OPONENTSKÝ POSUDEK DISERTAČNÍ PRÁCE

Autor práce: Ing. Adam Husár

Název práce: Programování rekonfigurovatelných systémů pomocí vyššího programovacího jazyka (Programming of Reconfigurable Systems Using a Higher Programming Language).

Předložená disertační práce je zaměřena na návrh, programování a optimalizaci architektur aplikačně specifických procesorů. Cílem je vytvoření sady nástrojů pro automatizaci návrhového procesu, která umožní jednak efektivní a kvalitní návrh programovatelných výpočetních architektur, a dále pak povede ke zvýšení produktivity práce návrháře a zkrácení doby návrhu. Toto téma je dnes velmi aktuální jak z vědeckého, tak z praktického hlediska.

Disertace se skládá ze čtyř kapitol, včetně úvodu a závěru, seznamu použité literatury a pěti příloh.

V první kapitole (úvod) jsou na 4 stranách uvedeny cíle práce. Je zde diskutováno urychlování výpočtů pomocí paralelismu na úrovni vláken (TLP), na úrovni instrukcí (ILP) a na úrovni dat (DLP). S pomocí Amdahlova zákona autor ukazuje, že celkové urychlení úlohy je shora omezeno sekvenční částí výpočtu, kterou již nelze pomocí TLP urychlit. Tato skutečnost motivuje autora, aby svůj výzkum zaměřil na urychlování sekvenčních částí vykonávaných úloh, ve kterých se může uplatnit ILP a DLP. Toto urychlení se provádí na úrovni architektury procesorů s aplikačně specifickými sadami instrukcí. Autor se v práci omezuje pouze na architektury procesorů s aplikačně specifickou sadou instrukcí (ASIP), u kterých lze staticky plánovat jak ILP, tak DLP.

Dále je zde vytýčen cíl práce, kterým je rozšíření stávajícího projektu Lissom o procesorové jádro a sadu potřebných vývojových nástrojů pro rychlou tvorbu aplikací, přičemž primární motivací je zvýšení produktivity práce. Navržený proces návrhu se skládá z následujících kroků: (1) použití šablony procesoru popsaného v jazyce ADL, (2) vygenerování překladače, assembleru a simulátoru pro daný procesor, (3) kompilace a profilování aplikačního kódu; identifikace míst v kódu, pro která je vhodné navrhnout nové aplikačně specifické instrukce a (4) doplnění ADL modelu o popis nově vytvořených instrukcí. Kroky (2) - (4) se opakují, dokud nejsou splněna předem daná kritéria pro danou aplikaci (výkonnost, cena atd.).

Autor se ve své práci zaměřil na chybějící nástroje pro navržený proces návrhu, jedná se o šablonu procesoru, nástroje pro generování kompilátoru, podpora profilování aplikačního kódu, identifikace a generování nových instrukcí a optimalizace architektury procesoru.

Druhá kapitola shrnuje současný stav poznání v dané oblasti. Autor se zde na 43 stranách věnuje nástrojům pro návrh aplikačně specifických procesorů. Uvádí popis jazyka CodAL, který umožňuje jak popis architektury procesorů, tak jejich mikro-architektury pro generování HDL pro syntézu hardware a architektury výpočetních struktur, které lze rozdělit na jazyky pro popis struktury, instrukční sady a jejich kombinace rozšířené pro popis na úrovni RTL.

Dále podrobně popisuje problematiku kompilátorů, které generují kód pro různé cílové architektury. Jedná se o nástroje LLVM, GCC, SUIF, CoSy a některé další. Na základě podrobné analýzy jejich vlastností, dostupnosti a licenčních podmínek vybral autor pro tvorbu kompilátoru platformu LLVM.

Další část je věnována problematice dvou architektur rekonfigurovatelných procesorů – Xtensa (Transilica, Cadence) a ARC (Synopsys), které mají rozšiřitelnou sadu instrukcí. Z uvedené analýzy vyplývá, že přestože návrhové nástroje pro oba procesory nejsou příliš zautomatizovány, jsou tyto architektury komerčně poměrně úspěšné. Pokud se tedy autor ve své práci zaměřuje na zvýšení produktivity práce při návrhu, je zřejmé, že toto může vést i k zajímavému komerčnímu využití.

Jsou zde dále popsány známé techniky pro optimalizaci instrukční sady procesorů. Klasické přístupy využívají schopnosti návrháře identifikovat v kódu aplikace místa, které je výhodné akcelerovat v hardware specializovanou instrukcí. Tento proces vyžaduje velké znalosti a zkušenosti návrháře a je časově velmi náročný, na druhou stranu však umožňuje optimalizovat aplikaci přes delší úseky kódu, což lze automatizovaně jen obtížně. Jsou zde též diskutovány automatizované optimalizační techniky, které buď vedou na NP-úplný problém, nebo řešení aproximují. Jako nejvhodnější se autorovi jeví přístup, který kombinuje automatizované metody a interakci s návrhářem.

Na základě studia současného poznání v dané oblasti a cílů, které si vytýčil v úvodu práce, autor definuje následující dílčí úkoly, které v práci budou dále řešeny: Jedná se o využití jazyků pro popis architektury, ze kterých lze generovat překladač, assembler, simulátor s debuggerem a popis procesoru na úrovni RTL. Pro generování překladače bude využit existující nástroj LLVM. Cílem je též nalézt takové řešení, ve kterém bude použit jeden jazyk, jak pro generování kompilátoru, tak simulátoru. S ohledem na potřeby reálných aplikací se v práci nebudou uvažovat 8bitové architektury procesorů, ale jen 16 a 32bitové. V práci bude navržena taková architektura procesoru, která nebude omezena jen na možnost konfigurace předefinovaných výpočetních jednotek, jak je tomu např. u komerčních produktů (Xtensa či ARC), ale bude dostatečně univerzální a generická, aby vyhověla co nejširšímu spektru aplikací, a přitom natolik jednoduchá, aby se s ní dalo dobře pracovat. Optimalizační proces bude rozdělen na automatizovanou a manuální část. Díky tomu bude možno využít výhod obou přístupů pro efektivní návrh (čas, úsilí) efektivních (výkonnost, příkon, cena) architektur procesorů.

Třetí kapitola se na 37 stranách věnuje způsobu řešení cílů, které byly stanoveny v první kapitole a které byly, na základě studia a analýze stavu poznání, upřesněny ve druhé kapitole.

Extraktor sémantiky (kapitola 3.1.) analyzuje CodAL model, vytváří seznam instrukcí, identifikuje jejich typy, chování a zjišťuje informace potřebné pro alokování registrů. Tento nástroj je navržen tak, aby pracoval rychle a jednoznačně. Instrukční sada je v jazyku CodAL popsána bezkontextovou gramatikou, pomocí které se prování transformace instrukce popsané v jazyku symbolických instrukcí do binární podoby. Nejprve se pomocí bezkontextové gramatiky popisující syntaxi instrukcí, spolu s operandy v registrech, vytvoří sada instrukcí procesoru, která je transformována do reprezentace vhodné pro zjednodušení.

Dále se lokalizují zdroje procesoru (registry se kterými pracují instrukce, pomocné registry, programový čítač a přístupy do paměti), které dané instrukce využívají. Následně se provede zjednodušení kódu s cílem dosáhnout co nejjednoduššího reprezentace popisu sémantiky či chování instrukce. Pro extrakci sémantiky jsou v daném pořadí prováděny LLVM optimalizace a aplikovány původní specializované transformace. Pro potřeby popisu specializovaných operací (např. pro práci v plovoucí řádkové čárce) byly do jazyka CodAL doplněny specializované funkce. Extraktor sémantiky byl autorem vyvíjen řadu let a byl postupně rozšiřován.

Autor navrhl a implementoval postupy při návrhu, které velmi usnadňují a urychlují práci. Jedná se např. o nalezení nesprávně definovaných instrukcí již v počátku procesu návrhu, rozdělení transformací do jednoduchých průchodů, tisk mezikódu, mapování do originálního CodAL kódu pro rychlé nalezení problematických částí kódu, atd. Výsledky dosažené extraktorem sémantiky jsou demonstrovány na 18 architekturách procesorů, z nichž autor 8 vytvořil. Výsledky ukazují, že navržené nástroje a celkový proces návrhu jsou velmi efektivní.

Generátor LLVM překladače pro různé cílové architektury představuje další velkou část práce (kapitola 3.2.). Z CodAL popisu je zde generován překladač jazyka C. Autorův přínos je v celkovém návrhu generátoru, vytvoření řady modelů architektur pro jeho testování, vedení implementačních a plánování vývojových prací.

Další přídavné optimalizace byly provedeny v rámci bakalářských a magisterských prací vedených autorem. Vstupem generátoru je jednak sémantika instrukcí generovaná extraktorem sémantiky, který autor navrhl, dále pak může uživatel systému definovat sémantiku vlastní, zadávat pravidla pro provádění podmíněných instrukcí, plánování vykonání instrukcí a lze též rozšířit či přepsat automaticky generovaný LLVM kód. LLVM kompilátor a generátor kódu byly v rámci autorem vedených bakalářských a magisterských prací rozšířeny o několik výhodných vlastností a optimalizací. Patří mezi ně např. podpora VLIW a SIMD architektur.

Podobně jako v případě extraktoru sémantiky, i zde autor navrhl a implementoval postupy při návrhu s cílem zvýšit produktivitu práce návrháře. Vygenerované výsledky, které byly při testování 13 architektur procesorů dosaženy, ukazují, že navržené nástroje a celkový proces návrhu mají velký přínos. Automatizovaně generované výstupy snesou srovnání (a v některých případech i předčí) s výsledky ručního návrhu, ovšem za nesrovnatelně kratší čas a s menší pravděpodobností chyby. Nespornou výhodou je to, že CodAL model potřebuje pouze jeden popis sémantiky instrukcí a že není třeba modifikovat LLVM kód. Dále je možno během návrhu snadno přidávat či odstraňovat instrukce a tím i rychle prohledávat prostor možných řešení.

V kapitole 3.3. se autor věnuje problematice rekonfigurovatelných architektur procesorů. Autor navrhl několik šablon procesorů včetně jejich mikroarchitektury jako CodAL modely. Jedná se o architektury Codix uRISC, Codix STREAM, Codix RISC, které představují portfolio pro různé aplikační domény. Coda uRISC je jednoduchá architektura, podobná MIPS, která je vhodná pro testovací a výukové účely. 16bitová architektura Codix STREAM je určena pro aplikace, které pracují převážně s datovými toky.

Testovací implementace v FPGA řady Virtex 5 (Xilinx) pracuje na frekvenci 100 MHz a umožňuje zpracovat 18 snímků videa s rozlišením 640x480 pixelů za sekundu. Rozšíření sady instrukcí o specializovanou instrukci pro realizaci Sobelova filtru bylo dosaženo urychlení výpočtu více než 5 x. Codix RISC je 32bitová zřetězená architektura, která byla navržena ve spolupráci s dalšími autory a představuje nejpokročilejší z uvedených architektur. Výsledy implementace daného procesoru na různých FPGA platformách ukazují, že procesor lze v daných čipech efektivně realizovat jak s ohledem na potřebné zdroje, tak na dosahovanou rychlost. Podobně je tomu i v případě implementace ve 40nm technologii TSMC. V porovnání s jinými soft procesorovými jádry vychází Codix RISC srovnatelně, podobně je tomu ve srovnání s architekturami ARMv7, Microblaze a ARC. Architektura byla též dále optimalizována s dobrými výsledky.

Největší výhodou oproti dostupným řešením je ovšem skutečnost, že si uživatel může navrhnout vlastní instrukce, díky kterým pak pro danou aplikaci bude procesor dosahovat výrazně lepších výsledků než konvenční architektury. Krom výáše uvedeného se autor dále věnuje návrhu dalších VLIW architektur (VIX a Codix VLIW).

Kapitola 3.4. se věnuje způsobu identifikace instrukcí, které je vhodné akcelerovat. Na základě studia literatury a vlastních zkušeností s vedením studentských prací na téma automatické identifikace se autor rozhodl, že se bude věnovat spíše automatizaci uživatelem vedeného hledání akcelerovatelných částí kódu, které jsou vhodné při implementaci aplikačně specifických instrukcí. K tomuto účelu navrhl postupy, které byly též implementovány v jím vedené bakalářské práci.

V závěru (kapitola. 4) jsou stručně shrnuty výsledky celé práce.

Formální stránka práce je na standardní úrovni. Rozsah kapitol není úplně vyvážen. Kap. 2 popisující současný stav poznání má 43 stran a kap. 3, která představuje těžiště práce, má stran pouze 37 a závěr shrnující výsledky práce má pouze 1 stranu. Členění na kapitoly není příliš logické, což ztěžuje orientaci v textu. Bylo by dobré rozdělit kapitolu 3 na části – na kapitoly věnované extraktoru syntaxe (3.1.), generování kompilátoru (3.2.) a rekonfigurovatelným procesorovým jádrům (3.3.).

Práce je napsána velmi dobrou a čtivou angličtinou s minimem chyb (chybí některé členy apod.). V práci chybí seznam zkratek, přičemž některé zkratky nejsou v textu před svým uvedením definovány (např. FPGA, ASIC na straně 7). Dále chybí některé odkazy na literaturu (např. Amdahlův zákon, str. 4). Na straně 71 je obrázek 3.8., na který jsem v textu nenalezl odkaz.

Závěr: Výsledkem práce Ing. Adama Husára je významné rozšíření uceleného návrhového systému Lissom pro tvorbu rekonfigurovatelných procesorů s aplikačně specifickým instrukčním souborem.

Stěžejní původní přínos autora a těžiště práce vidím v návrhu a implementaci převážné části nástrojů, které umožňují generování prakticky použitelných aplikačně specifických procesorů, konkrétně se jedná o extraktor sémantiky, generátor překladače a rekonfigurovatelná procesorová jádra.

Z praktického hlediska je tedy důležité, že se autorovi podařilo prokazatelně zrychlit, ve srovnání s existujícími řešeními, návrh aplikačně specifických architektur procesorů.

Původní záměr, tedy vytvoření sady nástrojů pro automatizaci návrhového procesu, které nejen že umožňují efektivní a kvalitní návrh aplikačně specifických procesorů, ale především vedou ke zvýšení produktivity práce návrháře a zkrácení doby návrhu, se autorovi podařilo zcela naplnit.

Je škoda, že v práci nejsou formálněji popsány některé původní autorem navržené transformaci a optimalizace (např. způsobem, jakým je popsána transformace „phi“), které podle mého názoru představují důležitý původní vědecký přínos autora k dané problematice.

V práci by bylo též vhodné souhrnně popsat původní výsledky, kterých autor v rámci celého rozsáhlého projektu dosáhl. Tyto jsou sice v textu průběžně přímo či nepřímo zmiňovány, avšak pro čtenáře, který nezná související projekty a produkty, na kterých autor spolupracoval, je obtížné přesně rozlišit kolektivní dílo od původní práce autora.

Výsledky práce byly průběžně publikovány na dostatečně kvalitním mezinárodním fóru. Autor byl též členem řešitelského týmu 4 výzkumných projektů a je spoluautorem 4 produktů. Velmi kladně hodnotím to, že se dva výsledky práce podařilo patentovat na mezinárodní úrovni.

Na základě předložené práce a dosavadní publikační činnosti konstatuji, že Ing. Adam Husár prokázal schopnost tvůrčí práce. Předložená práce splňuje podmínky kladené na disertační práci, a proto ji doporučuji k obhajobě před komisí.

V Brně dne 18. 12. 2014

Doc. Dr. Ing. Otto Fučík
Ústav počítačových systémů
Fakulta informačních technologií
Vysoké učení technické v Brně

Otázky pro doktoranda:

1. Byl navržený systém použit v praxi?
2. Jaké plánujete další práce v dané oblasti?