Výzkum a vývoj aplikací pro Codasip
Souhrnná výzkumná zpráva za rok 2015

Codasip s.r.o.
FIT VUT v Brně

Řešení projektu se skládalo v roce 2015 z několika etap a stav řešení jednotlivých etap je následující.

V první etapě byl vytvořen návrh grafického specifikačního jazyka. Pro podporu návrhu mikroprocesoru byl využit rozšiřující mechanismus profilů. Rovněž byla zvolena množina diagramů, jež umožňuje nejlépe specifikovat požadavky, které jsou kladeny na navrhovaný mikroprocesor. Vybrané diagramy byly rozšířeny o nové prvky (pomocí zmíněných profilů), jež jsou typické pro oblast HW/SW co-designu.

V rámci etapy *Návrh transformace do popisného jazyka* byla rozpracována problematika transformace informací do jazyka pro popis architektury CodAL. V rámci práce byly identifikovány společné rysy pro popis strukturálních a behaviorálních vlastností mikroprocesoru. Tyto společné rysy umožnily provést přímý převod z grafické do textové reprezentace, konkrétně transformace umožnila generovat strukturální model, behaviorální model a hierarchii instrukčních analyzátorů v jazyce CodAL.

V další fázi byl navržen algoritmus pro transformaci informací, jež musely být do jazyka dodány pomocí rozšiřujícího mechanismu (zmíněných profilů). Tento transformační mechanismus umožnil generovat model instrukční sady v jazyce CodAL.

Etapa *Vývoj simulátoru a implementace jednoduchého vývojového prostředí* je rozdělena do dvou oblastí a to do

- oblasti vývoje simulátoru a
- oblasti implementace jednoduchého vývojového prostředí.

Vývoj simulátoru vychází z etapy *Návrh implementace rychlých síťových simulátorů a jejich ladicích nástrojů*. Díky nově vytvořeným principům , bylo možné se zabývat rozšířeními simulátoru, které poskytnou návrháři informace o úzkých místech v navrhované architektuře mikroprocesoru a v aplikačních programech během simulace.

Při návrhu nového typu mikroprocesoru potřebuje návrhář získávat během simulace takzvané *profilovací informace*. Profilovací informace obsahují statistické údaje o tom, která instrukce se volala nejčastěji a pro který zdrojový prvek, kolik taktů se vykonávala daná instrukce, která instrukce nejčastěji způsobovala zastavení linky zřetězení atp.

V rámci probíhajícího výzkumu se pracuje na řešení, kdy je využit formální model simulátoru a vlastní simulátor je rozšířen o prvky, jež umožňují sběr profilovacích informací během simulace. V případě, že je požadována maximální rychlost simulátoru, nedojde k obohacení formálního modelu o tyto prvky a simulátor pak pracuje s maximální možnou rychlostí. Princip, kdy je jádrem softwaru formální struktura, poskytuje větší volnost při specifikaci dodatečných požadavků kladených na software, přičemž nové požadavky jsou velmi rychle v softwaru reflektovány.

Implementace jednoduchého vývojového prostředí je založena na specifikaci, která byla vytvořena v rámci etapy *Návrh jednoduchého tvaru IDE*. Vlastní vývojové prostředí je implementováno pomocí volně dostupné a bez licenčních omezení použitelné platformy *Eclipse*. Prostředí je založeno na takzvaných *perspektivách*, které poskytují uživateli typově podobné funkce (prostředky), pomocí kterých lze provádět určitý druh činnosti. Touto činností je:

- návrh modelu mikroprocesoru a jeho aplikací;
- ladění aplikačních programů na simulátorech.

V současné fázi vývoje jsou implementovány dvě základní perspektivy a to:

- *perspektiva editorů* a
- *ladící perspektiva*.

Perspektiva editorů umožňuje vytváření modelů mikroprocesorů a jejich aplikací v jazyce assembleru, perspektiva dále umožňuje manipulovat s dříve vytvořenými softwarovými nástroji pro překlad aplikačních programů, které byly vytvořeny v rámci dokončených vývojových etap *1a)*

Prototyp některých softwarových nástrojů a *Prototyp některých softwarových nástrojů – druhá generace*.

Ladicí perspektiva umožňuje provádět simulaci a ladění aplikačních programů na dříve vytvořených simulátorech za pomocí navržených ladících nástrojů, jež byly vytvořeny v rámci etap *Prototyp jednoprocesorového interpretujícího simulátoru* a *Návrh implementace rychlých síťových simulátorů a jejich ladicích nástrojů*.

Vytvářené vývojové prostředí je navrhováno a implementováno s ohledem na podporu víceprocesorové simulace a je založeno na takzvané *třívrstvé architektuře*, jež se vyznačuje tím, že většina funkcí které má uživatel k dispozici ve vývojovém prostředí se vykonává na straně serveru. Server rovněž poskytuje víceuživatelský přístup, tedy více uživatelů může prostřednictvím svých integrovaných vývojových prostředí na tomto serveru vyvíjet své nové mikroprocesory a jejich aplikační programy. Toto je zejména výhoda ve větších organizacích, ve kterých bude systém nasazen, neboť není nezbytný nákup nového, dostatečně výkonného hardwaru pro každého vývojáře, ale stačí mít k dispozici pouze jeden dostatečně výkonný server.

Dokončená vývojová etapa *Realizace modelu mikroprocesoru v hardwaru - druhá fáze návrhu* navazovala na etapu *Realizace modelu mikroprocesoru v hardwaru - první fáze návrhu*, ve které byla pro realizaci mikroprocesoru v hardwaru použita *negenerativní technika*. Tato negenerativní technika byla označena jako *princip šablon*.

Další etapa byla naopak založena na *generativní technice*. Tato technika byla založena na formálních popisech jednotlivých částí mikroprocesoru a umožňovala plně automatizovat proces realizace nejkomplexnějších stavebních bloků mikroprocesoru v hardwaru. Těmito stavebními bloky jsou *řadič* a *instrukční dekodéry*. Pro realizaci ostatních částí mikroprocesoru (*funkční jednotky*) se použil dříve zmíněný princip šablon.

Na základě navržené techniky se implementoval generátor hardwaru, jehož vstupem byl model mikroprocesoru v popisném jazyce *CodAL*. Hlavní kritérium, které musel generátor splňovat bylo, že vygenerovaný hardware bude mít ekvivalentní chování se simulátorem, tedy že aplikační program poběží totožně jak v simulátoru, tak i v hardwaru. Dalším kritérium které musel generátor splňovat bylo, že výstup z generátoru je vždy realizovatelný v hardwaru. Tedy pro jakýkoliv vstup musí být výstup generátoru realizovatelný v podobě čipu.

Závěrem můžeme konstatovat, že obě předchozí kritéria byla splněna, což jsme si rovněž mohli ověřit pomocí dostupných komerčních nástrojů a různě složitých navrhovaných mikroprocesorů.

Jak již bylo zmíněno jak generativní tak i negenerativní principy realizace mikroprocesoru v hardwaru musí být vhodně propojeny. Tato propojení nelze realizovat automaticky a musí být vytvořeno ručně návrhářem. Z důvodů možných chyb v manuálně vytvořených propojeních bylo nutné navrhnout prostředky pro validaci návrhářem vytvořených propojení.

Byla proto navržena technika, jež informuje návrháře mikroprocesoru o tom, jak úspěšně se mu tato propojení podařila vytvořit, resp. zda realizovaný kompletní mikroprocesor v hardwaru odpovídá předloze, tedy modelu mikroprocesoru, který byl úspěšně simulován.

Etapy *Návrh konceptu rekonfigurovatelného překladače C* se zabývala analýzou možností vytvoření rekonfigurovatelného překladače C, tedy takového překladače, jež by na základě informací týkajících se architektury mikroprocesoru generoval takový aplikační program z vyššího programovacího jazyka, jež by byl spustitelný na dané architektuře mikroprocesoru.

V první fázi návrhu byl analyzován obecný koncept překladače jazyka C za účelem identifikace částí, jež umožňují provést rekonfiguraci dle informací o architektuře mikroprocesoru. Jako jediná nutná část pro rekonfiguraci byla identifikována zpětná část překladače C. Protože psaní vlastního kompletního překladače je velmi časově náročné (muže trvat i více let), rozhodli jsme se využít existující dostupné platformy, ve které by byla modifikována pouze zmíněná zpětná

část rekonfigurovatelného překladače C (cca 30% existující platformy). V druhé fázi etapy bylo vybráno několik exitující platforem a byla provedena hloubková analýza. Na základě několika kriterií byla nakonec zvolena platforma LLVM. Tato platforma se stala základem pro vytváření prvotního konceptu rekonfigurovatelného překladače C.

Bylo nutné detekovat konkrétní místa v platformě LLVM, která je nutné změnit dle cílové architektury mikroprocesoru. Bylo rovněž nutné určit způsob rekonfigurace platformy (konfigurační soubory či generování zdrojových souborů v jazyce C++, které je nutné překládat). Dále se etapa zabývala vlastním konceptem generátoru, který na základě informací o architektuře vygeneruje specifické zdrojové soubory dle architektury mikroprocesoru pro platformu LLVM. Při vytváření konceptu jsme dospěli k přesvědčení, že v první fází vývoje nebude možné rekonfigurovatelný překladač C plně generovat, tedy budou nutné ruční zásahy do aplikace překladače. Zásahy budou nutné zejména v části optimalizací pro danou architekturu. Z tohoto důvodu byl důraz kladen na poskytnutí jednoduchého rozšiřujícího mechanismu vygenerovaného překladače C, do něhož si může uživatel překladače C vkládat vlastní specifické požadavky.