SIGPL 2024 Trip Report

이준협

February 2, 2024

1 감상

성심당은 맛있다. 특히 샌드위치들이 (1일차: 참치, 2일차: 소세지) 참 맛있다. 1일차 조별활동 닭도리탕 맛있었다.

2 강연

이번 SIGPL에서는 기존에 많이 보지 못한 연구자 분들의 강연을 들을 수 있도록 편성을 하였다고 한다. 간단하게 이번에 들은 강연들의 내용을 여기에 정리해보겠다.

딥 전이학습 기반 프로그램 자동 수정: 산업 적용 사례를 중심으로 (김미수 (전남대)) 정보검색 기반 버그 추적 자동화 (IRBL: Information retrieval-based bug localization): 버그 리포트를 받아서 결함이 있는 부분이 어디인지 알려주는. 실제 버그 리포트를 받아서 그 버그 리포트의 성질 분석. 이후 버그 리포트에서 중요한 부분을 찾기 위한 기술 연구 (최적화 기반 쿼리 축소, 워드 임베딩 기반 쿼리 확장)

자동 프로그램 수정 (Automated Program Repair) 특히, 패치 생성 기법: 템플릿 기반 패치 생성: 템플릿을 생성하는 방법? (비용함수 정의, 비용을 최소화하는 최적화 문제로 템플릿 생성)/ 딥러닝(이미 있는 패치에 대한 훈련): "잡음 토큰"이 있다는 것 파악, 제거하니까 성능이 88% 향상/ 딥전이학습(이미 있는 모델에 패치 생성능력 장착): 삼전 모바일 사업부 의뢰, 코틀린 프로젝트에 대한 자동 프로그램 수정. 간단한 오류에 대한 수정? 패치 생성할 때 시간을 최소화. 실시간으로 수정할 수 있는 기술? 딥러닝 파이프라인 안에 통합할 수 있는, 결함(SonarQube 정적분석기가 알려주는)–패치(실제 패치) 코드 쌍 학습 $\rightarrow$ 패치 생성하는 모델.

실제 버그를 많이 보고 어떤 기술이 개발자들에게 가장 피부로 다가오는 결과를 줄 수 있는지 고민한 흔적이 보이는 좋은 연구였다. 실제 사용에 중요한 점: (1) 기존 파이프라인에 적응성 및 유연성 (2) 정확도 보장 PL을 LLMC에 학습할 때의 이슈: (1) 자연어 받는 모델에 어떻게 프언 이해시키나? (2) 데이터 셋이 작은 언어에 대한 데이터 생성?

임베디드 시스템을 위한 컴파일러 디자인 (허선영 (경희대)) Automatic Program Partitioning (분산시스템, 특히 여러 다른 기기들이 협동해야하는 분산시스템을 만들기 간편한 컴파일 기법). 물건 중심

프로그래밍 방식으로 프로그래밍하면, 각 기기별 Class가 저절로 잘 협동하도록(method call→remote call) 컴파일을 해주는 기술 개발.

Compilers for embedded systems (정보 처리 시스템이 더 포괄적인 일을 하는 제품의 일부로 포함되는 경우) 저전력 시스템: 거의 어셈블리 레벨에서 프로그래밍해야함. 운영체제 도움 없이. 그런데, 요즘에는 머신러닝 모델을 저전력 환경에서 실행하고 싶어함. IoT에 머신러닝 모델 들어가면 계산량 늘어나고, 동시에 외부 환경과의 상호작용 해야하므로, 그러나 전력/메모리 제약이 있고. 이러한 제약하에서 잘 프로그래밍 하기 위해 Tiny ML framework라는 것들이 만들어지고 있음. 저전력 임베디드 시스템을 위해 컴파일이 되도록 API를 제공해주는 틀이다. 특히, TensorFlow Lite: 메모리 제약 극복하기 위한: 동적 메모리 사용 최적화(activation을 공용 가능한 메모리 구역에 배치, tensor arena), 부동소수점 정확도 낮추기 (quantization), 실행 파일 크기 압축 (필요한 연산자 정의만 가져옴). 이러한 접근의 제약: 개발과정이 라이브러리에 매우 강하게 의존적임, arena 크기 사람이 정해줘야함, 메모리 효율을 위한 조건문 때문에 성능이 떨어짐.

Edge Impulse (application-level) μtvm (operation-level) 등이 제약점 해결하기 위해 나옴. 교수님의 해결책: TinyGen. 이미 있는 model binary에서 최적화한 모델 **코드**를 뽑아내서 라이브러리에 대한 의존도를 최대한 낮추겠다! 그 코드만 가지고 프로그래밍하면 되기 때문. 구현: MLIR에서 제공해주는 TOSA IR, EmitC IR을 활용해서 모델을 C++까지 번역함. 어려워보이는 최적화: Static memory plan을 해야함(liveness analysis)

근사 컴퓨팅의 자동 적용을 위한 컴파일러 최적화 기법 (박영준 (연세대)) GPU를 사용하는 근사 컴퓨팅: GPU의 구조는, 같은 일을 하는, 서로 의존하지 않는 많은 작업이, 동시적으로 돌아가는 구조를 가지고 있다. GPU의 구조를 가장 잘 이용할 수 있으려면, 내 프로그램 특성에 맞춰서 코드를 짜야한다. 하지만 환경과 상호작용에 따라 프로그램 실행 특성이 많이 달라질 수 있다: 동적으로 최적화를 할 필요가 있다.

근사컴퓨팅: 최소한 만족 가능한 품질 내에, 최대한의 성능을 이끌어낼 수 있는 근사를 해야한다. 근사컴퓨팅을 위한 컴파일 최적화: 최적화 패턴이 몇가지 있고, 그 패턴 중 여러가지를 해봄. 그 중 품질이 괜찮은 것을 선택. 하지만, 당연히도, 패턴을 많이 만들어 놓아야 최적화 기회가 많이 생긴다. 또한, 같은 의미를 가지지만 모습이 달라서 패턴 적용 못하는 경우 많이 있음. 그러면, 의미 분석 거친 IR 레벨/ASM 레벨에서 패턴 정의하자! 또한, 최적화마다 입력에 따라 품질에 영향을 많이 주고/안주고 차이가 있음. 최적화마다 품질에 미치는 영향의 입력-민감도를 고려해서 적용해야하지 않나?

ASM 레벨 dataflow 그래프를 봐서 최적화 가능성을 판단하자!

LeakPair: 단일페이지 웹 어플리케이션내의 메모리 누수를 고치기 위한 선제적 디버깅 (김동선 (경북대)) Memory leak의 refactoring하는 연구를 하겠다! (최근에, OOM 에러로 webpage crash가 많이 일어남) 일단, 단일페이지(한 페이지 내에서 비동기적으로 필요한 컨텐츠만 가져오는)에서 나타나는 에러에 대한 자동수정 연구를 해보겠다. 그런데, 디버깅하기 어렵고, 재현하기 어려움, 즉 잘 안 고침. 그러나, 패턴 파악을 잘 하고, non-intrusive하다는 것을 설득하면?

문제 파악: 브라우저 문제인가? 아니다. GC로 회수할 수 없는 문제. 무조건 새 주소를 할당하는 방식이니까. (`removeEventListner`같은 것을 쓰지 않음) 그러면 이런 버그를 어떻게 디버깅할 것인가?

“Proactive Debugging” (vs. Classical (reactive) debugging: 먼저 탐지하고, 나중에 고침): 그러지 말고, 일단 고쳐보고, 이후에 검증하자. 패치 생성을 어떻게 할 것인가? 패턴 파악. StackOverflow, GitHub PR 바탕. 메모리 사용량으로 검증.

다중 노드 환경에서 빅데이터 스트리밍 엔진의 성능 평가 방법 (양신형 (연세대)) 컴퓨터 **사이**에 소요 시간을 측정하는 방법? 결국 시간 동기화 문제. NTP: 35[ms] 정도 오차 존재. Precision Time Protocol (PTP): 정확하지만, 비쌈. Return-trip Measurement Method: ACK으로 시간 확인. 제안: Cloudprofiler (CP) Duration Measurement Method: Global Clock, ‘초’단위 없앰. 그러면 어떤 단위로 시간을 파악할까? CPU에서 측정하는 TSC(timestamp counter). 내 CPU와 상대 CPU 사이에 comm을 해서, 내 CPU의 TSC와 상대 CPU의 TSC 사이에 어떤 관계가 있는지 알고 싶다. RTT 측정을 할 때, 상대 TSC에 대응되는 나의 TSC를 정확히 알 수 없지만, 적어도 내가 보내고 받은 사이에 있음을 알 수 있다.

소프트 오류로부터의 신뢰성 향상을 위한 비트 단위 정적 분석 기법 (고유선 (연세대)) 믿을 수 있는 하드웨어를 설계하는 과정에서, 컴파일러 단위에서, 부팅까지 인증받지 않은 코드가 실행되지 않도록 보장할 수 있을까? 소프트 오류: random bit flip에 의해 코드가 변형되는 경우. CPU에서 이러한 오류가 발견된다면? 아니면 일부로 물리적으로 해킹을 위해 bit flip을 일으키는 경우가 있음.

이러한 소프트 에러를 컴파일러 관점에서 어떻게 모델링할 수 있을까? CPU에 있는, 소프트 에러가 발생 할 수 있는 구조마다 에러 발생에 의한 영향을 모델링해보자. Control-flow integrity (PC soft error, iCache), data-flow integrity (register file, iCache), memory-data integrity (memory, iCache). Evaluation: 실행 시간이 더 길어지기 때문에 소프트 에러가 발생할 수 있는 가능성이 더 높다 $\rightarrow$ 샘플링을 더 많이 해야한다.

Bit-Level Error Coalescing: 코드가 소프트 에러에 대하여 취약한 부분을 정적 분석을 통해, 비트 단위에서 파악하고 싶다! 먼저, 각 비트가 가질 수 있는 값을 top-down으로 포섭한 후, bottom-up으로 각 비트 변형이 일어났을 때 어떤 소프트 에러가 일어날 수 있는지 포섭함.

코드 속성 그래프의 부분 그래프 매칭을 통한 대규모 취약점 탐지 (위성일 (UNIST)) 프로그램 분석과, 보안 취약점을 드러낼 수 있는 테스트 입력을 생성하여 웹에서 일어날 수 있는 보안 취약점을 없애는 연구를 하고 계심. 이번 발표에서는, 코드 성질을 정적분석을 통해 그래프로 나타내고, 부분그래프 동일성 매칭을 통해 보안 취약점을 일으킬 수 있는 코드를 파악하는 연구.

코드 성질 그래프(Code Property Graph, CPG)=AST+CFG+PDG(Program Dependency Graph). 버그 찾는 graph query를 정의해서, PDG traversal을 통해서 버그 탐색을 할 수 있다. Traversal: sink function으로부터 역으로 흘러들어가는, sanitization 받지 않은 input을 찾음. 그러나, sanitization이 안전한지까지 검증하지 못한다.

이미 알려진 취약점들을 바탕으로, CPG의 부분그래프가 그러한 취약점과 유사한지 탐지한다. 하지만, subgraph isomorphism은 NP-complete 문제여서, 큰 프로그램에 대해 어렵다. 이를 해결하기 위해, CPG 가지치기 진행. (실제 취약점에 꼭 필요한 간선만 유지) 또한, 생긴 것이 동등하지 않아도 의미는 같을 수 있다. CPG를 잘 요약해야겠다.