ProfileMe: Hardware-Support for Instruction-Level Profiling on Out-of-Order Processors

Jeffrey Dean Jamey Hicks Carl Waldspurger
William Weihl George Chrysos

Digital Equipment Corporation

digital™

Motivation

- **Identify Performance Bottlenecks**
  - especially unpredictable dynamic stalls
    e.g. cache misses, branch mispredicts, etc.
  - complex out-of-order processors make this difficult
- **Guide Optimizations**
  - help programmers understand and improve code
  - automatic, profile-driven optimizations
- **Profile Production Workloads**
  - low overhead
  - transparent
  - profile whole system

Outline

- **Obtaining Instruction-Level Information**
- ***ProfileMe***
  - sample instructions, not events
  - sample interactions via paired sampling
- **Potential Applications of Profile Data**
- **Future Work**
- **Conclusions**

Existing Instruction-Level Sampling

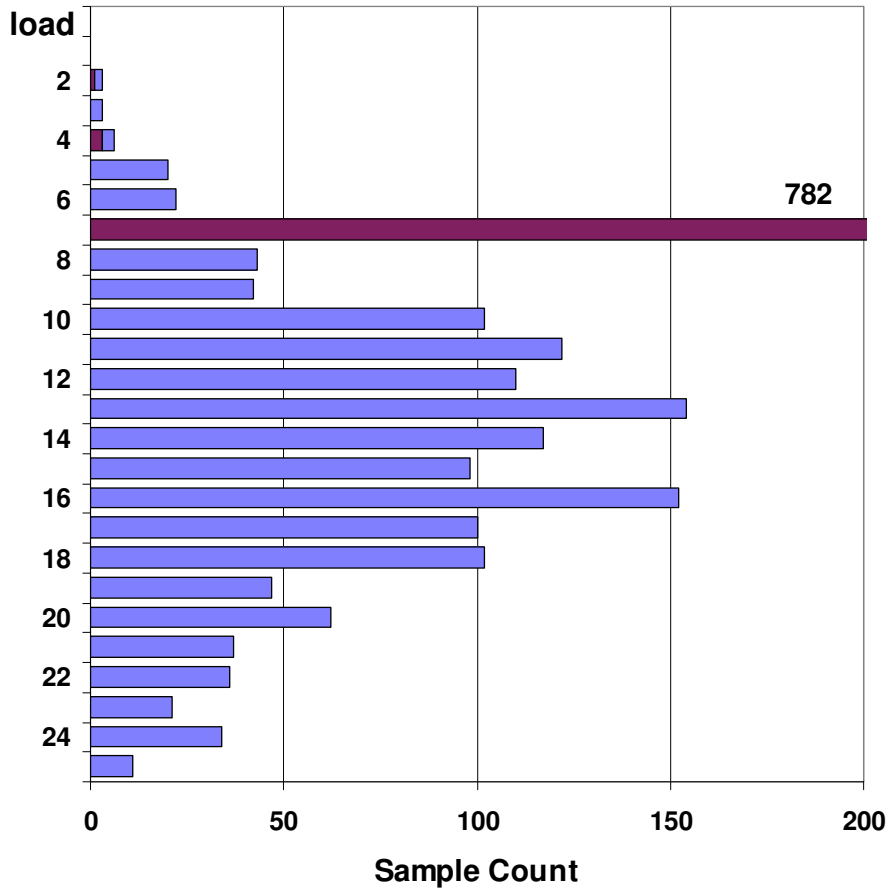
- **Use Hardware Event Counters**
  - small set of software-loadable counters
  - each counts single event at a time, *e.g.* dcache miss
  - counter overflow generates interrupt
- **Advantages**
  - low overhead *vs.* simulation and instrumentation
  - transparent *vs.* instrumentation
  - complete coverage, *e.g.* kernel, shared libs, etc.
- **Effective on In-Order Processors**
  - analysis computes execution frequency
  - heuristics identify possible reasons for stalls
  - example: DIGITAL's Continuous Profiling Infrastructure

Problems with Event-Based Counters

- **Can't Simultaneously Monitor All Events**
- **Limited Information About Events**
  - "event has occurred", but no additional context
    e.g. cache miss latencies, recent execution path, ...
- **Blind Spots in Non-Interruptible Code**
- **Key Problem: Imprecise Attribution**
  - interrupt delivers restart PC, *not* PC that caused event
  - problem worse on out-of-order processors

Problem: Imprecise Attribution

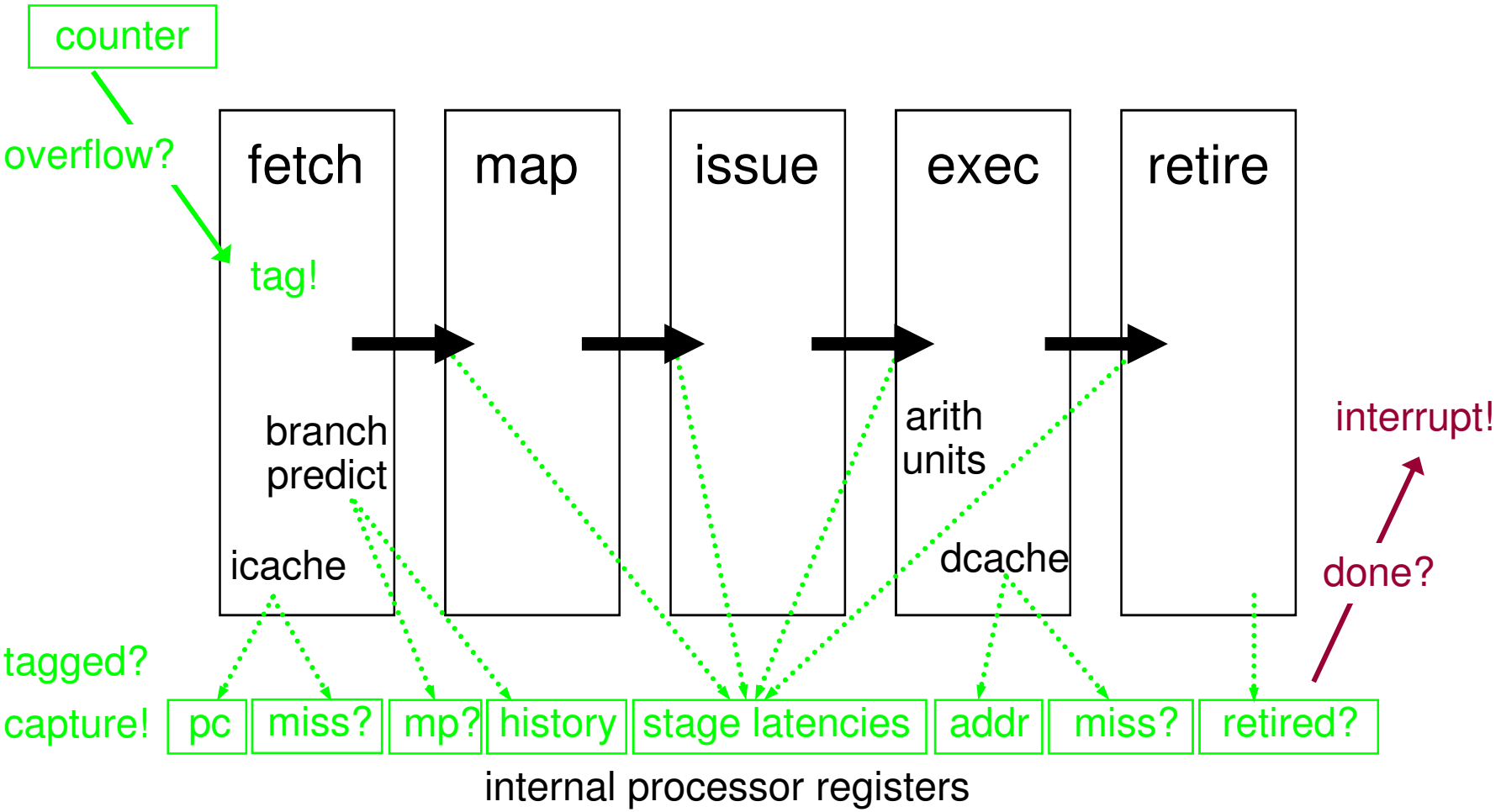
- **Experiment**
  - monitor data loads
  - loop: single load + hundreds of nops
- **In-Order Processor**
  - Alpha 21164
  - skew; large peak
  - analysis plausible
- **Out-of-Order Processor**
  - Intel Pentium Pro
  - skew and smear
  - analysis hopeless



Outline

- Obtaining Instruction-Level Information
- ***ProfileMe***
  - sample instructions, not events
  - sample interactions via paired sampling
- Potential Applications of Profile Data
- Future Work
- Conclusions

ProfileMe: Instruction-Centric Profiling



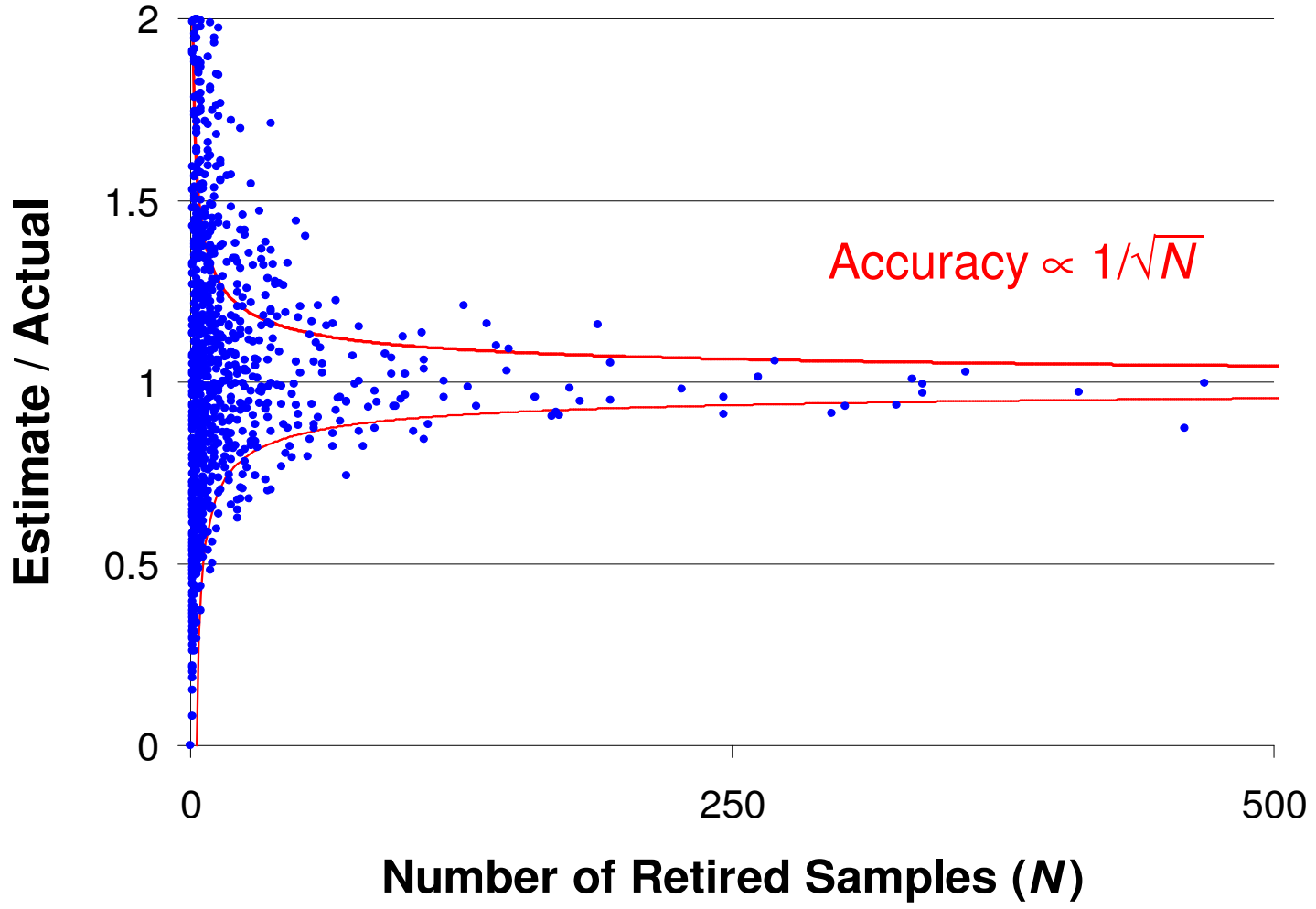
Instruction-Level Statistics

- PC + Retire Status ⇨ execution frequency
- PC + Cache Miss Flag ⇨ cache miss rates
- PC + Branch Mispredict ⇨ mispredict rates
- PC + *Event* Flag ⇨ *event* rates

- PC + Branch Direction ⇨ edge frequencies
- PC + Branch History ⇨ path execution rates

- PC + Latency ⇨ instruction stalls

Example: Retire Count Convergence



Identifying True Bottlenecks

- ***ProfileMe*: Detailed Data for *Single* Instruction**
- **In-Order Processors**
  - *ProfileMe* PC + latency data identifies stalls
  - stalled instructions back up pipeline
- **Out-of-Order Processors**
  - explicitly designed to mask stall latency
    e.g. dynamic reordering, speculative execution
  - stall does not necessarily imply bottleneck
- **Example: Does This Stall Matter?**

```
load r1, …
add  …,r1,…      average latency: 35.0 cycles
… other instructions …
```

Issue: Need to Measure Concurrency

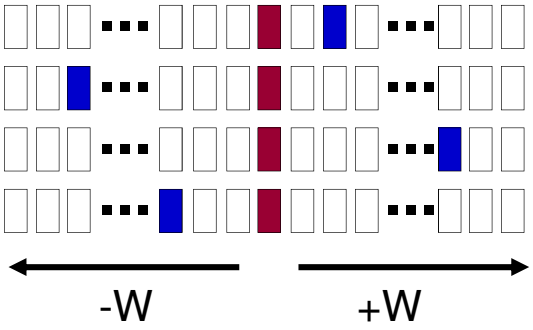
- **Interesting Concurrency Metrics**
  - retired instructions per cycle
  - issue slots wasted while an instruction is in flight
  - pipeline stage utilization

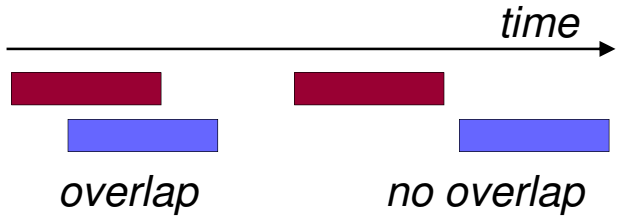
How to Measure Concurrency?

- **Special-Purpose Hardware**
  - some metrics difficult to measure
    e.g. need retire/abort status
- **Sample Potentially-Concurrent Instructions**
  - aggregate info from *pairs* of samples
  - statistically estimate metrics

Paired Sampling

- **Sample Two Instructions**
  - may be in-flight simultaneously
  - replicate *ProfileMe* hardware, add intra-pair distance
- **Nested Sampling**
  - sample window around first profiled instruction
  - randomly select second profiled instruction
  - statistically estimate frequency for *F*(first, second)





Other Uses of Paired Sampling

- **Path Profiling**
  - two PCs close in time can identify execution path
  - identify control flow, *e.g.* indirect branches, calls, traps
- **Direct Latency Measurements**
  - data load-to-use
  - loop iteration cost

Outline

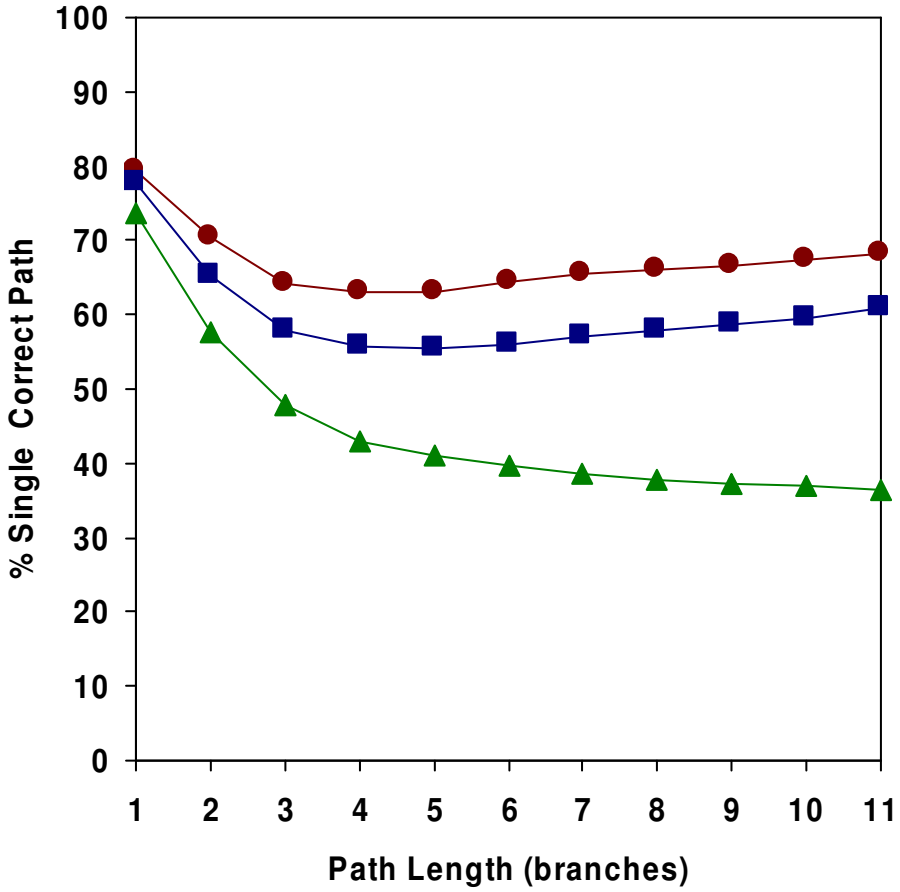
- Obtaining Instruction-Level Information
- *ProfileMe*
  - sample instructions, not events
  - sample interactions via paired sampling
- **Potential Applications of Profile Data**
- Future Work
- Conclusions

Exploiting Profile Data

- **Latencies and Concurrency**
  - identify and understand bottlenecks
  - improved scheduling, code generation
- **Cache Miss Data**
  - code stream rearrangement
  - guide prefetching, instruction scheduling
- **Miss Addresses**
  - inform OS page mapping policies
  - data reorganization
- **Branch History, PC Pairs**
  - identify common execution paths
  - trace scheduling

Example: Path Profiles

- **Experiment**
  - intra-procedural path reconstruction
  - control-flow merges
  - SPECint95 data
- **Execution Counts**
  - most likely path based on frequency
- **History Bits**
  - path consistent with global branch history
- **History + Pairs**
  - path must contain both PCs in pair



Future Work

- **Analyze Production Systems**
- **Develop New Analyses for *ProfileMe* Data**
  - "cluster" samples using events, branch history
  - reconstruct frequently-occurring pipeline states
- **Explore Automatic Optimizations**
  - better scheduling, prefetching, code and data layout
  - inform OS policies
- ***ProfileMe* for Memory System Transactions**
  - can sample memory behavior not visible from processor
  - sample cache sharing and interference

Related Work

- **Westcott & White (IBM Patent)**
  - collects latency and some event info for instructions
  - only for retired instructions
  - only when instruction is assigned particular *inum,* which can introduce bias into samples

- **Specialized Hardware Mechanisms**
  - CML Buffer (Bershad *et al.*) - locations of frequent misses
  - Informing Loads (Horowitz *et al*.) - status bit to allow SW to react to cache misses
  - can often obtain similar info by analyzing *ProfileMe* data

Conclusions

- ***ProfileMe*: "Sample Instructions, Not Events"**
  - provides wealth of instruction-level information
  - paired sampling reveals dynamic interactions
  - modest hardware cost
  - useful for in-order processors, essential for out-of-order
- **Improvements Over Imprecise Event Counters**
  - precise attribution
  - no blind spots
  - improved event collection
    e.g. branch history, concurrency, correlated events

Further Information

DIGITAL's Continuous Profiling Infrastructure project:

http://www.research.digital.com/SRC/dcpi