If-Conversion
SSA Framework and Transformations

SSA’09

Christian Bruel
29 April 2009
Motivations

- Embedded VLIW processors have architectural constraints
  - No out of order support, no full predication, Several functional units
  - Need conditional support and compiler techniques to provide aggressive ILP

- Traditional if-conversion techniques for partial predication:
  - Local if-then-else constructs or pattern matching style optimizations.
  - Global Hyperblock approach followed by a full predication if-conversion algorithm.
  - No “global” framework allowing to extend the ISA set of predicated instructions => need a configurable if-conversion algorithm.

- This paper presents:
  - A SSA if-conversion algorithm using the “select” instruction and speculation
  - An extension to this algorithm to support a configurable set of predicated instructions
If-Conversion

- Process to convert a control flow region into a sequence of conditional instructions.
  - Remove conditional branches and simplify control flow
  - Need architectural support
- Increase ILP
- Increase locality
- Some optimisations are more efficient with a single basic block
  - (software pipelining, instruction scheduling)

Global problem
- Control flow regions can be complex
- Not limited to simple regions ("hammock") or pattern matching
- Lot of tradeoffs
- Limit code size explosion or even reduce it

Local problem
- Predicate construction and allocation
- Variable renaming
- Predicated instruction constructions
- Predicated instruction optimizations
3 types of architectural support

Example: conditional assignment

<table>
<thead>
<tr>
<th>“select” if-conv</th>
<th>Partial predication</th>
<th>Full predication</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c = \text{cmp } t,0$</td>
<td>$c = \text{cmp } t,0$</td>
<td>$c = \text{cmp } t,0$</td>
</tr>
<tr>
<td>$t = r+1$</td>
<td>$c \ ? r = \text{idw}$</td>
<td>$c \ ? r = r+1$</td>
</tr>
<tr>
<td>$r = \text{select } c,t,r$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Framework is able to support all or a mix of those architectural supports
  - Balance speculation and predication
- Minimum requirement is a form of conditional move and predicate building (cmp) and merging (logical and/or)
If-Conversion (why SSA)

Q1: which r to use?
Q2: what is the best connection to the definition? (smallest predicate set)
Q3: what are the value that needs to hold predicate?
If-Conversion (why SSA)

Q1: which r to use?
Q2: what is the best connection to the definition? (smallest predicate set)
Q3: what are the value that needs to hold predicate?

A1: SSA does the renaming
A2: PHI shows variables that are conditional, merge the immediate condition
A3: Only those that have a reaching point. Other a speculated
If-Conversion (why SSA)

SSA holds (almost) all the information we need
Need a global framework to handle more complex regions
And a phi walking process

r1 depends on TRUE predicate
r2 depends on 'c'

l is not processed: speculed
Local SSA transformation (predicate merging)

Costs:
- One predicate register
- One extra op in critical path
- Higher dependence height

- \( r_1 = s + 1 \)
- \( r_2 = 2 \)
- \( r = \phi(r_1, r_2) \)
- \( c = c_1 \land c_2 \)
- \( r = \text{select } c ? r_1, r_2 \)
IFC-SSA Global Framework

- A set of transformations are applied iteratively in postorder on the CFG
  - Considered regions are group of basic blocks with a single conditional entry
  - Blocks reached on multiple conditions are detected (predicate merge)
  - Side entries can be removed using block duplication

- While the region grows, when do we stop?
  - Decision to continue reconsidered at each iteration based on cost functions and static information
  - Decision to continue until all instructions from the candidate region can be removed, speculated or predicated
  - Objective function is local so easy to compute
    - whole don’t exceed the sum of the parts
  - Stop on hazardous instructions (generally)
IFC-SSA running example

Region processed in postorder: BB4, BB2, BB1
SSA for partial predication

- Problem: transforming $\phi$ into select was realizing join points
  - Need to relink select into the equivalent predicated instructions
  - Principle: Transforms speculation into predication

```
t=test1 br<t>
P=test2 r1=ldw r2=sub s=select p?r1:r2
r3=add
r=\phi(s,r3)
```
SSA for partial predication

- Problem: transforming $\phi$ into select was realizing join points
  - Need to relink select into the equivalent predicated instructions
  - Principle: Transforms speculation into predication

- Not very convenient: Not SSA anymore.
  - Predication introduced a renaming problem.
  - Need to keep the select form and add a new speculation->predication pass after out-of-ssa
    - Breaks objective function and incompatible with SSA code generator
  - Or generate directly an extended SSA for predication (current implementation uses $\psi$-ssa)

```
t = test1
br <t>

P = test2
r1 = ldw
r2 = sub
s = select p?r1, r2

r = $\phi$(s, r3)

r3 = add

P = test2
P ? s = ldw
!p ? s = sub

r = $\phi$(s, r3)

r3 = add
```
Nested predicates are automatically supported by select speculation – like any other instruction
- Predicate are handled has data operand dependency

Predicated version even more difficult to express fully in SSA

Need to express SSA renaming with “speculated merging points”
- And after SSA complicated renaming
- Let SSA do it with an extension

r1 is defined on ‘p’ and r2 is defined on !p -> coalesced, can be renamed

s is defined for ‘t’, r3 defined on ‘!t’ -> can be renamed
Status

- Implemented for 3 kinds of conditional instructions support
  - ST231 speculative model
    - 4 issue VLIW (4 32 bit ALUs, 2 multipliers, 1 load store, 64 x 32 bit registers
    - “select” instruction, 8 1 bit branch registers, Wired and/or instructions
    - Speculative loads using static analysis or “dismissible load”
    - Speculative stores using compiler generated dummy stack slots
  - ST240 Predicated variant using predicated loads and stores
  - Internal experimental core with fully predicated ISA

- Implemented in the production C/C++ Open64 code generator
  - After middle end optimizations and code selection
  - Before loop unrolling, local and global schedulings

- Further possible improvements
  - Allow multiple exits regions -> efficient region selection
  - Add backedge coalescing to improve if-converted loop bodies
  - Improve scalar optimisations on predicated instructions
Results

- Bench set with MIBENCH and SPEC 2000
- No code size bloat. (or even decrease)
- Select speculated model:
  - 23 % geometrical mean for multimedia applications
  - (~3% for scalar applications)
- Partially Predicated model with $\psi$ SSA:
  - 25 % geometrical mean for multimedia applications
- wc.c: only one back edge branch remaining.
  - Static schedule loop into a single basic block.
  - 31 cycles -> 23 cycles
Conclusion

- Performance experience
  - Fitness to resources availability
    - Objective function very easy to compute locally
    - The if-converted region cost doesn’t exceed the sum of the parts ponderated by edge frequencies
  - Dynamically adapt to speculated or predicated model when alternative possible
  - Speculation shows more efficient than predication due to
    - Smaller dependence height
    - Predication need many predicate registers (while speculation needs more general registers)
    - But predication increases the set of if-converted regions (alleviate hazards)

- SSA naturally sets the environment for if-conversion analysis
  - Need to be maintained to be used incrementally
  - Nested conditionalized can be refeeded to the process

- Speculated model
  - We enter in strict SSA form and produces strict SSA
  - Conditional instructions are realized into a select instruction

- Predicated model
  - Enter in strict SSA form and produces extended ($\psi$) SSA or strict SSA+select (need post SSA transformations)

- Papers:
  - If-Conversion SSA framework for partially predicated VLIW architectures – C.Bruel
    CGO – ODES-04: Workshop on Optimizations for DSP and Embedded Systems
  - If-Conversion for Embedded VLIW Architectures – C.Bruel
    - IJES - International Journal of Embedded Systems. To be published
Thanks

- Thanks to Christophe Guillon and Francois de Ferriere for their feedbacks, suggestions and (sometime) bug reports.

- Questions ?