이 주어진 작업 아이템에 대해 실행되지 않을 수 있지만, 관련 실행 시간은 루프 반복을 위한 실행 시간의 총합에 여전히 기여한다.

[0024] 실행 시퀀스(320)는 변형된 SIMD 파이프라인에 할당될 때 앞서 도시된 제어 흐름 그래프(200)에 대한 대안의 실행 시퀀스를 나타낸다. 단일 루프 반복의 실행 시간은 루프 내 기본 블록들 각각의 실행 시간의 합보다는, 단일 루프 반복에서 실제 실행되는 기본 블록들의 실행 시간의 합이다. 실행 시퀀스(320)는 제어 흐름 그래프(200)의 실행을 변환한다. 컴파일러는 소스 코드를 객체 코드로 컴파일 중 이러한 변환을 수행할 수 있다.

[0025] 일부 실시예에서, 컴파일러는 실행될 때 실행할 다음 기본 블록을 식별하도록, 각각의 기본 블록의 종료시 코드를 생성한다. 생성 코드는 기본 블록의 종점에 삽입될 수 있다. 대안으로서, 분기 명령어가 기본 블록의 종료시 삽입되어, 다음 기본 블록으로 제어를 넘기기 전에 제어 흐름을 추가 코드에 전달할 수 있다. 추가 코드는 중앙 기본 블록을 의미하는 BB C로 표현된다. 실행될 때, 각각의 분기 기본 블록, 가령, BB 1, BB 4, BB 6 는 제어를 전달할 다음 기본 블록을 식별한다. 식별은 분기 해상력에 기초하며, 이는 특정 레코드의 데이터 및 분기 명령어에 또한 기초한다. 실행될 때, BBC는 각각의 작업 아이템으로부터 목표 기본 블록의 어드레스를 수신하고, 각자의 목표 기본 블록 실행을 위한 스레드 레지스터를 셋업한다. SIMD 파이프라인에서 실행되고 있는 서로 다른 작업 아이템 간에, 주어진 작업 아이템은 분기, 점프, 및 케이스 문과 같이, 제어 흐름 전달 명령어를 위한 단일 목표를 가진다.

[0026] 실행 시퀀스(320)에서, BB 4의 완료시, 제 1 작업 아이템이 BB 5로 분기할 수 있고 대응 어드레스를 BB C에 전달할 수 있다. BB 4 완료시, 제 2 작업 아이템이 BB 6로 분기할 수 있고, 대응하는 어드레스를 BB C로 전달할 수 있다. 컴파일러는 동시에 실행될, 목표 기본 블록 각각으로부터의 명령어를 포함하는 DVLIW(dynamic very large instruction word)를 생성할 수 있다. 실행될 때, 컴파일러에 의해 생성된 코드는 다음 프로그램 카운터(PC) 값에 대응하여 실행될 다음 DVLIW의 크기를 업데이트할 수 있다. 추가적으로, 실행될 때, 생성된 코드는 주어진 병렬 실행 레인에서 실행되는 주어진 작업 아이템과, 실행할 다음 DVLIW 내의 명령어에 대한 포인터 간의 매핑을 업데이트할 수 있다. 포인터는 인출될 다음 DVLIW 내의 명령어들 중 연관된 명령어를 식별하는 오프셋일 수 있다.

[0027] 도 4를 참조하면, 프로세서용 SIMD 마이크로구조의 로직 레이아웃(400)의 일 실시예를 예시하는 일반화된 블록도가 도시된다. 프로세서는 데이터 및 명령어의 저장을 위해 동적 랜덤 액세스 메모리(DRAM)(450)를 가진다. 일부 실시예에서, 주어진 레벨의 캐시 메모리 서브시스템이 DRAM에 추가하여 사용된다. 도시되는 바와 같이, 프로세서는 연산 유닛들의 각각의 로우(row)에 대해 제어 로직(420)과 함께 그룹화된 비교적 작은 캐시 메모리 서브시스템(430)을 가질 수 있다. 프로세서 내의 데이터 흐름은 파이프라인화될 수 있지만, 파이프라인 레지스터와 같은 저장 요소들이 설명의 단순화를 위해 도시되지 않는다. 주어진 파이프라인 스테이지에서, 연산 유닛은 이러한 스테이지 내 관련 명령어가, 선택되지 않은 분기와 같이, 앞서 불합격된 검사에 기초하여 실행되지 않을 경우, 사용되지 않을 수 있다.

[0028] SIMD 파이프라인은 레인 A-F을 포함하는 작업 아이템(460)을 포함한다. 각각의 레인 A-F은 연산 유닛들을 포함하는 수직 병렬 하드웨어 레인들 중 각자에 대응할 수 있다. 파이프라인은 벡터 레지스터(462)를 추가로 포함할 수 있다. 벡터 레지스터(462)는 병렬 실행 레인 각각에 대한 엔트리, 필드, 또는 비트 범위를 포함할 수 있다. 각각의 엔트리는 각자의 작업 아이템 상에서 실행되는 주어진 트레이스의 식별을 위한 제 1 비트 수와, 특별 코드 지원을 위한 제 2 비트 수를 포함하는 총 비트 수를 포함할 수 있다. 특별 코드는 대기 또는 슬립 상태, 루프 탈출 상태, 루프 종료와는 다른 실행의 중단을 위한 장벽 식별자, 이벤트 식별자, 등을 식별할 수 있다. 특별한 코드가 주어진 엔트리에 저장되지 않을 때, 저장되는 값은 실행할 연관 레인에 대한 DVLIW 내의 명령어들 중 각자를 식별할 수 있다.

[0029] 프로그램 카운터(PC) 레지스터(466)는 i-cache와 같은 메모리로부터 인출할 다음 DVLIW를 가리키는 포인터 값 또는 어드레스를 저장할 수 있다. 프로세서는 DVLIW의 크기 또는 길이를 저장하는 크기 레지스터(468)를 더 포함할 수 있다. 일부 실시예에서, 크기는 가변 길이 DVLIW 내의 다수의 명령어들을 나타내는 정수일 수 있다.

[0030] DVLIW(464) 내의 명령어 Instr A 내지 Instr G 각각은 제어 흐름 그래프에서 실행 트레이스를 나타난다. 컴파일

러는 i-cache와 같이, 메모리 내에 DVLIW를 레이아웃할 수 있다. 한 예에서, 작업 아이템(460) 내의 레인 B는 SIMD 파이프라인 내 좌측으로부터의 제 2 수직 실행 레인에 대응할 수 있다. 벡터 레지스터(462)에 저장되는 "오프셋 B"는 레인 B와 연관될 수 있고, DVLIW(464) 내의 제 1 명령어(Instr A)를 가리킬 수 있다. 따라서, 레인 B는 처리할 Instr A를 수신할 수 있다. 마찬가지로, 작업 아이템(460) 내의 레인 A는 SIMD 파이프라인 내 최좌측 수직 실행 레인에 대응할 수 있다. 벡터 레지스터(462)에 저장된 오프셋 A는 레인 A와 연관될 수 있고, DVLIW(464) 내 최종 명령어(Instr G)를 가리킬 수 있다. 따라서, 레인 A는 처리할 Instr G를 수신할 수 있다.

[0031] 도시되지 않지만, 명령어 캐시(i-cache)는 DVLIW를 지원하기 위한 복수의 구현 중 하나를 포함할 수 있다. i-cache는 DVLIW에 대응하는 주어진 단일 PC를 위한 하나 이상의 명령어를 인출하기 위한 복수의 더 작은 캐시를 포함할 수 있다. 동일한 PC가 DVLIW의 크기에 따라 더 작은 캐시들 중 하나 이상 내의 유효 명령어를 인덱싱할 수 있다. I-cache는 PC 레지스터(466)에 저장된 포인터 또는 어드레스 값에 추가하여 사이즈 레지스터(468)에 저장된 크기를 수신할 수 있다. 대안으로서, 이러한 I-cache는 동일 유효 캐시 라인 또는 캐시 세트 내 하나 이상의 명령어에 액세스하기 위해 복수의 데이터 포트를 가질 수 있다. 다시, 인출할 유효 명령어의 수는 사이즈 레지스터(468)로부터 수신된 크기와 동일할 수 있다.

[0032] 이제 도 5를 살펴보면, 객체 코드 배열(500)을 예시하는 일 실시예의 일반화된 블록도가 도시된다. 코드 배열(500)은 도 2 및 도 3에 각각 도시되는 제어 흐름 그래프(200) 및 동반 실행 시퀀스(320)를 위한, 컴파일러에 의해 생성될 수 있는 객체 코드 레이아웃의 일 실시예를 도시한다. 기본 블록 코드(504)는 각자의 기본 블록을 위한 코드를 나타낸다. 예를 들어, 기본 블록 0-3 및 7의 코드의 일 사본이 레이아웃(500)으로 배열되는 것으로 도시된다. 기본 블록 4-5용 코드의 2개의 사본이 레이아웃(500)으로 배열되는 것으로 도시된다. 기본 블록 6용 코드의 4개의 사본이 레이아웃(500)으로 도시된다.

[0033] 코드(502)는 루프와 같은 영역의 엔트리를 위해 컴파일러에 의해 생성 및 삽입될 수 있다. 나중에 실행될 때, 코드(502)는 다음 DVLIW의 크기를 업데이트할 수 있고, 목표 프로세서 내 병렬 실행 레인과 인출된 DVLIW 내의 명령어 간의 매핑을 업데이트할 수 있다. 예를 들어, 목표 프로세서 내 크기 및 벡터 레지스터는 실행 코드(502)에 의해 업데이트되는 저장된 콘텐츠를 가질 수 있다. 도시되는 바와 같이, 코드(502)는 기본 블록 0, 1, 7의 시점에 삽입될 수 있다.

[0034] 코드(506)는 기본 블록 1, 4, 6의 종점과 같이, 분기되는 지점으로 전이하기 위해, 컴파일러에 의해 생성 및 삽입될 수 있다. 나중에 실행될 때, 코드(506)는 DVLIW의 크기 변화를 결정할 수 있고, 목표 프로세서 내 병렬 실행 레인과 인출된 DVLIW 내의 명령어 간의 대응하는 매핑 변화를 결정할 수 있다. 따라서, 크기 및 매핑은 제어 흐름 그래프 내 분기 지점 및 집중 지점에서 업데이트된다. 컴파일러는 DVLIW의 크기가 변화하는 지점 및 매핑이 변화하는 지점을 식별한다. BBC(0,1) 표기법을 이용 - 제 1 인덱스는 트레이스 식별자(ID)를 표시하고 제 2 인덱스는 기본 블록(BB) ID를 표시함 - 하여, 코드(506) 삽입을 위한 식별점이, BBC (0, 1), BBC (0, 4), BBC (0,6), BBC (1, 4), BBC (1, 6), BBC (2, 6) 및 BBC (3, 6)에 존재할 수 있다. 본 예의 트레이스 ID는 대응하는 오프셋과 동일할 수 있다.

[0035] 객체 코드 배열(500)의 시점에서, 코드(502)는 초기화 단계를 수행하고, DVLIW 크기를 1로 세팅할 수 있다. 벡터 레지스터의 각각의 엔트리 내 오프셋은 0의 오프셋과 같이 BB 0 내의 동일 명령어를 가리키도록 세팅될 수 있다. 따라서, PC 0은 값 0 또는 다른 적절한 시작 어드레스로 세팅될 수 있다. 도시되는 바와 같이, 4개의 병렬 트레이스가 가능하지만, 작업 아이템의 수는 독립적일 수 있다. 예를 들어, SIMD 파이프라인은 할당된 작업 아이템을 처리하기 위한 8, 16, 64, 또는 다른 개수의 병렬 실행 레인을 가질 수 있다. SIMD 파이프라인 내의 각각의 작업 아이템은 벡터 레지스터 내 0의 저장된 오프셋을 갖고, 동일 명령어를 실행할 것이다. 각각의 작업 아이템에 대한 이와 같이 동일한 명령어는 BB 0으로부터의 명령어다. BB 0의 명령어는 각각의 작업 아이템에 의해 일대일로 실행되며, PC는 각각의 명령어 인출 후 증분된다.

[0036] BB 0의 실행 완료 후, 루프 엔트리 블록 BB 1이 다음에 처리될 것이다. BB 1의 시점에서 코드(502)는 DVLIW 크기를 1로 유지하고 각각의 작업 아이템에 대한 오프셋을 0으로 유지한다. 벡터 레지스터의 각각의 엔트리 내 오프셋은 0의 오프셋과 같이 BB 1 내의 동일 명령어를 가리키도록 세팅될 수 있다. PC는 BB 0의 완료시 증분된 값을 유지할 수 있다. SIMD 파이프라인 내 각각의 작업 아이템은 벡터 레지스터에 0의 저장 오프셋을 갖고, 동일 명령어를 실행할 것이다. 각각의 작업 아이템에 대해 이러한 동일한 명령어가 BB 1로부터의 명령어다. BB 1의 명령어는 각각의 작업 아이템에 의해 일대일로 실행되며, PC는 각각의 명령어 인출 후 증분된다.

[0037] 실행될 때, BB 1 종점에 위치한 BBC(0, 1)에서의 코드(506)는 저장된 DVLIW 크기를 1로부터 2로 변화시킨다. BB 3 내의 명령어는 이제, 인출된 DVLIW에 추가된다. 추가적으로, 실행될 때, BBC(0, 1)에서의 코드(506)는 값 1의

저장을 위해 BB 3 로 분기하는 작업 아이템에 대한 벡터 레지스터 내 엔트리를 세팅한다. BB 2 로 분기하는 작업 아이템에 대한 벡터 레지스터 내 엔트리는, 0 저장을 계속함으로써 불변으로 유지된다. 0 및 1의 값들이 이러한 방식으로 사용되지만, 대응하는 표시 및 매핑을 세팅하기 위해 다른 수치 값들이 사용될 수 있다. 이 시점에서, DVLIW는 2개의 명령어를 가지며, 이는 2개의 분리된 기본 블록 BB 2 및 BB 3로부터 섞인다. PC가 계속하여 증분됨에 따라, BB 2 프로세싱이 완료될 때까지, 인출된 DVLIW가 이러한 2개의 기본 블록으로부터 섞인 명령어들을 계속하여 포함한다. 컴파일러는 DVLIW 내의 명령어들의 병렬 실행을 지원하기 위해, 메모리 내에 이러한 방식으로 섞이는 명령어들을 배열하였을 수 있다.

[0038] BB 2의 완료시, DVLIW 크기는 2로 유지된다. 벡터 레지스터 내 저장된 오프셋 역시 그 값을 유지한다. 그러나, 이제 오프셋 0은 BB 2 보다는 BB 4 내 명령어에 대응한다. BBC(0, 4) 완료시, 트레이스 0에서 BB 4 종점의 코드(506)는 3을 저장하도록 사이즈 레지스터를 업데이트하고, 2의 저장을 위해 BB 6로 분기하는 작업 아이템의 엔트리를 업데이트한다. 단일 PC 및 저장된 크기를 i-cache에 전송 후, i-cache로부터 길이 3의 DVLIW가 인출된다. DVLIW는 BB 3 또는 BB 4, BB5 및 BB 6로부터 섞인 명령어를 포함한다. 벡터 레지스터 내 0의 연관되어 저장된 오프셋을 가진 작업 아이템은 BB(0, 5)로부터 인출된 명령어를 획득한다. 벡터 레지스터 내 1의 연관되어 저장된 오프셋을 가진 작업 아이템은, PC가 얼만큼 증분되었느냐에 따라, BB(1, 3) 또는 BB(1, 4)로부터의 결과를 획득한다. 벡터 레지스터 내 2의 연관되어 저장된 오프셋을 가진 작업 아이템은 BB(2, 6)로부터 인출된 명령어를 획득한다. 컴파일러는, 단일 PC 및 저장된 크기가 인출할 DVLIW의 타입을 i-cache에 표시하도록, 이러한 방식으로 메모리에 명령어들을 앞서 배열하였다.

[0039] BBC(1, 4) 완료시, 트레이스 1에서 BB 4 종점의 코드(506)는 4를 저장하도록 사이즈 레지스터를 업데이트하고, 3의 저장을 위해 BB(3,6)로 분기하는 작업 아이템의 엔트리를 업데이트한다. 단일 PC 및 저장된 크기를 i-cache에 전송 후, i-cache로부터 길이 4의 DVLIW가 인출된다. DVLIW는 BB 6의 제 1 사본, BB 5의 단일 사본, BB 6의 제 2 사본, 및 BB 6의 제 3 사본으로부터 섞인 명령어를 포함한다. 벡터 레지스터 내 0의 연관되어 저장된 오프셋을 가진 작업 아이템은 BB(0, 6)로부터 인출된 명령어를 획득한다. 벡터 레지스터 내 1의 연관되어 저장된 오프셋을 가진 작업 아이템은 BB(1, 5)로부터 결과를 획득한다. 벡터 레지스터 내 2의 연관되어 저장된 오프셋을 가진 작업 아이템은 BB(2, 6)로부터 인출된 명령어를 획득한다. 벡터 레지스터 내 3의 연관되어 저장된 오프셋을 가진 작업 아이템은 BB(3, 6)로부터 인출된 명령어를 획득한다. 컴파일러는, 단일 PC 및 저장된 크기가 인출할 DVLIW의 타입을 i-cache에 표시하도록, 이러한 방식으로 메모리에 명령어들을 앞서 배열하였다.

[0040] BB(0, 6), BB(1, 6), BB(2, 6), BB(3, 6) 각각에 대하여, BB 6의 종점에서의 제어 흐름은, 루프의 다른 반복을 위해 BB 1으로 리턴될 수 있고, 또는 루프로부터 종료될 수 있다. 대응하는 레코드 내의 연관된 분기 명령어 및 데이터는 제어 흐름 방향을 런타임 시에 결정할 것이다. 일부 작업 아이템은 다른 반복과 함께 계속될 수 있고, 다른 작업 아이템은 루프를 빠져나갈 수 있다. 특별 코드 상태가 벡터 레지스터 내 대응 엔트리에 저장되어, 어느 경로가 선택되는지를 표시할 수 있다. 주어진 작업 아이템이 다른 루프 반복을 계속하고 복수의 작업 아이템 중 적어도 하나의 다른 작업 아이템은 관련 기본 블록의 코드를 여전히 처리하고 있음을 결정함에 응답하여, 슬립 상태 인코딩이 주어진 작업 아이템의 벡터 레지스터 내 관련 엔트리에 저장될 수 있다.

[0041] 주어진 작업 아이템이 루프를 빠져나감을 결정함에 응답하여, 주어진 작업 아이템의 벡터 레지스터 내 관련 엔트리에 엑시트 상태 인코딩(exit state encoding)이 저장될 수 있다. 슬립(sleep) 및 엑시트(exit) 상태 인코딩 각각은 루프 반복 중 사용되는 오프셋으로부터 고유하고, 서로로부터 고유하다. 일부 실시예에서, 슬립 또는 엑시트 상태인 주어진 작업 아이템의 경우에, 코드(506)는 주어진 작업 아이템에 대한 실행을 중지시키고, 고속 불러오기를 위한 스택 메모리와 같은 메모리에, 적어도 다음 프로그램 카운터(PC) 및 작업 아이템 식별자(ID)를 저장한다.

[0042] BB 6의 종점에서 코드(506)는 각각의 작업 아이템의 상태를 점검할 수 있다. 각각의 작업 아이템이 슬립 상태에 있는지 또는 각각의 작업 아이템이 엑시트 상태에 있는지를 결정함에 응답하여, 프로세서는 코드(506)를 실행하면서, 각자의 저장된 다음 PC로 분기함으로써 각각의 작업 아이템에 대한 실행을 재개할 수 있다. 각각의 작업 아이템이 중지되고 적어도 하나의 작업 아이템이 다른 작업 아이템과 다른 상태에 있음을 결정함에 응답하여, 프로세서는 코드(506)를 실행하면서, 각자의 저장된 다음 PC로 분기함으로써 슬립 상태의 작업 아이템들의 경우에만 실행을 재개할 수 있다. 적어도 하나의 작업 아이템이 루프 내 기본 블록 내의 명령어를 여전히 처리 중이라면, 실행은 적어도 하나의 작업 아이템에 대해 계속되며, 특별 상태의 나머지 작업 아이템들은 대기한다. 슬립 상태를 떠나는 작업 아이템은 BB 1으로 다시 분기한다. 추가적으로, BB 1의 시점에서의 코드(502)는 벡터 및 크기 레지스터를 재초기화한다. 엑시트 상태를 떠나는 작업 아이템은 BB 7으로 분기한다. 추가적으로, BB 7의

시점에서의 코드(502)는, 따라서, 벡터 및 크기 레지스터를 재초기화한다.

[0043] 위 예에서, 루프는 단일 엑시트를 가진다. 복수의 엑시트를 갖는 다른 경우에, 적어도 하나의 다음 PC 및 작업 아이템 ID와 같은 대응 상태 정보가, 스택과 같이 메모리에 저장될 수 있다. 후에, 상태 정보가, 재개를 위해, 스택으로부터의 팝(popping)과 같이, 불러들여질 수 있다. 슬립 상태 또는 엑시트 상태에 있는 작업 아이템들 모두는 스택과 같이 메모리에 저장되는 상태 정보를 가질 수 있다. 서로 다른 작업 아이템이 서로 다른 루프 반복에서 루프를 빠져나감에 따라, 상태 정보를 가진 복수의 엔트리들이 스택과 같이, 메모리에 위치할 수 있다. 재개 시간에서, 실행될 때, 컴파일러-생성 코드는 상태 정보를 팝할 수 있고, 동일한 다음 PC로부터 재개되는 작업 아이템의 정보를 조합할 수 있다.

[0044] 이제 도 6을 참조하면, 컴파일러 기술을 이용하여 프로세서 내 복수의 작업 아이템의 병렬 실행을 최적화하기 위한 방법(600)의 일 실시예가 도시된다. 논의를 위해, 본 실시예의 단계들 및 나중에 설명되는 방법의 후속 실시예들의 단계들이 순차적으로 도시된다. 그러나, 다른 실시예에서 일부 단계들이 도시되는 것과는 다른 순서로 나타날 수 있고, 일부 단계는 동시에 수행될 수 있으며, 일부 단계는 다른 단계들과 조합될 수 있고, 일부 단계는 생략될 수 있다.

[0045] 블록(602)에서, 소프트웨어 프로그램 또는 서브루틴의 위치가 파악되어 분석될 수 있다. 프로그램 코드는 C 또는 다른 언어와 같은 하이 레벨 언어로 설계자에 의해 기록될 수 있다. 이러한 소프트웨어 프로그램은 게임, 비즈니스, 의료, 및 기타 분야에서와 같이, 병렬 데이터 애플리케이션의 컴파일 및 실행을 위해 기록될 수 있다. 프로그램 코드는 소프트웨어 애플리케이션, 서브루트, 동적 링크 라이브러리, 등의 임의의 부분을 의미할 수 있다. 경로명(pathname)은 사용자에 의해 명령 프람프트에서 입력될 수 있다. 대안으로서, 소스 코드의 컴파일을 시작하기 위해, 주어진 디렉토리 위치로부터 또는 기타 위치로부터 경로명이 판독될 수 있다. 프로그램 코드 내의 명령어는 컴파일 중 검사, 변환, 최적화, 및 추가 처리될 수 있다.

[0046] 일부 실시예에서, 소스 코드는 정적으로 컴파일된다. 이러한 실시예에서, 프론트엔드 컴파일 중, 소스 코드는 중간 표현(IR)으로 변환될 수 있다. 백엔드 컴파일 단계는 IR을 기계 코드로 변환시킬 수 있다. 정적 백엔드 컴파일은 더 많은 변환 및 최적화를 수행할 수 있다. 다른 실시예에서, 소스 코드는 JIT(Just-In-Time) 법으로 컴파일될 수 있다. JIT 법은 시스템 구성 획득 후 적절한 이진 코드를 생성할 수 있다. 어느 방법을 이용하여서도, 컴파일러는 함수 콜, 루프, 루프 내의 트레이스, 프로그램 코드 내 기본 블록을 식별할 수 있다. 하나 이상의 제어 흐름 그래프가 프로그램 분석 중 구성될 수 있다.

[0047] 다양한 실시예에서, 프로그램 코드는 범용 프로세서와 같이, 프로세서 상에서 컴파일된다. 프로그램 코드는 SIMD 마이크로구조와 같은 병렬 마이크로구조를 포함하는, 목표 프로세서용으로 컴파일될 수 있다. 하나 이상의 관련 데이터 레코드는 하나 이상의 작업 아이템을 생성하기 위해 함수 콜에 할당될 수 있다.

[0048] 프로그램 코드 내 임의의 분기 지점을 검출하기 전에, 컴파일러는 프로그램 코드에 나타남에 따라 메모리 내 분석된 그리고 변환된 명령어를 레이아웃할 수 있다. 본질적으로, 컴파일러는 1의 크기로 VLIW를 생성할 수 있다. 컴파일러가 식별된 루프 내의 분기점을 검출할 경우(조건부 블록(604)), 블록(606)에서, 컴파일러는 VLIW(very large instruction words)를 생성할 수 있다. 컴파일러는 분기점과 대응하는 집중점 사이에서 복수의 기본 블록으로부터 섞인 명령어들을 메모리에 배열함으로써 VLIW를 생성할 수 있다. 하나 이상의 관련 데이터 레코드는 하나 이상의 연관된 작업 아이템을 생성하기 위해, 생성된 VLIW 내의 섞인 명령어에 동반되도록 배열 및 할당될 수 있다.

[0049] 블록(608)에서, 생성 코드가 삽입될 수 있고, 생성 코드는 실행될 때, 목표 프로세서 내 복수의 병렬 실행 레인 중 주어진 레인에, VLIW 내의 명령어를 가리키는 오프셋을 매핑한다. 대안으로서, 매핑은 오프셋과 작업 아이템 ID 사이에 놓일 수 있다. 블록(610)에서, 실행될 때, 인출할 다음 VLIW의 매핑과 VLIW의 크기를 실행될 때 업데이트하도록, 생성된 코드를 삽입할 수 있다. 블록(612)에서, 생성된 코드는 루프 종점에 삽입될 수 있고, 실행될 때, 슬립 또는 엑시트 상태로 진행하는 실행 레인을 위한 상태 정보를 저장한다. 생성 코드는 앞서 예에서 설명한 바와 같이 프로그램 코드의 특정 지점에 삽입될 수 있다. 분기점 및 집중점과 연관된 기본 블록들은 매핑 및 DVLIW 크기에 대한 업데이트를 유지하기 위한 더 많은 삽입 코드를 가질 수 있다.

[0050] 이제 도 7을 참조하면, 하드웨어 기술을 이용하여 프로세서 내 복수의 작업 아이템의 병렬 실행을 최적화하기 위한 방법(700)의 일 실시예가 도시된다. 논의를 위해, 본 실시예 및 나중에 설명되는 방법들의 후속 실시예의 단계들이 순차적으로 도시된다. 그러나 다른 실시예에서, 일부 단계들이 도시되는 것과는 다른 순서로 나타날 수 있고, 일부 단계는 동시에 수행될 수 있으며, 일부 단계는 다른 단계들과 조합될 수 있고, 일부 단계는 생략

될 수 있다.

[0051] 블록(702)에서, 관련 데이터 레코드가 복수 작업 아이템 생성을 위해 컴파일된 코드에 할당된다. 블록(704)에서, 작업 아이템은 SIMD(single instruction multiple data) 마이크로구조를 이용하여 목표 프로세서에 스케쥴링된다. 블록(706)에서, 업데이트되는 VLIW 크기 및 단일 프로그램 카운터(PC)를 이용하여, VLIW가, 업데이트되는 VLIW 크기와 동일한 길이로, i-cache와 같은 메모리로부터 인출된다. VLIW의 명령어는 루프 내 분기점과 집중점 사이의 별도의 기본 블록으로부터 올 수 있다.

[0052] 블록(708)에서, 작업 아이템을 실행하는 프로세서 내 병렬 실행 레인과 인출되는 VLIW 내의 명령어들 사이에서 정보를 매핑하기 위해 벡터 레지스터가 판독된다. 매핑 정보는 주어진 작업 아이템 및 대응하는 실행 레인에 대해 결정될 수 있으며, 처리할 VLIW 내의 명령어다. 블록(710)에서, VLIW 내 명령어들이 병렬 실행 레인을 이용하여 동시에 실행된다. 주어진 작업 아이템에 대한 루프의 종점에 도달할 경우(조건 블록(712)) 그리고 어떤 작업 아이템도 액티브하다고 검출되지 않는 경우(조건 블록(714)), 블록(716)에서, 병렬 실행 레인에 할당된 각각의 작업 아이템에 대해 각자의 상태 정보가 판독된다. 상태 정보는 적어도 하나의 다음 PC 및 작업 아이템 ID를 포함할 수 있다. 상태 정보는 병렬 실행 레인 내 실행의 계속을 위해 사용될 수 있다. 주어진 작업 아이템에 대한 루프의 종점에 도달할 경우(조건 블록(712)) 그리고 어떤 작업 아이템이 액티브하다고 검출될 경우(조건 블록(714)), 블록(718)에서, 주어진 작업 아이템에 대한 상태 정보가 차후 사용을 위해 저장된다. 주어진 작업 아이템은 실행을 중지시킬 수 있고, 슬립 또는 엑시트 상태로 배치될 수 있다.

[0053] 이제 도 8을 살펴보면, 객체 코드 배열(800)을 예시하는 다른 실시예의 일반화된 블록도가 도시된다. 코드 배열(800)은 도 2 및 도 3에 각각 도시되는 제어 흐름 그래프(200) 및 동반 실행 시퀀스(320)에 대하여 컴파일러에 의해 생성될 수 있는 객체 코드 레이아웃의 일 실시예를 도시한다. 코드(502-506)는 앞서 설명한 바와 동일한 기능을 수행할 수 있다.

[0054] 기본 블록의 크기는 컴파일 시간에 알려진다. 컴파일러는 프로그램 코드의 명령어를 배열 및 스케줄링할 수 있어서, 초기화 코드를 최소화시키고 DVLIW 크기를 감소시킬 수 있다. 제어 흐름 그래프(200) 및 객체 코드 배열(500)을 이용한 앞서 예에서, BB 6의 4개의 사본이 사용된다. 컴파일러는 갭을 삽입함으로써, 트레이스의 수를 감소시킬 수 있고, 동반되는 오프셋을 감소시킬 수 있다. 갭은 nop 작동을 이용할 수 있다.

[0055] 코드 배열(500)과 관련하여 앞서 설명한 단계들이 여기에 사용될 수 있다. BB 2의 완료시, DVLIW 크기는 2로 유지된다. 그러나, 인출되는 DVLIW 내 2개의 명령어 중 하나는 배열(800) 내 갭을 제공하는 nop 작동이다. 오프셋 0은 nop 작동에 대응하고, 오프셋 1은 BB 3의 명령어에 대응한다. BB 3의 완료시, DVLIW가 2로부터 1로 감소한다. 이제 오프셋 0은 BB 4의 명령어에 대응하고, 오프셋 1은 nop 작동에 대응한다.

[0056] BBC(0, 4) 완료시, 트레이스 0의 BB 4 종점에서 코드(506)는 사이즈 레지스터를 1로부터 2로 업데이트한다. 코드(506)는 1의 저장을 위해 BB 6 로 분기하는 작업 아이템에 대한 엔트리를 또한 업데이트한다. 단일 PC 및 저장된 크기를 i-cache에 전송 후 i-cache로부터 길이 2의 DVLIW가 인출된다. DVLIW는 BB 5 및 BB 6로부터 섞인 명령어를 포함한다. 벡터 레지스터 내 연관되어 저장된 0의 오프셋을 가진 작업 아이템은 BB(0, 5)로부터 인출된 명령어를 획득한다. 벡터 레지스터 내 연관되어 저장된 1의 오프셋을 가진 작업 아이템은 BB(1, 6)로부터 인출된 명령어를 획득한다.

[0057] BB(1, 6)의 완료시, 코드(506)는 DVLIW 크기를 2로부터 1로 업데이트한다. BB(1, 6)에 대응하는 작업 아이템은 실행을 중지하고, 상태 정보를 저장하며, 다음 PC에서 실행 재개시까지 대기한다. 다음 PC는 BB 1 또는 BB 7을 가리킬 수 있다. 다른 단계들이 먼저 설명된 바와 같이 수행될 수 있다. 코드 배열(800)은 검출되는 분기점과 대응 집중점 사이의 제 1 트레이스 경로가 주어진 분기점 및 대응 집중점 사이의 제 2 트레이스 경로보다 작다고 결정함에 응답하여, 컴파일러가 제 1 트레이스 경로의 완료와 대응하는 집중점 사이에서 생성되는 VLIW 내 제 2 트레이스 경로에 대응하는 명령어로 nop를 그룹화할 수 있다.

[0058] 앞서 설명되는 각각의 실시예에 대하여, 추가적인 병렬화가 나타날 수 있고, 코드는 루프 내부의 코드와 함께 루프 외부에서 병렬화된다. 예를 들어, BB 7의 프로그램 코드는 BB 1과 병렬화되어, 루프를 완성시키는 작업 아이템에 대한 프로그램을 종료시킬 수 있다. 추가적으로, 오프셋이 단일 명령어보다, VLIW 내에 복수의 리소스-독립적 명령어에 대응함을 검출함에 응답하여, 관련 작업 아이템 및 실행 레인이 실행 레인 내에서 복수의 리소스-독립적 명령어를 동시에 실행할 수 있다.

[0059] 더욱이, 컴파일러는 레지스터 할당을 이용하여 DVLIW 크기를 감소시킬 수 있다. 프로그램 코드는 다음의 문장을 포함할 수 있다: X = (A+B) + (B+M). 여기서, 2개의 작동이 동일한 연산부호(opcode)를 이용한다. 제 1 트레이

스는 T1 = A+B 와 같은 ADD 연산을 포함한다. 제 2 트레이스는 T2 = C+D 와 같은 ADD 연산을 포함한다. 0과 같은 오프셋과 함께 기본 블록 X(BB X)로부터 T1을 이용하는 작업 아이템들이 존재한다. 1과 같은 오프셋과 함께 BB Y로부터 T2를 이용하는 다른 작업 아이템들이 존재한다. 제 1 연산자 쌍 "C" 및"A", 제 2 연산자 쌍 "B" 및"D", 및 결과 쌍 "T1" 및 "T2" 가 각각 BB X 및 BB Y의 동일 레지스터에 할당될 경우, 수식 r3 = r1 + r2 가 1의 크기를 가진 DVLIW 로 사용될 수 있다. 대응하는 오프셋이 0으로 세팅되어, 디코딩 시간을 절감하거나 슬롯을 비울 수 있다.

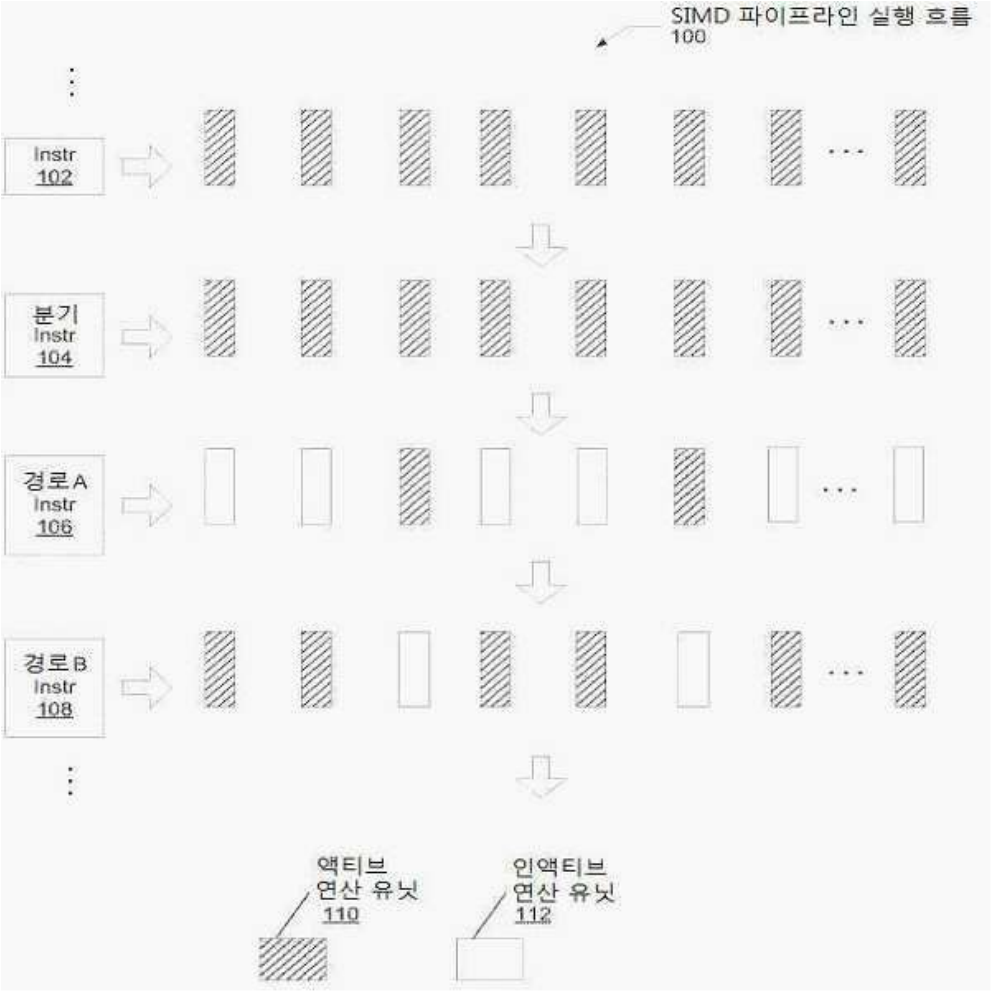
[0060] 위에서 설명한 실시예들이 소프트웨어를 포함할 수 있다. 이러한 실시예에서, 방법 및/또는 메커니즘을 구현하는 프로그램 명령어가 컴퓨터 판독가능 매체 상에 전달 또는 저장될 수 있다. 프로그램 명령어를 저장하도록 구성된 수많은 타입의 매체들이 가용하며, 하드 디스크, 플로피 디스크, CD-ROM, DVD, 플래시 메모리, 프로그래머블 ROM(PROM), 랜덤 액세스 메모리(RAM), 및 다양한 다른 형태의 휘발성 또는 비휘발성 저장 수단을 포함한다. 일반적으로 말해서, 컴퓨터 액세스가능 저장 매체는 컴퓨터에 명령어 및/또는 데이터를 제공하기 위해 사용 중 컴퓨터에 의해 액세스가능한 임의의 저장 매체를 포함할 수 있다. 예를 들어, 컴퓨터 액세스가능 저장 매체는 자기 또는 광학 매체, 가령, 디스크(고정식 또는 제거가능식), 테이프, CD-ROM, 또는 DVD-ROM, CD-R, CD-RW, DVD-R, DVD-RW, 또는 블루-레이와 같은 저장 매체를 포함할 수 있다. 저장 매체는 RAM(가령, 동기식 동적 RAM(SDRAM), 더블 데이터 레이트(DDR, DDR2, DDR3, 등) SDRAM, 저전력 DDR (LPDDR2, 등) SDRAM, 램버스 DRAM (RDRAM), 정적 RAM (SRAM), 등), ROM, 플래시 메모리, 범용 시리얼 버스(USB) 인터페이스, 등과 같은 주변 인터페이스를 통해 액세스가능한 비휘발성 메모리(가령, 플래시 메모리)와 같은, 휘발성 또는 비휘발성 메모리 매체를 더 포함할 수 있다. 저장 매체는 네트워크 및/또는 무선 링크와 같은 통신 매체를 통해 액세스가능한 저장 매체와, 마이크로일렉트로미캐니컬 시스템(MEMS)를 포함할 수 있다.

[0061] 추가적으로, 프로그램 명령어는 C와 같은 하이 레벨 프로그래밍 언어, 또는, Verilog, VHDL과 같은 설계 언어(HDL), 또는, GDS II 스트림 포맷(GDSII)와 같은 데이터베이스 포맷으로 하드웨어 기능의 거동-레벨 설명 또는 레지스터-전달 레벨(RTL) 설명을 포함할 수 있다. 일부 경우에, 설명(description)은 합성 툴에 의해 판독될 수 있고, 이러한 합성 툴은 이러한 설명을 합성하여, 합성 라이브러리로부터 게이트들의 리스트를 포함하는 네트리스트를 생성할 수 있다. 네트리스트는 시스템을 포함하는 하드웨어의 기능을 또한 나타내는 한 세트의 게이트를 포함한다. 네트리스트는 그 후, 마스크에 적용될 기하학적 형상을 설명하는 데이터 세트를 생성하도록 배치 및 전달될 수 있다. 그 후 마스크는 다양한 반도체 제조 단계에 이용되어, 시스템에 대응하는 반도체 회로(들)을 생산할 수 있다. 대안으로서, 컴퓨터 액세스가능 저장 매체 상의 명령어는 요망되는 바와 같이, (합성 라이브러리 있는 또는 없는) 네트리스트, 또는 데이터 세트일 수 있다. 추가적으로, Cadence®, EVE®, 및 Mentor Graphics®과 같은 벤더로부터 하드웨어 기반 타입의 에뮬레이터에 의한 에뮬레이션 용도로 명령어가 이용될 수 있다.

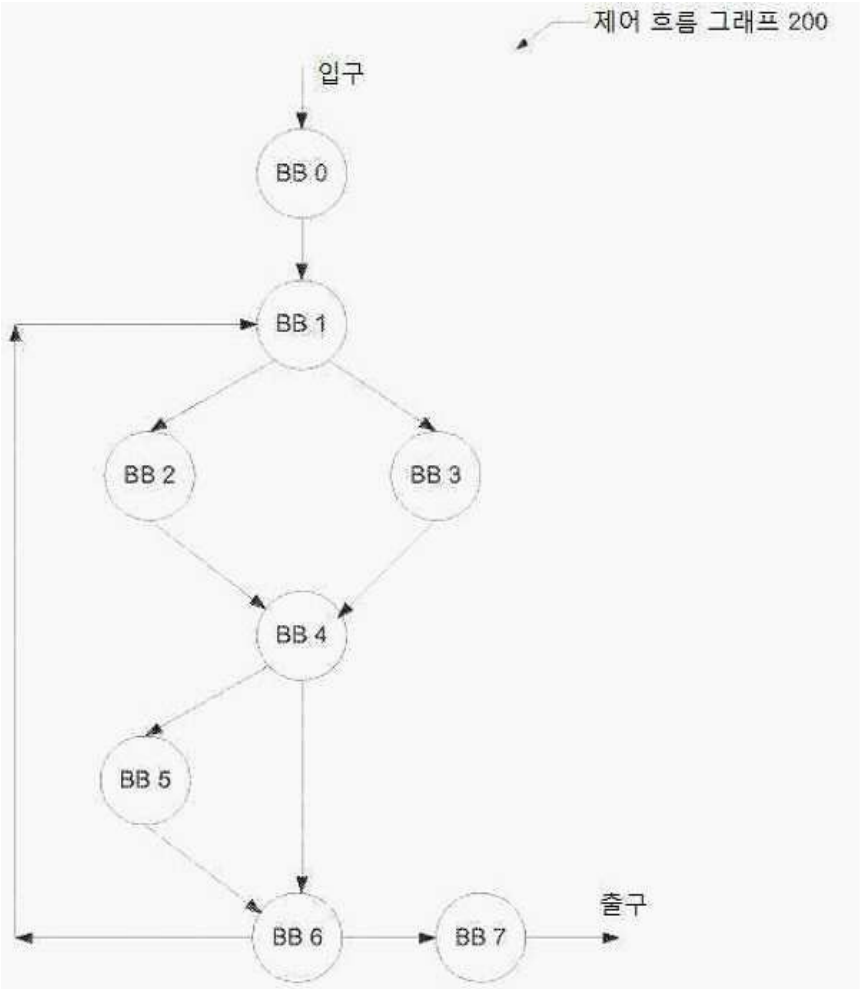
[0062] 위 실시예들이 상상히 세부적으로 설명되었으나, 위 설명을 완전히 이해하면 수많은 변형 및 변형예들이 당 업자에게 명백해질 것이다. 다음의 청구범위는 이러한 모든 변형 및 변형예들을 포괄하도록 해석되어야 한다.

도면

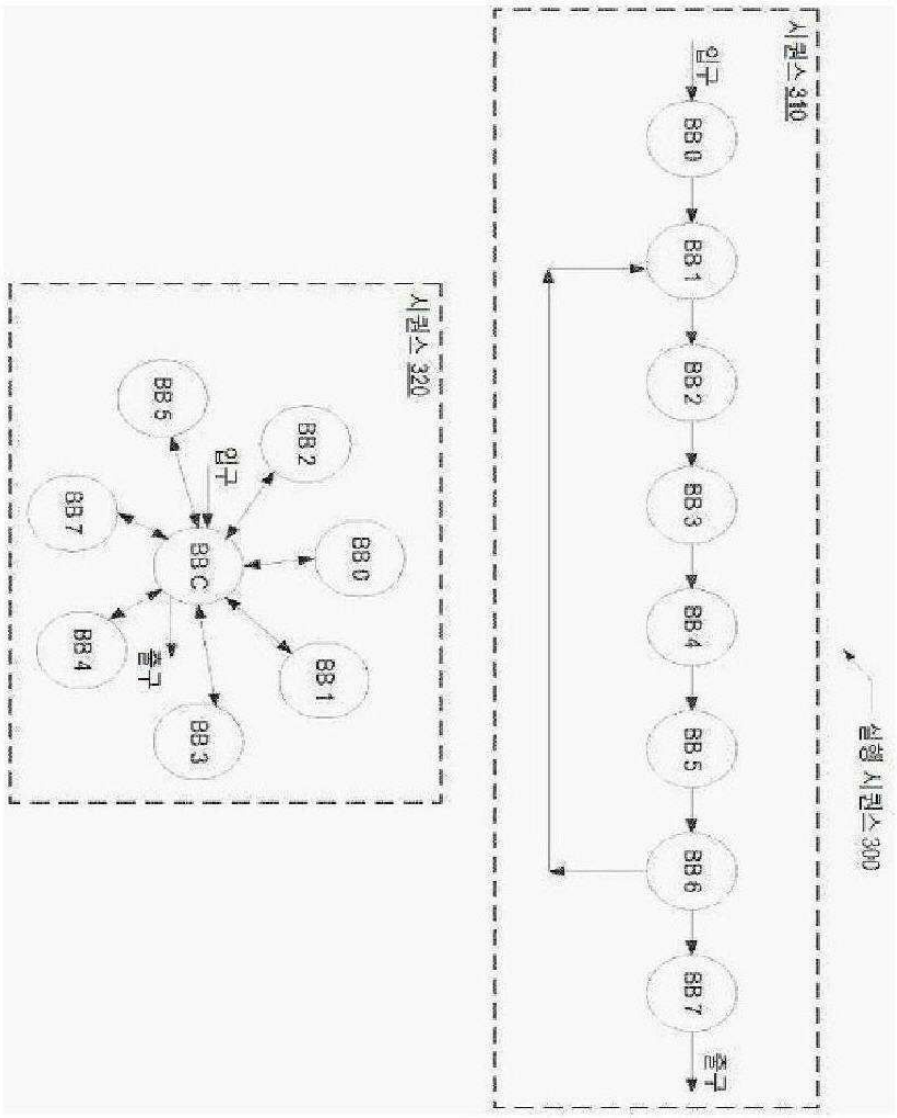
도면1


SIMD 파이프라인 실행 흐름
100
Instr
102
분기
Instr
104
경로A
Instr
106
경로B
Instr
108
액티브
연산 유닛
110
인액티브
연산 유닛
112

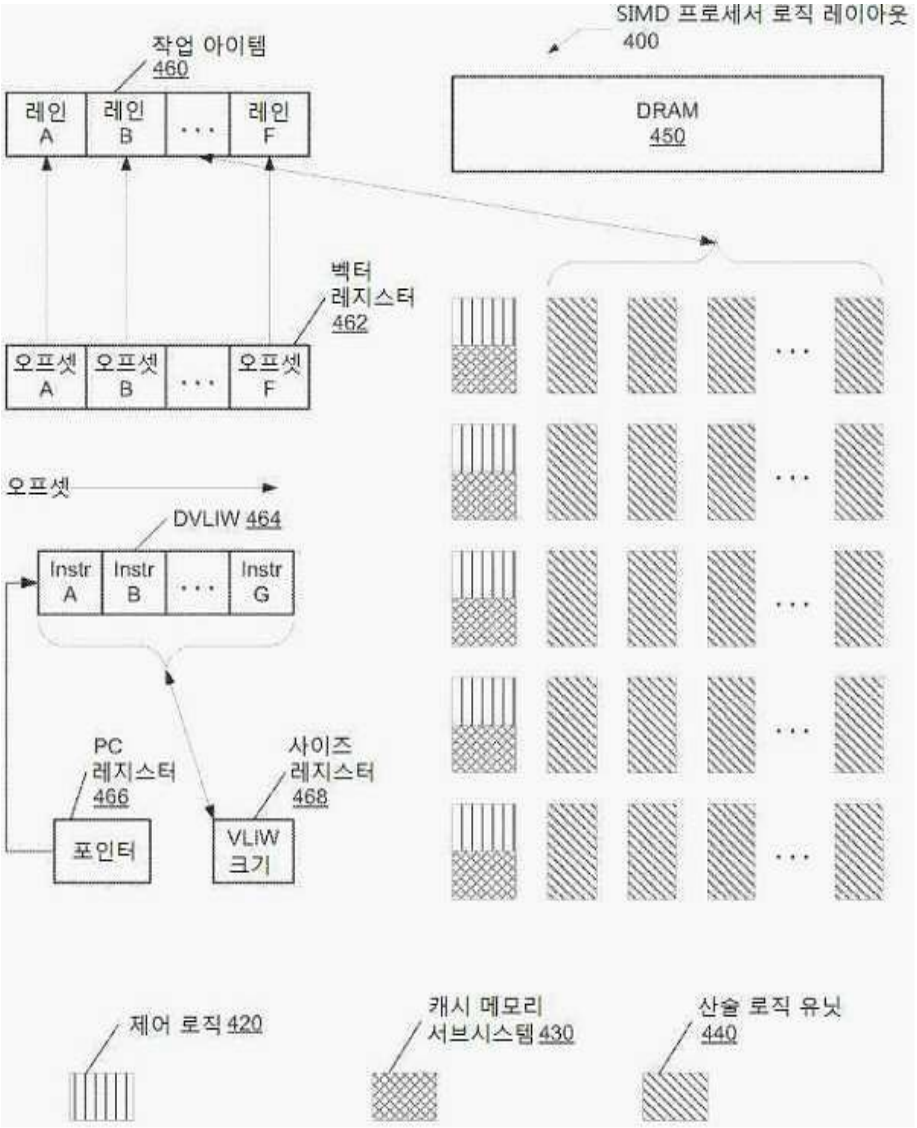
도면2



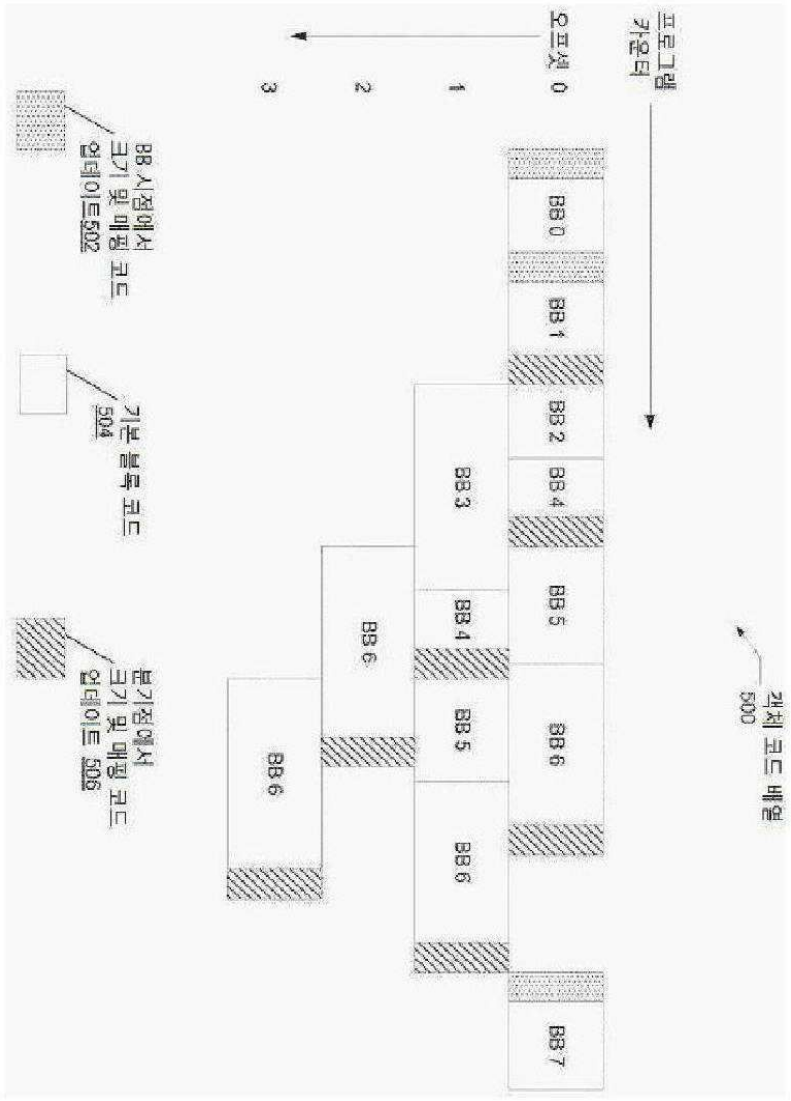
도면3


실행 시퀀스 300
시퀀스 310
입구
BB 0
BB 1
BB 2
BB 3
BB 4
BB 5
BB 6
BB 7
출구
시퀀스 320
BB 5
BB 2
입구
BB 7
BB C
BB 0
BB 4
출구
BB 1
BB 3

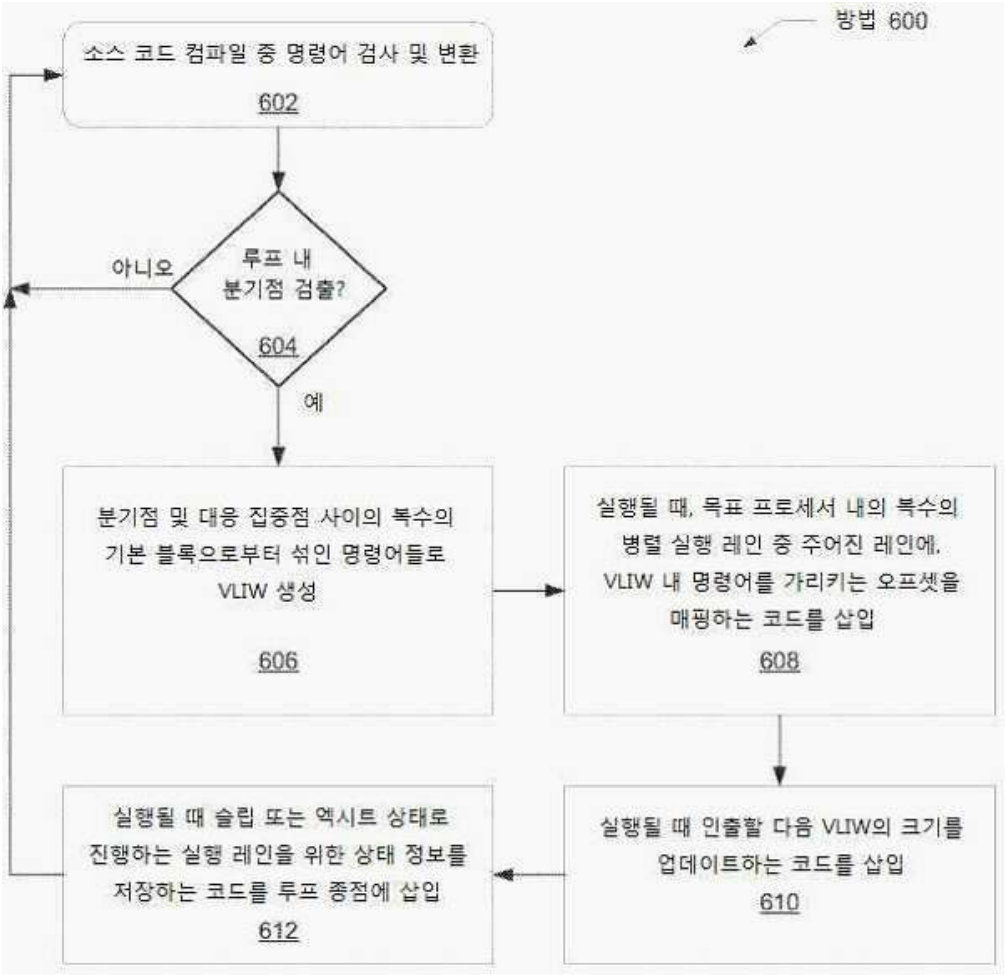
도면4


SIMD 프로세서 로직 레이아웃
400
작업 아이템
460
레인 A
레인 B
레인 F
DRAM
450
벡터 레지스터
462
오프셋 A
오프셋 B
오프셋 F
오프셋
DVLIW 464
Instr A
Instr B
Instr G
PC 레지스터
466
사이즈 레지스터
468
포인터
VLIW 크기
제어 로직 420
캐시 메모리 서브시스템 430
산술 로직 유닛
440

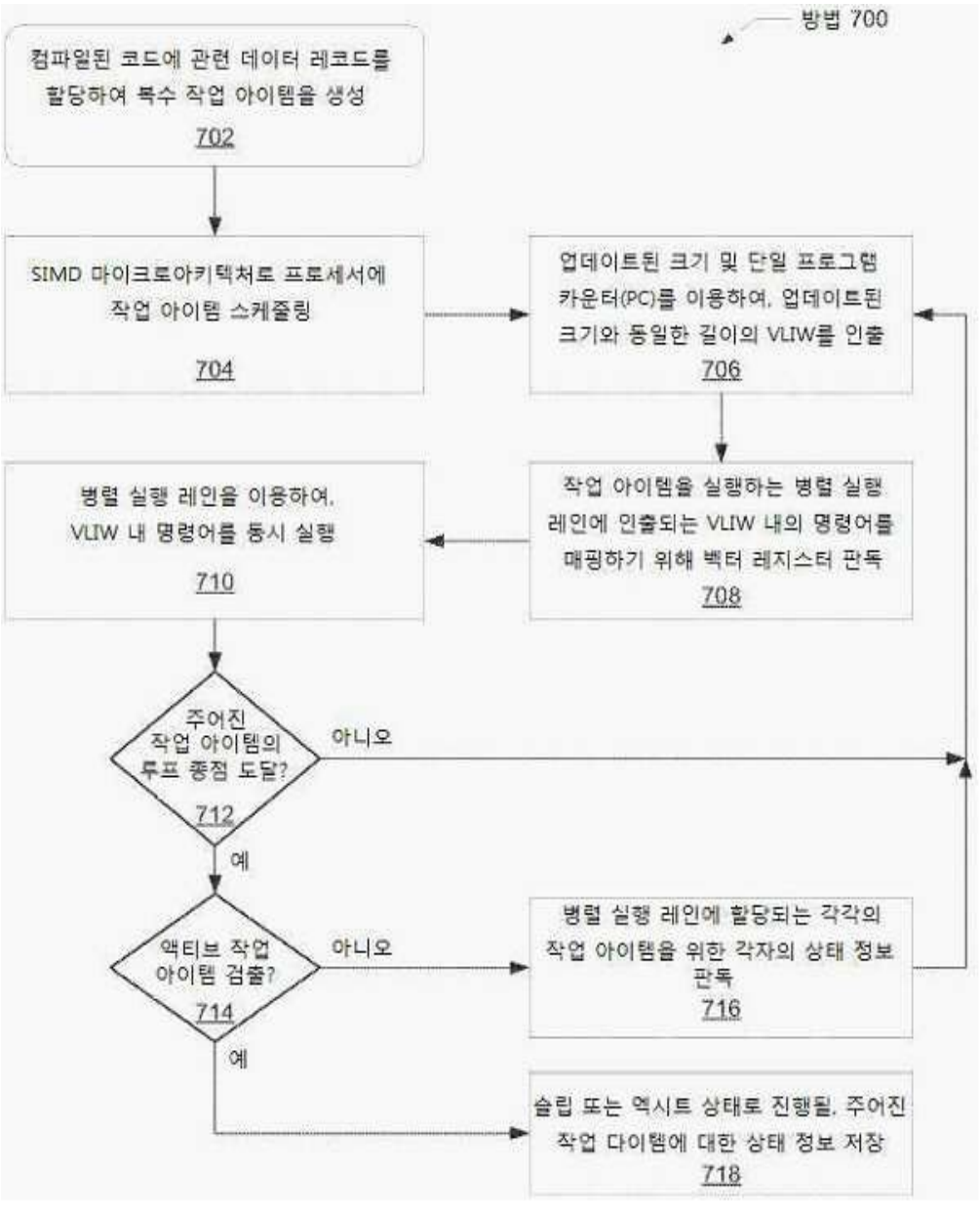
도면5


프로그램 카운터
오프셋 0
1
2
3
BB 0
BB 1
BB 2
BB 3
BB 4
BB 5
BB 6
BB 7
BB 4
BB 5
BB 6
BB 6
객체 코드 배열 500
BB 시점에서 크기 및 매핑 코드 업데이트502
기본 블록 코드 504
분기점에서 크기 및 매핑 코드 업데이트 506

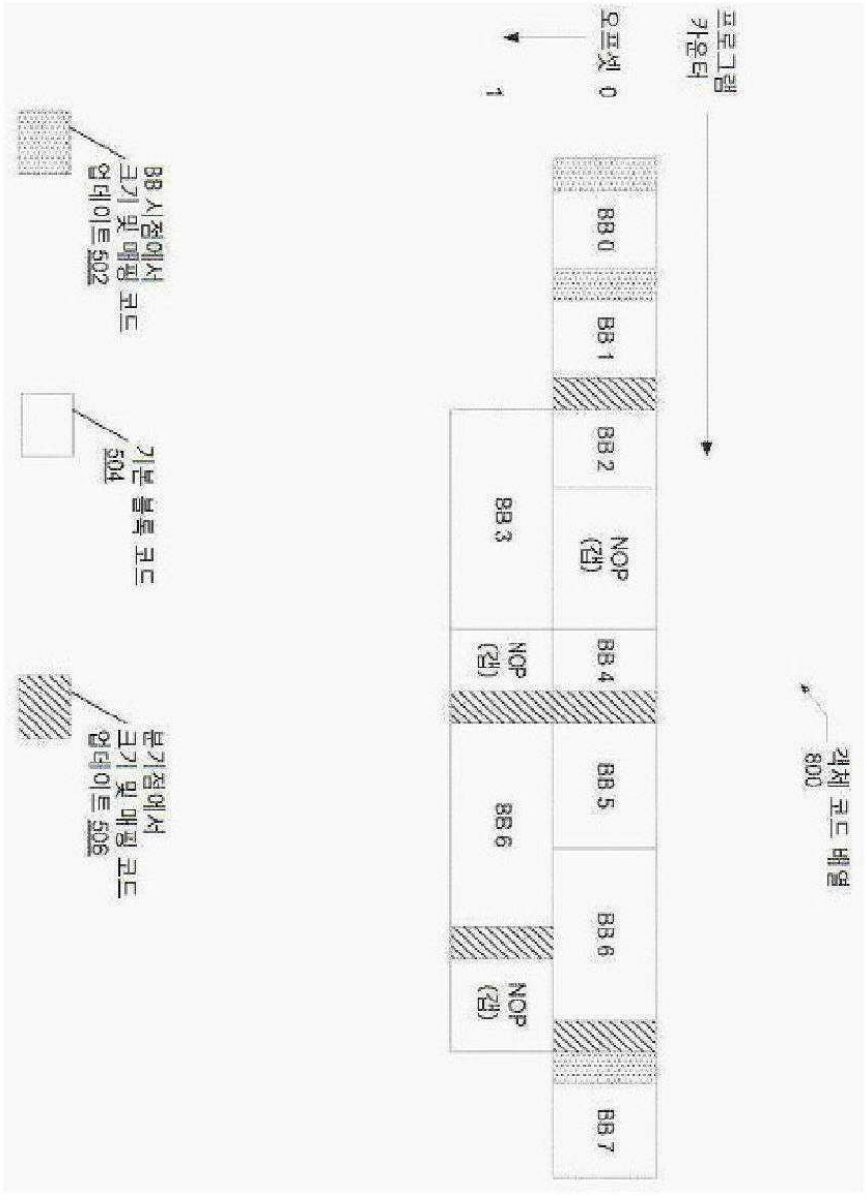
도면6


방법 600
소스 코드 컴파일 중 명령어 검사 및 변환
602
루프 내
분기점 검출?
604
아니오
예
분기점 및 대응 집중점 사이의 복수의 기본 블록으로부터 섞인 명령어들로 VLIW 생성
606
실행될 때, 목표 프로세서 내의 복수의 병렬 실행 레인 중 주어진 레인에, VLIW 내 명령어를 가리키는 오프셋을 매핑하는 코드를 삽입
608
실행될 때 인출할 다음 VLIW의 크기를 업데이트하는 코드를 삽입
610
실행될 때 슬립 또는 엑시트 상태로 진행하는 실행 레인을 위한 상태 정보를 저장하는 코드를 루프 종점에 삽입
612

도면7


방법 700
컴파일된 코드에 관련 데이터 레코드를 할당하여 복수 작업 아이템을 생성
702
SIMD 마이크로아키텍처로 프로세서에 작업 아이템 스케줄링
704
업데이트된 크기 및 단일 프로그램 카운터(PC)를 이용하여, 업데이트된 크기와 동일한 길이의 VLIW를 인출
706
작업 아이템을 실행하는 병렬 실행 레인에 인출되는 VLIW 내의 명령어를 매핑하기 위해 벡터 레지스터 판독
708
병렬 실행 레인을 이용하여, VLIW 내 명령어를 동시 실행
710
주어진 작업 아이템의 루프 종점 도달?
712
아니오
예
액티브 작업 아이템 검출?
714
아니오
병렬 실행 레인에 할당되는 각각의 작업 아이템을 위한 각자의 상태 정보 판독
716
예
슬립 또는 엑시트 상태로 진행될, 주어진 작업 다이템에 대한 상태 정보 저장
718

도면8


객체 코드 배열
800
프로그램 카운터
오프셋 0
1
BB 0
BB 1
BB 2
BB 3
NOP (갭)
BB 4
NOP (갭)
BB 5
BB 6
BB 6
NOP (갭)
BB 7
BB 시점에서 크기 및 매핑 코드 업데이트 502
기본 블록 코드 504
분기점에서 크기 및 매핑 코드 업데이트 506