NC STATE UNIVERSITY

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

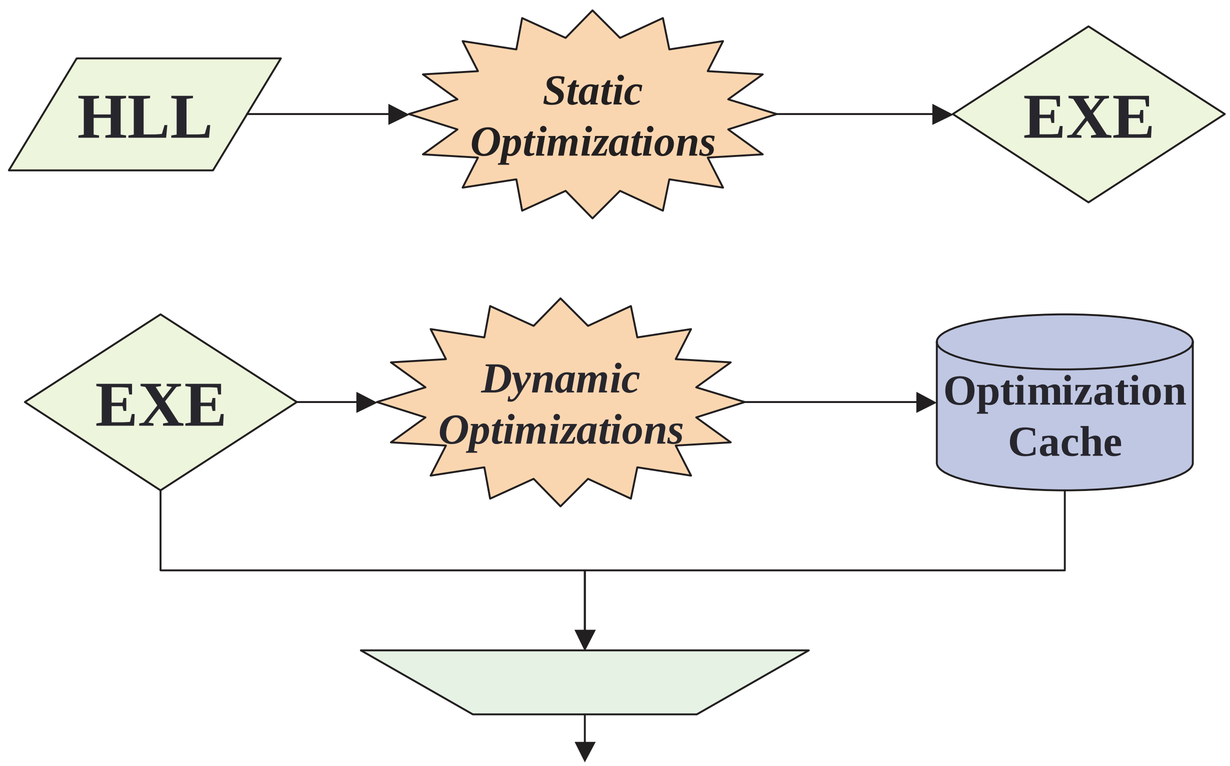
- Static if-conversion doesn't take into account actual runtime behavior
- There is a need for specialized dynamic optimizations – the problems with current runtime optimizations are:
  - High overhead
  - Low Coverage
  - Low quality
  - Overspecialization

Presentation Outline

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

Dynamic Optimization

- Static
- Dynamic



- 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

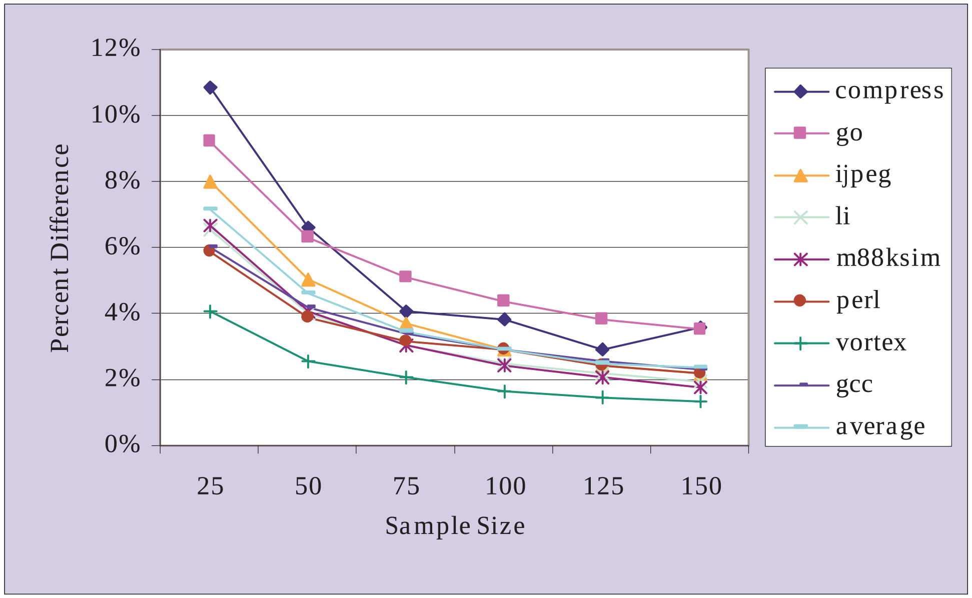
NC STATE UNIVERSITY

First N Occurrences

- Test correlation of first n occurrences and overall behavior

M	M	C	M	C	= 0.60

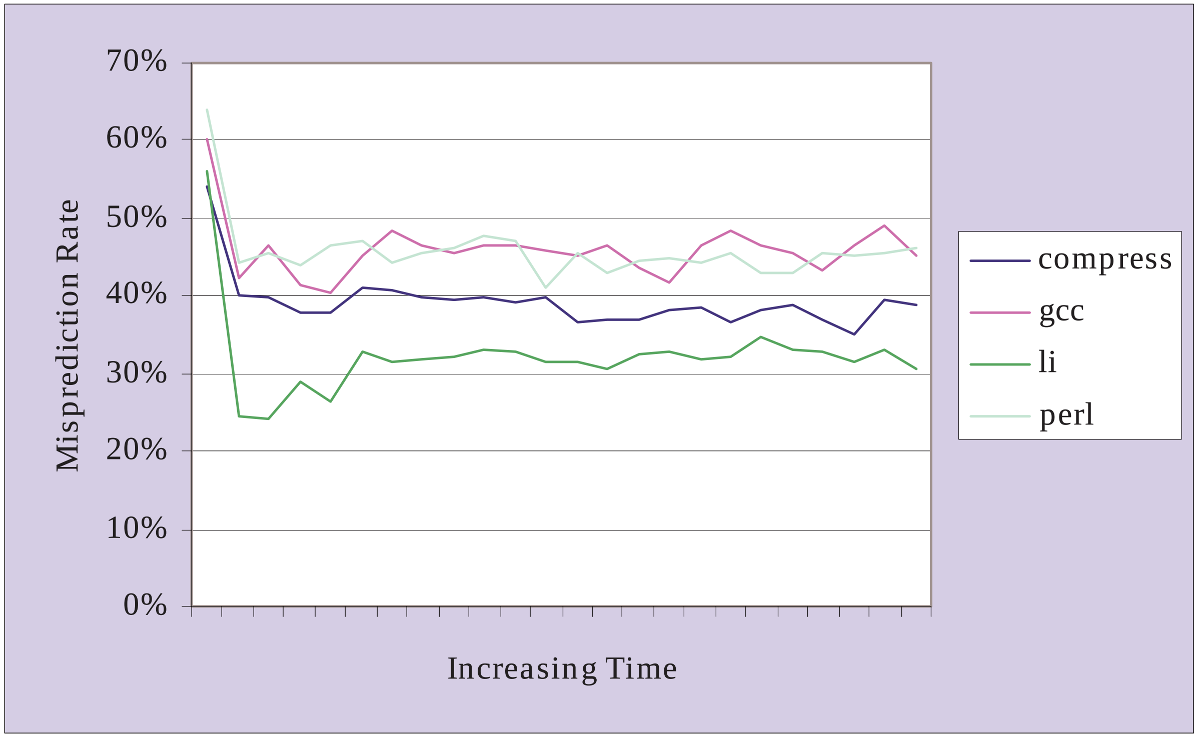
M	M	C	M	C	M	○ ○ ○	C	C	= 0.55



Branch predictor: PAS/Gshare hybrid

Adaptive Warmup Exclusion

- Recognize an end-of-warmup condition, then collect statistics



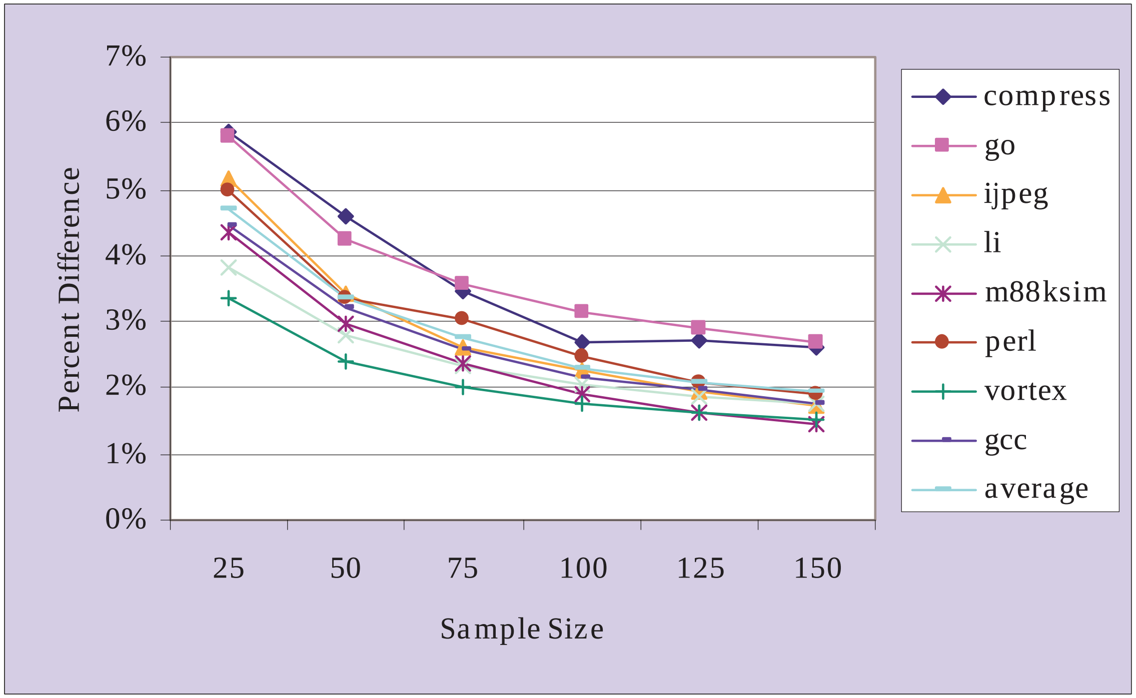
NC STATE UNIVERSITY

Adaptive Warmup Exclusion

$$| P_{MISS\_A} - P_{MISS\_B} | < T$$

P_{MISS_A} = last misprediction rate P_{MISS_B} = this misprediction rate

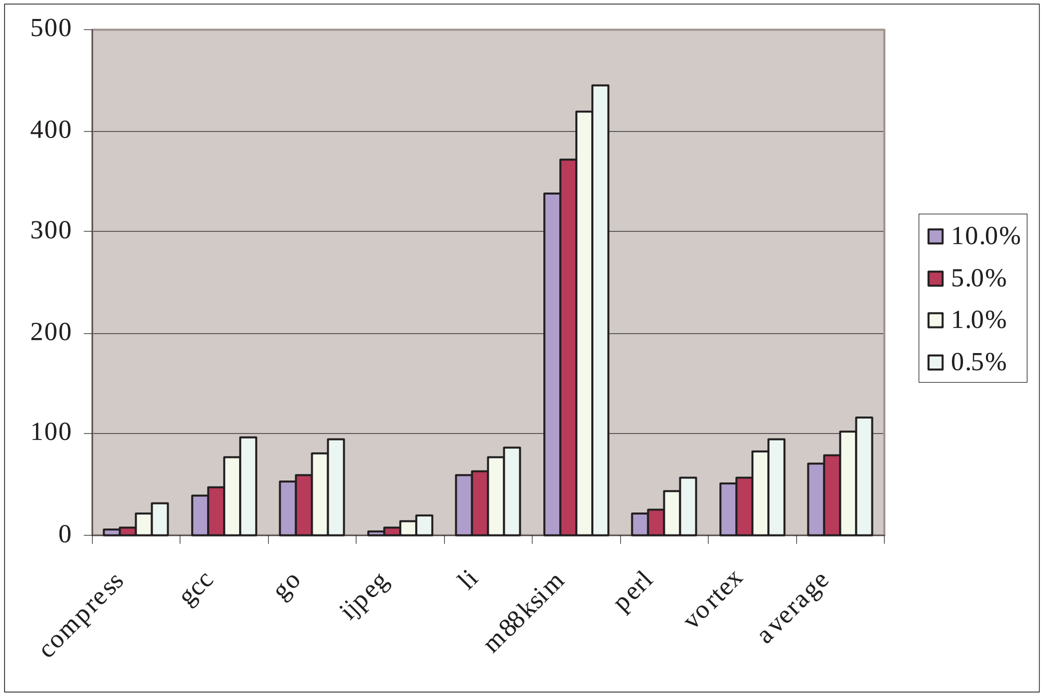
T = threshold



NC STATE UNIVERSITY

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

Problem: Doesn't take into account actual runtime behavior

Dynamic If-Conversion

- An optimization that can be performed at runtime
- Can be implemented in the optimization pass of any modern dynamic optimizer
- Dynamic version of static if-conversion
  - Takes into account actual branch / predicate behavior
- 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:

$$P_{MISS} * L_{MISS} \geq P_{FALSE} * L_{FALSE} * (1+error)$$

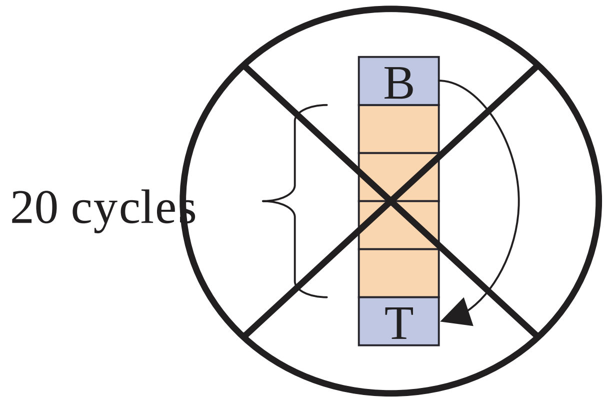
P_{MISS} = odds of mispredicting branch

L_{MISS} = misprediction penalty

P_{FALSE} = odds of a false predicate

L_{HIT} = cycles to execute predicated instructions

Maximum Branch Distance



- The Maximum Allowable Branch Distance

$$A_T - A_B < L_{MISS} * P_{MISS} * S_{INSTR}$$

$$A_T - A_B > 0$$

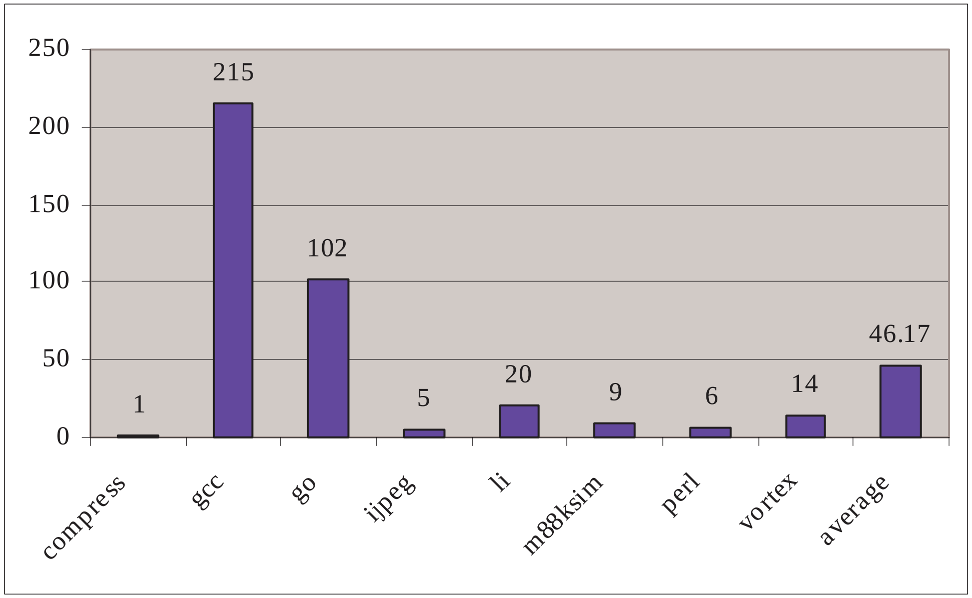
A_T = target address A_B = branch address

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

NC STATE UNIVERSITY

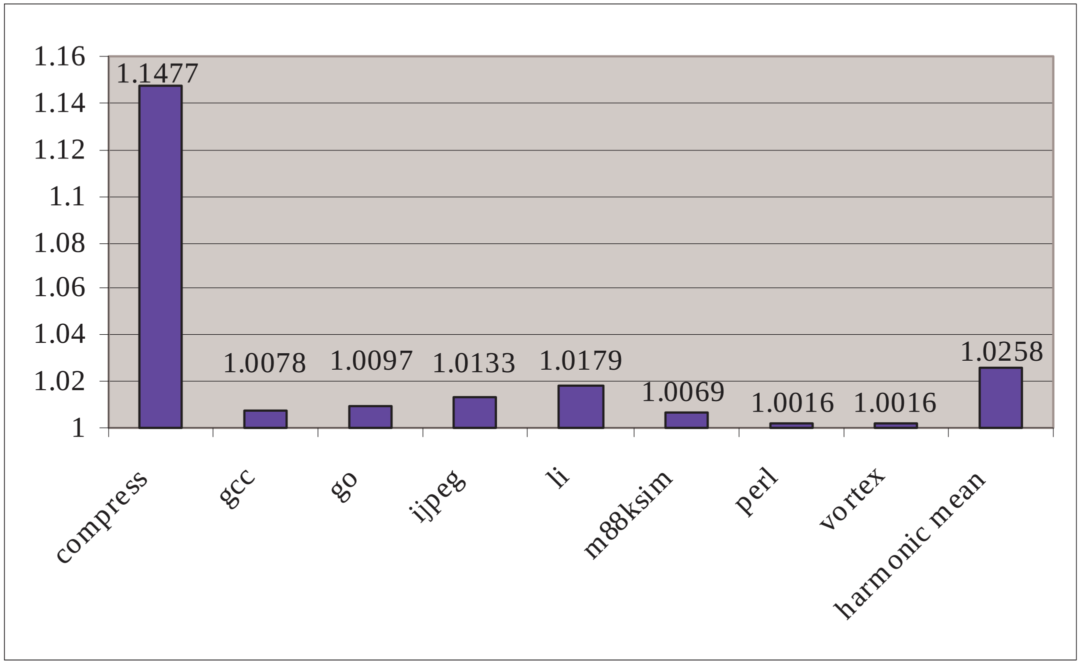
Branches Converted to Predicates

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



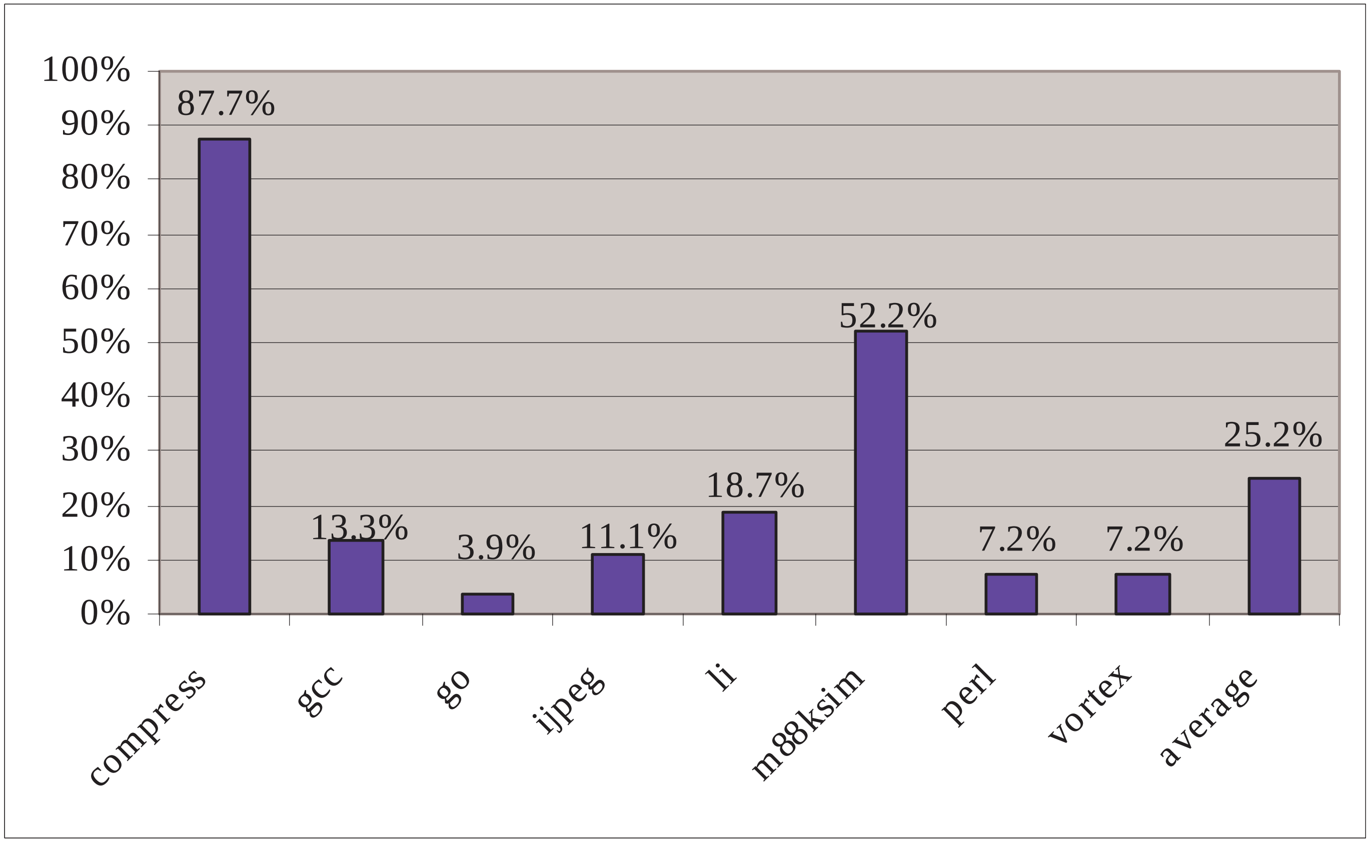
Speedup – Dynamic If-Conversion

- 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



NC STATE UNIVERSITY

Mispredictions Eliminated



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

$$P_{PRED'} * L_{PRED} \geq P_{MISS} * L_{MISS}$$

$P_{PRED'}$ = odds of false predicate
P_{MISS} = odds of mispredict

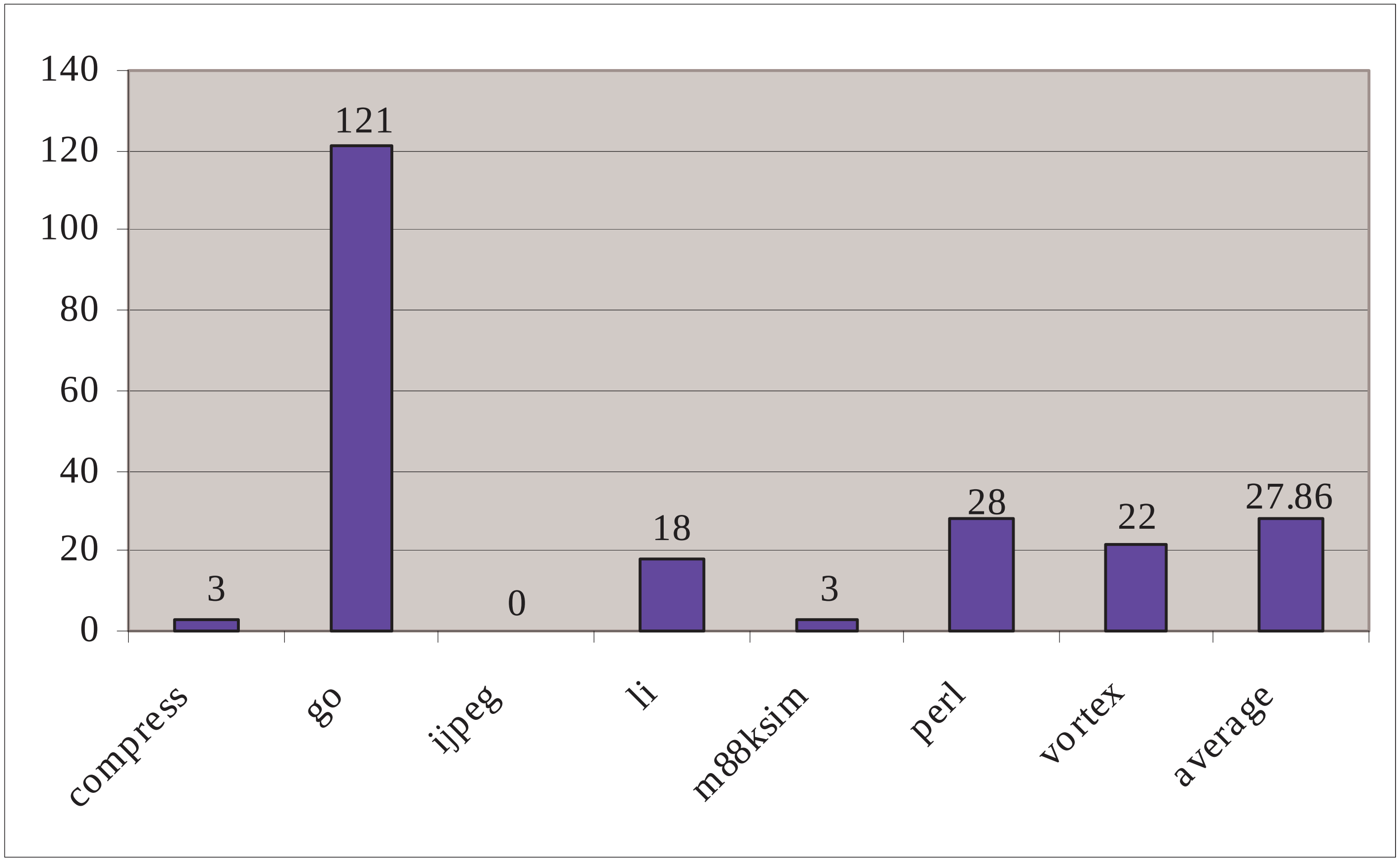
L_{PRED} = number of predicated cycles
L_{MISS} = misprediction penalty

```
p3 = false if cond
(p3) add r1=r2,r3
(p3) mul r2=r1,r3
(p3) ld r1, (r2)
(p3) st (r3), r2
```

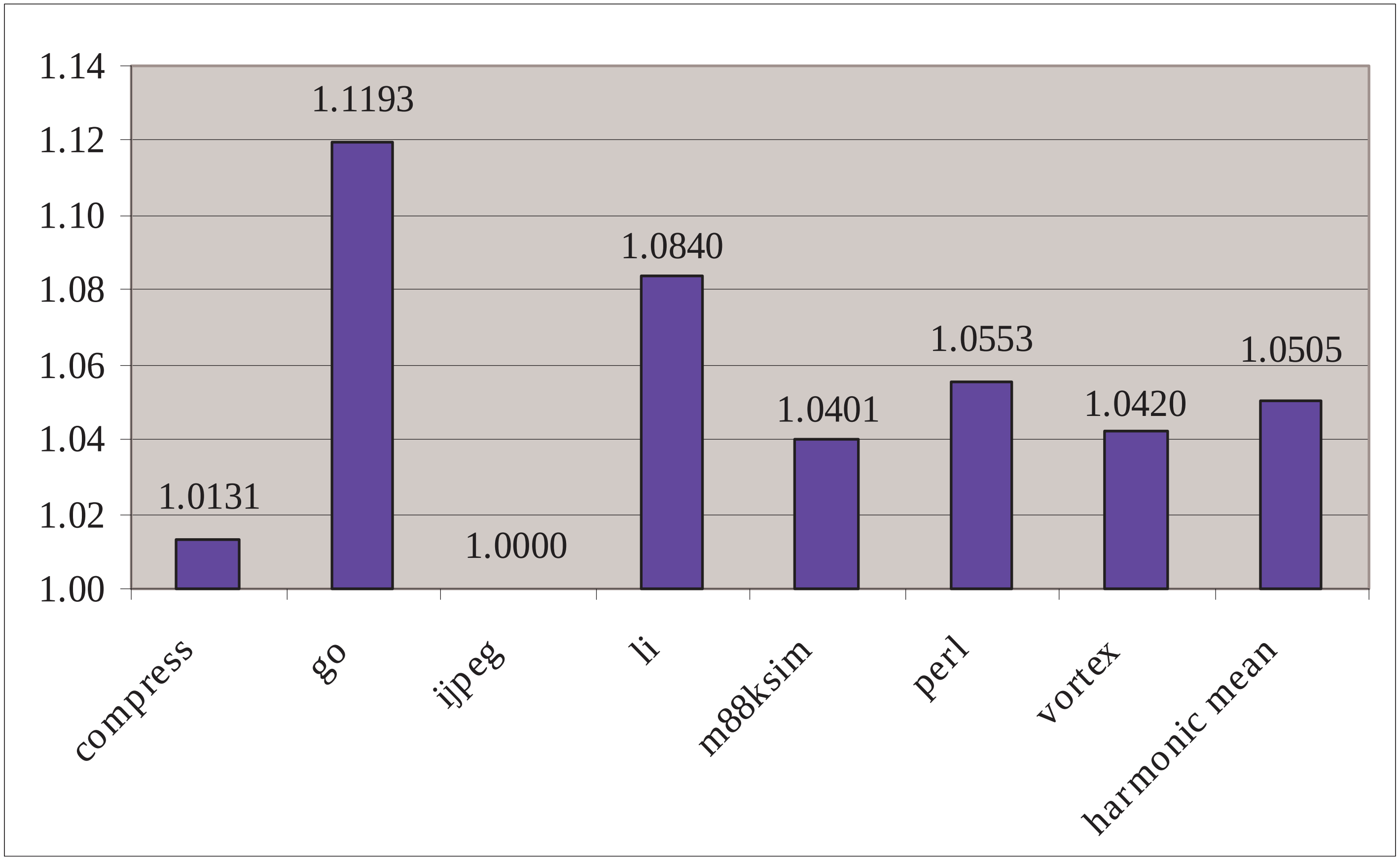
→

```
p3 = false if cond
(!p3) br label
add r1=r2,r3
mul r2=r1,r3
ld r1, (r2)
st (r3), r2
label:
```

Predicates Converted to Branches



Speedup – Reverse If-Conversion



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!

NC STATE UNIVERSITY

Contact Information

Kim Hazelwood kim_hazelwood@ncsu.edu

Tom Conte conte@ncsu.edu

Tinker Research Group

NC State University

www.tinker.ncsu.edu