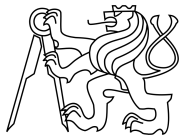
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Lehký just-in-time kompilátor pro aritmetické výrazy
Student:	Bc. Igor Kokorev
Vedoucí:	doc. Ing. Stefan Ratschan
Studijní program:	Informatika
Studijní obor:	Systémové programování
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Hodně softwarových balíků (např. tabulkové procesory, numerické řešiče) musí pořád počítat stejné aritmetické výrazy pro různé hodnoty. Cílem této práce je vývoj open-source kompilátoru, který produkuje účinný kód pro aritmetické výrazy a který lze snadno používat během běhu existujícího softwaru.
Pokyny:
- Vytvořte přesnou specifikaci kompilátoru.
- Naimplementujte prototyp kompilátoru, který překládá jazyk aritmetických výrazů do kódu pro balík GNU lightning a umí tento kód spustit.
- Rozšiřte kompilátor o výpočet derivací na základě automatického derivování (forward a reverse mode).
- Naimplementujte rozhraní mezi kompilátorem a typickým balíkem pro numerické výpočty (např. numerická optimalizace).
- Podrobně analyzujte chování kompilátoru v rámci výsledného balíku.
- Analyzujte důvody případné neúčinnosti a navrhněte zlepšení kompilátoru, např. pokročilými metodami generování kódu anebo automatického derivování.
- Vytvořte dokumentaci výsledného programu.

Seznam odborné literatury

- Rall, Louis B. (1981). Automatic Differentiation: Techniques and Applications. Lecture Notes in Computer Science 120. Springer. ISBN 3-540-10861-0.
- Griewank, Andreas; Walther, Andrea (2008). Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Other Titles in Applied Mathematics 105 (2nd ed.). SIAM. ISBN 978-0-89871-659-7.
- Aho, Lam, Sethi, Ullman: Compiler: Principles, Techniques and Tools (2nd ed.), 2010.
- GNU lightning documentation (http://www.gnu.org/software/lightning/manual/lightning.html).

L.S.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 23. října 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky

Diplomová práce

Lehký just-in-time kompilátor pro aritmetické výrazy

Bc. Igor Kokorev

Vedoucí práce: doc. Ing. Stefan Ratschan

10. ledna 2017

Poděkování

Rád bych poděkoval svému vedoucímu, panu doc. Ing. Stefanu Ratschanovi. Bez jeho bezmezné trpělivosti, vstřícnosti a nesmírně užitečných rad by realizace této práce nebyla možná.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. ledna 2017 .....

Odkaz na tuto práci

Kokorev, Igor. *Lehký just-in-time kompilátor pro aritmetické výrazy.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Úkolem této práce bylo vytvořit knihovnu, která je využitelná ve spojení s některou z knihoven pro nelineární optimalizaci. Pro daný aritmetický výraz (zadaný jako bajtový řetězec v kódování ASCII) je za běhu programu vytvořen strojový kód pro výpočet hodnoty funkce a také gradientu funkce v daném bodě. Tudíž není potřeba rekompilovat program pro každý nový aritmetický výraz reprezentující problém.

Klíčová slova just-in-time, kompilace, aritmetický, výraz, derivace

Abstract

The goal of this work was to create a library that could be used in conjunction with some library used for nonlinear optimization. For a given arithmetic expression (in the form of a byte string in ASCII encoding) machine code is generated during run time. This machine code computes the value of the function and also the gradient at the specified point. Thus it is not necessary to recompile the program for every new arithmetic expression representing a problem.

Keywords just-in-time, compilation, arithmetic, expression, derivative

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Cílem práce je vytvořit knihovnu, jejíž funkce generují strojový kód za běhu programu. Tento kód slouží k výpočtu hodnoty funkce v určeném bodě.

Pro úspěšné splnění zadání práce je potřeba následujících kroků v daném pořadí:

1. Vytvořit první specifikaci kompilátoru, aby bylo možné vytvořit prototyp.
2. Vytvořit prototyp podle specifikace z bodu 1.
3. Rozšířit specifikaci o novou funkcionalitu, mimo jiné specifikaci automatického derivování, aby bylo možné implementovat konečnou knihovnu.
4. Implementovat knihovnu.
5. Testovat účinnost výpočtu při použití a bez použití implementované knihovny.
6. Vyvodit závěry z předchozího bodu.
7. Sepsat uživatelskou příručku pro využití knihovny.

V následujících kapitolách budou jednotlivé kroky popsány podrobněji.

Kapitola 1

Prerekvizity

Pro úspěšné vytvoření dané knihovny je potřeba několika nástrojů. Než je možné začít generovat strojový kód, je potřeba provést syntaktickou analýzu vstupního bajtového řetězce. Z tohoto důvodu byly zvoleny tyto nástroje pro dosažení potřebných cílů dané knihovny:

- **GNU lightning**[1] pro generování strojového kódu během běhu programu
- **GNU Bison**[2] pro vytvoření parseru pro parsování vstupního bajtového řetězce

Samotná knihovna je psaná v jazyce C.

Kapitola 2

Počáteční specifikace kompilátoru

V následujících sekcích budou specifikovány tři komponenty:

- gramatika pro generování vstupních bajtových řetězců
- popis druhů uzlů v stromu, který je vytvořen z vstupního bajtového řetězce
- popis kusů strojového kódu vygenerovaného z jednotlivých uzlů stromu

Jelikož se jedná o počáteční specifikaci využitou při implementaci prototypu, specifikované jednotlivé komponenty nejsou konečné a budou pro výslednou knihovnu rozšířeny o nové funkcionality.

2.1 Specifikace vstupních řetězců

Vstupním parametrem jednotlivých funkcí knihovny (a také prototypu) je bajtový řetězec v kódování ASCII. Tento řetězec je slovo, které lze vygenerovat pomocí pravidel gramatiky v tabulce 2.1. Podtržené prvky v tabulce (kromě podtržítek v názvech neterminálů) jsou terminály. Daná tabulka není kompletní ze subjektivních důvodů a je aktuální pro implementovaný prototyp a bude rozšířena o nové elementy pro výslednou knihovnu.

2.2 Specifikace druhů uzlu v stromu

Strom vytvořený z vstupního řetězce reprezentuje daný vstupní aritmetický výraz. Jednotlivé druhy uzlů jsou popsány v tabule 2.2. Každý druh uzlu musí mít určitý přesný počet potomků. Tabulka 2.2 není úplná ze subjektivních důvodů.

levá strana pravidla	**pravá strana pravidla**	**příklad**
an_expression	faktor | an_expression operator an_expression	$x + 5$
faktor	variable_faktor | a_number | (an_expression)	(x)
...	...	...

Tabulka 2.1: Pravidla gramatiky, neúplná tabulka.

uzel	**povolené hodnoty**	**levý potomek**	**pravý potomek**
NUMDBL	čísla v pohyblivé řádové čárce standard IEEE 754 dvojitá přesnost	ne	ne
OPER	$+ - \times \div \wedge$	ano	ano
...	...	...	...

Tabulka 2.2: Druhy uzlů stromu, neúplná tabulka.

2.3 Specifikace posloupností strojových instrukcí pro uzly stromu

Z každého uzlu stromu z předchozí sekce je vytvořena posloupnost strojových instrukcí. Tabulka 2.3 obsahuje popis těchto posloupností pro každý z uzlů. Tabulka je ze subjektivních důvodů neúplná.

uzel	**vygenerovaný kód**
...	...

Tabulka 2.3: Kód generovaný z uzlů stromu, neúplná tabulka.

Kapitola 3

Popis vytvořeného prototypu

Vytvořený prototyp je program ovládaný přes příkazovou řádku. Zjednodušený popis fungování programu:

- Parametrem programu je bajtový řetězec v kódování ASCII.
- Aritmetický výraz reprezentovaný řetězcem musí obsahovat přesně jednu proměnnou. Toto omezení je odstraněno ve výsledné knihovně.
- Program vstupní parametr syntakticky analyzuje, vytvoří z něj strom a ze stromu strojový kód pro výpočet hodnoty funkce v daném bodě.
- Dále program po uživateli požaduje hodnotu pro jedinou proměnou v aritmetickém výrazu, obratem program vypíše na standardní výstup hodnotu funkce v daném bodě.
- Program je ukončen, jakmile uživatel dodá na vstup nevalidní hodnotu pro proměnou.

Kapitola 4

Rozšíření specifikace pro výslednou knihovnu

Tato kapitola by měla obsahovat rozšíření specifikace komponent z jedné z předchozích kapitol. Ze subjektivních důvodů je však tato kapitola neúplná.

Kapitola 5

Analýza chování kompilátoru

Zde by měly být testy účinnosti výpočtu řešení instancí problémů pomocí některého balíku pro řešení optimalizačních problémů s a bez využití kompilátoru pro aritmetické výrazy. Vzhledem k tomu, že implementace knihovny nebyla ze subjektivních důvodů dokončena, nelze provést testy a vyvodit z výsledků těchto testů závěry.

Lze ovšem odhadnout, že naivně vygenerovaný kód pomocí GNU lightning za běhu programu z vstupního bajtového řetězce bude méně efektivní než kód vygenerovaný z kusu kódu v jazyce C pomocí kompilátoru GCC. Konkrétní příklad pro porovnání:

- Vygenerovaný strojový kód z funkce myfunc z [3] pomocí GCC.
- Naivně vygenerovaný strojový kód pomocí GNU lightning z vstupního řetězce „x_2^(1/2)“.

Závěr

Ze subjektivních důvodů daná práce nemohla být dokončena v rozsahu, ve kterém bylo v plánu danou práci dokončit.

Kvůli nedokončené implementaci knihovny nebylo možné otestovat efektivitu. I bez testů na konkrétních problémech však lze odhadnout, že efektivita kódu vygenerovaného za běhu programu bude nižší než při vytvoření kódu pomocí kompilátoru GCC.

Kvůli nedokončené specifikaci nebylo možné dokončit implementaci knihovny.

Dokončený prototyp demonstruje pouze malou podmnožinu funkcionality výsledné implementace, pokud by byla dokončena.

Literatura

[1] Free Software Foundation: *Using GNU lightning*. Dostupné z: https://www.gnu.org/software/lightning/manual/lightning.html

[2] Free Software Foundation: *Bison 3.0.4*. Dostupné z: https://www.gnu.org/software/bison/manual/bison.html

[3] Massachusetts Institute of Technology: *NLopt Tutorial, Example in C/C++*. Dostupné z: http://ab-initio.mit.edu/wiki/index.php/NLopt_Tutorial#Example_in_C.2FC.2B.2B

Příloha **A**

Seznam použitých zkratek

ASCII American Standard Code for Information Interchange

IEEE Institute of Electrical and Electronics Engineers

GCC GNU Compiler Collection

GNU GNU is Not Unix