Understanding the Differences Between Value Prediction and Instruction Reuse

Avinash Sodani

Guri Sohi

Computer Science Department

University of Wisconsin-Madison

Introduction

Instructions perform same computation repeatedly

- produce same results again and again ➔ **Redundancy!**

Exploitation ➔ Collapse true dependences

Two techniques

- **Value Prediction (VP):** predict results

  ➔ perform dependent computation in parallel

- **Instruction Reuse (IR):** reuse earlier results

  ➔ avoid performing same computation again

Purpose of this work

Effectiveness of any technique depends on

- how well it performs by itself
- how it interacts with base μ-arch

VP and IR are different techniques

➔ interact differently

Purpose

Understand the differences and their impact

➔ will help in designing better hybrid schemes

Outline

❶ VP and IR ➔ differences

❷ Potential for capturing redundancy

❸ Interactions: Qualitative

❹ Interactions: Some results

In the paper

- More results
- Estimate of fraction of total redundancy captured by IR

VP and IR

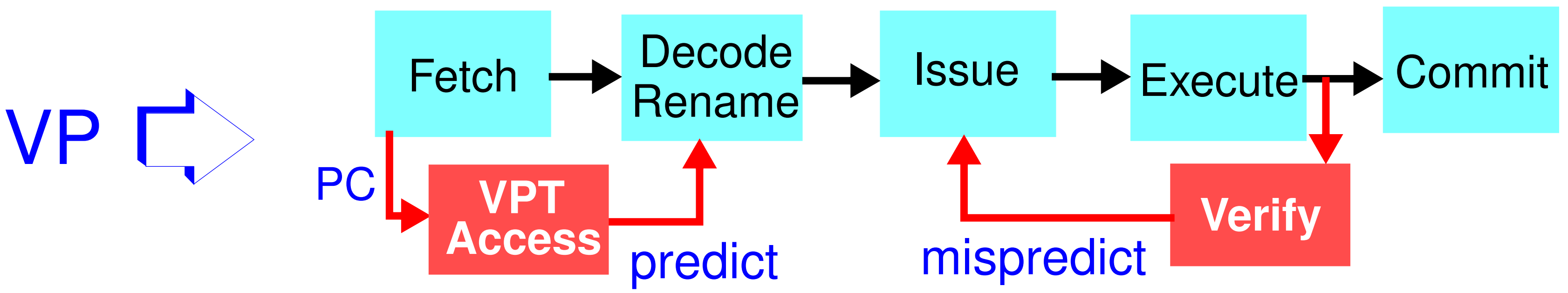
VP: Main idea

- Save instruction results in **VP Table** (VPT)
- Next time
  - predict result ➔ dependent inst. free to execute
  - if mispredicted ➔ re-execute dependent inst.

IR: Main idea

- Save instruction inputs and result in **Reuse Buffer** (RB)
- Next time
  - check RB for instruction entry
  - if inputs same ➔ reuse result ➔ skip computation

Pipelines



VP

Fetch

Decode Rename

Issue

Execute

Commit

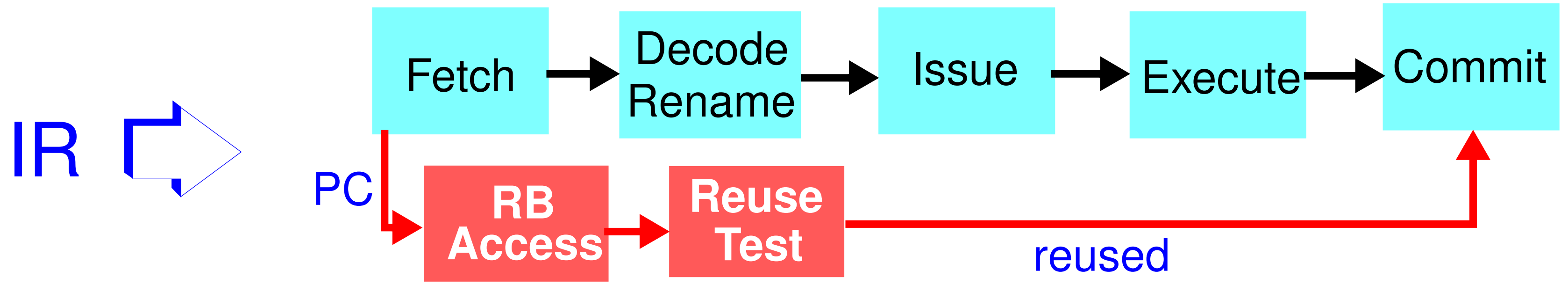
PC

VPT Access

predict

Verify

mispredict



IR

Fetch

Decode Rename

Issue

Execute

Commit

PC

RB Access

Reuse Test

reused

Differences

VP: Verifies results **after** Use ➔ **speculative**

IR : Verifies results **before** Use ➔ **non-speculative**

Due to this, they **differ** in

❶ amount of redundancy they capture

❷ their interaction with base μ-arch

➔ hence differ in performance gained

Potential for Capturing Redundancy

IR: conservative

- inst not reused if
    - inputs not ready, or
    - they are different

VP: aggressive

- can correctly predict in above cases

VP captures more redundancy than IR

Outline

~~❶ VP and IR ➔ differences~~

~~❷ Potential for capturing redundancy~~

❸ Interactions: Qualitative

❹ Interactions: Some Results

μ-arch Interactions

VP and IR have different impacts on

- Branch prediction
- Contention for resources (FU, cache ports, etc.)
- Execution latency of inst.

Interaction: Branches

Branch misprediction ➔ inhibits performance

- penalty less if misprediction detected sooner

VP and IR impact branch misprediction penalty

❶ Both collapse true dependences

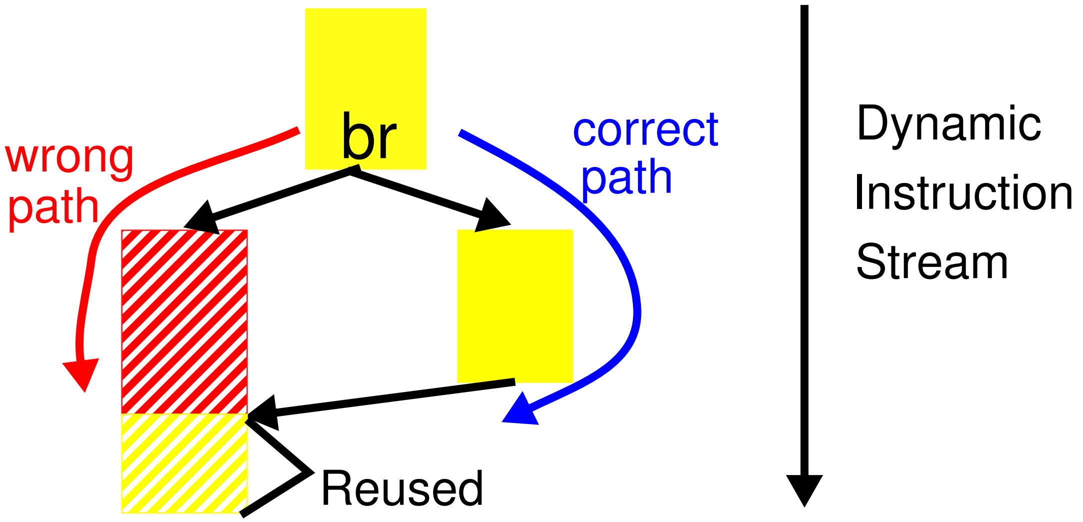
- thereby can execute branches sooner
  - ➔ detect misprediction early
  - ➔ reduce misprediction penalty

Interaction: IR Specific

❷ IR reduces branch misprediction penalty further

① by detecting misprediction still earlier

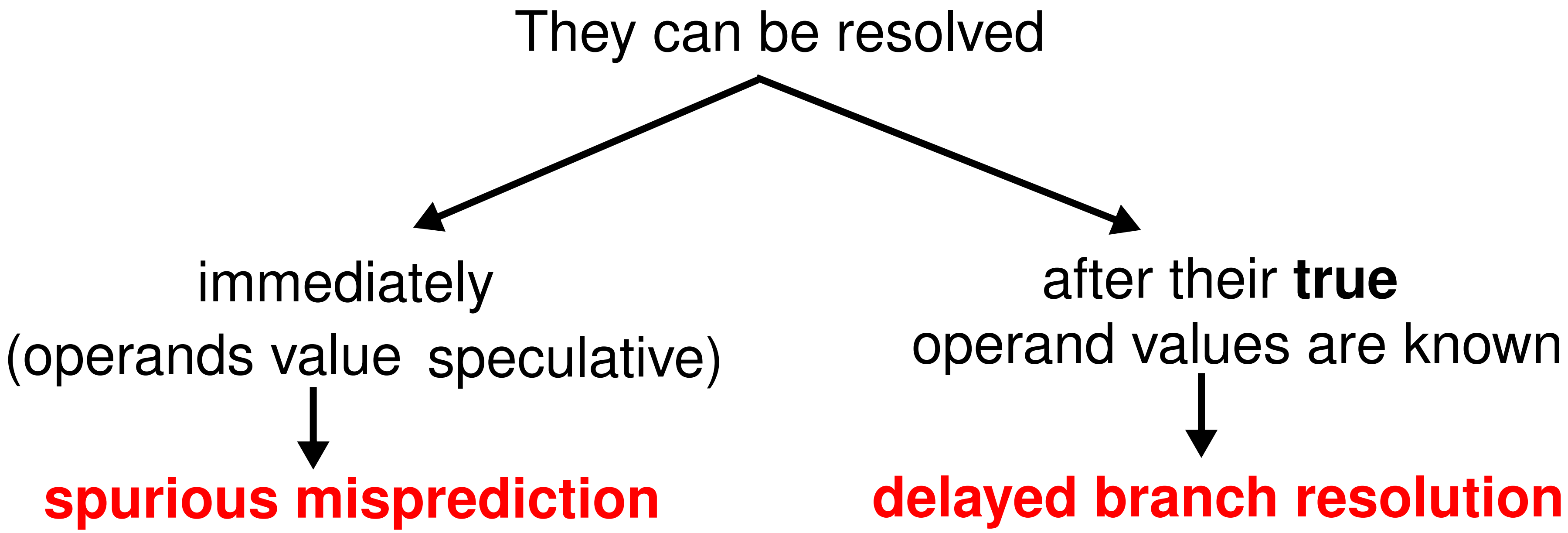
② by recovering useful work from squashes



Interaction: VP Specific

❸ VP tends to increase branch misprediction penalty

- Branches execute with **value-speculative operands**



They can be resolved

immediately (operands value speculative)

spurious misprediction

after their **true** operand values are known

delayed branch resolution

Either way more cycles lost due to branch misprediction

Resource Contention

Inst. contends for different <u>resources</u> in pipeline

(e.g., FU, cache ports, inst. issue ports)

Interactions

❶ VP and IR may ↑ or ↓ contention

- they cluster or spread requests for resources

❷ IR tends to reduce contention

- reused instructions don’t execute

❸ VP tends to increase contention

- mispredicted instructions re-execute

Execution Latency of Instructions

- **VP**: instructions execute to verify prediction
  - exec. latency of individual inst. not affected
- **IR**: reused instructions don’t execute
  - exec. latency eliminated

Impacts: Quantitative

VP : 16K-entry VPT Table, 4-way set assoc. 2-bit conf. counters

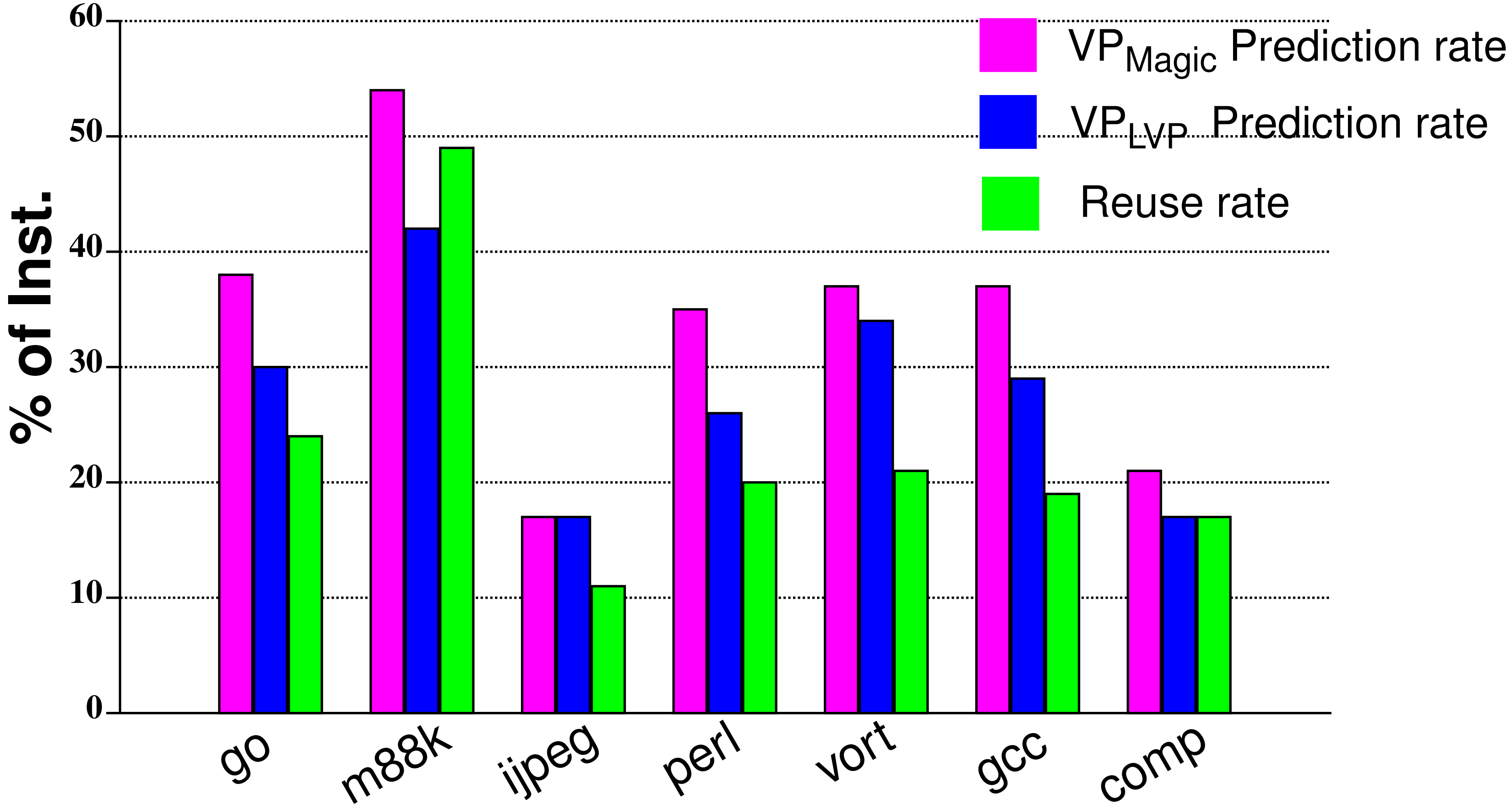
- VP_{Magic}: 4 instances per instruction
    - Correct prediction chosen magically
- VP_{LVP} : Last value used as prediction
- Selective re-issuing on misprediction

IR : 4K-entry Reuse Buffer, 4-way set assoc.

- 4 instances per instruction
    - Reuse test performed in parallel; successful one is reused

Machine : 4-way OOO execution, 32 inst. window, 16K-entry Gshare

Value Prediction and Reuse Rates



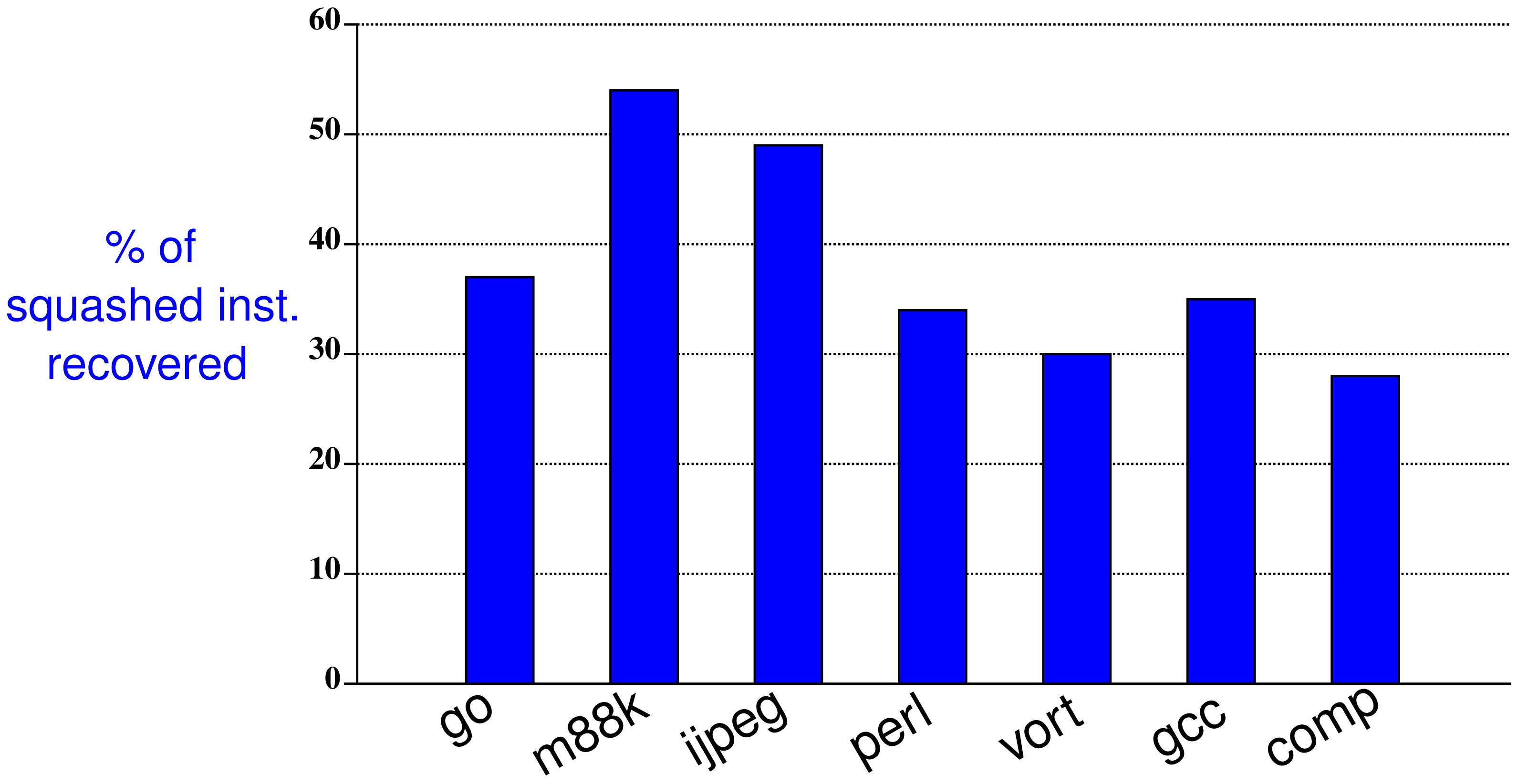
More instructions correctly predicted than reused

Value Misprediction Rates

Benchmarks	VP_{Magic} (%)	VP_{LVP} (%)
go	3.3	4.5
m88ksim	0.6	2.7
ijpeg	0.9	4.4
perl	1.2	1.7
vortex	1.1	3.3
gcc	1.9	3.9
compress	0.2	0.6

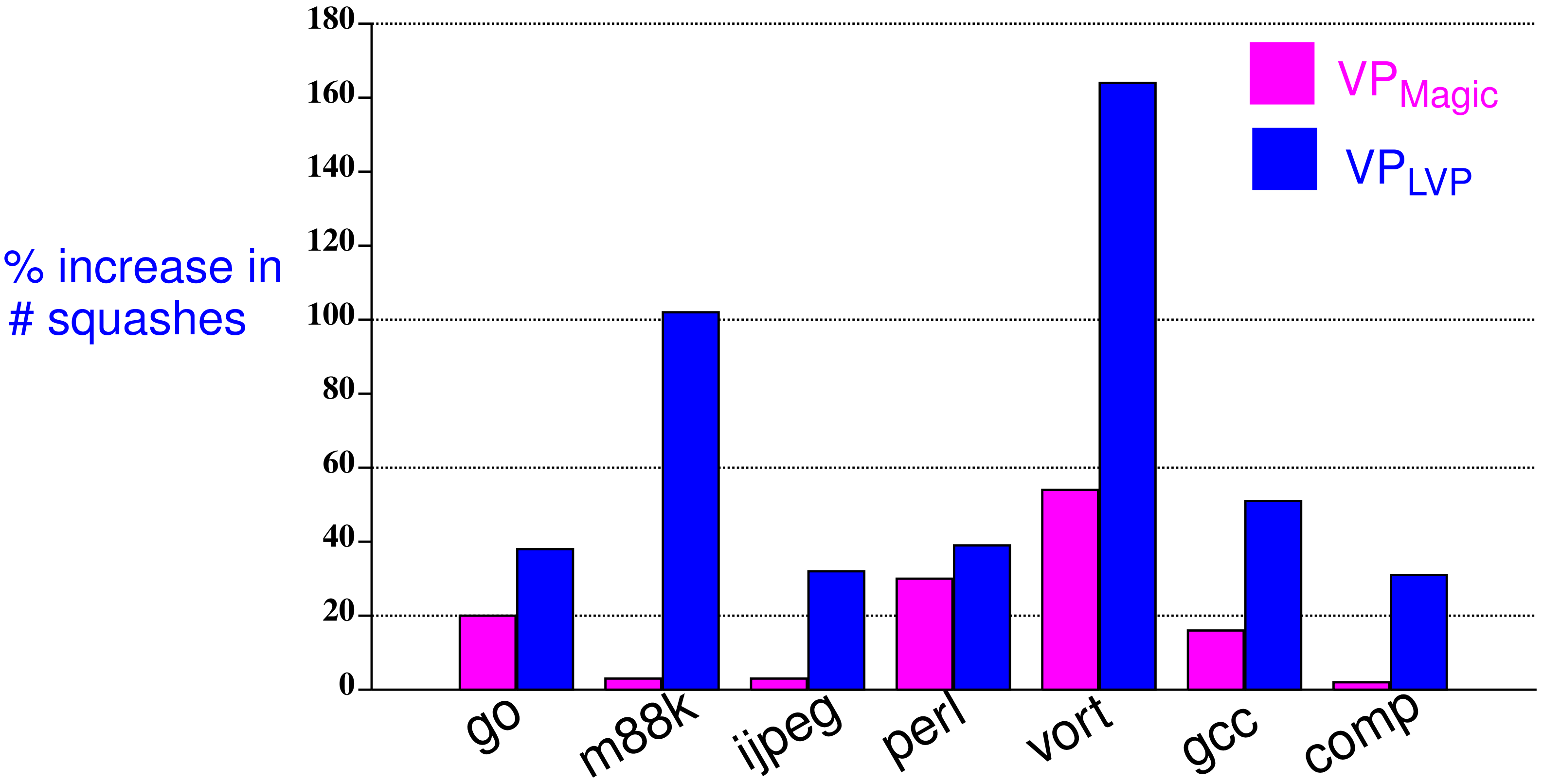
Misprediction rates: ❶ overall low ❷ but higher for VP_{LVP}

Squashed Inst. Recovered by IR



Faster recovery from branch misprediction → less penalty

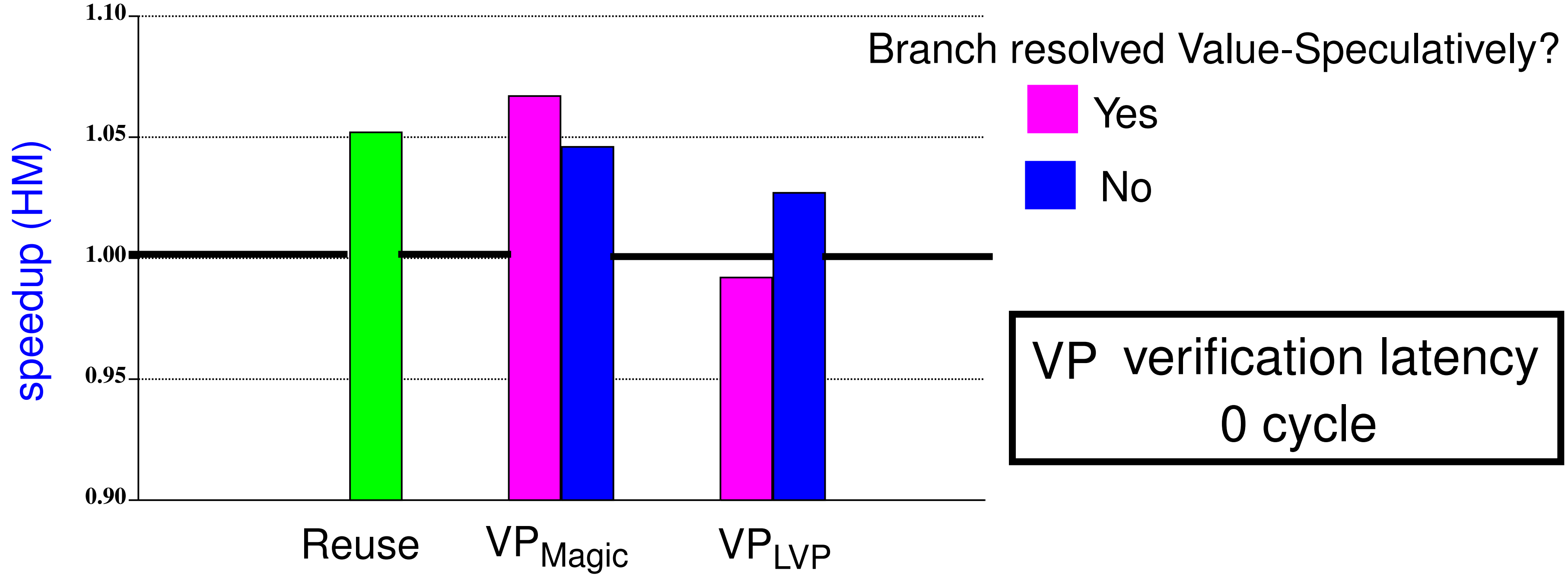
Spurious Branch Mispredictions



Significant spurious mispredictions caused

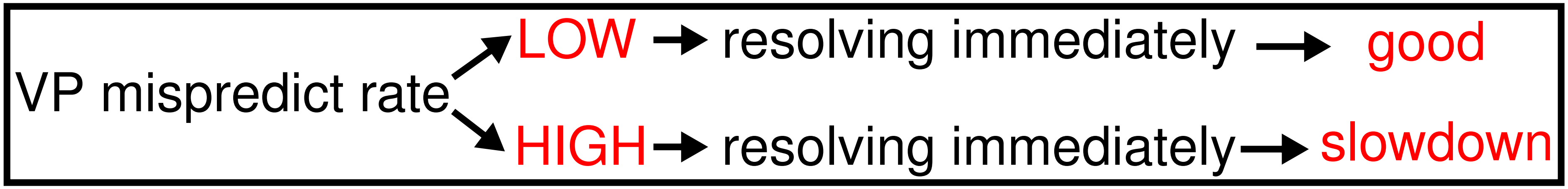
- especially for VP_{LVP} where value mispredictions are more

Performance



VP verification latency
0 cycle

VP performance sensitive to when branches resolve



VP mispredict rate

LOW → resolving immediately → good

HIGH → resolving immediately → slowdown

Summary

❶ Both VP and IR collapse true dependences

- **VP**: <u>speculative</u> (verifies results after use)
- **IR**: <u>non-speculative</u> (verifies results before use)

❷ VP higher potential for capturing redundancy

- but may interact adversely with branch prediction

❸ IR is conservative

- but alleviates branch misprediction penalty
- and is always correct

❹ Also in paper : an estimation of how conservative is IR