【서지사항】

【서류명】 특허출원서

【참조번호】 YS24L005P

【출원구분】 특허출원

【출원인】

【명칭】 연세대학교 산학협력단

【특허고객번호】 2-2005-009509-9

【대리인】

【성명】 정부연

【대리인번호】 9-2007-001006-1

【발명의 국문명칭】 블록 샘플 기반의 애플리케이션 성능 개선 장치 및 방법

【발명의 영문명칭】 DEVICE AND METHOD FOR IMPROVING APPLICATION PERFORMANCE BASED ON BLOCK SAMPLES

【발명자】

【성명】 정진규

【성명의 영문표기】 Jinkyu Jeong

【주민등록번호】 820126-1XXXXXX

【우편번호】 06288

【주소】 서울시 강남구 삼성로 150, 208동 602호

【출원언어】 국어

【심사청구】 청구

【공지예외적용대상증명서류의 내용】

【공개형태】 논문 발표

【공개일자】 2024.07.12

【이 발명을 지원한 국가연구개발사업】

【과제고유번호】 1711201093

【과제번호】 00321688

【부처명】 과학기술정보통신부

【과제관리(전문)기관명】 한국연구재단

【연구사업명】 원천기술개발사업

【연구과제명】 한국형 엑사스케일 응용 SW 개발 환경 (KEASE) 프레임워크 개발

【과제수행기관명】 연세대학교산학협력단

【연구기간】 2024.05.01 ~ 2025.04.30

【이 발명을 지원한 국가연구개발사업】

【과제고유번호】 2710006677

【과제번호】 RS-2020-II201361

【부처명】 과학기술정보통신부

【과제관리(전문)기관명】 정보통신기획평가원

【연구사업명】 정보통신방송혁신인재양성(R&D)

【연구과제명】 인공지능대학원지원(연세대학교)

【과제수행기관명】 연세대학교 산학협력단

【연구기간】 2024.01.01 ~ 2024.12.31

2024-10-24

【취지】 위와 같이 특허청장에게 제출합니다.

대리인 정부연 (서명 또는 인)

【수수료】

【출원료】	0	면	46,000 원
【가산출원료】	39	면	0 원
【우선권주장료】	0	건	0 원
【심사청구료】	12	항	778,000 원
【합계】	824,000원		
【감면사유】	전담조직(50%감면)[1]		
【감면후 수수료】	412,000 원		

【첨부서류】 1.공지예외적용대상(신규성상실의예외, 출원시의특례)규정을 적용받기 위한 증명서류_1통 2.공지예외적용대상(신규성상실의예외, 출원시의특례)규정을 적용받기 위한 증명서류_1통 3.위임장_1통

1 : 공지예외적용대상(신규성상실의예외,_출원시의특례)규정을_적용받기_위한_증명서류

PDF 파일 첨부

2 : 공지예외적용대상(신규성상실의예외,_출원시의특례)규정을_적용받기_위한_증명서류

PDF 파일 첨부

3 : 위임장

PDF 파일 첨부

【발명의 설명】

【발명의 명칭】

블록 샘플 기반의 애플리케이션 성능 개선 장치 및 방법{DEVICE AND METHOD FOR IMPROVING APPLICATION PERFORMANCE BASED ON BLOCK SAMPLES}

【기술분야】

【0001】 본 발명은 애플리케이션 성능 개선 기술에 관한 것으로, 보다 상세하게는 온-CPU 이벤트 및 오프-CPU 이벤트를 통합적으로 인과관계 프로파일링 하여 가상성능향상을 수행할 수 있는 블록 샘플 기반의 애플리케이션 성능 개선 장치 및 방법에 관한 것이다.

【발명의 배경이 되는 기술】

【0003】 애플리케이션 프로파일링은 두가지 유형의 이벤트, 즉 온-CPU 이벤트(on-CPU event)과 오프-CPU 이벤트(off-CPU event) 분석을 포함한다. 온-CPU 이벤트는 CPU에서 수행되는 명령(instruction)을 의미한다. 여기에는 연산(ALU, FPU) 및 메모리 접근(load, store) 명령이 포함되며, 명령을 수행하다가 스레드가 블록(block) 되는 구간은 포함되지 않는다. 오프-CPU 이벤트는 스레드가 명령을 수행하다가 블록 되는 구간을 의미한다. 온-CPU 이벤트와 오프-CPU 이벤트는 보완적(complemental)이다. 오프-CPU 이벤트는 저장장치 블로킹(blocking) 입출력, 스레드 간 동기화(synchronization), CPU 스케줄링에 의한 대기 등 CPU에서 실제로 실

행되고 있지 않은 구간을 포함하고 있다.

【0004】 인과관계 프로파일링은 애플리케이션을 실제로 최적화하지 않고, 최적화했을 때의 효과를 확인하는 프로파일링 기법이다.

【0005】 최신(state-of-the-art) 인과관계 프로파일러(이하, 'COZ' 라함)는 가상성능향상(virtual speedup) 기반의 인과관계 프로파일링을 수행한다. 가상성능향상은 성능 향상 예측 대상 작업이 실제 최적화한 것과 같은 효과를 재현하기 위해 의도적으로 동시에 실행되고 있는 다른 스레드에게 지연을 주입한다. 애플리케이션 실행 종료 시, 의도한 지연의 양과 실제 애플리케이션이 느려진 정도의 차이를 통해 작업의 성능 향상 예측치를 계산한다.

【0006】 스레드 간 의존성(dependency)이 존재하는 경우, 가상성능향상 수행 시 주입되는 의도적인 지연은 성능 향상 예측 결과의 오류를 초래할 수 있다. 이는 다른 스레드의 작업 완료를 대기하고 있는 스레드는 이중으로 지연을 받을 수 있기 때문이다.

【0007】 기존 COZ는 이와 같은 상황을 해결하기 위해 의존성이 존재하여 대기중인 스레드는 블록된 구간 동안 누적된 지연을 면제함으로써 성능 향상 예측을 올바르게 수행할 수 있다.

【0008】 COZ는 병렬 프로그램 코드에 최적화가 필요한 지점을 특정해준다는 점에서 유용한 인과관계 프로파일러이다. COZ는 가상성능향상을 수행하기 위해 IP(instruction pointer) 및 콜체인(callchain)과 같은 애플리케이션의 실행 정보

를 수집하고 주기적으로 이를 확인하여 성능 향상 예측 대상 작업이 수행되고 있는지 여부를 판단한다. 만약, 대상 작업이 수행되고 있었다면 의도적인 지연을 동시에 실행되고 있는 다른 스레드들에게 부여한다.

【0009】 하지만, COZ는 애플리케이션 실행 정보를 리눅스 perf 서브시스템을 활용한 샘플링에 의존하고 있기 때문에 오프-CPU 이벤트에 대한 성능 향상 예측이 불가능하다는 한계점이 존재한다. 샘플링은 주기적으로 IP 및 콜체인을 수집하고 있지만, 실행되던 스레드가 시스템 콜(system call) 호출하여 블록 되는 경우 비활성화 되어 블록 구간 동안의 실행 정보 수집이 불가능하다. 따라서, COZ는 오프-CPU 이벤트 정보가 존재하지 않아 가상성능향상 대상에 오프-CPU 이벤트가 포함될 수 없기 때문에 온-CPU 및 오프-CPU 이벤트가 혼재하는 애플리케이션에 대해서 유용한 결과를 제공할 수 없다.

【선행기술문헌】

【특허문헌】

【0011】 **(특허문헌 0001)** 한국공개특허 제10-2016-0003502호 (2016.01.11)

【발명의 내용】

【해결하고자 하는 과제】

【0012】본 발명의 일 실시예는 블록 샘플 기반의 애플리케이션 샘플링을 수행하여 통합된 온-CPU 및 오프-CPU 이벤트에 대한 가상성능향상을 수행할 수 있는 블록 샘플 기반의 애플리케이션 성능 개선 장치 및 방법을 제공하고자 한다.

【0013】본 발명의 일 실시예는 오프-CPU 이벤트를 가상성능향상하기 위한 지연을 오프-CPU 이벤트가 완료된 직후 처리하도록 하여 의존성 처리에 의해 스레드가 올바르게 지연을 면제받을 수 있는 블록 샘플 기반의 애플리케이션 성능 개선 장치 및 방법을 제공하고자 한다.

【과제의 해결 수단】

【0015】실시예들 중에서, 블록 샘플 기반의 애플리케이션 성능 개선 장치는 스레드가 CPU 실행 상태에 있든 또는 블로킹 상태에 있든 관계없이 발생되는 이벤트를 샘플링하여 블록 샘플들(blocked samples)을 생성하는 블록 샘플 생성부; 상기 블록 샘플들의 온-CPU 이벤트와 오프-CPU 이벤트를 통합적으로 분석하고 애플리케이션의 성능 병목을 식별하는 제1 프로파일러; 상기 온-CPU와 상기 오프-CPU 이벤트 간의 상호 의존성을 분석하여 특정 이벤트의 가상 최적화를 통한 상기 성능 병목에 대한 성능 향상을 예측하는 제2 프로파일러; 및 상기 특정 이벤트의 최적화 전략을 생성하는 최적화 전략 생성부를 포함한다.

【0016】상기 블록 샘플 생성부는 상기 블로킹 상태인 경우로서 I/O 대기, 동기화 대기 또는 스케줄링 대기로 인해 상기 스레드가 CPU에서 실행되지 못하는

상태에서도 상기 이벤트를 샘플링할 수 있다.

【0017】 상기 블록 샘플 생성부는 상기 스레드가 스케쥴 아웃 및 스케쥴 인 시점에서 타이머를 통해 상기 스레드의 상태를 확인하여 상기 블록 샘플들을 샘플링하여 블로킹 이벤트를 기록할 수 있다.

【0018】 상기 블록 샘플 생성부는 상기 블로킹 이벤트의 중복을 줄이기 위해 반복횟수를 나타내는 가중치를 기록하여 동일한 속성의 블로킹 이벤트를 하나의 블로킹 이벤트로 묶어 처리할 수 있다.

【0019】 상기 블록 샘플 생성부는 상기 스레드가 스케쥴 아웃될 때 상기 스레드가 블로킹되기 직전 실행된 명령어 주소(IP), 상기 스레드가 호출한 함수들의 스택 트레이스를 나타내는 호출체인, 블로킹 이벤트의 유형을 기록하는 타입 및 상기 스레드가 블로킹된 시점을 나타내는 블로킹 타임스탬프를 상기 블로킹 이벤트에 저장하고, 상기 블로킹 이벤트의 유형은 상기 I/O 대기, 상기 동기화 대기 또는 상기 스케줄링 대기 중 하나에 해당할 수 있다.

【0020】 상기 블록 샘플 생성부는 상기 스레드가 웨이크-업될 때 상기 스레드의 대기 시간과 대기 이유를 추적하기 위해 웨이크-업 타임스탬프를 상기 블로킹 이벤트에 저장할 수 있다.

【0021】 상기 블록 샘플 생성부는 상기 스레드가 스케쥴 인될 때 블로킹 이벤트의 전체 지속 시간(Tblocked)과 스케줄링 대기 시간(Tsched)을 계산하기 위해 스케쥴-인 타임스탬프를 상기 블로킹 이벤트에 저장할 수 있다.

【0022】상기 제1 프로파일러는 블로킹 이벤트에서 상기 온-CPU 이벤트와 상기 오프-CPU 이벤트를 결정하고 각 이벤트가 애플리케이션 실행 중 차지하는 오버헤드 정보를 분석할 수 있다.

【0023】상기 제1 프로파일러는 상기 오버헤드 정보의 분석을 통해 I/O 대기, 동기화 대기 또는 스케줄링 대기에 따른 서브클래스로 분류하고 상기 서브클래스의 성능 병목을 결정하여 상기 애플리케이션의 성능 병목을 식별할 수 있다.

【0024】상기 제2 프로파일러는 상기 온-CPU와 상기 오프-CPU 이벤트 간의 인과관계 분석을 수행하고 가상 속도 향상 (Virtual Speedup) 기법을 통해 특정 이벤트를 가상으로 가속화하여 상기 성능 향상을 예측할 수 있다.

【0025】상기 최적화 전략 생성부는 상기 특정 이벤트에서 병목을 일으키는 코드의 최적화 전략을 생성할 수 있다.

【0026】실시예들 중에서, 블록 샘플 기반의 애플리케이션 성능 개선 방법은 블록 샘플 기반의 애플리케이션 성능 개선 장치에서 수행되는 블록 샘플 기반의 애플리케이션 성능 개선 방법에 있어서, 스레드가 CPU 실행 상태에 있든 또는 블로킹 상태에 있든 관계없이 발생되는 이벤트를 샘플링하여 블록 샘플들(blocked samples)을 생성하는 블록 샘플 생성단계; 상기 블록 샘플들의 온-CPU 이벤트와 오프-CPU 이벤트를 통합적으로 분석하고 애플리케이션의 성능 병목을 식별하는 제1 프로파일링 단계; 상기 온-CPU와 상기 오프-CPU 이벤트 간의 상호 의존성을 분석하여 특정 이벤트의 가상 최적화를 통한 상기 성능 병목에 대한 성능 향상을 예측하

는 제2 프로파일링 단계; 및 상기 특정 이벤트의 최적화 전략을 생성하는 최적화 전략 생성단계를 포함한다.

【발명의 효과】

【0028】 개시된 기술은 다음의 효과를 가질 수 있다. 다만, 특정 실시예가 다음의 효과를 전부 포함하여야 한다거나 다음의 효과만을 포함하여야 한다는 의미는 아니므로, 개시된 기술의 권리범위는 이에 의하여 제한되는 것으로 이해되어서는 아니 될 것이다.

【0029】 본 발명의 일 실시예에 따른 블록 샘플 기반의 애플리케이션 성능 개선 장치 및 방법은 블록 샘플 기반의 애플리케이션 샘플링을 수행하여 통합된 온-CPU 및 오프-CPU 이벤트에 대한 가상성능향상을 수행할 수 있다.

【0030】 본 발명의 일 실시예에 따른 블록 샘플 기반의 애플리케이션 성능 개선 장치 및 방법은 오프-CPU 이벤트를 가상성능향상하기 위한 지연을 오프-CPU 이벤트가 완료된 직후 처리하도록 하여 의존성 처리에 의해 스레드가 올바르게 지연을 면제받을 수 있다.

【0031】 따라서, 본 발명은 가상성능향상 대상에 오프-CPU 이벤트가 포함될 수 있으며, 의존성 처리에 의해 스레드가 올바르게 지연을 면제받을 수 있고, 애플리케이션의 성능 최적화가 애플리케이션 코드 최적화뿐만 아니라 애플리케이션 실행과 연관된 시스템 코드 혹은 실행에 활용되는 디바이스(저장장치, 가속기 등)의

성능 최적화를 통해 이뤄질 수 있다.

【도면의 간단한 설명】

【0033】 도 1은 컴퓨터 시스템의 구성을 설명하는 도면이다.

도 2는 본 발명에 따른 애플리케이션 성능 개선 장치를 설명하는 도면이다.

도 3은 본 발명에 따른 블록 샘플 기반의 애플리케이션 성능 개선 방법을 설명하는 순서도이다.

도 4는 본 발명에 따른 블록 샘플 생성 과정의 일 실시예를 설명하는 도면이다.

도 5는 본 발명에 따른 제1 프로파일링 결과의 일 실시예를 설명하는 도면이다.

도 6은 본 발명에 따른 제2 프로파일링 수행의 일 실시예를 설명하는 도면이다.

도 7은 본 발명에 따른 제2 프로파일링 결과의 일 실시예를 설명하는 도면이다.

도 8 및 도 9는 본 발명에 따른 블록 샘플 기반의 애플리케이션 성능 개선 방법에 관한 실험 결과를 설명하는 도면이다.

【발명을 실시하기 위한 구체적인 내용】

【0034】본 발명에 관한 설명은 구조적 내지 기능적 설명을 위한 실시예에 불과하므로, 본 발명의 권리범위는 본문에 설명된 실시예에 의하여 제한되는 것으로 해석되어서는 아니 된다. 즉, 실시예는 다양한 변경이 가능하고 여러 가지 형태를 가질 수 있으므로 본 발명의 권리범위는 기술적 사상을 실현할 수 있는 균등물들을 포함하는 것으로 이해되어야 한다. 또한, 본 발명에서 제시된 목적 또는 효과는 특정 실시예가 이를 전부 포함하여야 한다거나 그러한 효과만을 포함하여야 한다는 의미는 아니므로, 본 발명의 권리범위는 이에 의하여 제한되는 것으로 이해되어서는 아니 될 것이다.

【0035】한편, 본 출원에서 서술되는 용어의 의미는 다음과 같이 이해되어야 할 것이다.

【0036】"제1", "제2" 등의 용어는 하나의 구성요소를 다른 구성요소로부터 구별하기 위한 것으로, 이들 용어들에 의해 권리범위가 한정되어서는 아니 된다. 예를 들어, 제1 구성요소는 제2 구성요소로 명명될 수 있고, 유사하게 제2 구성요소도 제1 구성요소로 명명될 수 있다.

【0037】어떤 구성요소가 다른 구성요소에 "연결되어"있다고 언급된 때에는, 그 다른 구성요소에 직접적으로 연결될 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성요소가 다른 구성요소에 "직접 연결되어"있다고 언급된 때에는 중간에 다른 구성요소가 존재하지 않는 것으로 이해되어야 할 것이다. 한편, 구성요소들 간의 관계를 설명하는 다른 표현들, 즉 "~사이에"와 "바로 ~사이에" 또는 "~에 이웃하는"과 "~에 직접 이웃하는" 등도

마찬가지로 해석되어야 한다.

【0038】 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한 복수의 표현을 포함하는 것으로 이해되어야 하고, "포함하다"또는 "가지다" 등의 용어는 실시된 특징, 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것이 존재함을 지정하려는 것이며, 하나 또는 그 이상의 다른 특징이나 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

【0039】 각 단계들에 있어 식별부호(예를 들어, a, b, c 등)는 설명의 편의를 위하여 사용되는 것으로 식별부호는 각 단계들의 순서를 설명하는 것이 아니며, 각 단계들은 문맥상 명백하게 특정 순서를 기재하지 않는 이상 명기된 순서와 다르게 일어날 수 있다. 즉, 각 단계들은 명기된 순서와 동일하게 일어날 수도 있고 실질적으로 동시에 수행될 수도 있으며 반대의 순서대로 수행될 수도 있다.

【0040】 본 발명은 컴퓨터가 읽을 수 있는 기록매체에 컴퓨터가 읽을 수 있는 코드로서 구현될 수 있고, 컴퓨터가 읽을 수 있는 기록 매체는 컴퓨터 시스템에 의하여 읽혀질 수 있는 데이터가 저장되는 모든 종류의 기록 장치를 포함한다. 컴퓨터가 읽을 수 있는 기록 매체의 예로는 ROM, RAM, CD-ROM, 자기 테이프, 플로피 디스크, 광 데이터 저장 장치 등이 있다. 또한, 컴퓨터가 읽을 수 있는 기록 매체는 네트워크로 연결된 컴퓨터 시스템에 분산되어, 분산 방식으로 컴퓨터가 읽을 수 있는 코드가 저장되고 실행될 수 있다.

【0041】 여기서 사용되는 모든 용어들은 다르게 정의되지 않는 한, 본 발명

이 속하는 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 것으로 해석되어야 하며, 본 출원에서 명백하게 정의하지 않는 한 이상적이거나 과도하게 형식적인 의미를 지니는 것으로 해석될 수 없다.

【0043】 도 1은 컴퓨터 시스템의 구성을 설명하는 도면이다.

【0044】 도 1을 참조하면, 컴퓨터 시스템(100)은 구성 요소의 급속한 발전으로 인해 CPU 외부 즉, 가속기, 저장장치, 네트워크 장치 등의 오프-CPU에서 실행되는 이벤트가 더욱 다양해졌다. 온-CPU 이벤트는 CPU 내부에서 수행되는 명령을 의미한다. 오프-CPU 이벤트는 CPU에서 명령이 수행되는 동안 대기하게 된다.

【0045】 이러한 컴퓨팅 환경은 애플리케이션의 동작을 더욱 복잡하게 만들어 병목 현상이 다양화되고 있다.

【0046】 본 발명은 CPU 내부 및 외부 이벤트를 동시에 프로파일링하여 성능 병목 현상을 식별할 수 있도록, 블록 샘플(Blocked samples) 이라는 샘플링 기법을 통해 두 이벤트를 동시에 샘플링하고 통계 기반의 프로파일러 및 인과관계 기반의 프로파일러를 통해 통합적으로 분석하여 성능 병목을 식별하고 최적화 성능 개선을 예측할 수 있는 기법을 제안한다.

【0047】

【0048】도 2는 본 발명에 따른 애플리케이션 성능 개선 장치를 설명하는 도면이다.

【0049】도 2를 참조하면, 애플리케이션 성능 개선 장치(200)는 애플리케이션 성능 최적화 전략을 생성하기 위한 본 발명에 따른 블록 샘플 기반의 애플리케이션 성능 개선 방법을 수행할 수 있다. 이를 위하여, 애플리케이션 성능 개선 장치(200)는 컴퓨터 시스템(100)의 내부에 복수의 기능적 구성들을 포함하여 구현될 수 있다. 구체적으로, 애플리케이션 성능 개선 장치(200)는 블록 샘플 생성부(210), 제1 프로파일러(230), 제2 프로파일러(250), 최적화 전략 생성부(270) 및 제어부(290)를 포함할 수 있다.

【0050】이때, 애플리케이션 성능 개선 장치(200)는 상기의 기능적 구성들을 동시에 모두 포함해야 하는 것은 아니며, 각각의 실시예에 따라 상기의 구성들 중 일부를 생략하거나, 상기의 구성들 중 일부 또는 전부를 선택적으로 포함하여 구현될 수도 있다. 이하, 각 기능적 구성들의 동작을 구체적으로 설명한다.

【0051】블록 샘플 생성부(210)는 스레드(thread)가 CPU 실행 상태에 있든 또는 블로킹 상태에 있든 관계없이 발생되는 이벤트를 샘플링하여 블록 샘플들(blocked samples)을 생성할 수 있다. 여기에서, 블로킹 상태는 스레드가 CPU에서 실행되지 못하고 대기하는 상태를 의미한다. 블록 샘플 생성부(110)는 블로킹 상태인 경우로서 I/O 대기, 동기화 대기 또는 스케줄링 대기로 인해 스레드가 CPU에서 실행되지 못하는 상태에서도 이벤트를 샘플링할 수 있다. I/O 대기는 파일 읽기/쓰기, 네트워크 통신 등과 같은 I/O 작업이 완료되기를 기다리는 상태이다. 동

기화 대기는 여러 스레드가 공유 자원을 사용할 때, 자원을 안전하게 사용하기 위해 동기화가 필요한데, 이때 자원이 잠겨 있는 동안 다른 스레드는 대기하는 것을 말한다. 스케줄링 대기는 운영체제의 스케줄러가 스레드를 실행하기 위해 CPU 자원을 할당하지 않았을 때 발생하는 대기 상태로, CPU에서 실행 가능한 다른 스레드들이 먼저 실행되고 있으면 대기해야 한다.

【0052】 블록 샘플 생성부(210)는 스레드가 스케쥴 아웃 및 스케쥴 인 시점에서 타이머를 통해 스레드의 상태를 확인하여 블록 샘플들을 샘플링하여 블로킹 이벤트를 기록할 수 있다. 스케쥴 아웃은 스레드가 현재 CPU에서 실행되고 있다가 더 이상 실행되지 않고 대기 상태로 넘어가는 과정으로, 운영체제의 스케줄러가 해당 스레드를 CPU에서 내리고 다른 스레드를 실행할 때 발생한다. 스케쥴 인은 대기 중이던 스레드가 다시 CPU에서 실행되도록 선택되는 과정으로, 스레드는 대기 상태에서 벗어나 CPU에서 작업을 처리하게 된다. 블록 샘플 생성부(210)는 스케쥴 인 시점에 스레드의 상태를 체크하여 스레드가 CPU에서 실행 중인지 아니면 블로킹 상태인지를 확인하여 블로킹 상태일 때 해당 상태를 샘플링하여 블로킹 이벤트로서, 스레드가 왜 블로킹 상태에 있는지(예: I/O 대기, 동기화 대기, 스케줄링 대기 등), 그리고 이 상태가 얼마나 지속되었는지를 기록할 수 있다.

【0053】 블록 샘플 생성부(210)는 블로킹 이벤트의 중복을 줄이기 위해 반복 횟수를 나타내는 가중치를 기록하여 동일한 속성의 블로킹 이벤트를 하나의 블로킹 이벤트로 묶어 처리할 수 있다. 블로킹 이벤트는 스레드가 CPU에서 실행되지 못하고 대기 상태에 있을 때 발생하는 이벤트로서, 스레드가 블로킹 된 길이는 짧을수

도 길수도 있다. 블로킹 상태가 오래 유지되는 경우, 이를 각각의 개별 이벤트로 기록하면 중복된 데이터가 과도하게 생성될 수 있어 분석에 드는 리소스도 늘어나게 된다. 여기에서, 동일한 속성은 블로킹 이벤트의 원인 및 상황이 같은 경우를 의미한다. 블록 샘플 생성부(210)는 블로킹 이벤트가 긴 경우 가중치를 기록하여 불필요한 중복을 방지하면서도 이벤트 발생 횟수에 대한 통계적 정보를 유지할 수 있다.

【0054】또한, 블록 샘플 생성부(210)는 스레드가 스케쥴 아웃될 때 스레드가 블로킹되기 직전 실행된 명령어 주소(IP), 스레드가 호출된 함수들의 스택 트레이스를 나타내는 호출체인, 블로킹 이벤트의 유형을 기록하는 타입 및 스레드가 블로킹된 시점을 나타내는 블로킹 타임스탬프를 블로킹 이벤트에 저장할 수 있다. 블로킹 이벤트의 유형은 I/O 대기, 동기화 대기 또는 스케줄링 대기 중 하나에 해당할 수 있다. 블록 샘플 생성부(210)는 스레드가 스케쥴 아웃되는 시점에서 스레드가 CPU에서 실행되지 않게 되는 이유를 파악하고 그 상태를 기록할 수 있다. 이때 기록되는 정보는 스레드의 상태 변화를 분석하는 데 사용될 수 있다. 여기에서, 명령어 주소는 스레드가 CPU에서 마지막으로 실행한 명령어의 메모리 주소이며, 스레드가 어느 부분에서 블로킹 상태로 전환되었는지를 추적하는 데 사용될 수 있고, 특히 프로그램의 어떤 명령어가 실행 중에 스레드가 대기 상태로 전환되었는지 확인함으로써 성능 병목이 발생하는 위치를 파악할 수 있다. 호출체인은 스레드가 현재 실행 중인 함수가 호출된 경로(스택 트레이스)를 의미하며, 스택 트레이스를 통해 블로킹 이벤트가 발생했을 때 해당 이벤트가 어떤 함수의 호출 경로

에서 일어났는지 파악할 수 있다. 예를 들어, 함수 A가 함수 B를 호출하고, B가 다시 함수 C를 호출한 상황이라 가정하면, 스택 트레이스는 A→B→C 가 된다. 즉, 블로킹 이벤트에는 명령어 주소(블로킹되기 직전 실행된 명령어 위치), 호출체인(블로킹이 발생한 함수 호출 경로), 블로킹 유형(I/O 대기, 동기화 대기, 스케줄링 대기 중 하나), 타임스탬프(블로킹이 발생한 시점)가 저장될 수 있다. 블로킹 이벤트에 저장된 정보들은 성능 분석에 활용될 수 있다.

【0055】블록 샘플 생성부(210)는 스레드가 웨이크-업될 때 스레드의 대기 시간과 대기 이유를 추적하기 위해 웨이크-업 타임스탬프를 블록킹 이벤트에 저장할 수 있다. 또한, 블록 샘플 생성부(210)는 스레드가 스케쥴 인될 때 블로킹 이벤트의 전체 지속 시간(Tblocked)과 스케줄링 대기 시간(Tsched)을 계산하기 위해 스케쥴-인 타임스탬프를 블로킹 이벤트에 저장할 수 있다.

【0056】제1 프로파일러(230)는 블록 샘플들의 온-CPU 이벤트와 오프-CPU 이벤트를 통합적으로 분석하고 애플리케이션의 성능 병목을 식별할 수 있다. 제1 프로파일러(230)는 블로킹 이벤트에서 온-CPU 이벤트와 오프-CPU 이벤트를 결정하고 각 이벤트가 애플리케이션 실행 중 차지하는 오버헤드 정보를 분석할 수 있다. 애플리케이션 성능은 CPU에서 실행되는 시간과 대기 시간의 균형이 중요할 수 있다. 예를 들어, CPU에서 너무 많은 시간을 사용하면 계산이 복잡하다는 의미일 수 있고, 오프-CPU 이벤트가 너무 많으면 I/O 대기나 자원 대기가 성능 병목이 된다는 의미일 수 있다. 제1 프로파일러(230)는 블로킹 이벤트 기록을 분석하여 애플리케이션에서 발생하는 온-CPU 이벤트와 오프-CPU 이벤트를 결정하고, 이벤트 각각이

애플리케이션 실행 중 차지하는 오버헤드를 분석하여 성능 병목을 식별할 수 있다. 여기에서, 온-CPU 오버헤드는 스레드가 CPU에서 실행되는 동안 과도한 CPU 사용으로 인해 발생하는 지연을 의미한다. 오프-CPU 오버헤드는 스레드가 블로킹 상태에 있는 동안 발생하는 지연을 의미한다.

【0057】 제1 프로파일러(230)는 오버헤드 정보의 분석을 통해 I/O 대기, 동기화 대기 또는 스케줄링 대기에 따른 서브클래스로 분류하고 서브클래스의 성능 병목을 결정하여 애플리케이션의 성능 병목을 식별할 수 있다. 제1 프로파일러(230)는 오버헤드 정보를 분석하여 애플리케이션 성능 저하에 영향을 미치는 블로킹 이벤트의 유형을 식별하고 이들할 서브클래스로 분류할 수 있다. 제1 프로파일러(230)는 각 서브클래스에 속하는 대기 시간을 분석하여 I/O 대기, 동기화 대기 또는 스케줄링 대기 중 어느 것이 성능 병목의 원인인지를 확인할 수 있다.

【0058】 제2 프로파일러(250)는 온-CPU와 오프-CPU 이벤트 간의 상호 의존성을 분석하여 특정 이벤트의 최적화를 통한 성능 병목에 대한 성능 향상을 예측할 수 있다. 제2 프로파일러(250)는 온-CPU와 오프-CPU 이벤트 간의 인과관계 분석을 수행하고 가상 속도 향상 (Virtual Speedup) 기법을 통해 특정 이벤트를 가상으로 가속화하여 성능 향상을 예측할 수 있다. 제2 프로파일러(250)는 오프-CPU 이벤트뿐만 아니라 온-CPU 이벤트들의 상호작용을 분석함으로써, I/O 장치 업그레이드나 CPU 코어 추가와 같은 다양한 최적화 방안을 탐색할 수 있다. 또한, 제2 프로파일러(250)는 특정 코드 라인뿐만 아니라 오프-CPU 서브클래스를 타겟으로 하여 가상 속도 향상을 예측함으로써 다양한 최적화 전략을 탐색할 수 있다. 가상 속도 향상

기법은 코드의 특정 라인을 최적화한 경우 성능이 어떻게 변할지를 가상으로 실험하는 방식이다. 제2 프로파일러(250)는 온-CPU 이벤트 뿐만 아니라 오프-CPU 이벤트에도 가상 속도 향상 기법을 적용하여 스레드 간의 의존성을 관리하고 가상 속도 향상을 위한 지연 시간 주입을 처리함으로써 실제로 코드 라인이 최적화되었을 때와 유사한 성능 변화를 예측할 수 있다.

【0059】 애플리케이션 성능 개선 장치(200)는 제1 프로파일러(230) 및 제2 프로파일러(250)를 통해 복잡한 애플리케이션의 성능 병목 현상을 정확하고 효율적으로 식별하고 최적화할 수 있다.

【0060】 최적화 전략 생성부(270)는 특정 이벤트의 최적화 전략을 생성할 수 있다. 최적화 전략 생성부(270)는 특정 이벤트에서 병목을 일으키는 코드의 최적화 전략을 생성할 수 있다. 일 실시예에서, 최적화 전략 생성부(270)는 제2 프로파일러(250)의 분석 결과를 통해 애플리케이션의 성능 병목 지점과 해당 병목을 최적화했을 때의 성능 향상을 예측하여 성능 개선 최적화 전략을 수립할 수 있다.

【0061】 제어부(290)는 애플리케이션 성능 개선 장치(200)의 전체적인 동작을 제어하고, 블록 샘플 생성부(210), 제1 프로파일러(230), 제2 프로파일러(250) 및 최적화 전략 생성부(270) 간의 제어 흐름 또는 데이터 흐름을 관리할 수 있다.

【0063】 도 3은 본 발명에 따른 블록 샘플 기반의 애플리케이션 성능 개선 방법을 설명하는 순서도이다.

【0064】도 3을 참조하면, 애플리케이션 성능 개선 장치(200)는 블록 샘플 기반의 애플리케이션 성능 개선 방법을 수행하기 위한 일련의 동작 단계들을 처리할 수 있다. 구체적으로, 애플리케이션 성능 개선 장치(200)는 블록 샘플 생성부(210)를 통해 스레드가 CPU 실행 상태에 있든 또는 블로킹 상태에 있든 관계없이 발생되는 이벤트를 샘플링하여 블록 샘플들(blocked samples)을 생성할 수 있다(단계 S310). 애플리케이션 성능 개선 장치(200)는 제1 프로파일러(230)를 통해 블록 샘플들의 온-CPU 이벤트와 오프-CPU 이벤트를 통합적으로 분석하고 애플리케이션의 성능 병목을 식별할 수 있다(단계 S330).

【0065】또한, 애플리케이션 성능 개선 장치(200)는 제2 프로파일러(250)를 통해 온-CPU와 오프-CPU 이벤트 간의 상호 의존성을 분석하여 특정 이벤트의 최적화를 통한 성능 병목에 대한 성능 향상을 예측할 수 있다(단계 S350). 애플리케이션 성능 개선 장치(200)는 최적화 전략 생성부(270)를 통해 특정 이벤트의 최적화 전략을 생성할 수 있다(단계 S370).

【0067】도 4는 본 발명에 따른 블록 샘플 생성 과정의 일 실시예를 설명하는 도면이다.

【0068】도 4를 참조하면, 애플리케이션 성능 개선 장치(200)는 스레드가 CPU 실행 상태에 있든 또는 블로킹 상태에 있든 관계없이 발생되는 이벤트를 샘플링하여 블록 샘플들(blocked samples)을 생성하는 동작을 수행할 수 있다. 블록

샘플은 I/O 완료 대기, 동기화 대기(예: 뮤텍스, 조건 변수 등), CPU 스케줄링 대기 등과 같은 스레드의 블로킹 이벤트에 대한 정보를 캡처한다. 기존의 CPU 이벤트 샘플링은 스레드 지향적이며(예: 리눅스 perf 서브시스템의 task-clock), 스레드가 CPU에서 명령어를 실행할 때 이벤트 기반 샘플링은 주기적으로 스레드 컨텍스트(IP와 호출체인 등)의 샘플을 수집한다. 도 4에 보여진 바와 같이, 스레드가 블로킹되면 스레드가 웨이크-업(wake-up) 되서 실행을 재개할 때까지 샘플링이 중단된다. 기존의 CPU 이벤트 샘플링과 달리, 블록 샘플은 스레드가 블로킹된 동안 누락된 샘플을 보완하여 블로킹 기간 동안의 실행 컨텍스트를 제공한다.

【0069】각 블록 샘플은 오프-CPU 이벤트를 추적하기 위해 스레드가 블로킹되기 직전 실행된 명령어 주소(IP), 호출체인, 가중치 및 유형의 네가지 속성을 포함한다. IP는 실제로 CPU 스케줄러를 호출한 반환 주소(예: Linux의 schedule 또는 io_schedule) 이다. 호출체인은 스레드가 호출한 함수들의 스택 트레이스 이다. 블로킹 이벤트에는 동일한 속성(예: 동일한 IP, 호출체인 등)을 가진 여러 개의 블록 샘플을 포함할 수 있다. 애플리케이션 성능 개선 장치(200)는 가중치 필드에 반복 횟수를 인코딩하여 블록 샘플을 처리할 때 공간과 시간을 절약할 수 있다.

【0070】애플리케이션 성능 개선 장치(200)는 스레드가 스케쥴 아웃되면 오프-CPU 간격의 시작을 나타내는 타임스탬프(블로킹 타임스탬프)와 함께 스레드 블로킹의 서브클래스를 기록하고, 스레드가 웨이크-업되면 오프-CPU 간격의 끝을 나타내는 타임스탬프(웨이크-업 타임스탬프)를 기록한다. 스레드가 스케쥴 인되면 애플리케이션 성능 개선 장치(200)는 스케쥴 인 타임스탬프를 기록하고 도 4와 같이,

블로킹시간(Tblocked)과 CPU 스케줄링 대기 시간(Tsched)을 계산한다. CPU 경합으로 인해 실행 가능하지만 실행 대기열에 남아 있는 스레드의 경우, 오프-CPU 간격은 웨이크-업 타임스탬프가 없으므로 블록 샘플들의 스케줄링 서브클래스에 속한다. 스케쥴 인 함수에서 블로킹 간격이 시간상 하나 이상의 샘플링 지점과 겹치면 새 샘플이 생성된다. 이 샘플에는 IP, 호출체인, 가중치 및 유형의 속성이 포함된다.

【0071】 도 4의 경우, 두개의 오프-CPU 이벤트인 블로킹 이벤트의 전체 지속 시간(Tblocked)과 스케줄링 대기 시간(Tsched)은 두개의 샘플링 지점을 포함하고 있다. 따라서, 두개의 샘플링 지점 중 하나는 블로킹을 위한, 다른 하나는 스케줄링을 위한 두개의 블록 샘플이 수집된다. 만약, 오프-CPU 간격이 어떤 샘플링 지점과도 겹치지 않는다면, 블록 샘플은 수집되지 않는다. 이는 세개의 훅(hook) 지점이 타임스탬프 작업만 수행하므로, 오프-CPU 이벤트가 자주 발생하더라도 블록 샘플 수집의 오버헤드를 최소화할 수 있다.

【0073】 도 5는 본 발명에 따른 제1 프로파일링 결과의 일 실시예를 설명하는 도면이다.

【0074】 도 5를 참조하면, 애플리케이션 성능 개선 장치(200)는 블록 샘플들의 온-CPU 이벤트와 오프-CPU 이벤트를 통합적으로 분석하고 애플리케이션의 성능 병목을 식별하는 제1 프로파일링을 수행할 수 있다. 애플리케이션 성능 개선 장치(200)는 제1 프로파일러(230)를 통해 샘플링 기반 프로파일링을 사용하여 애플리케

이션을 프로파일링하고 샘플링 결과에 대한 통계를 제공할 수 있다. 제1 프로파일러(230)는 블록 샘플을 지원하기 위해 리눅스 perf 도구를 확장한 버전으로, 설명의 편의를 위해 이하에서는 bperf로 지칭하겠다. perf와 유사하게 bperf는 프로그램이 실행되는 동안 샘플링 기반 프로파일링을 언제든지 연결 및 분리할 수 있다. 기본적으로 블록 샘플을 처리하는 방식은 기존 온-CPU 샘플을 처리하는 방식과 큰 차이가 없다. 샘플은 IP와 호출체인을 기준으로 분류되고, 이 정보를 사용하여 오버헤드 부분, 함수 심볼 및 객체 파일과 같은 통계가 도 7의 (b)와 같이 보고된다. 도 7의 (b)는 bperf를 사용하여 도 7의 (a)의 케이스 1에 대한 프로파일링 결과를 보여준다. bperf는 온-CPU 이벤트와 오프-CPU 이벤트를 통합적으로 분석하여, 다양한 이벤트의 오버헤드를 보다 정확하게 이해하도록 할 수 있다.

【0075】 도 7의 (b)의 경우, 각 스레드(T1, T2)는 명령어(Command)와 관련된 공유 객체(Shared Object) 및 함수 심볼과 연결되어 있다. 스레드 1(T1)에서는 pread가 가장 큰 오버헤드를 발생시키며, 그 다음으로 pwrite가 뒤따르고, pthread_cond_wait가 세번째로 큰 오버헤드를 차지한다. 스레드 2(T2)에서는 compute_heavy가 해당 스레드의 실행 시간을 전부 차지한다. 즉, bperf는 온-CPU와 오프-CPU 이벤트를 통합적으로 분석하여 각 스레드에서 어떤 작업이 성능 병목을 일으키는지 명확하게 파악할 수 있다. bperf는 각 스레드에서 발생한 오버헤드, 함수 호출 및 관련 파일의 통계를 제공하여 애플리케이션 성능 분석에 도움을 줄 수 있다. bperf를 사용하면, 운영 체제 커널 내에서 블로킹 이벤트와 상호작용을 심층적으로 분석할 수 있고, 프로파일링 결과를 토대로 성능 최적화 지침을 제공할 수

있다.

【0077】 도 6은 본 발명에 따른 제2 프로파일링 수행의 일 실시예를 설명하는 도면이고, 도 7은 본 발명에 따른 제2 프로파일링 결과의 일 실시예를 설명하는 도면이다.

【0078】 도 6 및 7을 참조하면, 애플리케이션 성능 개선 장치(200)는 온-CPU와 오프-CPU 이벤트 간의 상호 의존성을 분석하여 특정 이벤트의 최적화를 통한 성능 병목에 대한 성능 향상을 예측하는 제2 프로파일링을 수행할 수 있다. 애플리케이션 성능 개선 장치(200)는 제2 프로파일러(250)를 통해 블록 샘플로부터 얻은 심볼 수준의 정보를 사용하여 온-CPU와 오프-CPU 이벤트 간의 상호 작용을 정확하게 식별하고 가상 속도 향상을 통해 성능 개선을 예측할 수 있다. 블록 샘플들은 블로킹 이벤트 즉, I/O 작업이 완료된 후에 처리되므로 블록 샘플들의 처리로 인해 블로킹 이벤트에 의존성이 존재하는 스레드에 지연 주입에 문제가 발생할 수 있다. 도 6에 도시한 바와 같이, 스레드 B의 블로킹 I/O 작업의 완료를 스레드 A가 대기하고 있고, 해당 I/O 작업이 가상 성능 향상 대상 작업인 경우, 의존성 처리(Dependency handling)는 대기중이었던 스레드 A가 웨이크 업될 때 처리된다. 이때, 오프-CPU 이벤트를 가상 성능 향상하기 위한 지연은 스레드 A가 웨이크 업 후 주입될 수 있어 이중 지연 문제가 발생할 수 있다. 애플리케이션 성능 개선 장치(200)의 제2 프로파일러(250)는 오프-CPU 이벤트를 가상 성능 향상하기 위한 지연을 오프-CPU 이벤트가 완료된 직후 처리하도록 한다. 즉, 도 6에서 스레드 B가

스레드 A를 웨이크 업 하기 전에 주입해야 되는 지연을 즉시 처리하도록 하여 의존성 처리에 의해 스레드 A가 올바르게 지연을 면제받을 수 있도록 한다.

【0079】 제2 프로파일러(250)의 프로파일링 결과로서, 도 7의 (a) 및 (b)는 도 5의 (a)에 있는 케이스 1 및 2에 대한 가상 속도 향상 결과이다. 도 7의 (a) 및 (b)는 각각 케이스 1에서 compute_heavy를 최적화하고, 케이스 2에서 I/O 작업 특히, pread을 최적화하여 실제 성능 향상을 얻을 수 있음을 보여준다. 또한, 가상 속도 향상 결과는 실제 성능 향상이 중요 경로가 이동하는 지점에 의해 제한됨을 나타낸다.

【0081】 도 8 및 도 9는 본 발명에 따른 블록 샘플 기반의 애플리케이션 성능 개선 방법에 관한 실험 결과를 설명하는 도면이다.

【0082】 도 8 및 도 9를 참조하면, 애플리케이션 성능 개선 장치(200)는 본 발명에 따른 블록 샘플 기반의 애플리케이션 성능 개선 방법을 수행할 수 있다. 구체적으로, 실험은 초당 최대 540K I/O 작업(IOPS)의 성능을 제공할 수 있는 Intel Xeon Gold 5218 CPU(2.30GHz, 16개의 물리적 코어), 375GB DDR4 DRAM, 플래시 기반 SSD(PM983)가 장착된 머신에서 수행되었다.

【0083】 도 8의 경우, 페이스북 오픈소스 워크로드인 Prefix Dist의 읽기 전용 실행에 대한 기존 방법(COZ) 및 본 발명에서 제안한 방법을 이용한 가상성능향상 기반의 인과관계 프로파일링 결과를 보여준다. 도 8의 (a)는 프로그램 속도 향

상과 라인 속도 향상 간의 관계를 보여주며, 기존 방법(점선)과 제안 방법(실선)을 사용하여 가상 속도 향상 결과를 도시하였다. 결과에서, 두가지 작업인 GetDataBlockFromCache와 ReadBlockContents가 병목 현상으로 식별되었다. 캐시 조회 작업은 작업자가 블록 캐시의 락을 두고 경쟁하기 때문에 실제 병목 지점이다. 도 8의 (a)에서 보듯이, 캐시 조회 작업이 최적화되면 제안 방법은 최대 60%의 속도 향상을 보여주고, 블록 읽기 I/O 작업이 최적화되면 최대 20%의 속도 향상을 보여주는 것을 확인할 수 있다. 두 작업 모두 오프-CPU 이벤트를 수반하지만, 기존 방법은 오프-CPU 이벤트를 프로파일링에 반영할 수 없기 때문에 두 작업에 대해 가상 속도 향상 결과를 전혀 보여주지 못하였다. 즉, 본 발명에서 제안한 인과관계 프로파일링 방법은 캐시 조회와 블록 읽기와 같은 오프-CPU 이벤트를 고려하여 성능 향상을 분석할 수 있다.

【0084】가상 속도 향상 결과를 검증하기 위해 두 작업에 대한 최적화를 수행하였다. 첫번째로, 플래시 기반 SSD를 최대 1,500K IOPS 성능을 제공하는 더 빠른 SSD로 교체하였다. 이 최적화는 SSD+로 표기한다. 도 8의 (b)에서 볼 수 있듯이, SSD+에서는 성능 향상이 없었다. 이는 락 경쟁이 주요 병목이기 때문이다. 두번째 최적화는 블록 캐시를 여러 샤드로 나누는 샤딩을 적용하였다. 이는 Shard-N(상기 N은 샤드의 수)로 표기한다. 도 8의 (b)에서 볼 수 있듯이, Shard-N은 성능이 개선되었다. 이는 샤드의 수가 많을수록 락 경쟁이 줄어들어 더 높은 처리량을 보여주는 것이다. 이 경향은 도 8의 (a)에서도 나타남을 알 수 있다.

【0085】도 9 (a)의 경우, CPU 코어 수를 하나로 제한했을 때 기존 방법(COZ)과 제안 방법(BCOZ)을 사용한 주요 연산 코드 라인의 프로파일링 결과로서, 기존 방법(COZ)은 성능 향상 가능성을 소폭 추정한 반면, 제안 방법(BCOZ)은 해당 코드 라인들이 최적화되면 잠재적인 성능 향상을 예측하였다. CPU 경쟁이 높은 상태(예: 32개의 스레드와 1개의 코어)에서는 스레드가 자주 스케쥴 아웃되기 때문에 오프-CPU 이벤트가 자주 발생한다. 이 경우, 제안 방법(BCOZ)은 스케줄링 서브클래스의 오프-CPU 이벤트가 제거되면 성능이 향상될 수 있는 최적화 기회를 예측할 수 있다.

【0086】도 9 (b)의 경우, 코어 수가 1개에서 32개로 증가함에 따라 스케줄링 서브클래스 수준의 가상 속도 향상에 대한 프로파일링 결과로서, 가장 높은 CPU 경쟁 상태(코어 1개만 사용할 때)에서 예측되는 프로그램 속도 향상은 최대치에 도달하였다. 그러나 코어 수가 증가할수록 CPU 경쟁이 줄어들어 속도 향상 효과도 줄어드는 것을 확인하였다.

【0087】도 9의 (c)의 경우, 코어 수가 X에서 32로 변경됨에 따라 즉, CPU 경쟁이 높은 상태에서 없는 상태로 전환될 때의 가상 및 실제 프로그램 속도 향상을 보여준다. 실제 속도 향상은 제안 방법이 예측한 속도 향상과 일치함을 보여주었다. 이 결과는 고도로 병렬화된 작업 부하에서 오프-CPU 이벤트를 정확하게 프로파일링하는 것이 중요하며, 제안 방법은 블록 샘플을 활용하여 유용한 프로파일링 결과를 제공함을 확인해준다.

【0088】본 발명에서 제안한 블록 샘플 기반의 프로파일링 기법은 온-CPU 이벤트와 오프-CPU 이벤트를 통합하여 애플리케이션의 병목 현상을 동일한 차원에서 식별할 수 있다. 또한, 본 발명은 블록 샘플을 활용하여 이벤트 실행 시간을 기반으로 애플리케이션 병목을 식별하는 프로파일러 및 오프-CPU 이벤트에 대한 가상 속도 향상을 제공하는 인과관계 프로파일러를 사용하여 기존 프로파일링에서 식별되지 않았던 I/O 및 동기화 작업과 관련된 병목 현상을 식별할 수 있으며, 이러한 작업들을 가상 속도 향상을 통해 최적화할 수 있다.

【0089】

【0090】상기에서는 본 발명의 바람직한 실시예를 참조하여 설명하였지만, 해당 기술 분야의 숙련된 당업자는 하기의 특허 청구의 범위에 기재된 본 발명의 사상 및 영역으로부터 벗어나지 않는 범위 내에서 본 발명을 다양하게 수정 및 변경시킬 수 있음을 이해할 수 있을 것이다.

【부호의 설명】

【0092】100: 컴퓨터 시스템 200: 애플리케이션 성능 개선 장치

210: 블록 샘플 생성부 230: 제1 프로파일러

250: 제2 프로파일러 270: 최적화 전략 생성부

290: 제어부

【청구범위】

【청구항 1】

스레드가 CPU 실행 상태에 있든 또는 블로킹 상태에 있든 관계없이 발생되는 이벤트를 샘플링하여 블록 샘플들(blocked samples)을 생성하는 블록 샘플 생성부;

상기 블록 샘플들의 온-CPU 이벤트와 오프-CPU 이벤트를 통합적으로 분석하고 애플리케이션의 성능 병목을 식별하는 제1 프로파일러;

상기 온-CPU와 상기 오프-CPU 이벤트 간의 상호 의존성을 분석하여 특정 이벤트의 가상 최적화를 통한 상기 성능 병목에 대한 성능 향상을 예측하는 제2 프로파일러; 및

상기 특정 이벤트의 최적화 전략을 생성하는 최적화 전략 생성부를 포함하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 2】

제1항에 있어서, 상기 블록 샘플 생성부는

상기 블로킹 상태인 경우로서 I/O 대기, 동기화 대기 또는 스케줄링 대기로 인해 상기 스레드가 CPU에서 실행되지 못하는 상태에서도 상기 이벤트를 샘플링하는 것을 특징으로 하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 3】

제2항에 있어서, 상기 블록 샘플 생성부는

상기 스레드가 스케쥴 아웃 및 스케쥴 인 시점에서 타이머를 통해 상기 스레드의 상태를 확인하여 상기 블록 샘플들을 샘플링하여 블로킹 이벤트를 기록하는 것을 특징으로 하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 4】

제3항에 있어서, 상기 블록 샘플 생성부는

상기 블로킹 이벤트의 중복을 줄이기 위해 반복횟수를 나타내는 가중치를 기록하여 동일한 속성의 블로킹 이벤트를 하나의 블로킹 이벤트로 묶어 처리하는 것을 특징으로 하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 5】

제3항에 있어서, 상기 블록 샘플 생성부는

상기 스레드가 스케쥴 아웃될 때 상기 스레드가 블로킹되기 직전 실행된 명령어 주소(IP), 상기 스레드가 호출한 함수들의 스택 트레이스를 나타내는 호출체인, 블로킹 이벤트의 유형을 기록하는 타입 및 상기 스레드가 블로킹된 시점을 나타내는 블로킹 타임스탬프를 상기 블로킹 이벤트에 저장하고,

상기 블로킹 이벤트의 유형은 상기 I/O 대기, 상기 동기화 대기 또는 상기

스케줄링 대기 중 하나에 해당하는 것을 특징으로 하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 6】

제5항에 있어서, 상기 블록 샘플 생성부는

상기 스레드가 웨이크-업될 때 상기 스레드의 대기 시간과 대기 이유를 추적하기 위해 웨이크-업 타임스탬프를 상기 블로킹 이벤트에 저장하는 것을 특징으로 하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 7】

제6항에 있어서, 상기 블록 샘플 생성부는

상기 스레드가 스케쥴 인될 때 블로킹 이벤트의 전체 지속 시간(Tblocked)과 스케줄링 대기 시간(Tsched)을 계산하기 위해 스케쥴-인 타임스탬프를 상기 블로킹 이벤트에 저장하는 것을 특징으로 하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 8】

제1항에 있어서, 상기 제1 프로파일러는

블로킹 이벤트에서 상기 온-CPU 이벤트와 상기 오프-CPU 이벤트를 결정하고

각 이벤트가 애플리케이션 실행 중 차지하는 오버헤드 정보를 분석하는 것을 특징으로 하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 9】

제8항에 있어서, 상기 제1 프로파일러는

상기 오버헤드 정보의 분석을 통해 I/O 대기, 동기화 대기 또는 스케줄링 대기에 따른 서브클래스로 분류하고 상기 서브클래스의 성능 병목을 결정하여 상기 애플리케이션의 성능 병목을 식별하는 것을 특징으로 하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 10】

제1항에 있어서, 상기 제2 프로파일러는

상기 온-CPU와 상기 오프-CPU 이벤트 간의 인과관계 분석을 수행하고 가상 속도 향상 (Virtual Speedup) 기법을 통해 특정 이벤트를 가상으로 가속화하여 상기 성능 향상을 예측하는 것을 특징으로 하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 11】

제1항에 있어서, 상기 최적화 전략 생성부는

상기 특정 이벤트에서 병목을 일으키는 코드의 최적화 전략을 생성하는 것을 특징으로 하는 블록 샘플 기반의 애플리케이션 성능 개선 장치.

【청구항 12】

블록 샘플 기반의 애플리케이션 성능 개선 장치에서 수행되는 블록 샘플 기반의 애플리케이션 성능 개선 방법에 있어서,

스레드가 CPU 실행 상태에 있든 또는 블로킹 상태에 있든 관계없이 발생되는 이벤트를 샘플링하여 블록 샘플들(blocked samples)을 생성하는 블록 샘플 생성단계;

상기 블록 샘플들의 온-CPU 이벤트와 오프-CPU 이벤트를 통합적으로 분석하고 애플리케이션의 성능 병목을 식별하는 제1 프로파일링 단계;

상기 온-CPU와 상기 오프-CPU 이벤트 간의 상호 의존성을 분석하여 특정 이벤트의 가상 최적화를 통한 상기 성능 병목에 대한 성능 향상을 예측하는 제2 프로파일링 단계; 및

상기 특정 이벤트의 최적화 전략을 생성하는 최적화 전략 생성단계를 포함하는 블록 샘플 기반의 애플리케이션 성능 개선 방법.

【요약서】

【요약】

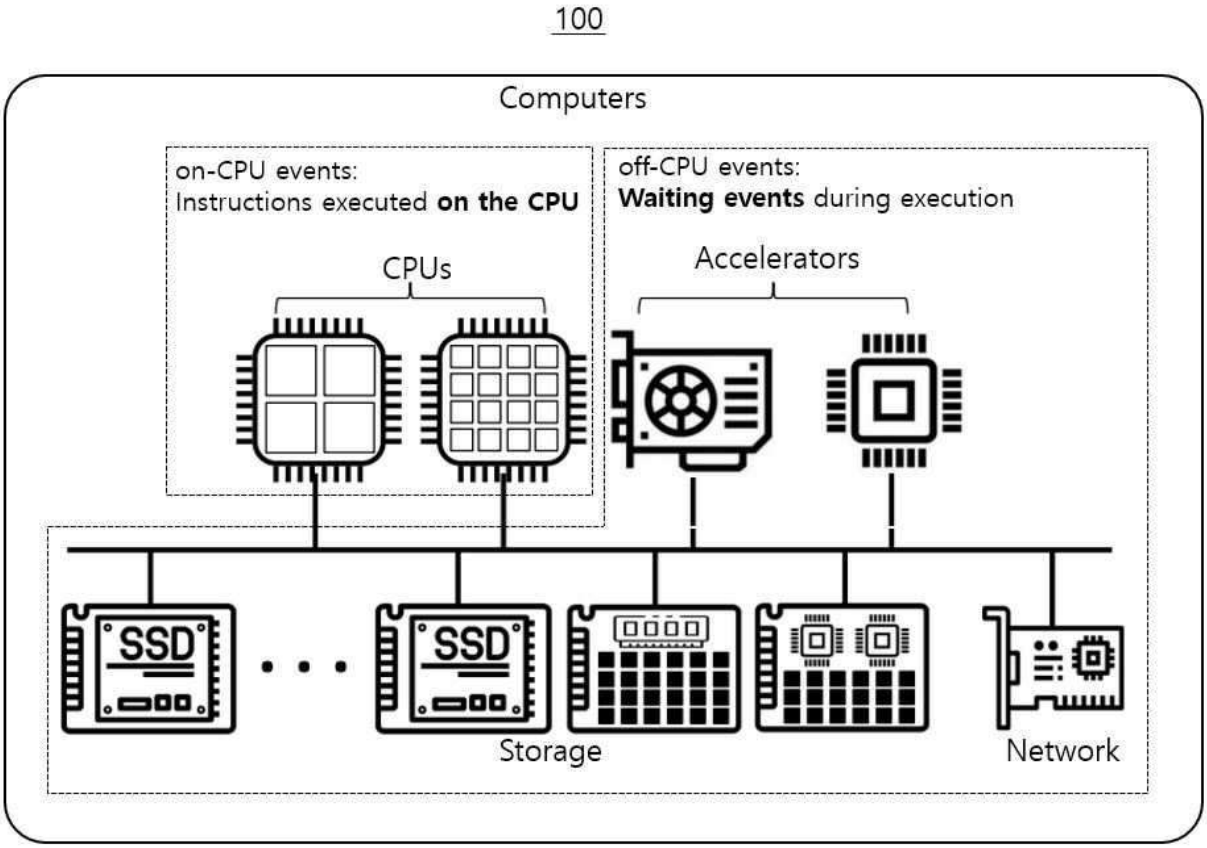
본 발명은 블록 샘플 기반의 애플리케이션 성능 개선 장치 및 방법에 관한 것으로, 상기 장치는 스레드가 CPU 실행 상태에 있든 또는 블로킹 상태에 있든 관계없이 발생되는 이벤트를 샘플링하여 블록 샘플들(blocked samples)을 생성하는 블록 샘플 생성부; 상기 블록 샘플들의 온-CPU 이벤트와 오프-CPU 이벤트를 통합적으로 분석하고 애플리케이션의 성능 병목을 식별하는 제1 프로파일러; 상기 온-CPU와 상기 오프-CPU 이벤트 간의 상호 의존성을 분석하여 특정 이벤트의 가상 최적화를 통한 상기 성능 병목에 대한 성능 향상을 예측하는 제2 프로파일러; 및 상기 특정 이벤트의 최적화 전략을 생성하는 최적화 전략 생성부를 포함한다.

【대표도】

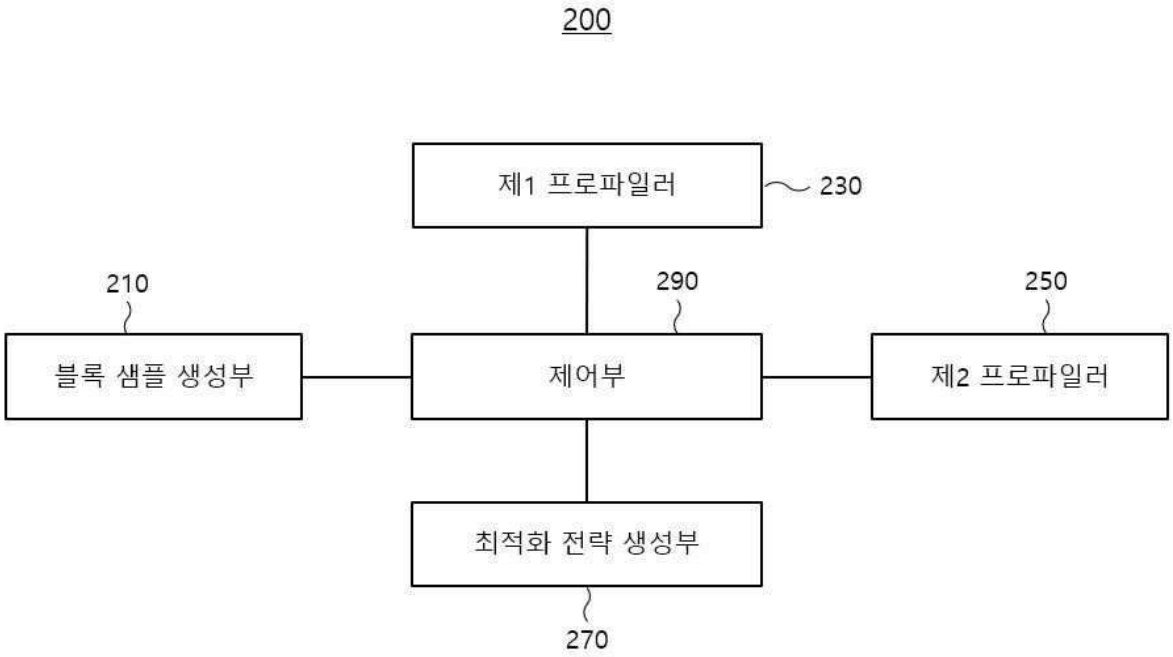
도 2

【도면】

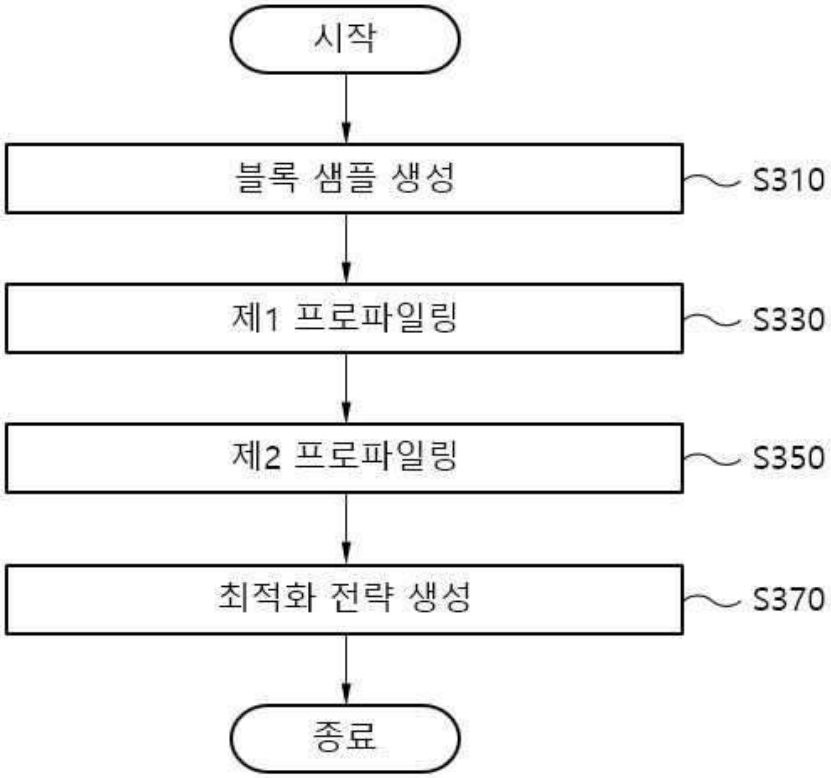
【도 1】


100
Computers
on-CPU events:
Instructions executed **on the CPU**
off-CPU events:
Waiting events during execution
CPUs
Accelerators
SSD
SSD
Storage
Network

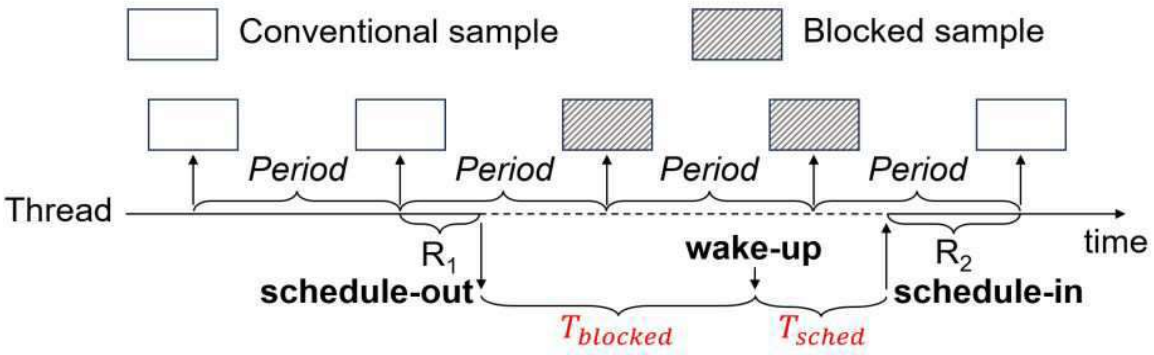
【도 2】


200
제1 프로파일러
230
210
290
250
블록 샘플 생성부
제어부
제2 프로파일러
최적화 전략 생성부
270

【도 3】

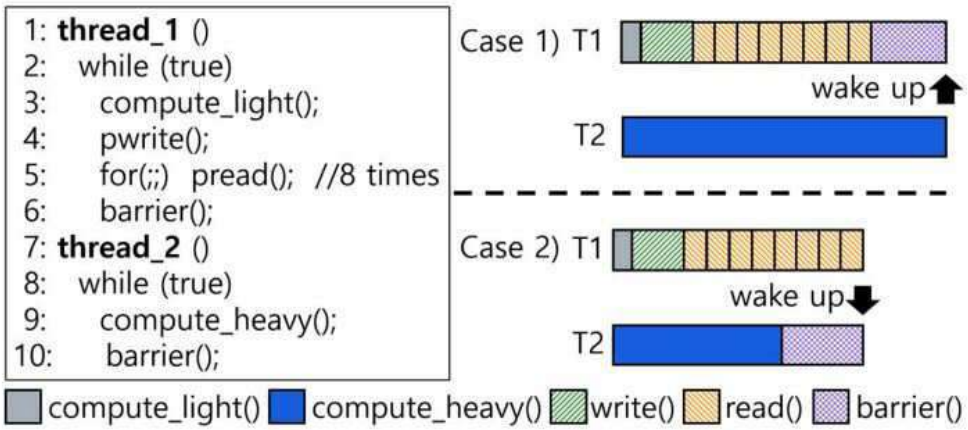

시작
블록 샘플 생성
S310
제1 프로파일링
S330
제2 프로파일링
S350
최적화 전략 생성
S370
종료

【도 4】


Conventional sample
Blocked sample
Period
Period
Period
Period
Thread
time
R1
R2
wake-up
schedule-out
schedule-in
Tblocked
Tsched

【도 5】

(a)



(b)

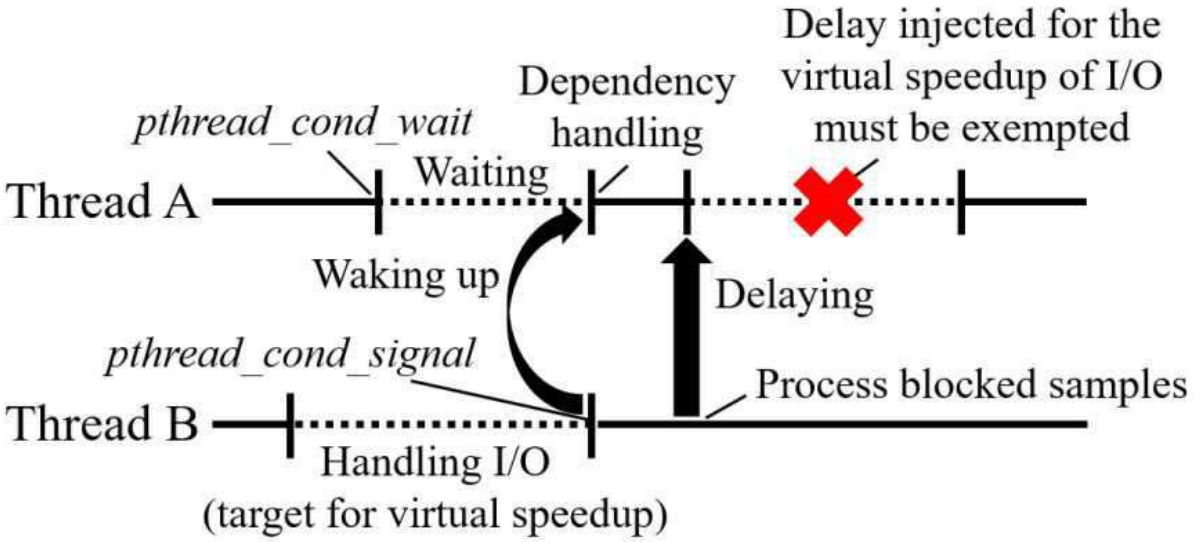
T1

# Overhead	Command	Shared Object	Symbol
64.54%	a.out	libpthread	[I] __libc_pread64
14.94%	a.out	[kernel.kallsyms]	[I] __libc_pwrite64
5.50%	a.out	a.out	[L] pthread_cond_wait
2.59%	a.out	a.out	[.] compute_light
...			

T2

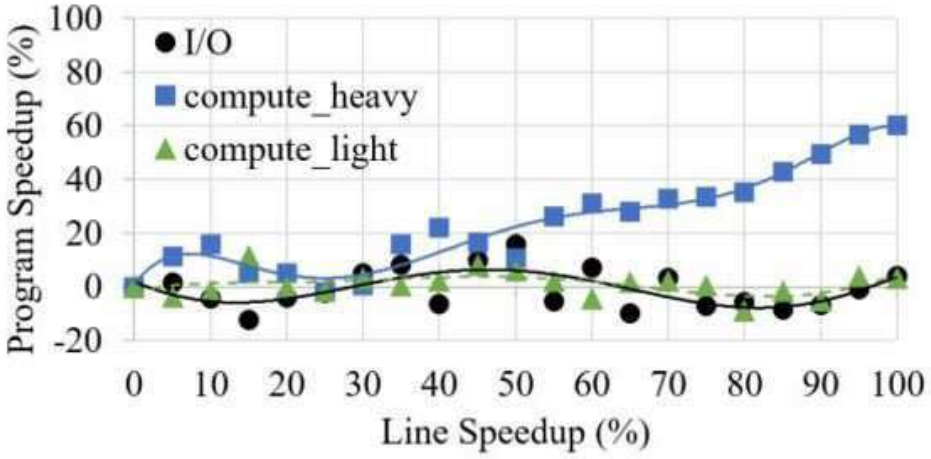
# Overhead	Command	Shared Object	Symbol
99.97%	a.out	a.out	[.] compute_heavy
0.02%	a.out	libpthread	[L] pthread_cond_wait
...			

【도 6】

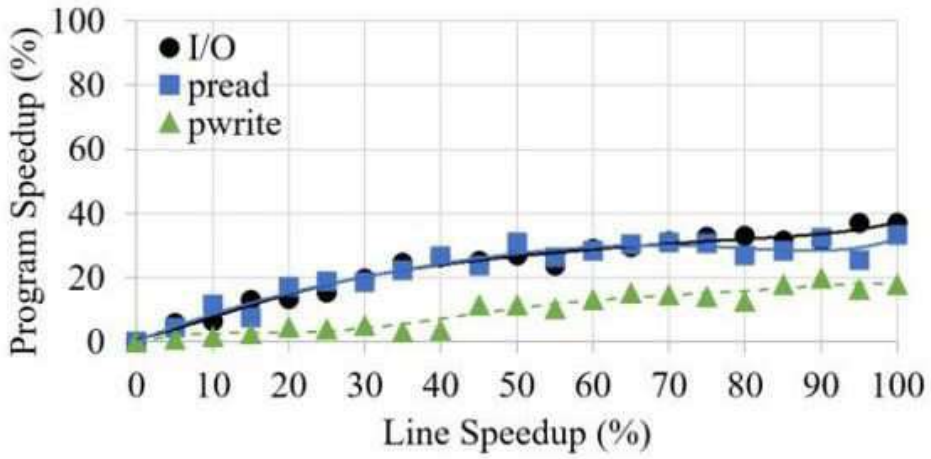

Delay injected for the
virtual speedup of I/O
must be exempted
Dependency
handling
pthread_cond_wait
Waiting
Thread A
Waking up
Delaying
pthread_cond_signal
Process blocked samples
Thread B
Handling I/O
(target for virtual speedup)

【도 7】

(a)

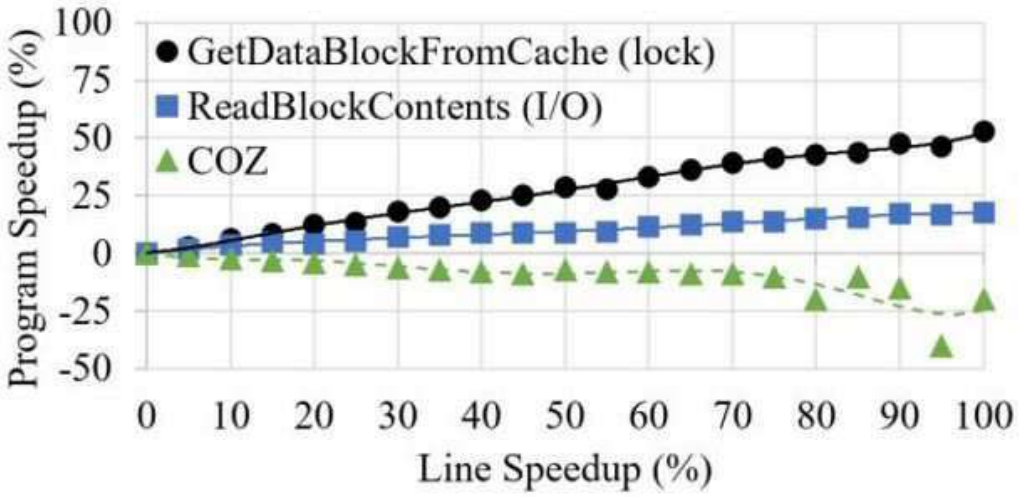

Program Speedup (%)
100
80
60
40
20
0
-20
I/O
compute_heavy
compute_light
0 10 20 30 40 50 60 70 80 90 100
Line Speedup (%)

(b)


Program Speedup (%)
100
80
60
40
20
0
I/O
pread
pwrite
0 10 20 30 40 50 60 70 80 90 100
Line Speedup (%)

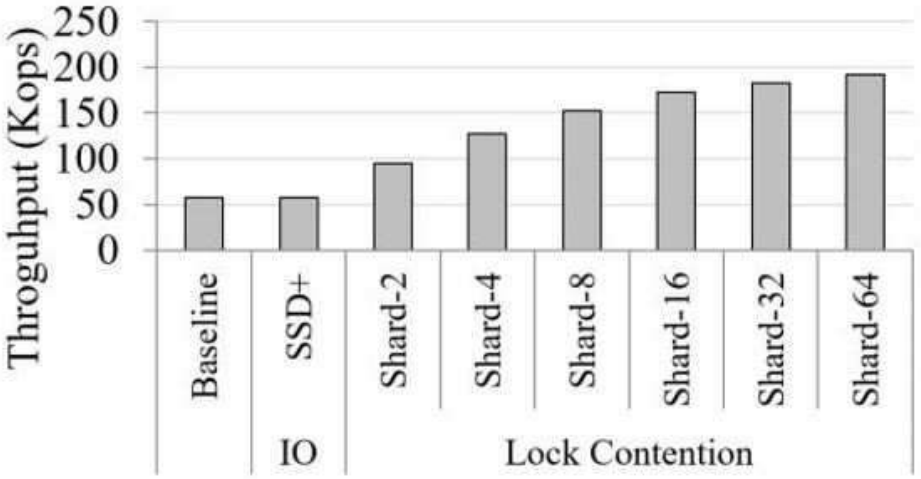
【도 8】

(a)



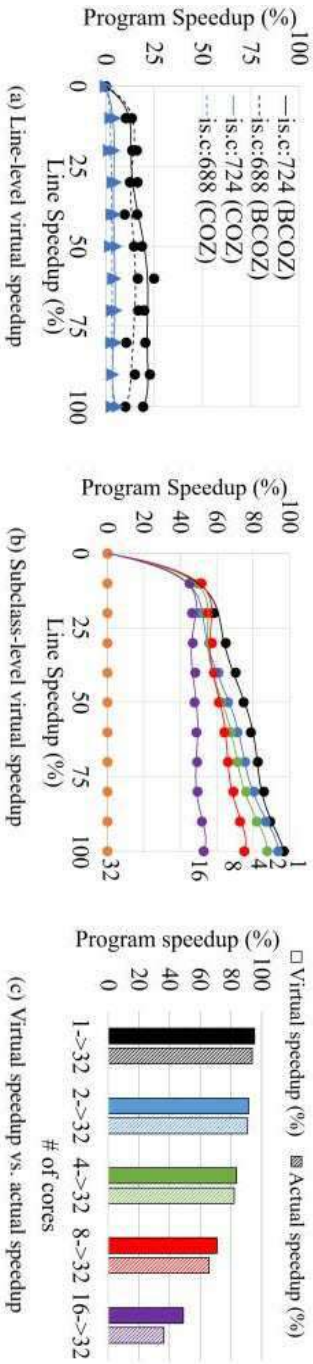
(a) Causality analysis

(b)



(b) Optimization results

【도 9】


Program Speedup (%)
0
25
50
75
100
Line Speedup (%)
is.c:724 (BCOZ)
is.c:688 (BCOZ)
is.c:724 (COZ)
is.c:688 (COZ)
(a) Line-level virtual speedup
Program Speedup (%)
Line Speedup (%)
1
2
4
8
16
32
(b) Subclass-level virtual speedup
Program speedup (%)
Virtual speedup (%)
Actual speedup (%)
1->32
2->32
4->32
8->32
16->32
# of cores
(c) Virtual speedup vs. actual speedup