TID, November 25, 2009

SCATTERED CONTEXT GRAMMARS & VLIW ARCHITECTURE MODELING

Jakub Křoustek
ikroustek@fit.vutbr.cz

Motivation

- PhD thesis: Debugging Tools for Optimized Code of VLIW Architectures (doc. Kolář)
- Applied research – Exploitation of SCG in VLIW Architecture Modeling
  1. Generator of Proper VLIW Assembler Code
  2. VLIW assembler code analysis (parsing)
  3. New techniques for instruction scheduling
- Co-author: Stanislav Židek

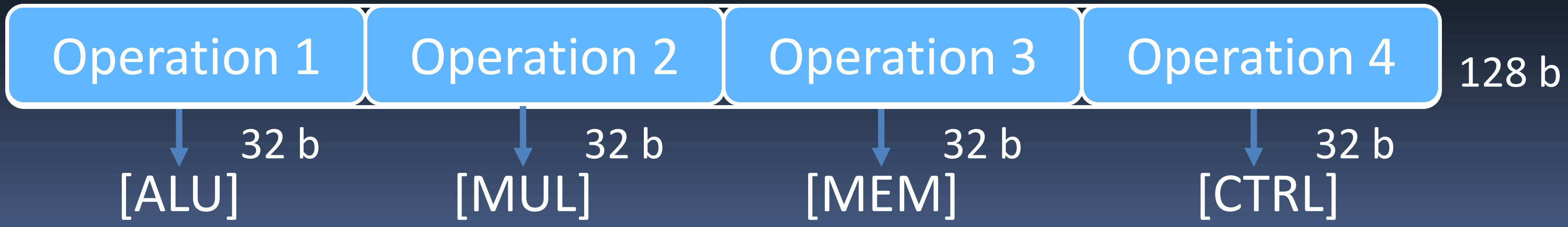
Contents

PART 1

INTRODUCTION

VLIW Architecture Overview

- Very Long Instruction Word
- 1980 (Josh Fisher)
- Instruction Level Parallelism
- Control of many (10+) function units in every cycle
  - ALU, MEMORY, BRANCH UNIT, MMU …
- Instructions contain many independent operations



VLIW Constraints

- Very complicated compilers (schedulers)
  - High ILP = utilization of all units
    - Not always possible ⇒ NOPs
  - Planning of function unit utilizations at compile time!
  - Knowledge of operation latencies, dependencies, …
  - Must control code for conflicts (no HW runtime check!)
    - RAW (read after write), WAR (write after read)
    - Write conflicts - WAW (write after write)

add r1, r2, r3	mul r4, r2, r3	load r1, [r5]	nop

(Propagating) Scattered Context Grammar

- (Propagating) SCG is quadruple G = (V, T, P, S)

- V is finite set of symbols
- T is set of terminals, $T \subset V$
- S is the start symbol, $S \in V - T$
- P is a finite set of productions of the form

$$(A_1, \dots, A_n) \rightarrow (x_1, \dots, x_n),$$

where $\forall A_i : A_i \in V - T, \forall x_i : x_i \in V^*$

(Propagating SCG: $x_i \in V^+$)

Derivation Step, Generated Language

- Derivation Step

  Let $(A_1, \dots , A_n) \rightarrow (x_1, \dots , x_n) \in P$ and for $1 \leq i \leq n+1$ let $u_i \in V^*$:

  $$u_1A_1u_2A_2 \dots u_nA_nu_{n+1} \Rightarrow_G u_1x_1u_2x_2 \dots u_nx_nu_{n+1}$$

- Let $\Rightarrow^*_G$ be a reflexive transitive closure of $\Rightarrow_G$
- Generated Language
  - $L(G) = \{x \in T^* : S \Rightarrow_G^* x\}$

Generative Power

- $\mathscr{L}(\mathrm{SC}) = \mathscr{L}(\mathrm{RE})$
- $\mathscr{L}(\mathrm{CF}) \subset \mathscr{L}(\mathrm{PSC}) \subseteq \mathscr{L}(\mathrm{CS})$

PART 2

GENERATOR OF PROPER VLIW ASSEMBLER CODE

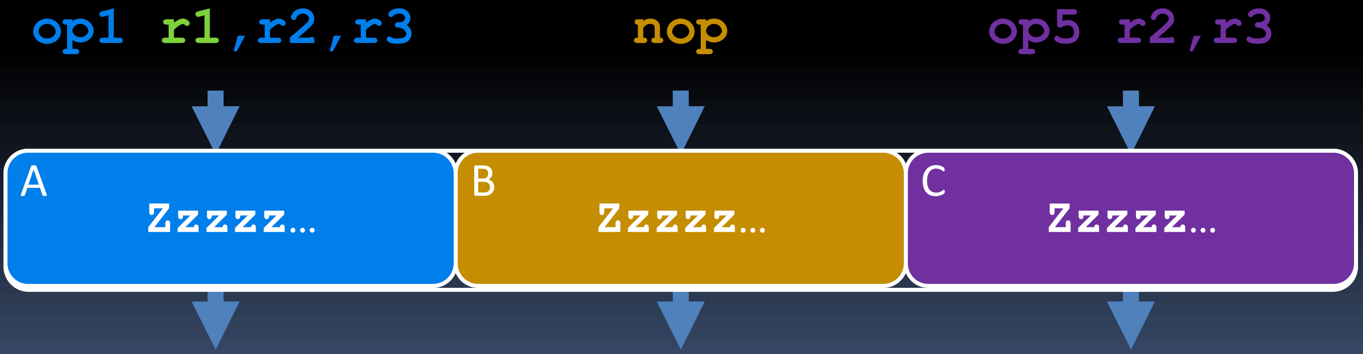
Example

- VLIW processor with 8 registers r_1, …, r_8
- 3 function units – A, B, C (e.g. ALU, MUL, MEM)
  - A operations: op1 r_1, r_2, r_3; op2 r_1, r_2, r_3 *($r_1 = r_2$ op r_3)*
  - B operations: op3 r_1, r_2, r_3 *($r_1 = r_2$ op r_3)*
  - C operations: op4 r_1, r_2; op5 r_1, r_2 *($r_1 = [r_2]$ or $[r_1] = r_2$)*
  - Function units are not pipelined
  - nop operation = no new job for function unit
- Operation latency
  - A: op1, op2 – 1 cycle
  - B: op3 – 2 cycles
  - C: op4 – 2 cycles; op5 – 3 cycles
- Write conflicts on instruction level are prohibited

VLIW Assembler Code Illustration (1/4)

```
(1)   op1 r1,r2,r3    nop               op5 r2,r3     ;;
(2)   op2 r6,r3,r2    op3 r7,r3,r2      nop           ;;
(3)   nop             nop               nop           ;;
(4)   op1 r4,r2,r3    nop               op4 r4,r2     ;;
```

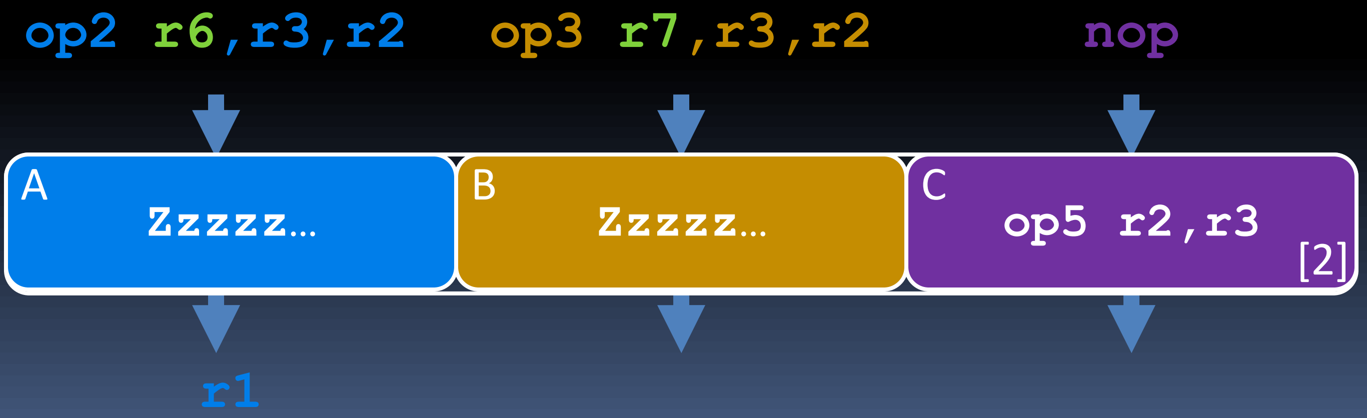
No conflicts



VLIW Assembler Code Illustration (2/4)

```
(1)   op1 r1,r2,r3   nop              op5 r2,r3    ;;
(2)   op2 r6,r3,r2   op3 r7,r3,r2     nop          ;;
(3)   nop            nop              nop          ;;
(4)   op1 r4,r2,r3   nop              op4 r4,r2    ;;
```

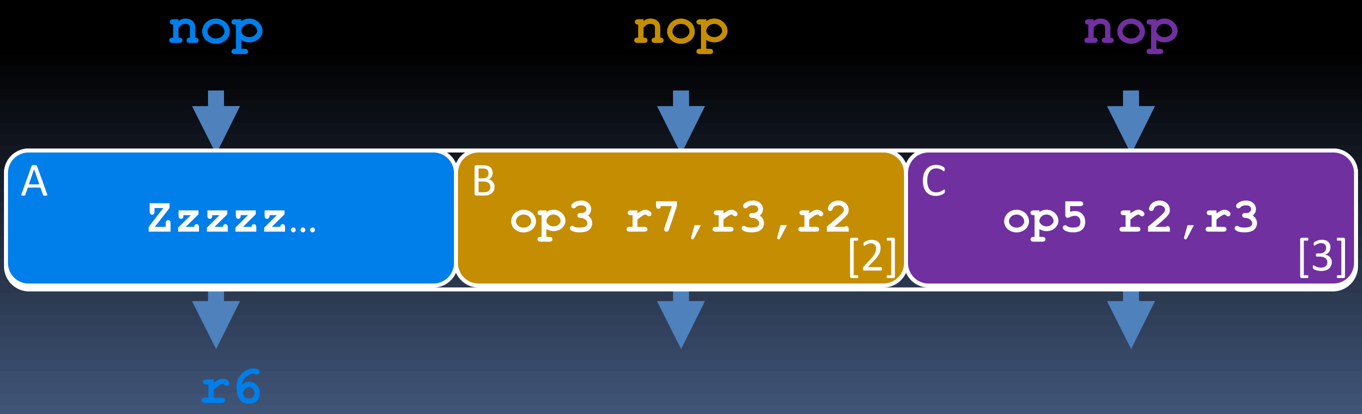
No conflicts



VLIW Assembler Code Illustration (3/4)

```
(1)   op1 r1,r2,r3    nop              op5 r2,r3    ;;
(2)   op2 r6,r3,r2    op3 r7,r3,r2     nop          ;;
(3)   nop             nop              nop          ;;
(4)   op1 r4,r2,r3    nop              op4 r4,r2    ;;
```

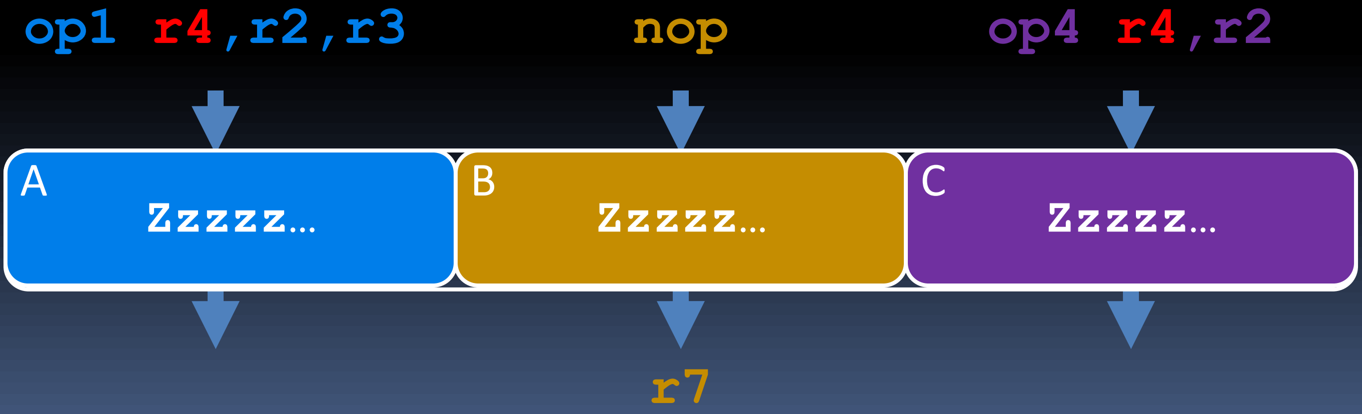
No conflicts



VLIW Assembler Code Illustration (4/4)

```
(1)   op1 r1,r2,r3    nop             op5 r2,r3    ;;
(2)   op2 r6,r3,r2    op3 r7,r3,r2    nop          ;;
(3)   nop             nop             nop          ;;
(4)   op1 r4,r2,r3    nop             op4 r4,r2    ;;
```

Write conflict!



Generator – Motivation

- State of the art
  - Manually created assembler code generators
    - Reservation tables
    - Register allocation
    - …
- We need formal method => automation
- Minimal number of rules
- Easy to parse

Methods 1

- Assume we have only ONE instruction:
  - Number of allowed combinations of operations is finite
  - Instruction is a sentence of language
  - Finite set of all allowed sentences is language (regular)
  - 67 millions of combinations!
  - 52 millions of allowed combinations!
- But we need to model also latencies and other constraints

  => This approach is not a solution!

„Slicing“ Methods 2 (No Latencies)

- Right regular grammar
  - 1255 rules

  $S \Rightarrow op_1\ A^{\emptyset} \Rightarrow op_1\ r_1\ B^{\{r1\}} \Rightarrow op_1\ r_1,\ C^{\{r1\}} \Rightarrow op_1\ r_1,\ r_2 D^{\{r1\}} \Rightarrow^*$

  $\Rightarrow op_1\ r_1,\ r_2,\ r_3\ |\ op_3\ r_3,\ r_5,\ r_4\ |\ op_4\ X^{\{r1,r3\}} \Rightarrow \ldots$
- Right linear grammar
  - 1165 rules

  $S \Rightarrow op_1\ A^{\emptyset} \Rightarrow op_1\ r_1,\ B^{\{r1\}} \Rightarrow op_1\ r_1,\ r_2, C^{\{r1\}} \Rightarrow \ldots$
- Context free grammar
  - 447 rules

  $S \Rightarrow op_1\ A^{\emptyset} \Rightarrow op_1\ r_1,\ R,\ R\ |\ B^{\{r1\}} \Rightarrow op_1\ r_1,\ R,\ R\ |\ op_3\ C^{\{r1\}} \Rightarrow$

  $\Rightarrow op_1\ r_1,\ R,\ R\ |\ op_3\ r_2,\ R,\ R\ |\ D^{\{r1,r2\}} \Rightarrow \ldots$

„Slicing“ Methods 3 (No Latencies)

- Propagating SCG
  - 447 rules – no improvement against CFG
- SCG
  - 146 rules
  - $(@^{M}, W) \rightarrow (\varepsilon, r_i\ @^{M \cup \{ri\}})$

$S \Rightarrow @^{\emptyset}ABC\# \Rightarrow @^{\emptyset}\ op_1\ W, R, R|BC\# \Rightarrow op_1\ r_1\ @^{\{r1\}}, R, R|BC\# \Rightarrow$

$\Rightarrow op_1\ r_1\ @^{\{r1\}}, R, R\ |\ op_3\ W, R, R\ |\ C\# \Rightarrow$

$\Rightarrow op_1\ r_1, R, R\ |\ op_3\ r_2\ @^{\{r1,r2\}}, R, R\ |\ C\# \Rightarrow$

$\Rightarrow op_1\ r_1, R, R\ |\ op_3\ r_2\ @^{\{r1,r2\}}, R, R\ |\ op_4\ W, R\ \# \Rightarrow$

$\Rightarrow op_1\ r_1, R, R\ |\ op_3\ r_2, R, R\ |\ op_4\ r_3\ @^{\{r1,r2,r3\}}, R\# \Rightarrow$

$\Rightarrow op_1\ r_1, R, R\ |\ op_3\ r_2, R, R\ |\ op_4\ r_3, R;; \Rightarrow \dots$

„Slicing“ Methods 4 (With Latencies)

- Right regular, right linear, CFG, …
  - 2500+ rules
- Propagating SCG
  - 654 rules
  - $(@^M, OP_1) \rightarrow (op_1, @^M); (@^M, R) \rightarrow (r_1, @^M); \ldots$
- SCG
  - 150 rules

$S \Rightarrow @^{\emptyset}ABC\$L_AL_BL_C\# \Rightarrow @^{\emptyset}\ op_1\ W, R, R|BC\$AL_BL_C\# \Rightarrow$

$\Rightarrow op_1\ r_1\ @^{\{r1\}}, R, R|BC\$AL_BL_C\# \Rightarrow$

$\Rightarrow op_1\ r_1\ @^{\{r1\}}, R, R\ |\ op_3\ W, R, R\ |C\$AB_1L_C\# \Rightarrow^*$

$\Rightarrow op_1\ r_1, r_2, r_3\ |\ op_3\ r_2\ @^{\{r1,r2\}}, r_6, r_5\ |\ nop\$AB_1C\# \Rightarrow$

$\Rightarrow op_1\ r_1, r_2, r_3\ |\ op_3\ r_2, r_6, r_5\ |\ nop;;@^{\emptyset}AB_1C\$L_AL_BL_C\# \Rightarrow\ldots$

PART 3

FUTURE RESEARCH

Motivation

- SCG-based „slicing“ generator – pretty good, but…
- The number of rules is still too high
  - @-rules are problem (large number of registers)
  - Replace @-rules with forbidding rules

$$(W_{R1}, W_{R1}) \rightarrow (Z, Z)$$

  - How to make sure that this rule will be applied instead of a rule $(W_{R1}) \rightarrow (r_1)$?
  - „Default“ SCG is not good enough
  - We need to add priority to rules!

(Propagating) SCG with Priority Rules

- (Propagating) SCG with Priority Rules is quintuple $G = (V, T, P, \pi, S)$
- V is finite set of symbols
- T is set of terminals, $T \subset V$
- S is the start symbol, $S \in V - T$
- P is a finite set of productions of the form

  $$(A_1, \dots , A_n) \rightarrow (x_1, \dots , x_n),$$

  where $n \geq 1$, $\forall A_i : A_i \in V - T$, $\forall x_i : x_i \in V^*$

  (Propagating SCG with Priority Rules: $x_i \in V^+$)
- π is priority function: $\pi: P \rightarrow \mathbb{N}$

Priority Derivation, Generated Language

- Let $G = (V, T, P, \pi, S)$ is (P)SCG-P, $p \in P$, for $1 \leq i \leq n+1$ let $u_i \in V^*$: $u = u_1A_1 \dots u_nA_nu_{n+1}$,

  $v = u_1x_1 \dots u_nx_nu_{n+1}$,

  $w = u_1y_1 \dots u_ny_nu_{n+1}$
- Define Priority Derivation Step as

  $u \;{}_{\pi}{\Rightarrow}_G\; v\ [p]$

  iff $u \Rightarrow_G v\ [p]$ and there is no $r \in P$ satisfying $\pi(r) > \pi(p)$ such that $u \Rightarrow_G w\ [r]$.
- Generated Language
  - $L(G) = \{x \in T^* : S \;{}_{\pi}{\Rightarrow}_G^*\; x\}$

Generative Power

- $\mathscr{L}$(SC-P) = ? $\mathscr{L}$(RE)
- $\mathscr{L}$(PSC-P) = ? $\mathscr{L}$(CS)

- Proofs needed

„Forbidding“ Method

- (P)SCG with priority rules (no latency)
  - 42 rules
- (P)SCG with priority rules (with latency)
  - 46 rules
  - $\pi((W_{R1}, W_{R1}) \rightarrow (Z, Z)) = 2$ (Z is „block“ symbol)
  - $\pi((W_{R1}) \rightarrow (r_1)) = 1$
  - $\pi((\$, \#) \rightarrow (;;, \$L_A L_B L_C\#)) = 0$

  $S \Rightarrow ABC\$L_A L_B L_C\# \Rightarrow^*$

  $\Rightarrow op_1\ W_{R1}, r_2, r_3 \mid op_3\ W_{R1}, r_6, r_5 \mid nop\ \$AB_1C\# \Rightarrow$

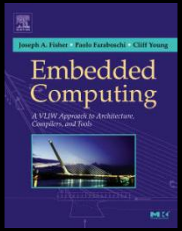
  $\Rightarrow op_1\ Z, r_2, r_3 \mid op_3\ Z, r_6, r_5 \mid nop\ \$AB_1C\#$

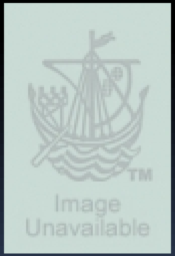
Conclusion

- New approach of VLIW assembler generators
  - Minimal number of rules
  - Still easy to parse
  - Comparison of methods
  - (P)SCG-based generators
- (P)SCG with priority

References

- Fisher, Faraboschi, Young: Embedded Computing - A VLIW Approach to Architecture, Compilers, and Tools, 2005
- Kolář: Exploitation of Scattered Context Grammars to Model Constraints between Components, ASIS 2009
- Meduna, Techet: Scattered Context Grammars and their Applications, 2009 (2010)


Joseph A. Fisher · Paolo Faraboschi · Cliff Young
Embedded Computing
A VLIW Approach to Architecture, Compilers, and Tools



Questions?

THANK YOU FOR YOUR ATTENTION