A Lightweight Algorithm for Dynamic If-Conversion During Dynamic Optimization

Kim Hazelwood
Thomas M. Conte

Tinker Research Group
Department of Electrical & Computer Engineering
North Carolina State University
Dynamic If-Conversion: The Basic Idea

Apply if-conversion and reverse if-conversion dynamically (at runtime) to complement and correct static compilation decisions.
Dynamic If-Conversion: 
Motivation

• Static if-conversion doesn’t take into account actual runtime behavior

• There is a need for specialized dynamic optimizations – the problems with current runtime optimizations are:
  • High overhead
  • Low quality
  • Low Coverage
  • Overspecialization
Presentation Outline

• Dynamic Optimization Overview
• Case Study: Sampling Correlation
• Dynamic If-Conversion
• Dynamic Reverse If-Conversion
• Conclusions
Dynamic Optimization

- Static
  - HLL
  - Static Optimizations
  - EXE

- Dynamic
  - EXE
  - Dynamic Optimizations
  - Optimization Cache

- Any optimization performed after the initial compile
- Native optimization of a program binary
Motivation for Dynamic Optimization

• Consistency in optimization
• Leverage runtime information
• Personalized optimization
• Scalability
• Complementary optimization opportunity
Study: When Should We Perform Dynamic Optimizations?

- **Timing** is crucial in runtime optimizations
- Because of overhead, we must **sample** the information required to make dynamic optimization decisions
- *But how representative of overall behavior is a sample statistic?*
- Two heuristics were studied:
  - Sampling based on First $N$ Occurrences
  - Adaptive Warmup Exclusion
**First N Occurrences**

- Test correlation of first n occurrences and overall behavior

\[
\begin{array}{cccccc}
M & M & C & M & C & C = 0.60 \\
M & M & C & M & C & M & C & C = 0.55 \\
\end{array}
\]

Branch predictor: PAS/Gshare hybrid
Adaptive Warmup Exclusion

- Recognize an end-of-warmup condition, then collect statistics
Adaptive Warmup Exclusion

\[ |P_{\text{MISS}_A} - P_{\text{MISS}_B}| < T \]

\( P_{\text{MISS}_A} \) = last misprediction rate  
\( P_{\text{MISS}_B} \) = this misprediction rate  
\( T \) = threshold

![Graph showing percent difference vs. sample size for various programs such as compress, go, jpeg, li, m88ksim, perl, vortex, gcc, and average.](image)
Adaptive Warmup Exclusion

Number of branch occurrences before reaching end-of-warmup condition
Problem with Static If-Conversion

Basic Compile-time If Conversion
[ParkSchlansker91]

BEFORE:
if (cond) Branch L1
r2 = MEM[A]
r1 = r2 + 1
r0 = MEM[r1]
L1: r9 = r3 + r4

AFTER:
p1, p2' = cond
(p2) r2 = MEM[A]
(p2) r1 = r2 + 1
(p2) r0 = MEM[r1]
L1: r9 = r3 + r4

Problem: Doesn’t take into account actual runtime behavior
Dynamic If-Conversion

- An optimization that can be performed \textit{at runtime}
- Can be implemented in the optimization pass of any modern dynamic optimizer
- Dynamic version of static if-conversion
  - Takes into account \textit{actual} branch/predicate behavior
- \textbf{Complements} static if-conversion
Dynamic If-Conversion

• Some portions of code may not have been if-converted at compile time, but would benefit from it at runtime

• The Criteria:

\[ P_{\text{MISS}} \times L_{\text{MISS}} \geq P_{\text{FALSE}} \times L_{\text{FALSE}} \times (1+\text{error}) \]

\[ P_{\text{MISS}} = \text{odds of mispredicting branch} \]
\[ L_{\text{MISS}} = \text{misprediction penalty} \]
\[ P_{\text{FALSE}} = \text{odds of a false predicate} \]
\[ L_{\text{HIT}} = \text{cycles to execute predicated instructions} \]
Maximum Branch Distance

- The Maximum Allowable Branch Distance

\[ A_T - A_B < L_{MISS} \times P_{MISS} \times S_{INSTR} \]

\[ A_T - A_B > 0 \]

\( A_T = \) target address \( A_B = \) branch address

\( L_{MISS} = \) miss penalty \( P_{MISS} = \) miss rate \( S_{INSTR} = \) instr size

20 cycles
Branches Converted to Predicates

- **EPIC-style** execution-driven simulator
- Scheduled using the LEGO backend compiler (based on HPL PlayDoh Architecture)
- Most modern static optimizations including static if-conversion
Speedup – Dynamic If-Conversion

- Compared to *statically if-converted* code
- Includes *overhead*
  - Order of 10’s of clock cycles (for a 6-wide machine)
  - Dependent on number of instructions converted
Mispredictions Eliminated

![Bar chart showing mispredictions eliminated for various programs.](image-url)
Dynamic Reverse If-Conversion

- Sometimes it is better to branch over instructions whose predicates are predominantly false
- Correct biased predicates by converting them back to branches

\[ P'_{\text{pred}} \cdot L_{\text{pred}} \geq P_{\text{miss}} \cdot L_{\text{miss}} \]

- \( P'_{\text{pred}} \): odds of false predicate
- \( L_{\text{pred}} \): number of predicated cycles
- \( P_{\text{miss}} \): odds of mispredict
- \( L_{\text{miss}} \): misprediction penalty

\[ p3 = \text{false if cond} \]
\[ (p3) \text{ add } r1=r2, r3 \]
\[ (p3) \text{ mul } r2=r1, r3 \]
\[ (p3) \text{ ld } r1, (r2) \]
\[ (p3) \text{ st } (r3), r2 \]

\[ \rightarrow \]
\[ p3 = \text{false if cond} \]
\[ (!p3) \text{ br label} \]
\[ \text{add } r1=r2, r3 \]
\[ \text{mul } r2=r1, r3 \]
\[ \text{ld } r1, (r2) \]
\[ \text{st } (r3), r2 \]
\[ \text{label:} \]
Predicates Converted to Branches

- compress: 3
- go: 121
- jpeg: 0
- li: 18
- m88ksim: 3
- perl: 28
- vortex: 22
- average: 27.86
Speedup – Reverse If-Conversion

![Bar chart showing speedup values for different programs and their harmonic mean. The chart includes the following programs: compress, go, jpeg, hi, m8gksim, perl, vortex, and harmonic mean. The speedup values range from 1.00 to 1.14. The bar heights represent the speedup values for each program.]
Conclusions

- Dynamic optimization allows for a level of **customized** optimization that is not possible with traditional compilation models.
- By **skipping the warmup period**, we can achieve higher sampling accuracy.
- Dynamic if-conversion is a **worthwhile** dynamic optimization.
- More **runtime algorithm research** is necessary!
Contact Information

Kim Hazelwood  kim_hazelwood@ncsu.edu
Tom Conte      conte@ncsu.edu

Tinker Research Group
NC State University
www.tinker.ncsu.edu