GENERIC INSTRUCTION SIMULATOR

Stanislav KŘÍŽ, Master Degree Programme (5)
Dept. of Information Systems, FIT, BUT
E-mail: xkrizs00@fit.stud.vutbr.cz

Supervised by: Dr. Dušan Kolář

ABSTRACT

This work deals with automatic generation of simulators for models of microdevices with instruction sets. Source for generator is description of simulated processor writen in special language designed for processor modeling called ISAC. Created simulators are using compiled simulation principle and instruction accurate precision.

1 ÚVOD

S rostoucími požadavky na rychlost vývoje a zlevnění celkových nákladů při výrobě nových procesorů rostou nároky na efektivitu návrhu. Jako reakce na tuto situaci se v posledních letech objevila nová kategorie softwaru, který dokáže podpořit všechny vývojové fáze projektu. Využívá speciální jazyky pro popis procesoru, na jejichž základě se automaticky generují všechny nástroje, které jsou potřeba k ověření funkčnosti architektury a dokonce i k vývoji a testování softwaru pro fyzicky zatím neexistující procesory. Jedním z nejdůležitějších generovaných nástrojů je rychlý instrukční simulátor, kterým se zabývá tato práce.

2 ROZBOR

Simulátor je v zásadě složený ze dvou vzájemně na sobě závislých modulů. Jedním z nich je popis zdrojů architektury zahrnující různé registry, paměti, cache, mapování, apod. a druhým je popis chování (instrukcí), které modifikují zdroje a tím vytvářejí výstupy. Tyto části jsou propojené logikou simulátoru, která kromě efektivního modelování chování umožňuje i předávání informaci a statistik o stavu běžícího programu.

2.1 ZDROJE PROCESORU

Zdroji procesoru se rozumí především paměťové prvky (registry a jejich soubory, operační paměť RAM, vyrovnávací paměť cache) ale i další funkce, které ovlivňují práci s hardwarem (mapování paměti, pipeline). Prostředky pro modelování simulátoru jsou omezené schopnostmi jazyka pro hostitelský systém (architekturu, na které se bude simulace provádět). V této práci je cílovým jazykem C++.

Prostředky jazyka umožňují s výhodou použití tříd pro jednotlivé paměťové prvky. Základním stavebním kamenem zdrojů je třída uchovávající jednu nejmenší nedělitelnou hodnotu procesoru (obvykle závislé na bitové délce slova). Kromě skladování dat se třída stará také o vedení statistik o přístupech k hodnotě. Toho je dosaženo přetížením matematických a dalších operátorů jazyka tak aby zachovávali omezení kladená na data (například bitovou šířku) a zároveň korektně modifikovaly statistiky.

Takováto třída může potom být základem pro všechny další paměťové prvky. Jednoduchý registr lze reprezentovat přímo jí i bez speciálních úprav. Soubor registrů a zároveň s ním i operační paměť potom odpovídá poli těchto tříd doplněnému o metody pro získávání hromadných statistik (zároveň pro celé nebo část pole). Větší odlišností se vyznačuje až vyrovnávací paměť cache. Třída, která ji reprezentuje je taktéž složená z takového pole, ale zásadně se liší způsob přístupu k hodnotě. Musí přetěžovat operátory přístupu k datům (nejčastěji [ ]) tak, aby správně modifikovaly i asociovaný zdroj a zachovávaly pravidla pro chování vyrovnávacích pamětí stanovených modelem.

Mapování paměti se odlišuje především absencí konkrétního prostoru pro uchování dat. Operátor přístupu k datům implementuje přepočet indexu v mapovaném prostoru na index do konkrétního paměťového prvku.

Generování popisu zdrojů využívá jako vstup XML popis modelované architektury. Ten je zpracován parserem a na základě získaných údajů se vytvářejí jednotlivé objekty výše popsaných tříd. Speciálním případem je vytváření třídy pro mapování paměti. Vzhledem k tomu, že nelze předem nadefinovat obecnou třídu mapy, je nejen vytvořen objekt, ale zároveň je vygenerována i celá jeho třída.

2.2 CHOVÁNÍ PROCESORU

Jako způsob simulace chování procesoru pro tuto práci byla zvoleno plně kompilovaná metoda. Vyznačuje se největší efektivností pokud jde o výkonnost, ale je omezená především z hlediska vykonávání programů s dynamicky se měnícím kódem.

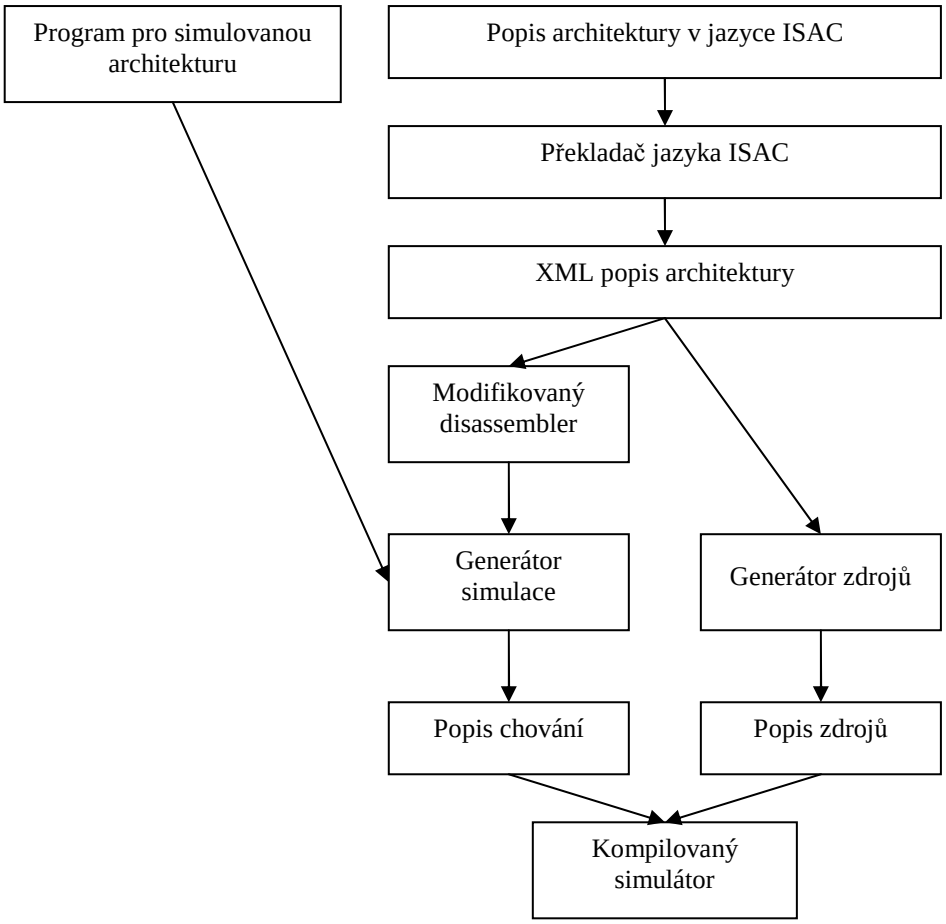
Na rozdíl od pomalejší interpretované kompilace (každá instrukce je opakovaně načítána a dekódována) vytváří kompilovaný simulátor kompletně nový simuláční program, ve kterém je každá instance instrukce simulovaného programu převedena na odpovídající kód cílové platformy. Tím odpadá prodleva při načítání a dekódování. Vedlejším efektem je větší časová náročnost generování simulátoru, která ale ve většině případů nevadí.

Celý proces generování lze rozdělit do tří fází. V první fázi se používá částečně modifikovaný generátor disassembleru k vytvoření programu, který ze vstupního XML souboru popisujícího architekturu, vytváří aplikaci pro dekódování instrukce architektury. Výsledkem je program, který přijme jako vstup binární soubor (program simulované architektury) a na výstupu generuje soubor s funkcí v C++, která plně reprezentuje chování. Tím je ukončena druhá fáze a jako poslední krok dojde k propojení tohoto souboru s vygenerovaným popisem zdrojů a hlavním souborem simulátoru. Tento program po přeložení provádí kompilovanou simulaci programu modelované architektury na cílovém (hostitelském) počítači.

2.3 STRUKTURA SIMULÁTORU

Základním požadavkem simulátoru je vysoká výkonnost a schopnost poskytovat run-time informace o průběhu simulace. Z toho důvodu je nutné efektivně navrhnout interaktivní

simulátor. Program je proto rozdělený do dvou paralelně probíhajících vláken. První vykonává simulační smyčku (část vygenerovaná v rámci analýzy chování programu) a druhá zpracovává externí příkazy. Styčnou plochou obou vláken jsou definované zdroje procesoru. Pomocí příkazů pro simulátor lze za běhu modifikovat data (ale z principu kompilované simulace ne program), získávat mezivýsledky, upravovat stav procesoru (pozastaven, běžící, apod.) a monitorovat statistiky.



Obr. 1: *Schéma jednotlivých stupňů tvorby kompilovaného simulátoru*

3 ZÁVĚR

Hlavní obsah zbývající práce na simulátoru spočívá v prohloubení přesnosti simulace (zahrnutí dalších parametrů, které poskytuje modelovací jazyk) a optimalizaci navržených tříd a funkcí. Další vývoj práce by měl směřovat k úpravě simulace z instruction-accurate úrovně na přesnější cycle-accurate úroveň, která poskytuje detailní informace o časování instrukcí a umožňuje kvalitně nasimulovat instrukční paralelismus.

LITERATURA

[1] Hruška, T. Návrh jazyka ISAC 0.0. Brno: Interní materiál FIT VUT Brno, 2004.

[2] Hoffmann, A., Meyr, H., Leupers, R. Architecture exploration for embedded processors with LISA. Boston: Kluwer Academic Publishers, 2002.