(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2018년10월02일
(11) 등록번호 10-1903805
(24) 등록일자 2018년09월21일

(51) 국제특허분류(Int. Cl.)
G06F 9/48 (2018.01) *G06F 9/44* (2018.01)
(21) 출원번호 10-2013-7015985
(22) 출원일자(국제) 2011년12월20일
심사청구일자 2016년11월17일
(85) 번역문제출일자 2013년06월20일
(65) 공개번호 10-2014-0000283
(43) 공개일자 2014년01월02일
(86) 국제출원번호 PCT/US2011/066280
(87) 국제공개번호 WO 2012/088171
국제공개일자 2012년06월28일
(30) 우선권주장
12/972,792 2010년12월20일 미국(US)
(56) 선행기술조사문헌
US06044475 A*
(뒷면에 계속)

(73) 특허권자
마이크로소프트 테크놀로지 라이센싱, 엘엘씨
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
라이브만 스티븐
미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션
스톨 조나톤 마이클
미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션
허즈밴드즈 패리 존스 레지널드
미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션
(74) 대리인
제일특허법인(유)

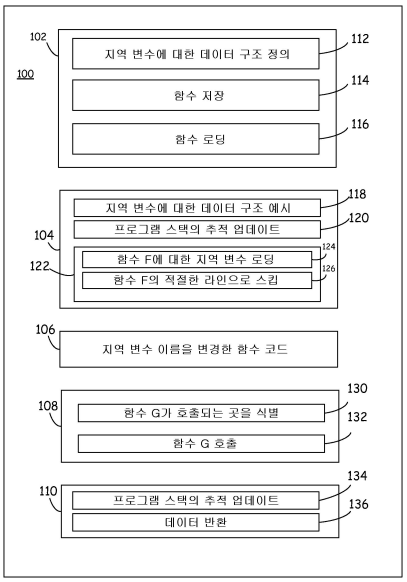
전체 청구항 수 : 총 20 항 심사관 : 유진태

(54) 발명의 명칭 **프로그램 상태를 체크포인팅하며 복원하기 위한 방법**

(57) 요 약

컴퓨터의 운영 체제로부터의 협력을 필요로 하지 않으면서 체크포인트로부터 중단된 프로그램 실행의 복원을 가능하게 하는 기술이 설명된다. 이러한 기술은 중단된 프로그램 실행의 복원을 가능하게 하는 명령어를 추가하는 자동화된 도구를 이용하여 기존의 코드를 수정함으로써 구현될 수 있다.

대 표 도



(56) 선행기술조사문헌

US20060156157 A1*

KR1019970059930 A

M. Bozyigit 외 1명. 'User-level process checkpoint and restore for migration'. ACM SIGOPS Operating Systems Review, Vol.35, Issue 2, 2001.4, pp.86-96.

US7673181 B1

*는 심사관에 의하여 인용된 문헌

명 세 서

청구범위

청구항 1

하나 이상의 동작을 수행하기 위해 마이크로프로세서를 제어하도록 구성된 제 1 컴퓨터 판독 가능 명령어를 수신하는 단계 - 상기 제 1 컴퓨터 판독 가능 명령어는 복수의 함수를 가짐 -, 및

마이크로프로세서를 이용하여 상기 제 1 컴퓨터 판독 가능 명령어에 기초하여 제 2 컴퓨터 판독 가능 명령어를 생성하는 단계를 포함하되,

상기 제 2 컴퓨터 판독 가능 명령어는, 상기 하나 이상의 동작을 수행하며,체크포인트로부터 상기 복수의 함수 내의 함수의 실행을 재개하도록 구성되고,

상기 함수는 적어도 코드의 제 1 라인, 상기 코드의 제 1 라인에 후속하는 코드의 제 2 라인 및 상기 코드의 제 2 라인에 후속하는 코드의 제 3 라인을 포함하며,

상기 함수의 실행의 재개는

함수를 호출하고,

상기 함수의 이전의 실행 중에 상기 체크포인트에서 저장된 데이터를 상기 함수에 의해 사용하기 위해 로딩(loading)하며,

상기 함수의 상기 코드의 제 1 라인에서 상기 코드의 제 2 라인으로 스킵 - 상기 코드의 제 2 라인은 상기 함수의 이전의 실행 동안 상기 체크포인트에서 실행 중이었음 - 함으로써,

상기 코드의 제 1 라인을 실행하지 않고, 상기 코드의 제 2 라인에 대응되는 상기 체크포인트로부터 이루어지는

방법.

청구항 2

제 1 항에 있어서,

상기 방법은 상기 제 1 컴퓨터 판독 가능 명령어를 포함하는 소스 코드를 상기 제 2 컴퓨터 판독 가능 명령어를 포함하는 목적 코드로 컴파일하는 컴파일러에 의해 수행되는

방법.

청구항 3

제 1 항에 있어서,

상기 제 2 컴퓨터 판독 가능 명령어는 상기 함수에 의해 사용되는 데이터를 저장하도록 추가적으로 구성되는

방법.

청구항 4

제 1 항에 있어서,

상기 제 2 컴퓨터 판독 가능 명령어는 상기 함수를 포함하는 프로그램의 프로그램 스택을 추적하도록 추가적으로 구성되는

방법.

청구항 5

제 1 항에 있어서,

상기 제 2 컴퓨터 판독 가능 명령어는, 상기 제 2 컴퓨터 판독 가능 명령어를 실행하는 디바이스 상에서 실행되는 운영체제와 독립적으로(independent), 상기 체크포인트로부터 상기 함수의 실행을 재개하도록 구성되는

방법.

청구항 6

명령어를 저장한 컴퓨터 판독 가능 저장 매체로서,

상기 명령어는 실행될 때,

하나 이상의 동작을 수행하기 위해 마이크로프로세서를 제어하도록 구성된 제 1 컴퓨터 판독 가능 명령어를 수신하는 단계 - 상기 제 1 컴퓨터 판독 가능 명령어는 복수의 함수를 가짐 -, 및

마이크로프로세서를 이용하여 상기 제 1 컴퓨터 판독 가능 명령어에 기초하여 제 2 컴퓨터 판독 가능 명령어를 생성하는 단계

를 포함하는 방법을 실행하되,

상기 제 2 컴퓨터 판독 가능 명령어는, 상기 하나 이상의 동작을 수행하고, 적어도 코드의 제 1 라인, 상기 코드의 제 1 라인에 후속하는 코드의 제 2 라인 및 상기 코드의 제 2 라인에 후속하는 코드의 제 3 라인을 포함하는 함수를 포함하는 프로그램의 프로그램 스택을 추적하며, 체크포인트로부터 상기 복수의 함수 내에서 상기 함수의 실행을 재개하도록 구성되고,

상기 함수의 실행의 재개는

함수를 호출하고,

상기 함수의 이전의 실행 중에 상기 체크포인트에서 저장된 데이터를 상기 함수에 의해 사용하기 위해 로딩(loading)하며,

상기 함수의 상기 코드의 제 1 라인에서 상기 코드의 제 2 라인으로 스킵 - 상기 코드의 제 2 라인은 상기 함수의 상기 이전의 실행 동안 상기 체크포인트에서 실행 중이었음 - 함으로써,

상기 코드의 제 1 라인을 실행하지 않고, 상기 코드의 제 2 라인에 대응되는 상기 체크포인트로부터 이루어지는

컴퓨터 판독 가능 저장 매체.

청구항 7

제 6 항에 있어서,

상기 방법은 상기 제 1 컴퓨터 판독 가능 명령어를 포함하는 소스 코드를 상기 제 2 컴퓨터 판독 가능 명령어를 포함하는 목적 코드로 컴파일하는 컴파일러에 의해 수행되는

컴퓨터 판독 가능 저장 매체.

청구항 8

제 6 항에 있어서,

상기 제 2 컴퓨터 판독 가능 명령어는 상기 함수에 의해 사용되는 데이터를 저장하도록 추가적으로 구성되는

컴퓨터 판독 가능 저장 매체.

청구항 9

제 6 항에 있어서,

상기 제 2 컴퓨터 판독 가능 명령어는 상기 함수에 의해 사용되는 데이터를 저장하기 위한 데이터 구조를 정의하도록 추가적으로 구성되는

컴퓨터 판독 가능 저장 매체.

청구항 10

제 6 항에 있어서,

상기 제 2 컴퓨터 판독 가능 명령어는, 상기 제 2 컴퓨터 판독 가능 명령어를 실행하는 디바이스 상에서 실행되는 운영체제와 독립적으로(independent), 상기 체크포인트로부터 상기 함수의 실행을 재개하도록 구성되는

컴퓨터 판독 가능 저장 매체.

청구항 11

적어도 코드의 제 1 라인, 상기 코드의 제 1 라인에 후속하는 코드의 제 2 라인 및 상기 코드의 제 2 라인에 후속하는 코드의 제 3 라인을 포함하는 함수의 실행을 재개하도록 구성된 컴퓨터 판독 가능 명령어를 저장한 컴퓨터 판독 가능 저장 매체 및

상기 컴퓨터 판독 가능 명령어를 실행하도록 구성된 적어도 하나의 마이크로프로세서를 포함하되,

상기 함수의 실행의 재개는

함수를 호출하고,

상기 함수의 이전의 실행 중에 체크포인트에서 저장된 데이터를 상기 함수에 의해 사용하기 위해 로딩하며,

상기 함수의 상기 코드의 제 1 라인에서 상기 코드의 제 2 라인으로 스킵 - 상기 코드의 제 2 라인은 상기 함수의 이전의 실행 동안 상기 체크포인트에서 실행 중이었음 - 하고,

상기 함수에 의해 사용된 데이터를 저장하며,

상기 함수를 포함하는 프로그램의 프로그램 스택을 추적함으로써,

상기 코드의 제 1 라인을 실행하지 않고, 상기 코드의 제 2 라인에 대응되는 상기 체크포인트로부터 이루어지는

시스템.

청구항 12

제 11 항에 있어서,

상기 적어도 하나의 마이크로프로세서는 제 1 마이크로프로세서 및 제 2 마이크로프로세서를 포함하는

시스템.

청구항 13

제 12 항에 있어서,

상기 제 1 마이크로프로세서는 상기 함수를 실행하며, 상기 제 1 마이크로프로세서를 이용한 상기 함수의 실행이 중단되면, 상기 함수의 실행은 상기 제 2 마이크로프로세서를 이용하여 상기 체크포인트로부터 계속되는

시스템.

청구항 14

제 13 항에 있어서,

상기 시스템은

상기 제 1 마이크로프로세서를 포함하는 제 1 장치, 및

상기 제 2 마이크로프로세서를 포함하는 제 2 장치를 포함하는

시스템.

청구항 15

제 14 항에 있어서,

상기 제 1 장치는 제 1 운영 체제를 실행하며, 상기 제 2 장치는 상기 제 1 운영 체제와 다른 제 2 운영 체제를 실행하는

시스템.

청구항 16

제 11 항에 있어서,

상기 체크포인트에서 상기 데이터를 저장할지 여부를 결정하기 위한 컴퓨터 판독 가능 명령어를 더 포함하는

시스템.

청구항 17

제 11 항에 있어서,

상기 컴퓨터 판독 가능 명령어의 서브셋은 상기 함수에 의해 사용되는 상기 데이터를 저장하기 위한 데이터 구조를 정의하는

시스템.

청구항 18

제 11 항에 있어서,

상기 컴퓨터 판독 가능 명령어의 서브셋은 하나 또는 그 이상의 함수가 하나 또는 그 이상의 다른 함수를 호출하는 상기 함수 내의 위치를 가리키는

시스템.

청구항 19

제 11 항에 있어서,

상기 컴퓨터 판독 가능 명령어는, 상기 컴퓨터 판독 가능 명령어를 실행하는 디바이스 상에서 실행되는 운영체제와 독립적으로(independent), 상기 체크포인트로부터 상기 함수의 실행을 재개하도록 구성되는

시스템.

청구항 20

제 11 항에 있어서,

상기 컴퓨터 판독 가능 명령어는 컴파일러에 의해 생성되는 목적 코드인

시스템.

발명의 설명

기 술 분 야

[0001] 본 명세서에 설명된 기술은 중단 점(point of interruption)에서 컴퓨터 프로세스의 실행의 복원을 가능하게 하는 것에 관한 것이다.

배 경 기 술

[0002] 응용 프로그램 또는 다른 컴퓨터 프로그램이 갑자기 종료되거나 설계된 대로 동작하는 것을 중단하는 프로그램 충돌이 발생하는 것으로 알려져 있다. 충돌이 발생할 때 데이터 손실량이 제한될 수 있도록 일부 프로그램은 수시로 정보를 저장한다. 예를 들면, 일부 워드 프로세싱 프로그램은 사용자가 문서 작업 중일 때 문서의 초안을 여러 번 자동으로 저장한다. 워드 프로세싱 프로그램이 충돌하는 경우, 사용자는 문서가 워드 프로세싱 프로그램에 의해 자동으로 저장된 가장 최근의 지점에서 저장된 데이터를 로딩(loading)하여 문서를 복원할 수 있다.

[0003] 일부 운영 체제는 프로그램이 실행될 때 프로그램에서 정보를 수신하는 능력을 제공하며, 저장된 상태로부터 프로그램의 실행을 복원할 수 있다. 그러나, 그러한 기법에서, 프로그램의 실행이 복원되도록 하기 위해 운영 체제로부터의 협력이 필요하다.

발명의 내용

해결하려는 과제

[0004] 상술한 바와 같이, 중단 점에서 프로그램의 실행을 복원하기 위한 일부 기존 기술은 운영 체제로부터의 협력을 필요로 한다. 이러한 기술은 운영 체제가 응용 프로그램 데이터를 저장하며 실행이 중단될 때 응용 프로그램의 동작 상태를 복원하는데 적절한 인터페이스를 제공하도록 요구한다. 그러나, 모든 운영 체제가 이러한 기능성을 제공하지는 못한다. 운영 체제로부터의 협력을 필요로 하지 않으면서 응용 프로그램 수준에서 프로그램의 실행을 복원하는 기능을 제공하는 것이 유리할 것이다.

[0005] 일부 응용 프로그램은 특히 프로그램 데이터를 저장하며 실행이 중단될 때 프로그램 데이터를 복원하는 기능을 포함하도록 설계되고 프로그램되었다. 그러나, 응용 프로그래머가 새로운 응용 프로그램을 작성하거나 기존의 응용 프로그램을 수정할 때 이러한 기능을 제공하는 응용 프로그램을 설계하고 프로그램하는 것은 시간 소모적이다.

[0006] 본 명세서에 설명되는 기술은 프로그램 실행의 체크포인팅 및 복원을 가능하게 하는 프로그램 레벨에서 유연한 프레임워크를 제공한다. 출원인은 컴퓨터의 운영 체제로부터의 협력을 필요로 하지 않고 실행이 중단된 시점에서 프로그램 실행을 복원하는 기술을 개발하였다. 이러한 기술은 프로그램이 실행되는 운영 체제 환경과 관계없이 프로그램 레벨에서 구현될 수 있다. 일부 실시예에서, 이러한 기술을 구현하기 위한 코드는 프로그램을 수정하고 중단된 프로그램 실행의 복원을 가능하게 하기 위한 명령어를 추가하는 자동화된 도구를 사용하여 기존 프로그램 코드에 통합될 수 있다. 따라서, 응용 프로그램을 개발하는 프로그래머는 중단된 프로그램 실행의

복원을 가능하게 하기 위한 코드가 자동화된 도구를 사용하여 자동으로 통합될 수 있는 그런 특징을 포함하도록 응용 프로그램 코드를 설계하거나 프로그램할 필요가 없다.

과제의 해결 수단

[0007] 일부 실시예는 방법에 관한 것으로서, 방법은 하나 이상의 동작을 수행하기 위해 마이크로프로세서를 제어하도록 구성된 제 1 컴퓨터 판독 가능 명령어를 수신하는 단계, 및 마이크로프로세서를 사용하여 제 1 컴퓨터 판독 가능 명령어에 기초하여 제 2 컴퓨터 판독 가능 명령어를 생성하되, 제 2 컴퓨터 판독 가능 명령어는 하나 이상의 동작을 수행하며, 함수를 호출하고, 함수의 이전의 실행 중에 체크포인트에서 저장된 데이터를 함수에 의해 사용하기 위해 로딩하며, 체크포인트에 도달하기 전에 함수의 이전의 실행 중에 실행된 함수의 부분을 스킵함으로써 체크포인트로부터의 함수의 실행을 재개하도록 구성되는 단계를 포함한다. 일부 실시예는 실행될 때 상술한 방법을 수행하는 명령어를 저장한 컴퓨터 판독 가능 저장 매체에 관한 것이다.

[0008] 일부 실시예는 시스템에 관한 것으로서, 시스템은 체크포인트로부터의 함수를 호출하고, 함수의 이전의 실행 중에 체크포인트에서 저장된 데이터를 함수에 의해 사용하기 위해 로딩하고, 체크포인트에 도달하기 전에 상기 함수의 이전의 실행 중에 실행된 함수의 부분을 스킵하고, 함수에 의해 사용된 데이터를 저장하며, 함수를 포함하는 프로그램의 프로그램 스택을 추적함으로써 체크포인트로부터의 함수의 실행을 재개하도록 구성된 컴퓨터 판독 가능 명령어를 저장한 컴퓨터 판독 가능 저장 매체를 포함한다. 시스템은 또한 컴퓨터 판독 가능 명령어를 실행하도록 구성된 적어도 하나의 마이크로프로세서를 포함한다.

[0009] 상술한 것은 일부 실시예의 비제한적인 요약이다.

도면의 간단한 설명

[0010] 도면에서, 여러 도면에서 예시된 각각의 동일하거나 거의 동일한 각 구성 요소는 동일한 참조 부호로 나타낸다. 명료성을 위해, 모든 구성 요소가 모든 도면에서 참조 부호가 부여되는 것은 아닐 수 있다. 도면은 반드시 축척에 따라 도시되는 것은 아니며, 대신 본 발명의 여러 양태를 예시하는 것이 강조된다.

도 1은 일부 실시예에 따라 컴퓨터 프로그램의 함수 F의 부분으로서 동작을 수행하기 위한 컴퓨터 판독 가능 명령어를 포함하는 소프트웨어 모듈의 다이어그램을 도시한다.

도 2는 일부 실시예에 따라 여러 체크포인트에서 프로그램 상태 및 프로그램 데이터를 저장하는 단계를 포함하는 프로그램을 실행하는 방법의 흐름도를 도시한다.

도 3은 일부 실시예에 따라 체크포인트로부터의 함수의 실행을 재구축하는 방법의 흐름도를 도시한다.

도 4는 일부 실시예에 따라 체크포인트로부터 프로그램의 복원 동작을 가능하게 하도록 코드를 수정하는 방법의 흐름도를 도시한다.

도 5는 본 명세서에 설명된 기술이 구현될 수 있는 복수의 마이크로프로세서를 가진 컴퓨팅 장치의 일례를 도시한다.

도 6은 본 명세서에 설명된 기술이 구현될 수 있는 컴퓨팅 장치를 포함하는 컴퓨팅 환경의 일례를 도시한다.

발명을 실시하기 위한 구체적인 내용

[0011] 상술한 바와 같이, 본 명세서에 설명된 기술은 컴퓨터의 운영 체제로부터의 협력을 필요로 하지 않으면서 실행이 중단될 때 체크포인트로부터 프로그램 실행을 복원할 수 있게 한다. 이러한 기술은 기반 하드웨어 또는 소프트웨어의 불안정성에 대해 보호할 수 있으며, 시스템에서 운영 체제 충돌, 정전 또는 다른 고장 후에 프로그램 실행을 복원할 수 있다. 이러한 기술은 특히 이러한 함수를 포함하기 위해 응용 프로그램을 프로그래밍할 프로그래머의 필요없이 자동화된 도구를 사용하여 기존의 코드를 수정하여 구현될 수 있다. 일부 실시예에서, 컴파일러 또는 다른 프로그램 변환 도구는 오류가 발생한 후에 프로그램의 실행을 저장하고 복원하는 능력을 제공하기 위해 기존의 코드를 재구성할 수 있다.

[0012] 유리하게, 이러한 기술은 멀티스레드 멀티코어 환경(multithreaded multicore environment)에서 및 프로그램이 다수의 머신에서 실행되는 환경에 대해 강할 수 있다. 실행이 중단되고 계속할 수 없을 때, 프로그램은 다른

프로세서 또는 다른 머신, 심지어 다른 운영 체제를 실행하는 머신에서도 실행을 계속할 수 있다. 이러한 기술은 다수의 머신이 프로그램을 실행하는 데 이용할 수 있는 클라우드 컴퓨팅 환경에서 유리하게 사용될 수 있다. 예시적인 실시예에 대한 상세한 논의는 이하에 제공된다.

[0013] 1. 복원 실행을 지원하기 위해 프로그램 코드를 수정하는 예

[0014] 일반적으로 사용되는 프로그램 언어는 프로그램 실행 중에 호출될 수 있는 함수를 정의하는 능력을 제공한다. 코드는 함수가 수용하는 입력, 함수에 의해 수행되는 동작, 및 함수에 의해 반환되는 데이터와 같은 함수의 중요한 측면을 정의한다.

[0015] 예를 들면, 다음의 코드는 F 및 G를 정의한다. 함수 F는 입력으로 정수 a 및 b를 수신한다. 함수 F는 곱 a·b과 동일하게 설정되는 정수이도록 지역(local) 변수 x를 초기화하며, 함수 G의 반환 값과 동일하게 설정되는 정수이도록 변수 y를 초기화한다. 함수 F는 값 x+y를 가진 정수이다. 함수 G는 제각기 함수 G 내의 지역 변수 a 및 b에 대응하는 입력으로 정수 x 및 b를 수신한다. 함수 G는 a+b와 동일하게 설정되는 정수이도록 지역 변수 x를 초기화한다. 함수 G는 x의 값과 동일한 정수를 반환한다.

[0016]
```
int F(int a, int b)
{
    int x = a*b;
    int y = G(x,b);
    return x+y;
}

int G(int a, int b)
{
    int x = a+b;
    return x;
}
```

[0017] 프로그램이 실행되면, 그 후 함수 G를 호출하는 함수 F가 호출된다. 함수 F 및/또는 G의 실행 중에 프로그램 충돌하거나 또는 실행이 그와 달리 중단되면, 함수 F 및/또는 G의 실행 시에 행해지는 진행은 없어질 수 있다. 실행을 계속하기 위해, 프로그램은 다시 시작될 필요가 있고, 함수 F는 처음부터 다시 시작될 필요가 있으며, 그 다음에 함수 G가 다시 호출될 필요가 있다. 예시적인 함수 F 및 G를 실행하는데 필요한 처리 능력의 양이 크지 않을 지라도, 복잡한 프로그램을 실행할 때 중요한 데이터가 손실될 수 있으며 함수 F 및/또는 G의 실행이 중단된 지점에 도달하기 위해 중요한 처리가 다시 수행될 필요가 있을 수 있다는 것이 이해되어야 한다. 손실된 작업의 양은 예들 들어 모델링 및 시뮬레이션과 같은 복잡한 프로그래밍 태스크를 실행할 때 특히 중요할 수 있다.

[0018] 일부 실시예에서, 추가적인 코드가 "체크포인트(checkpoint)"라 하는 여러 포인트에서 프로그램의 동작의 상태를 저장할 수 있도록 프로그램에 삽입될 수 있다. 코드는 어느 함수가 실행되는지에 관한 표시를 저장함으로써 프로그램 스택을 추적하기 위해 포함될 수 있다. 코드는 또한 함수에 의해 사용되는 지역 변수를 저장하며, 실행 시에 도달된 함수 내의 라인의 표시를 저장하기 위해 포함될 수 있다. 이러한 타입의 코드의 추가는 체크포인트에서 동작 중에 있는 하나 이상의 함수를 호출하고, 함수에 의해 사용 중에 있는 지역 변수를 로딩하며, 중단 전에 이미 실행된 함수의 부분을 스킵하여 프로그램을 재구축할 수 있다.

[0019] 도 1은 컴퓨터 프로그램의 함수 F의 동작을 수행하며 프로그램 실행의 복원을 가능하게 하기 위한 컴퓨터 판독 가능 명령어를 포함하는 소프트웨어 모듈(100)의 일례를 도시한다. 소프트웨어 모듈(100)의 컴퓨터 판독 가능 명령어는 프로그램 내에서 함수의 실행의 상태를 추적하고, 프로그램 실행 중에 여러 체크포인트로부터의 함수에 의해 사용되는 데이터를 저장하며, 프로그램이 중단될 때 체크포인트로부터의 함수의 실행을 복원하는 것을 가능하게 한다. 도 1의 예에서, 소프트웨어 모듈(100)은 헬퍼(helper) 코드(102), 함수 프리앰블(104), 함수 F의 동작을 수행하기 위한 함수 코드(106), 호출 사이트 라벨 코드(108), 및 함수 에필로그(epilogue)(110)를 포함한다.

[0020] 헬퍼 코드(102)는 함수 F의 지역 변수 및 입력 인수(input argument)를 저장하기 위한 데이터 구조를 정의하는 코드(112)를 포함할 수 있다. 헬퍼 코드(102)는 또한 함수 F에 대한 데이터를 저장하기 위한 저장 함수를 정의하는 코드(114), 및 저장 장치로부터 함수 F에 대한 데이터를 로딩하기 위한 로딩 함수를 정의하는 코드(116)를 포함할 수 있다. 저장 함수는 저장된 변수를 체크포인트 파일에 첨부한다. 로딩 함수는 현재 지점에서의 변수를 체크포인트 파일에 로딩하고 파일 포인터를 전진시킨다. 개념을 예시하기 위한 소스 코드로 나타내지만, 헬퍼 코드(102) 및 본 명세서에 설명된 어떤 다른 코드는 목적 코드 또는 어떤 다른 적절한 타입의 코드로 구현될

수 있다는 것이 이해되어야 한다. 데이터 구조를 정의하며 저장 및 로딩 함수를 제공하는 데 적절한 헬퍼 코드(102)의 예가 아래에 도시된다.

[0021]
```
struct F_Locals : Locals
{
    // local variables
    int x;
    int y;

    // input arguments
    int a;
    int b;

    virtual void Save()
    {
        g_pCheckpoint->SaveLocals(this);
    };
    virtual void Load()
    {
        g_pCheckpoint->LoadLocals(this);
    }
};
```

[0022]
```
// Base frame structure from which function-specific versions derive
struct Locals
{
    virtual void Save() = 0;
    virtual void Load() = 0;

    int __CallSite;

    // Next structure. This is used to preserve ordering of stack frames in
the example implementation,
    // but any of several alternative solutions can be used for this, as app
ropriate.
    Locals * m_pNext;
};
```

[0023] 도 1에 도시된 바와 같이, 함수 프리앰블(104)은 함수 F의 지역 변수에 대한 데이터 구조 객체를 예시하는 코드(118), 및 함수 F가 push() 함수를 사용하여 실행되고 있다는 표시를 저장하여 프로그램 스택을 추적하기 위한 코드(120)를 포함할 수 있다. 함수 프리앰블(104)은 또한 체크포인트로부터의 함수 F의 실행을 재구축할 수 있는 코드(122)를 포함할 수 있다. 함수 F의 실행이 재구축되면, 코드(124)는 체크포인트에서 저장된 함수 F에 대한 지역 변수를 로딩하기 위해 실행된다. 코드(126)는 체크포인트에 도달하기 전에 이미 실행된 함수 F의 부분을 스킵하도록 실행된다. 예를 들면, 프로그램의 실행이 중단되었을 때 함수 G가 함수 F의 범위 내에서 실행 중이면, 코드(126)는 함수 G를 호출하는 함수의 라인으로 스킵할 수 있다. 일부 실시예에서 이용될 수 있는 함수 프리앰블(104)의 예가 아래에 도시된다.

[0024]
```
int F(int a, int b)
{
          F_Locals l;
    Push(&l);

    // Checkpointing
    if (g_RebuildMode)
    {
        // Deserialize locals from checkpoint
        l.Load();

        // Jump to proper line
        int i = l.__CallSite;
        switch(i) {
        case 1: goto FunctionCall1; //where G
        }                           // is called
    }
```

[0025] 도 1에 도시된 바와 같이, 함수 코드(106)는 함수, 예를 들어 함수 F의 동작을 수행하기 위해 포함된다. 함수 F가 예로서 논의되지만, 본 명세서에 설명된 기술은 어떤 함수가 컴퓨터 프로그램의 하나 이상의 동작을 수행하는 데 사용될 수 있다. 함수 코드(106)는 원래의 함수, 예를 들어 함수 F의 코드로부터 재기록될 수 있음으로

써, 지역 변수는 변수가 나중에 함수 F의 실행을 재구축하는데 필요할 경우에 로딩될 수 있도록 실행 중에 데이터의 저장을 용이하게 하기 위해 코드(112)에 의해 정의된 데이터 구조에 저장된다. 일부 실시예에서 사용하기 위한 함수 코드(106)의 예는 아래에 도시된다.

[0026]
```
    l.x = a*b;
```

[0027] 함수 F에 대해 상술한 바와 같이, 변수 x는 곱 a · b과 동일하게 설정된다. 함수 코드(106)에서, 동일한 곱셈 연산은 함수 F에서와 같이 수행되지만, 코드는 함수가 재구축될 필요가 있을 경우에 지역 변수를 다시 로딩하는 것을 용이하게 하기 위해 코드(112)에 의해 정의된 데이터 구조에 결과가 저장되도록 수정된다.

[0028] 도 1에 도시된 바와 같이, 호출 사이트 라벨 코드(130)는 다른 함수가 함수 F 내에서 호출되는 식별 코드를 포함할 수 있다. 예를 들면, 함수 F는 다른 함수 G를 호출할 수 있으며, 호출 사이트 라벨 코드(130)는 함수 G가 호출되는 함수 F 내의 위치를 식별할 수 있다. 또한, 코드(132)는 함수 G를 호출하기 위한 함수 F에 포함된다. 일부 실시예에서 사용하기 위한 호출 사이트 라벨 코드(130)의 예가 아래에 도시된다.

[0029]
```
        FunctionCall1:
    l.__CallSite = 1;
    l.y = G(l.x,b);
```

[0030] 이 시점에서, 다음과 같은 추가적인 함수 코드(106)가 포함될 수 있으며, 그것은 위의 함수 F에서와 같이 변수 x 및 y의 합을 계산하기 위해 실행된다.

[0031]
```
    int temp = l.x+l.y;
```

[0032] 도 1에 도시된 바와 같이, 함수 에필로그(110)는 함수 F의 실행이 완료될 때 pop() 함수를 사용하여 프로그램 스택의 추적을 업데이트하기 위한 코드(134)를 포함할 수 있다. 함수 에필로그(110)는 또한 함수 F에 의해 데이터를 반환하기 위한 코드(136)를 포함할 수 있다. 일부 실시예에서 사용될 수 있는 함수 에필로그(110)의 예는 아래에 도시된다.

[0033]
```
            Pop();
    return temp;
}
```

[0034] Push() 및 Pop()의 샘플 구현은 예로서 아래에 도시된다.

[0035]
```
Locals * g_pStack = NULL;
void Push(Locals * pNewFrame)
{
    pNewFrame->m_pNext = g_pStack;
    g_pStack = pNewFrame;
}
void Pop()
{
    Locals * pTop = g_pStack;
    g_pStack = pTop->m_pNext;
    pTop->m_pNext = NULL;
}
```

[0036] 다음의 코드는 그 후 "Locals" 객체의 Load() 및 Save() 메소드에 의해 사용되는 LoadLocals() 및 SaveLocals()의 구현을 가진 "Checkpoint" 클래스의 예를 정의한다.

```
// False if running normally. True if rebuilding the stack from a che
ckpoint.
bool g_RebuildMode = false;

class Checkpoint
{
public:
    Checkpoint()
    {
        idx = 0;
    }
    //
    // Helpers for reading a checkpoint
    //

    // Copy locals out of checkpoint into the data structure
    template<class T>
    void LoadLocals(T * p)
    {
        Locals * pNext = (Locals*) p;

        int cb = sizeof(T);
        memcpy(p, &pBuffer[idx], cb); // Alternatively, read from per
sistent storage at this step.

        ((Locals*)p)->m_pNext = pNext;

        idx+= cb;
    };


    //
    // Helpers for creating a checkpoint
    //
    template<class T>
    void SaveLocals(T * p)
    {
        int cb = sizeof(T);
         * memcpy(&pBuffer[idx], p, cb); // Alternatively, write directl
y to persistent storage at this step.
        idx+= cb;
    };

    void Done()
    {
        idx = 0;
    }


private:
    int idx;
    BYTE pBuffer[1000]; // For example purposes, a buffer in memory i
s used. Can be replaced with allocation of persistent storage.
};
Checkpoint * g_pCheckpoint = NULL;
```

[0037]

[0038] II. 체크포인트에서의 프로그램 상태 저장의 예

[0039] 도 2는 다양한 체크포인트에서 프로그램을 실행하고 프로그램 데이터를 저장하는 방법(200)의 예를 도시한다. 프로그램 상태를 저장하기 위해, 지역 객체의 스택이 검토될 수 있으며, 각각의 객체는 차례로 저장될 수 있다. 프로그램의 실행은 단계(202)에서 시작한다. 단계(203)에서, 프로그램이 프로그램의 상태를 저장하는 옵션을 갖는 체크포인트 기회가 온다. 이러한 프로그램은 프로그램 상태를 저장할 체크포인트 기회를 이용할 지 여부에 관해 단계(203)에서 판단할 수 있다. 프로그램이 체크포인트 기회를 이용할 지에 관한 결정은 마지막 체크포인트 또는 다른 적절한 기준으로부터 경과한 시간량에 기초하여 행해질 수 있다. 어떤 경우에, 사용자는 체크포인트를 명시적으로 요청할 수 있다. 프로그램 상태를 저장하는 빈도(frequency)와 프로그램의 성능 사이에는 트레이드 오프(tradeoff)가 있다. 프로그램 상태가 저장되는 빈도를 증가시키는 것은 데이터 손실량을 제한할 수 있지만, 프로그램의 성능을 저하시킬 수 있다. 체크포인트의 기회가 취해지는 빈도는 응용 프로그램에 따라 다를 수 있다.

[0040] 프로그램이 프로그램 상태를 저장하기 위해 체크포인트 기회를 사용하기로 결정하면, 이러한 정보는 단계(204)에서 저장된다. 예를 들면, 프로그램은 헬퍼 코드(102)에서 코드(114)에 의해 정의된 저장 함수를 사용할 수 있다. 프로그램이 체크포인트 기회를 사용하지 않기로 결정하면, 프로그램의 실행은 단계(206)에서 계속한다. 프로그램의 실행은 다음 체크포인트가 도달될 때까지 계속하며, 어느 시점에서 방법은 단계(203)에서 다음 체크포인트 기회로 복귀한다. 방법(200)은 프로그램이 종료될 때까지 계속할 수 있다.

[0041] III. 체크포인트로부터 프로그램 실행을 복원하는 예

[0042] 도 3은 일부 실시예에 따라 체크포인트로부터의 함수의 실행을 재구축하는 방법(300)을 도시한다. 단계(302)에서, 프로그램은 충돌 또는 다른 고장이 발생한 후에 다시 시작된다. 단계(304)에서, 프로그램은 상술한 함수 F와 같은 제 1 함수를 호출한다. 체크포인트에서 저장된 함수 F에 대한 지역 변수는 단계(306)에서 저장 장치로부터 로딩된다. 단계(308)에서, 프로그램은 체크포인트에서 실행 중에 있는 함수 F의 라인으로 스킵하며, 함수 F의 실행은 그 시점으로부터 재개된다. 따라서, 단계(306 및 308)는 프로그램 상태가 체크포인트에서 저장되었을 때 함수 F의 실행을 실행 시점으로 복원하는 것을 용이하게 한다. 함수 G가 체크포인트에서 실행 중에 있는 경우, 프로그램은 함수 G를 호출하는 함수 F의 라인으로 스킵할 수 있으며, 그 다음에 함수 G는 단계(310)에서 호출된다. 그 후, 단계(306 및 308)는 함수 G에 대해 반복된다. 체크포인트에서 저장된 함수 G에 대한 지역 변수는 저장 장치로부터 로딩되며, 프로그램은 체크포인트에서 실행 중에 있는 함수 G의 라인으로 스킵한다. 다른 함수가 이전의 함수의 범위 내에서 실행 중에 있지 않으면, 방법은 종료하며, 프로그램 실행은 체크포인트로부터 계속한다. 따라서, 프로그램의 실행은 체크포인트가 프로그램의 중단에 앞서 도달되기 전에 실행된 모든 동작을 다시 실행할 필요없이 체크포인트로부터 재개할 수 있다.

[0043] IV. 체크포인트로부터 상태의 저장을 가능하게 하고 프로그램 실행을 복원하는 코드 생성

[0044] 도 4는 일부 실시예에 따라 체크포인트로부터 프로그램의 복원 동작을 가능하게 하도록 코드를 수정하는 방법을 도시한다. 상술한 바와 같이, 본 명세서에 설명된 기술은 이점으로 프로그래머에 의한 수동 코딩없이 자동으로 프로그램에 통합될 수 있다는 것이다. 컴파일러와 같은 프로그램 변환 도구는 체크포인트로부터 프로그램의 복원 동작을 가능하게 하도록 기존의 코드를 수정할 수 있다. 예를 들면, 단계(402)에서, 컴파일러 또는 다른 프로그램 변환 도구는 하나 이상의 함수를 정의하는 코드를 포함하는 프로그램 코드를 수신할 수 있다. 프로그램 코드는 본 명세서에 설명된 기술이 프로그램에 의해 수행되는 동작의 타입에 대해 제한되지 않을 때에 어떤 적절한 동작을 수행하도록 설계될 수 있다. 일례로서, 프로그램 코드는 상술한 바와 같이 함수 F 및 G를 정의하는 코드를 포함할 수 있다. 단계(404)에서, 함수 F 및 G를 실행하기 위한 코드는 프로그램이 체크포인트로부터 실행을 복원할 수 있도록 수정된다. 예를 들면, 컴파일러 또는 다른 프로그램 변환 도구는 상술한 바와 같이 헬퍼 코드(102), 함수 프리앰블(104), 호출 사이트 라벨 코드(108) 및 함수 에필로그(110)를 삽입할 수 있다. 체크포인트의 기회는 또한 데이터를 저장하기 위한 코드 내에서 적절한 위치에 삽입될 수 있다. 함수 코드(106)는 헬퍼 코드(102)에 정의된 지역 변수를 저장하기 위한 데이터 구조를 이용하도록 수정될 수 있다. 컴파일러가 방법(400)을 수행하는 데 사용되면, 결과적으로 중단이 발생할 때 체크포인트로부터 함수 F 및 G의 실행의 복원을 가능하게 하는 추가적인 능력을 가지면서 함수 F 및 G를 수행하도록 구성되는 목적 코드가 생성될 수 있다.

[0045] V. 응용 프로그램

[0046] 본 명세서에 설명된 기술은 긴 실행 시간을 가진 프로그램에 매우 중요할 수 있는 고장 허용 한계(fault tolerance) 및 페일오버(failover) 방안을 제공할 수 있다. 이것은 계산의 크기가 증가함에 따라 손실 작업의 증가되는 비용 때문이다. 더욱 많은 하드웨어가 계산을 수행하는 데 사용될 때에 고장의 가능성이 높음에 따라 이러한 기술은 또한 대규모 컴퓨팅 자원을 이용하는 분산 프로그램에 매우 유용할 수 있다. 새로운 컴퓨팅 하드웨어의 동향은 하드웨어의 각 다음 세대가 기하 급수적으로 빨라지는 패러다임에서 각 다음 세대가 기하 급수적으로 더 많은 계산 코어를 갖는 패러다임으로 이동하였다. 본 명세서에 설명된 기술은 다수의 계산 코어 또는 머신의 클러스터에서 실행하는 멀티스레드 프로그램을 이용하여 상당량의 계산 작업을 수행할 수 있도록 하는 데 사용될 수 있다. 예를 들면, 이와 같은 기술은 다수의 머신 및/또는 다수의 코어가 프로그램 데이터를 처리할 시에 포함되는 클라우드 컴퓨팅 환경에서 이용될 수 있다.

[0047] 본 명세서에 설명된 기술의 이점은 프로그램이 하나의 프로세서/머신에서 시작될 수 있으며, 그 다음에 충돌 또는 다른 고장이 발생할 때 다른 프로세서/머신에서 처리가 계속할 수 있다는 것이다. 하나의 프로세서 또는 머신이 고장 나면, 처리는 중단을 최소화하면서 다른 프로세서 또는 머신에서 재개할 수 있다. 본 명세서에 설명된 기술이 운영 체제의 협력을 필요로 하지 않으므로, 일부 실시예에서, 처리는 다른 운영 체제를 실행하는 프

로세서/머신에서 계속할 수 있다. 적절한 버전이 새로운 아키텍처에서 컴파일되며 체크포인트에 대한 파일 형식이 아키텍처에서 쓰여질 수 있는 경우에 프로그램은 다른 운영 체제에서 실행할 수 있다

[0048] 도 5는 복수의 마이크로프로세서(502 및 503)를 가진 컴퓨팅 장치(501)의 일례를 도시한다. 실행을 중단시키는 충돌 또는 다른 고장이 발생할 때 전체 프로그램 또는 프로그램 스레드는 마이크로프로세서(502)에서 실행할 수 있다. 본 명세서에 설명된 기술을 이용하여, 프로그램 또는 프로그램 스레드는 프로그램의 실행 상태를 나타내는 영구 저장 장치(persistent storage)(504)에 저장된 데이터를 이용하여 마이크로프로세서(503)에서 프로그램의 실행을 계속할 수 있다.

[0049] 도 6은 마이크로프로세서(602)를 가진 컴퓨팅 장치(601) 및 마이크로프로세서(604)를 가진 컴퓨팅 장치(603)를 포함하는 컴퓨팅 환경의 일례를 도시한다. 실행을 중단시키는 충돌 또는 다른 고장이 발생할 때 프로그램 또는 프로그램 스레드는 마이크로프로세서(602)에서 실행할 수 있다. 본 명세서에 설명된 기술을 이용하여, 프로그램 또는 프로그램 스레드는 장치(603)의 마이크로프로세서(604)에서 프로그램의 실행을 계속할 수 있다. 장치(601 및 603)는 어떤 적절한 통신 연결부에 의해 연결될 수 있다. 장치(501, 601 및 603)는 예를 들어 범용 컴퓨터와 같은 어떤 적절한 컴퓨팅 장치, 또는 본 명세서에 설명된 다른 장치일 수 있다.

[0050] 따라서, 본 발명의 적어도 하나의 실시예의 여러 양태를 설명하였지만, 다양한 변경, 수정 및 개선은 당업자에게는 쉽게 발생하게 될 것이라는 점이 이해되어야 한다.

[0051] 이러한 변경, 수정 및 개선은 본 개시의 일부이며 본 발명의 사상 및 범위 내에 있도록 의도된다. 따라서, 상술한 설명 및 도면은 예일 뿐이다.

[0052] 본 발명의 상술한 실시예는 다양한 방식으로 구현될 수 있다. 예를 들면, 실시예는 하드웨어, 소프트웨어 또는 이의 조합을 이용하여 구현될 수 있다. 소프트웨어에서 구현되는 경우, 소프트웨어 코드는 단일의 컴퓨터에 제공되거나 다수의 컴퓨터 중에 분산되거나 간에 프로세서의 어떤 적절한 프로세서 또는 프로세서의 집합(collection)에서 실행될 수 있다. 이러한 프로세서는 집적 회로로서 구현될 수 있으며, 집적 회로 구성 요소에는 하나 이상의 프로세서가 있다. 하지만, 프로세서는 임의의 적절한 형식으로 회로를 이용하여 구현될 수 있다.

[0053] 더욱이, 컴퓨터는 랙 마운트(rack-mounted) 컴퓨터, 데스크톱 컴퓨터, 랩톱 컴퓨터 또는 태블릿 컴퓨터와 같은 다수의 형태로 실시될 수 있다는 점이 이해될 것이다. 추가적으로, 컴퓨터는 개인 휴대용 단말기(PDA), 스마트폰 또는 어떤 다른 적절한 휴대용 또는 고정 전자 장치를 포함하며 일반적으로 컴퓨터로 간주되지 않지만 적절한 처리 능력을 가진 장치에 내장될 수 있다.

[0054] 또한, 컴퓨터는 하나 이상의 입력 및 출력 장치를 가질 수 있다. 이러한 장치는 특히 사용자 인터페이스를 제공하는 데 사용될 수 있다. 사용자 인터페이스를 제공하는 데 사용될 수 있는 출력 장치의 예는 출력의 시각적 표현을 위한 프린터 또는 디스플레이 스크린, 및 출력의 청각적 표현을 위한 스피커 또는 다른 사운드 생성 장치를 포함한다. 사용자 인터페이스에 이용될 수 있는 입력 장치의 예는 키보드, 및 마우스, 터치 패드 및 디지타이징 태블릿과 같은 포인팅 장치를 포함한다. 다른 예로서, 컴퓨터는 음성 인식을 통해 또는 다른 청각적 형식으로 입력 정보를 수신할 수 있다.

[0055] 이러한 컴퓨터는 기업 네트워크 또는 인터넷과 같은 근거리 통신망 또는 광역 통신망을 포함하는 하나 이상의 네트워크에 의해 어떤 적절한 형식으로 상호 연결될 수 있다. 이러한 네트워크는 어떤 적절한 기술에 기초할 수 있고, 어떤 적절한 프로토콜에 따라 동작할 수 있으며, 무선 네트워크, 유선 네트워크 또는 광섬유 네트워크를 포함할 수 있다.

[0056] 또한, 본 명세서에 설명된 다양한 방법 또는 프로세스는 다양한 운영 체제 또는 플랫폼 중 어느 하나를 채용하는 하나 이상의 프로세서에서 실행할 수 있는 소프트웨어로 코딩될 수 있다. 추가적으로, 이러한 소프트웨어는 다수의 적절한 프로그래밍 언어 및/또는 프로그래밍 또는 스크립팅 도구를 이용하여 작성될 수 있으며, 또한 프레임워크 또는 가상 머신에서 실행되는 실행 가능한 머신 언어 코드 또는 중간 코드로 컴파일될 수 있다.

[0057] 이 점에서, 본 발명은 하나 이상의 컴퓨터 또는 다른 프로세서에서 실행될 때 본 발명의 다양한 실시예를 구현하는 방법을 수행하는 하나 이상의 프로그램으로 인코딩되는 컴퓨터 판독 가능 저장 매체(또는 다수의 컴퓨터 판독 가능한 매체)(예를 들어, 컴퓨터 메모리, 하나 이상의 플로피 디스크, 콤팩트 디스크(CD), 광 디스크, 디지털 비디오 디스크(DVD), 자기 테이프, 플래시 메모리, 필드 프로그래머블 게이트 어레이 또는 다른 반도체 장치의 회로 구성, 또는 다른 비일시적 유형(non-transitory, tangible) 컴퓨터 저장 매체)로 실시될 수 있다. 컴퓨터 판독 가능 저장 매체는 여기에 저장된 프로그램이 상술한 바와 같이 본 발명의 다양한 양태를 구현하기

위해 하나 이상의 서로 다른 컴퓨터 또는 다른 프로세서에 로딩될 수 있도록 수송할 수 있다. 여기에 사용된 바와 같이, 용어 "비일시적 컴퓨터 판독 가능 저장 매체(non-transitory computer-readable storage medium)"는 제조(즉, 제조품) 또는 머신인 것으로 간주될 수 있는 컴퓨터 판독 가능한 매체만을 포함한다. 대안적으로 또는 추가적으로, 본 발명은 전파 신호와 같은 컴퓨터 판독 가능 저장 매체와 다른 컴퓨터 판독 가능한 매체로 실시될 수 있다.

[0058] 용어 "프로그램(program)" 또는 "소프트웨어(software)"는 일반적인 의미로 여기서 상술한 바와 같이 본 발명의 다양한 양태를 구현하기 위해 컴퓨터 또는 다른 프로세서를 프로그래밍하는 데 사용될 수있는 어떤 타입의 컴퓨터 코드 또는 컴퓨터 실행 가능한 명령어의 세트를 나타내는 데 사용된다. 추가적으로, 본 실시예의 한 양태에 따르면, 실행될 때 본 발명의 방법을 수행하는 하나 이상의 컴퓨터 프로그램은 단일의 컴퓨터 또는 프로세서에 상주할 필요가 없지만, 본 발명의 다양한 양태를 구현하기 위해 다수의 서로 다른 컴퓨터 또는 프로세서 사이에서 모듈러 형식으로 분산될 수 있다.

[0059] 컴퓨터 실행 가능한 명령어는 하나 이상의 컴퓨터 또는 다른 장치에 의해 실행되는 프로그램 모듈과 같은 많은 형식일 수 있다. 일반적으로, 프로그램 모듈은 특정 태스크를 수행하거나 특정 추상 데이터 타입을 구현하는 루틴, 프로그램, 객체, 구성 요소, 데이터 구조 등을 포함한다. 전형적으로, 프로그램 모듈의 기능은 다양한 실시예에서 원하는대로 조합되거나 분산될 수 있다.

[0060] 본 발명의 다양한 양태는 단독, 조합 또는 다양한 배치로 이용될 수 있으며, 이는 특히 상술한 실시예에서 논의되지 않았으며, 따라서 상술한 설명에서 예시되거나 도면에 도시된 구성 요소의 상세 사항 및 배치에 대한 응용에 제한되지 않는다. 예를 들면, 일 실시예에서 설명된 양태는 다른 실시예에 설명된 양태와 어떤 방식으로 조합될 수 있다.

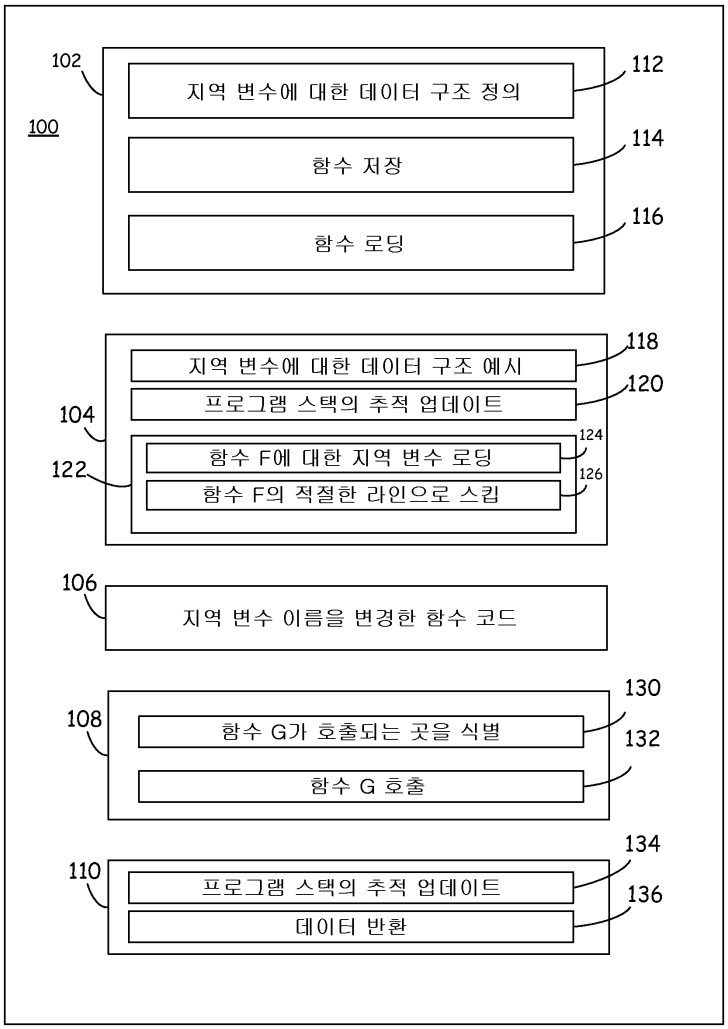
[0061] 또한, 본 발명은 그 예가 제공된 방법으로 실시될 수 있다. 방법의 부분으로 수행된 동작은 어떤 적절한 방식으로 순서화될 수 있다. 따라서, 예시적인 실시예에서 순차적인 동작으로 도시되었지만, 동작이 예시된 것과 다른 순서로 수행되며 일부 동작을 동시에 수행하는 것을 포함할 수 있는 실시예가 구성될 수 있다.

[0062] 청구항의 구성 요소를 수식하기 위해 청구범위에서 "제 1", "제 2", "제 3" 등과 같은 서수 용어의 사용은 그 자체로 하나의 청구항 구성 요소의 다른 구성 요소에 대한 어떤 우선 순위, 우선권, 또는 순서를 의미하거나 또는 방법의 동작이 수행되는 시간적 순서를 의미하지 않으며, 단지 청구항의 구성 요소를 구별하기 위해 어떤 이름을 가진 하나의 청구항의 구성 요소와 동일한 이름을 가진 다른 구성 요소를 구별하도록(하지만 서수 용어의 사용을 위해서) 표시로 사용된다.

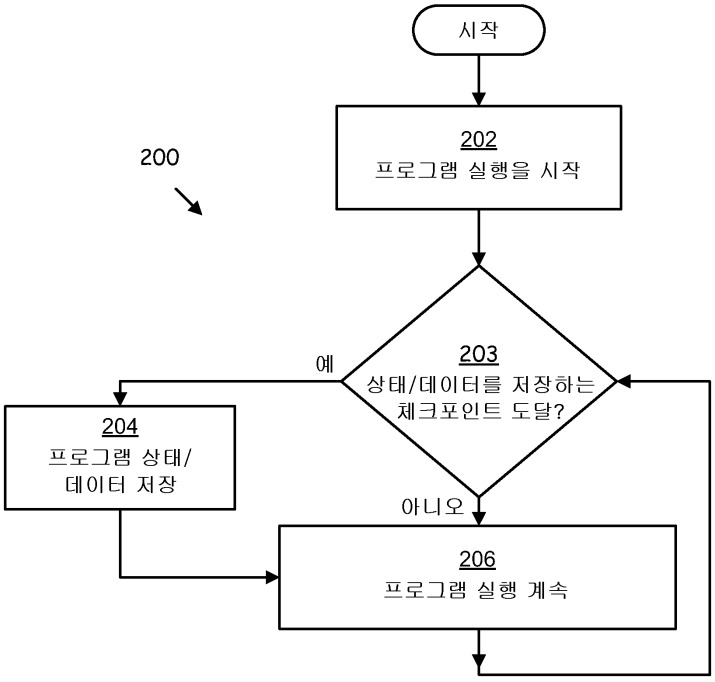
[0063] 또한, 본 명세서에 사용된 어법 및 전문 용어는 설명을 위한 것이며, 제한하는 것으로 간주되지 않아야 한다. 본 명세서에서 "포함하는(including, comprising)" 또는 "가진(having, containing, involving)" 및 이의 변형의 사용은 그 뒤에 열거되는 항목 및 균등물뿐만 아니라 추가적인 항목을 포함하는 것을 의도한다.

도면

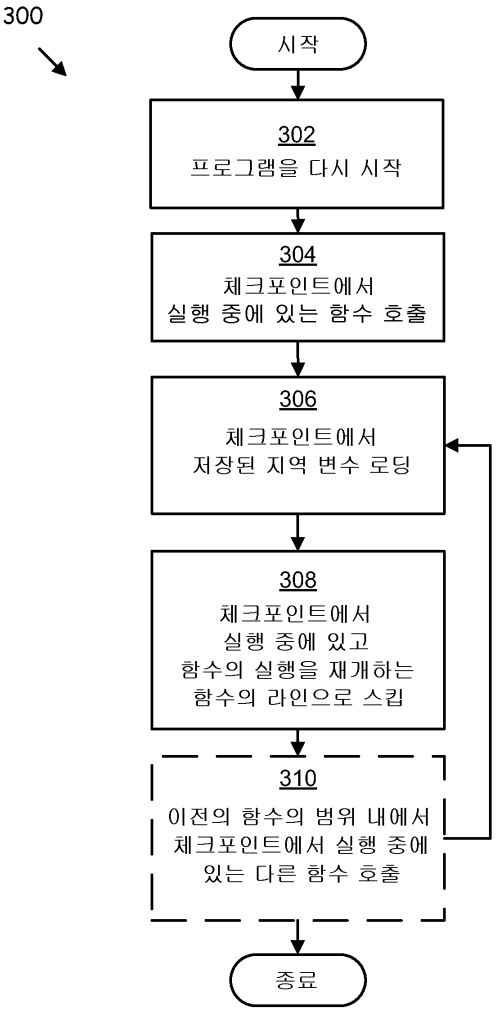
도면1


100
102
지역 변수에 대한 데이터 구조 정의
112
함수 저장
114
함수 로딩
116
104
지역 변수에 대한 데이터 구조 예시
118
프로그램 스택의 추적 업데이트
120
122
함수 F에 대한 지역 변수 로딩
124
함수 F의 적절한 라인으로 스킵
126
106
지역 변수 이름을 변경한 함수 코드
108
함수 G가 호출되는 곳을 식별
130
함수 G 호출
132
110
프로그램 스택의 추적 업데이트
134
데이터 반환
136

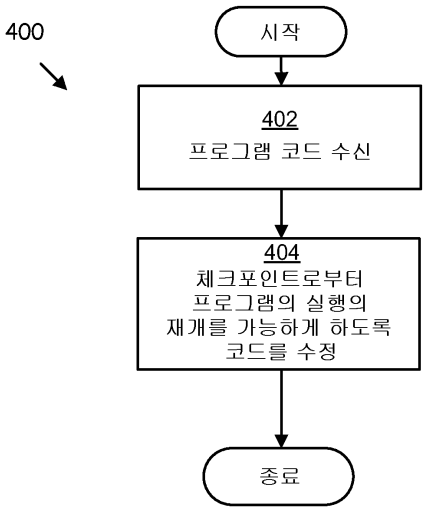
도면2


시작
200
202
프로그램 실행을 시작
203
상태/데이터를 저장하는
체크포인트 도달?
예
204
프로그램 상태/
데이터 저장
아니오
206
프로그램 실행 계속

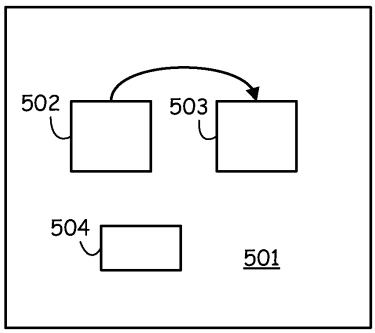
도면3


300
시작
302
프로그램을 다시 시작
304
체크포인트에서
실행 중에 있는 함수 호출
306
체크포인트에서
저장된 지역 변수 로딩
308
체크포인트에서
실행 중에 있고
함수의 실행을 재개하는
함수의 라인으로 스킵
310
이전의 함수의 범위 내에서
체크포인트에서 실행 중에
있는 다른 함수 호출
종료

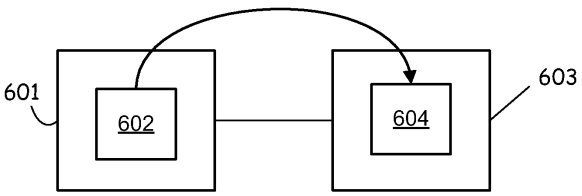
도면4


400
시작
402
프로그램 코드 수신
404
체크포인트로부터
프로그램의 실행의
재개를 가능하게 하도록
코드를 수정
종료

도면5


502
503
504
501

도면6


601
602
604
603