TRANSFORMATION FROM C TO VHDL LANGUAGE

Martin Mecera
Master Degree Programme (2), FIT BUT
E-mail: xmecer00@stud.fit.vutbr.cz

Supervised by: Karel Masařík
E-mail: masarik@fit.vutbr.cz

ABSTRACT

This paper describes the process of transformation of the behavior of processor described in C language into VHDL language. Differencies between common compilers and hardware compilers are discussed. The need for automatized circuit verification is emphasized. Algorithms of hardware-specific optimizations (especially tree-height reduction and planning) are explained. Transformation from C to VHDL is an example of High Level Synthesis.

1. ÚVOD

Mikroprocesory lze navrhovat v různých jazycích. Nejčastěji je návrh realizován v jazycích pro popis hardwaru (např. VHDL nebo Verilog). Nevýhodami jazyků pro popis hardwaru jsou zejména obtížné odhalování chyb, nízká úroveň popisu, vysoké nároky na znalosti a zkušenosti návrháře. Další skupinou jazyků pro návrh mikroprocesorů jsou smíšené jazyky pro popis architektury. Mezi tyto jazyky patří jazyk ISAC. Jazyk ISAC kombinuje deklarativní popis struktury mikroprocesoru s procedurálním popisem chování mikroprocesoru (v jazyku C). Z popisu v jazyku ISAC je vygenerován popis v jazyku VHDL a rychlý simulátor ve vyšším programovacím jazyku (např. C). Simulátor koresponduje s modelem mikroprocesoru v jazyku VHDL. Díky této korespondenci lze efektivně odhalovat chyby v návrhu mikroproprocesoru. Výhodou jazyka C je to, že odstiňuje návrháře od nízkoúrovňového popisu hardwaru. Vyšší nároky jsou kladeny na algoritmus transformace místo na znalosti návrháře. Tato práce se zabývá generováním modelu mikroprocesoru v jazyku VHDL z popisu chování mikroprocesoru v jazyku C resp. z jazyku ISAC.

2. PROCES TRANSFORMACE

Proces transformace z jazyku C do jazyku VHDL sdílí společné rysy s překladem vyšších programovacích jazyků do jazyků symbolických instrukcí (assemblerů). Překladač C do VHDL má přední část, která převádí zdrojový kód do vnitřní reprezentace, vnitřní část určenou k optimalizacím a zadní část, která generuje objektovou reprezentaci. Z objektové reprezentace lze snadno generovat cílový kód ve VHDL. Oproti běžným překladačům vyšších prog. jazyků se liší zejména ve vnitřní reprezentaci kódu, množině optimalizací a výstupech zadní části.

Procesor vygenerovaný z jazyku ISAC musí být co nejrychlejší. Vysoké rychlosti je dosaženo maximální možnou paralelizací výpočtů. Významná část překladače C do VHDL se věnuje odhalování paralelismu uvnitř výpočtů a plánování výpočtů do vhodných výpočetních jednotek. Za účelem odhalení datového paralelismu je zvolena vnitřní reprezentace pomocí grafu. Datový paralelismus lze modelovat pomocí grafu datových toků. Řízení toku výpočtu je popsáno pomocí grafu řízení výpočtu. Vnitřní reprezentace kombinuje datový paralelismus na úrovni základních bloků s grafem řízení výpočtu. Tento kombinovaný typ popisu se nazývá graf řídících a datových toků *(control/data-flow graph)* [1].

Procesor vytvořený z modelu ve VHDL nelze po syntéze změnit. Velký důraz je proto kladen na ověření funkčnosti procesoru před jeho nasazením na trh. Otestování procesoru se provádí tak, že na jeho vstup jsou přiváděny vstupní hodnoty, pro které jsou známy správné výstupní hodnoty. Porovnáním na shodu jsou odhaleny chyby. Správné výstupní hodnoty lze automatizovaně vypočítat díky tomu, že chování procesoru je popsáno v jazyku C a pro jazyk C existují překladače do jazyku symbolických instrukcí (assembleru) osobního počítače. Program v assembleru na osobním počítači vrátí soubor se správnými výstupy.

3. OPTIMALIZACE

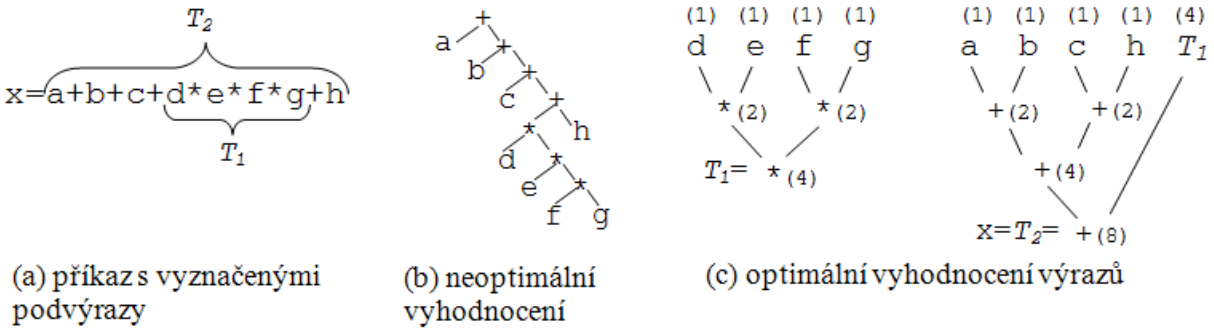
Optimalizace se týká různých cílů, mnohdy protichůdných. Procesory bývají optimalizovány na rychlost, velikost čipu, spotřebu, bezpečnost, spolehlivost aj. Priority optimalizace se časem mění. Rychlost výpočtu byla vždy prioritou číslo jedna, avšak významně ovlivňována byla velikostí čipu, tedy cenou. S klesající cenou komponent v posledních letech toto omezení slábne. Do popředí zájmu se dostávají energetické cíle. Zvláštní kapitolu optimalizací tvoří doménově specifické optimalizace např. na bezpečnost či spolehlivost. Transformace z vyšších programovacích jazyků má potenciál k tomu, aby podle cílů návrháře procesoru automaticky upravovala výsledný model, aniž by se musel změnit kód ve vyšším prog. jazyce. Tato práce se zaměřuje na tradiční cíl optimalizace – rychlost výpočtu při využití co nejmenšího počtu výpočetních jednotek v obvodu (např. ALU).

3.1. REDUKCE VÝŠKY STROMU VÝRAZŮ

Výpočet matematických výrazů může být značně neefektivní, pokud je prováděn sekvenčně. Redukce výšky stromu výrazů provádí vhodné uzávorkování nad výrazy tak, aby jejich podvýrazy mohly být vyhodnoceny paralelně. Například pravé straně příkazu `x=a+b+c+d*e*f*g+h` by odpovídal efektivní paralelní výpočet při uzávorkování `x=((a+b)+(c+h))+((d*e)*(f*g))`. Přední části překladačů obvykle vracejí neefektivní (sekvenční) variantu. Algoritmy redukce výšky stromu výrazů využívají asociativity, komutativity a distributivity operátorů k přeskládání a uzávorkování výrazů. Optimální algoritmus redukce výšky stromu je postaven na modifikaci Huffmanova kódovacího stromu[2]. Na obrázku 1 je uvedeno srovnání efektivního a neefektivního vyhodnocení výrazu.

Algoritmus pracuje takto:

1. přiřadí všem operandům váhu 1,
2. nalezne podvýrazy tvořené stejnými operátory,
3. vyváží podstromy pro komutativní a asociativní operátory tak, že slučuje operandy s nejnižší váhou a váhy sčítá.



(a) příkaz s vyznačenými podvýrazy

(b) neoptimální vyhodnocení

(c) optimální vyhodnocení výrazů

Obrázek 1: Redukce výšky stromu

3.2. PLÁNOVÁNÍ OPERACÍ

Cílem plánování je rozložení operací (např. sčítání nebo násobení) na odpovídající obvody (ALU, násobičky aj.) tak, aby výpočet proběhl co nejdříve a neplýtvalo se prostředky. Je žádoucí, aby byly obvody znovupoužity, čímž se sníží velikost čipu a zároveň cena. Algoritmy plánování obvykle začínají všechny stejně: pro každou operaci vypočítají, kdy nejdříve ji lze vykonat s ohledem na datové závislosti (tzv. algoritmus ASAP[1]). Tímto prvním krokem je zjištěn nejkratší možný čas výpočtu, který ovšem není citlivý k výpočetním prostředkům. Algoritmy se následně liší v tom, jakým způsobem vypočítají minimální počet potřebných výpočetních prostředků, aby algoritmus proběhl v tomto čase. Jedná se obecně o těžký problém. Existují algoritmy, které naleznou optimální řešení za cenu delšího výpočtu, a algoritmy hledající suboptimální řešení pomocí heuristiky. Hledání optimálního řešení je v této práci věnována velká pozornost. Vhodný algoritmus optimálního plánování je založený na lineárním programování[3]. Úkolem algoritmu je převézt graf datových závislostí operací na soustavu lineárních nerovnic. Nástroj pro řešení problému lineárního programování, např. GLPK, vyřeší soustavu lin. nerovnic tak, aby byla minimalizována cena prostředků. Kombinování více optimalizací lze řešit rozšiřováním soustavy nerovnic o další nerovnice.

4. ZÁVĚR

Práce nastínila cíle a základní postupy při transformaci z jazyka C do VHDL. Transformace z vyšších programovacích jazyků do jazyků pro popis hardware je obor, který se rychle vyvíjí. Mezi přední oblasti výzkumu patří optimalizace a plánování. Významný směr výzkumu v oboru vysokoúrovňové syntézy *(High Level Synthesis)* je zaměřen na vývoj nástrojů, které návrhářům umožní optimalizovat systémy pro více cílů najednou. Doba návrhu systému je díky těmto moderním nástrojům několikanásobně kratší než tradiční postupy. Kvalita výsledných procesorů je zaručena pomocí verifikace.

LITERATURA

[1] Gajski, D.D., Dutt , N., Wu A., Lin S.: High-Level Synthesis. Kluwer, Boston, 1992.

[2] Coons, K a kol.: Optimal Huffman Tree-Height Reduction for Instruction-Level Parallelism, dostupné on-line: <http://www.cs.utexas.edu/users/mckinley/380C/lecs/treeheight-tr-2008.pdf>.

[3] J.-H. Lee, Y.-C. Hsu and Y.-L. Lin, A New Integer Linear Programming Formulation for the Scheduling Problem in Data Path Synthesis, ICCAD, 1989.