



# Efficient Profiling Systems

---

Major Area Presentation  
Hussam Mousa  
07/12/2005

## Committee Members:

- Chandra Krintz (chair)
- Tim Sherwood
- Rich Wolski

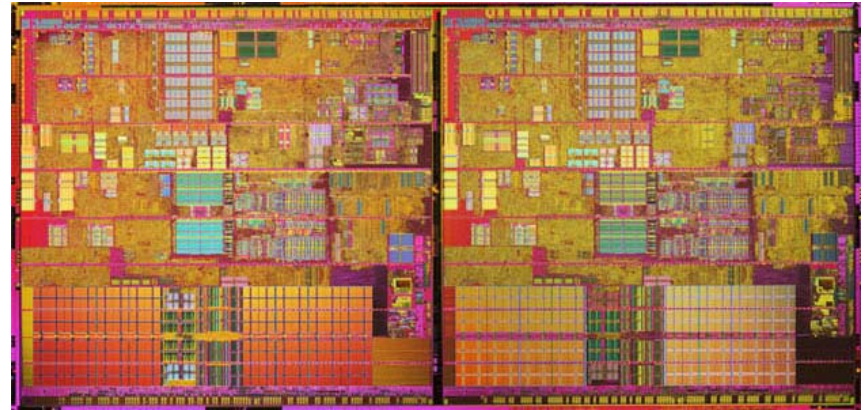
UCSB

UNIVERSITY OF CALIFORNIA  
SANTA BARBARA

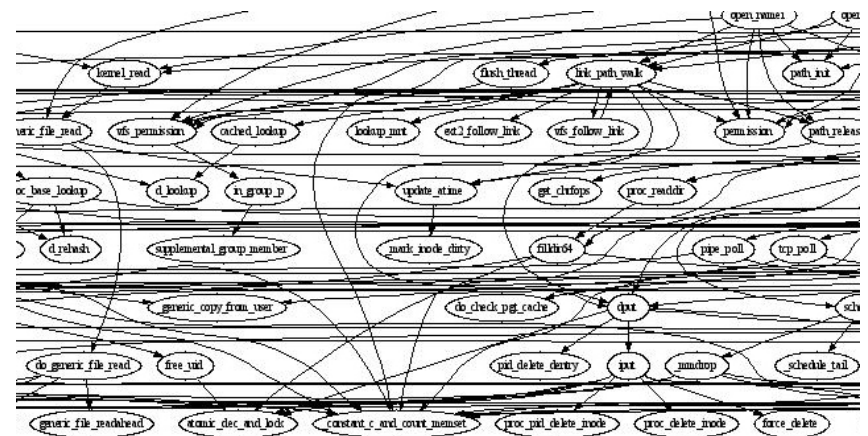


# Program Performance

- Architectural complexity is growing
- S/W - H/W interaction is hard to understand
- Static analysis fails to capture the essence of real-time dynamic execution
- Key to improving performance is understanding dynamic behavior



Source: Intel Corp.



Source: Unpublished Data, Yousseff et al.



# Profiling

---

- Profiling is used to understand the run-time behavior of applications
- A profile is:  
“A set of frequencies associated with events observed in the runtime of an application”
- Examples
  - Hot Methods
  - Hot Data Streams
  - Hot Call Pairs
- Qualities of a “good” profiler
  - Transparent
  - Efficient
  - Flexible
  - Fast
  - Accurate



# Outline

---

- Profile usage
- Efficient profiling approaches
- Software profiling systems
- Hardware profiling systems
- Hybrid profiling systems
- Future work



# Profile Usage

---

- Program analysis
- Feedback-directed optimization
- Debugging and bug isolation



# Offline Program Analysis

---

- Understand application/architecture interaction
  - Recently used in understanding VM and OO program interactions  
[Sweeney'04, Hauswirth'04, Eeckhout'03]
- Identify memory access patterns [Chilimbi '02]



# Feedback-Directed Static Optimization

---

- Make profile information available to compiler for improved optimization
- Traditionally off-line
- Better than static optimization alone
- Major research until today
- Limitations include
  - Sometimes code needs to be shipped unoptimized
    - E.g. Mobile code and dynamically linked applications.
  - Fails to capture specific behavior based on user inputs



# Online Feedback-Directed Optimization

---

- Late nineties trend in performing optimization in conjunction with execution
  - Closely linked with VM Just In Time Compilation
  - [Arnold'02] Described such a system for Java
- Also used in non-VM settings
  - Dynamic data prefetching [Chilimbi et al 02]
  - Online page migration [Hollingsworth '04]



# Outline

---

- Profiling applications
- Efficient profiling approaches
- Software profiling systems
- Hardware profiling Systems
- Hybrid profiling Systems
- Future Work



# Efficient Profiling Approaches

---

- **Sampling**
  - **Software**
  - **Hardware**
- **Software Instrumentation**
- **Dedicated hardware extensions**
- **Hybrid Combinations of the above**
- **Software Tomography**



# Sampling

---

- To estimate the profile by probing the running application periodically for the purpose of gathering runtime information
- Approaches to sampling vary on:
  - Establishing the sampling periodicity
  - Executing the probing functionality (data gathering)



# Sampling: When To Sample

---

- External timer/event
  - Time (every  $n$  clock cycles)
  - Thread switches
  - Garbage collection intervals
- Internal program events
  - Every  $n$  procedure calls and/or loop iterations



# Sampling: Gathering Data

---

- Dedicated hardware profiling structures
- External probing of runtime structures
  - Runtime stack
  - Register values (e.g. PC)
- Execution of “instrumented” profiling instructions that are part of program



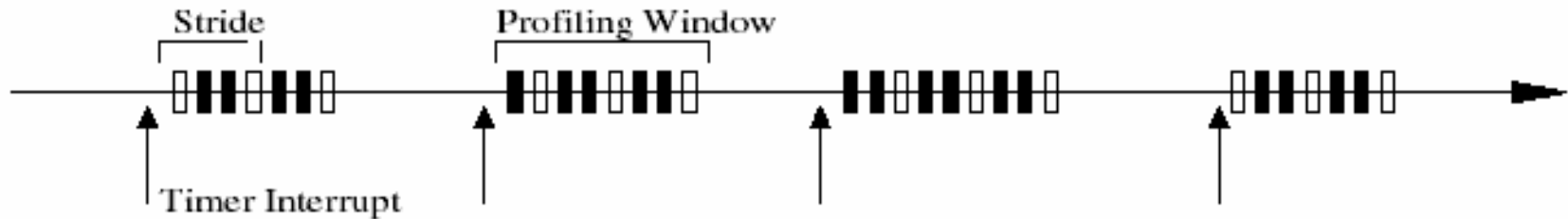
# Timer Sampling

---

- Popular profiling technique in VM [Whaley '00]
  - Uses a timer interrupt
  - Gathers data from each thread in queue
    - PC, Stack pointer, CPU time
  - Construct PCCT and estimate hot methods
- Advantages
  - Relatively inexpensive
- Disadvantages
  - Can only gather limited types of profiles accurately
  - Does not correlate with program events
  - Maximum sampling frequency too low
    - E.g. every 10 ms == 1/10M instructions on 1 GHz CPU

# Stride-enabled Timer Sampling

- Combine timer-based and program event-based sampling [Arnold'05]

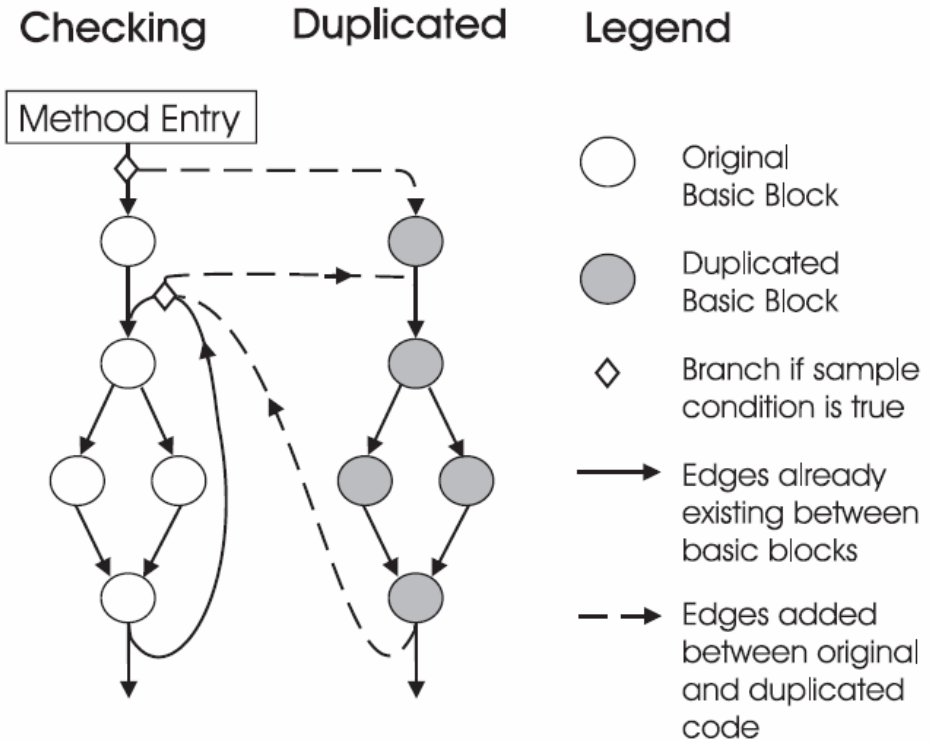


- Parameters include:
  - How many samples to collect per interrupt
  - Spacing "stride" that separates the samples
- Used to estimate a dynamic call graph
- Approx. 70%+ accuracy with 1-2% overhead
  - Compare to ~37% on similar budget

Source: Arnold et al '05

# Sampling Framework for Instrumentation [Arnold'01]

- Duplicate code
  - One with profiling instructions
  - One with instructions that control transitions between the copies
- Advantages
  - Sample according to program behavior
  - Flexible control of sampling parameters
  - Collects larger variety of profiles
- Drawbacks
  - "Basic overhead"
  - Code duplication
  - Side Effect on Program



Source: Arnold '01

Avg of basic OH. 4.9% (1-10%)

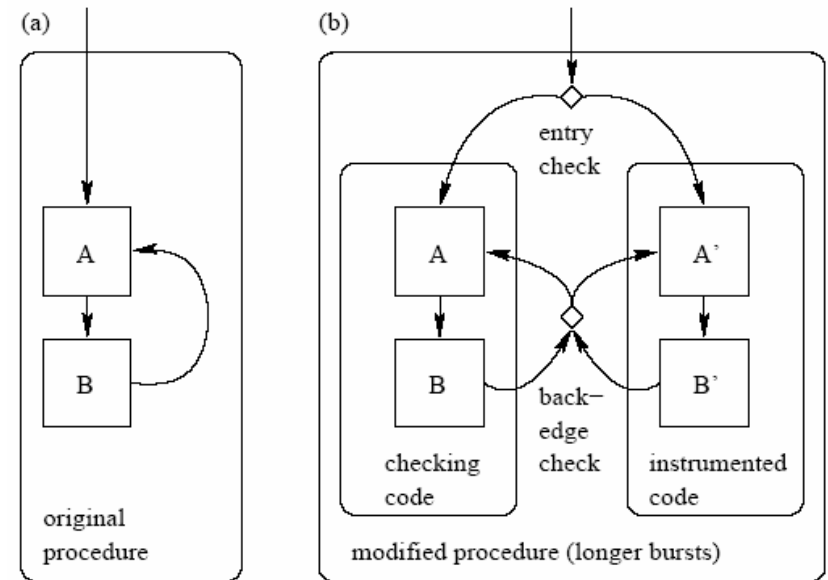
Avg of profiling OH 4.2% (1/100 freq.)

Accuracy improvement 63% -> 84%

# Variable Burst-length Sampling

## [Hirzel'01]

- Added ability to **stay** in duplicated region for **multiple sampling "bursts"**
- Improved quality of *temporal* profiles
- Measured impact of framework on binary executables
- **Bursty** tracing
- Same pros/cons as the sampling framework



Source: Hirzel'01

- Avg of basic OH. 16%(6-35%)
- Reduced OH by about 50% by reducing number of checks
- Accuracy improvement ~15% on average for same sampling frequency



# Other Software Sampling Systems

---

## ■ Ephemeral instrumentation [Traub'00]

- Replace instruction of interest with an unconditional branch to a profiling stub
- Dynamically revert to original instruction when profiling is concluded
- Used to gather branch biases and edge profiles

## ■ Dyninst [Hollingsworth '00]

- API for dynamic code instrumentation
- Similar methodology but more elaborate support
- General purpose but can (and was) used for profiling.



# Outline

---

- Profiling applications
- Efficient profiling approaches
- Software profiling systems
- **Hardware profiling Systems**
- Hybrid profiling Systems
- Future Work



# Hardware Profiling Systems

---

- Performance counters
  - Efficient and flexible value sampling [Weihl '00]
  - Memory profiling via hardware counters [Itzkowitz '03]
- Specialized hardware add-on
  - Identifying program *hot* spots [Hwu '99]
  - Frequent loop detection [Vahid '03]
- Dedicated profiling co-processor [Sohi '01]



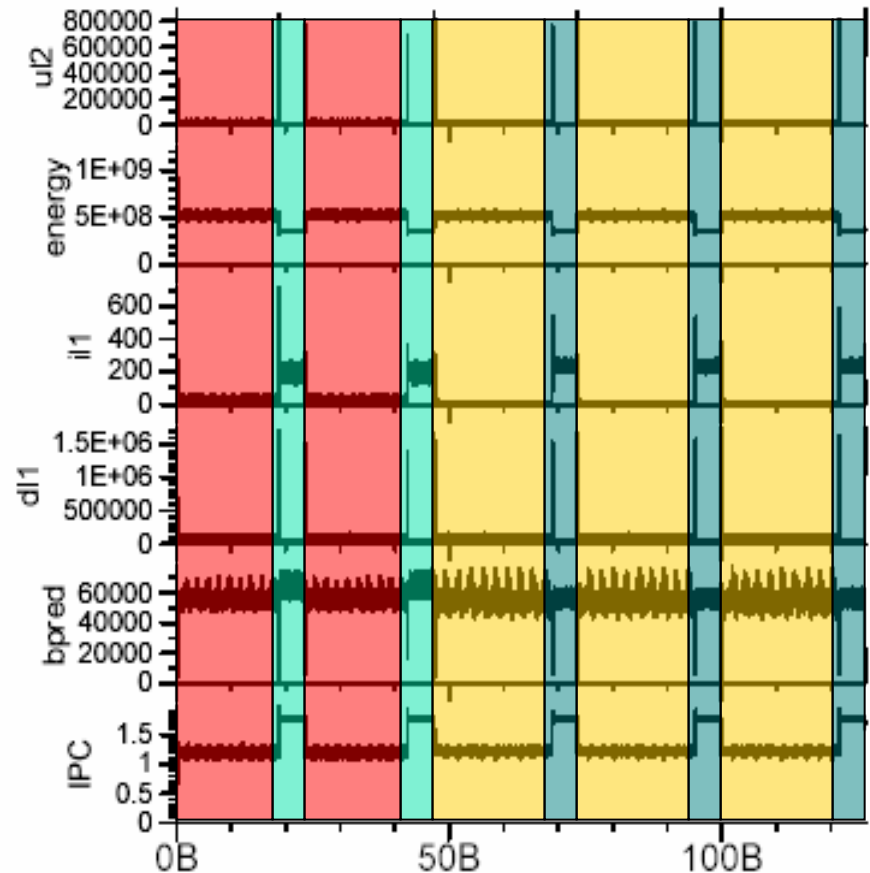
# Outline

---

- Profiling applications
- Efficient profiling approaches
- Software profiling systems
- Hardware profiling Systems
- Hybrid profiling Systems
  - Phase-Aware Profiling
  - Hybrid Profiling Support
- Future Work

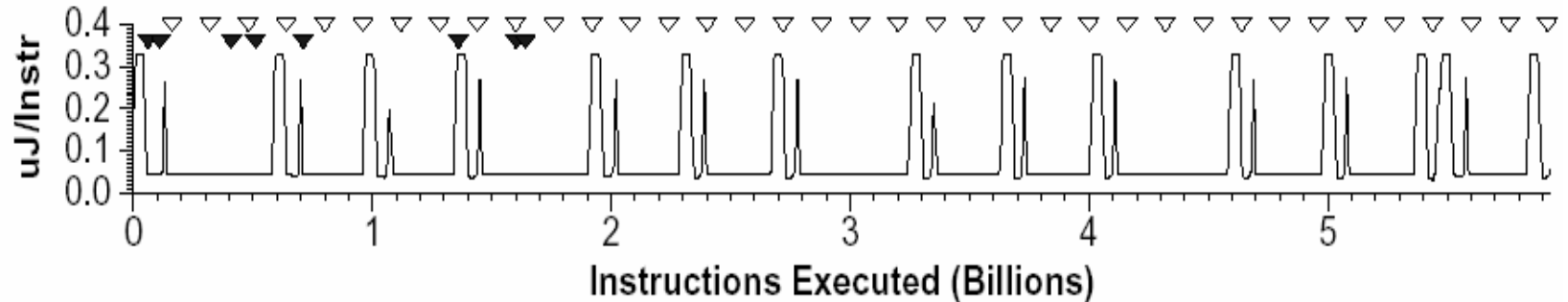
# Phase Behavior

- Programs exhibit repeating patterns in execution, i.e., phases [Sherwood'03]
- Two intervals are in same phase if they exhibit similar behavior
- This behavior can be exploited for efficient profile collection [Nagpurkar '05]



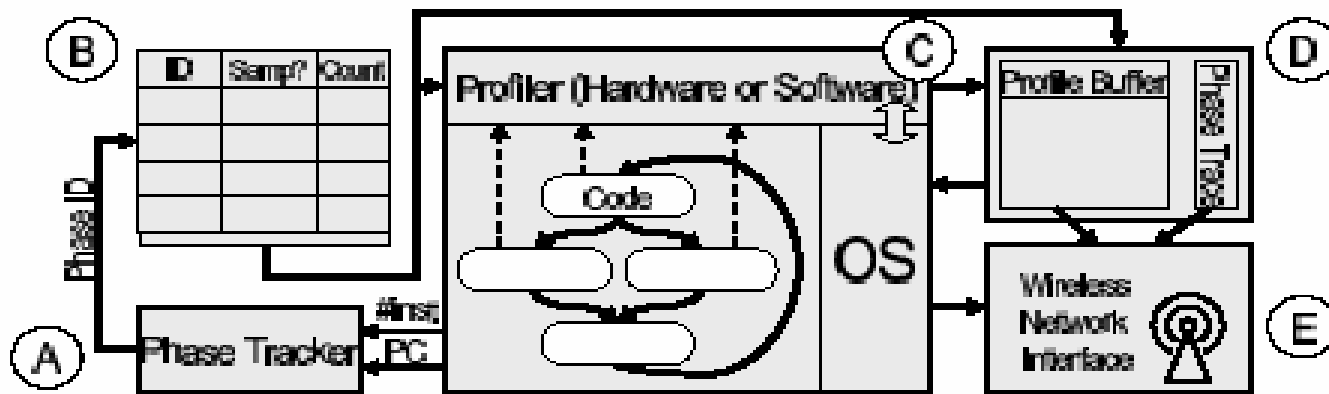
Source: Sherwood et al '03,  
Art from Major Area Presentation, Priya Nagapurkar

# Phase Aware Sampling



- HW metric patterns correspond to different phases
- A periodic sampler (white triangles)
  - Continue to gather samples throughout the execution
- Phase-aware profiler (black triangles)
  - Only gathers a profile when a new phase is encountered
  - Highly accurate (>95%); Low overhead (55-80% impr.)

# Phase Aware Sampling



Source: Nagpurkar05

- Phase tracker used to decide **when** to profile
- Question remains: **how** to toggle profiling?



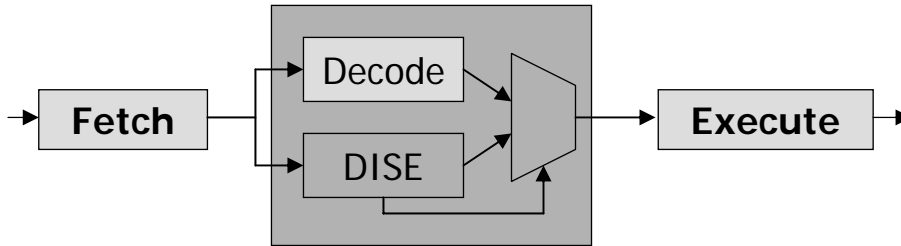
# Recap

---

- So far we have seen that
  - Software profiling can be
    - Accurate
    - Flexible
    - Easy to Manage
  - Hardware systems are
    - Inexpensive
    - Transparent
  - Phase awareness (hybrid) improves accuracy but needs flexible profiling support

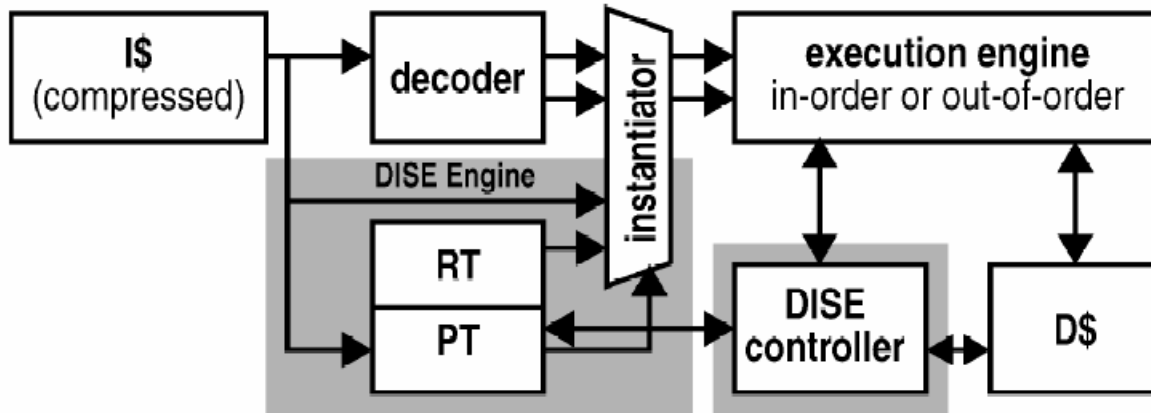
# Our Work: [Mousa'05]

## Hybrid Profiling Support (HPS)



- Extends and employs *Dynamic Instruction Stream Editing (DISE)* to define an efficient sample-based profiling system
- HPS Benefits:
  - Flexible software-like usage model
  - Low/negligible, hardware-like overhead
  - Easy to manage at run time

# HPS: Dynamic Instruction Stream Editing (DISE)



Source: Corliss'05

- Processor extension that matches an instruction and replaces it with an alternate stream of parameterized instructions [Corliss'03]
- DISE-private registers + HW data structures
- User specifies patterns & replacements via specification language (DISE Productions).



# HPS: Sample-based Profiling Using DISE

---

- ***Bursty*** tracing using DISE
  - Manage when to sample via a sampling flag
  - Profile instructions of interest if flag is set
- Advantages
  - Transparent
  - Flexible
- Disadvantages
  - TOO many instrumentations
    - ALL calls and backwards branches
    - ALL instructions of interest regardless of the state of sampling flag
    - Overhead up to 106%



# HPS: Profiling Productions

## Sampling Framework Productions

# We are not profiling, check for procedure call and backward branch  
# increment the sampling counter - CC1- and fail production

**P1:** T.OPCLASS == proc\_call && !overflow\_1 ⇒ null, CC1

**P2:** T.OPCLASS == branch && T.PC < T.Target && !overflow\_1  
⇒ null, CC1

# We are profiling, check for procedure call and backward branch,  
# increment the burst counter - CC2 - and fail production

**P3:** T.OPCLASS == proc\_call && overflow\_1 && ! overflow\_2  
⇒ null, CC2

**P4:** T.OPCLASS == branch && T.PC < T.Target  
&& overflow\_1 && ! overflow\_2 ⇒ null, CC2

# We are done profiling because burst counter status (overflow\_2)  
# has overflowed. reset counters (this unsets overflow\_1 and  
# overflow\_2) causing sampling to stop

**P5:** T.OPCLASS == proc\_call && overflow\_1 && ! overflow\_2  
⇒ null, CC3

**P6:** T.OPCLASS == branch && T.PC < T.Target  
&& overflow\_1 && overflow\_2 ⇒ null, CC3

**CC1:** inc\_1;

**CC2:** inc\_2;

**CC3:** set\_1(sampleFreq); set\_2(burstLength);

## Profile Type Productions

### Hot Data Stream Analysis

**P1:** T.OPCLASS == mem\_op && overflow\_1 ⇒ R1, null

**R1:** call(DataStream\_handler)  
T.INSN

### Hot Call Pair Analysis

**P2:** T.OPCLASS == proc\_call && overflow\_1 ⇒ R2, null

**R2:** call(CallPair\_handler)  
T.INSN

### Hot Method Analysis

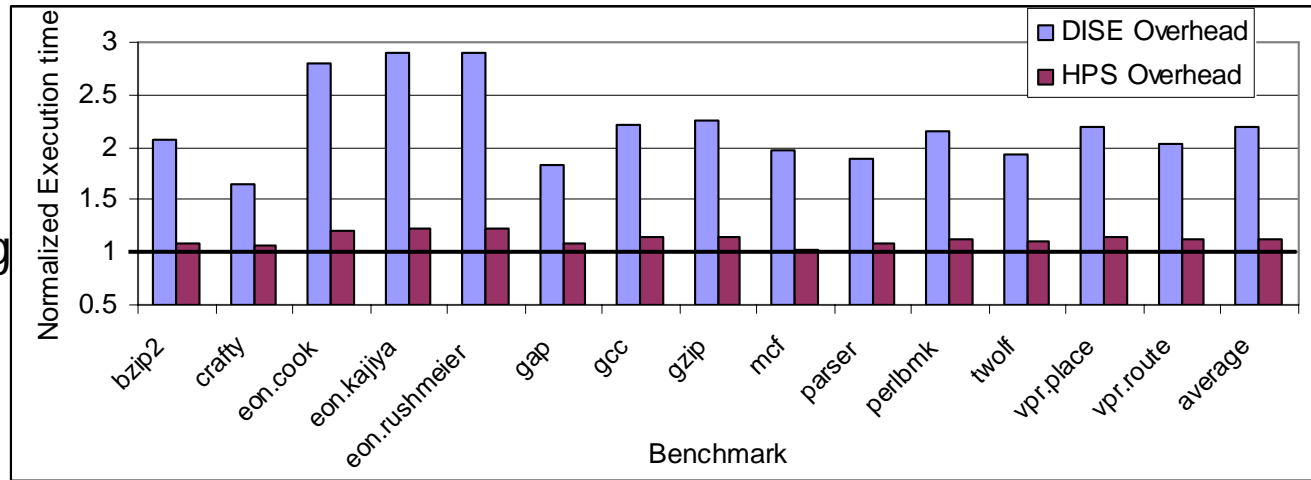
**P3:** T.OPCLASS == proc\_call

|| (T.OPCLASS == branch && T.PC < T.Target)  
&& overflow\_1 ⇒ R3, null

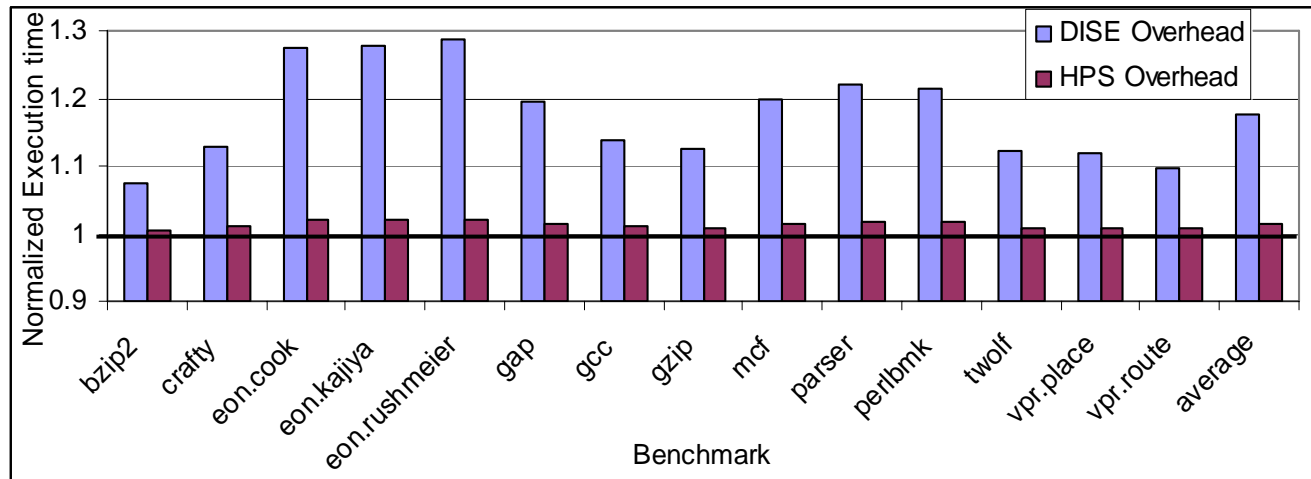
**R3:** call(HotMethod\_handler)  
T.INSN

# HPS Overhead

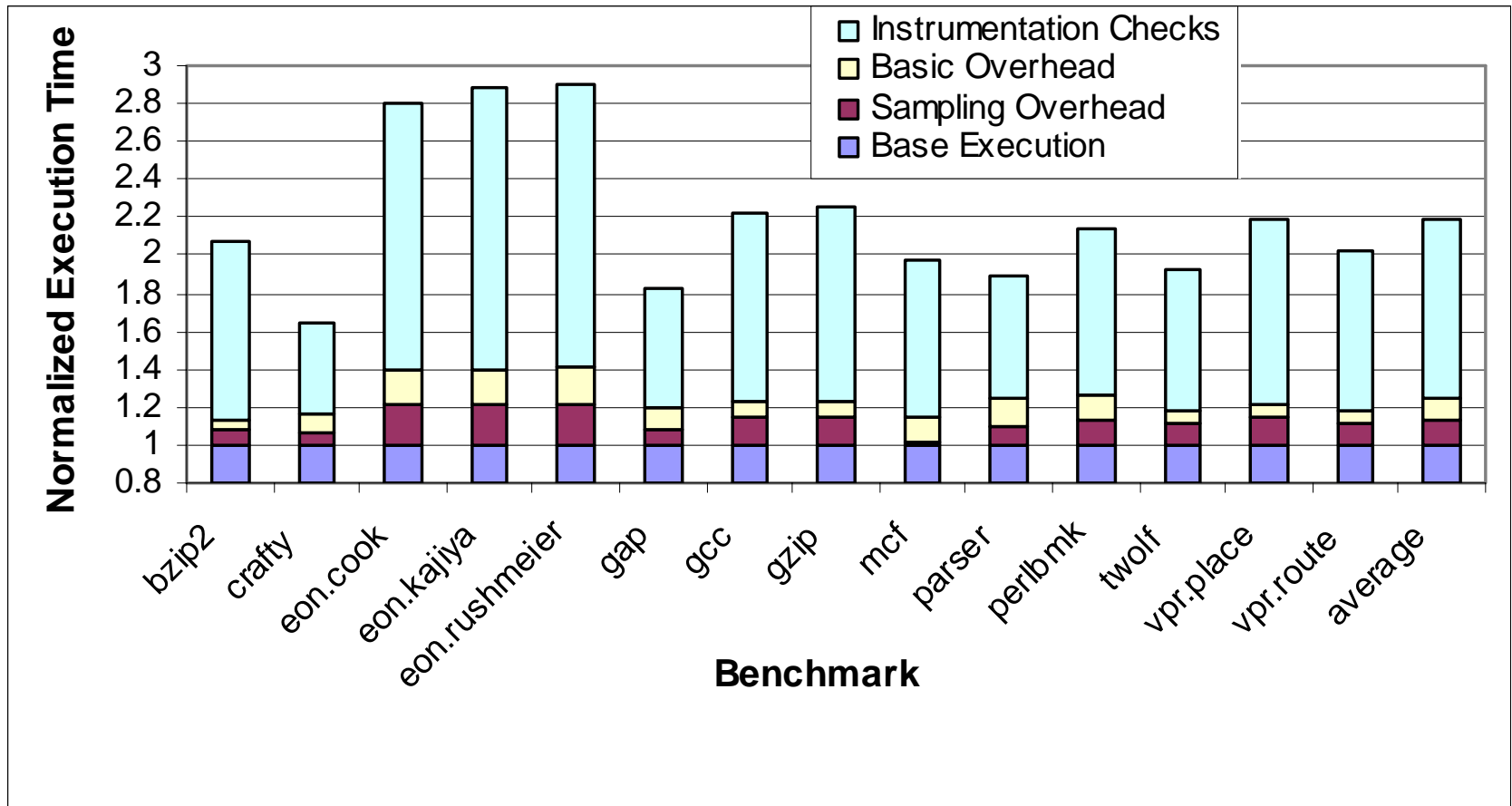
Hot data  
stream sampling



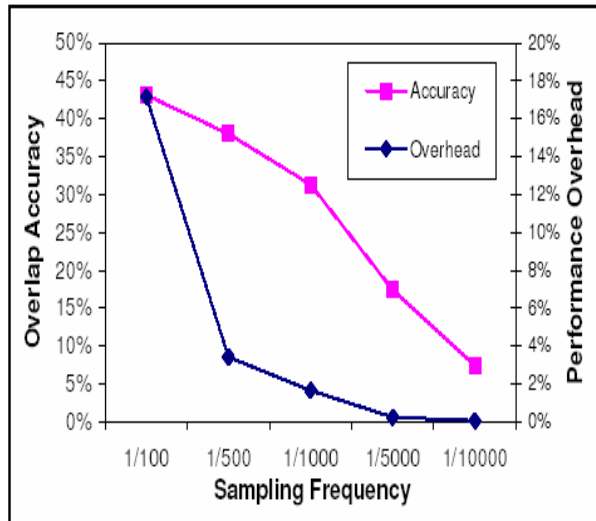
Hot call  
pair sampling



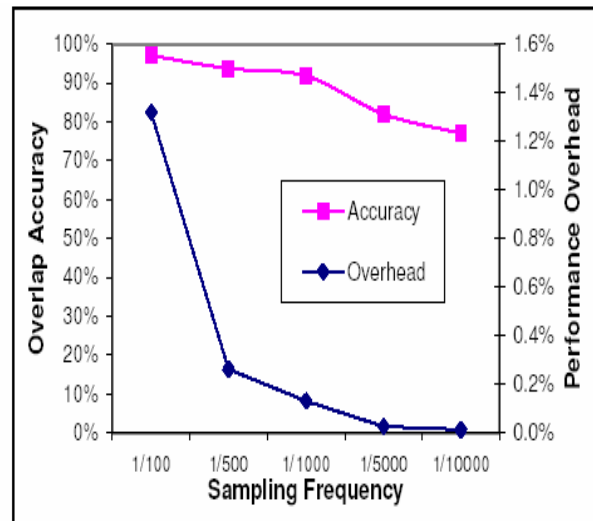
# HPS: Overhead Breakdown



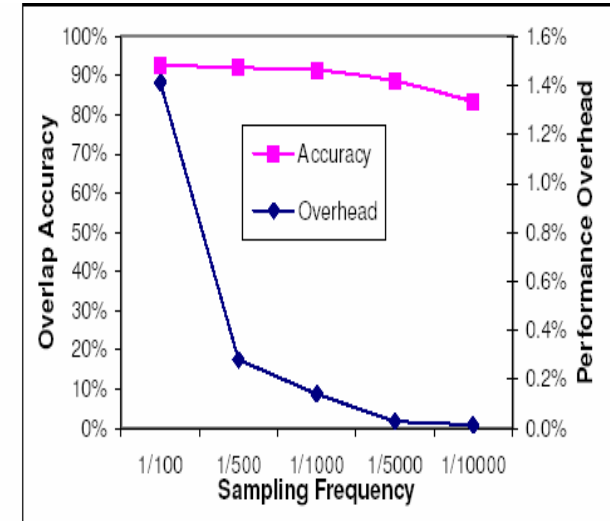
# HPS Overhead/Accuracy Tradeoffs



Hot data stream



Hot call pair

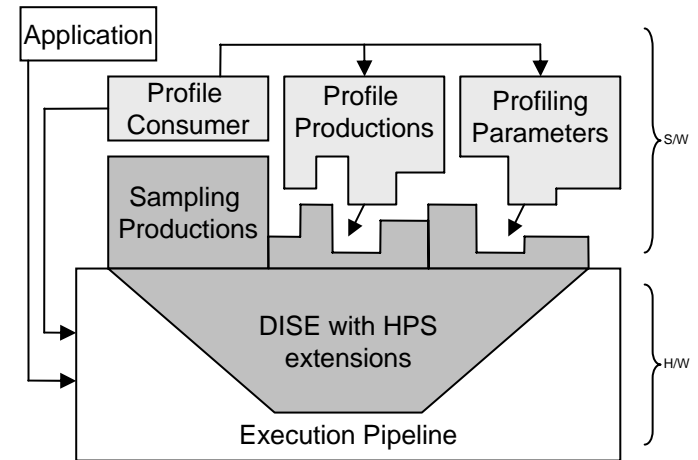


Hot method

# HPS Summary

■ The HPS Framework consists of

- An optimized DISE engine
- Sampling productions
- Simple profile definition primitives
- Mechanism for managing online profiling





# Conclusion

---

- Efficient online profiling is critical for harvesting performance of next generation software and hardware
- Software techniques are flexible but have high overhead
- Hardware techniques are inexpensive but more rigid and may not be universally available
- Hybrid techniques hold great potential since they combine the advantages of both without the drawbacks



# Future Work

---

- Online dynamic optimizers require profilers that:
  - Easy to deploy and undeploy
  - Exhibit little or no overhead
  - Flexible
  - Fast convergence
  - Accurate
- Between phase-aware profiling and HPS all of the above are achieved
- Investigate the potential of rapid profile collection for aggressive dynamic optimization