Improving Virtual-Function-Call Target Prediction via Dependence-Based Pre-Computation

Amir Roth, Andreas Moshovos and Guri Sohi

amir,sohi@cs.wisc.edu

moshovos@ece.nwu.edu

Computer Sciences Department

University of Wisconsin-Madison

Introduction

Goal: Reduce branch/target mispredictions

Idea: Dependence-Based Pre-Computation

- Supplement conventional prediction
- Pre-compute selected targets/branch outcomes
    - Identify instructions that compute targets/branches
    - Speculatively pre-execute these instruction sequences
    - Use results as predictions
- This work: Virtual-Function-Call (V-Call) targets
    - Proof of concept
    + Simple implementation

Overview: Problem and Technique

Why do conventional predictors mispredict?

```
for (i = 0; i < ASIZE; i++)
    if (a[i]->valid == TRUE)
        print(a[i]);
```

They rely on expressed correlation (which may not exist)

- Local: **a[i]->valid == TRUE** using **a[i-1]->valid == TRUE**?
- Global: **a[i]->valid == TRUE** using i < ASIZE?

No Correlation? Use Pre-Computation

- Identify br anch computation: **a[i]->valid == TRUE**
- Using **a**,**i** as inputs, pre-compute and store the result
- Use stor ed result as a prediction

+ No correlation necessary!

© 1999 Amir Roth UW-Madison

Talk Outline

- ~~Intr oduction~~
- V irtual Function Calls (V-Calls)
- Dependence-Based Pr e-Computation
- Number s
- Summ ary

Virtual Function Calls (V-Calls)

Use: Polymorphism (C++/Java)

- Multiple dynamic function tar gets from single static call site
- Object type selects tar get at runtime

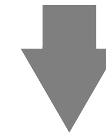
C++ types

```
class Base
    virtual int Valid();
    virtual void Print();
```

```
class Derived : Base
    int Valid();
    void Print();
```

Statically: one call site

```
for (i = 0; i < ASIZE; i++)
    if (a[i]->Valid())
        a[i]->Print();
```



Dynamically: multiple targets

```
a[0]->Base::Valid()
a[0]->Base::Print()
a[1]->Derived::Valid()
a[1]->Derived::Print()
```


© 1999 Amir Roth UW-Madison

Conventional V-Call Target Prediction

BTB's (Branch Target Buffers) don't work

- Single target per static call (need multiple)

Correlated (path-based) BTB's are better

- Target history index [Driesen&Hoelzle ISCA97,98]

```
for (i = 0; i < ASIZE; i++)
    if (a[i]->Valid())
        a[i]->Print();
```

- Local: **a[i]->Valid()** using **a[i-1]->Valid()**? No (different object)
+ Global 1: **a[i]->Print()** using **a[i]->Valid()**? Yes (same object)
- Global 2: **a[i]->Valid()** using **a[i-1]->Print()**? No (different object)

There is room for improvement!

Dependence-Based Pre-Computation

Idea: Watch the program and imitate

Three step process:

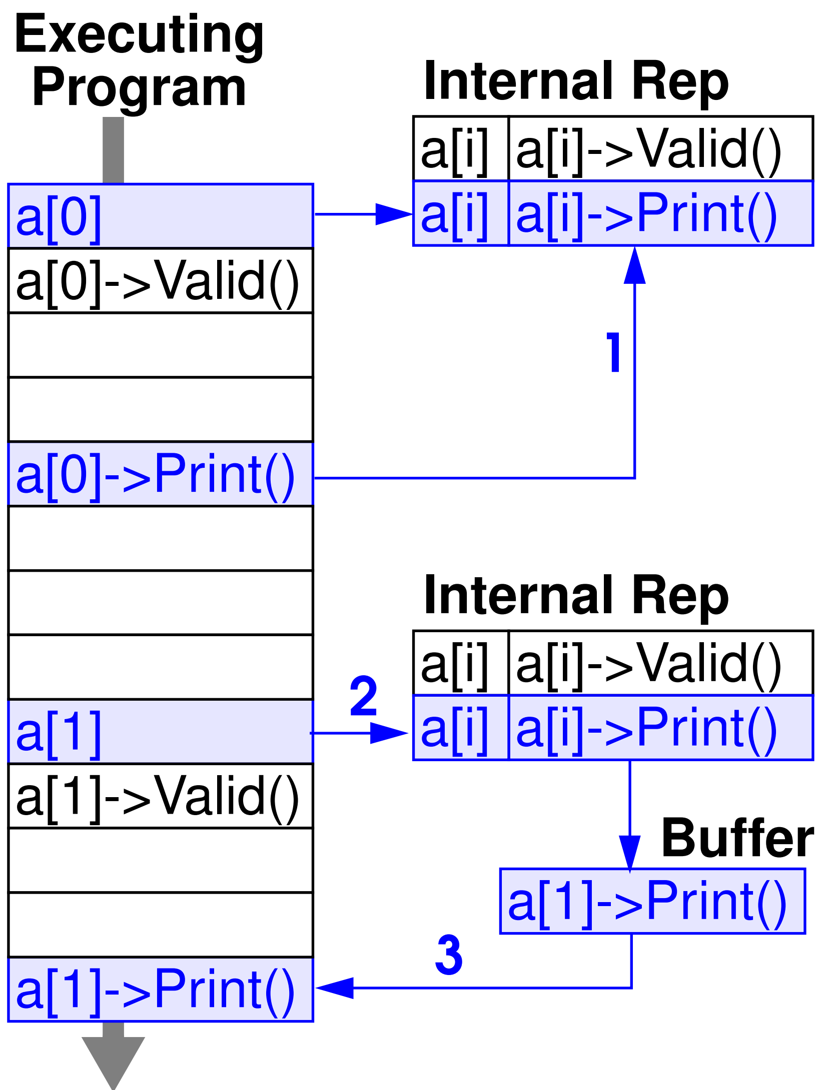
1. Identify and cache relevant instruction sequences
2. Speculatively instantiate with appropriate inputs
3. Match pre-computed results with predictions (challenge)

Why V-Calls?

+ Simple dependence chain makes steps 1+2 easy

© 1999 Amir Roth UW-Madison

Pre-Computation Mechanics



1. Isolate relevant instructions

- Build inter nal representation
- W ork backwards from call
- Track dependences (names)

2. Pre-Compute

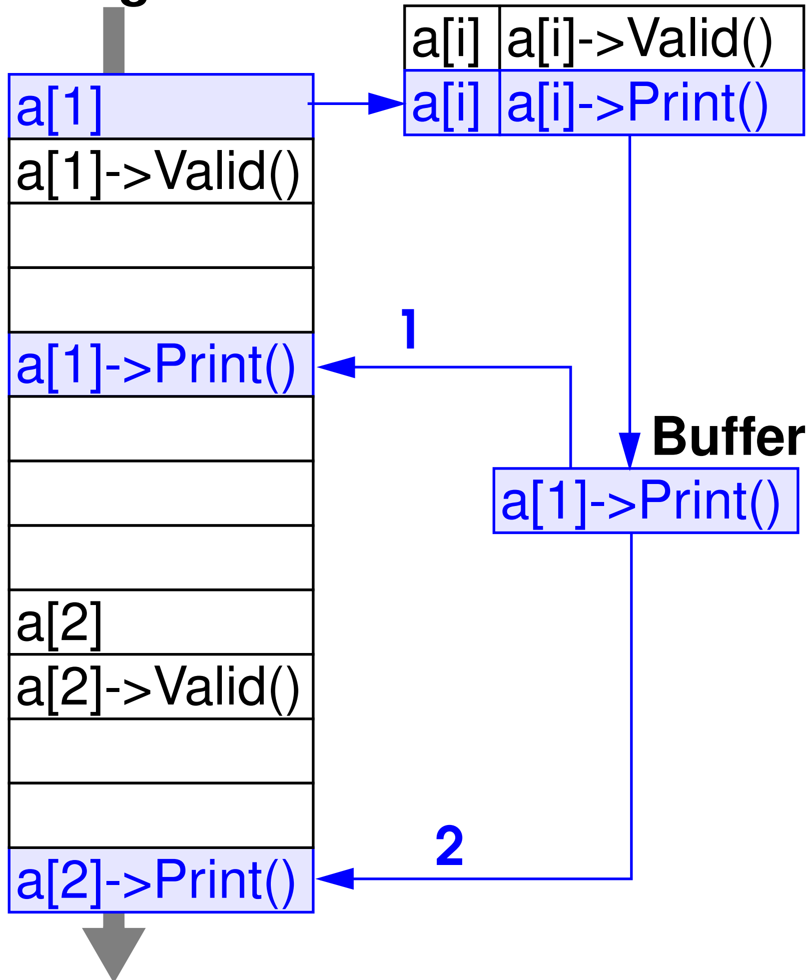
- Star t from **a[i]**
- Unr oll representation
- Mor e? [ASPLOS 98]

3. Use Pre-Computation

- Buf fer pre-comp result
- Pic k up stored result

One Problem

Pre-computation and fetch/prediction are in a race



Pre-computation wins? Great

Prediction wins? Problems

1. Ineffectiveness/Waste

- Late pr e-comps don't help
- Pr e-computed for nothing

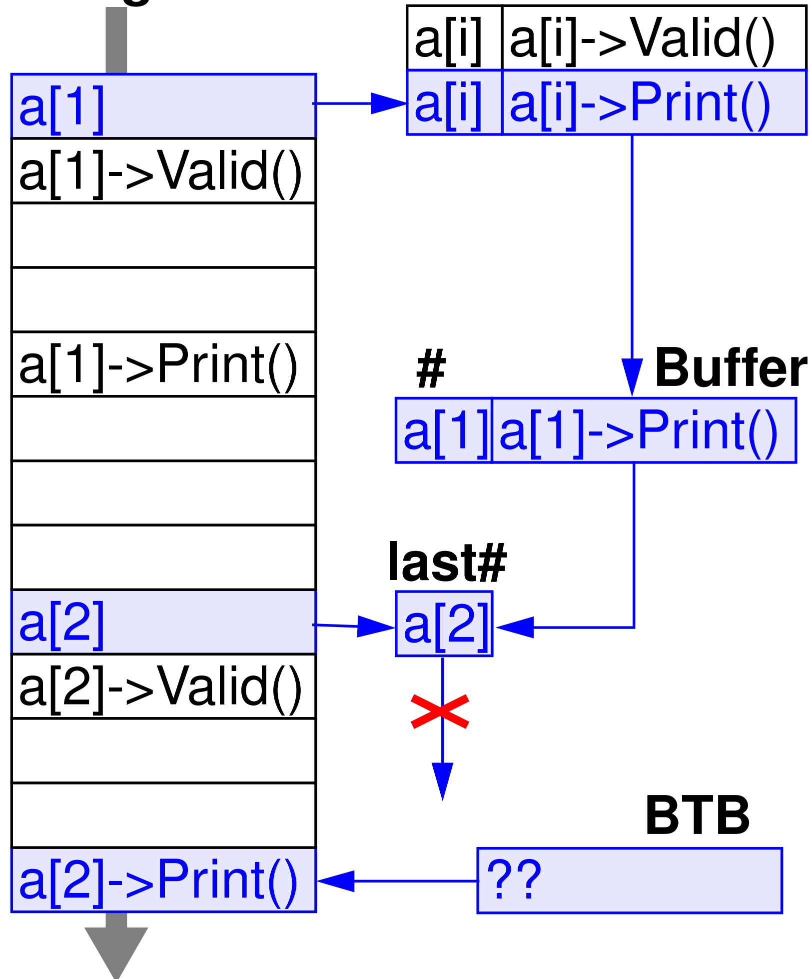
2. Introduced Mispredictions

- **a[1]->Print()** may mess up **a[2]->Print()** prediction

© 1999 Amir Roth UW-Madison

Preventing Introduced Mispredictions

Idea: Invalidate a[1] pre-comps when a[2] is fetched



Mechanism

- Tag pre-comp with **a[i]** seq#
- Pre-comp good if seq# is most recent for **a[i]**

How it works

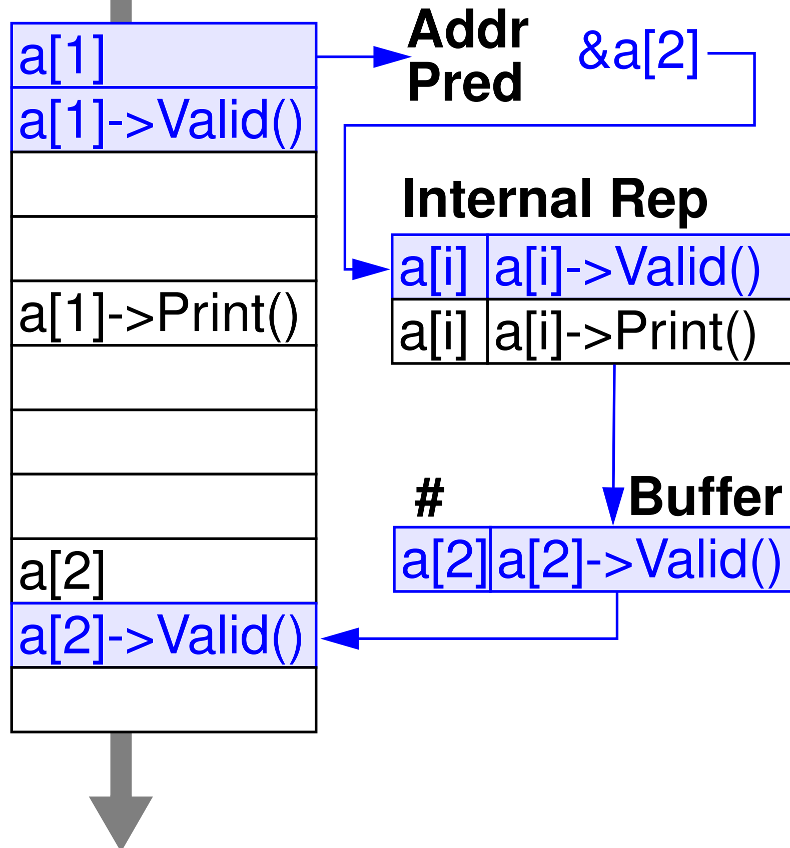
- **a[1]->Print()** pre-comp seq# is **a[1]**
- At **a[2]->Print()** prediction time, most recent seq# is **a[2]**
- Pre-comp with seq# **a[1]** stale (use BTB)

(see paper for more details)

Ineffectiveness: Lookahead Pre-Computations

Problem: Not enough distance from a[i] to a[i]->Valid()

Idea: Exploit distance from a[i-1] to a[i]->Valid()



Mechanism

- **a[i]** usually address predictable
- Using **&a[i-1]**, predict **&a[i]**
- Launch **a[i]->Valid()**
- Incor porate into seq# scheme (see paper)

Two schemes

- **Lookahead**: address prediction
- **Simple**: no address prediction

© 1999 Amir Roth UW-Madison

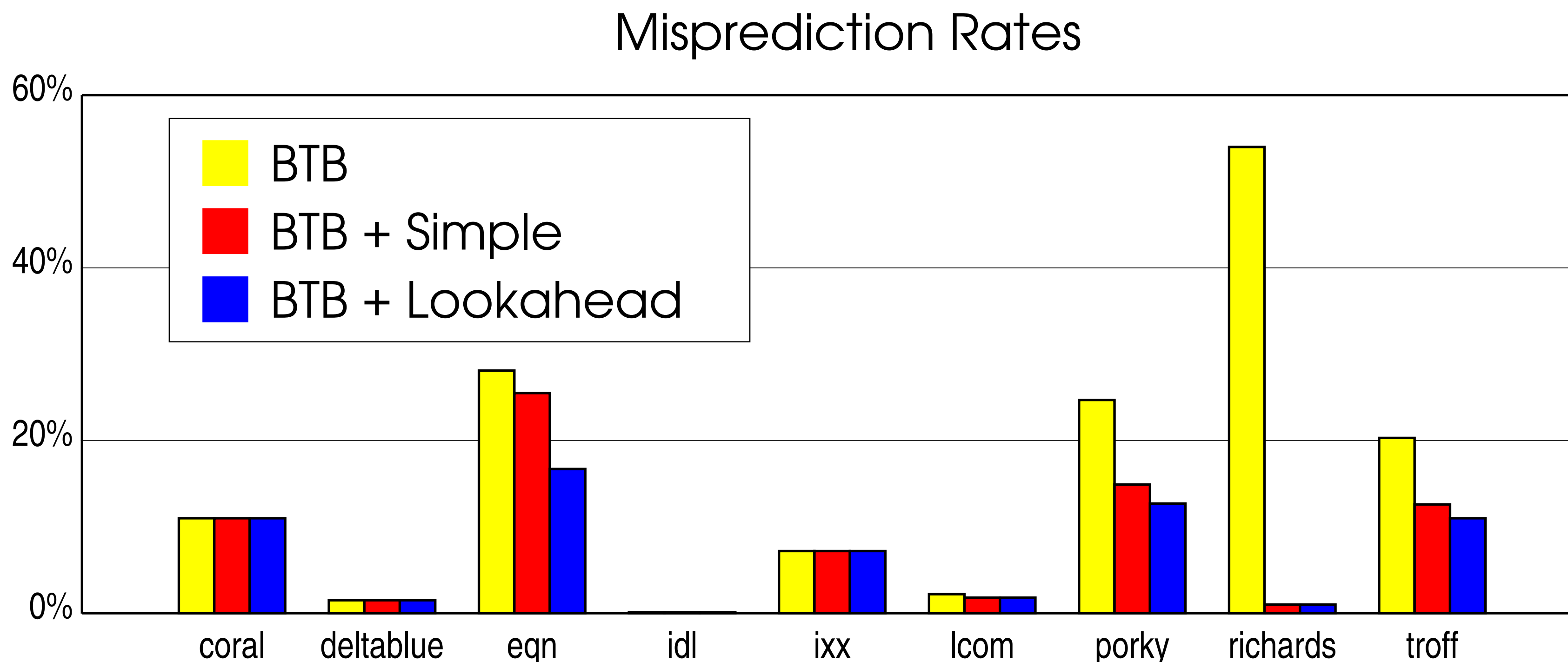
Experiments

Benchmarks: OOCSB (C++)

Simulations: SimpleScalar (MIPS, GCC)

- 4-wide super scalar, 5-stage pipe
- Speculativ e OOO-issue, 64 instructions in-flight
- 64 KB L1 D-Cache, 512KB L2 U-Cache
- Br anches: 8K-entry combined 10-bit GSHARE + 2-bit counters
- T arget prediction:
    - **BTB**: 2K-entry, 4-way associative
    - **PATH**: BTB + 2K-entry, DM, 2-level BTB, 3 target history

Numbers: BTB base predictor



richards, eqn, lcom, porky, troff:

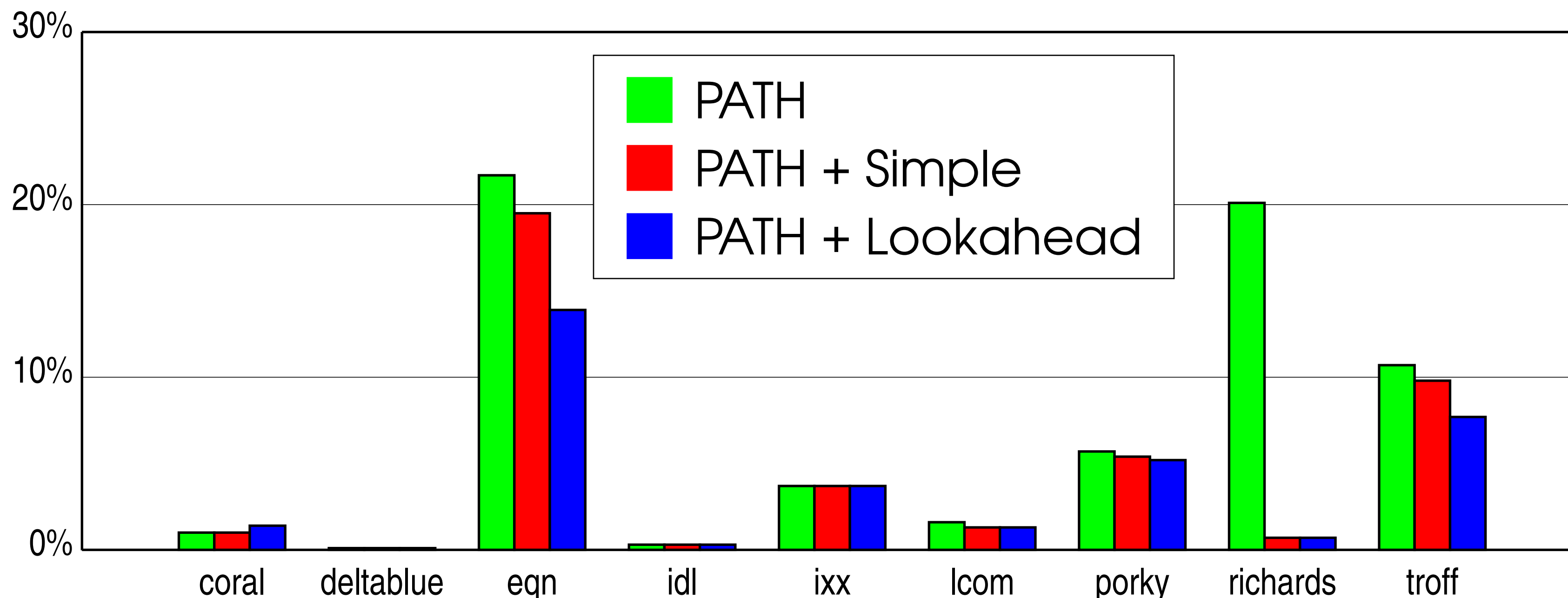
- + **Simple** handles long distance cases (**a[i]->Print()**)
- + **Lookahead** handles short distance cases (**a[i]->Valid()**)

others:

- - **Simple**: short distances, **lookahead**: unpredictable addresses

© 1999 Amir Roth UW-Madison

Numbers: PATH base predictor



overall:

- **PATH** handles correlated cases (**a[i]->Print()**)

richards, eqn, troff:

+ **Lookahead** helps uncorrelated (**a[i]->Valid()**)

© 1999 Amir Roth UW-Madison

Numbers: Explanations

What about overall performance?

- V -Call rate low in absolute terms (1 per 200-1000 instructions)
- P erformance improves by 0-2%

Sometimes (coral) more harm than good

- Lookahead pr e-computation relies on address prediction
- Wr ong address prediction? Wrong pre-computation

+ Not common

Summary

Dependence-Based Pre-Computation

+ Can be used to augment branch/target prediction
+ Succeeds where statistical correlated prediction fails

- Similar technique pr efetches linked structures [ASPLOS98] (where statistical address prediction also fails)

Closely related

- Br anch Flow Window [Farcy et.al., MICRO98]

Can be generalized to handle all branches

© 1999 Amir Roth UW-Madison