Posudek diplomové práce

předložené na Matematicko-fyzikální fakultě
Univerzity Karlovy v Praze

Posudek oponenta

Autor:	Bc. Martin Liška
Název práce:	Optimizing large applications
Stud. program a obor:	Softwarové systémy
Rok odevzdání:	2013
Jméno a tituly oponenta:	Mgr. Martin Mareš, Ph.D.
Pracoviště:	Katedra aplikované matematiky
Kontaktní e-mail:	mares@kam.mff.cuni.cz

Do nedávné doby byla většina optimalizací v překladačích testována zejména na malých, výpočetně intensivních programech, typicky benchmarcích SPECint a SPECfp. Předpokládalo se, že tyto benchmarky dostatečně věrně modelují chování skutečně používaného softwaru. Ukazuje se, že u rozsáhlých programů mají vliv na rychlost i jevy, které na malých testovacích příkladech nejsou pozorovatelné, mezi jinými doba načítání programu z disku nebo efektivita využívání cachí.

Předložená práce si vytkla za cíl zkoumat práci překladače GCC na rozsáhlých aplikacích – jako příklady si vybírá open-source projekty s řádově miliony řádků kódu, jako je třeba jádro Linuxu, webový prohlížeč Mozilla Firefox, nebo kancelářský balík Open Office.

První část práce podává úvod do problematiky. Popisuje tradiční způsoby překladu a linkování programu a různá jejich rozšíření o globální optimalizace, jež překračují hranice funkcí či dokonce jednotlivých překladových jednotek. Podrobně se také věnuje dění při spuštění programu: chování loaderu a dynamického linkeru a různé známe způsoby, jak je zrychlit (prelink, přeuspořádání funkcí, ElfHack, apod.).

V následující kapitole autor analyzuje chování vybraných aplikací při překladu s různými globálními optimalizacemi (a zmiňuje četné chyby v aplikacích i kompilátoru a linkeru, na které přitom narazil). Také pomocí svého vlastního nástroje zkoumá činnost systému při startu aplikace. Zejména demonstruje nevýhodné pořadí čtení jednotlivých částí programu z disku.

Poté autor navrhuje dvě nové optimalizace pro překladač GCC. První z nich přeuspořádává funkce v závislosti na profilu změřeném za běhu programu. Rozšiřuje proto profilovací infrastrukturu GCC, aby zaznamenávala, kdy bylo do funkcí poprvé a naposledy vstoupeno. Na příkladech středně velkých aplikací ukazuje, jak přeuspořádání pomáhá startu programu.

Druhá optimalizace je podstatně zajímavější: vyhledává v celém programu sémanticky ekvivalentní funkce (které mohou snadno vzniknout například z generik) a sjednocuje je. Pomocí modifikovaného algoritmu pro value numbering rozděluje funkce na kongruenční třídy a poté v rámci každé třídy ověřuje, že funkce jsou opravdu ekvivalentní. Opět na reálných příkladech ukazuje, jak tato optimalizace pomáhá. Také ji srovnává s obdobnou optimalizací implementovanou v linkeru `gold` – překvapivě není ani jedna optimalizace nadmnožinou té druhé (pravděpodobně proto, že jedna pracuje na

úrovni mezikódu, zatímco druhá na úrovni vygenerovaných instrukcí), takže je lze účinně kombinovat.

Práci po odborné stránce považuji za velice kvalitní. Přidává do reálného překladače dvě zajímavé optimalizace, které mají slušnou šanci být začleněny do budoucích verzí GCC. Navíc ukazuje, že tyto optimalizace mají nejen teoretický význam, ale zlepšují i chování skutečných programů.

Po stránce formální by se ovšem práci dalo ledacos vytknout. Je zjevně sepsána ve spěchu, což se projevuje zejména velkým množstvím překlepů a obtížně srozumitelných vět. Některé části jsou také příliš stručné – například u ElfHacku ani `kdeinitu` není pořádně vysvětlena jejich funkce.

Navrhuji proto práci přijmout jako diplomovou a hodnotit ji známkou **velmi dobře.**

V Praze dne 30. srpna 2013
Martin Mareš