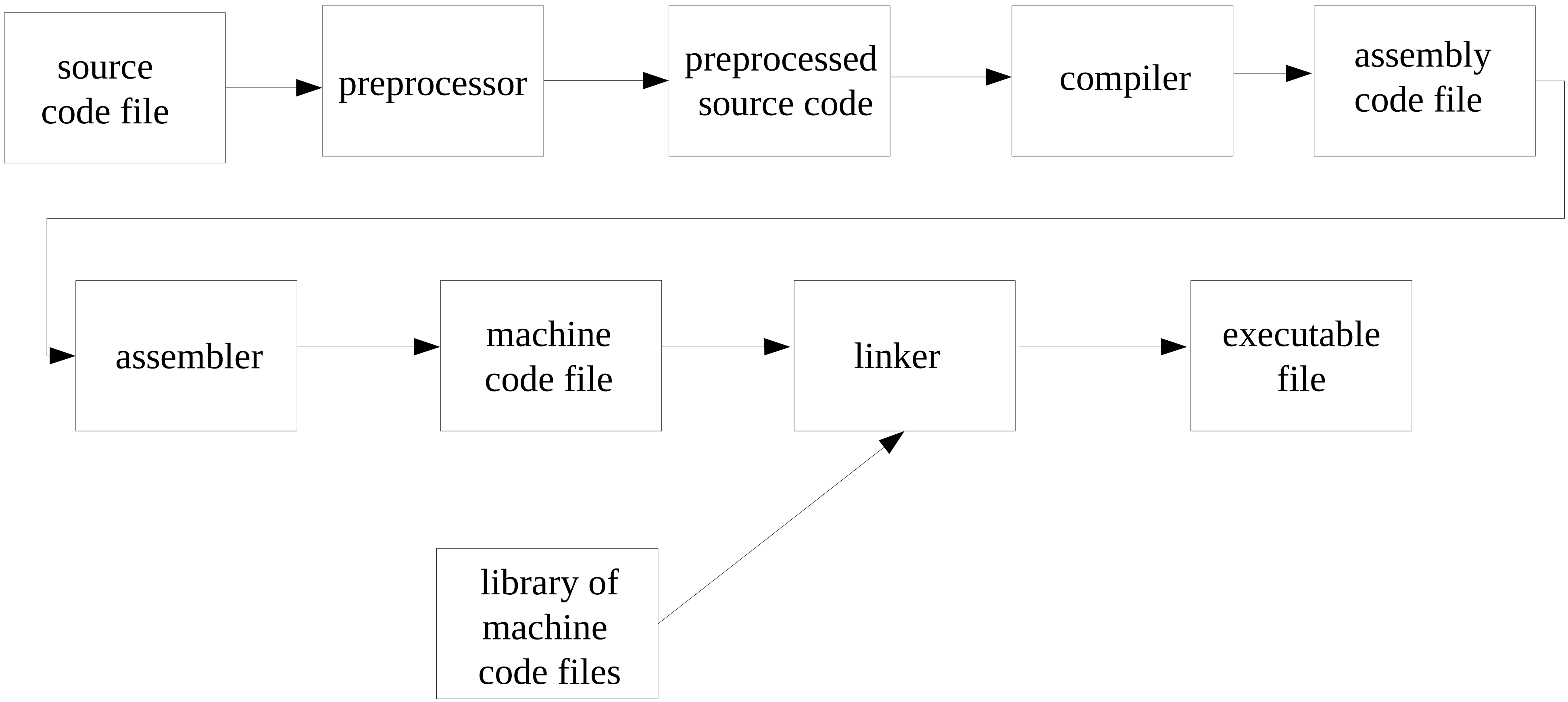
Branch Elimination via Multi-Variable Condition Merging

William Kreahling

Outline of Presentation

- **Background and tools.**
- Introduce technique for multi-variable condition merging.
- Rules for merging multiple variables.
- Describe framework for performing the analysis.
- Results & Conclusion.

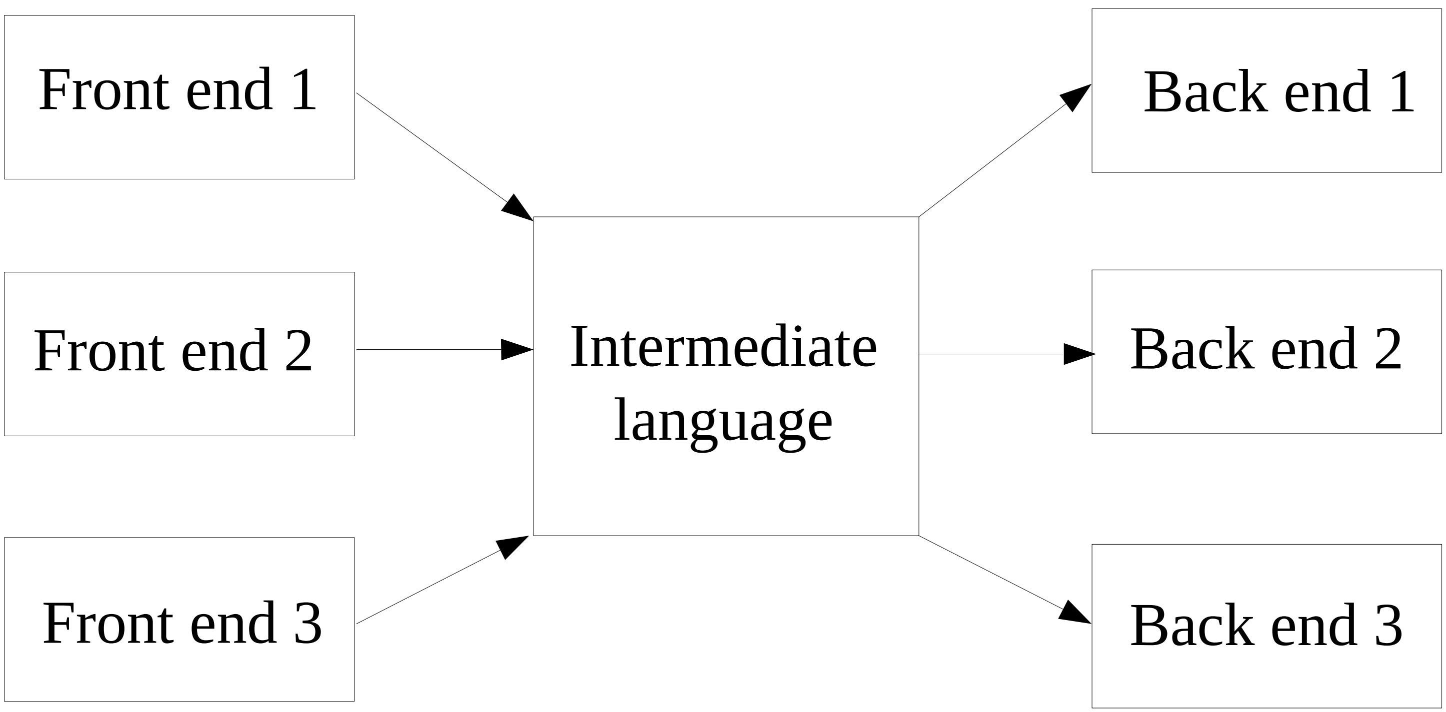
General Compilation Process



Front and Back Ends

- Front end
  - takes high-level source code as input
  - produces intermediate code as output
- Back end
  - takes intermediate code as input
  - produces assembly or machine code

Intermediate Language



Very Portable Optimizer (VPO)

- Research Compiler
- Very simple front end
- Translates source code into intermediate language.

Register Transfer Lists

- “Generic assembly language”
- Used by many compilers including GCC.
- All transformations in VPO done at the RTL level.

Example RTLs

– r[2] = r[4] -- move

– r[5] = r[6] + r[7] -- add add r5,r6,r7

– M[r[2] + 4] = r[3] -- store

– IC = r[2] ? r[3] -- comparison

Outline of Presentation

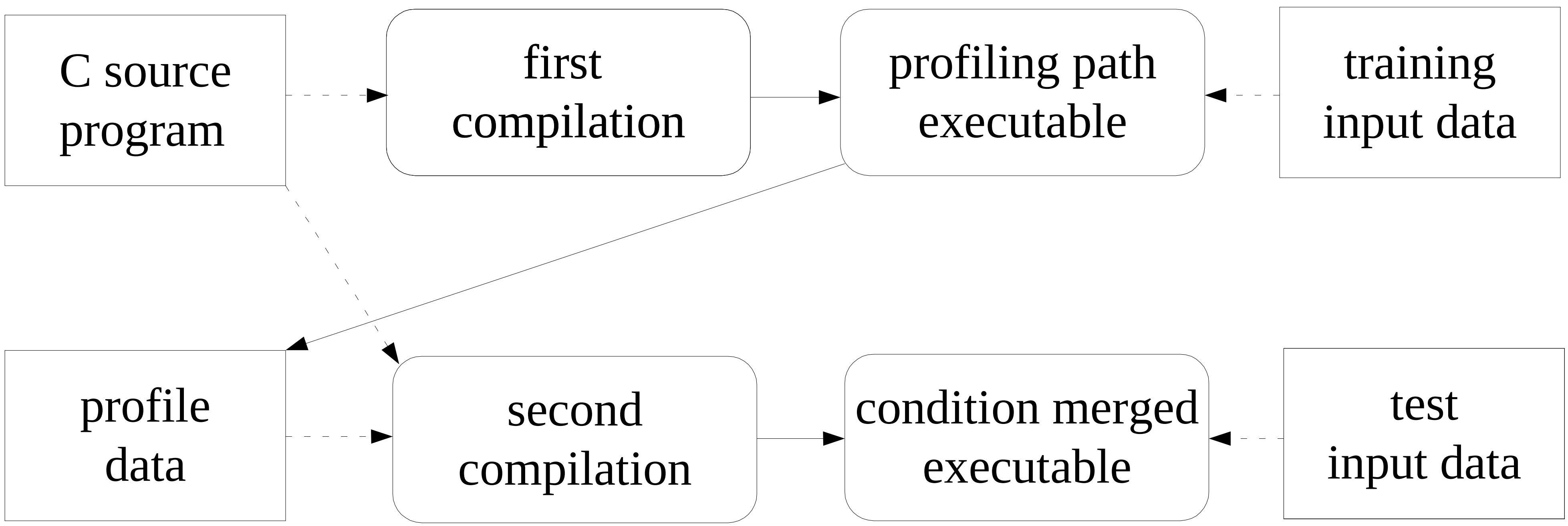
- Background and tools.
- **Introduce technique for multi-variable condition merging.**
- Rules for merging multiple variables.
- Describe framework for performing the analysis.
- Results & Conclusion.

Motivation

- The execution of conditional branches is expensive.
- Causes pipeline flushes.
- Inhibits other code improving transformations.

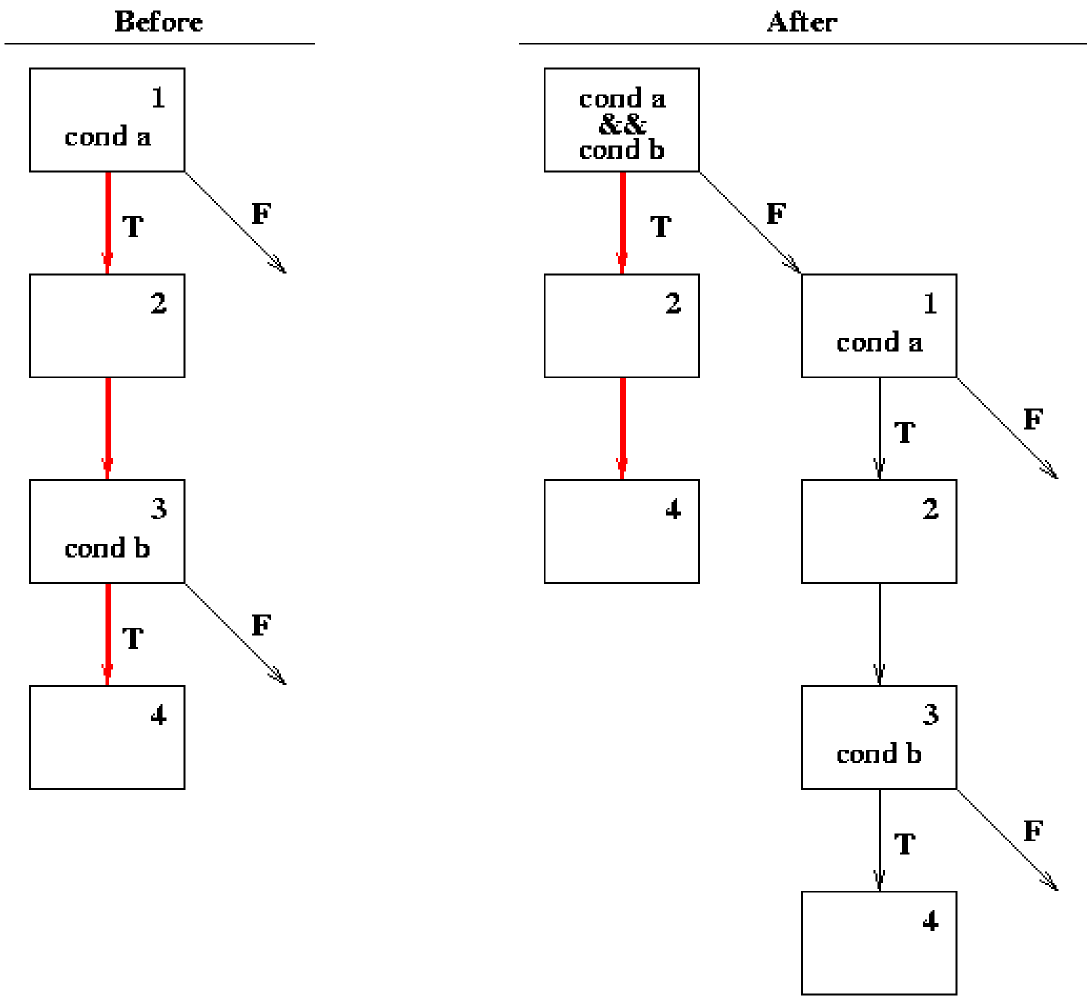
IF	ID	EX	MEM	WB

Compilation Process for Condition Merging



Condition Merging

- Merging branches means replacing the execution of two or more branches with the execution of a single branch.



Tests Involving Multiple Variables

- How can conditions that test the results of comparisons of multiple variables be merged?

Tests Involving Multiple Variables

- How can conditions that test the results of comparisons of multiple variables be merged?
  - Consider comparisons to zero.
  - Consider the use of logical operations.

Tests Involving Multiple Variables (Equal to Zero)

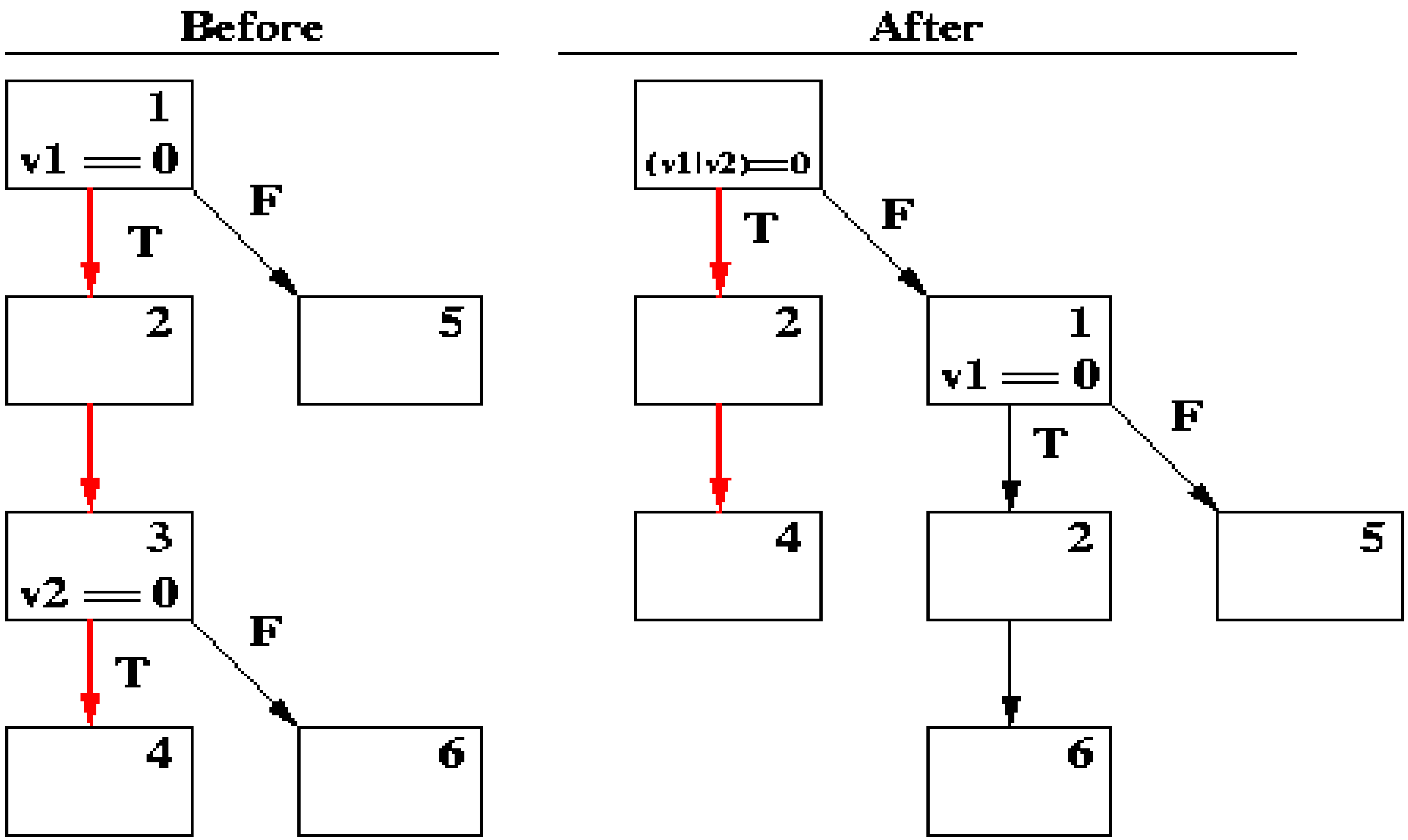
`v1==0 && v2==0` $\leftrightarrow$ **`??`**

Tests Involving Multiple Variables (Equal to Zero)

```
v1==0 && v2==0 ↔ (v1|v2)==0
```

Tests Involving Multiple Variables (Equal to Zero)

`v1==0 && v2==0` ↔ `(v1|v2)==0`



Checking If Multiple Variables Are Equal to Zero (cont.)

- n sets of compares and branches are replaced by n-1 logical operations and 1 branch

```
r[t]=r[1]|r[2];
r[t]=r[t]|r[3];
...
IC=(r[t]|r[n])?0;
PC=IC!=0,<bypass target>;
```

Outline of Presentation

- Background and tools.
- Introduce technique for multi-variable condition merging.
- **Rules for merging multiple variables.**
- Describe framework for performing the analysis.
- Results & Conclusion.

Merging Conditions Comparing Multiple Variables to 0 or -1

v1== 0 && v2== 0 ↔ (v1| v2)==0

v1== 0 && v2==-1 ↔

Merging Conditions Comparing Multiple Variables to 0 or -1

```
v1== 0 && v2== 0 ↔ (v1| v2)==0
v1== 0 && v2==-1 ↔ (v1|~v2)==0
```

Merging Conditions Comparing Multiple Variables to 0 or -1

```
v1== 0 && v2== 0 ↔ (v1| v2)==0

v1== 0 && v2==-1 ↔ (v1|~v2)==0

v1 < 0 && v2 < 0 ↔
```

Merging Conditions Comparing Multiple Variables to 0 or -1

```
v1== 0 && v2== 0 ↔ (v1| v2)==0
v1== 0 && v2==-1 ↔ (v1|~v2)==0
v1 < 0 && v2 < 0 ↔ (v1& v2)< 0
```

Merging Conditions Comparing Multiple Variables to 0 or -1

```
v1== 0 && v2== 0 ↔ (v1| v2)==0

v1== 0 && v2==-1 ↔ (v1|~v2)==0

v1 < 0 && v2 < 0 ↔ (v1& v2)< 0

v1>= 0 && v2>= 0 ↔
```

Merging Conditions Comparing Multiple Variables to 0 or -1

```
v1== 0 && v2== 0 ↔ (v1| v2)==0
v1== 0 && v2==-1 ↔ (v1|~v2)==0
v1 < 0 && v2 < 0 ↔ (v1& v2)< 0
v1>= 0 && v2>= 0 ↔ (v1| v2)>=0
```

Merging Conditions Comparing Multiple Variables to 0 or -1

```
v1== 0 && v2== 0 ↔ (v1| v2)==0
v1== 0 && v2==-1 ↔ (v1|~v2)==0
v1 < 0 && v2 < 0 ↔ (v1& v2)< 0
v1>= 0 && v2>= 0 ↔ (v1| v2)>=0
v1 < 0 && v2>= 0 ↔
```

Merging Conditions Comparing Multiple Variables to 0 or -1

```
v1== 0 && v2== 0 ↔ (v1| v2)==0

v1== 0 && v2==-1 ↔ (v1|~v2)==0

v1 < 0 && v2 < 0 ↔ (v1& v2)< 0

v1>= 0 && v2>= 0 ↔ (v1| v2)>=0

v1 < 0 && v2>= 0 ↔ (v1&~v2)< 0
```

Merging Conditions Comparing Multiple Variables to 0 or -1

```
v1== 0 && v2== 0 ↔ (v1| v2)==0
v1== 0 && v2==-1 ↔ (v1|~v2)==0
v1 < 0 && v2 < 0 ↔ (v1& v2)< 0
v1>= 0 && v2>= 0 ↔ (v1| v2)>=0
v1 < 0 && v2>= 0 ↔ (v1&~v2)< 0
v1>= 0 && v2 < 0 ↔
```

Merging Conditions Comparing Multiple Variables to 0 or -1

```
v1== 0 && v2== 0 ↔ (v1| v2)==0
v1== 0 && v2==-1 ↔ (v1|~v2)==0
v1 < 0 && v2 < 0 ↔ (v1& v2)< 0
v1>= 0 && v2>= 0 ↔ (v1| v2)>=0
v1 < 0 && v2>= 0 ↔ (v1&~v2)< 0
v1>= 0 && v2 < 0 ↔ (v1|~v2)>=0
```

Are Multiple Variables != to Zero?

- How can we detect if n variables are all not equal to zero in a single branch?

Are Multiple Variables != to Zero?

- How can we detect if n variables are all not equal to zero in a single branch?
  - Are one or more bit positions in all of the variables set?

(v1&v2)!=0 → v1!=0 && v2!=0

Are Multiple Variables != to Zero?

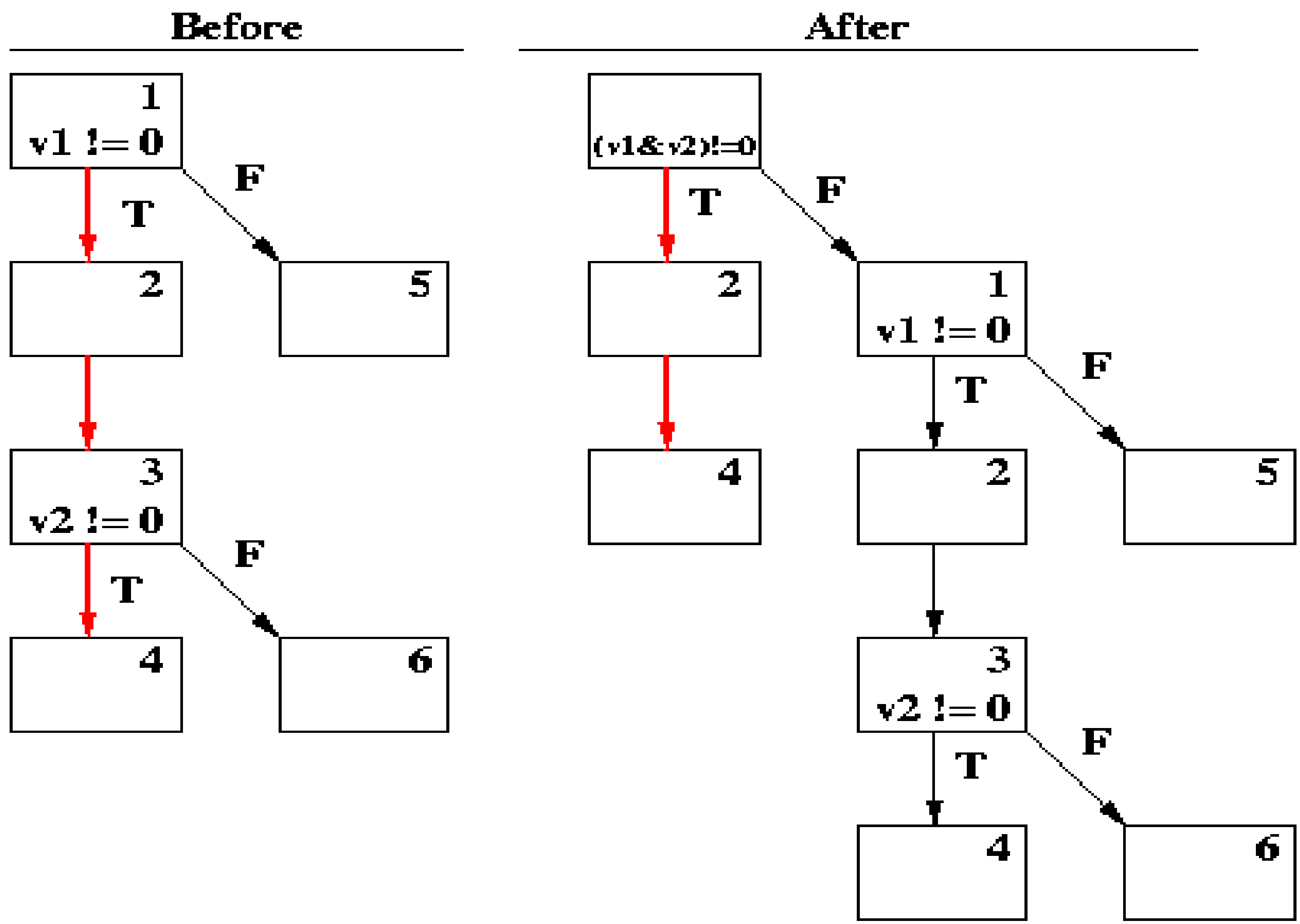
- How can we detect if n variables are all not equal to zero in a single branch?
  - Are one or more bit positions in all of the variables set?

    `(v1&v2)!=0` $\rightarrow$ **`v1!=0 && v2!=0`**

  - example:

```
if (p1 && p2 && p1->val == p2->val)
```

Checking If Multiple Variables Are Not Equal to Zero (cont.)



Checking If Multiple Variables Are Not Equal to Multiple Constants

- How can we check if n variables are all not equal to n constants in a single branch?

Checking If Multiple Variables Are Not Equal to Multiple Constants (cont.)

- Is a bit position set in all of the variables and is the same bit position clear in all of the constants?

`(v1&v2)&~(c1|c2)!=0` → `v1!=c1 && v2!=c2`

- Say v1=6, v2=10, c1=9, and c2=5
- $(0110_2 \ \& \ 1010_2) \ \& \ {\sim}(1001_2 \ | \ 0101_2) =$

$(0010_2)\&{\sim}(1101_2) = (0010_2)\&(0010_2) = 0010_2 \ != 0$

Checking If Multiple Variables Are Not Equal to Multiple Constants (cont.)

- Consider the case where the constants are all zero.

(v1&v2)&~(c1|c2)!=0 → v1!=c1 && v2!=c2

(v1&v2)&~(0|0)!=0

(v1&v2)&0xFFFFFFFF != 0

(v1&v2) != 0

Checking If Multiple Variables Are Not Equal to Multiple Constants (cont.)

- Is a bit position clear in all of the variables and is the same bit position set in all of the constants?

`~(v1|v2)&(c1&c2)!=0` → `v1!=c1 && v2!=c2`

- Say v1=6, v2=10, c1=9, and c2=5
- $\sim(0110_2 \mid 1010_2)\ \&\ (1001_2\ \&\ 0101_2) =$

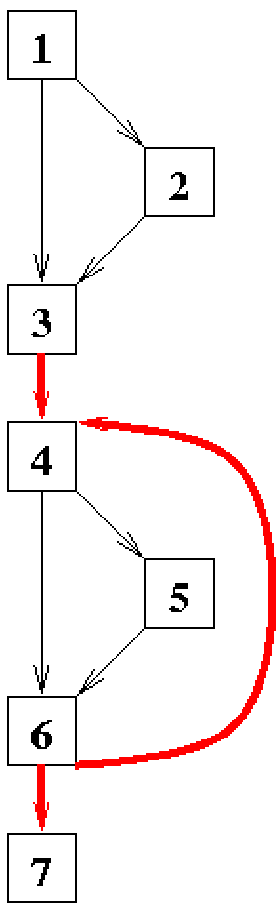
$\sim(1110_2)\&(0001_2) = (0001_2)\&(0001_2) = 0001_2\ != 0$

Outline of Presentation

- Background and tools.
- Introduce technique for multi-variable condition merging.
- Rules for merging multiple variables.
- **Describe framework for performing the analysis.**
- Results & Conclusion.

Framework for Obtaining Path Profile Information

- Detect paths dynamically during a profile run.
- Paths do not cross loop boundaries.



Using the Path Profile Information

- Accumulate statistics and estimate the benefit to merge each set.
  - likelihood that the path will be taken given that the first branch is reached
  - instructions executed if the dominant path is taken
  - instructions executed if the dominant path is not taken
- Merge sets in the order of the most beneficial sets of branches first.

Using the Path Profile Information

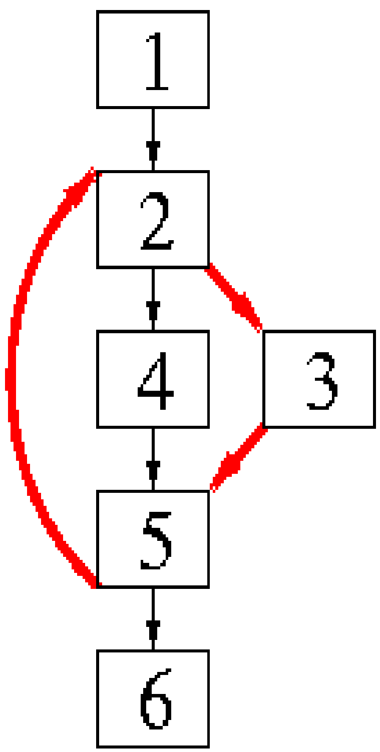
- Rely on other optimizations to improve the modified code.
  - loop-invariant code motion
  - common subexpression elimination

Paths Across Loop Boundaries

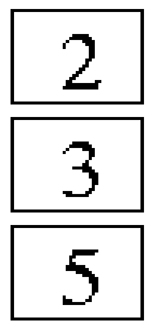
- During profile run, detect a frequently executed path, that is followed by itself.
- Insert a new path into the data structure.
  - The original path, with every block duplicated.
  - Treated as a separate and distinct path.

Paths Across Loop Boundaries

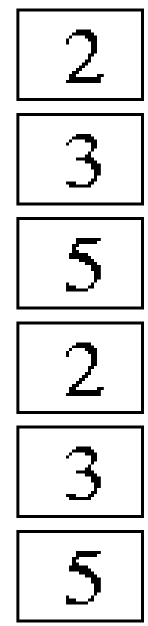
Loop



Frequently executed path



Duplicate path



Choosing Which Branches to Merge

- Once we have detected all the paths, we examine them for mergeable branches and collect these into sets.
- Just because a set is valid, does not mean it would be beneficial to actually merge.
- Possible that the same sets of mergeable branches will have been detected by more than one of the rules.

Choices Driven by Estimates

- Have to estimate the benefit from merging a set of branches.

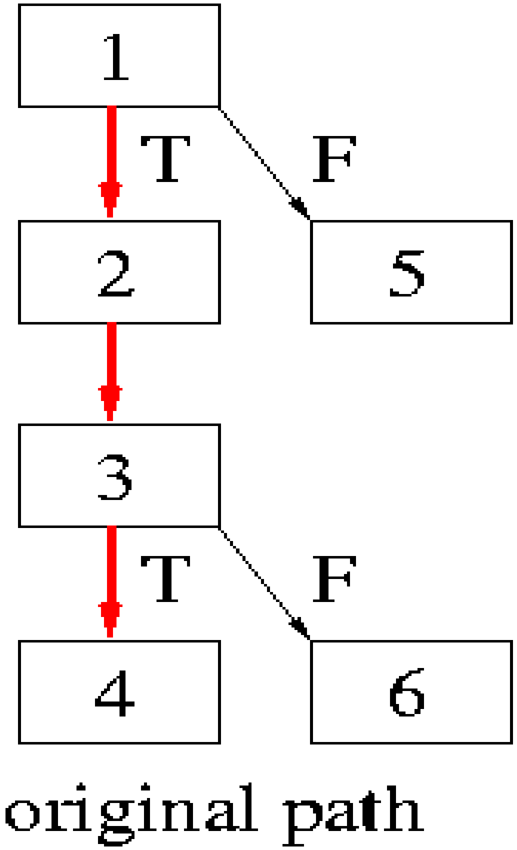
- Generalized formula:
  ((win path ratio * saved instructions) -
  (lose path ratio * added instructions))*
  total execution when the path is reached.

Win/Lose/Breakeven Paths

- Once we have picked a set, we must duplicate code with the new merged branches and their targets.

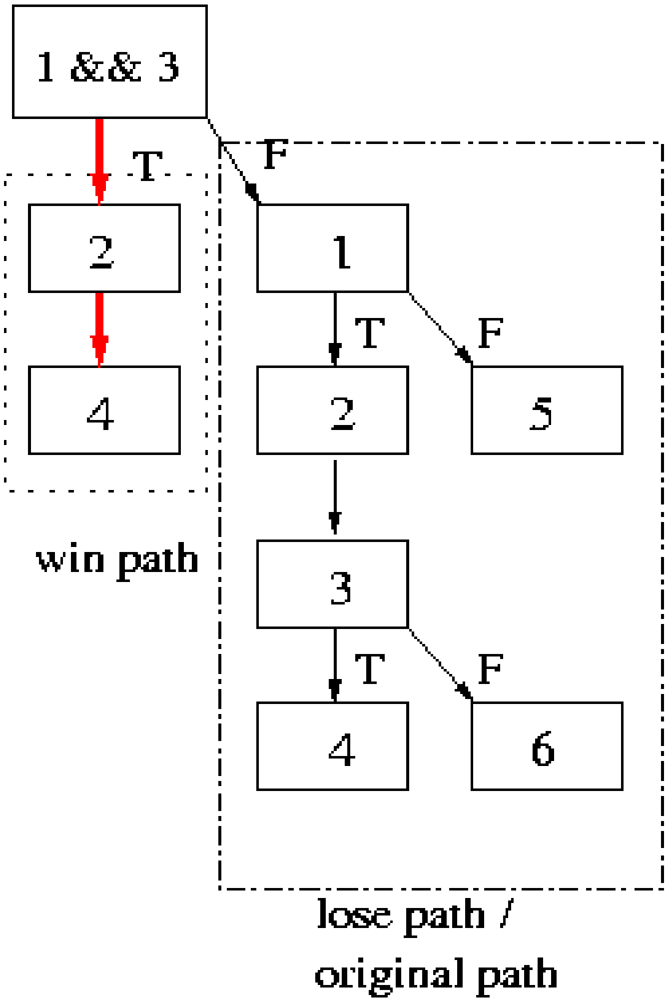
- With many paths we will create a win path, a lose path, and a breakeven path.

Example Win/Lose/Breakeven Path



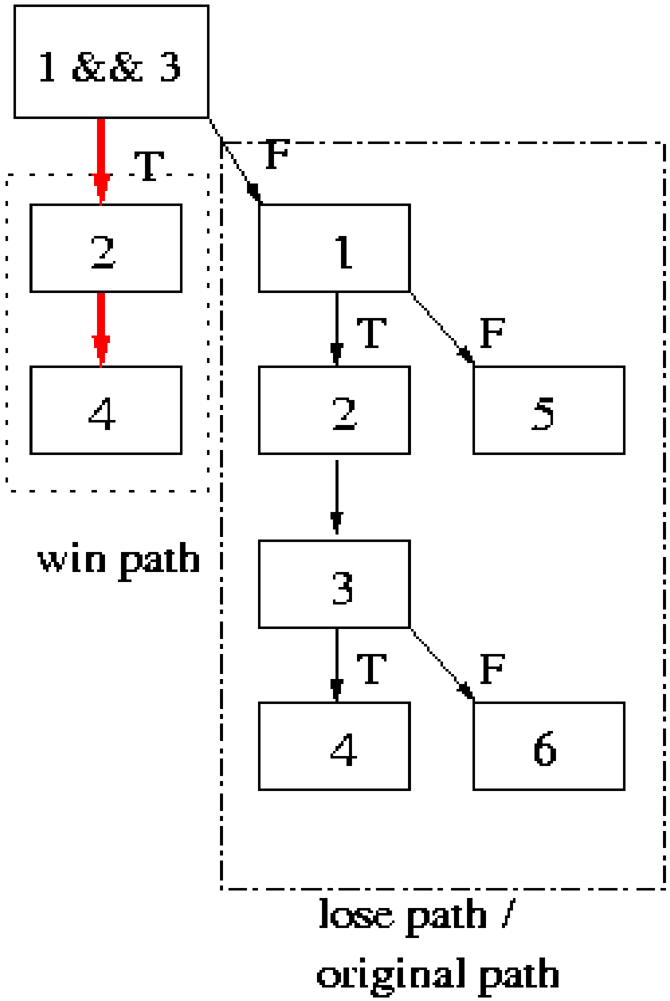
Reach block 6 (original 2)

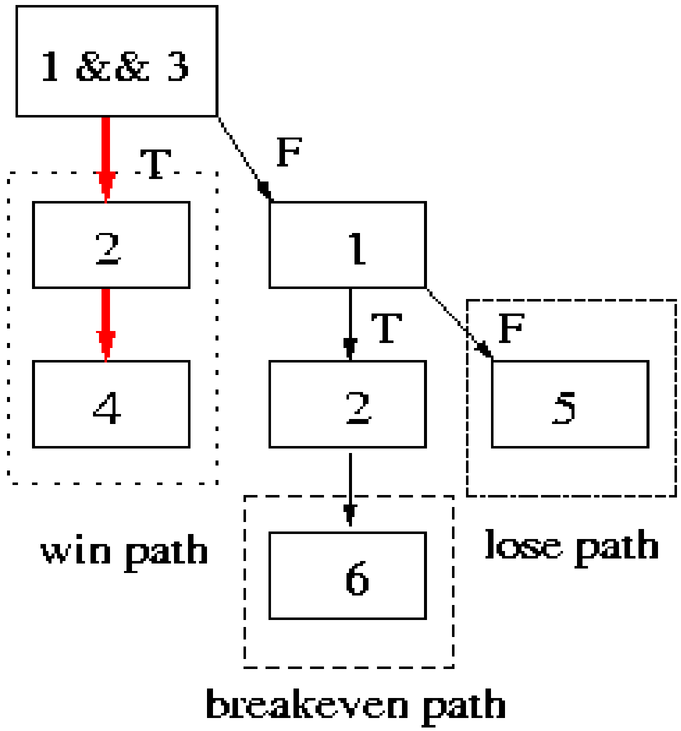
Example Win/Lose/Breakeven Path



Reach block 6 (original 2) (lose 3)

Example Win/Lose/Breakeven Path



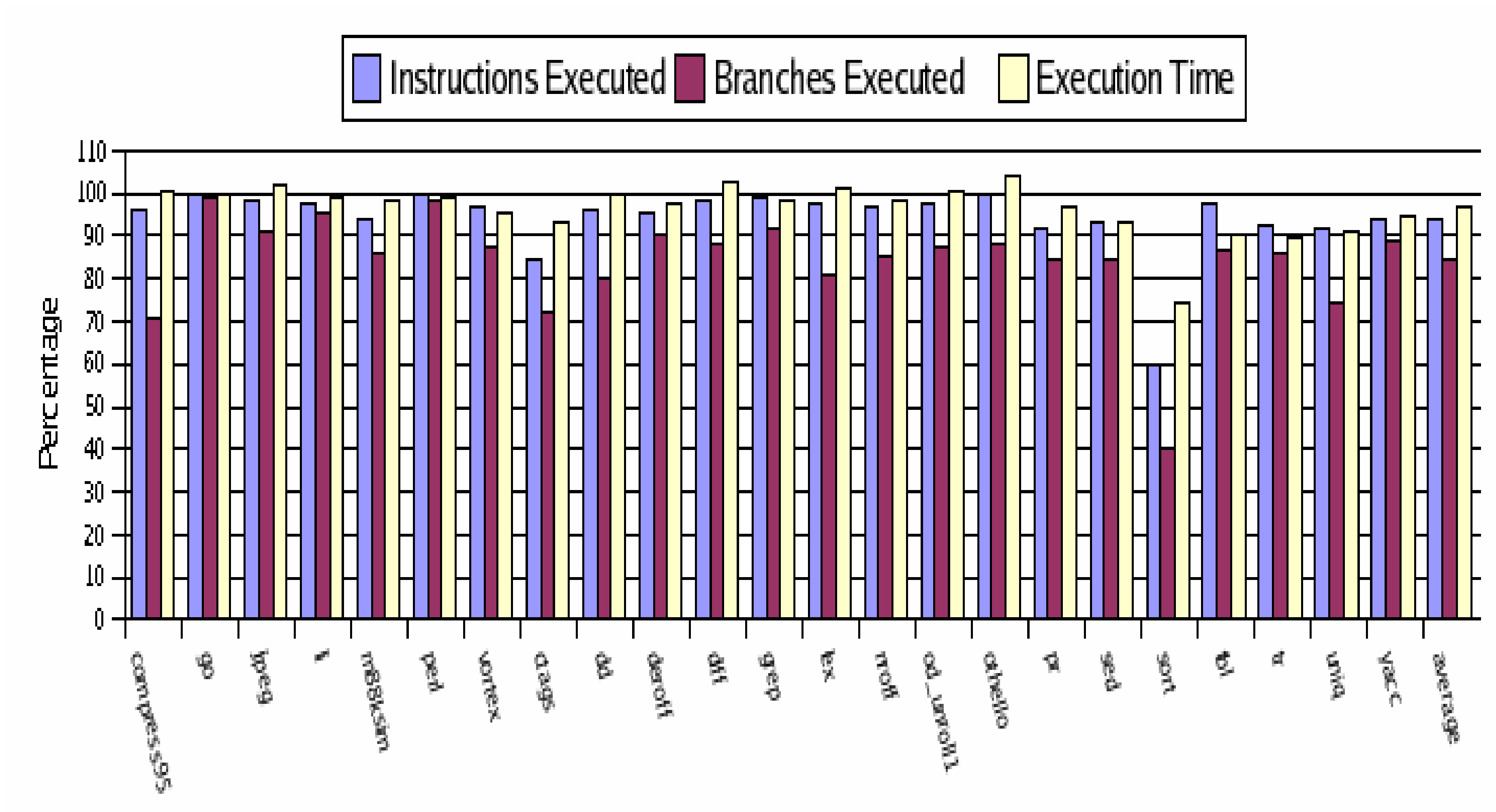


Reach block 6 (original 2) (lose 3) (breakeven 2)

Outline of Presentation

- Background and tools.
- Introduce technique for multi-variable condition merging.
- Rules for merging multiple variables.
- Describe framework for performing the analysis.
- **Results & Conclusion.**

Results



Instructions Executed
Branches Executed
Execution Time
Percentage
0
10
20
30
40
50
60
70
80
90
100
110
compress95
go
ijpeg
li
m88ksim
perl
vortex
ctags
dd
deroff
diff
grep
lex
nroff
od_unroll1
othello
pr
sed
sort
tbl
tr
uniq
yacc
average

Conclusions

- Contributions
  - Able to merge conditions involving multiple variables on a conventional scalar processor.
  - Obtained benefits by merging conditions in paths that are not the most frequent.

Conclusions

- Contributions
  - Able to merge conditions involving multiple variables on a conventional scalar processor.
  - Obtained benefits by merging conditions in paths that are not the most frequent.

Acknowledgments

- Dr. David Whalley
- Dr. Mark Bailey
- Dr. Xin Yuan
- Dr. Gang-Ryung Uh
- Dr. Robert van Engelen

Publications

- “Branch Elimination by Condition Merging” by W. Kreahling et. al., in *Software Practice and Experience*, 35(1), PP 51-74, January 2005.

- “Branch Elimination via Multi-Variable Condition Merging” by W. Kreahling et. al., in *The Proceedings of the Euro-Par '03 Conference on Parallel Processing*, PP 261-270, August 2003.

Current Research

- Reducing Branch costs using “Implicit Comparisons”.
- Hardware modifications.
- New instructions to decouple the comparison definition from the actual comparison of values.