A Programmable Co-processor for Profiling

Craig Zilles and Guri Sohi

University of Wisconsin - Madison

International Symposium on
High Performance Computer Architecture

January 2001

Overview

Profiling is increasingly important

- necessitates efficient collection of profile information

Intelligent Instruction Sampling

- directed sampling
- on-line summarizing

Programmable Profiling Co-processor

- flexible – can implement many profiling applications
- primitives in hardware, policies in software
- small and simple; ~1/2 million transistors (10KB SRAM)

Outline

- Motivation
  - Profiling Overview
  - Future Trends
  - Support for Future Trends
- Intelligent Instruction Sampling
- Profiling Co-processor Overview
- Results
- Related Work
- Conclusion

Feedback Directed Optimization (FDO)

Performance is dictated by a program's dynamic behavior

- e.g. branch- and memory-behavior

Modern hardware reacts to dynamic behavior

- dynamic branch prediction
- out-of-order execution

Feedback-directed optimizations complement hardware

- Larger scope
- Non-speculative

CODE LAYOUT, SUPERBLOCK SCHEDULING, INLINING, SHRINK WRAPPING, HOT-COLD OPTIMIZATIONS, IF-CONVERSION, REGISTER ALLOCATION, TRACE SCHEDULING, ADVANCED LOADS, PRE-FETCHING, MEMOIZATION, SELECTIVE VALUE PREDICTION, SPECIALIZATION, BRANCH ALIGNMENT, PRE-EXECUTION..

Significantly improve performance

Example Profiling System

ProfileMe [Micro '97]**: sample instructions in hardware**

- **instruction tagged at fetch**
- **pipeline collects information**
- **interrupt at retirement; processed data in software**

Digital Continuous Profiling Infrastructure (DCPI) [SOSP '97]

- **attributes execution time to static instructions**
- **low overhead (1-3%) through low sampling rate**
- **requires long runs**
  - **good for compile-profile-recompile methodology**

Future Trends

More complicated optimizations

- value profiles, dependence profiles, etc.

Dynamic Optimization

- perform FDO online
  - can’t rely on software vendors; support legacy code
- collect data quickly
- minimize overhead

Future Trends

More complicated optimizations

- value profiles, dependence profiles, etc. → more samples

Dynamic Optimization

- perform FDO online
  - can't rely on software vendors, support legacy code
- collect data quickly → less time
- minimize overhead

$$\frac{\#\ \text{SAMPLES}\ \uparrow}{\text{TIME}\ \downarrow} = \text{SAMPLING RATE}\ \uparrow$$

$$\text{OVERHEAD}\ \downarrow = \text{SAMPLING RATE}\ \downarrow$$

Conflicting desires

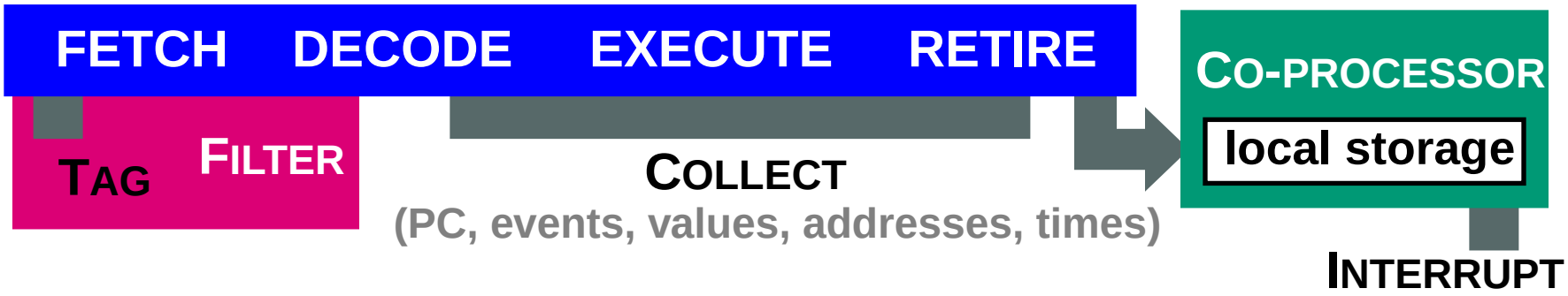
Supporting Profiling Future Trends

Goal: Improve Sampling

- collect the right samples
- reduce the cost of collecting those samples

Solution: Additional Hardware

- general enough to support many profiles

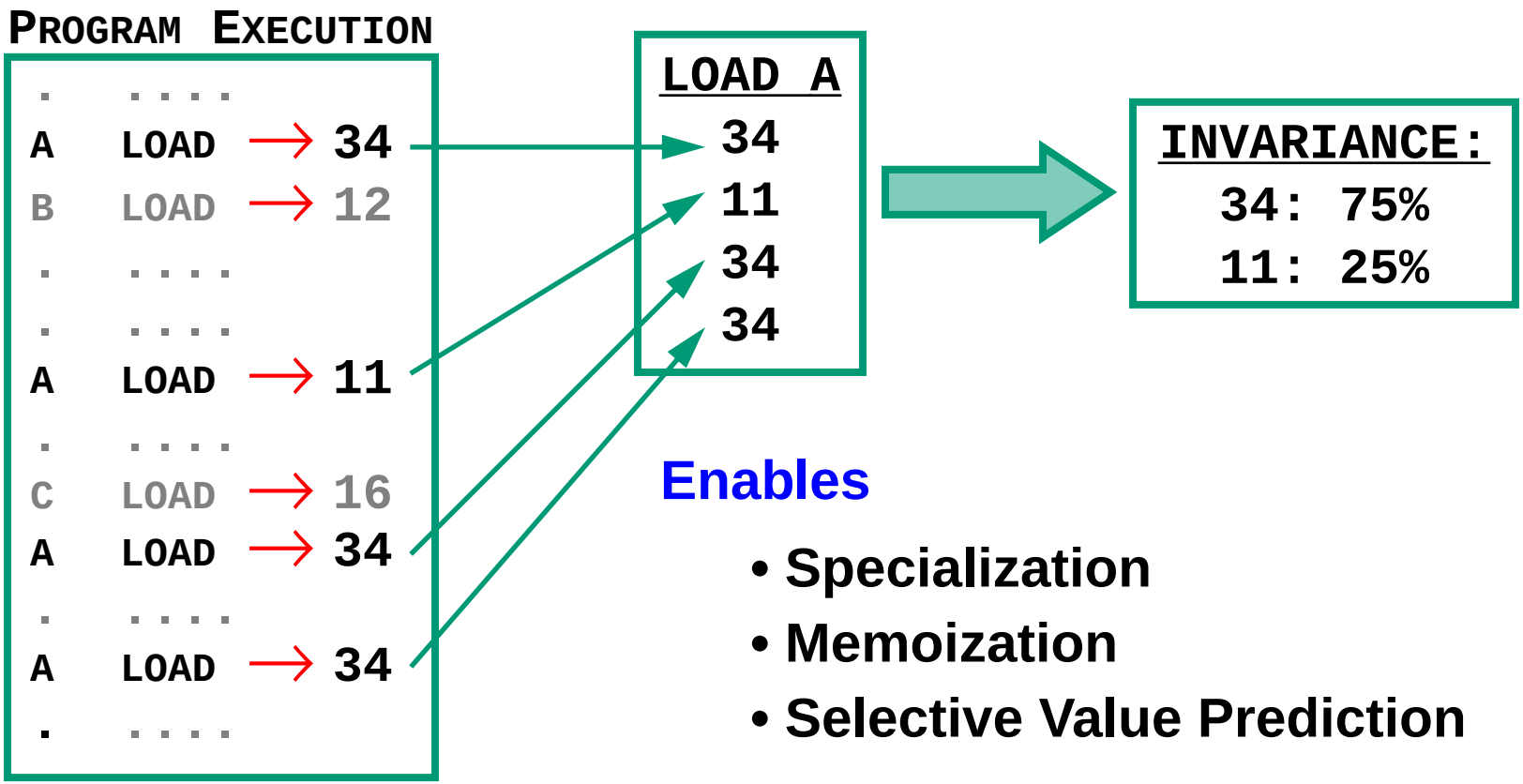


- hardware filters to guide instruction sampling
- post-processing co-processor summarizes samples

Outline

- Motivation
- Intelligent Instruction Sampling
  - Example: Value Profiling
  - Algorithms
- Profiling Co-processor Overview
- Results
- Related Work
- Conclusion

Example Application: Load Value Profiling



Enables

- Specialization
- Memoization
- Selective Value Prediction

Most interested in

- frequently executed instructions
- values with high invariances

Directed Sampling

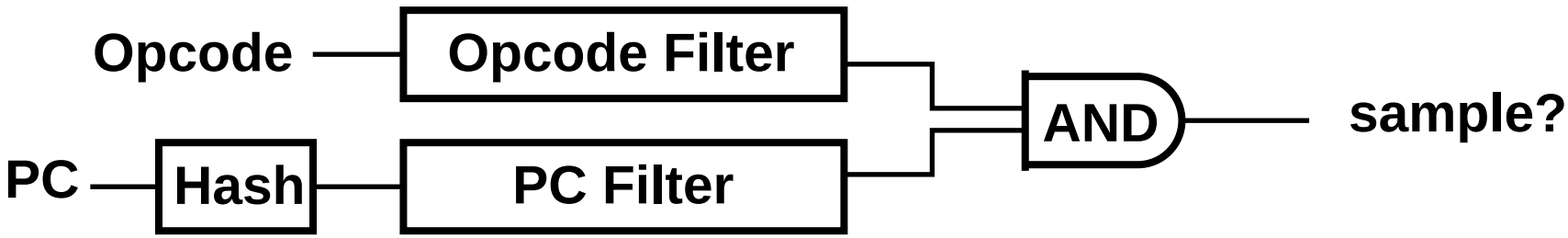
Which are the right instructions to sample?

Loads

- **filter which selects by opcode group**

Instructions not yet characterized

- **mark characterized static instructions in a table by PC**
- **do not profile marked instructions**
- **à la Convergent Profiling** [Micro '97]**, but in hardware**



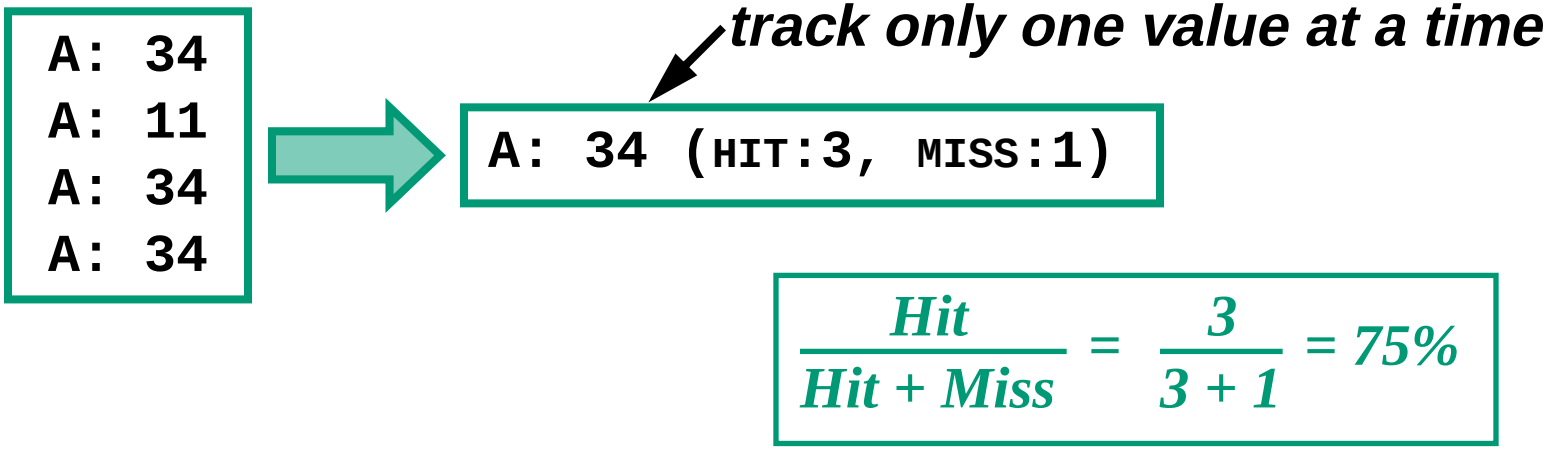
Reducing Per-sample Overhead

Overhead: Processing performed during profile interrupts

Summarize Samples with Co-processor

- **Constraint:** Limited Local Storage

Only care about most frequent values



$$\frac{Hit}{Hit + Miss} = \frac{3}{3 + 1} = 75\%$$

Identify frequent values by

- statistically likely to select frequent values
- re-select if measured invariance is low

Reducing Per-sample Overhead (2)

Constraint: Limited Local Storage

- local storage << program size

Sample instructions in groups

- most frequent → least frequent
  - most important
  - easiest to profile

We don't know *a priori* which are most frequent

- replacement decisions based on # samples collected

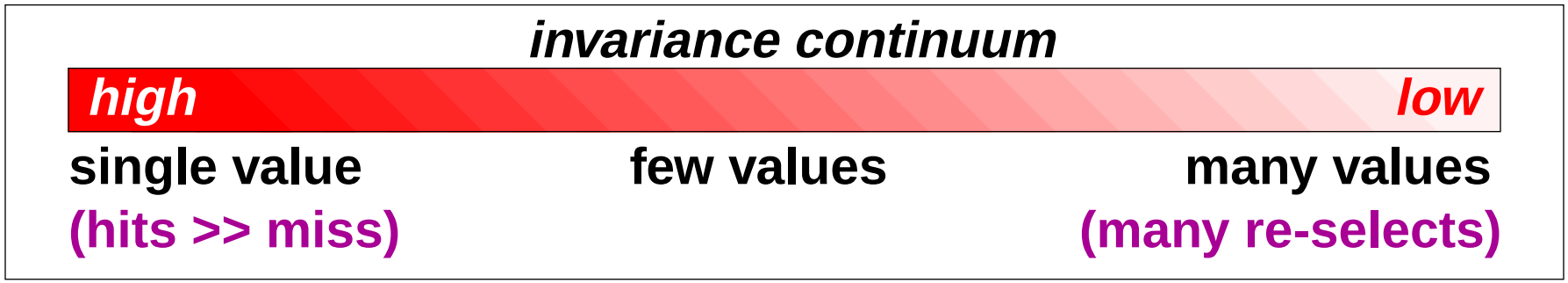
Characterization Prediction

How do we know when an instruction has been characterized?

No way to know for sure

- because of phase changes

Predict convergence with a simple test



The ends of the continuum

- can be characterized with “small” number of samples
- capture many instructions

Periodically re-sample to detect phase changes

Outline

- Motivation
- Intelligent Instruction Sampling
- **Profiling Co-processor Overview**
- **Results**
- **Related Work**
- **Conclusion**

Generic Hardware

Filter/Summarize methodology is widely applicable

- problem instruction profiling
- edge/path profiling
- memory dependence profiling
- cache conflict profiling
- stall profiling

These algorithms require

- similar storage structures and operations
- different algorithms and policies

Use Programmable/Configurable hardware

- summarizing/replacement done in software

Programmable Profiling Co-processor

Goal: High throughput with few resources

- tailor co-processor specifically for profiling

Structure processor for profiling

- sample stream processed by implicit loop
- microcoded to exploit available parallelism efficiently

Provide profiling primitives in hardware

- instruction field extraction
- associative array for table lookups and matching

estimated size: one-half million transistors (~10KB SRAM)

Evaluation Methodology

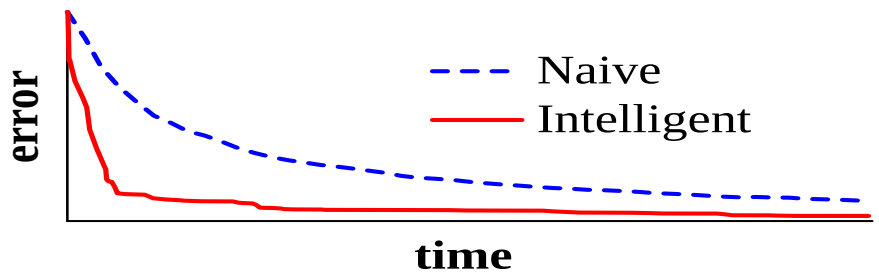
Timing simulator-based evaluation

- profiling co-processor timing simulator
- simplescalar-based
- value profiling co-processor microcode
- interrupt handlers that assemble complete profile

Compared

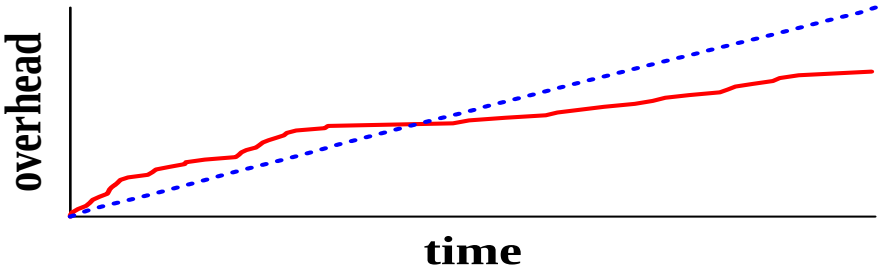
- Naive
  - random sampling, buffers samples to amortize interrupt
- Intelligent
  - directed sampling, summarizes samples in co-processor

Results



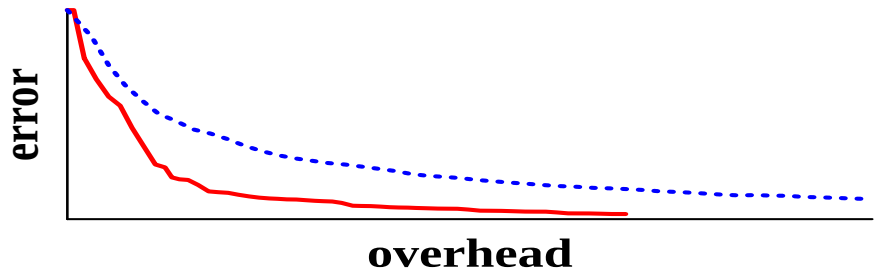
Much faster convergence

- start optimization sooner



Self-tuning

- stops upon convergence



Better accuracy for overhead

- for all benchmarks

Sensitivity analysis of co-processor hardware (in paper)

- storage: less storage reduces collection rate
- clock frequency: largely insensitive → easy to design

Related Work

Hardware Summarizing of Profile Data:

- Profile Buffer: Conte, et al., Micro 1994
- Hot Spot Detector: Merten, et al., ISCA 1999

Co-processor Observation of Retirement Stream:

- I-COP: Chou, et al., ISCA 2000

Profiling Architecture:

- Relational Profiling Architecture: Heil and Smith, Micro 2000

Conclusion

Future microprocessors will

- monitor application behavior
- react accordingly

Algorithms and policies for a self-tuning profiling system

- directed sampling
- on-line summarizing

Efficient, flexible hardware implementation of these techniques

- programmable profiling co-processor