EVOLUČNÍ NÁVRH A OPTIMALIZACE APLIKAČNĚ SPECIFICKÝCH MIKROPROGRAMOVÝCH ARCHITEKTUR

Miloš Minařík
DVI4, 2. ročník, prezenční studium
Školitel: Lukáš Sekanina

Fakulta informačních technologií, Vysoké učení technické v Brně
Božetěchova 2, 612 00 Brno

iminarikm@fit.vutbr.cz

Abstrakt. Aplikačně specifické architektury lze v současnosti nalézt v celé řadě různých systémů. K implementaci těchto systému se často využívají mikroprogramové architektury. Návrh takových mikroprogramových architektur je však obecně poměrně složitou úlohou. Tento článek se zabývá možnostmi usnadnění návrhu a optimalizace těchto architektur pomocí evolučních algoritmů. Jsou v něm nastíněna možná využití evoluce v této oblasti, určeny cíle výzkumu a rozebrány možnosti jejich splnění. Na závěr jsou shrnuty dosavadní výsledky výzkumu a jejich zhodnocení.

Klíčová slova. mikroarchitektura, mikroprogram, evoluce, optimalizace

1 Úvod

Masová výroba vestavěných systémů započala v souvislosti s výrobou raket Minuteman vyráběných od roku 1961. Zde se díky velkému objemu produkce podařilo snížit cenu použitého integrovaného obvodu z 1000 dolarů na 3 dolary. Díky tomuto poklesu ceny začalo být výhodné používat integrované obvody v celé řadě dalších výrobků a v současnosti jsou vestavěná řešení základem široké škály aplikací.

V začátcích vývoje vestavěných systémů bylo nutné navrhovat celý systém ručně. Později začalo být díky zvyšující se hustotě integrace prvků na čipu a snižující se ceně výhodné nahrazovat mikrokontroléry obecnými mikroprocesorovými architekturami. Pro návrh vestavěných systémů založených na těchto architekturách existuje v současnosti široká škála nástrojů. Existují však specifické aplikace vestavěných systémů, které použití mikroprocesorových architektur neumožňují například kvůi přísným požadavkům na plochu nebo rychlost. Při návrhu takového systému již často není možné použít obecné postupy jako u mikroprocesorových architektur. Místo toho je nutné navrhovat systém již od počátku s ohledem na jeho konkrétní aplikaci a využívat její specifika. Tento úkol může být pro návrháře velmi obtížný. Je tedy snaha hledat postupy, které by návrh nebo alespoň jeho části usnadnily.

Jednou z takových možností je použití evolučních technik. Jedná se o techniky inspirované evolucí organismů v přírodě. Během posledních let se ukázalo, že evolučně vytvořená řešení mohou v některých oblastech konkurovat řešením vytvořeným člověkem [1]. Pomocí evolučních technik bylo dokonce navrženo i několik patentovatelných vynálezů. Použití těchto technik pro úplný návrh vestavěného systému může být poměrně obtížné, ale lze je využít například pro jeho optimalizaci.

2 Současný stav řešené problematiky

V této části je popsána problematika vysokoúrovňové syntézy obvodů, která je v současnosti jednou z nejvíce používaných metod pro návrh obvodů. Je diskutován současný stav této metody a možné využití v tématu disertační práce. Dále je diskutována problematika evolučních technik včetně výběru použitelných variant.

2.1 Vysokoúrovňová syntéza

Vysokoúrovňová syntéza (High Level Synthesis, HLS) se zabývá převodem popisu systému na vysoké úrovni abstrakce na popis na nižší úrovni při dodržení daných omezení [2]. Počáteční úrovní může být například behaviorální popis systému na systémové nebo algoritmické úrovni. K tomuto popisu se zpravidla využívají procedurální jazyky (například VHDL). Cílem syntézy je obvykle převod systému na popis na úrovni RT. Z této úrovně již lze systém převést pomocí známých postupů. Kromě popisu chování na algoritmické úrovni mohou být součástí zadání také daná omezení, která musí výsledný systém splňovat (například maximální plocha nebo příkon).

Prvním krokem HLS je převod počátečního zadání na vnitřní reprezentaci specifickou pro daný syntetizační nástroj. Většinou se využívá grafová reprezentace obsahující graf datového a řídicího toku. Nad těmito grafy se následně provádí různé vysokoúrovňové transformace, jež mají za cíl jejich optimalizaci ve smyslu rychlosti nebo prostorové náročnosti. Jakmile je vytvořena optimalizovaná vnitřní reprezentace, přichází na řadu plánování a přidělování. Plánování spočívá v rozvržení dílčích operací do jednotlivých kroků, přičemž krok je v synchronních systémech základní jednotkou, která obvykle odpovídá hodinovému taktu. Přidělování slouží k přiřazení hardwarových zdrojů jednotlivým operacím a proměnným. Plánování a přidělování se provádí již s ohledem na platformu, na níž bude výsledný systém založen.

HLS obsahuje i v současné době řadu otevřených problémů, které nejsou v současnosti stále ideálně vyřešeny. To je dáno především komplexností samotné syntézy, neboť je dokázáno, že mnoho dílčích kroků jsou NP úplné problémy. Nalezení optimální obvodové realizace dané specifikace tedy není při současných možnostech výpočetní techniky snadný úkol.

2.2 Evoluční výpočetní techniky

Tyto techniky jsou inspirovány evolucí v přírodě, jak ji popsal Charles Darwin a později teorie neodarwinismu využívající poznatků molekulární genetiky. Využívání těchto technik pro automatizované řešení problémů započalo v 50. letech minulého století a od té doby bylo úspěšně použito v mnoha oblastech výzkumu.

Evoluční techniky pracují s populaci jedinců (kandidátních řešení) vyskytujících se v určitém prostředí. Vliv prostředí na vývoj jedinců je v evolučních technikách zastoupen fitness funkcí, která určuje míru přizpůsobení jedince danému prostředí. Stejně jako v přírodě mají i v umělé evoluci lépe přizpůsobení jedinci vyšší šanci na přežití a rozmnožení. Za předpokladu, že potomci těchto jedinců „zdědí“ vlastnosti svých rodičů zvyšující pravděpodobnost přežití, se bude celá populace postupně zkvalitňovat. V evolučních algoritmech se při vytváření následující generace využívají genetické operátory, které jsou obvykle opět inspirovány přírodou.

Evoluční techniky mají řadu různých variant. Jednotlivé varianty se liší především použitými evolučními operátory, způsobem selekce a velikostí populace. Pro téma disertační práce se jako nejvhodnější jeví genetické algoritmy a genetické programování.

3 disertační práce

3.1 Motivace

V předchozí části byl shrnut aktuální stav v oblasti návrhu obvodů a evolučních technik. Bylo zjištěno, že automatizovaný návrh obvodů je v současnosti poměrně dobře prozkoumaným odvětvím. Přesto však stále existuje celá řada problémů, které nejsou v této oblasti uspokojivě vyřešeny. Z hlediska tématu disertační práce je největším nedostatkem, že HLS se v posledních letech soustřeďuje především na návrh velkých systémů (například víceprocesorových), přičemž na nástroje pro syntézu a optimalizaci menších obvodů (například jednodušších mikroprogramových architektur) je kladen minimální důraz. Stále však existují aplikace, kde je nutné použít aplikačně specifické integrované obvody (ASIC), například kvůli striktním požadavkům na plochu nebo příkon výsledného systému. Typickým použitím těchto architektur je výpočet netriviální funkce nad daty ze senzoru v průmyslových aplikacích.

Další zkoumanou oblastí byly evoluční techniky. Z pohledu disertační práce jsou důležité techniky schopné usnadnit návrh HW a SW mikroprogramové architektury. Z hlediska návrhu digitálních obvodů patří mezi nejvhodnější kandidáty genetické algoritmy a některé varianty genetického programování, například kartézské genetické programování (CGP). Pro optimalizaci parametrů aplikačně specifických architektur lze použít evoluční strategie a genetické algoritmy. Při návrhu SW lze využít GP s určitými úpravami. K optimalizaci parametrů programu lze použít evoluční programování a genetické algoritmy.

V oblasti jednodušších aplikačně specifických mikroprogramových architektur je často nutné provádět složitější výpočty za použití omezených HW prostředků. V tomto případě může být výhodné použít mikroprogramovou architekturu, která umožňuje na základě definovaného mikroprogramu opakovaně využívat existující funkční jednotky. Výhodou tohoto přístupu je, že mikroprogram lze v případě potřeby změnit tak, aby obvod prováděl jinou funkci, kterou lze vyčíslit pomocí dostupných funkčních jednotek.

Požadovanou mikroprogramovou architekturu může být možné navrhnout evolučně. Na základě zkušeností s evolučním návrhem obvodů realizujících iterační algoritmy [3] se však lze domnívat, že celý návrh architektury pomocí evolučních technik je v současné době možný pouze u architektur menšího rozsahu. Při optimalizaci obvodů se však evoluční techniky ukázaly jako poměrně dobře použitelné. S ohledem na tyto skutečnosti lze formulovat pracovní hypotézu následujícím způsobem.

3.1.1 Hypotéza

Pomocí evoluční optimalizace mikroprogramu a mikroarchitektury lze vylepšit zvolené parametry (např. minimalizovat plochu) aplikačně specifického systému menšího rozsahu (typicky realizujícího speciální aritmetické operace)využívajícího mikroprogramovou architekturu vzhledem ke stávajícím řešením používaným v průmyslových aplikacích. Současně nebudou podstatně zhoršeny ostatní parametry.

4 Cíle disertační práce

1. Vytvořit model umožňující specifikovat řešený problém pomocí vstupně-výstupních závislostí, omezujících podmínek a dalších požadavků na funkci výsledné architektury tak, aby možnosti specifikace problému pokrývaly co nejvíce případů použití.
2. Vytvořit simulátor, pomocí nějž lze ověřit správnou funkci mikroprogramové architektury.
3. Vytvořit platformu pro optimalizaci mikroprogramových architektur pomocí evolučních technik.
4. Ověřit funkčnost navržené platformy při optimalizaci průmyslových aplikací.
5. Na základě získaných výsledků vytvořit co nejobecnější metodiku popisující, jak provádět optimalizaci jednotlivých částí systémů využívajících mikroprogramové architektury.

5 Způsob řešení

5.1 Model problému

Specifikaci problému lze rozdělit na dvě části. První částí je specifikace chování mikroprogramové architektury jako celku v rámci systému, v němž je obsažena. Z tohoto pohledu lze na mikroarchitekturu nahlížet jako na černou skříňku, do které lze přivádět vstupy a získávat z ní výstupy. Vztah mezi vstupy a výstupy by mělo být možné určit různými způsoby, například tabulkou hodnot nebo funkčním předpisem.

Druhá část specifikace určuje interní podobu mikroprogramové architektury. Některé obvodové prvky musí být v architektuře obsaženy, jiné mohou být volitelné podle daného použití. V rámci této části specifikace lze určit funkční jednotky (moduly), které mohou být pro implementaci použity, a jejich počet. Tímto způsobem lze výslednou architekturu omezit například tak, aby neobsahovala více než dvě sčítačky. Omezení počtu modulů lze zadat také cenovou funkcí.

5.2 Simulátor

Hlavním požadavkem na simulátor je, aby dokázal simulovat všechny architektury vyhovující zavedenému modelu. Vstupem simulátoru jsou kromě simulované architektury také časové průběhy vstupů, které jsou nutné pro simulaci vnějších podmínek architektury (tedy jejího zapojení ve výsledném systému). Nejdůležitějším výstupem simulátoru jsou bezesporu časové průběhy výstupů, u nichž lze následně ověřit, zda splňují časové specifikace dané v zadání. Pokud se jedná o systém s konečným počtem kombinací vstupů, lze tímto způsobem výslednou architekturu i validovat.

Dalšími výstupy mohou být různé informace usnadňující optimalizaci systému. Mezi tyto informace lze zařadit například údaje o využití jednotlivých modulů během simulace. Na základě těchto údajů lze následně rozhodnout, zda by bylo vhodné do systému přidat nějaký modul nebo jej naopak ze systému odstranit. Kromě toho může simulátor poskytovat také informace sloužící k ladění, protože během optimalizace architektury se mohou vyskytnout problémy, které v původní architektuře nebyly. Například při optimalizaci počtu použitých registrů se může stát, že se dva moduly pokusí zapisovat výsledek do stejného registru. Tyto situace je nutné sledovat a vhodným způsobem na ně reagovat.

5.3 Evoluční platforma

Základem funkčnosti evoluční platformy jsou použité evoluční techniky. Techniky vhodné pro evoluční optimalizaci architektury a mikroprogramu byly popsány v části 2.2. Výběr konkrétních technik bude proveden na základě praktických výsledků jejich použití na jednodušších architekturách.

Jedním z největších problémů, které lze očekávat v souvislosti s optimalizací mikroprogramové architektury, je provázanost architektury s programem. Pokud se hardwarová část nezmění, lze programovou část zpravidla optimalizovat nezávisle. Naopak změny v hardwarové části většinou vyžadují změnu programu. Dalším aspektem evoluční platformy je možnost ovlivnění parametrů dané evoluční techniky. Evoluční platforma by měla umožňovat použití pevně daných hodnot nebo změnu těchto pravděpodobností podle aktuální situace. Toho lze dosáhnout pomocí uživatelem definované funkce, která se bude volat vždy mezi generacemi a nastavovat parametry evoluce.

Z výše popsaných skutečností je zřejmé, že při hledání řešení se může měnit celá řada parametrů architektury. Jedná se tedy o vícekriteriální optimalizaci. Pro řešení vícekriteriálních optimalizací existuje řada metod, např. NSGA–II (Nondominated Sorting Genetic Algorithm) [4]. Pomocí jedné z těchto metod budou nalezena Pareto–optimální řešení, kterých může existovat více. Z těchto řešení bude následně třeba vybrat nejvhodnější řešení pro dané použití systému.

5.4 Ověření funkčnosti

Jakmile bude systém implementován, bude třeba jej ověřit na jendodušších i složitějších obvodech. Zpočátku se bude s největší pravděpodobností jednat především o obvody realizující různé iterační algoritmy. Později budou provedeny pokusy o implementaci architektur, které jsou reálně nasazené v průmyslových aplikacích. Po ověření správné funkčnosti obvodu bude provedeno srovnání s existujícím průmyslovým řešením a výsledky budou vyhodnoceny.

5.5 Vytvoření metodiky

Pokud budou výsledky ověření funkčnosti uspokojivé, bude potvrzena i hypotéza. V tom případě by mělo být možné vytvořit obecnou metodiku pro optimalizaci mikroprogramových architektur pomocí evolučních technik. Tato metodika by měla poskytnout návod podrobně popisující jednotlivé kroky nutné pro úspěšnou optimalizaci. Bude v ní popsán způsob vytvoření modelu architektury, která se má optimalizovat, a nastavení parametrů použitých evolučních technik.

6 Dosavadní výsledky

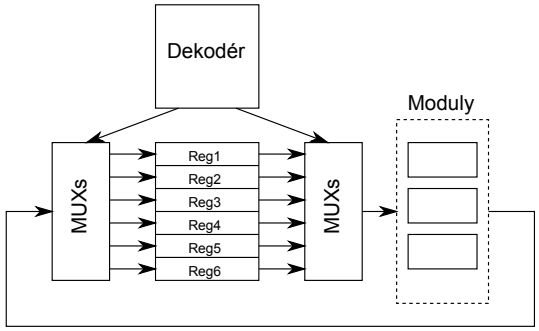
6.1 Evoluční návrh iteračních algoritmů pomocí CGP

Na toto téma byl v rámci doktorského studia publikován konferenční článek zabývající se návrhem iteračních algoritmů pomocí upravené varianty CGP [3]. Úprava CGP spočívala v iterační aplikaci fenotypu s cílem nalézt program aproximující cílovou funkci, která počítá výsledek $i + 1$. iterace pomocí výsledků i. iterace. S takto upravenou variantou CGP bylo provedeno několik experimentů. Během těchto experimentů byly nalezeny například algoritmy pro Newtonovo iterační dělení, Goldschmidtovo dělení a Euklidův algoritmus pro určení největšího společného dělitele. Z experimentů, které nevedly k úspěšnému řešení byl nejvýznamnější evoluční návrh algoritmu CORDIC, kde se ukázalo, že použitá evoluční technika neumožňuje současně vyvíjet více větví výpočtu.

6.2 Model a simulátor

Jak bylo popsáno v části 5.2, simulátor musí být schopen simulovat všechny architektury odpovídající návrhu modelu. Vývoj simulátory tedy silně závisí na vývoji modelu. Tato závislost je však oboustranná, neboť během vývoje simulátoru se mohou vyskytnout problémy, které nelze vyřešit jinak, než změnou modelu architektury. Z toho důvodu probíhá v rámci disertační práce vývoj modelu architektury a simulátoru souběžně a stejným způsobem je popsán v této části. Uvedená architektura vychází z potřeb praxe a představuje typické zadání (ve smyslu složitosti obvodu), kterým se disertační práce bude zabývat. Zjednodušené schéma modelu architektury je znázorněno na obr. 1. K popisu struktury architektury je nutné určit počet registrů, použité moduly, propojení jednotlivých částí, vstupy a výstupy architektury a mikroinstrukční sadu.

Registrů může být v architektuře obsažen libovolný počet, přičemž každý z těchto registrů může mít libovolnou šířku. Při určování modulů použitých v mikroarchitektuře lze využít předdefinované moduly, u nichž lze nastavit bitové šířky vstupů a výstupů. Pokud požadovaný modul neexistuje, lze jej snadno vytvořit jako novou třídu. Po specifikování modulů je nutné určit propojení registrů s jejich vstupy a jejich výstupů s registry. Na tato propojení nejsou kladena žádná omezení, i když určité kombinace mohou při simulaci způsobit konflikty. Z pohledu simulátoru není propojení chápáno jako pole multiplexorů (jímž je zpravidla ve výsledné architektuře realizováno), ale pouze jako předpis určující, jak se mají hodnoty přenášet mezi registry a moduly. Dalšími povinnými parametry architektury jsou specifikace počtu a bitových šířek vstupů a výstupů a mikroinstrukční sady. Mikroinstrukční sada obsahuje mikroinstrukce, které modelovaná architektura dokáže provádět.



Obrázek 1: Schéma modelu architektury.

Po určení výše uvedených parametrů je model hardwarové části mikroprogramové architektury úplný. Dalším krokem je určení programové části, která je tvořena programem složeným z instrukcí definovaných mikroinstrukční sadou. V simulátoru je program realizován lineárním blokem paměti, z nějž se postupně načítají jednotlivé instrukce. Definováním programu je dokončena specifikace modelu architektury. Pro simulaci je však třeba určit prostředí, v němž bude architektura pracovat. To je realizováno definováním časových průběhů vstupů. Tyto průběhy lze určit tabulkou hodnot nebo funkčním předpisem. Jakmile jsou určeny všechny výše uvedené části, lze provést simulaci činnosti architektury. Výsledkem této simulace jsou časové průběhy výstupů. Kromě těchto průběhů jsou k dispozici další informace zjištěné během simulace, například detekované kolize, využití zdrojů architektury atd.

7 Závěr

V tomto dokumentu byla prezentována problematika evolučního návrhu a optimalizace mikroprogramových architektur. Byl popsán současný stav v oblasti automatizovaného návrhu obvodů a využití evolučních technik. Dále byly zformulovány cíle disertační práce a navrženy možné způsoby řešení jednotlivých cílů. Některé z těchto cílů jsou v současné době již splněny nebo rozpracovány. Doposud nebyly zjištěny žádné skutečnosti nasvědčující tomu, že by nebylo možné splnit všechny cíle.

Poděkování

Tento článek byl vypracován v rámci projektů Pokročilé bezpečné, spolehlivé a adaptivní IT (FIT-S-11-1) a Natural Computing on Unconventional Platforms (GAP103/10/1517).

Reference

[1] Koza, J. R.: Human-competitive results produced by genetic programming, Genetic Programming and Evolvable Machines, 2010, Vol. 11, pp. 251–284

[2] Coussy, P., Morawiec, A.: High-Level Synthesis: from Algorithm to Digital Circuit, 2008, ISBN 978-1-40-208587-1

[3] Minarik, M., Sekanina, L.: Evolution of iterative formulas using Cartesian genetic programming, Proceedings of the 15th international conference on Knowledge-based and intelligent information and engineering systems - Volume Part I, 2011, pp. 11–20, LNCS 6881

[4] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation, 2002, Vol 6, pp. 182–197