명령어 생략 기반 컴퓨터구조 분석 도구

여운장 [1], 조영필 [2]
[1] 한양대학교 컴퓨터소프트웨어학과 (미래자동차-SW 융합전공) 석사과정
[2] 한양대학교 컴퓨터소프트웨어학과 교수
lexture17@hanyang.ac.kr, ypcho@hanyang.ac.kr

SKIP based Technological Research Agent for Computer-structural Exploration and Response

Un-Jang Yeo[1] , Yeong-Pil Cho[2]
[1]Dept. of Computer and Software (Automotive-Computer Convergence), Han-yang University
[2]Dept. of Computer Science, Han-yang University

요 약

Fuzzing Test 와 같은 자동화된 소프트웨어 테스트 기법이 점차 출현함에 따라, 소프트웨어 테스트의 시간적 효율성과 성능의 상충을 조절하여 최적의 테스트를 진행하려는 시도가 발생하고 있다. 본 연구는 이러한 소프트웨어 테스트 기법을 하드웨어 단위에서 지원할 수 있는 기능을 확률 기반 명령어 생략 구조를 통해 제시하였다.

1. 서론

소프트웨어의 중요도가 증가함에 따라 소프트웨어 테스트 기법 또한 발전하고 있다. 그러나 이러한 소프트웨어 테스트는 소프트웨어 수준에서의 입력값 제어를 통한 동적 반복 분석이나 코드의 정적 분석을 기반으로 하는 경우가 대다수이며, 이로 인해 병렬화의 어려움과 시간 소모 증가 등의 문제점을 안고 있다.

SKIPTRACER 는 하드웨어 수준에서 소프트웨어 테스트를 지원할 수 있도록 컴퓨터구조 수준에서 명령어 생략 기능을 추가한 시도로, 본 연구에서는 해당 기능 추가의 효용성과 의의를 제시하였다.

2. 배경지식

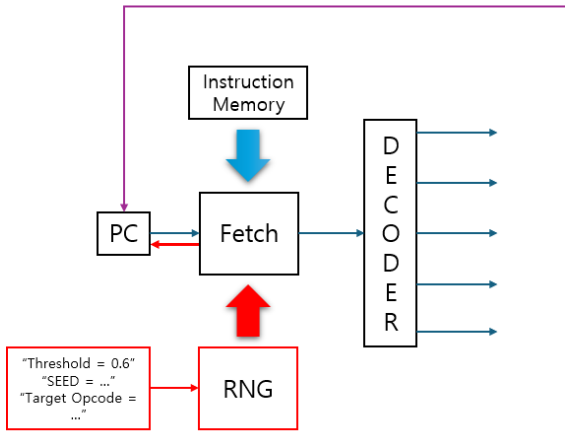
“소프트웨어 테스트”는 개발한 소프트웨어가 개발 의도대로 동작하는지를 검사, 검증하는 과정을 통칭한다. 검사(Verification)는 소프트웨어의 개발 단계별 테스트, 검증(Validation)은 개발 종료 및 완성 단계에서의 전체 소프트웨어 동작에 대한 테스트를 의미한다[1]. 본 연구는 검사와 검증 양면에서 각각의 소프트웨어 기능 시험에 대한 시간 소모를 줄이는 테스트 기법에 대해 다루고 있다.

3. SKIPTRACER

SKIPTRACER, SKIP based Technological Research Agent for Computer-structural Exploration and Response 는 특정 명령어에 대한 확률 기반 생략 기능을 도입해 기존 하드웨어/소프트웨어 분석 기능을 지원하는 보조 기능을 지칭한다. 본 기능의 중추인 난수 생성과 명령어 생략 기능은 하드웨어에서 구현되어 소프트웨어 작성자에게 별도의 추가 작업을 요구하지 않는다. 또한 명령어 생략의 중추를 제 3 의 모듈로서 구현하여, 해당 모듈에 접근할 수 있는 실험자가 테스트 중간에도 생략 여부 및 확률 등의 변수를 직접적으로 조작해 안정적으로 테스트에 개입할 수 있다.

무분별하게 적용된 명령어 생략은 소프트웨어의 제어 흐름을 방해하여 소프트웨어 테스트 결과의 무결성을 해칠 수 있다. 예를 들어 실행파일의 흐름에 조건분기 루틴이 포함될 경우, 조건분기 명령어만을 생략해 다음 명령어를 실행하는 것은 제어 흐름을 해칠 가능성이 높다. 이에 SKIPTRACER 는 생략 대상 명령어의 종류를 설계 수준에서 직접 지정하고, 각각의 명령어가 실행되었을 경우의 다음 PC(Program Counter)값을 Branch 예측 등을 통해 예상해 이동해야 하는 주소값을 계산하도록 설계되었다.

확률적 명령어 생략은 Instruction Fetch 과정을 수정하여 구현한다. 난수를 생성한 뒤 설정된 확률 역치에 따라 TRUE/FALSE 를 생성하는 별도의 난수생성기를 설치한 뒤, Instruction Fetch 과정에서 명령어의 Opcode 와 난수생성기의 결과값, 해당 Opcode 에 대해 설정한 PC 변경값 등을 확인하여 해당 Instruction 의 생략 여부와 생략 시 참조해야 할 다음 명령어 위치 등을 결정한다.



(그림 1) RISC-V 에 SKIPTRACER 를 적용하였을 경우를 가정한 간략한 IF Stage 도식.

명령어의 생략이 확정될 경우 해당 명령어는 NOP(No Operation) 명령어와 동일하게 처리된다. 따라서 명령어의 생략은 Fetch 사이클의 Stall 을 유발함을 명시한다.

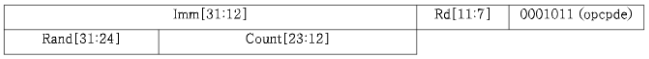
이러한 명령어 생략은 반복되고 자원 소모가 심한 서브루틴 및 런타임 라이브러리나, 그러한 기능을 자주 사용할 것으로 예상되는 소프트웨어의 테스트에 효과적이다. 한 번의 시스템 테스트에서 시간소모가 큰 서브루틴 A 를 자주 사용할 것으로 예상되는 경우, 50%의 역치를 설정한 SKIPTRACER 환경을 병렬적으로 두어 비교적 적은 시간소모로 테스트할 수 있다.

4. 적용 및 실증

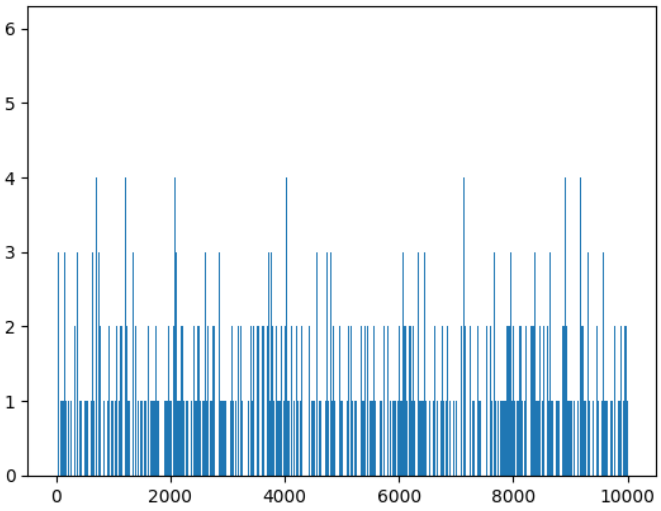
개념의 적용 및 실증을 위해 FPGA 장비를 통해 RISC-V 컴퓨터구조를 구현하여 실험환경을 설정하였다. 본 실험에선 Genesys 2 FPGA 와 openhwgroup cva6 RISC-V 구조를 적용한 Linux 를 채택하였다.

용이한 실증을 위해 본 실험에선 SKIP 이라는 커스텀 명령어군을 정의한다. 본 명령어는 보유하고 있는 역치값 Rand, PC 조정값 Count 를 통해 SKIPTRACER 의 확률 기반 명령어 생략 로직을 모방한다. Rand/256 값을 역치 확률로 두며, 생략이 결정된 경우 SKIP 이 삽입된 PC 와 Count*4 값을 기준으로 다음 PC 를 결정한다. 실험에 적용한 PRNG 는 난수생성에 LFSR 을 사용하도록 구성되었으며, 매 실험마다 고정 시드값을 변경하여 테스트케이스를 생성한다.

본 실증에서는 시간 소모가 큰 런타임 루틴으로서 Sanitizer, 그 중에서도 AddressSanitizer 를 선택하였다. 전역 및 각 스코프의 지역 메모리에 대해 명시적으로 AddressSanitizer 를 발생시키는 루틴을 배치하여 그 위치를 확인한 뒤, 실행파일의 컴파일 과정에서 AddressSanitizer 루틴 진입점을 생략 대상으로 하는



(그림 2) 실증에 사용한 SKIP 명령어의 포맷



(그림 3) 서브루틴의 전체 사용 빈도 그래프 중 하나

SKIP 을 삽입한다. 삽입된 각각의 SKIP 에 대해 Rand 를 조정하며 AddressSanitizer 진입 횟수와 총 시간 소모를 측정한다.

그림 3 은 10000 번의 ASan 사용이 예상되는 실행파일에 대해 역치 12.5%를 적용한 SKIPTRACER 배치를 8 회 수행한 시나리오에 대해, 각 서브루틴이 수행된 횟수를 나타낸 그래프이다. 배치 1 회에 대해 수행된 서브루틴 수는 1250 개 내외로 집계되며, 산술적으로 100000 번의 12.5%로 집계된다. 실증한 전체 테스트케이스에 대해, 전체 배치에 대해 단 한번도 수행되지 않은 서브루틴 수는 평균 약 3000 번이다.

5. 결론 및 향후 연구방향

SKIPTRACER 는 Instruction 의 생략 여부를 결정하는 외부 모듈을 통해 구현되기 때문에 기존의 컴퓨터 구조에 적용하는 데에 큰 노력을 요하지 않는다. 또한 적은 비용으로 자원 소모가 심한 소프트웨어 시스템에 대해 동적 분석의 소형화 및 병렬화를 지원하며, 시스템 내의 각각의 기능들에 대한 세부설정으로 더 폭넓은 문제에 대응할 수 있도록 한다.

그러나 확률 기반으로 대상 명령어를 생략하는 만큼 성능 유지와 제어 흐름 무결성을 입증할 추가 실험이 요구된다.

예를 들어, 전체적인 분포를 생략 대상들의 실제 생략이 특정 대상으로 편향될 수 있다. 확률 역치를 일정 수준 이상 높게 설정하여 완화할 수 있으나, 방지하기 위해선 추가적인 알고리즘을 요구한다.

또한 생략 대상이 제어 흐름에 큰 영향을 주거나 그 제어 흐름을 예측하기 어려운 경우, 또는 생략 대상들끼리 상호작용하는 흐름이 존재하는 경우 현재의 SKIPTRACER 구조만으로 제어 흐름의 무결성을 보장할 수 없다. 실험에 사용된 형태의 명령어 생략 기능은 명령어 생략이 결정될 경우 Stall 이 발생한다는 문제가 있어 제어 흐름 무결성을 해칠 가능성이 비교적 높다.

이러한 예상 문제점에 대한 후속 연구가 진행될 경우, 대규모 소프트웨어의 테스트를 병렬화된 소형 테스트케이스 형태로 테스트하여 자원 소모를 크게 줄일 수 있을 것으로 기대된다.

이 논문은 과학기술정보통신부의 재원으로 정보통신기획평가원(No. 2020-0-01840, 스마트폰의 내부데이터 접근 및 보호 기술 분석)과 한국연구재단(No. NRF-2022R1A4A1032361, Processing-in-Memory 보안 기술 개발)의 지원을 받아 수행된 연구임

참고문헌

[1] Jai Gaur, Akshita Goyal, Tanupriya Choudhury, Sai Sabitha, “A Walk Through of Software Testing Techniques”, 2016 International Conference System Modeling & Advancement in Research Trends (SMART), Moradabad, India, 2016