GENERATING PROPER VLIW ASSEMBLER CODE USING SCATTERED CONTEXT GRAMMARS

Jakub Křoustek, Stanislav Židek
Doctoral Degree Programme (1), FIT BUT
E-mail: {ikroustek | izidek}@fit.vutbr.cz

Supervised by: Dušan Kolář, Alexander Meduna
E-mail: {kolar | meduna}@fit.vutbr.cz

ABSTRACT

The very long instruction word (VLIW) processor architecture is focused on a high instruction level parallelism. Program execution is scheduled statically at compilation time. Therefore, there is no need for run-time scheduling and dependency checking mechanisms. On the other hand, all these constraints must be controlled by the compiler. This paper describes usage of scattered context grammars in order to model instruction level limitations of these processors. Resulting grammar generates proper assembler code. This concept has two advantages – formal description of the dependency checking process and high reduction of production rules over other methods.

1 INTRODUCTION

The idea of the very long instruction word (VLIW) processor architecture is almost 30 years old. VLIW architecture has been very popular in last decades because of its innovation techniques in parallel computing. Today, the digital signal processors (DSP) are typical representatives of VLIW architecture.

Primary advantage of this architecture is quite simple, but powerful hardware that provides high instruction level parallelism. VLIW processors can work quickly and smoothly because there is no need for conflict checking or planning at run-time. This is also the main disadvantage – scheduling of parallel computation must be planned statically at program compilation time. Therefore, compilers must be very sophisticated. They have to control constraints of parallel operations and operation latencies. One of their key tasks is to avoid read and write conflicts of operations. Control functions are often handwritten by compiler developers and must be executed multiple times in order to check all constraints.

Therefore, it would be handy to have a formal method for constraints checking on assembler grammar level. We propose a grammar that assures correctness of generated program by itself.

2 VLIW ARCHITECTURE OVERVIEW

VLIW processors consist of clusters with functional units [1]. Each functional unit fulfills a different role, typically arithmetic logic unit (ALU), multiplier, unit for memory access, etc.

Activity of every unit is managed by RISC-like operations.

Most of modern processors are pipelined. However, not all functional units have to be pipelined, for example that is a case of expensive functional units like dividers. Therefore, not every unit can handle one operation per cycle. Number of clock cycles needed for operation execution is called *latency*. In general, latency of operations may significantly differ – integer addition may often be done in just one cycle, while floating point division can take up to tens of cycles.

2.1 CONSTRAINTS

The hardware complexity is reduced since execution is statically scheduled by the compiler, but the architecture is bounded by constrains. Compilers for VLIW architecture have to take care of all these constraints, mainly correctness of instruction sequences, which means building instructions with appropriate operations. Compiler must also ensure that operations in the same instruction will be independent and operation latencies are taken into account. Data or control conflicts, like register read and write conflicts, are typical dependencies between two operations.

3 DEFINITIONS

Basic knowledge of formal language theory is expected (see, for instance [3, 4]).

Definition 1: A context-free grammar (CFG) is a quadruple $G = (V, T, P, S)$, where V is a total alphabet, $T \subset V$ is the set of terminals, $S \in V \setminus T$ is the start symbol of G and P is a finite set of rules of the form $A \to x, A \in V \setminus T, x \in V^*$.

Definition 2: A scattered context grammar (SCG) is a quadruple $G = (V, T, P, S)$, where V is a total alphabet, $T \subset V$ is the set of terminals, $S \in V \setminus T$ is the start symbol of G and P is a finite set of rules of the form $(A_1, \ldots, A_n) \to (x_1, \ldots, x_n), n \geq 1, \forall A_i : A_i \in V \setminus T, \forall x_i : x_i \in V^*$.

Definition 3: Let $G = (V, T, P, S)$ be a SCG,

$$y = u_1 A_1 u_2 \ldots u_n A_n u_{n+1},$$
$$z = u_1 x_1 u_2 \ldots u_n x_n u_{n+1},$$

$y, z \in V^*$, $p = (A_1, \ldots, A_n) \to (x_1, \ldots, x_n) \in P$. Then y directly derives z according to the rule p, $y \Rightarrow_G z$ $[p]$ (or simply $y \Rightarrow_G z$). Let $\Rightarrow^+$ and $\Rightarrow^*$ denote transitive and reflexive-transitive closure of $\Rightarrow$, respectively.

Definition 4: Let $G = (V, T, P, S)$ be a SCG. Language generated by G is denoted by $L(G)$ and defined as $L(G) = \{w : w \in T^*, S \Rightarrow^* w\}$.

We can see that an application of scattered context rule simulates application of several context free rules in parallel.

4 SCATTERED CONTEXT GRAMMAR AS CONSTRAINTS MODELLER

Assume we want to model one elementary constraint in each instruction – register write conflict. A number of instruction bits is finite, therefore, number of all allowed instruction combinations is also finite. Nevertheless, even for very simple hypothetical VLIW processor (see section 4.2) there are more than 52 million allowed combinations. If we want to model the latency

constraints, the number of combinations grows even more. As we can see, constraint modelling by legal combination enumeration is not possible. Therefore, we need another, formal solution, such as scattered context grammars. Their exploitation in similar situations is presented in [2].

4.1 SCG GENERATING PROPER VLIW ASSEMBLER CODE

In this section, we present an algorithm for construction SCG generating proper VLIW assembler code.

Input Specification of the VLIW architecture:

- Let n be a number of computational units. Let $(u_1, \ldots, u_n) : u_1, \ldots, u_n \in U$ be an n-tuple of computational units respecting order of VLIW instruction parts.
- Let Reg be a set of registers $Reg = \{r_1, \ldots, r_{|Reg|}\}$.
- Let I be a set of operations, $\mathrm{nop} \in I$. Let comp be a relation $\mathrm{comp} \subseteq U \times I$, $(u, i) \in \mathrm{comp}$ iff computational unit u can compute operation i. ($\forall u \in U : (u, \mathrm{nop}) \in \mathrm{comp}$)
- Let params be a function $\mathrm{params} : I \rightarrow 2^{\{R,W\}^*}$, such that $\mathrm{params}(i)$ specifies all formats of operation parameters: W for a register to be written and R for a register to be read. ($\mathrm{params}(\mathrm{nop}) = \{\varepsilon\}$)
- Let lat be a function $\mathrm{lat} : I \rightarrow \mathbb{N}$ such that $\mathrm{lat}(i)$ specifies the latency of operation i (how many cycles it takes to compute). ($\mathrm{lat}(nop) = 1$)

Output A SCG generating VLIW assembler code that respects latencies and prevents write conflicts

Method Construct V and T as follows:

- Initially, let $T = Reg \cup I \cup \{';'\}$, $N = \{S\}$.
- For every computational unit u add U_u and L_u to N. Let $U_{Lat} = \emptyset$. Let $\mathrm{ml}(u)$ be a maximal latency of operation that can be computed by unit u. For every computational unit u add $1_u, \ldots, \mathrm{ml}(u)_u$, to U_{Lat}. Add contents of U_{Lat} to N.
- Add $\$, \#, R$, and W to N.
- Let $X = \{x : x \subseteq Reg, |x| < n\}$. For every $x \in X$ add $@_x$ to N.
- Let $V = N \cup T$.

Construct P as follows:

- Initially, let $P = \{S \rightarrow \varepsilon, S \rightarrow @_\emptyset U_{u_1} \ldots U_{u_n} \$ L_{u_1} \ldots L_{u_N} \#\}$.
- For every $r \in Reg$ add $R \rightarrow r$ to P.

- Let nlat be a partial function $\mathrm{nlat} : U \times I \to U_{Lat} \cup U$ such that if $(u, i) \in \mathrm{comp}$:

$$\mathrm{nlat}(u,i) = \begin{cases} \langle \mathrm{lat}(i) \rangle_u & \text{if } \mathrm{lat}(i) > 1 \\ U_u & \text{otherwise} \end{cases}$$

  For every operation format $f \in \mathrm{params}(i)$ and computational unit u such that $(u, i) \in \mathrm{comp}$ add rule $(U_u, L_u) \to (i\ f, \mathrm{nlat}(u, i))$.

- Let A be an integer and decr be a function $\mathrm{decr} : U_{Lat} \to U_{Lat} \cup U$ such that:

$$\mathrm{decr}(\langle A \rangle_u) = \begin{cases} \langle A-1 \rangle_u & \text{if } A > 1 \\ U_u & \text{otherwise} \end{cases}$$

  For every $\langle A \rangle_u \in N$ add rule $(\langle A \rangle_u, L_u) \to (\mathrm{nop}, \mathrm{decr}(\langle A \rangle_u))$.

- Add special rules to handle write conflicts to P. For every $x \in X, r \in Reg$:
  - if $|x| < n - 1$ and $r \notin x$, add $(@_x, W) \to (\varepsilon, r@_{x \cup \{r\}})$ to P
  - if $|x| = n - 1$ and $r \notin x$, add $(@_x, W) \to (\varepsilon, r@_{\emptyset})$ to P

- Add rule generating next instruction to P:
  $\forall x \in X$ add $(@_x, \$, \#) \to (\varepsilon, ';', @_{\emptyset}, \$L_{u1} \ldots L_{un} \#)$.

- Add rules that finish generation to P:
  $\forall \langle A \rangle_u \in V \setminus T$ add $(\$, \langle A \rangle_u) \to (\$, \varepsilon)$
  $\forall u \in U$ add $(\$, U_u) \to (\$, \varepsilon)$
  $\forall x \in X$ add $(@_x, \$, \#) \to (\varepsilon, ';', \varepsilon)$

4.2 EXAMPLE

Imagine VLIW processor with eight registers ($r1$, …, $r8$), three functional units (A, B, C) and five operations ($\mathrm{op1}$, …, $\mathrm{op5}$). Latencies of operations are 1-1-2-2-3. First two operations could be executed in functional unit A, third operation in B and the rest in C. Each of the first four operations read operands from registers and writes result to a register specified as its first argument. Fifth operation only reads from two registers. With the previously defined algorithm, we construct scattered context grammar for generation of proper assembler code for this processor. Derivation of simple code with one instruction may proceed as follows:

$$\begin{array}{lllllll}
S & \Rightarrow & & & & @_{\emptyset}\ U_A\, U_B\, U_C & \$\ L_A\, L_B\, L_C\ \# \\
 & \Rightarrow^* & & @_{\emptyset}\ \mathrm{op1}\ W\ R\, R\ \mathrm{op3}\ W\ R\, R\ \mathrm{op4}\ W\ R & & & \$\ U_A \langle 2 \rangle_B \langle 3 \rangle_C\ \# \\
 & \Rightarrow & \mathrm{op1} & @_{\emptyset}\ W\ R\, R\ \mathrm{op3}\ W\ R\, R\ \mathrm{op4}\ W\ R & & & \$\ U_A \langle 2 \rangle_B \langle 3 \rangle_C\ \# \\
 & \Rightarrow & \mathrm{op1\ r1} & @_{\{r1\}}\ R\, R\ \mathrm{op3}\ W\ R\, R\ \mathrm{op4}\ W\ R & & & \$\ U_A \langle 2 \rangle_B \langle 3 \rangle_C\ \# \\
 & \Rightarrow^* & \mathrm{op1\ r1\ r2\ r3} & \mathrm{op3} & @_{\{r1\}}\ W\ R\, R\ \mathrm{op4}\ W\ R & & \$\ U_A \langle 2 \rangle_B \langle 3 \rangle_C\ \# \\
 & \Rightarrow & \mathrm{op1\ r1\ r2\ r3} & \mathrm{op3\ r4} & @_{\{r1,r4\}}\ R\, R\ \mathrm{op4}\ W\ R & & \$\ U_A \langle 2 \rangle_B \langle 3 \rangle_C\ \# \\
 & \Rightarrow^* & \mathrm{op1\ r1\ r2\ r3} & \mathrm{op3\ r4\ r5\ r6} & \mathrm{op4} & @_{\{r1,r4\}}\ W\ R & \$\ U_A \langle 2 \rangle_B \langle 3 \rangle_C\ \# \\
 & \Rightarrow^* & \mathrm{op1\ r1\ r2\ r3} & \mathrm{op3\ r4\ r5\ r6} & \mathrm{op4\ r7} & @_{\emptyset}\ R & \$\ U_A \langle 2 \rangle_B \langle 3 \rangle_C\ \# \\
 & \Rightarrow^* & \mathrm{op1\ r1\ r2\ r3} & \mathrm{op3\ r4\ r5\ r6} & \mathrm{op4\ r7\ r8} & @_{\emptyset} & \$\ U_A \langle 2 \rangle_B \langle 3 \rangle_C\ \# \\
 & \Rightarrow^* & \mathrm{op1\ r1\ r2\ r3} & \mathrm{op3\ r4\ r5\ r6} & \mathrm{op4\ r7\ r8} & & ;\ ;
\end{array}$$

The last derivation step can be changed in to generation of next instruction. As we see it is not possible to generate a sentence (i.e., program) with write conflicts or incorrect latency.

Table 1 shows comparison of several types of grammars that can be used to define the language of proper assembler programs (with and without latency checking). The number of rules needed by scattered context grammar is significantly lower than others. Note: the number of rules in right regular grammar and right linear grammar is the same. This is because of assembler syntax chosen for this simple example. Generally, the number of rules in right linear grammar is lower.

Architecture	Pipelined	Not-pipelined
Grammar type	Rules	
Right Regular	1165	4788
Right Linear	1165	4788
Context Free	449	1848
Scattered Context	324	333

Table 1: Assembler generators for simple VLIW processor.

5 CONCLUSIONS

High performance of VLIW processor architecture has also its drawbacks. Constraints checking of programs for VLIW architecture has to be done statically at compilation time, which means register checking of write conflicts, keeping of operation latency, etc.

As shown in this paper, scattered context grammars are effective tools for modelling these constraints. Number of rules needed for generation of proper assembler code is significantly lower than the number needed by other types of grammars.

Further research of this topic is necessary, especially utilization of scattered context grammars by compiler schedulers. With this concept we will be able to automate code generation process.

This work was supported by the research funding MPO ČR, No. FT-TA3/128 – Language and Development Environment for Microprocessor Design, FR-TI1/038 - System for Programming and Realization of Embedded Systems, BUT FIT grant FIT-S-10-2 and by the Research Plan No. MSM, 0021630528 – Security-Oriented Research in Information Technology.

REFERENCES

[1] Fisher, J. A., Faraboschi, P., Young, C.: Embedded Computing – A VLIW Approach to Architecture, Compilers, and Tools. Morgan-Kaufmann Elsevier Publishers, 2005, pp. 712, ISBN 978-1-55860-766-8.

[2] Kolář, D.: Exploitation of Scattered Context Grammars to Model Constraints between Components, In: Proceedings of 31st Autumn International Colloqium ASIS 2009, Advanced Simulation of Systems, Ostrava, CZ, MARQ, 2009, pp. 13-18, ISBN 978-80-86840-47-5.

[3] Meduna, A.: Automata and Languages: Theory and Applications, London, GB, Springer, 2005, pp. 892, ISBN 1-85233-074-0.

[4] Meduna, A., Techet, J.: Scattered Context Grammars and their Applications, WIT Press, GB, WIT, 2009, pp. 217, ISBN 978-1-84564-426-0.