Ubiquitous Memory Introspection (UMI)

Qin Zhao, NUS
Rodric Rabbah, IBM
Saman Amarasinghe, MIT
Larry Rudolph, MIT
Weng-Fai Wong, NUS

CGO 2007, March 14 2007
San Jose, CA

The complexity crunch

hardware complexity ↑

\+

software complexity ↑

⇓

too many things happening concurrently, complex interactions and side-effects

⇓

less understanding of program execution behavior ↓

The importance of program behavior characterization

Better understanding of program characteristics can lead to more robust software, and more efficient hardware and systems

- S/W developer
  - Computation vs. communication ratio
  - Function/path frequency
  - Test coverage
- H/W designer
  - Cache behavior
  - Branch prediction
- System administrator
  - Interaction between processes

Common approaches to program understanding

Overhead	Space and time overhead
Level of detail	Coarse grained summaries vs. instruction level and contextual information
Versatility	Portability and ability to adapt to different uses

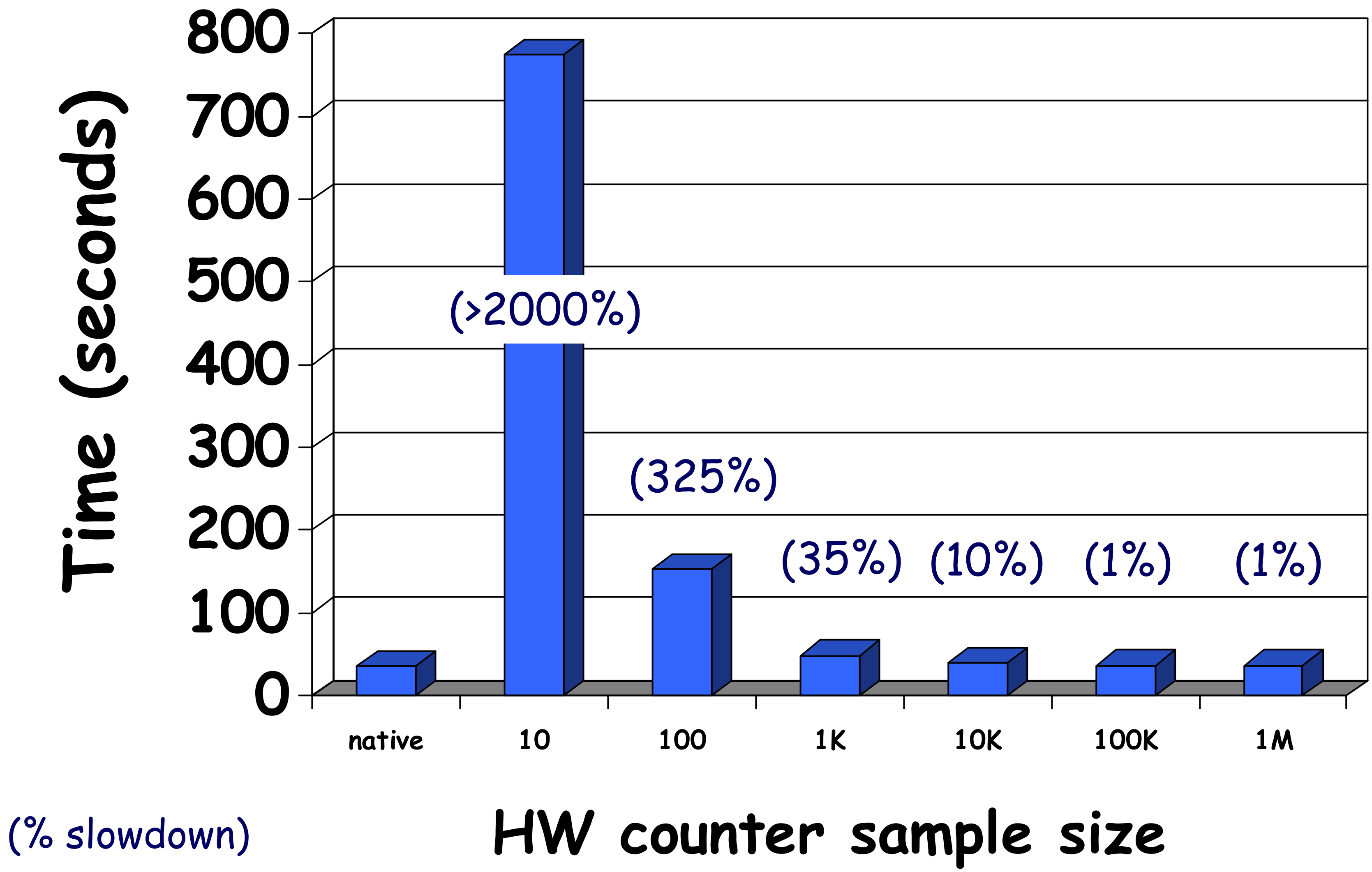
Common approaches to program understanding

	Profiling	Simulation
Overhead	high	very high
Level of detail	very high	very high
Versatility	high	very high

Common approaches to program understanding

	Profiling	Simulation	HW Counters
Overhead	high	very high	very low
Level of detail	very high	very high	very low
Versatility	high	very high	very low

Slowdown due to HW counters
(counting L1 misses for 181.mcf on P4)



Common approaches to program understanding

	Profiling	Simulation	HW Counters	Desirable Approach
Overhead	high	very high	very low	low
Level of detail	very high	very high	very low	high
Versatility	high	very high	very low	high

Common approaches to program understanding

	Profiling	Simulation	HW Counters	**UMI**
Overhead	high	very high	very low	low
Level of detail	very high	very high	very low	high
Versatility	high	very high	very low	high

Key components

- Dynamic Binary Instrumentation
  - Complete coverage, transparent, language independent, versatile, …
- Bursty Simulation
  - Sampling and fast forwarding techniques
  - Detailed context information
  - Reasonable extrapolation and prediction

Ubiquitous Memory Introspection

online mini-simulations analyze short memory access profiles recorded from frequently executed code regions

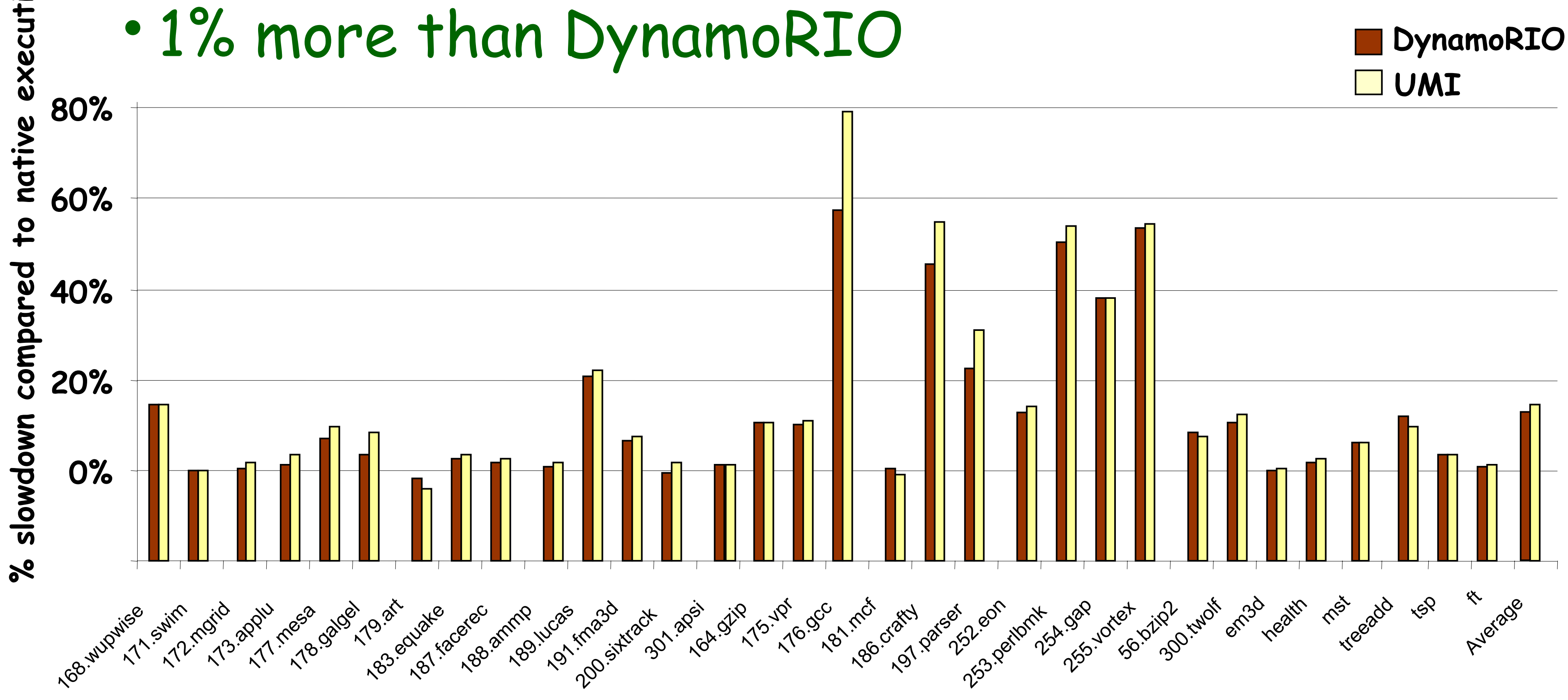
- Key concepts
  - Focus on hot code regions
  - Selectively instrument instructions
  - Fast online mini-simulations
  - Actionable profiling results for online memory optimizations

Working prototype

- Implemented in DynamoRIO
  - Runs on Linux
  - Used on Intel P4 and AMD K7
- Benchmarks
  - SPEC 2000, SPEC 2006, Olden
  - Server apps: MySQL, Apache
  - Desktop apps: Acrobat-reader, MEncoder

UMI is cheap and non-intrusive (SPEC2K reference workloads on P4)

- Average overhead is 14%
- 1% more than DynamoRIO

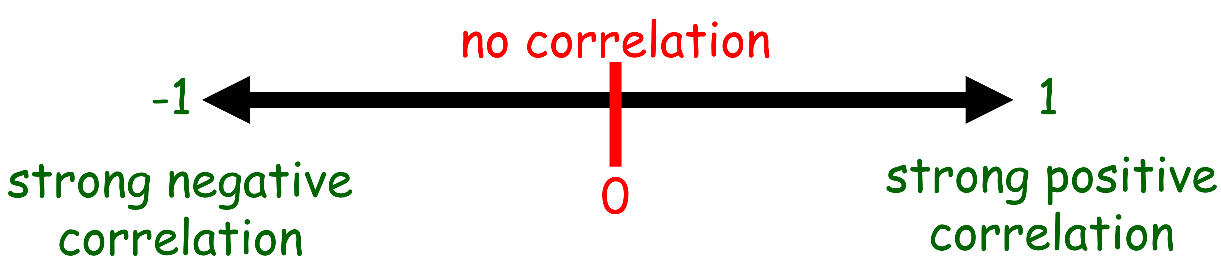


What can UMI do for you?

- Inexpensive introspection everywhere
- Coarse grained memory analysis
  - Quick and dirty
- Fine grained memory analysis
  - Expose opportunities for optimizations
- Runtime memory-specific optimizations
  - Pluggable prefetching, learning, adaptation

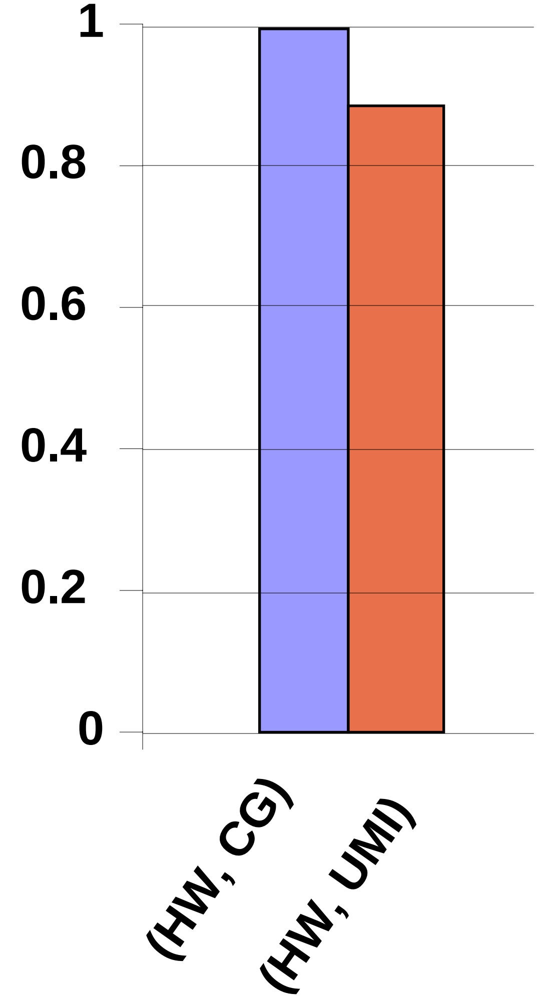
Coarse grained memory analysis

- Experiment: measure cache misses in three ways
  - HW counters
  - Full cache simulator (Cachegrind)
  - UMI
- Report correlation between measurements
  - Linear relationship of two sets of data



Cache miss correlation results

- HW counter vs. Cachegrind
  - 0.99
  - 20x to 100x slowdown
- HW counter vs. UMI
  - 0.88
  - Less than 2x slowdown in worst case

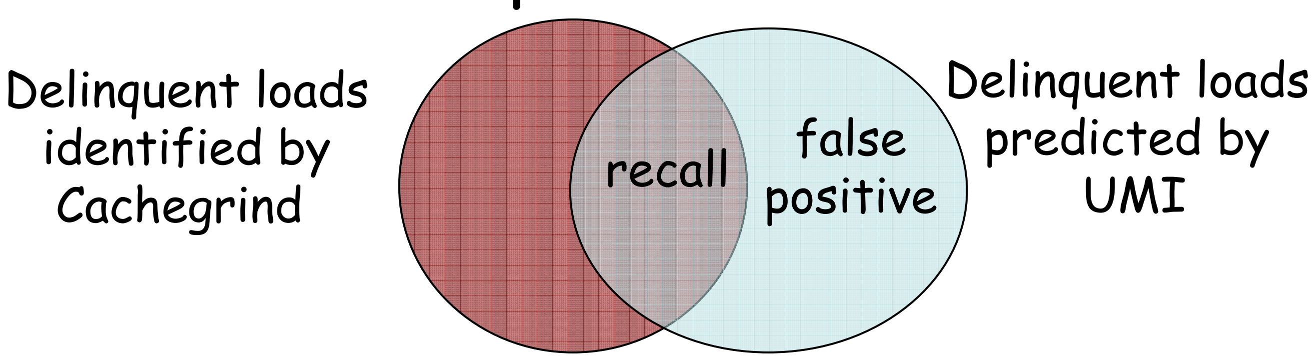


What can UMI do for you?

- Inexpensive introspection everywhere
- Coarse grained memory analysis
  - Quick and dirty
- Fine grained memory analysis
  - Expose opportunities for optimizations
- Runtime memory-specific optimizations
  - Pluggable prefetching, learning, adaptation

Fine grained memory analysis

- Experiment: predict delinquent loads using UMI
  - Individual loads with cache miss rate greater than threshold
- Delinquent load set determined according to full cache simulator (Cachegrind)
  - Loads that contribute 90% of total cache misses
- Measure and report two metrics



UMI delinquent load prediction accuracy

	Recall (higher is better)	False positive (lower is better)
benchmarks with ≥ 1% miss rate	88%	55%
benchmarks with < 1% miss rate	26%	59%

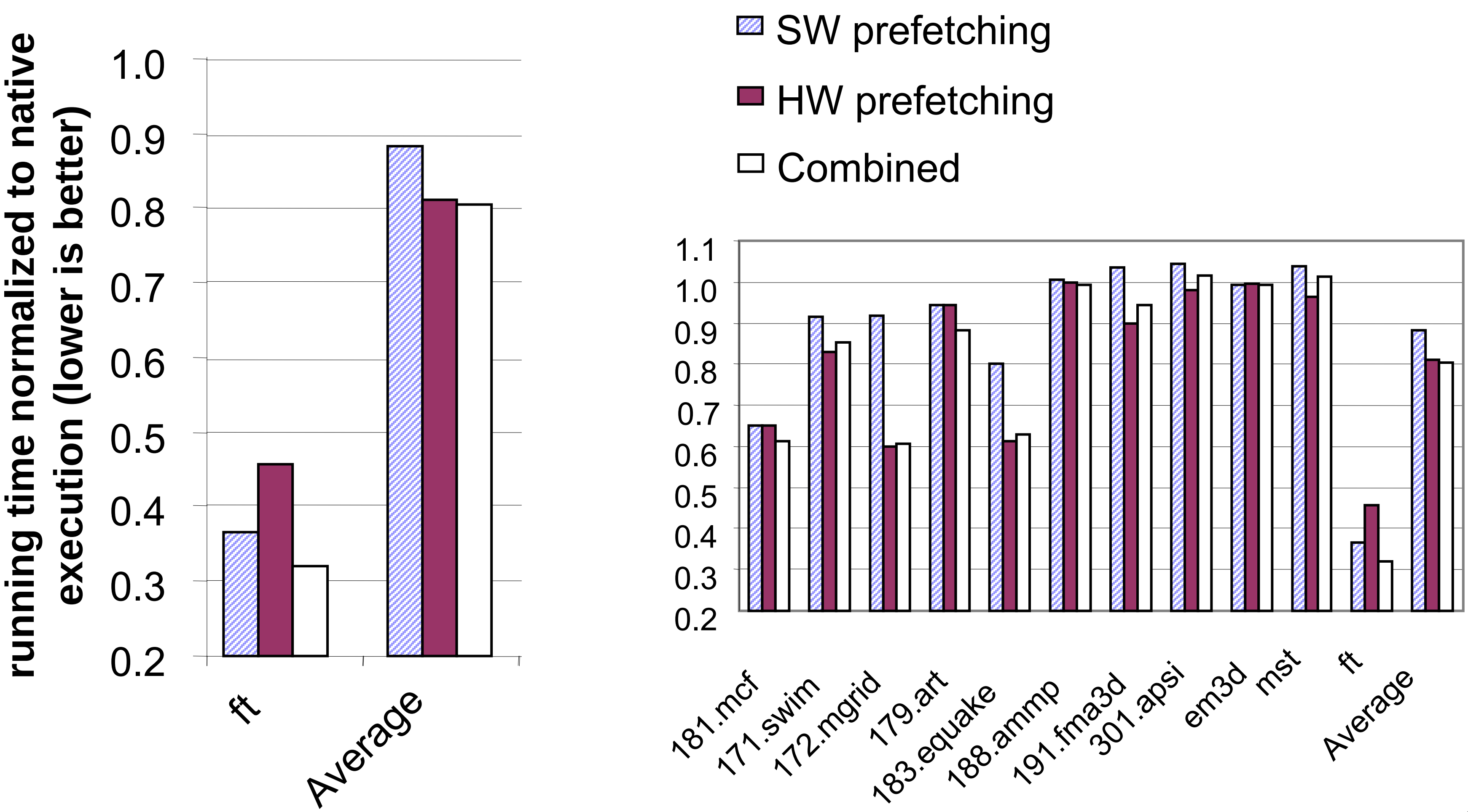
What can UMI do for you?

- Inexpensive introspection everywhere
- Coarse grained memory analysis
  - Quick and dirty
- Fine grained memory analysis
  - Expose opportunities for optimizations
- Runtime memory-specific optimizations
  - Pluggable prefetcher, learning, adaptation

Experiment: online stride prefetching

- Use results of delinquent load prediction
- Discover stride patterns for delinquent loads
- Insert instructions to prefetch data to L2
- Compare runtime for UMI and P4 with HW stride prefetcher

Data prefetching results summary



The Gory Details

UMI components

- Region selector
- Instrumentor
- Profile analyzer

Region selector

- Identify representative code regions
  - Focus on traces, loops
  - Frequently executed code
  - Piggy back on binary instrumentor tricks

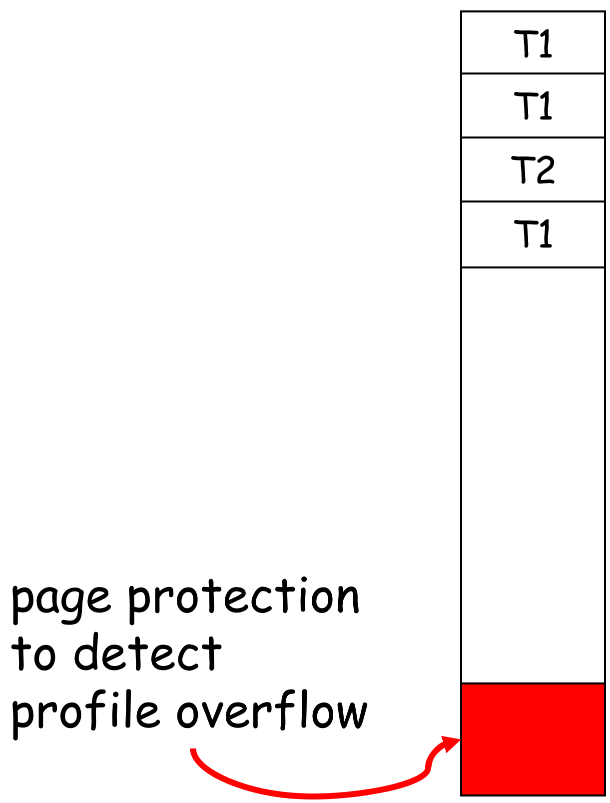
- Reinforce with sampling
  - Time based, or leverage HW counters
  - Naturally adapt to program phases

Instrumentor

- Record address references
  - Insert instructions to record address referenced by memory operation
- Manage profiling overhead
  - Clone code trace (akin to Arnold-Ryder scheme)
  - Selective instrumentation of memory operations
    - E.g., ignore stack and static data

Recording profiles

Code Trace Profile



Address Profiles

Code Trace T1

counter →

op1	op2	op3
0x011	0x024	0x100
0x012	early trace exist	early trace exist
0x013	0x028	0x104

Code Trace T2

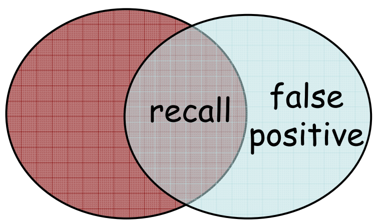
counter →

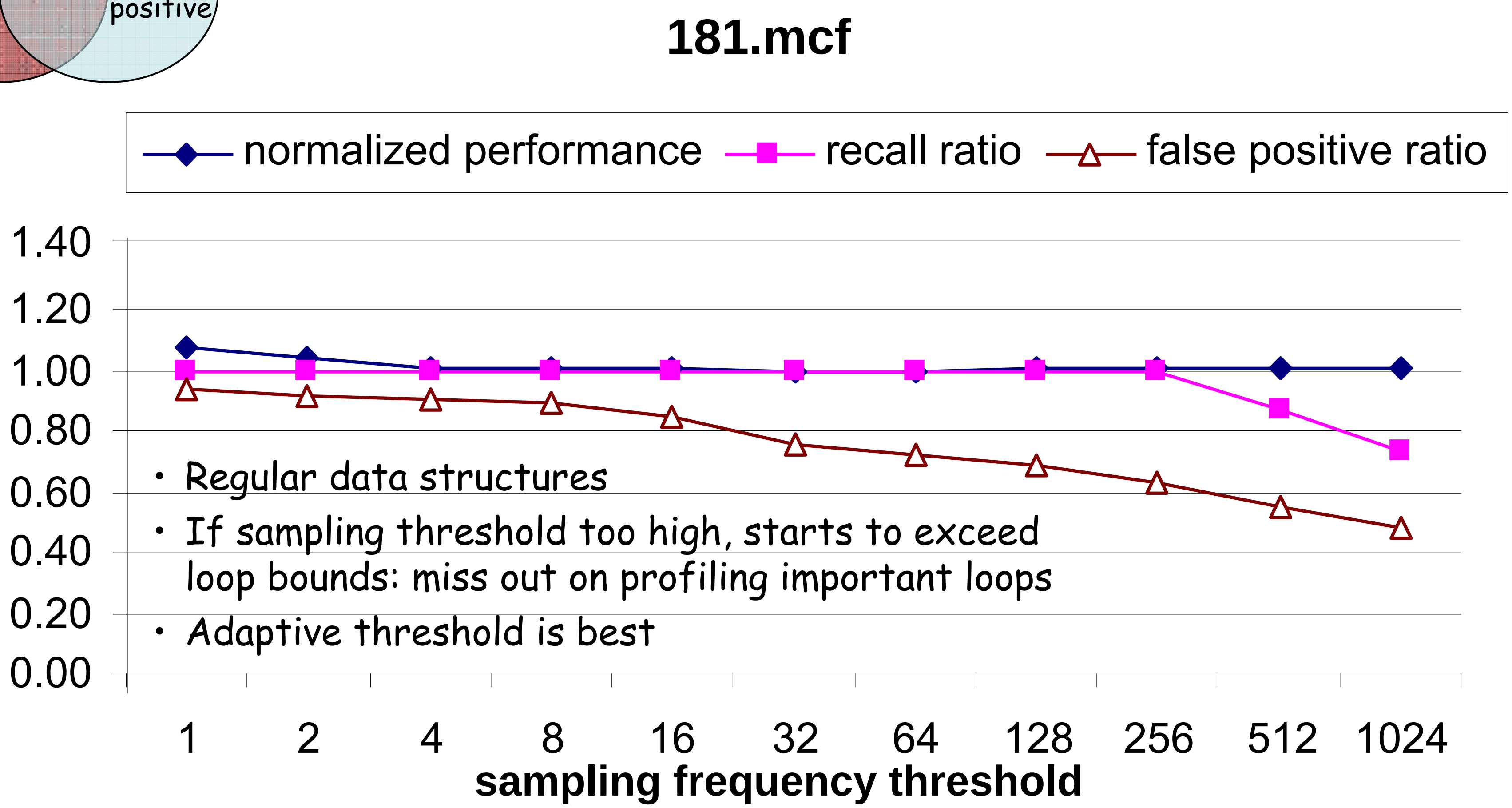
op1	op2
0x032	0x031

Mini-simulator

- Triggered when code or address profile is full
- Simple cache simulator
  - Currently simulate L2 cache of host
  - LRU replacement
  - Improve approximations with techniques similar to offline fast forwarding simulators
    - Warm up and periodic flushing
- Other possible analyzer
  - Reference affinity model
  - Data reuse and locality analysis

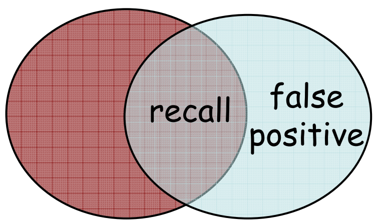
Mini-simulations and parameter sensitivity

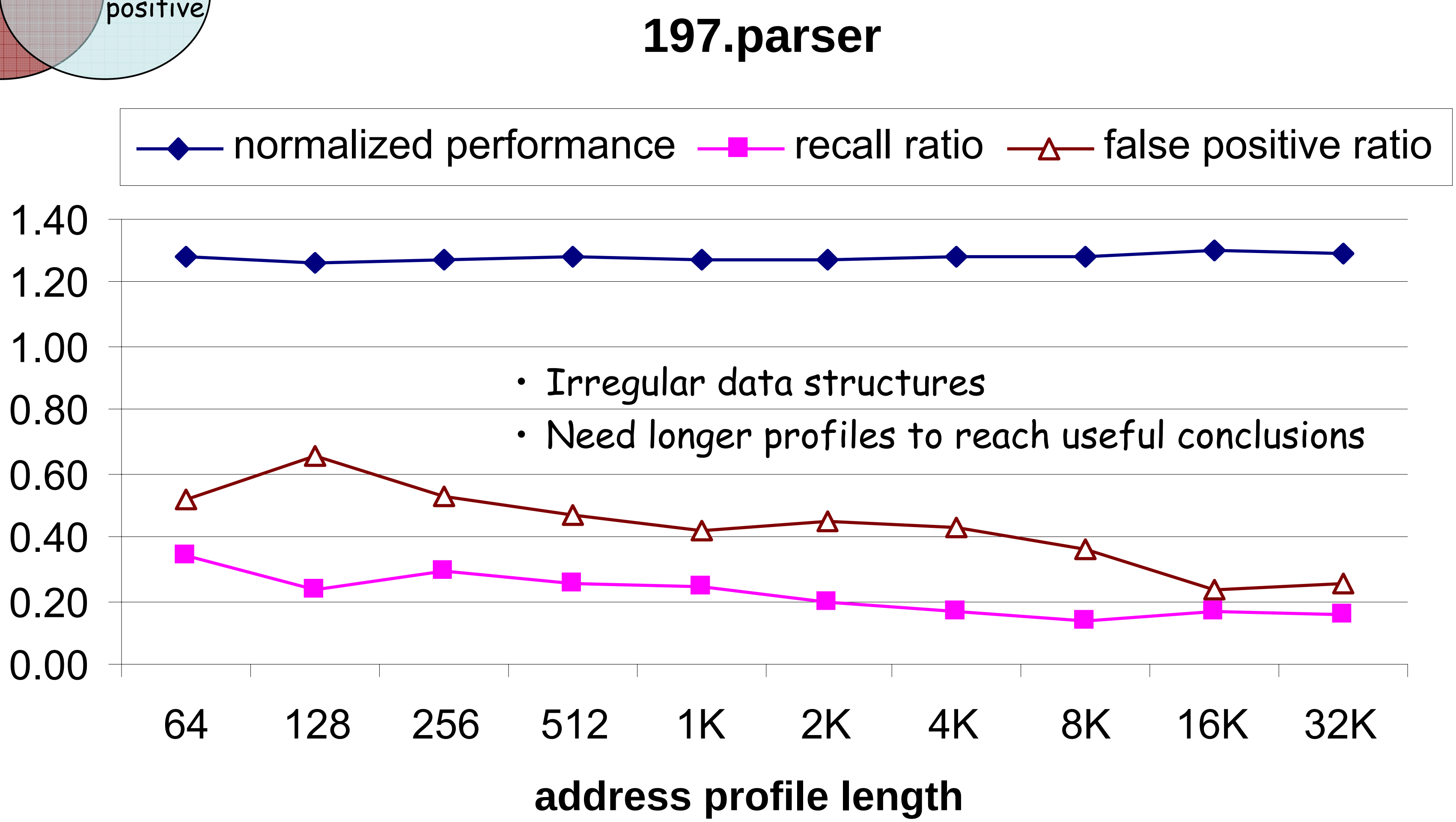




- Regular data structures
- If sampling threshold too high, starts to exceed loop bounds: miss out on profiling important loops
- Adaptive threshold is best

Mini-simulations and parameter sensitivity





Summary

- UMI is lightweight and has a low overhead
  - 1% more than DynamoRIO
  - Can be done with Pin, Valgrind, etc.
  - No added hardware necessary
  - No synchronization or syscall headaches
  - Other cores can do real work!
- Practical for extracting detailed information
  - Online and workload specific
  - Instruction level memory reference profiles
  - Versatile and user-programmable analysis
- Facilitate migration of offline memory optimizations to online setting

Future work

- More types of online analysis
  - Include global information
  - Incremental (leverage previous execution info)
  - Combine analysis across multiple threads
- More runtime optimizations
  - E.g., advanced prefetch optimization
    - Hot data stream prefetch
    - Markov prefetcher
  - Locality enhancing data reorganization
    - Pool allocation
    - Cooperative allocation between different threads
- Your ideas here…