Phi-Predication for Light-Weight If-Conversion

Weihaw Chuang
Brad Calder
Jeanne Ferrante

University of California, San Diego
Benefits of If-Conversion

- Eliminates hard to predict branches
  - Important for deep pipelines
- How? Executes all paths with qualified update
  - False path - no updates nor sides-effects

```
CMP P6,P7=test ;;
(P6) MOV R33=1
(P7) MOV R33=20 ;;
ADD =R33,5
```

E.g. if P6=1, P7=0
R33 at ADD is 1
Predication Challenge

- Objective: Apply predication benefits to aggressive Out-Of-Order Execution
  - Why? Deep pipelines ⇒ Long branch misprediction penalty
  - If-Conversion

- Challenge: Multiple Definition Problem
  - Predication causes unnecessary stalls during Out-Of-Order register renaming

- Solution: ALWAYS-WRITE Phi-Predication.
Conditional-Writer Predication

- Today’s predication is Conditional-Writer
  - Qualify state update and side-effects via predicate
    \[(P6) \text{ MOV R33=1} \Rightarrow \text{if}(P6=T) \text{ then R33=1}\]
  - e.g. IA64, TriMedia, ARM

- Typically heavy-weight \(\Rightarrow\) Implementation cost is high
  - Most instructions qualified
  - Opcode space cost
  - Needs predicate-aware compiler analysis
Outline

- Multiple-Definition Problem
- Light-Weight Phi-Predication
- Code Generation
- Performance Result
Multiple Definition Problem

- Conditional-writer on Out-of-Order execution
- In OOO Register renaming
  - If predicate not ready $\Rightarrow$ ambiguity choosing last def
- One Solution: Stalling till predicate ready $\Rightarrow$ very costly

---

Instruction Propagation through Out-Of-Order Pipeline

- RENAME
  - ISSUE
  - REGFILE
  - EXE

- ADD $=R33,5$
- (P6) MOV R33=1
- (P7) MOV R33=20
- CMP P6,P7=test

University of California, San Diego
Outline

- Multiple-Definition Problem
- Light-Weight Phi-Predication
- Code Generation
- Performance Result
Phi-Predication ISA

- PHI
  
  \[
  \text{phi } r32=(p1),r33,r34 \\
  \text{phi } r32=(p1),100,r34 \\
  \text{fphi } f32=(p1),f33,f34
  \]

- MEMORY
  
  \[
  \text{ld } r32=[r33],(p1) \\
  \text{st } [r32]=r33,(p1)
  \]

- ORP
  
  \[
  \text{orp } p1=p2,p3,p4
  \]

- UNCONDITIONAL COMPARE
  
  \[
  \text{cmp.eq.unc } p1,p2=r3,r4,(p3) \\
  \text{cmp.gt.unc } p1,p2=r3,r4,(p3) \\
  \text{cmp.ge.unc } p1,p2=r3,r4,(p3)
  \]
Phi-Instruction

- Key idea: Always-assign register destination
- Qualify register dataflow via selection
  - From Select operation (Multiflow ISA)
  - Regular compiler analysis

Example:
\[
\text{phi } r32 = (p1), r33, r34
\]
Phi Evaluation Chain

- Multiple join edges ⇒ maybe multiple PHI’s
  - Rename definitions
  - For N definitions at join blk need N-1 PHI’s to select correct definition
  - Evaluate phi’s OPERANDS in topological order

MOV R35₁=1
MOV R35₂=2
MOV R35₃=3

MOV R10=1
MOV R20=2
MOV R30=3

PHI R2=(P2),R20,R10
PHI R35=(P3),R30,R2
Small Region If-Conversion

- Emphasize benefit of mispredicting branch removal
  - against penalty of executing all paths
- Small Regions
  - Best captures benefit
  - Similar schedule for branch, conditional-writer, Phi-Predication

<table>
<thead>
<tr>
<th>Cond-Writer Predication</th>
<th>Phi Predication</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP P6,P7=R34,R32</td>
<td>MOV R35=1</td>
</tr>
<tr>
<td>;;</td>
<td>CMP P0,P7=R34,R32</td>
</tr>
<tr>
<td>(P6) MOV R33=1</td>
<td>;;</td>
</tr>
<tr>
<td>(P7) MOV R33=20</td>
<td>PHI R33=(P7),20,R35</td>
</tr>
<tr>
<td>;;</td>
<td>;;</td>
</tr>
<tr>
<td>ADD =R33,5</td>
<td>ADD =R33,5</td>
</tr>
</tbody>
</table>
If-Conversion Schedule Issues

- Avoid **execution resource constraints** - execute all paths lengthens schedule

- Avoid **imbalanced paths** - might penalize short path

- Avoid **long Phi-Chains** - long evaluation sequences caused by multiple join edges
Qualifying Memory Operations

- Loads and Stores
  - Qualified exception and memory update
  - False loads assign destination value 0 - obey always-
    write

```
CMP P6,P7=R34,R32 ;
LD R33=[R35],(P6)  
LD R38=[R36],(P7)  ;;
PHI R33=(P7),R37,R38  ;;
ADD R4=R33,5
```

```
CMP P6,P7=R34,R32  
(p6) br taken
LD R37=[R35],(P6) LD R38=[R36],(P7)  ;;
PHI R33=(P7),R37,R38  ;;
ADD R4=R33,5
```
Qualifying Predicate Generators

- Unconditional Compares- (from IA64) for nested control flow
  - If qualified false ⇒ must not set predicates true

```plaintext
CMP P6,P7=test
CMP.UNC P9,P10=test2

CMP P6,P7=test ;
CMP P9,P10=test2,(P6)
```
Generating Join Block Predicates

- OR’ing of predicates (ORP)
  - Predicates must obey always-write principle

```
CMP P6, P7 = test

CMP.UNC P9, P10 = test2

OPR P11 = P7, P10

CMP P6, P7 = test ;;
CMP P9, P10 = test2, (P6) ;;
ORP P11 = P7, P10
```
Outline

- Multiple-Definition Problem
- Light-Weight Phi-Predication
- Code Generation
- Performance Result
Compiler Steps

Region Picker

- find acyclic region
  - single entry/exit
  - no calls/ret/system

- estimate phi

- estimate schedule
  - pick small region

- Don’t qualify expensive instructions or side-exit branches

- Determine schedule
  - estimate Phi insertion points

Modified Intel’s IA64 compiler
Compiler Steps

Code-Generator

- find predicates
  - insert ORP

- insert PHI

- qualify LD/ST/CMP

- collapse region

- combining optimization

- Insert predicate generators

- Qualifies register data-flow

- Qualifies memory side-effects and predicate data-flow

- Delete branches, fold all path

- Delete unnecessary moves
Code Gen Example

- from WC (word count)
- popular example from earlier papers
Guard Predicate

Guard predicate

Selection:
\[ \phi r22 = (p15), 1, r38 \]

Qualification:
\[ \text{cmp.unc } p7, p6 = 32, r28, (p8) \]

Modified from: Park and Schlanksker (RK Algorithm)

University of California, San Diego
Predicate Generation

- Locate predicate generators
- Control Dependence
  - B46: CD(B48) = {-B46}
  - Gen P8 at B46, neg edge:
    cmp.unc -, p8 = test, (p12)
Phi Insertion Point

- Find insertion point
- Modified Static-Single Assignment (SSA)
- Dominance Frontier (DF)
- Register dataflow decision point(s) at join
- Ex.
  - $\text{DF}(B49) = \{B52\}$
  - $\text{DF}(B51) = \{B52\}$

modified from Cytron et al. (SSA)

University of California, San Diego
Unique Register Definitions

- Rename all assignments to unique instances in region
- Forward propagate to Uses
- Collect in phi-lists (at DF pt)

\[
V_{10} = \text{phi-list}(V_{10}, V_{11}, V_{12})
\]
Phi Code Generation

- Phi-List operands sorted topologically
- Associated w/operand
  - Def-block guard predicate \( \Rightarrow \) selector predicate
- Code gen: picks operand pairs
  - Use more specific selector predicate

\[ V_{10}^{=?} \]

\[ V_{10} = \text{phi-list}(V_{10}, V_{11}, V_{12}) \]

\[ V_{11} = 1 \]

\[ V_{12} = \text{phi}(p_{15}) V_{11}, V_{10} \]

\[ V_{13} = \text{phi}(p_{15}) V_{11}, V_{10} \]

\[ V_{10} = \text{phi}(p_{14}) V_{12}, V_{13} \]
B45:
{
45    ld1              r28=[r29],1
47    add              r26=1,r36
51    add              r24=1,r37
} {
45    cmp4.eq.unc      p13,p12=10,r28
45    cmp4.eq.unc      p10,p0=10,r28
49    phi              r36=(p10),r26,r36
} {
46    cmp4.eq.unc      p9,p8=9,r28,(p12)
48    cmp4.eq.unc      p7,p6=32,r28,(p8)
48    nop.i            0
} {
49    orp              p15=p13,p9,p7
50    cmp4.eq.unc      p0,p14=r38,r0,(p6)
52    phi              r37=(p14),r24,r37
} {
52,49   phi              r22=(p15),1,r38
52,51   phi              r38=(p14),0,r22
52    br.ctop          .B45
}
Outline

- Multiple-Definition Problem
- Light-Weight Phi-Predication
- Code Generation
- Performance Result
Simulation Models

- "Select-op"
  - CMOV-like
  - allow only PHI, CMP.UNC

- "Wang et. al."
  - IA64: conditional-writer, heavy-weight ISA
  - RAT generated select-ops to avoid multiple definition problem

- "Phi-Op"
  - This paper
  - PHI, CMP.UNC, ORP, LD/ST
OOO Speedup (12 cycle penalty)

-15%  -10%  -5%  0%  5%  10%  15%

-10%  -5%  0%  5%  10%  15%  20%  25%

Select-op, OOO
Cond-Write
Pred, Wang, OOO
Phi-Pred, OOO

Normalized against no predication
OOO Speedup
(30 cycle penalty)

-20%
-10%
0%
10%
20%
30%
40%
50%

Sprangle and Carmean
(deep pipelines)

University of California, San Diego
Conclusion

- Future aggressive OOO processors will have very deep pipelines
- If-conversion helps reduce deep pipeline branch misprediction penalties
- **Phi-Predication** provides a light-weight implementation that solves the multiple-definition problem
  - described compiler Phi-Predication if-conversion
Backup Slides

- ISA Comparison
- More Code-Generation Tutorial
- Multiple-definition problem does not affect memory qualification
Comparison

Conventional:
- Heavy-Weight: Most instruction qualified
- OOO implementation has multiple-definition problem
- Predicate def-use dependency
- Requires predicate aware compiler analysis

Phi-Predication:
- Light-weight:
  - four instruction types
  - phi, ld/st, cmp, ORP
- Always write definition
- Predicate promotion
- Regular compiler analysis
Code Gen Example:
Inner Loop of cnt/wc.c

```c
for(c=buf; len--; ++C) {
    switch(*C) {
        case NL:
            ++linecnt;
        case TAB:
        case SPACE:
            gotsp = 1;
            continue;
        default:
            if(gotsp) {
                gotsp = 0;
                ++wordct;
            }
    }
}
```

University of California, San Diego
Predicate Generation: Multiple Incoming Edge

- Control Dependence (CD)
  - B46: CD(B49) = \{B47, +B46, +B48\}

- OR’ing control predicates to get guard predicate

- Control predicate guards edge (≠ guard predicates!)
  - p15 = ORP(p13, p9, p7)

- Gen Uncond Comp for control predicate

University of California, San Diego
Qualified Memory Update

- Ld/St memory multiple-definition problem not problematic
- Typically predicates are ready

<table>
<thead>
<tr>
<th>RENAME</th>
<th>ISSUE</th>
<th>REGFILE</th>
<th>EXE</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
</table>

PHI R32=(P6),R33,R35
LD R33=[R34](P6)  LD R35=[R36](P7)
CMP P6,P7=R30,R31