(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2021년09월24일
(11) 등록번호 10-2305084
(24) 등록일자 2021년09월16일

(51) 국제특허분류(Int. Cl.)
G06F 9/48 (2018.01) *G06F 9/44* (2018.01)
(52) CPC특허분류
G06F 9/4843 (2013.01)
G06F 9/4494 (2018.02)
(21) 출원번호 10-2015-7033437
(22) 출원일자(국제) 2014년04월23일
심사청구일자 2019년03월25일
(85) 번역문제출일자 2015년11월23일
(65) 공개번호 10-2016-0004335
(43) 공개일자 2016년01월12일
(86) 국제출원번호 PCT/US2014/035098
(87) 국제공개번호 WO 2014/176313
국제공개일자 2014년10월30일
(30) 우선권주장
61/815,052 2013년04월23일 미국(US)
(56) 선행기술조사문헌
KR1020090018113 A*
(뒷면에 계속)
전체 청구항 수 : 총 30 항

(73) 특허권자
아브 이니티오 테크놀로지 엘엘시
미국 02421 매사추세츠주 렉싱턴 스프링 스트리트 201
(72) 발명자
스탠필, 크레이그 더블유.
미국 메사추세츠 01773 링컨 허클베리 힐 로드 43
(74) 대리인
인비전 특허법인

심사관 : 유진태

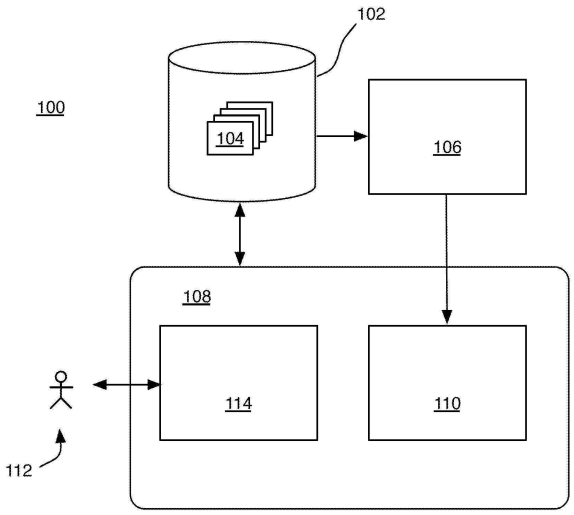
(54) 발명의 명칭 **컴퓨팅 시스템에 의해 수행되는 태스크 제어**

(57) 요 약

그래프 기반 프로그램 명세서(104)(graph-based program specification)는 태스크들을 표시하는 복수의 노드들을 포함하고, 상기 그래프 기반 프로그램 명세서는 상기 노드들에 의해 표시된 복수의 태스크들 중 적어도 일부의 순서를 명시하는 노드들 사이의 방향성 엣지(directed edges)를 포함함; 및 적어도 하나의 프로세서를 사용하

(뒷면에 계속)

대 표 도 - 도1



여, 상기 그래프 기반 프로그램 명세서에 의해 명시된 프로그램을 실행하는 단계를 포함하되, 상기 프로그램을 실행하는 단계는 제1 태스크에 대응하는 제1 서브루틴을 실행하는 단계, 여기서 상기 제1 서브루틴은 상기 제1 태스크를 수행하는 제1 태스크부를 포함함; 가능한 상태들의 집합으로부터 선택된 상기 제1 태스크의 상태를 지시하는 상태 정보를 저장하는 단계, 여기서 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하기 위해 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함함; 및 제2 태스크에 대응하는 제2 서브루틴을 실행하는 단계를 포함하되, 상기 제2 서브루틴은 상기 제2 태스크를 수행하는 제2 태스크부, 및 상기 저장된 상태 정보에 의해 지시된 상기 제1 태스크의 상태에 적어도 부분적으로 기반하여 상기 제2 태스크부의 실행을 제어하는 제어부를 포함한다.

(52) CPC특허분류
G06F 9/4881 (2013.01)

(56) 선행기술조사문헌
JP2012108576 A*
KR1020110116178 A
JP2010244563 A
JP2010286931 A
*는 심사관에 의하여 인용된 문헌

명 세 서

청구범위

청구항 1

컴퓨팅 시스템(computing system)에 의하여 수행되는 태스크(task) 제어 방법에 있어서

데이터 스토리지 시스템 내에 그래프 기반 프로그램 명세서(graph-based program specification)를 위한 데이터 구조를 저장하는 단계, 여기서 상기 그래프 기반 프로그램 명세서는 태스크들을 표시하는 복수의 노드들을 포함하고, 상기 그래프 기반 프로그램 명세서는 상기 노드들에 의해 표시된 복수의 태스크들 중 적어도 일부의 순서를 명시하는 노드들 사이의 방향성 엣지(directed edges)를 포함함; 및

적어도 하나의 프로세서를 사용하여, 상기 그래프 기반 프로그램 명세서에 의해 명시된 프로그램을 실행하는 단계를 포함하되, 상기 프로그램을 실행하는 단계는

제1 프로세스에 의해 제1 태스크에 대응하는 제1 서브루틴을 실행하는 단계, 여기서 상기 제1 서브루틴은 상기 제1 태스크를 수행하는 제1 태스크부를 포함하고,

상기 그래프 기반 프로그램 명세서에서 상기 제1 태스크에 대한 적어도 하나의 대응하는 후속 태스크들의 실행들을 제어하기위한 로직을 포함하는 상기 제1 서브루틴에 포함된 제1 제어부를 포함하고, 여기서, 상기 제1 제어부는 제2 태스크에 대응하는 제2 서브루틴을 호출하기위한 제1 함수 호출을 포함하고, 여기서, 상기 제2 태스크는 상기 제2 태스크의 실행 시간을 추적하는 지연 타이머가 지연 임계치를 초과하는 시간 값에 도달하여 상기 제1 프로세스가 상기 제2 태스크의 실행을 인수하는 제2 프로세스를 생성하게 할 때까지 상기 제1 프로세스를 사용하여 실행을 시작하고, 여기서, 상기 제1 제어부는 상기 지연 임계치를 초과하는 상기 지연 타이머에 응답하여 상기 제1 프로세스가 상기 제2 프로세스를 생성 한 후 상기 제1 프로세스 상에서 실행을 시작하고, 상기 제2태스크의 실행을 인수하여 상기 제2 태스크와 제3 태스크가 동시에 실행되도록 하는 상기 제3 태스크에 대응하는 제3 서브루틴을 호출하기 위한 제2 함수 호출을 더 포함하며;

가능한 상태들의 집합으로부터 선택된 상기 제1 태스크의 상태를 지시하는 상태 정보를 저장하는 단계, 여기서 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하기 위해 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함함; 및

상기 제2 태스크에 대응하는 상기 제2 서브루틴을 실행하는 단계를 포함하되, 상기 제2 서브루틴은 상기 제2 태스크를 수행하는 제2 태스크부, 및 상기 저장된 상태 정보에 의해 지시된 상기 제1 태스크의 상태에 적어도 부분적으로 기반하여 상기 제2 태스크부의 실행을 제어하는 제2 제어부를 포함하는 것을 특징으로 하는 방법 .

청구항 2

제1항에 있어서, 상류 노드로부터 하류 노드로 향하는 방향성 엣지는 상기 일부의 순서 내에서 상기 상류 노드에 의해 표시된 태스크가 상기 하류 노드에 의해 표시된 태스크에 우선하는 것을 지시하는 것을 특징으로 하는 방법.

청구항 3

제1항 또는 제2항에 있어서, 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하였음을 나타내는 완료 상태, 상기 제1 태스크부가 상기 제1 태스크의 수행을 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함하는 것을 특징으로 하는 방법.

청구항 4

제3항에 있어서, 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하는 과정(process)에 있음을 나타내는 활성 상태를 포함하는 것을 특징으로 하는 방법.

청구항 5

제1항에 있어서, 상기 제1 제어부는 상기 제2 태스크부가 호출되는지 여부를 판단하는 로직을 포함하는 것을 특징으로 하는 방법.

청구항 6

제5항에 있어서, 상기 로직은 상기 제2 태스크부가 플래그(flag)의 값을 기반으로 호출되는지 여부를 판단하는 것을 특징으로 하는 방법.

청구항 7

제5항에 있어서, 상기 로직은, 제2 태스크를 표시하는 노드와 방향성 엣지에 의해 연결된 노드들에 의해 표시된 모든 태스크들의 상태들 중 적어도 일부에 기반하여 상기 제2 태스크부가 호출되는지 여부를 판단하는 것을 특징으로 하는 방법.

청구항 8

제7항에 있어서, 상기 로직은, 상기 제2 태스크를 표시하는 노드와 방향성 엣지에 의해 연결된 노드들에 의하여 표시되는 모든 태스크들이 유예 상태이면, 상기 제2 태스크부가 호출되지 않는 것으로 판단하는 것을 특징으로 하는 방법.

청구항 9

제7항에 있어서, 상기 로직은 상기 제2 태스크를 표시하는 노드와 방향성 엣지에 의해 연결된 노드들에 의하여 표시되는 적어도 하나의 태스크가 완료 상태이고, 상기 제2 태스크를 표시하는 노드를 향하는 방향성 엣지에 의해 연결된 노드들에 의해 표시되는 태스크들 중 어느 것도 대기 상태가 아닌 것을 특징으로 하는 방법.

청구항 10

태스크(task)를 제어하는 컴퓨터 프로그램을 저장하는 컴퓨터 판독가능한 저장 매체에 있어서, 상기 컴퓨터 프로그램은 컴퓨팅 시스템에

그래프 기반 프로그램 명세서(graph-based program specification)를 위한 데이터 구조를 저장하고, 여기서 상기 그래프 기반 프로그램 명세서는 태스크들을 표시하는 복수의 노드들을 포함하고, 상기 그래프 기반 프로그램 명세서는 상기 노드들에 의해 표시된 복수의 태스크들 중 적어도 일부의 순서를 명시하는 노드들 사이의 방향성 엣지(directed edges)를 포함함; 그리고

상기 그래프 기반 프로그램 명세서에 의해 명시된 프로그램을 실행하도록 구성된 명령들을 포함하되, 상기 실행은

제1 프로세스에 의해 제1 태스크에 대응하는 제1 서브루틴을 실행하는 단계, 여기서 상기 제1 서브루틴은 상기 제1 태스크를 수행하는 제1 태스크부를 포함하고,

상기 그래프 기반 프로그램 명세서에서 상기 제1 태스크에 대한 적어도 하나의 대응하는 후속 태스크들의 실행

들을 제어하기위한 로직을 포함하는 상기 제1 서브루틴에 포함된 제1 제어부를 포함하고, 여기서, 상기 제1 제어부는 제2 태스크에 대응하는 제2 서브루틴을 호출하기위한 제1 함수 호출을 포함하고, 여기서, 상기 제2 태스크는 상기 제2 태스크의 실행 시간을 추적하는 지연 타이머가 지연 임계치를 초과하는 시간 값에 도달하여 상기 제1 프로세스가 상기 제2 태스크의 실행을 인수하는 제2 프로세스를 생성하게 할 때까지 상기 제1 프로세스를 사용하여 실행을 시작하고, 여기서, 상기 제1 제어부는 상기 지연 임계치를 초과하는 상기 지연 타이머에 응답하여 상기 제1 프로세스가 상기 제2 프로세스를 생성 한 후 상기 제1 프로세스 상에서 실행을 시작하고, 상기 제2태스크의 실행을 인수하여 상기 제2 태스크와 제3 태스크가 동시에 실행되도록 하는 상기 제3 태스크에 대응하는 제3 서브루틴을 호출하기 위한 제2 함수 호출을 더 포함하며;

가능한 상태들의 집합으로부터 선택된 상기 제1 태스크의 상태를 지시하는 상태 정보를 저장하는 단계, 여기서 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하기 위해 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함함; 및

상기 제2 태스크에 대응하는 상기 제2 서브루틴을 실행하는 단계를 포함하되, 상기 제2 서브루틴은 상기 제2 태스크를 수행하는 제2 태스크부, 및 상기 저장된 상태 정보에 의해 지시된 상기 제1 태스크의 상태에 적어도 부분적으로 기반하여 상기 제2 태스크부의 실행을 제어하는 제2 제어부를 포함하는 것을 특징으로 하는 컴퓨터 판독가능한 저장 매체.

청구항 11

태스크(task)를 제어하는 컴퓨팅 시스템에 있어서

데이터 스토리지 시스템 내에 그래프 기반 프로그램 명세서(graph-based program specification)를 위한 데이터 구조를 저장하는 데이터 저장 시스템, 여기서 상기 그래프 기반 프로그램 명세서는 태스크들을 표시하는 복수의 노드들을 포함하고, 상기 그래프 기반 프로그램 명세서는 상기 노드들에 의해 표시된 복수의 태스크들 중 적어도 일부의 순서를 명시하는 노드들 사이의 방향성 엣지(directed edges)를 포함함; 및

상기 그래프 기반 프로그램 명세서에 의해 명시된 프로그램을 실행하도록 구성된 적어도 하나의 프로세서를 포함하되, 상기 실행은

제1 프로세스에 의해 제1 태스크에 대응하는 제1 서브루틴을 실행하는 단계, 여기서 상기 제1 서브루틴은 상기 제1 태스크를 수행하는 제1 태스크부를 포함하고,

상기 그래프 기반 프로그램 명세서에서 상기 제1 태스크에 대한 적어도 하나의 대응하는 후속 태스크들의 실행들을 제어하기위한 로직을 포함하는 상기 제1 서브루틴에 포함된 제1 제어부를 포함하고, 여기서, 상기 제1 제어부는 제2 태스크에 대응하는 제2 서브루틴을 호출하기위한 제1 함수 호출을 포함하고, 여기서, 상기 제2 태스크는 상기 제2 태스크의 실행 시간을 추적하는 지연 타이머가 지연 임계치를 초과하는 시간 값에 도달하여 상기 제1 프로세스가 상기 제2 태스크의 실행을 인수하는 제2 프로세스를 생성하게 할 때까지 상기 제1 프로세스를 사용하여 실행을 시작하고, 여기서, 상기 제1 제어부는 상기 지연 임계치를 초과하는 상기 지연 타이머에 응답하여 상기 제1 프로세스가 상기 제2 프로세스를 생성 한 후 상기 제1 프로세스 상에서 실행을 시작하고, 상기 제2태스크의 실행을 인수하여 상기 제2 태스크와 제3 태스크가 동시에 실행되도록 하는 상기 제3 태스크에 대응하는 제3 서브루틴을 호출하기 위한 제2 함수 호출을 더 포함하며;

가능한 상태들의 집합으로부터 선택된 상기 제1 태스크의 상태를 지시하는 상태 정보를 저장하는 단계, 여기서 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하기 위해 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함함; 및

상기 제2 태스크에 대응하는 상기 제2 서브루틴을 실행하는 단계를 포함하되, 상기 제2 서브루틴은 상기 제2 태스크를 수행하는 제2 태스크부, 및 상기 저장된 상태 정보에 의해 지시된 상기 제1 태스크의 상태에 적어도 부분적으로 기반하여 상기 제2 태스크부의 실행을 제어하는 제2 제어부를 포함하는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 12

태스크(task)를 제어하는 컴퓨팅 시스템에 있어서

데이터 스토리지 시스템 내에 그래프 기반 프로그램 명세서(graph-based program specification)를 위한 데이터 구조를 저장하는 수단, 여기서 상기 그래프 기반 프로그램 명세서는 태스크들을 표시하는 복수의 노드들을 포함하고, 상기 그래프 기반 프로그램 명세서는 상기 노드들에 의해 표시된 복수의 태스크들 중 적어도 일부의 순서를 명시하는 노드들 사이의 방향성 엣지(directed edges)를 포함함; 및

상기 그래프 기반 프로그램 명세서에 의해 명시된 프로그램을 실행하는 수단을 포함하되, 상기 실행은

제1 프로세스에 의해 제1 태스크에 대응하는 제1 서브루틴을 실행하는 단계, 여기서 상기 제1 서브루틴은 상기 제1 태스크를 수행하는 제1 태스크부를 포함하고,

상기 그래프 기반 프로그램 명세서에서 상기 제1 태스크에 대한 적어도 하나의 대응하는 후속 태스크들의 실행들을 제어하기위한 로직을 포함하는 상기 제1 서브루틴에 포함된 제1 제어부를 포함하고, 여기서, 상기 제1 제어부는 제2 태스크에 대응하는 제2 서브루틴을 호출하기위한 제1 함수 호출을 포함하고, 여기서, 상기 제2 태스크는 상기 제2 태스크의 실행 시간을 추적하는 지연 타이머가 지연 임계치를 초과하는 시간 값에 도달하여 상기 제1 프로세스가 상기 제2 태스크의 실행을 인수하는 제2 프로세스를 생성하게 할 때까지 상기 제1 프로세스를 사용하여 실행을 시작하고, 여기서, 상기 제1 제어부는 상기 지연 임계치를 초과하는 상기 지연 타이머에 응답하여 상기 제1 프로세스가 상기 제2 프로세스를 생성 한 후 상기 제1 프로세스 상에서 실행을 시작하고, 상기 제2태스크의 실행을 인수하여 상기 제2 태스크와 제3 태스크가 동시에 실행되도록 하는 상기 제3 태스크에 대응하는 제3 서브루틴을 호출하기 위한 제2 함수 호출을 더 포함하며;

가능한 상태들의 집합으로부터 선택된 상기 제1 태스크의 상태를 지시하는 상태 정보를 저장하는 단계, 여기서 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하기 위해 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함함; 및

상기 제2 태스크에 대응하는 상기 제2 서브루틴을 실행하는 단계를 포함하되, 상기 제2 서브루틴은 상기 제2 태스크를 수행하는 제2 태스크부, 및 상기 저장된 상태 정보에 의해 지시된 상기 제1 태스크의 상태를 적어도 부분적으로 기반하여 상기 제2 태스크부의 실행을 제어하는 제2 제어부를 포함하는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 13

제1항에 있어서, 상기 제2 태스크의 상기 제2 제어부는 상기 제1 태스크를 포함하는 다수의 태스크들의 그룹의 각각의 상기 상태에 적어도 부분적으로 기반하여 상기 제2 태스크부가 호출되는지 여부를 판단하는 로직을 포함하고;

여기서, 상기 다수의 태스크들의 각각의 상태는 상기 저장된 상태 정보로 지시되고,

상기 다수의 태스크들의 각각은 제어 방향성 엣지에 의해 상기 제2 태스크를 표시하는 노드에 연결된 노드로 표시되고,

상기 저장된 상태 정보는 상기 그래프 기반 프로그램 명세서에 명시된 상기 프로그램을 실행하는 동안 여러 번 업데이트되며, 여기서 상기 업데이트는 상기 다수의 태스크들의 각각의 상태를 지시하기 위해 상기 다수의 태스크들에 해당하는 서브 루틴들의 제어부들에 의해 수행되는 것을 특징으로 하는 방법.

청구항 14

제1항에 있어서, 상기 제2 태스크부는 상기 제2 태스크와 연관된 활성 카운터에 기반하여 호출되고, 상기 활성 카운터는 적어도 하나의 태스크의 상태 변경들에 응답하여 조정되는 카운터 값을 포함하고, 상기 제2 태스크를 실행하는 단계는 미리 결정된 값과 일치하는 상기 조정된 활성 카운터에 응답하여 상기 제2 태스크를 실행하는 것을 포함하는 것을 특징으로 하는 방법.

청구항 15

제1항에 있어서, 다수의 태스크들의 그룹의 각각의 상기 상태 정보는 상기 다수의 태스크들의 그룹의 각각에 의해 저장되고 상기 제2 서브 루틴에 포함된 상기 제2 제어부에 의해 액세스되는 것을 특징으로 하는 방법.

청구항 16

제10항에 있어서, 상기 제1 제어부는 상기 제2 태스크부가 호출되는지 여부를 판단하는 로직을 포함하는 것을 특징으로 하는 컴퓨터 판독가능한 저장 매체.

청구항 17

제16항에 있어서, 상기 로직은 상기 제2 태스크부가 플래그의 값을 기반으로 호출되는지 여부를 판단하는 것을 특징으로 하는 컴퓨터 판독가능한 저장 매체.

청구항 18

제16항에 있어서, 상기 로직은, 상기 제2 태스크를 표시하는 노드와 방향성 엣지에 의해 연결된 노드들에 의해 표시된 모든 태스크들의 상태들 중 적어도 일부에 기반하여 상기 제2 태스크부가 호출되는지 여부를 판단하는 것을 특징으로 하는 컴퓨터 판독가능한 저장 매체.

청구항 19

제10항에 있어서, 상기 제2 태스크부는 상기 제2 태스크와 연관된 활성 카운터에 기반하여 호출되고, 상기 활성 카운터는 적어도 하나의 태스크의 상태 변경들에 응답하여 조정되는 카운터 값을 포함하고, 상기 제2 태스크를 실행하는 단계는 미리 결정된 값과 일치하는 상기 조정된 활성 카운터에 응답하여 상기 제2 태스크를 실행하는 것을 포함하는 것을 특징으로 컴퓨터 판독가능한 저장 매체.

청구항 20

제10항에 있어서, 다수의 태스크들의 그룹의 각각의 상기 상태 정보는 상기 다수의 태스크들의 그룹의 각각에 의해 저장되고 상기 제2 서브루틴에 포함된 상기 제2 제어부에 의해 액세스 되는 것을 특징으로 하는 컴퓨터 판독가능한 저장 매체.

청구항 21

제11항에 있어서, 상기 제1 제어부가 상기 제2 태스크부가 호출되는지 여부를 판단하는 로직을포함하는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 22

제21항에 있어서, 상기 로직이 상기 제2 태스크부가 플래그의 값을 기반으로 호출되는지 여부를 판단하는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 23

제21항에 있어서, 상기 로직은, 상기 제2 태스크를 표시하는 노드와 방향성 엣지에 의해 연결된 노드들에 의해 표시된 모든 태스크들의 상태들 중 적어도 일부에 기반하여 상기 제2 태스크부가 호출되는지 여부를 판단하는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 24

제11항에 있어서, 상기 제2 태스크부는 상기 제2 태스크와 연관된 활성 카운터에 기반하여 호출되고, 상기 활성 카운터는 적어도 하나의 태스크의 상태 변경들에 응답하여 조정되는 카운터 값을 포함하고, 상기 제2 태스크를 실행하는 단계는 미리 결정된 값과 일치하는 상기 조정된 활성 카운터에 응답하여 상기 제2 태스크를 실행하는 것을 포함하는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 25

제11항에 있어서, 다수의 태스크들의 그룹의 각각의 상기 상태 정보는 상기 다수의 태스크들의 그룹의 각각에 의해 저장되고 상기 제2 서브 루틴에 포함된 상기 제2 제어부에 의해 액세스되는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 26

제12항에 있어서, 상기 제1 제어부는 상기 제2 태스크부가 호출되는지 여부를 판단하는 로직을 포함하는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 27

제26항에 있어서, 상기 로직은 상기 제2 태스크부가 플래그의 값을 기반으로 호출되는지 여부를 판단하는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 28

제26항에 있어서, 상기 로직은, 상기 제2 태스크를 표시하는 노드와 방향성 엣지에 의해 연결된 노드들에 의해 표시된 모든 태스크들의 상태들 중 적어도 일부에 기반하여 상기 제2 태스크부가 호출되는지 여부를 판단하는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 29

제12항에 있어서, 상기 제2 태스크부는 상기 제2 태스크와 연관된 활성 카운터에 기반하여 호출되고, 상기 활성 카운터는 적어도 하나의 태스크의 상태 변경들에 응답하여 조정되는 카운터 값을 포함하고, 상기 제2 태스크를 실행하는 단계는 미리 결정된 값과 일치하는 상기 조정된 활성 카운터에 응답하여 상기 제2 태스크를 실행하는 것을 포함하는 것을 특징으로 하는 컴퓨팅 시스템.

청구항 30

제12항에 있어서, 다수의 태스크들의 그룹의 각각의 상기 상태 정보는 상기 다수의 태스크들의 그룹의 각각에 의해 저장되고 상기 제2 서브 루틴에 포함된 상기 제2 제어부에 의해 액세스되는 것을 특징으로 하는 컴퓨팅 시스템.

발명의 설명

기 술 분 야

[0001] 본 출원은 2013년 4월 23일에 출원된 미국 출원 번호 61/815,052 에 대해 우선권을 주장한다.

[0002] 본 발명은 컴퓨팅 시스템에 의해 수행되는 태스크 제어에 관한 것이다.

배 경 기 술

[0003] 본 명세서는 컴퓨팅 시스템에 의해 수행되는 태스크 제어에 관한 것이다.

[0004] 컴퓨팅 시스템에 의해 수행되는 태스크를 제어하는 일부 기술에서는, 개별적인 태스크는 그 태스크를 위해 생성된 프로세스(process) 또는 스레드(thread)에 의해 수행되고, 상기 태스크 또는 스레드는 상기 태스크가 완료된 후에 종료한다. 컴퓨팅 시스템의 운영 체제 또는 운영 체제의 특징을 사용하는 다른 중앙화된 제어 개체는, 서로 다른 태스크들을 스케줄하거나 서로 다른 태스크들 간의 통신을 관리하기 위해 사용될 수 있다. 태스크들의 부분 순서를 정의하기 위해 어떠한 상류 태스크들(예를 들어, 태스크 A)은 다른 하류 태스크들(예를 들어, 태스크 B)이 시작하기 전에 완료하여야 함을 지시함으로써 제어 흐름도가 사용될 수 있다. 제어 흐름도에 따라서 태스크들을 수행하는 새로운 프로세스들의 생성을 관리하는 제어 프로세스가 존재할 수 있다. 제어 프로세스가 태스크 A를 수행하는 제어 프로세스 A를 생성하고 나서, 제어 프로세스는 운영 체제로부터 프로세스 A가 종료하였음에 대한 통지를 기다린다. 프로세스 A가 종료된 후에, 운영 체제는 제어 프로세서에 알리고, 다음으로 제어 프로세스는 태스크 B를 수행하는 프로세스 B를 생성한다.

발명의 내용

해결하려는 과제

[0005] 본 발명의 기술적 과제는 스케줄러로 또는 스케줄러로부터 스위칭하는 것에 대한 추가 부담 없이, 컴퓨팅 시스템은 작고 가벼우며 잠재적으로 동시에 발생할 수 있는 태스크들을 보다 효율적으로 실행함에 있다.

과제의 해결 수단

[0006] 일 양태에 따르면, 일반적으로, 컴퓨팅 시스템(computing system)에 의하여 수행되는 태스크(task) 제어 방법에 있어서 데이터 스토리지 시스템 내에 그래프 기반 프로그램 명세서(graph-based program specification)를 위한 데이터 구조를 저장하는 단계, 여기서 상기 그래프 기반 프로그램 명세서는 태스크들을 표시하는 복수의 노드들을 포함하고, 상기 그래프 기반 프로그램 명세서는 상기 노드들에 의해 표시된 복수의 태스크들 중 적어도 일부의 순서를 명시하는 노드들 사이의 방향성 엣지(directed edges)를 포함함; 및 적어도 하나의 프로세서를 사용하여, 상기 그래프 기반 프로그램 명세서에 의해 명시된 프로그램을 실행하는 단계를 포함하되, 상기 프로그램을 실행하는 단계는 제1 태스크에 대응하는 제1 서브루틴을 실행하는 단계, 여기서 상기 제1 서브루틴은 상기 제1 태스크를 수행하는 제1 태스크부를 포함함; 가능한 상태들의 집합으로부터 선택된 상기 제1 태스크의 상태를 지시하는 상태 정보를 저장하는 단계, 여기서 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하기 위해 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함함; 및 제2 태스크에 대응하는 제2 서브루틴을 실행하는 단계를 포함하되, 상기 제2 서브루틴은 상기 제2 태스크를 수행하는 제2 태스크부, 및 상기 저장된 상태 정보에 의해 지시된 상기 제1 태스크의 상태에 적어도 부분적으로 기반하여 상기 제2 태스크부의 실행을 제어하는 제어부를 포함한다.

[0007] 양태들은 아래의 특징들 중 하나 또는 그 이상을 포함할 수 있다.

[0008] 상류 노드로부터 하류 노드로 향하는 방향성 엣지는 상기 일부의 순서 내에서 상기 상류 노드에 의해 표시된 태

스크가 상기 하류 노드에 의해 표시된 태스크에 우선하는 것을 지시한다.

[0009] 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하였음을 나타내는 완료 상태, 상기 제1 태스크부가 상기 제1 태스크의 수행을 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함한다.

[0010] 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하는 과정(process)에 있음을 나타내는 활성 상태를 포함한다.

[0011] 상기 제어부는 상기 제2 태스크부가 호출되는지 여부를 판단하는 로직을 포함한다.

[0012] 상기 로직은 상기 제2 태스크부가 플래그(flag)의 값을 기반으로 호출되는지 여부를 판단한다.

[0013] 상기 로직은, 제2 태스크를 표시하는 노드와 방향성 엣지에 의해 연결된 노드들에 의해 표시된 모든 태스크들의 상태들 중 적어도 일부에 기반하여 상기 제2 태스크부가 호출되는지 여부를 판단한다.

[0014] 상기 로직은, 상기 제2 태스크를 표시하는 노드와 방향성 엣지에 의해 연결된 노드들에 의하여 표시되는 모든 태스크들이 유예 상태이면, 상기 제2 태스크부가 호출되지 않는 것으로 판단한다.

[0015] 상기 로직은 상기 제2 태스크를 표시하는 노드와 방향성 엣지에 의해 연결된 노드들에 의하여 표시되는 적어도 하나의 태스크가 완료 상태이고, 상기 제2 태스크를 표시하는 노드를 향하는 방향성 엣지에 의해 연결된 노드들에 의해 표시되는 태스크들 중 어느 것도 대기 상태가 아니다.

[0016] 다른 양태에 따르면, 일반적으로, 태스크(task)를 제어하는 컴퓨터 판독가능한 저장 매체 내에 저장된 컴퓨터 프로그램이 제공된다. 상기 상기 컴퓨터 프로그램은 컴퓨팅 시스템에 그래프 기반 프로그램 명세서(graph-based program specification)를 위한 데이터 구조를 저장하고, 여기서 상기 그래프 기반 프로그램 명세서는 태스크들을 표시하는 복수의 노드들을 포함하고, 상기 그래프 기반 프로그램 명세서는 상기 노드들에 의해 표시된 복수의 태스크들 중 적어도 일부의 순서를 명시하는 노드들 사이의 방향성 엣지(directed edges)를 포함함; 그리고 상기 그래프 기반 프로그램 명세서에 의해 명시된 프로그램을 실행하도록 구성된 명령들을 포함하되, 상기 실행은 제1 태스크에 대응하는 제1 서브루틴을 실행하는 단계, 여기서 상기 제1 서브루틴은 상기 제1 태스크를 수행하는 제1 태스크부를 포함함; 가능한 상태들의 집합으로부터 선택된 상기 제1 태스크의 상태를 지시하는 상태 정보를 저장하는 단계, 여기서 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하기 위해 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함함; 및 제2 태스크에 대응하는 제2 서브루틴을 실행하는 단계를 포함하되, 상기 제2 서브루틴은 상기 제2 태스크를 수행하는 제2 태스크부, 및 상기 저장된 상태 정보에 의해 지시된 상기 제1 태스크의 상태에 적어도 부분적으로 기반하여 상기 제2 태스크부의 실행을 제어하는 제어부를 포함한다.

[0017] 다른 양태에 따르면, 일반적으로, 태스크(task)를 제어하는 컴퓨팅 시스템에 있어서 데이터 스토리지 시스템 내에 그래프 기반 프로그램 명세서(graph-based program specification)를 위한 데이터 구조를 저장하는 데이터 저장 시스템, 여기서 상기 그래프 기반 프로그램 명세서는 태스크들을 표시하는 복수의 노드들을 포함하고, 상기 그래프 기반 프로그램 명세서는 상기 노드들에 의해 표시된 복수의 태스크들 중 적어도 일부의 순서를 명시하는 노드들 사이의 방향성 엣지(directed edges)를 포함함; 및 상기 그래프 기반 프로그램 명세서에 의해 명시된 프로그램을 실행하도록 구성된 적어도 하나의 프로세서를 포함하되, 상기 실행은 제1 태스크에 대응하는 제1 서브루틴을 실행하는 단계, 여기서 상기 제1 서브루틴은 상기 제1 태스크를 수행하는 제1 태스크부를 포함함; 가능한 상태들의 집합으로부터 선택된 상기 제1 태스크의 상태를 지시하는 상태 정보를 저장하는 단계, 여기서 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하기 위해 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함함; 및 제2 태스크에 대응하는 제2 서브루틴을 실행하는 단계를 포함하되, 상기 제2 서브루틴은 상기 제2 태스크를 수행하는 제2 태스크부, 및 상기 저장된 상태 정보에 의해 지시된 상기 제1 태스크의 상태에 적어도 부분적으로 기반하여 상기 제2 태스크부의 실행을 제어하는 제어부를 포함한다.

[0018] 다른 양태에 따르면, 일반적으로, 태스크(task)를 제어하는 컴퓨팅 시스템에 있어서 데이터 스토리지 시스템 내에 그래프 기반 프로그램 명세서(graph-based program specification)를 위한 데이터 구조를 저장하는 수단, 여기서 상기 그래프 기반 프로그램 명세서는 태스크들을 표시하는 복수의 노드들을 포함하고, 상기 그래프 기반 프로그램 명세서는 상기 노드들에 의해 표시된 복수의 태스크들 중 적어도 일부의 순서를 명시하는 노드들 사이의 방향성 엣지(directed edges)를 포함함; 및 상기 그래프 기반 프로그램 명세서에 의해 명시된 프로그램을 실행하는 수단을 포함하되, 상기 실행은 제1 태스크에 대응하는 제1 서브루틴을 실행하는 단계, 여기서 상기 제1

서브루틴은 상기 제1 태스크를 수행하는 제1 태스크부를 포함함; 가능한 상태들의 집합으로부터 선택된 상기 제1 태스크의 상태를 지시하는 상태 정보를 저장하는 단계, 여기서 상기 가능한 상태들의 집합은 상기 제1 태스크부가 상기 제1 태스크를 수행하기 위해 기다리는 중임을 나타내는 대기 상태, 및 상기 제1 태스크부가 상기 제1 태스크를 수행하는 것이 저지되었음을 나타내는 억제 상태를 포함함; 및 제2 태스크에 대응하는 제2 서브루틴을 실행하는 단계를 포함하되, 상기 제2 서브루틴은 상기 제2 태스크를 수행하는 제2 태스크부, 및 상기 저장된 상태 정보에 의해 지시된 상기 제1 태스크의 상태를 적어도 부분적으로 기반하여 상기 제2 태스크부의 실행을 제어하는 제어부를 포함한다.

발명의 효과

[0019] 양태들은 아래의 장점들 중 하나 또는 그 이상을 포함할 수 있다.

[0020] 컴퓨팅 시스템에 의하여 태스크들이 수행될 때, 태스크들을 실행하기 위한 새로운 과정들의 생성과 연관되고, 태스크 과정들 사이의 앞뒤로의 교체와 연관된 처리 시간, 그리고 스케줄러 또는 태스크 및 순서를 유지시키는 다른 중앙 처리의 처리 시간이 산출된다. 본 명세서에서 서술된 기술들은 계산적으로 효율적인 방법으로 새로운 과정들을 선택적으로 생성하거나, 또는 선택적으로 재사용될 수 있도록 한다. 컴파이러는 태스크들을 실행하기 위한 서브루틴들에 첨부된 상대적으로 작은 양의 코드를 기반하여 분산화된 스케줄링 메카니즘이 구비된 집중화된 스케줄러에 단독으로 의존하여야 할 필요를 회피할 수 있다. 태스크들의 완료는 컴퓨팅 시스템을, 동시 실행 및 조건부 논리를 허용하는 방법으로, 제어 흐름도와 같은 입력 제한에 따라 다른 태스크들을 자동으로 수행하도록 이끈다. 런타임에서 태스크 호출 함수와 연관된 컴파일러에 의해 발생된 코드는 카운터 및 플래그에 저장된 상태 정보에 기반하여 다른 태스크들을 수행할지 여부를 판단한다. 따라서, 컴파일러에 의해 발생된 코드는 런타임에서 태스크 서브루틴들의 호출을 제어하는 상태 머신을 효율적으로 구현한다. 스케줄러로 또는 스케줄러로부터 스위칭하는 것에 대한 추가 부담 없이, 컴퓨팅 시스템은 작고 가벼우며 잠재적으로 동시에 발생할 수 있는 태스크들을 보다 효율적으로 실행할 수 있다.

[0021] 본 발명의 다른 특징들 및 장점들은 이하의 명세서 및 청구항들로부터 보다 명백해질 것이다.

도면의 간단한 설명

[0022] 도 1은 컴퓨팅 시스템의 블록 다이어그램이다.

도 2A는 제어 흐름도의 다이어그램이다.

도 2B 내지 도 2D는 도 2의 제어 흐름도의 노드들에 대한 서브루틴들의 실행과 연관된 전체 과정에 대한 다이어그램이다.

도 3 및 도 4는 제어 흐름도의 다이어그램들이다.

발명을 실시하기 위한 구체적인 내용

[0023] 도 1은 태스크 제어 기술이 사용될 수 있는 컴퓨팅 시스템(100)의 일례를 도시한다. 시스템(100)은 태스크 명세서들(104)을 저장하는 스토리지 시스템(102), 태스크 명세서들을 태스크들을 수행하기 위한 태스크 서브투틴들로 컴파일하는 컴파일러(106), 및 로딩된 태스크 서브루틴들을 메모리 시스템(100) 내에서 실행하는 실행 환경(108)을 포함한다. 각각의 태스크 명세서(104)는 어떠한 태스크들이 수행되어야 하는지에 대한 정보 및, 서로 다른 태스크들 사이의 제한 순서들을 포함하여 이러한 태스크들이 언제 수행될 수 있는지에 대한 제한사항들을 부호화한다. 태스크 명세서들(104) 중 일부는 사용자(112)가 실행 환경(108)의 사용자 인터페이스(114)와의 상호작용에 의하여 구축된다. 상기 실행 환경(108)은 예를 들어, UNIX 운영 체제의 버전과 같은 적절한 운영 체제의 제어하에 있는 하나 또는 그 이상의 범용 컴퓨터 상에서 호스팅된다. 예를 들어, 실행 환경(108)은 다중 중앙 처리 장치(CPUs) 또는 프로세서 코어, 로컬(예를 들어, 대칭 멀티프로세싱(SMP) 컴퓨터와 같은 멀티프로세서 시스템), 또는 위치상 분산된(예를 들어, 클러스터들로써 연결된 다중 프로세서들 또는 대량병렬처리(massively parallel processing(MPP)) 시스템, 또는 원격, 또는 원격 분산된(예를 들어, 근거리 통신망(LAN)을 통해 연결된 다중 프로세서들 및/또는 광역통신망(WAN)), 또는 이들의 조합을 사용한 컴퓨터 시스템의 설정을 포함한다. 스토리지 시스템(102)을 제공하는 스토리지 장치(들)은 실행 환경(108)의 일부일 수 있다. 예를 들어, 실행 환

경(108)을 호스팅하는 컴퓨터에 연결된 스토리지 매체(예를 드렁, 하드 디스크) 상에 저장될 수 있거나, 또는 원격 연결(예를 들어, 클라우드 컴퓨팅 인프라구조에 의해 제공되는)을 통하여 실행 환경(108)을 관리하는 컴퓨터와 통신할 수 있다.

[0024] 도 2A는 컴퓨팅 시스템(100)에 의하여 수행되는 태스크들의 집합에 이용되는 일부의 순서를 정의하는 제어 흐름도(200)의 일례를 도시한다. 상기 제어 흐름도(200)에 의해 정의되는 일부의 순서는 저장된 태스크 명세서(104) 내에서 부호화된다. 일부 구현예에서는 사용자(112)는 제어 흐름도 내에 포함될 다양한 타입의 노드들을 선택하고, 이러한 노드들 중 일부를 연결하여 상기 연결된 노드들 사이에 제한 순서를 나타내도록 한다. 노드의 타입 중 하나는 태스크 노드이고 도 2A 내의 모서리가 각이 진 블록에 의해 나타내어진다. 각각의 태스크 노드는 수행되기 위한 서로 다른 태스크를 나타낸다. 제1 태스크 노드(방향성 링크의 출발점)로부터 제2 태스크 노드(방향성 노드의 도착점)로 연결된 방향성 링크는 제1 노드의 태스크가 제2 노드의 태스크가 시작되기 전에 완료되어야 한다는 순서의 제한을 한정한다. 다른 타입의 노드로는 중계 노드(junction node)가 있고, 도 2A에서 모서리가 둥근 블록에 의해 표시된다. 제어 흐름도가 행동 조건을 포함하지 않는 한, 중계 노드는 제한 순서를 한정하는데 도움을 준다. 하나의 입력 링크와 복수의 출력 링크가 연결되어 있는 중계 노드는 복수의 출력 링크에 의해 연결된 태스크 노드들의 태스크들이 시작되기 전에 하나의 입력 링크에 의해 연결된 태스크 노드의 태스크가 먼저 완료되어야 하는 제한 순서를 한정한다. 복수의 입력 링크와 하나의 출력 링크가 연결되어 있는 중계 노드는 하나의 출력 링크에 의해 연결된 태스크 노드의 태스크가 시작되기 전에 복수의 입력 링크에 의해 연결된 태스크 노드들의 태스크들이 먼저 완료되어야 하는 제한 순서를 한정한다. 태스크 노드는 복수의 입력 링크의 목적지일 수 있으며, 입력 링크에 의해 연결된 모든 태스크 노드들의 태스크들은 상기 태스크 노드의 태스크가 시작되기 전에 모두 안료되어야 하는 제한 순서를 한정한다. 조건부 행동에 의해 복수의 입력 링크들이 연결된 태스크 노드는 복수의 입력 링크들이 연결된 중계 노드와는 서로 다른 논리적 행동을 제공하고, 이하에서 보다 상세히 설명된다.

[0025] 제어 흐름도가 구축되고 나면, 컴파일러(106)는 태스크 명세서(104)를 컴파일하여, 상기 제어 흐름도에 의해 표시되는 태스크 정보 및 순서 정보를 부호화하고, 상기 태스크들을 수행하기 위한 명령을 발생시킨다. 상기 명령은 실행되기 위해 준비된 낮은 준위(low-level)의 기계화 코드의 형태일 수 있거나, 또는 궁극적으로 실행되기 위한 낮은 준위의 기계화 코드를 제공하기 위해 추가로 컴파일된 높은 준위의 코드의 형태일 수도 있다. 상기 발생된 명령은 각각의 태스크 노드에 대한 서브루틴(즉, “태스크 서브루틴”) 및 각각의 중계 노드에 대한 서브루틴(즉, “교차 서브루틴”)을 포함한다. 각각의 태스크 서브루틴은 대응하는 태스크를 수행하기 위한 태스크부(태스크 조직으로도 불림)를 포함한다. 태스크 노드는 컴파일러가 적절한 태스크부를 발생시킬 수 있도록 대응하는 태스크에 대한 일부 명세서를 포함한다. 예를 들어, 일부 구현예에 따르면, 태스크 노드는 호출되기 위한 특정한 함수, 실행될 프로그램, 또는 태스크부 내에 포함될 다른 실행가능한 코드를 식별한다. 일부 태스크 서브루틴은 제어 흐름도 내에서 다른 노드의 다음 서브루틴의 실행을 제어하는 제어부를 더 포함할 수 있다. 하류 노드에 연결되지 않은 태스크 노드에 대한 태스크 서브루틴은 완료된 이후에 다음의 태스크로 제어가 이동할 필요가 없기 때문에 제어부를 필요로 하지 않을 수 있다. 각각의 교차 서브루틴은 중계 노드의 목적이 제어 흐름에 대한 제한을 지정하기 위한 것으기 때문에 메인 조직에 제어부를 포함한다.

[0026] 제어부 내에 포함된 함수의 일례로, 제어 흐름도의 노드들에 연관된 상태 정보에 기반하여 다음 노드에 대한 서브루틴을 실행하기 위한 새로운 프로세스를 생성할지 여부를 결정하는 “chain(chain)” 함수가 있다. 상기 chain 함수의 인수는 다음의 노드를 식별한다. 아래의 표는 제어 흐름도(200)의 각각의 노드의 컴파일러에 의해 작성된 서브루틴들 내에 포함된 함수들의 일례를 나타낸다. 여기서, 태스크 서브루틴의 태스크부는 함수 호출 T#()에 의해 표시되고, 나머지 서브루틴은 제어부를 나타내도록 고려된다. (다른 일례로, 상기 태스크부는 복수의 함수 호출들을 포함할 수 있고, 태스크는 마지막 함수가 반환된 이후에 완료될 수 있다.) 교차 서브루틴들은 태스크부를 포함하지 않고, 따라서 제어부 전체로 이루어진다. 이 예시에서는, 서로 다른 함수 호출들은 호출될 순서에 따라 세미콜론에 의해 분리된다.

표 1

[0027]

노드(Node)	서브루틴(Subroutine)
태스크 노드(task node) T1	T1(); chain(J1)
중계 노드(junction node) J1	chain (T2); chain(T3)
태스크 노드(task node)T2	T2(); chain(J2)
태스크 노드(task node) T3	T3(); chain(J2)

중계 노드(junction node) J2	chain(T4)
태스크 노드(task node) T4	T4()

[0028] 태스크 명세서(104)가 컴파일되고 난 후, 컴퓨팅 시스템(100)은 발생된 서브루틴들을 실행 환경(108)의 메모리 시스템(110) 내로 로딩한다. 특정한 서브루틴이 호출될 때, 프로그램 카운터는 대응하는 서브루틴이 저장되는 메모리 시스템(110)의 주소 공간의 시작 부분의 주소에 대응하도록 설정된다.

[0029] 스케줄링된 시간에, 또는 사용자 입력 또는 미리 정해진 이벤트에 대한 응답으로, 컴퓨팅 시스템(100)은 제어 흐름도의 루트를 나타내는 로드된 서브루틴들 중 적어도 하나의 실행을 시작한다. 예를 들어, 제어 흐름도(200)에서, 컴퓨팅 시스템(100)은 태스크 노드 T1에 대한 태스크 서브루틴을 실행하는 프로세스를 생성한다. 상기 서브루틴의 실행이 시작됨에 따라 상기 프로세서는 태스크 노드 T1의 태스크를 수행하는 태스크부를 먼저 호출할 것이고, 다음으로, 태스크부가 반환되고 난 후(태스크 노드 T1의 태스크 수행을 지시하는), 프로세스는 서브루틴의 제어부 내의 chain 함수를 호출할 것이다. 특정한 노드에 대한 서브루틴을 실행하기 위한 새로운 프로세스를 생성할지를 결정하는 chain 함수에 의해 수용되는 상태 정보는 인수로써 특정한 노트로 호출된 이전의 chain 함수에 대한 이력을 캡쳐하는 정보이며, 이하에서 보다 상세히 설명된다.

[0030] 이력 정보는 서로 다른 노드들과 연관된 활성 카운터에 의해 유지될 수 있다. 상기 카운터의 값은, 예를 들어, 메모리 시스템(110)의 일부, 또는 다른 워킹 스토리지 내에 저장될 수 있다. 제1 프로세스가 생성되기 전에, 각각의 노드에 대한 활성 카운터의 값은 노드 내로 입력ㄱ되는 수에 의해 초기화된다. 따라서, 제어 흐름도(200)에 대하여, 아래의 6개의 활성 카운터 값이 초기화된다.

표 2

[0031]

노드(Node)	활성 카운터 값(Activation Counter Value)
태스크 노드(task node) T1	0
중계 노드(junction node) J1	1
태스크 노드(task node) T2	1
태스크 노드(task node) T3	1
중계 노드(junction node) J2	2
태스크 노드(task node) T4	1

[0032] 태스크 노드 T1이 입력 링크를 갖고 있지 않기 때문에, 이에 대한 활성 카운터는 0으로 초기화된다. 다시 말해, 초기 노드에 대해 입력 링크가 없는 경우에는 상기 노드와 연결된 활성 카운터를 필요로 하지 않는다. 입력 링크에 의해 연결된 서로 다른 노드들의 제어부는 하류 링크된 노드의 활성 카운터를 감소시키고, 상기 감소된 값에 기반하여 동작을 결정한다. 일부 구현예에 따르면, 카운터들에 접근하는 함수들은 카운터를 자동으로 감소시키는 최소 단위의 동작을 사용하고 상기 감소 동작의 전후의 카운터의 값을 판독한다(예를 들어, 최소 단위의 “감소 및 테스트 동작”). 일부 시스템에서, 이러한 동작들은 시스템의 내티브(native) 명령에 의해 지원된다. 다시 말해, 카운터의 값이 0에 도달할 때까지 카운터를 감소시키는 대신에, 카운터는 0에서 시작하고, 함수는 카운터의 값이 노드로 입력되는 입력 링크의 수로 초기화되는 미리 정해진 임계치에 도달할 때까지 증가될 수 있다(예를 들어, 최소 단위의 “증가 및 테스트 동작”).

[0033] chain 함수에 대한 호출 “chain(N)”은 노드 N에 대한 활성 카운터를 감소시키고, 감소된 값이 0인 경우, chain 함수는 새로 생성된 프로세스에 의해 노드 N에 대한 서브루틴의 실행을 트리거하고, 반환한다. 만약, 감소된 값이 0보다 큰 경우, chain 함수는 서브루틴에 대한 실행의 트리거 또는 새로운 프로세스에 대한 생성 없이 단순히 반환된다. 서브루틴의 제어부는 표 1의 중계 노드 J1의 교차 서브루틴과 같이 chain 함수에 다중 호출을 포함할 수 있다. 제어부의 마지막 함수가 반환되고 난 후, 서브루틴을 실행하는 프로세스는 종료되거나, 일부 함수 호출(예를 들어, 이하에서 설명될 “chainTo” 함수에 의하여”)을 위하여, 프로세스는 다른 서브루틴을 계속할 수 있다. 이러한 새로운 프로세스에 대한 조건부 생성은 새로운 프로세스 생성을 관리하기 위한 스케줄로 프로세스로 또한 이러부터 스위치의 필요 없이도, 요구되는 일부 순서에 따라 태스크 서브루틴의 실행(잠재적으로 동시발생되도록)을 가능하게 한다.

[0034] 표 1의 서브루틴들에 대하여, 태스크 서브루틴 T1의 태스크부가 J1 노드의 활성 카운터를 1에서 0으로 감소시킨

후에, chain 함수 “chain(J1)”의 호출은 chain 함수 “chain(T2)” 및 “chain(T3)”에 대한 호출을 포함하는 교차 서브루틴을 실행시킨다. 각각의 이러한 호출들은 노드 T2 및 T3에 대한 각각의 활성 카운터들을 1에서 0으로 감소시키고, 노드 T2 및 T3에 대한 태스크 서브루틴들을 실행시킨다. 두 개의 태스크 서브루틴들은 모두 노드 J2에 대한 활성 카운터를 감소시키는 “chain(J2),”를 호출하는 제어부를 포함한다. T2 및 T3 노드들 중 먼저 끝나는 태스크 조직들은 노드 J2에 대한 활성 카운터를 2에서 1로 감소시키는 chain 함수를 호출한다. 두 번째로 끝나는 태스크부는 노드 J2에 대한 활성 카운터를 1에서 0으로 감소시키는 chain 함수를 호출한다. 따라서, 마지막 태스크를 완료시키는 것은 chain 함수 “chain(T4)”에 대하여 마지막 호출을 이끄는 노드 J2에 대한 교차 서브루틴을 실행시키고, 노드 T4에 대한 태스크 서브루틴의 실행을 개시하는 노드 T4에 대한 활성 카운터를 1에서 0으로 감소시킨다. 노드 T4에 대한 태스크 서브루틴의 제어부가 존재하기 않기 때문에 노드 T4에 대한 태스크부가 반환되고 난 후, 제어 흐름은 완료된다.

[0035] 표 1의 서브루틴의 예에서, 제어 흐름도(200)의 각각의 노드의 서브루틴에 대한 새로운 프로세스가 생성된다. 일부 효율은 중앙 태스크의 모니터링 또는 스케줄링 프로세스의 필요 없이 새로운 프로세서를 생성할지에 대하여 결정하는 제어부를 포함하는 각각의 프로세서에 대한 서브루틴에 의하여 획득되는 반면, 제어부에 대한 소정의 컴파일러 최적화에 의해 더 좋은 효율성을 얻을 수 있다. 예를 들어, 하나의 컴파일러 최적화의 예로써, 제1 서브루틴의 제어부 내에서 chain 함수에 대한 단일 호출이 존재한다면, 다음 서브루틴(즉, chain 함수의 인자)이 제1 서브루틴을 실행하는 동일한 프로세스 내에서 실행될 수 있고(활성 카운터가 0에 도달할 때), 새로운 프로세스는 생성될 필요가 없다. 이것을 달성하기 위한 한가지 방법은 컴파일러가 노드에 대한 마지막 출력 링크에 대한 다른 함수 호출(예를 들어, “chain” 함수 대신에 “chainTo” 함수)을 명확하게 생성하도록 하는 것이다. chainTo 함수는 활성 카운터가 0일 때 서브루틴을 실행하기 위한 새로운 프로세스를 생성하는 대신에 그것의 인수로 서브루틴을 실행하기 위한 동일한 프로세스를 일으킨다는 점을 제외하고는 chain 함수와 동일하다. 만약 노드가 하나의 출력 링크를 갖는다면, 그것의 컴파일된 서브루틴은 chainTo 함수에 대한 단일 호출이 구비된 제어부를 가질 것이다. 하나의 노드가 다중 출력 링크를 갖는다면, 그것의 컴파일된 서브루틴은 chain 함수에 대한 하나 또는 그 이상의 호출과 chainTo 함수에 대한 하나의 호출이 구비된 제어부를 가질 것이다. 이것은 독립된 프로세스들 내에 생성되는 서브루틴의 수와 연관된 시작 오버헤드를 감소시킨다. 표 3은 이러한 컴파일러 최적화를 사용한 제어 흐름도(200)에서 생성될 수 있는 서브루틴들의 일례를 나타낸다.

표 3

[0036]

노드(Node)	서브루틴(Subroutine)
태스크 노드(task node) T1	T1(); chainTo(J1)
중계 노드(junction node) J1	chain(T2); chainTo(T3)
태스크 노드(task node) T2	T2(); chainTo(J2)
태스크 노드(task node) T3	T3(); chainTo(J2)
중계 노드(junction node) J2	chainTo(T4)
태스크 노드(task node) T4	T4()

[0037] 표 3의 예에서, 제1 프로세스는 노드 T1 및 J1의 서브루틴을 실행시키고, 제1 프로세스가 노드 T3의 서브루틴을 실행을 계속하는 동안에, 노드 T2의 서브루틴을 실행시키기 위한 새로운 프로세스가 생성된다. 이러한 두 프로세스 중 하나가 각각의 태스크부로부터 처음으로 반환되는 것은 중계 노드 J2의 활성 카운터를 처음으로 감소시키고(2에서 1로), 종료한다. 태스크부로부터 두번째로 반환되는 프로세스는 중계 노드 J2의 활성 카운터를 1에서 0으로 감소시키고, “chainTo(T4),”의 함수호출인 교차노드 J2의 서브루틴을 실행시키고, 태스크 노드 T4의 서브루틴을 마지막으로 실행시킴으로써 계속된다. 도 2B는 제어 흐름도(200) 내에서 서로 다른 노드들에 대한 서브루틴들을 실행시키는 제1 및 제2 프로세스 전체의 예를 도시하고, 여기서 노드 T3의 태스크는 노드 T2의 태스크보다 먼저 끝난다. 선들에 연결된 점들은 서로 다른 노드(대시 기호로 된 선들에 의해 연결된 지점들)들의 서브루틴의 실행에 대응하는 프로세스들을 나타낸다. 점들 사이에 연결된 선분의 길이는 필수적으로 소요 시간에 비례하지 않고, 서로 다른 서브루틴들이 실행되었음을 상대적으로 나타내고, 새로운 프로세스가 생성된 위치를 나타낸다.

[0038] 잠재적으로 효율성을 향상시킬 수 있는 수정의 예로는 특별한 서브루틴이 동시에 수행됨으로부터 유리함을 지시하는 임계치가 충족될 때까지 새로운 프로세스의 생성을 지연시키는 것이다. 서로 다른 프로세서에 의해 복수의 서브루틴들의 동시에 실행시키는 것은 각각의 서브루틴들이 완료되는 데에 상당한 시간이 소요되는 경우에만 유리하다. 반면에, 어떠한 서브루틴들이 다른 서브루틴들에 비하여 완료되는데 상당히 짧은 시간이 소요된다면,

그 서브루틴은 큰 손실 없이 다른 서브루틴들과 순차적으로 실행될 수 있다. 지연된 생성 메커니즘은 복수의 태스크들에 상당히 긴 시간을 허용하고, 동시에 작동하는 프로세서들에 의하여 수행됨으로써 함께 수행될 수 있지만, 더 짧은 태스크들에 대한 새로운 프로세스의 생성을 막는다.

[0039] 지연 생성을 사용하는 chain 함수에 대한 다른 구현예에서, chain 함수는 chainTo 함수와 마찬가지로, 그것의 인수로 서브루틴의 실행을 시작하는 동일한 프로세스를 발생시킨다. 하지만, chainTo 함수와는 달리, 타이머가 서브루틴을 실행하는데 걸리는 시간을 추적하고, 임계 시간이 초과되면, chain 함수는 서브루틴의 실행을 계속하기 위하여 새로운 프로세스를 생성한다. 최초 프로세스는 상기 서브루틴이 완료된 것처럼 계속할 수 있고, 제2 프로세스는 제1 프로세스가 남기고 간 서브루틴의 실행을 이어받는다. 이것을 달성하기 위하여 사용될 수 있는 하나의 메커니즘은 제2 프로세스가 제1 프로세스로부터 서브루틴의 스택 프레임을 상속받는 것이다. 상기 실행되는 서브루틴에 대한 스택 프레임은 특정한 명령을 지시하는 프로그램 카운터를 포함하고, 다른 값들은 서브루틴의 실행에 관한 값을 포함한다(예를 들어, 로컬 변수 및 등록 값). 이 예에서, T2에 대한 태스크 서브 루틴의 스택 프레임은 프로세스가 T2에 대한 태스크 서브루틴의 완료 후에 교차 서브루틴 J1으로 복귀하는 것을 가능하게 하는 리턴 포인터(return pointer)를 포함할 수 있다. 지연된 생성 타이머가 초과되면, 새로운 프로세스가 생성되고, 상기 새로운 프로세스는 T2에 대한 태스크 서브루틴의 기존 스택 프레임과 연관되고, 제1 프로세스는 즉시 J1(“chainTO(T3)” 를 호출하기 위해)에 대한 교차 서브루틴으로 복귀한다. 새로운 프로세스가 T2에 대한 태스크 서브루틴의 완료 후에 J1에 대한 교차 서브루틴으로 복귀할 필요가 없기 때문에, 상속된 스택 프레임 내의 리턴 포인터는 제거된다(즉, 널 아웃된다). 따라서, 지연된 생성은 경우에 새로운 프로세스를 생성과 관련된 오버 헤드를 후속 태스크가 태스크가 빠른 (구성 가능한 임계 값에 대해) 인 경우에 새로운 프로세스를 생성하는 오버 헤드없이 수행되도록하기위한 서브 루틴을 가능하게하고, 감소 하는 작업은 더 이상 기존의 스택 프레임을 상속하는 것이다.

[0040] 도 2C는 제어 흐름도(200) 내의 서로 다른 노드들에 대한 서브루틴들을 실행하는 제1 및 제2 프로세스의 전체를 도시하고, 노드 T2의 태스크는 지연된 생성 임계값보다 길다. 생성 임계 값에 도달 할 때, 스폰 1 노드 T2의 태스크를 수행하는 서브 루틴의 스택 프레임을 상속하는 서브 루틴의 실행을 계속한다 (2), 처리한다. 이 예에서, (공정 (1)에 의해 수행됨) 노드 T3의 태스크 노드 T2 (공정 (1)에 의해 시작되고 (2)에 의해 처리 완료)의 태스크를 완료하기 전에 종료한다. 그래서,이 예에서는 2-1 J2의 활성화 카운터를 감소 (후 종료) 1 공정이며, 공정 수행이 결과 1-0 J2의 활성화 카운터를 감소 공정 2 작업 노드 (T4)의 작업. 그러한 동시성은 전체 효율에 기여하는 것으로 결정된 후,이 예에서, 노드 T2와 T3의 노드 태스크의 태스크의 동시 실행이 허용된다.

[0041] 도 2D는 제어 흐름도(200) 내의 서로 다른 노드들에 대한 서브루틴들을 실행하는 하나의 프로세스의 전체를 도시하고, 노드 T2의 태스크는 지연된 생성 임계값보다 짧다. 이 예에서, (공정 (1)에 의해 수행됨) 노드 T2의 태스크 노드 T3의 작업 (도 1 공정에 의해 수행됨)하기 전에 종료한다. 그래서,이 예에서, 공정 (1)는 노드 T2의 작업을 완료 한 후 2-1 J2의 활성화 카운터를 감소시키고, 공정 (1) 프로세스의 결과, 노드 T3의 작업을 완료한 후 1-0 J2의 활성화 카운터를 감소 태스크 1 노드 T4의 작업을 수행하는 단계를 포함한다. 이 노드 T2의 태스크 빠르게 완료 할 수 있다고 판정 된 후,이 예에서, 노드 T2와 노드 T3의 태스크의 동시 실행은 제 2 프로세스를 생성 할 필요를 회피함으로써 얻은 효율을 희생한다.

[0042] 제어 흐름 그래프에 포함될 수있는 또 다른 유형의 노드는 도 1에 도시 된 제어 흐름 그래프 (300)에 원으로 표시 조건 노드이다. 3. 노드 상태는 노드 상태의 출력에 연결된 노드에 태스크의 태스크가 수행 할 수 있는지의 여부를 결정하기위한 조건들을 정의한다. 런타임에 정의 된 조건이 참이면 그 조건 노드를지나 하류 진행을 제어하지만, 런타임에 정의 된 조건이 거짓 인 경우, 흐름은 그 조건 노드를 더 이상 진행하지 않는 제어 할 수 있다. 조건이 거짓 인 경우 (다른 거짓 조건 노드에 의해 차단 자체와없는) 그 태스크 노드로 이어질 제어 흐름 그래프를 통해 다른 경로가있는 경우, 다음 조건 노드의 하류 태스크 노드의 작업 만 수행된다.

[0043] 컴파일러는 각각의 노드에 대한 상태 "조건 서브 루틴"을 생성하고, 또한 조건 노드의 하류에 다른 특정 노드의 서브 루틴을 수정하는 조건 노드에 의해 정의 된 조건을 사용한다. 예를 들면, 컴파일러는 제어 흐름 그래프에 의해 정의 제어의 흐름을 따라 실행시에 적용되는 "스킵 메카니즘"에 대한 지시를 생성 할 수있다. 스킵 메카니즘에서, 각각의 노드는 실행된다 대응 태스크 부분 (만약 있다면) 여부를 제어하는 "플래그를 건너 뛰고"관련있다. 스킵 플래그를 설정하면, 작업 구간의 다음 실행 (노드 "억제"상태 인 상태)을 억제하고,이 억제가함으로써 제어부 넣었다 적절한 제어 코드가 다른 태스크에 전파 될 수있다 컴파일러. 앞의 예에서, 서브 루틴의 태스크 태스크 부는 제어부 선행. 다음의 예에서, 어떤 작업 서브 루틴의 제어부는 태스크 부 후에 발생 및 제어 코드 (또한 "프롤로그"라고 함) 태스크 부 전에 발생 제어 코드를 포함한다 (또한 "에필로그"라고 함). 예를 들어, 컴파일러는 프롤로그에 포함이 스킵 메커니즘을 구현하기 위해 조건부 명령어 (즉, 코드는 작업 섹션 전에 실행

되는) (예를 들어, 문 경우) 인수로 호출되는 "이동 기능"을 호출하는 다운 스트림 노드를 식별한다. 컴파일러는 (즉, 코드가 작업 섹션 후 실행되는) 에필로그에 포함 체인 또는 chainTo 함수를 호출한다. 어떤 경우에는, 인해 스킵 플래그의 값으로 표현 저장된 상태 만 프롤로그 실행될 수 및 작업 부와 에필로그 모두 생략 될 수있다. 표 4는 제어 흐름 그래프 (300)에 대해 생성 될 서브 루틴의 예를 나타낸다.

표 4

[0044]

노드(Node)	서브루틴(Subroutine)
태스크 노드(task node) T1	T1(); chainTo(J1)
중계 노드(junction node) J1	chain(C1); chain(C2); chainTo(J3)

상태 노드(condition node) C1	if (<condition1>)
	chainTo(T2)
	else
	skip(T2)

상태 노드(condition node) C2	if (<condition2>)
	chainTo(T3)
	else
	skip(T3)

태스크 노드(task node) T2	if (skip)
	skip(J2)
	else
	T2(); chainTo(J2)

태스크 노드(task node) T3	if (skip)
	skip(J2)
	else
	T3(); chainTo(J2)

중계 노드(junction node) J2	if (skip)
	skip(T4)
	else
	chainTo(T4)

태스크 노드(task node) T4	if (skip)
	skip(J3)
	else
	T4(); chainTo(J3)

중계 노드(junction node) J3	if (skip)
	skip(T5)
	else
	chainTo(T5)

태스크 노드(task node) T5	if (skip)
	return
	else
	T5()

[0045] chain 및 chainTo 함수와 마찬가지로, skip 함수 “skip(N)”는 인수(node N)의 활성 카운터를 감소시키고, 만약 감소된 값이 0인 경우, 대응하는 서브루틴을 수행한다. 본 예에서 0 인 경우, 해당 서브 루틴을 실행하고, 스킵 기능의 동작을 이하 새로운 산란하지 않고 동일한 과정을 계속 사용하여 chainTo 기능하지만, 컴파일러는 체인으로 동작 스킵 기능의 두 가지 버전을 사용할 수 있으며 chainTo 기능과 유사한 방식으로 태스크 산란을 제어하기 위해 수행. 컴파일러는 서브 루틴 설정되어 실행중인 노드의 스킵 플래그 (즉, 부울 참 값으로 평가) 하고있는 경우,이 다운 스트림 노드에 이동 호출되도록 태스크를 호출없이 조건부 노드의 다운 스트림 노드에 대한 서브 루틴을 생성 섹션 및 스킵 플래그가 클리어되어 있으면 (노드가 태스크 노드 인 경우) 태스크 부 전화 않고 하류 노드 체인을 호출 (즉, 부울 거짓 값으로 평가). 대안 적으로, 컴파일러는 조건부 노드의 하류에 있는 노드를 결정하는 데 필요없이 기본적으로 서브 루틴의 제어부에서 조건문을 포함 할 수있다. 특히, 스킵 플래그를 확인하는 "경우"문 (즉, 스킵 플래그의 불필요한 검사 발생할 수 있지만)를 포함 할 것인지 여부를 결

정하는 데 컴파일러없이 모든 노드의 서브 루틴에 대해 기본적으로 포함될 수있다.

[0046] 제어 흐름 그래프의 노드 상태가 있는 경우 다수의 입력 노드의 유형에 따라 런타임 논리 동작을 취득하여, 다음 노드. 다중 입력 링크와 단일 출력 링크와 접합 노드는 입력 링크에 의해 연결된 적어도 하나의 입력 노드는 서브 루틴이 경우 체인 호출 (아닌 스킵 호)을 실행되어 있어야되도록, 논리 "OR"동작에 대응 출력 링크로 연결되어 출력 노드는 서브 루틴 호출 체인 (그리고 스킵 통화)의 인수를 가질 수있다. 다중 입력 링크와 단일 출력 링크가 태스크 노드는 입력 링크로 연결된 입력 모든 노드들이 서브 루틴 경우 체인 호출 (아닌 스킵 호)을 실행 가져야 이러한 논리 "AND"조작에 대응 태스크 노드의 서브 루틴 호출 체인 (그리고 스킵 통화)의 인자이어야 한다.

[0047] 이 논리 동작을 보장하기 위해, 노드와 관련된 스킵 플래그가 설정되고, 소정의 규칙에 따라 실행시에 해제된다. 스킵 플래그의 초기 값은 제어 흐름 그래프의 노드 중 임의의 서브 루틴의 실행 이전에 발생하는 초기화 단계 동안 제공하고, 또한 노드의 유형에 좌우된다. 컴파일러는 노드의 종류에 따라 다른 동작을 스킵 기능 및 체인 기능의 다른 버전을 사용한다. 다음 노드 N의 스킵 플래그를 변화시키기위한 소정의 규칙 세트 및 컴파일러에 의해 사용되는 기능의 서로 다른 버전의 동작의 일례이다.

[0048] ? 다중 입력 접합 노드 (OR 연산)의 경우 : 이동 플래그 처음 스킵 플래그를 변경하지 않는다. skip_OR (N), 설정, chain_OR (N)는 플래그를 건너 뛴다.

[0049] ? 다중 입력 작업 노드 (연산)의 경우 : 이동 플래그 처음 클리어, skip_AND (N) 세트 플래그를 건너, chain_AND (N)은 스킵 플래그를 변경하지 않는다.

[0050] ? 단일 입력 노드의 경우 : 플래그가 처음에 설정되어 스킵, 스킵 플래그를 변경하지 않는다. (N) 이동, 체인 (N)는 플래그를 건너 뛴다.

[0051] chainTo 함수의 동작은 스킵 플래그에 대해 체인 함수와 동일하다. 단일 입력 노드에 대해 OR 동작과 AND 연산의 동작은 동일하고, 하나는 (예를 들어이 예에서는 OR 연산의 동작 등) 사용될 수있다. 이 규칙 세트는 출발 노드 (들) (임의의 입력없이 링크 즉, 노드) (초기 값이 이미 해제되어 있지 않은 경우) 그 스킵 플래그가 클리어있다.

[0052] 제어 흐름 그래프 (300), 노드 C1에 대한 조건이 참인 경우를 고려 노드 C2에 대한 조건이 거짓이고, 노드 T3위한 태스크 노드 C2 상태 검사가 완료되기 전에 완료 : 노드 T3위한 서브 루틴 것 노드 J2의 스킵 플래그를 클리어하고 (2-1) 노드 J2에 대한 활성을 감소 카운터 쇄 논리 (논리를 건너 반대), 다음과; 다음 노드 (T4)의 서브 루틴 (스킵 플래그를 변경하지 않는다). 논리를 건너 뛰고 다음 및 노드 J2의 스킵 플래그가 이후 체인 (T5) 리드 (1-0) 노드 (J2),의 활성화 카운터를 감소 노드 (T3)의 서브 루틴에 의해 삭제.

[0053] 다른 규칙들도 가능하다. 다음 노드 N의 스킵 플래그를 변화시키기위한 소정의 규칙 세트 및 컴파일러에 의해 사용되는 기능의 서로 다른 버전의 동작의 다른 예이다.

[0054] ? 접합 노드의 경우 : 플래그가 처음에 설정되어 이동, skip_J (N)은 스킵 플래그를 변경하지 않는다. chain_J (N)는 플래그를 건너 뛴다.

[0055] ? 작업 노드 또는 조건부 노드의 경우 : 이동 플래그 처음 클리어, 이동 (N) 세트 플래그를 건너, 체인 (N)은 스킵 플래그를 변경하지 않는다.

[0056] 이 규칙 세트는 출발 노드 (들) (즉, 임의의 링크없이 입력 노드)은 또한 (초기 값이 이미 해제되어 있지 않은 경우) 그 스킵 플래그를 클리어 할 것이다.

[0057] 컴파일러는 임의로 조건문 또는 제어 흐름 그래프의 분석에 기초하여 서브 루틴의 제어부의 다른 명령들의 다양한 최적화를 수행 할 수 있다. 예를 들어, 제어 흐름 그래프 (300)에서, 그 태스크 노드 T5의 작업없이 접합 노드 J1 사이의 링크가 존재하기 때문에 조건 노드들 C1 및 C2의 조건이 참 또는 거짓인지의 스킵되지 않도록 결정될 수있다 결국 태스크 노드 T5의 태스크 실행으로 이어질 것이다 접합 노드 J3. 따라서, 컴파일러가 스킵 플래그의 검사를 피할 태스크 노드 T5위한 서브 루틴을 생성 할 수 있으며, 단순히 그 태스크 부 T5 ()를 호출한다. 다른 최적화는 중간 스킵 플래그 점검을 남겨 둠으로써, 예를 들어, 컴파일러에 의해 만들어 기능은 어떤 다른 입력만큼, 제어 흐름 그래프의 전체 섹션은 조건부 노드 후 스킵되어야하는 경우에 호출을 스킵 할 수있다 다운 스트림 노드는 스킵 부 적절히 처리 한 후 (즉, 다운 스트림 노드의 카운터를 감분 스킵 구간 중간 통화가 포함 된 경우는 감소했을 횟수).

[0058] 도 4는 다중입력 태스크 노드 T3을 포함하는 단순한 제어 플로우 그래프(400)의 일례를 도시하고, 여기서 상기 태스크 노드들 T1 및 T2는 각각 상태 노드(C1 및 C2 각각)를 따른다. 이 예에서, 태스크 노드 T3는 노드 T3의 작업 노드들 T1 및 T2는 모두가 실행되어야하는 태스크 (아닌 스킵)되도록 실행되는 논리 AND 연산에 대응한다. 표 5는 제어 흐름 그래프 (400)에 대해 생성 될 서브 루틴의 예를 나타낸다.

표 5

[0059]

노드(Node)	서브루틴(Subroutine)
중계 노드(junction node) J1	chain(C1); chainTo(C2)

상태 노드(condition node) C1	if (<condition1>)
	  chainTo(T1)
	else
	  skip(T1)

상태 노드(condition node) C2	if (<condition2>)
	  chainTo(T2)
	else
	  skip(T2)

태스크 노드(task node) T1	if (skip)
	  skip(T3)
	else
	  T1(); chainTo(T3)

태스크 노드(task node) T2	if (skip)
	  skip(T3)
	else
	  T2(); chainTo(T3)

태스크 노드(task node) T3	if (skip)
	  return
	else
	  T3()

[0060] 일부 구현 예에서, 중계 노드 (또는 다른 노드)에 대한 입력 노드의 특성에 의존하는 논리 연산의 다양한 종류를 제공하도록 구성 될 수있다. 모든 입력이 "선택적"입력으로 지정되는 경우, 예를 들어, 노드는 논리 OR 연산을 모든 입력은 "필수"입력으로 지정되는 경우에 논리 AND 연산을 제공하도록 구성 될 수있다. 어떤 입력이 "필요"로 지정되고, 일부 입력은 "선택적"으로 지정되는 경우, 소정의 규칙 세트는 노드에 의해 수행되는 논리 연산의 조합을 해석하는 데 사용될 수 있다.

[0061] 일부 구현 예에서, 제어부의 제어 코드로 나타낸 바와 같이, 제어부 내의 제어 코드에 의해 표시된, 컴파일러에 의해 구축된 상태 머신은 상기 서술된 액티스 카운터 및 스킵 플래그들을의 값에 의해 표현된 값들과 등가의 상태들을 나타내는 다른 상태 변수들을 사용할 수 있다. 예를 들어, 상태들의 집합은 "보류", “활성화”, "완료" 및 "억제"를 포함할 수 있다. 이 예에서, 작업은 보류 상태에서 시작합니다. 상태 사이의 허용 전환은 다음과 같습니다 : (1) 억제에 계류중인; (2) 활성에 보류 및 완료 활성. 활성화 카운터없이 활성화 카운터 시스템의 동작과 시스템 사이에 약간의 차이가 있습니다. 대신 하류 태스크의 서문의 실행의 기동 카운터가 0에 도달 한 것으로 판단 상류 작업 에필로그 의해 트리거되고, 작업의 실행이 적어도 하나의 상류 태스크가 완료 상태 인 것으로 판정 프롤로그 의해 트리거 될 아무도는 보류 상태에 없습니다. 그 프롤로그 모든 상류 작업이 억제 된 상태에 있다고 판단 할 경우 또는 해당 작업의 실행은 억제 될 것이다.

[0062] 활성 카운터를 사용하는 일부 그현 예는 프롤로그를 호출할 필요가 있는 횟수를 감소시킨다. 예를 들어, 활성화 카운터없이, 다운 스트림 작업의 프롤로그 직접 입력 링크로 연결된 모든 상류 작업에 대해 한 번 호출 할 필요가 있습니다. 반면, 활성화 카운터, 다운 스트림 작업의 프롤로그는 정품 인증 카운터를 감소하기 위해 지난 상류 작업에 호출 할 필요가 있습니다.

[0063] 컴파일러는 사용되는 상태 정보의 임의의 타입에 대한 제어 섹션을 추가로 최적화를 적용 할 수 있다. 예를 들어, 특정 토폴로지를 갖는 연결 노드의 특정 서브 세트를 위해, 컴파일러는 제어부의 감소 된 세트를 제공 할 수있다. 이러한 토폴로지의 일 예는 하나의 루트 노드의 상류의 하나 이상의 하류 노드와 노드의 트리이다. 이

나무의 서브 그래프의 일부가 아닌 제어 흐름 그래프의 모든 노드는 보통의 프롤로그와 에필로그가 있습니다. 그러나, 트리 내의 서브 그래프 노드 컴파일러는 루트 노드 및 서브 그래프 트리 단지 리프 노드 에필로그의 감소 된 세트에 대한 단일 결합 프롤로그를 제공한다. 컴파일러는 예를 들면, 루트 노드로부터 하류 제어 흐름 그래프를 통과하고 단지 루트 노드 또는 이전 트리에 추가 다른 노드로부터 입력을 갖는 임의의 노드를 추가하여, 트리 서브 그래프의 구성원을 결정할 수있다. 트리 서브 그래프의 결합 프롤로그는 실행하거나 함께 트리의 모든 노드의 작업 섹션을 억제하고, 모든 잎 노드의 에필로그를 호출하는 코드를 포함한다. 다른 컴파일러 최적화는 상류 노드에서 입력 링크가없는 모든 노드에 대한 프롤로그를 생략 또는 다운 스트림 노드에 대한 출력 링크가없는 모든 노드에 대한 에필로그를 생략 등이 있습니다.

[0064] 일부 구현 예에서, 노드가 나타내는 작업 플로우 그래프를 제어하는 외에 그래프 기반 프로그램 사양의 다양한 종류의 임의에 의해 지정된 부분 순서를 가질 수있다. 예를 들어, 그래프 기반 프로그램 사양의 일부 향하는 에지들은 데이터 흐름을 나타낼 수 그래프 기반 프로그램 사양 제어부 (동시에 그들의 태스크를 실행 링크 노드를 방지 순서 관계를 부과 할 수있다)의 흐름과 다른 방향 에지를 나타낼 수 (이는 동시에 자신의 작업을 실행 링크 노드를 방해하지 않는 종속 관계를 부과 할 수있다).

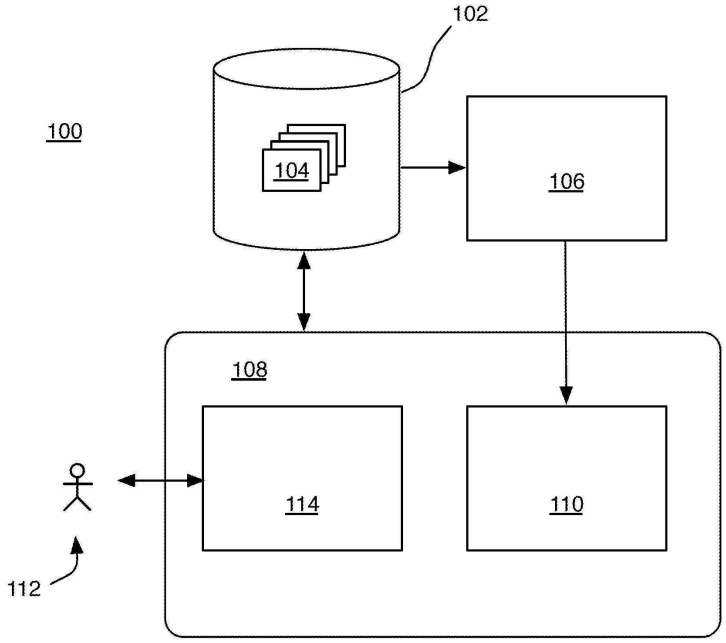
[0065] 상술 한 작업 제어 기술은 적합한 소프트웨어를 실행하는 컴퓨터 시스템을 이용하여 구현 될 수있다. 예를 들어, 소프트웨어는 하나의 절차 또는 상기 적어도 하나의 프로세서를 각각 포함하는 (예 : 분산 된 클라이언트 / 서버, 또는 그리드와 같은 다양한 아키텍처를 구성 될 수 있음), 하나 이상의 프로그램 된 또는 프로그램 가능한 컴퓨터 시스템상에서 실행 이상의 컴퓨터 프로그램을 포함 할 수있다 적어도 하나의 데이터 저장 시스템에서 적어도 하나의 입력 장치 또는 포트를 이용하여 입력을 수신하고, 적어도 하나의 출력을 사용하여 출력을 제공하기위한 (적어도 하나의 사용자 인터페이스 (휘발성 및 / 또는 비 휘발성 메모리 및 / 또는 저장 소자 포함) 장치 또는 포트). 소프트웨어는 데이터 플로우 그래프의 디자인, 구성 및 실행에 관련된 서비스를 제공하는 예를 들어 더 큰 프로그램의 하나 이상의 모듈을 포함 할 수있다. 프로그램 모듈들 (예를 들어, 데이터 플로우 그래프의 요소)은 데이터 저장소에 저장된 데이터 모델에 부합하는 데이터 구조 또는 기타 데이터 조직화로서 구현 될 수 있다.

[0066] 상기 소프트웨어는 CD-ROM 또는 다른 컴퓨터 판독가능한 매체 (예컨대, 범용 또는 특수 목적의 컴퓨팅 시스템 또는 장치에 의해 판독가능한)와 같은, 실체적인(tangible), 비일시적 매체에 제공되거나, 소프트웨어가 실행되는 컴퓨팅 시스템의 실체적인, 비일시적인 매체로 네트워크의 통신 매체를 거쳐 전달될 수 있다 (예컨대, 전파된 신호에 인코딩된다). 처리의 일부 또는 전부는 코프레서들(coprocessors) 또는 필드-프로그램가능한 게이트 어레이들(FPGAs) 또는 전용의, 응용 주문형 집적 회로(application-specific integrated circuits, ASICs)와 같은, 특수 목적의 하드웨어를 이용하여 수행되나, 특수 목적의 컴퓨터에서 수행될 수 있다. 상기 처리는 소프트웨어에 의해 지정된 컴퓨터의 상이한 부분들이 상이한 컴퓨팅 엘리먼트들(computing elements)에 의해 수행되는 배포된 방식으로 구현될 수 있다. 각각의 이러한 컴퓨터 프로그램은 바람직하게 저장 디바이시스 매체가 본 발명에서 설명된 처리를 수행하기 위해 컴퓨터에 의해 판독될때 상기 컴퓨터를 구성하고 동작하기 위하여, 범용 또는 특수 목적의 프로그램가능한 컴퓨터에 의해 액세스 가능한 저장 디바이스의 컴퓨터-판독가능한 저장 매체 (예컨대, 고체 상태 메모리(solid state memory) 또는 미디어, 또는 자기적 또는 광학적 미디어)에 저장되거나 다운로드될 수 있다. 본 발명의 시스템은 그렇게 구성되는 매체는 컴퓨터가 본 발명에서 설명된 적어도 하나의 처리 단계들을 수행하기 위하여 특정한 그리고 미리 정해진 방식으로 동작하도록하는, 컴퓨터 프로그램으로 구성된, 구체적이고(tangible), 비일시적인 매체로 구현되도록 역시 고려될 수 있다.

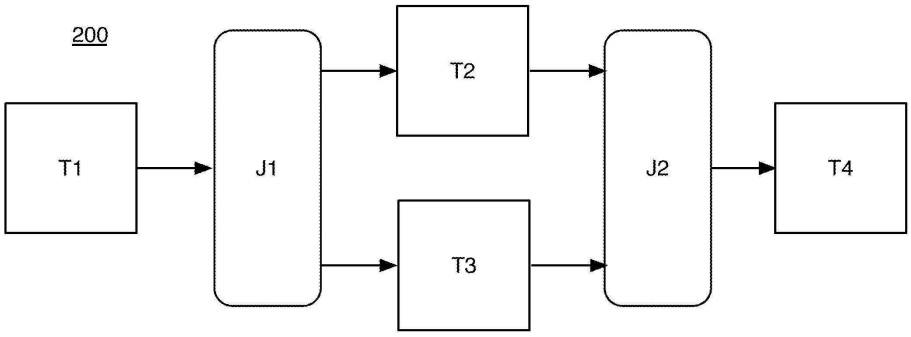
[0067] 본 발명의 다수의 실시 예가 기술되었다. 그럼에도 불구하고, 전술한 설명은 예시를 위한 것이며 다음의 청구범위에의해 정의되는 본 발명의 범위를 한정하는 것이 아닌 것으로 이해되어야 한다. 따라서, 다른 실시예들 또한 다음 청구범위 내에 있다. 예를 들어, 다양한 변형이 본 발명의 범위를 벗어남 없이 만들어 질 수 있다. 부가적으로, 상기 기술된 스텝들의 일부는 순서 독립적이므로 기술된 것과 다른 순서로 수행될 수 있다.

도면

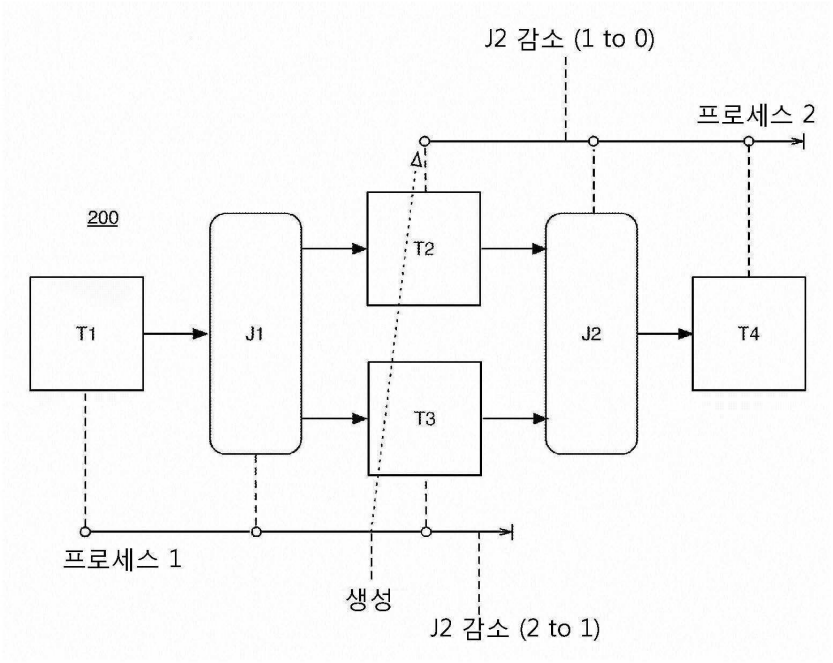
도면1



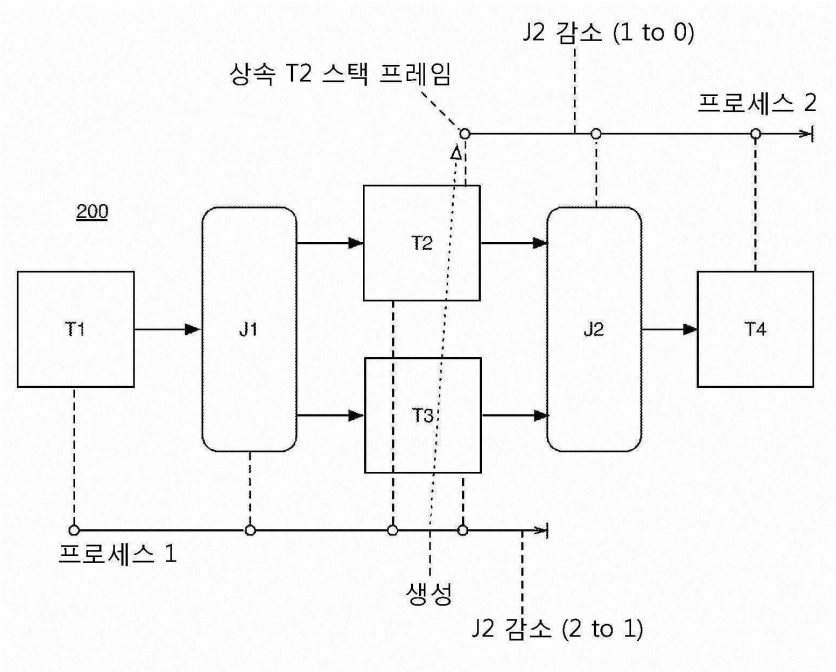
도면2a



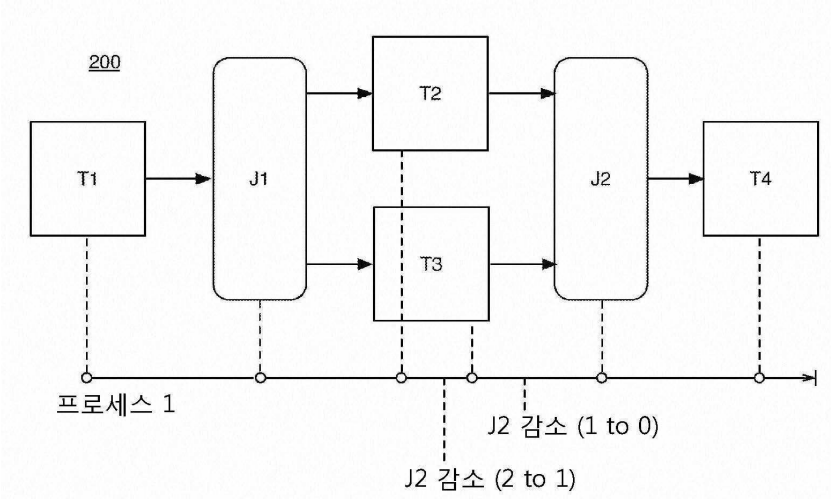
도면2b


J2 감소 (1 to 0)
프로세스 2
200
T1
J1
T2
T3
J2
T4
프로세스 1
생성
J2 감소 (2 to 1)

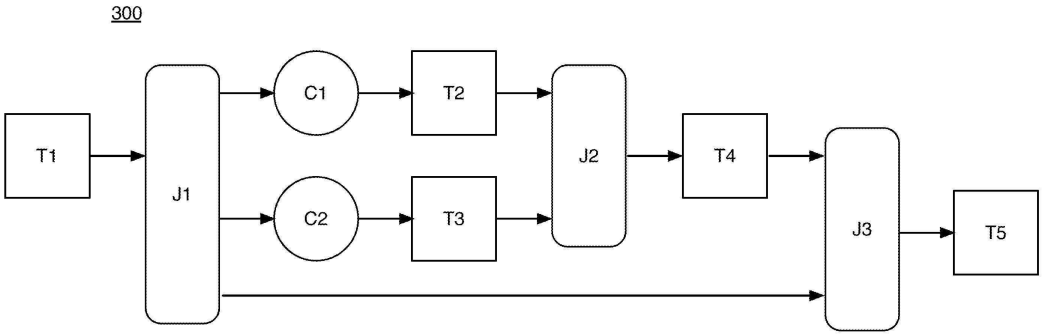
도면2c


J2 감소 (1 to 0)
상속 T2 스택 프레임
프로세스 2
200
T1
J1
T2
T3
J2
T4
프로세스 1
생성
J2 감소 (2 to 1)

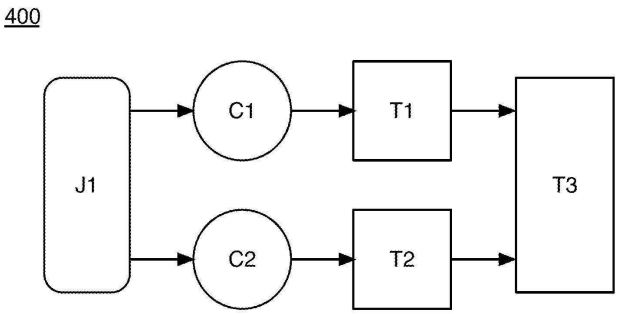
도면2d


200
T1
J1
T2
T3
J2
T4
프로세스 1
J2 감소 (1 to 0)
J2 감소 (2 to 1)

도면3


300
T1
J1
C1
C2
T2
T3
J2
T4
J3
T5

도면4


400
J1
C1
C2
T1
T2
T3