Smart Media Journal / Vol.15, No.7 / ISSN:2287-1322
https://dx.doi.org/10.30693/SMJ.2026.15.7.36

# CASA: 정적 접근 패턴 기반 캐시 분석 프레임워크의 설계 및 구현

## (CASA: Design and Implementation of a Cache Analysis Framework Based on Static Access Patterns)

**박준용*, 신지우**, 맹훈규***, 전현수**, 박성민****, 허금숙****, 강지호****, 정진만*****

(Junyong Park, Jiwoo Shin, Hunkyu Maeng, Hyeonsoo Jeon, Seongmin Park, Geumsook Heo, Jiho Kang, Jinman Jung)

**요 약**

캐시 동작은 프로그램 성능에 큰 영향을 미치며, 실행 이전에 코드 구조를 기반으로 캐시 사용 양상을 분석하는 것은 성능 병목 파악과 최적화 방향 도출에 중요하다. 본 논문은 LLVM IR에서 추출한 정적 접근 패턴을 기반으로, 프로그램 실행 없이 캐시 동작을 분석하는 CASA(Cache Analysis Framework Based on Static Access Patterns)의 설계와 구현을 제안한다. CASA는 정적 접근 패턴을 메모리 접근 시퀀스로 변환하고, YAML 설정으로 구성된 캐시 모델에서 hit/miss를 분석한다. 또한 완전 연관 비교 캐시를 이용하여 캐시 미스를 분류하고, 결과를 객체 및 Load/Store 연산 단위로 집계한다. PolyBench 벤치마크를 대상으로 Cachegrind와 비교한 결과, CASA는 L1 및 LL 캐시 미스 수에서 각각 0.7%와 0.1%의 평균 절대 백분율 오차를 보였으며, 기존 동적 프로파일링 도구와 일관된 분석 결과를 산출함을 확인하였다.

■ 중심어 : 캐시 분석 ; 정적 분석 ; 접근 패턴 ; 캐시 미스 분류 ; LLVM

**Abstract**

Cache behavior has a significant impact on program performance, and analyzing cache usage before execution is important for identifying performance bottlenecks and deriving optimization directions. This paper presents the design and implementation of CASA (Cache Analysis Framework Based on Static Access Patterns), which analyzes cache behavior without executing the target program using static access patterns extracted from LLVM IR. CASA transforms static access patterns into a memory access sequence and analyzes hit/miss behavior on a YAML-configured cache model. It also classifies cache misses using a fully associative comparison cache and aggregates the results by memory object and Load/Store operation. In an evaluation with PolyBench benchmarks, CASA showed mean absolute percentage errors of 0.7% and 0.1% for L1 and LL cache miss counts compared with Cachegrind, demonstrating results consistent with an existing dynamic profiling tool.

■ keywords : Cache Analysis ; Static Analysis ; Access Pattern ; Cache Miss Classification ; LLVM

## Ⅰ. 서 론

캐시 계층은 현대 프로세서에서 연산 장치와 주 메모리 사이의 속도 격차를 완화하는 핵심 구성 요소이다. 메모리 접근이 캐시에 적중하는 비

* 준회원, 인하대학교 전기컴퓨터공학과
** 정회원, 인하대학교 전기컴퓨터공학과
*** 준회원, 인하대학교 컴퓨터공학과
**** 정회원, LIG Defense & Aerospace
***** 종신회원, 인하대학교 컴퓨터공학과
본 연구는 LIG Defense & Aerospace의 『비행소프트웨어 시뮬레이터 개발』(Y26-W014) 산학협력과제 지원으로 연구되었습니다.
접수일자 : 2026년 07월 10일 게재확정일 : 2026년 07월 27일
수정일자 : 2026년 07월 24일
교신저자 : 정진만 e-mail : jmjung@inha.ac.kr

율은 프로그램 성능에 직접적인 영향을 미치며 [1], 프로그램의 캐시 사용 양상을 파악하는 것은 성능 분석 및 최적화의 중요한 출발점이 된다.

캐시 동작을 분석하는 전통적인 방법은 프로그램 실행 중 발생하는 메모리 접근 트레이스를 수집하고, 이를 바탕으로 캐시 동작을 분석하는 동적 프로파일링이다. 대표적인 도구로는 Cachegrind[2]와 gem5[3]가 있다. 그러나 동적 프로파일링은 런타임에 발생하는 메모리 접근 정보를 수집하고 처리해야 하므로, 대규모 프로그램에서는 상당한 시간적·공간적 오버헤드를 유발할 수 있다. 이러한 한계는 프로그램 실행 전에 코드 구조를 기반으로 캐시 동작을 예측하는 정적 분석의 필요성을 제기한다.

정적 캐시 분석의 필요성은 항공우주와 같은 안전 필수(Safety-Critical) 실시간 환경에서 더욱 두드러진다. 실시간 시스템에서는 평균 성능뿐만 아니라 실행 시간의 예측 가능성(Timing Predictability)이 중요하며, 캐시는 실행 시간 변동을 유발하는 주요 요인 중 하나이다. 가능한 모든 입력 조합에 대해 프로그램을 사전에 실행하는 것이 현실적으로 어렵기 때문에, 코드 구조로부터 캐시 동작을 분석하는 정적 접근이 필요하다. 또한 NASA Power of 10[4]과 같은 안전 필수 코딩 규칙은 고정된 루프 상한, 재귀 제한, 초기화 이후 동적 메모리 할당 제한 등을 요구하므로, 정적 접근 시퀀스 전개를 위한 본 논문의 분석 가정과 잘 부합한다. 본 논문은 이러한 조건을 만족하는 프로그램을 기본 분석 대상으로 가정하며, 구체적인 지원 범위와 제한 사항은 3장에서 기술한다.

본 논문은 LLVM IR로부터 추출한 정적 접근 패턴을 입력으로 받아 설정된 캐시 모델의 동작을 분석하는 도구인 CASA(Cache Analysis Framework Based on Static Access Patterns)의 설계 및 구현을 제안한다. 입력으로 사용되는 접근 패턴은 저자가 개발한 Access-Pattern-Extractor(APE)가 LLVM IR로부터 메모리 접근 정보, 반복 구조, 함수 호출 구조, 객체 정보를 추출하여 생성한 중간 표현이다.

CASA는 APE로부터 전달받은 접근 패턴에 대해 반복 구조 전개와 함수 인라이닝을 수행하여 선형 접근 시퀀스를 생성하고, 이를 바탕으로 캐시 계층에서의 hit/miss를 분석한다. 또한 완전 연관(Fully-Associative) 비교 캐시를 함께 운용하여 단일 분석 과정에서 캐시 미스를 Compulsory, Capacity, Conflict miss로 분류한다. 분석 결과는 메모리 객체 및 Load/Store 연산 단위로 집계되며, 이를 통해 프로그램의 캐시 병목 원인을 파악할 수 있다. 아울러 YAML 설정 파일을 통해 L1/LLC 및 하위 메모리 계층, 캐시 크기, 라인 크기, 연관도, 교체 정책, 쓰기 정책 등을 구성할 수 있도록 하였다.

본 논문의 핵심 기여는 캐시 미스 분류와 완전 연관 비교 캐시와 같은 기존 개별 기법을 LLVM IR 기반 정적 접근 패턴 분석과 결합하여, 실행 전 캐시 동작 분석을 위한 단일 파이프라인으로 구현하고 검증한 데 있다. CASA는 정적 접근 패턴을 메모리 접근 시퀀스로 전개하고, 설정된 캐시 모델에서 캐시 hit/miss를 산출하며, 완전 연관 비교 캐시를 이용해 캐시 미스를 Compulsory miss, Capacity miss, Conflict miss로 분류한다. 또한 분석 결과를 객체 및 Load/Store 연산 단위로 집계하여, 캐시 비효율의 원인을 데이터 객체와 접근 유형 관점에서 분석할 수 있도록 한다. 정확성 검증은 PolyBench[5] 벤치마크를 대상으로 Cachegrind와 CASA의 L1 및 LL 캐시 미스 수를 비교하여 수행하였으며, 이를 통해 CASA가 기존 동적 프로파일링 도구와 일관된 분석 결과를 산출함을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 동적·정적 캐시 분석과 캐시 미스 분류에 관한 관련 연구를 정리한다. 3장에서는 CASA의 설계 목표, 분석 범위, 전체 분석 파이프라인을 설명한다. 4장에서는 Cachegrind 비교 기반 정확성

검증과 PolyBench 기반 평가 결과를 제시하며, 마지막 5장에서 결론과 향후 과제를 논한다.

## Ⅱ. 관련 연구

### 1. 동적 캐시 시뮬레이션

캐시 동작을 분석하는 전통적인 방법은 프로그램 실행 중 발생하는 메모리 접근을 추적하고, 이를 캐시 시뮬레이터에 입력하는 트레이스 기반 동적 시뮬레이션이다. Valgrind의 Cachegrind는 바이너리 계측을 통해 실행 중 발생하는 메모리 접근을 포착하고, 캐시 적중 및 미스 정보를 집계하는 대표적인 profiling 도구이다. gem5[3]는 프로세서 마이크로아키텍처와 메모리 계층을 함께 모델링할 수 있는 모듈형 시스템 시뮬레이터로, 캐시와 메모리 시스템을 포함한 상세한 하드웨어 수준 분석에 활용된다.

이러한 동적 분석 도구들은 실제 실행 경로에서 발생한 메모리 접근을 기반으로 구체적인 캐시 동작 정보를 제공한다는 강점이 있다. 그러나 분석 대상을 실행 가능한 형태로 준비해야 하며, 분석 결과가 입력값과 실행 경로에 종속된다는 한계를 지닌다. 또한 전체 메모리 접근 트레이스를 수집하고 처리하는 과정에서 상당한 시간적·공간적 오버헤드가 발생할 수 있다. 이와 달리 본 연구는 프로그램 실행 중 트레이스를 수집하지 않고, 컴파일 타임에 추출된 정적 접근 패턴을 입력으로 사용하여 캐시 동작을 분석한다.

### 2. 재사용 거리 기반 캐시 분석

프로그램의 캐시 동작을 직접 실행하지 않거나 제한된 실행 정보만으로 예측하는 방법으로 재사용 거리(Reuse Distance) 기반 분석이 널리 연구되어 왔다[6]. 재사용 거리는 동일한 메모리 위치에 대한 두 연속 접근 사이에서 접근된 고유 메모리 위치의 수를 의미하며, 데이터 지역성을 정량화하는 대표적인 지표로 사용된다. 이러한 접근은 재사용 거리 분포를 기반으로 캐시 적중률이나 미스율을 추정하거나, 메모리 계층 최적화의 근거를 제공하는 데 활용된다.

재사용 거리 기반 분석은 캐시 동작을 전체 접근 트레이스의 요약된 지역성 정보로 표현한다는 점에서 효율적이다. 그러나 분석 결과가 주로 분포 또는 모델 수준의 미스율 추정으로 제시되는 경우가 많아, 개별 메모리 접근이 어떤 캐시 계층에서 hit/miss를 발생시키는지, 그리고 그 미스가 객체 단위로 어떻게 누적되는지를 직접적으로 제공하기에는 한계가 있다. 반면 CASA는 LLVM IR로부터 추출한 정적 접근 패턴을 선형 접근 시퀀스로 전개하고, 이를 다단계 캐시 모델에 직접 입력하여 접근 단위의 hit/miss와 미스 유형을 산출한다. 이를 통해 단순한 미스율 추정이 아니라 객체 단위의 캐시 병목 원인 분석을 지원한다.

### 3. 실시간 시스템에서의 캐시 분석

안전 필수 실시간 시스템에서는 최악 실행 시간(WCET)을 보수적으로 추정하기 위해 캐시와 같은 하드웨어 효과를 정적으로 분석하는 연구가 활발히 이루어져 왔다[7]. 대표적으로 Ferdinand와 Wilhelm은 추상 해석(Abstract Interpretation)을 기반으로 메모리 접근의 캐시 동작을 Always-Hit, Always-Miss, Not-Classified와 같은 범주로 분류함으로써, 실행 전에 캐시 동작을 예측하는 방법론을 제시하였다[8].

이러한 WCET 지향 캐시 분석은 모든 가능한 실행 경로에 대해 안전한 상한을 제공하는 것을 목표로 하므로, 분석 결과가 의도적으로 보수적일 수 있다. 반면 CASA는 엄격한 WCET 상한을 도출하기 위한 보수적 분석기가 아니라, 정적으로 전개 가능한 접근 패턴에 대해 캐시 동작을 분석하고 객체 단위의 미스 원인을 분석하기 위한 진단형 도구로 위치한다. 따라서 본 연구는 실시간 시스템에서 요구되는 정적 분석의 필요성을 공유하지만, 분석 목적은 시간 상한 보장보다 캐시 병목 구조의 이해와 구현 수준의 분석

결과 제공에 있다.

### 4. 캐시 미스 분류와 캐시 분석

캐시 미스는 일반적으로 최초 접근으로 인해 발생하는 Compulsory miss, 제한된 캐시 용량으로 인해 발생하는 Capacity miss, 제한된 연관도 또는 매핑 충돌로 인해 발생하는 Conflict miss로 구분된다. 이러한 캐시 미스 분류는 단순한 전체 미스율을 넘어, 캐시 성능 저하의 원인을 구체적으로 해석하는 데 유용하다.

Hill과 Smith는 CPU 캐시의 연관도 변화가 성능에 미치는 영향을 평가하기 위해 실제 설정 캐시와 완전 연관 캐시 모델을 비교하는 시뮬레이션 방법을 제시하였다[9]. 이러한 비교 방식은 제한된 캐시 용량에 의한 영향과 제한된 연관도에 의한 영향을 구분하는 기준을 제공한다.

CASA는 이러한 비교 기반 캐시 미스 분류 방식을 LLVM IR 기반 정적 접근 패턴 분석 파이프라인에 통합한다. 구체적으로 사용자 설정에 따라 구성된 캐시 모델과 동일한 용량의 완전 연관 비교 캐시를 병렬로 운용하여, 단일 분석 과정에서 캐시 미스를 Compulsory miss, Capacity miss, Conflict miss로 분류한다. 또한 분류 결과를 메모리 객체 및 Load/Store 연산 단위로 집계함으로써, 단순한 전체 미스율이 아니라 캐시 비효율이 발생한 원인을 데이터 객체와 접근 유형 관점에서 분석할 수 있도록 한다.

따라서 본 연구는 동적 캐시 시뮬레이션의 구체성, 정적 분석의 실행 전 분석 가능성, 그리고 캐시 미스 분류의 원인 분석 능력을 결합하여, 프로그램 실행 없이 캐시 병목 구조를 분석할 수 있는 구현 중심의 정적 캐시 분석 프레임워크를 제안한다.

## Ⅲ. CASA 분석 파이프라인

CASA는 LLVM IR 수준에서 추출된 정적 메모리 접근 패턴을 입력으로 받아, 대상 프로그램을 직접 실행하지 않고 캐시 동작을 분석하는 정적 캐시 분석 프레임워크이다. 본 장에서는 CASA의 설계 목표와 분석 범위를 설명하고, 정적 접근 패턴이 캐시 분석 결과로 변환되는 전체 파이프라인을 기술한다.

### 1. 설계 목표와 분석 범위

CASA는 프로그램 실행 없이 캐시 동작을 분석하기 위한 정적 분석 파이프라인이다. LLVM IR에서 추출된 반복 구조, 함수 호출 구조, 메모리 접근 정보, 객체 정보를 입력으로 사용하며, 동일한 입력에 대해 재현 가능한 분석 결과를 제공한다.

CASA는 캐시 구조를 코드에 고정하지 않고 YAML 설정 파일을 통해 지정할 수 있도록 설계하였다. 이를 통해 캐시 크기, 라인 크기, 연관도, 교체 정책, 쓰기 정책, 계층별 지연 시간 등을 설정 파일 수준에서 조정할 수 있다.

다만 CASA의 정적 분석은 입력 프로그램이 일정한 조건을 만족할 때 가능하다. 본 도구는 반복 횟수가 컴파일 시점에 결정되는 고정 루프, 재귀가 없는 함수 구조, 정적으로 결정되는 객체 크기와 배치, 그리고 루프 유도 변수와 상수 기반의 배열 인덱스를 기본 분석 대상으로 한다. 반면, 가변 루프, 재귀 호출, 동적 메모리 할당, 비정형 포인터 접근은 분석에 포함하지 않는다.

### 2. 전체 파이프라인

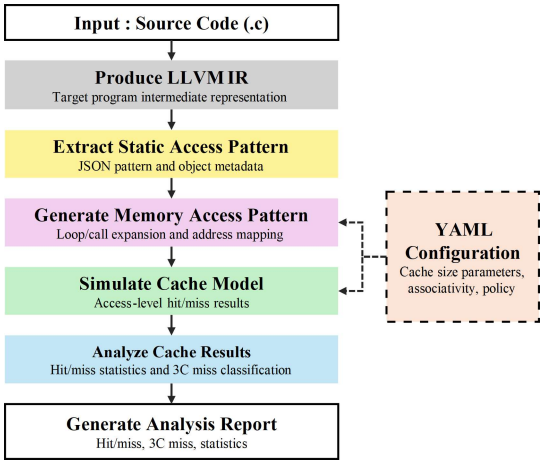


그림 1. CASA 전체 시스템 아키텍처

그림 1은 CASA의 전체 분석 파이프라인을 나타낸다. 소스 코드로부터 생성된 LLVM IR을 기반으로 정적 접근 패턴과 객체 메타데이터를 추출하고, 이를 메모리 접근 시퀀스로 변환한다. 변환된 시퀀스는 YAML 설정으로 구성된 캐시 모델에서 시뮬레이션되며, CASA는 접근별 hit/miss 결과와 캐시 미스 유형별 통계를 최종 분석 결과로 생성한다.

이러한 단방향 구조는 입력 해석, 주소 변환, 캐시 분석, 결과 분석의 책임을 분리하여 각 단계의 확장과 검증을 용이하게 한다. 또한 캐시 설정 변경이나 입력 프론트엔드 변경이 전체 구조에 미치는 영향을 줄이고, 동일한 입력에서 다양한 캐시 구성을 비교할 수 있도록 한다.

이때 분석용 주소는 실제 실행 시점의 가상 주소나 물리 주소를 의미하지 않는다. 이는 분석을 위해 객체 배치와 오프셋을 바탕으로 구성한 논리적 주소이다. 따라서 CASA의 분석 결과는 실제 하드웨어 주소 배치에 대한 관측값이 아니라, 주어진 객체 배치와 캐시 설정에 대한 정적 분석 결과로 해석해야 한다.

### 3. 캐시 미스 분류 및 집계

CASA는 집계된 캐시 미스를 Compulsory miss, Capacity miss, Conflict miss로 분류한다. 이를 위해 YAML 설정에 따라 구성된 캐시 모델과 동일한 총용량을 갖는 완전 연관 비교 캐시를 함께 사용한다. 그림 2는 완전 연관 비교 캐시를 이용한 캐시 미스 분류 방식을 나타낸다.

CASA는 캐시 라인의 최초 접근 여부를 먼저 확인하여, 처음 접근된 라인에서 발생한 미스를 Compulsory miss로 분류한다. 이후 접근된 라인에서 미스가 발생하면 사용자 설정 캐시 모델과 완전 연관 비교 캐시의 결과를 비교한다.

완전 연관 비교 캐시에서 히트가 발생하면 제한된 연관도나 집합 매핑 충돌에 의한 Conflict miss로, 두 캐시에서 모두 미스가 발생하면 캐시 용량 부족에 의한 Capacity miss로 분류한다.

CASA는 분류된 캐시 미스를 메모리 객체와 Load/Store 연산 단위로 집계하여, 캐시 미스가 주로 발생하는 데이터 객체와 접근 유형을 확인할 수 있도록 한다. 이러한 결과는 프로그램의 캐시 병목 원인을 분석하고, 데이터 배치 변경이나 접근 순서 개선과 같은 최적화 방향을 도출하는 데 활용될 수 있다.

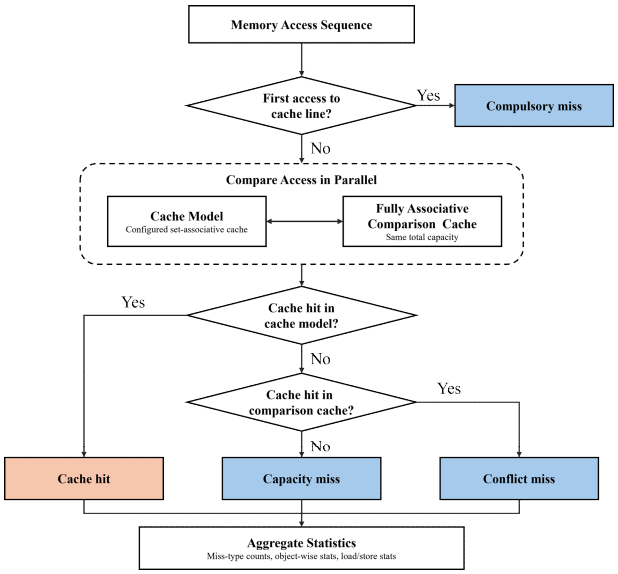


그림 2. 완전 연관 비교 캐시를 통한 미스 분류

## Ⅳ. 실험 및 평가

본 장에서는 CASA를 두 가지 관점에서 평가한다. 첫째, Cachegrind와의 비교를 통해 CASA가 주어진 캐시 설정에서 기존 동적 프로파일링 도구와 일관된 캐시 미스 수를 산출하는지 확인한다. 둘째, PolyBench 벤치마크에 CASA를 적용하여 다양한 배열 접근 패턴에 대한 적용 가능성과 분석 결과의 유용성을 확인한다.

여기서 정확성은 실제 하드웨어 성능 카운터 값과의 일치가 아니라, 동일하거나 유사한 캐시 모델과 접근 조건에서 기존 도구와 일관된 결과를 산출하는지를 의미한다.

### 1. 실험 설정

실험에 사용한 캐시 구성은 표 1과 같다. 본 실험은 단일 코어 환경을 가정하며, 캐시 계층은 L1 캐시, L2(LLC), 그리고 주기억장치로 구성된

다. 캐시 크기, 라인 크기, 연관도, 교체 정책, 쓰기 정책은 YAML 설정 파일을 통해 지정하였다. 입력 프로그램은 Cachegrind 비교 검증을 위한 PolyBench 벤치마크로 구성된다.

표 1. 실험에 사용한 캐시 구성

| 항목 | L1 | L2 (LLC) |
|---|---|---|
| 용량 | 32 KiB | 2 MiB |
| 캐시 라인 크기 | 32 B | 32 B |
| 연관도 | 4-way | 4-way |
| 교체 정책 | LRU | LRU |
| 쓰기 정책 | Write-back | Write-back |
| 쓰기 할당 정책 | Write-allocate | Write-allocate |
| 지연 사이클 | 1 cycle | 10 cycles |

## 2. Cachegrind 비교 기반 정확성 검증

정확성 검증은 CASA가 주어진 접근 패턴과 캐시 구성에 대해 캐시 미스 수를 일관되게 산출하는지 확인하기 위해 수행하였다. 이를 위해 동적 캐시 프로파일링 도구인 Cachegrind의 결과와 CASA의 분석 결과를 비교하였다. Cachegrind는 프로그램 실행 중 발생한 메모리 접근을 기반으로 L1 및 LL 캐시 미스 수를 산출하므로, CASA의 정적 분석 결과를 검증하기 위한 비교 기준으로 사용하였다.

비교 대상 프로그램으로는 PolyBench 벤치마크의 2mm, atax, correlation, gemm, jacobi를 사용하였다. 각 프로그램에 대해 Cachegrind에서 측정한 L1 및 LL 캐시 미스 수와 CASA가 산출한 L1 및 LL 캐시 미스 수를 비교하였다. 두 도구의 캐시 설정은 가능한 동일하게 맞추었으며, 비교 대상은 각 계층에서 발생한 캐시 미스 수로 설정하였다.

그림 3은 Cachegrind와 CASA의 캐시 미스 수를 비교한 결과를 나타낸다. 각 워크로드에서 CASA의 결과는 Cachegrind 결과와 전반적으로 유사한 경향을 보였으며, L1 캐시와 LL 캐시 모두에서 두 결과가 거의 일치하는 것을 확인할 수 있다. 특히 LL 캐시의 경우 모든 워크로드에서 매우 작은 차이를 보였으며, 전체 평균 절대 백분율 오차는 L1 캐시에서 0.7%, LL 캐시에서 0.1%로 나타났다.

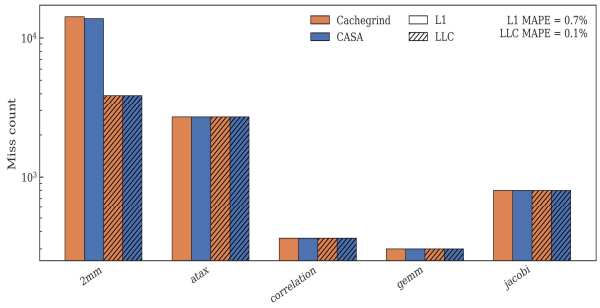


그림 3. Cachegrind와 CASA의 캐시 미스 수 비교

이 결과는 CASA가 정적으로 추출된 접근 패턴을 기반으로 캐시 미스 수를 계산하더라도, 실행 기반 도구인 Cachegrind와 유사한 분석 결과를 산출할 수 있음을 보여준다. 따라서 CASA는 주어진 캐시 설정과 정적으로 전개 가능한 접근 패턴에 대해 Cachegrind와 일관된 캐시 미스 분석 결과를 산출함을 확인하였다.

## 3. PolyBench 기반 평가

이 절에서는 CASA를 PolyBench 벤치마크에 적용하여, 다양한 배열 접근 패턴을 갖는 프로그램에서 캐시 구성 변화에 따른 분석 결과가 일관되게 산출되는지를 확인하였다. 먼저 PolyBench의 2mm 벤치마크를 대상으로 L1 캐시의 연관도를 2-way, 4-way, 8-way로 변경하면서 Cachegrind와 CASA의 캐시 미스 수를 비교하였다.

2mm 벤치마크는 두 단계의 행렬 곱셈을 수행하는 프로그램으로, 여러 배열에 대한 접근이 발생한다. 이 과정에서 행 우선 접근과 열 방향 접근이 함께 나타나므로, 캐시 연관도 변화에 따라 캐시 미스 수가 달라질 수 있다. 그림 4는 L1 캐시 연관도 변화에 따른 Cachegrind와 CASA의 L1 캐시 미스 수를 비교한 결과이다.

실험 결과, CASA는 모든 연관도 설정에서 Cachegrind와 유사한 변화 추세를 보였다. 2-way, 4-way, 8-way 설정에서 두 도구의 L1 캐시 미스 수는 각각 3.1%, 1.3%, 2.4%의 차이를 보였으며, LL 캐시 미스 수는 모든 설정에서 동일하게 나타났다. 이는 CASA가 정적으로 전개한

접근 패턴을 기반으로 하더라도, 캐시 연관도 변화에 따른 캐시 미스 수의 상대적 변화를 기존 동적 프로파일링 도구와 일관되게 반영할 수 있음을 보여준다.

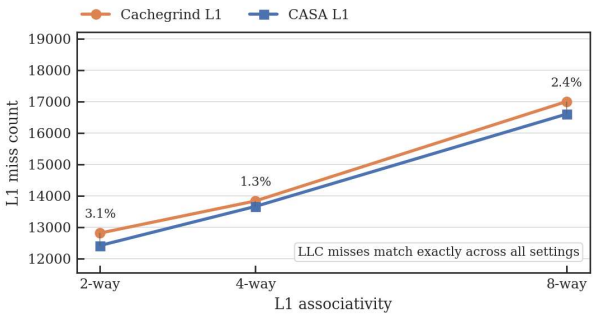


그림 4. L1 연관도 변화에 따른 벤치마크의 캐시 미스 수 비교

다음으로 PolyBench의 대표적인 워크로드인 2mm, atax, gemm, jacobi, correlation에 CASA를 적용하여 워크로드별 캐시 미스 유형 분포를 분석하였다. 각 워크로드는 서로 다른 배열 접근 패턴과 반복 구조를 가지므로, 캐시 미스의 발생 양상도 다르게 나타난다. 그림 5는 각 워크로드에서 발생한 L1 캐시 미스를 Compulsory miss, Capacity miss, Conflict miss로 분류한 결과를 나타낸다.

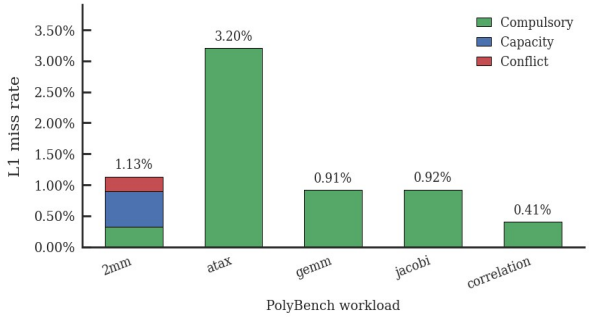


그림 5. Polybench 워크로드의 L1 캐시 미스 분류

분석 결과, 각 워크로드는 접근 패턴에 따라 서로 다른 캐시 미스 분포를 보였다. 예를 들어 2mm는 여러 배열에 대한 중첩 접근으로 인해 Compulsory miss 외에도 Capacity miss와 Conflict miss가 함께 나타났으며, atax와 jacobi는 상대적으로 Compulsory miss의 비중이 크게 나타났다. 이러한 결과는 CASA가 단순히 전체 캐시 미스 수를 산출하는 데 그치지 않고, 워크로드별 캐시 비효율의 원인을 미스 유형 관점에서 분석할 수 있음을 보여준다.

따라서 PolyBench 기반 평가를 통해 CASA가 다양한 배열 접근 패턴을 갖는 벤치마크에 적용 가능하며, 캐시 구성과 워크로드 특성에 따른 캐시 미스 양상을 분석할 수 있음을 확인하였다.

## V. 결론

본 논문은 정적 접근 패턴을 이용하여 프로그램 실행 없이 캐시 동작을 분석하는 프레임워크인 CASA를 설계 및 구현하고, 그 정확성과 적용 가능성을 평가하였다. PolyBench 벤치마크를 대상으로 Cachegrind와 비교한 결과, CASA는 L1 및 LL 캐시 미스 수에서 각각 0.7%와 0.1%의 평균 절대 백분율 오차를 보였으며, 기존 동적 프로파일링 도구와 일관된 분석 결과를 산출함을 확인하였다.

다만 본 평가는 PolyBench 벤치마크 중심으로 수행되었으므로, 다양한 실제 응용 프로그램과 하드웨어 환경에 대한 추가적인 검증이 필요하다. 현재 CASA는 단일 코어 환경과 LRU 교체 정책을 가정하므로, 향후 다중 코어 환경의 캐시 일관성, 다양한 교체 정책, 실제 하드웨어 특성을 반영하는 방향으로 확장할 계획이다. 또한 캐시 미스 유형 및 객체 단위 분석 결과를 WCET 분석과 연계하여, 정적 타이밍 검증을 지원하는 프레임워크로 발전시킬 계획이다.

## REFERENCES

[1] J. L. Hennessy and D. A. Patterson, “*Computer Architecture: A Quantitative Approach,*” 6th ed. Morgan Kaufmann, pp. 78-164, 2017.
[2] N. Nethercote and J. Seward, “Valgrind: a framework for heavyweight dynamic binary instrumentation,” *ACM Sigplan notices*, vol. 42, no. 6, pp. 89-100, 2007.
[3] N. Binkert et al, “The gem5 simulator,” *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1-7, 2011.
[4] G. J. Holzmann, “The power of 10: rules for developing safety-critical code,” *IEEE Computer*, vol. 39, no. 6, pp. 95 - 99, 2006.

[5] PolyBench/C the Polyhedral Benchmark suite (2012). https://www.cs.colostate.edu/~pouchet/software/polybench/ (accessed Jun., 7, 2026).
[6] A. Razzak, A. Barai, N. Santhi, and A.H. Badawy, "Static reuse profile estimation for array applications," *Proceedings of the International Symposium on Memory Systems*, pp. 235-244, Washington, DC, USA, 2024.
[7] M. Lv, N. Guan, J. Reineke, R. Wilhelm, and W. Yi, "A Survey on Static Cache Analysis for Real-Time Systems," *Leibniz Trans. Embedded Systems (LITES)*, vol. 3, no. 1, pp. 05:1 - 05:48, 2016.
[8] C. Ferdinand and R. Wilhelm, "Efficient and Precise Cache Behavior Prediction for Real-Time Systems," *Real-Time Systems*, vol. 17, no. 2 - 3, pp. 131 - 181, 1999.
[9] M. D. Hill and A. J. Smith, "Evaluating associativity in CPU caches," *IEEE Trans. Computers*, vol. 38, no. 12, pp. 1612 - 1630, 1989.

## 저 자 소 개

**박준용(준회원)**
2025년 인하대학교 컴퓨터공학과 학사 졸업.
2025년~현재 인하대학교 전기컴퓨터공학과 석사과정 재학
<주관심분야 : 운영체제, 임베디드 시스템, 시스템 소프트웨어>

**신지우(정회원)**
2023년 인하대학교 컴퓨터공학과 학사 졸업.
2025년 인하대학교 전기컴퓨터공학과 석사 졸업.
2025년~현재 인하대학교 전기컴퓨터공학과 박사과정 재학.
<주관심분야 : 지능형 임베디드 소프트웨어, 실시간 스케줄링>

**맹훈규(준회원)**
2020년~현재 인하대학교 컴퓨터공학과 학사과정 재학.
<주관심분야 : 실시간 운영체제, 시스템 소프트웨어, 임베디드 시스템>

**전현수(정회원)**
2024년 인하대학교 경제학과 학사 졸업.
2026년 인하대학교 전기컴퓨터공학과 석사 졸업.
2026년~현재 인하대학교 전기컴퓨터공학과 박사과정 재학.
<주관심분야 : 시스템 소프트웨어, 실시간 운영체제>

**박성민(정회원)**
2009년 홍익대학교 전자전기공학부 공학사 졸업.
2011년 홍익대학교 전자정보통신공학과 공학석사 졸업.
2011년~현재 LIG Defense & Aerospace 정지궤도위성 개발단 3팀 수석 연구원
<주관심분야 : 임베디드 시스템, 운영체제, 시스템 소프트웨어, 위성 비행 소프트웨어, 위성 시스템, 전파산란>

**허금숙(정회원)**
1996년 한림대학교 전자계산학과 이학사 졸업.
1996년~2007년 정보통신 분야 임베디드 SW 개발.
2011년~2019년 LIG 시스템(주) 무기체계 분야 임베디드 SW 개발
2020년~현재 LIG Defense & Aerospace 정지궤도위성 개발단 3팀 수석 연구원
<주관심분야 : 임베디드 SW, 운영체제, 시스템 소프트웨어, 위성 비행 소프트웨어, 위성 시스템>

**강지호(정회원)**
2023년 경희대학교 우주과학과 이학사 졸업.
2023년~현재 LIG Defense & Aerospace 정지궤도위성 개발단 3팀 선임 연구원
<주관심분야 : 임베디드 시스템, 운영체제, 시스템 소프트웨어>

**정진만(종신회원)**
2008년 서울대학교 컴퓨터공학과 학사 졸업.
2014년 서울대학교 전기컴퓨터공학과 박사 졸업.
2014~2021년 한남대학교 정보통신공학과 부교수.
2021년~2026년 인하대학교 컴퓨터공학과 부교수
2026년~현재 인하대학교 컴퓨터공학과 정교수
<주관심분야 : 운영체제, 임베디드 시스템, 시스템 소프트웨어>