Integrated Code Generation by Using Fuzzy Control System

Dipl.-Ing. Xiaoyan Jia
Outline

I Introduction
   - STA Architecture
   - Code Generation for STA

II Analysis of Problem in Code Generation
   - Access of Results in STA
   - Integrated Code Generation

III Algorithm
   - Basic Concepts in Fuzzy
   - Details of Integrated Code Generation

IV Experimental Result

V Conclusion
STA Architecture

- Modules consist of input and output ports
- Output ports are buffered
- Results can be held further in output registers until the execution of the next operation is finished
- Data at an input port is selected from a set of connected output ports by a MUX (DDR)

*Synchronous Transfer Architecture (STA)*
STA Architecture and Code Generation

Focus:
- To decide, the results should be transferred into general registers or be held further in output registers.

Problem:
- Phase-coupling problem specially for STA architecture.
Access of Results in STA

\[
\begin{align*}
\text{a} &= 0\times1 \\
\text{read } b & \\
\text{c} &= \text{a} + \text{b} \\
\text{b is stored in general register r0}
\end{align*}
\]

Original Instructions in MIR

TU Dresden, Xiaoyan Jia
Phase-Coupling Problem in STA

Scheduling & register allocation are separated

- **Register allocation is before scheduling**
  - Impossible to know whether DDR could be happened
  - All the results should be saved in registers
  - Poor performance and no STA feature is represented

- **Scheduling is before register allocation**
  - DDR is detected
  - FUs can not be used until results could be released
  - Reduction of instruction level parallelism
Framework of Compiler Backend

Traditional Compiler backend

1. variant
- Register allocation
- Instruction scheduling

2. variant
- Register allocation
- Instruction scheduling

Intermediate Representation → Instruction selection → Machine Description

Intermediate Representation → Instruction selection → Assembly Code

Integrated Compiler backend

- Register allocation
- Instruction scheduling

Optimizer

Intermediate Representation → Instruction selection → Assembly Code
Basic Concepts in Fuzzy

**Fuzzy truth** \( \mu \in [0, 1] \) represents the membership \((u)\) in vaguely defined sets

**Fuzzy set:** \( A := \{(u, \mu_A(u)) \mid u \in U, \mu_A(u) \in [0,1]\} \)

**Complement:** \( \overline{A} = \{(u, (1 - \mu_A(u))) \mid u \in U\} \)

**Inference:** \( \mu_c(u) = \mu_A(u) \cap \mu_B(u) = \min\{\mu_A(u), \mu_B(u)\} \quad \forall u \in U \)

**Composition:** \( \mu_c(u) = \mu_A(u) \cup \mu_B(u) = \max\{\mu_A(u), \mu_B(u)\} \quad \forall u \in U \)
P1: Truth of Fusion into Loops

int a = 6;
int i, c, f;
for (i = 0; a < 100; i++) {
    int b = 4;
    a = a + i;
    c = a - b;
}
int d = 9;
f = c * d;

\( L_R \): observed loop

\[
\begin{align*}
    i1 & : a = 6 \\
    i2 & : i = 0 \\
    i3 & : a < 100 \\
    i4 & : i = i + 1 \\
    i5 & : b = 4 \\
    i6 & : a = a + i \\
    i7 & : c = a - b \\
    i8 & : d = 9 \\
    i9 & : f = c \times d
\end{align*}
\]

\( P_1 \):

<table>
<thead>
<tr>
<th></th>
<th>i1</th>
<th>i2</th>
<th>i3</th>
<th>i4</th>
<th>i5</th>
<th>i6</th>
<th>i7</th>
<th>i8</th>
<th>i9</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Predecessor of \( L_R \): very easily

In \( L_R \): easily

Others: not easily

TU Dresden, Xiaoyan Jia
The ratio of the path’s length of an instruction against the longest path in the whole DAG is determined as the instruction’s $P_2$.

$k$ has the longest path in the whole DAG: 4
P3: Truth of DDR

indirect DDR:

read a
b = 2
e = a + b
f = 5
g = c + d
h = e + f
i = - g

direct DDR:

read a
b = 2
e = a + b
f = 5
g = c + d
h = e + f
P4: Truth of Instruction Scheduling

\[
\begin{align*}
a &= 1 \\
b &= 2 \\
c &= a + b \\
d &= -3 \\
e &= 7 \\
f &= d - e \\
h &= 0 \\
k &= 9 \\
j &= -k \\
i &= h \cdot j \\
g &= i + f \\
l &= 6 \\
m &= l / g
\end{align*}
\]

\begin{align*}
a &= 1 \\
b &= 1 \\
c &= 0 \\
d &= 0 \\
e &= 1 \\
f &= 0 \\
h &= 0 \\
k &= 0 \\
j &= 0 \\
i &= 1 \\
g &= 0 \\
l &= 0 \\
m &= 0
\end{align*}

- **already scheduled**
- **with unscheduled predecessor**
- **predecessors are all scheduled**
- **without predecessors**
Pd: Final Dynamic Factor

\[ P_d = P_1 \cap (P_2 \cup P_3) \cap P_4 = \min(P_1, \max(P_2, P_3), P_4) \]

\[ P_1, P_2, P_3, P_4 \in [0, 1] \]

\[
\begin{align*}
| & \quad P_1 & \quad P_2 & \quad P_3 & \quad P_4 & \quad P_d \\
\hline
\text{i1: } a & \quad 6 & & & & 1 \\
\text{i2: } i & \quad 0 & & & & 1 \\
\text{i3: } a & \quad < 100 & & & & 0 \\
\text{i4: } i & \quad = i + 1 & & & & 0 \\
\text{i5: } b & \quad = 4 & & & & 0 \\
\text{i6: } a & \quad = a + i & & & & 0 \\
\text{i7: } c & \quad = a - b & & & & 0 \\
\text{i8: } d & \quad = 9 & & & & 0 \\
\text{i9: } f & \quad = c \times d & & & & 0 \\
\end{align*}
\]

\[
\begin{align*}
| & \quad P_1 & \quad P_2 & \quad P_3 & \quad P_4 & \quad P_d \\
\hline
\text{i1: } a & \quad 1 & \quad 1 & \quad 0 & \quad 1 & \quad 1 \\
\text{i2: } i & \quad 1 & \quad 1 & \quad 0 & \quad 1 & \quad 1 \\
\text{i3: } a & \quad 0.5 & \quad 1 & \quad 0 & \quad 0 & \quad 0 \\
\text{i4: } i & \quad 0.5 & \quad 2/3 & \quad 0 & \quad 0 & \quad 0 \\
\text{i5: } i & \quad 0 & \quad 2/3 & \quad 0 & \quad 1 & \quad 0 \\
\text{i6: } i & \quad 0.5 & \quad 2/3 & \quad 0 & \quad 0 & \quad 0 \\
\text{i7: } i & \quad 0 & \quad 1/3 & \quad 0 & \quad 0 & \quad 0 \\
\text{i8: } i & \quad 0 & \quad 1/3 & \quad 0 & \quad 1 & \quad 0 \\
\text{i9: } i & \quad 0 & \quad 0 & \quad 0 & \quad 0 & \quad 0 \\
\end{align*}
\]

\[
\begin{align*}
| & \quad P_1 & \quad P_2 & \quad P_3 & \quad P_4 & \quad P_d \\
\hline
\text{i1: } a & \quad 1 & \quad 1 & \quad 0 & \quad 0 & \quad 0 \\
\text{i2: } i & \quad 1 & \quad 1 & \quad 0 & \quad 0 & \quad 0 \\
\text{i3: } a & \quad 0.5 & \quad 1 & \quad 0 & \quad 0 & \quad 0 \\
\text{i4: } i & \quad 0.5 & \quad 2/3 & \quad 1 & \quad 0 & \quad 0 \\
\text{i5: } i & \quad 0 & \quad 2/3 & \quad 0 & \quad 1 & \quad 0 \\
\text{i6: } i & \quad 0.5 & \quad 2/3 & \quad 1 & \quad 1 & \quad 0.5 \\
\text{i7: } i & \quad 0 & \quad 1/3 & \quad 0 & \quad 0 & \quad 0 \\
\text{i8: } i & \quad 0 & \quad 1/3 & \quad 0 & \quad 1 & \quad 0 \\
\text{i9: } i & \quad 0 & \quad 0 & \quad 0 & \quad 0 & \quad 0 \\
\end{align*}
\]
\[ P_d = P_1 \cap (P_2 \cup P_3) \cap P_4 = \min(P_1, \max(P_2, P_3), P_4) \]

\[ P_1, P_2, P_3, P_4 \in [0, 1] \]
$P_d = P_1 \cap (P_2 \cup P_3) \cap P_4 = \min(P_1, \max(P_2, P_3), P_4)$

$P_1, P_2, P_3, P_4 \in [0, 1]$
Register Allocation

- Knapsack Algorithm:

sld a
b = 5
c = a + b
d = -1
e = -d
f = c * d

For e: output register: 0
general-purpose register: write + read = 1 + 1 = 2

d = -1
...
e = -d
write d
...
read d
f = c * d
...

For f: output register: max(a,b) + c = 2 + 1 = 3
general-purpose register:
max((max(a,b) + c), (write + read)) = 2 + 1 = 3

General-purpose register is more free than FU decoder
A 42.7% to 62.5% code size is reduced by using integrated method in compared with the phaseseparated work.
Experimental Results: *Execution Time*

- Comparison of the one by phase-separated method, the gain of integrated backend is from **45.9% to 58.3%**
Experimental Results: Memory Spill Code

- **Almost no** additional memory spill code are generated by using our integrated compiler backend.
Experimental Results: *Comparison of the ILP*

- The backend by using fuzzy control system and the based on ILP have *almost the same* code performances.
- But the code generation by using fuzzy needs much *shorter* compilation time. For the previously mentioned testbenches, it takes only 1 second to about 20 minutes to generate code.
Conclution

- Comparison of the traditional phase-separated compiler backend
  - Take advantage of the STA features more efficiently
  - Generate assembly code with much better performance
  - Reduce memory spill codes greatly
  - Results much lower processor hardware power consumption

- Comparison of the Integer Linear Programming
  - Not so difficult to implement
  - Much shorter compilation time than ILP
Thank you for your attention!