Posudek diplomové práce

Matematicko-fyzikální fakulta Univerzity Karlovy

Autor práce	Jan Dubský		
Název práce	Extensible disassembler with support for interactive instruction reordering		
Rok odevzdání	2022		
Studijní program	Informatika	**Studijní obor**	Softwarové systémy

Autor posudku	Jakub Jermář	**Role**	Oponent
Pracoviště	externí		

Text posudku:

Autor se ve své diplomové práci zabývá návrhem metody pro detekci závislostí mezi instrukcemi strojového kódu. Její aplikací jsou nástroje pro ekvivalentní transformace a emulaci daného strojového kódu, typicky vygenerovaného optimalizujícím překladačem, za účelem usnadění jeho studia člověkem. V kontextu práce je ekvivalentními transformacemi myšlena především změna pořadí instrukcí. Autorovo řešení obsahuje jak platformně-nezávsilé části (transformace a emulace), tak části specifické pro konkrétní procesorovou architekturu (podpora dané instrukční sady) – v tomto případě RISC-V. Autor předpokládá snadnou přenositelnost na jiné procesorové architektury.

V rámci zvoleného řešení používá autor statickou analýzu, ve které rozdělí skupiny instrukcí strojového kódu na basic bloky. Každá instrukce je převedena na jakýsi mikrokód (autor užívá pojem intermediate code), který mimo jiné zahrnuje popis jejích výstupních efektů (zápis do paměti, registru). Každý výstupní efekt je určen jedním či více vstupními výrazy (čtení paměti, registrů, konstant, binární operace nad jinými výrazy). Celý mikrokód umožňuje detekovat závislosti mezi jednotlivými instrukcemi původního strojového kódu v rámci jednoho basic bloku a zároveň je jej možné interpretovat během případné emulace.

Diplomová práce je vyhotovena v anglickém jazyce.

Z formálního hlediska je práci možné vytknout poměrně velké množství překlepů a gramatických nedostatků. Autor ve většině textu používá 1. osobu množného čísla, ale ve 3. kapitole v několika větách přechází k číslu jednotnému. V textu dochází k nekonzistentnímu užívání typografických konvencí v označování binárních operací (*or* vs. OR), a terminologie (zero extension vs. unsigned extension).

I přes tyto formální nedostatky je text dobře strozumitelný, závěry kapitol obsahují užitečná shrnutí. Ve 4. kapitole popisující transformace kódu jsem ocenil názorné ilustrace, obzvláště pak ty analogicky zobrazující povolené a nepovolené redukce *width*-widgetu.

Při čtení práce si čtenář nutně začne klást určité otázky technického charakteru a s uspokojením na většinu z nich nalézá v dalších odstavcích či kapitolách odpovědi (např. ohledně algoritmu pro detekci basic bloků a jeho omezení nebo ohledně binárních operací, u kterých je možné určit konstantní výsledek na základně znalosti hodnoty jen jednoho z operandů).

Silnou stránkou práce je dle mého názoru myšlenka popsat cílovou instrukční sadu pomocí mikrokódu (intermediate code), hloubka jejího zpracování a také elegantní použití výsledného mikrokódu pro určení závislostí a emulaci.

Ve společné kapitole věnované analýze problému a návrhu jeho řešení jsem postrádal diskuzi o tom, proč byla zvolena právě statická metoda analýzy kódu, a ne třeba dynamická. Použití dynamické analýzy by přitom s ohledem na to, že jedním z cílů práce je emulace kódu, dávalo určitý smysl. V pozdějších kapitolách jsem se nemohl ubránit dojmu, že by některé technické problémy byly v takovém případě lépe řešitelné (např. kód, který modifikuje sám sebe, nalezení cílů nepřímých skoků).

Kapitola o parsování strojového kódu si klade poměrně ambiciózní cíl: navrhnout sekvencer strojového kódu tak, aby byl univerzálně použitelný pro libovolnou procesorovou architekturu – autor dokonce uvažuje o případných budoucích 128-bitových procesorech. Autor svůj návrh opírá o vlastnosti několika různých instrukčních sad: RISC-V, AMD64 a ARM. Nabízí se ovšem otázka, zda je tato množina instrukčních sad dostatečně reprezentativní. Na mysl přicházejí existující architektury s velmi dlouhým instrukčním slovem (VLIW), ve kterých může být skupina např. 3 instrukcí kódována dohromady v jednom *bundlu* (IA-64), a neplatí tak předpoklad uplatněný v práci, že každá instrukce začíná na jiné adrese. Jiným příkladem procesorové architektury, která moc nezapadá do celkového návrhu, je architektura SPARC, protože vybočuje z předpokladu práce, že k identifikaci adresového prostoru, se kterým pracuje instrukce, stačí statický řetězec (na SPARCu se může použít obsah registru). Vzhledem k ambicím na širokou přenositelnost na jiné instrukční sady považuji za příliš omezující vlastnost řešení rovněž fakt, že nepodporuje branch delay sloty, které jsou velice běžné (odhadem polovina existujících architektur je má, a nelze vyloučit, že v budoucnu budou vznikat nové, které je rovněž budou mít). Autor se k tomuto vyjadřuje ve 2. kapitole.

Celkově lze konstatovat, že autor v této práci dostatečným způsobem prokázal schopnost analyzovat daný problém a navrhnout jeho řešení. To funguje v rámci autorem zdokumentovaných omezení, která částečně plynou ze zvolené metody statické analýzy kódu a struktury použitého mikrokódu.

Práci doporučuji **k obhajobě.**

Práci nenavrhuji **na zvláštní ocenění.**

Datum 29.8.2022 **Podpis**