IBM Research

Accurate, Efficient, and Adaptive Calling Context Profiling

Xiaotong Zhuang Mauricio J. Serrano Harold W. Cain Jong-Deok Choi

Presented by
Brian Norris

February 1, 2012

© 2006 IBM Corporation

Overview

- **Earliest of four “calling context” papers we've studied**
  - **Bond and McKinley, “Probabilistic Calling Context” (2007)**
  - **Sumner, Zheng, Weeratunge, and Zhang, “Precise Calling Context Encoding” (2010)**
  - **Bond, Baker, and Guyer, “Breadcrumbs: Efficient Context Sensitivity for Dynamic Bug Detection Analyses” (2010)**
- **All reference this paper**
  - **All have some criticism for this paper**

© 2006 IBM Corporation

Outline

- Introduction
- Existing Approaches
- Our Approach: Adaptive Bursting
- Results
- Related Work
- Conclusion and Future Work

© 2006 IBM Corporation

Motivation

- **What is a calling context ?**
  - **Methods that are on the stack when an event happens**

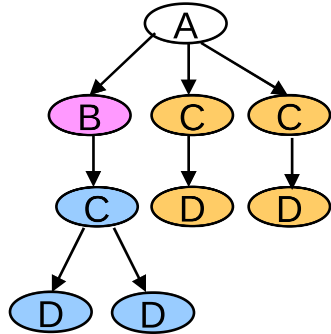
```
java.lang.ArrayIndexOutOfBoundsException: 3 >= 3
    at java.util.Vector.elementAt(Vector.java:427)
    at junit.samples.VectorTest.testElementAt(Vector
    at sun.reflect.NativeMethodAccessorImpl.invoke0(
    at sun.reflect.NativeMethodAccessorImpl.invoke(N
    at sun.reflect.DelegatingMethodAccessorImpl.invo
    at java.lang.reflect.Method.invoke(Method.java:3
```

```
Call Trace:
 [] __handle_sysrq+0x58/0xc6
 [] write_sysrq_trigger+0x23/0x29
 [] vfs_write+0xb6/0xe2
 [] sys_write+0x3c/0x62
 [] syscall_call+0x7/0xb
```

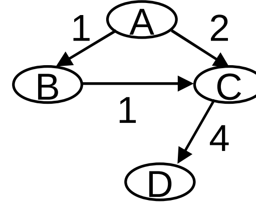
- **Applications of calling context information**
  - **Optimizations based on profiling: inlining, devirtualization, etc..**
  - **Program understanding**
    - **Large server applications have a complex method-level profile**
  - **Debugging**

© 2006 IBM Corporation

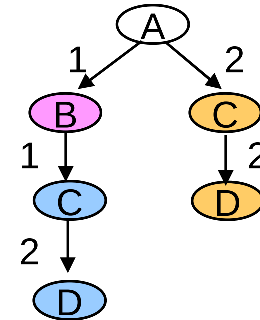
Examples



Call Tree (CT)



Call Graph (CG)



Calling-Context Tree (CCT)

- **Call Tree: complete calling context info, but huge tree**
- **Call Graph: no context information**
- **Calling Context Tree: merges identical child nodes of the same parent node → *much* smaller than Call Tree**

© 2006 IBM Corporation

Collecting Calling Context Profile

- **Existing approaches incur high-overhead**
  - **OO program: highly *interprocedural***
  - **Exhaustive: 50x overhead?**

- **New approach**
  - **Reduce overhead while maintaining high accuracy**
  - **Use an adaptive scheme:**
    - **Bursty mode sampling**
    - **Disable bursts when similar contexts are found**
    - **Re-enable bursts when accuracy could be decreased**

© 2006 IBM Corporation

Contributions

- **Improved:**
  - **Efficiency**
  - **Accuracy**
  - **Portability (i.e., doesn't rely on HW features)**
- **New metric (overlap vs. hot-edge coverage)**
- **Rigorous comparison of efficiency and accuracy**

© 2006 IBM Corporation

Outline

- Introduction
- Existing Approaches
- Our Approach: Adaptive Bursting
- Results
- Related Work
- Conclusion and Future Work

© 2006 IBM Corporation

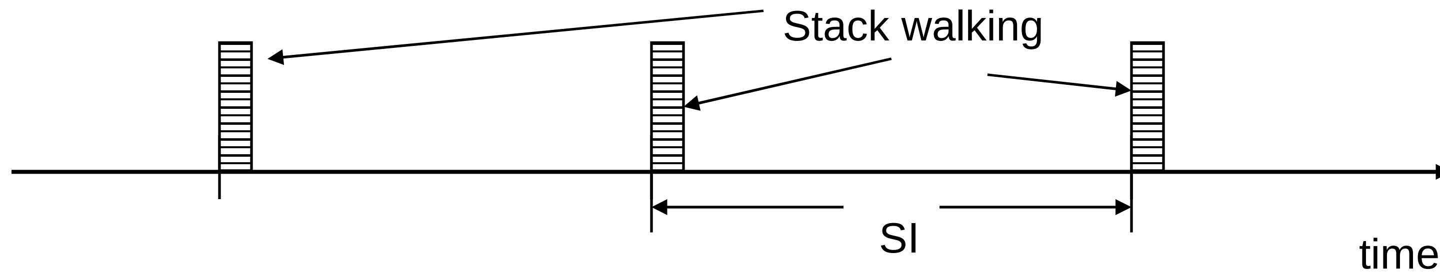
Building CCT: Exhaustive Approach

- **Capture all calls and returns**

- **High instrumentation cost:**
  - **Authors' experiments indicate 50 times slowdown based on JVMPI**

© 2006 IBM Corporation

Building CCT: Sampled Stack Walking

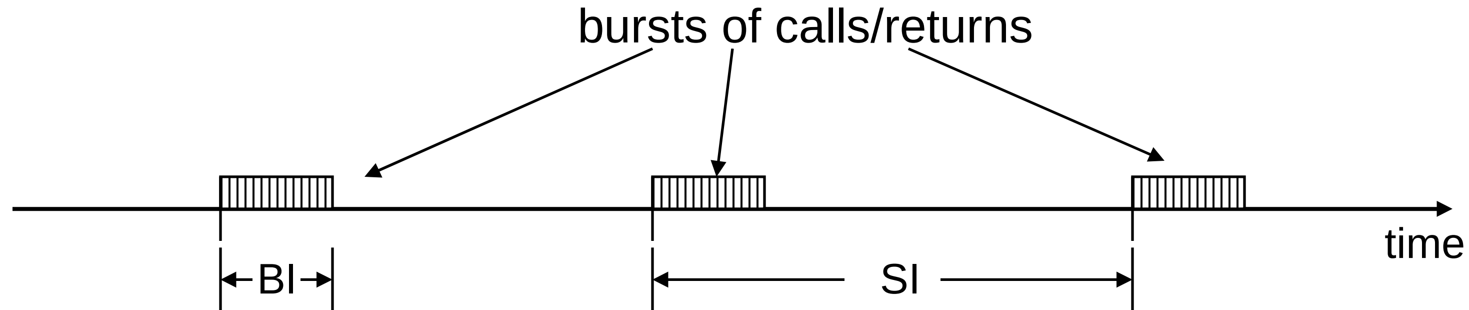


SI: Sampling Interval

- **At each sampling point, walk the full stack back**
  - **What about long method calls?**
- **Stack-walking is quite efficient (at 10 ms interval)**
  - **But on some platforms, the interval cannot be smaller**
  - **Sacrifice accuracy**

© 2006 IBM Corporation

Building CCT: Bursting



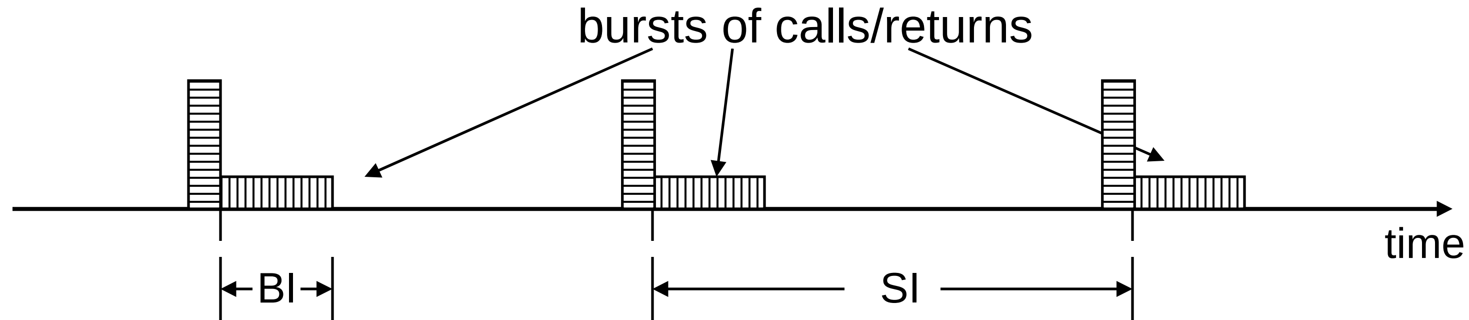
BI: Burst Interval

SI: Sampling Interval

- **At each sampling point, capture a burst of method calls and returns**
- **Useful to build call graph profiles, not useful for CCT**

© 2006 IBM Corporation

Building CCT: Static Bursting



BI: Burst Interval

SI: Sampling Interval

- **Perform stack walking before each burst**
- **Gets expensive with longer burst intervals or shorter sampling intervals for a precise CCT**

© 2006 IBM Corporation

Outline

- Introduction
- Existing Approaches
- Our Approach: Adaptive Bursting
- Results
- Related Work
- Conclusion and Future Work

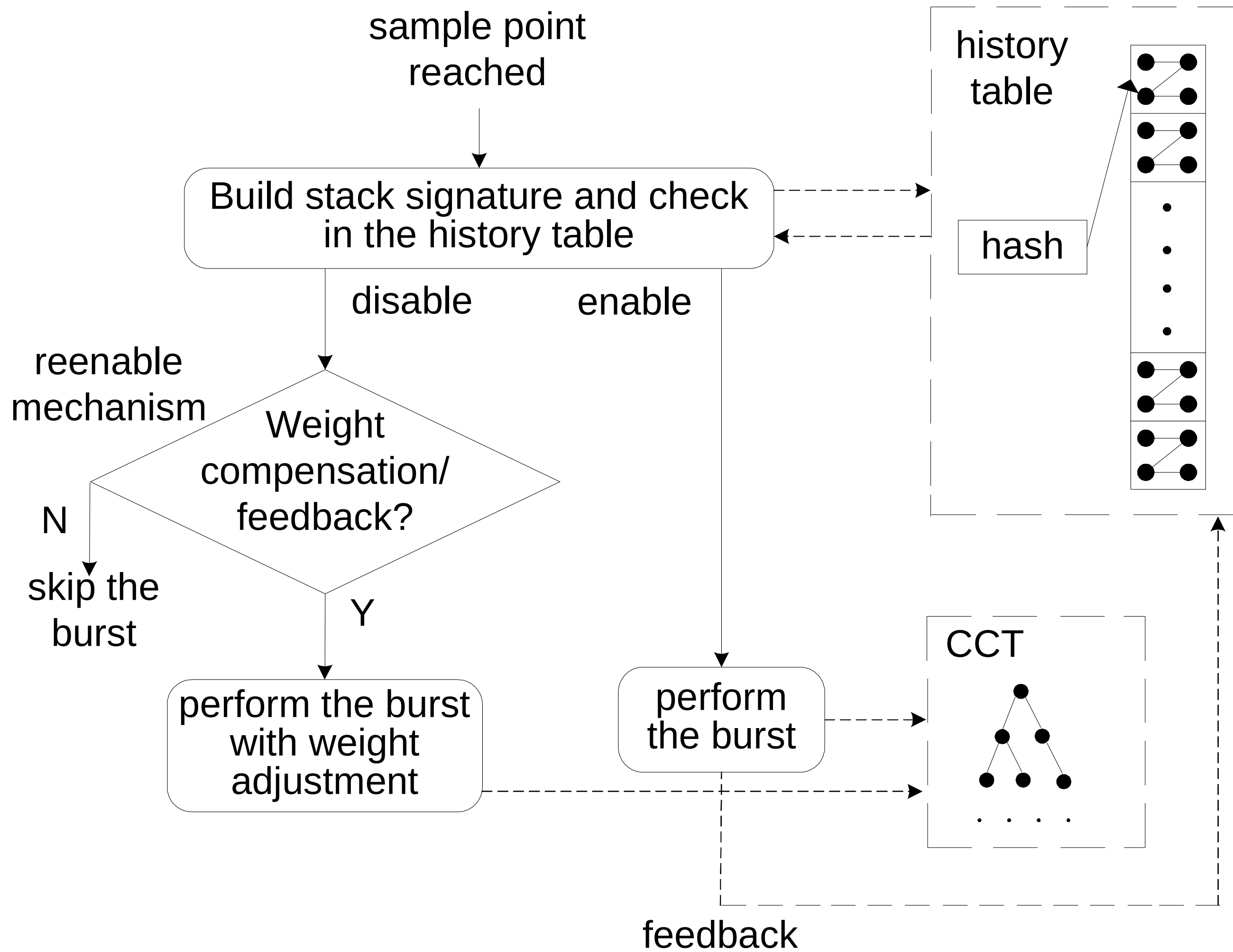
© 2006 IBM Corporation

Adaptive Bursting: Reduce Redundant Bursts

- **Control flow is highly repetitive (e.g. loops) → bursts are redundant**
- **Selectively disable previously sampled calling contexts**
- **Call stack information can serve as a good signature → a hash of methods on the stack at the beginning of the burst**
- **Use a history table to record if similar burst has occurred earlier**

© 2006 IBM Corporation

Overview of Adaptive Bursting



© 2006 IBM Corporation

Adaptive Bursting: Weight Compensation

- **Disabling redundant bursts loses CCT edge weights**

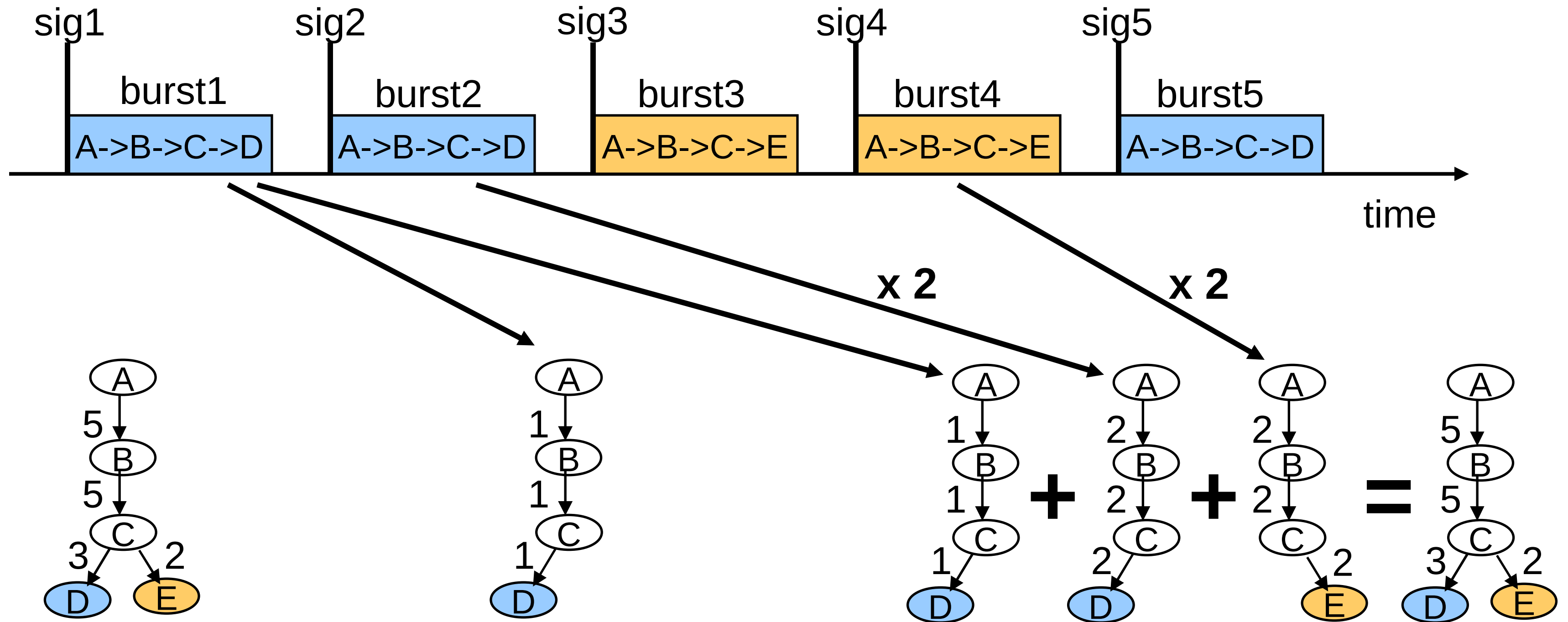
- **Statistically reenable some of the disabled bursts, with a Reenable Ratio (*RR*) between 0 and 1.**

  - **The probability a burst is reenabled is *RR*. Every counter value added to the CCT is multiplied by 1/*RR*.**

  - **Ex. *RR* = 0.25, enable 1 per 4 disabled bursts, multiply each counter by 4.**

© 2006 IBM Corporation

Weight Compensation Example



Static Bursting

Adaptive Bursting w/o reenable

Adaptive Bursting w/ reenable ratio=0.5

© 2006 IBM Corporation

Outline

- Introduction
- Existing Approaches
- Our Approach: Adaptive Bursting
- Results
- Related Work
- Conclusion and Future Work

© 2006 IBM Corporation

Benchmarks & Setup

NAME	DESCRIPTION	PLAT FORM	Call Graph		CCT
			# nodes	# edges	# nodes
checkit	jvm98 - check program	x86	988	1827	9115
compress	jvm98 - Modified Lempel -Ziv method	x86	721	1227	7581
db	jvm98 - database simulation	x86	744	1310	8666
ipsixql	Persistent XML -database	x86	802	1330	9439
jack	jvm98 - Java Parser Generator	x86	987	1996	58422
javac	jvm98 - java compiler	x86	1505	4144	917986
specjbb	Java business application	x86	2467	5368	66792
jess	jvm98 - Expert Shell System	x86	1101	2106	24194
kawa	Java -based Scheme system	x86	2454	5496	430557
mpegaudio	jvm98 - decompress audio files	x86	898	1516	14019
JAS	SpecJAppServer2004:3 tier java server	AIX	6918	14597	256189

- **Two configurations: Windows/Sun JVM, AIX/J9-3tier**
- **Sampling Interval=10ms, Burst Interval=0.2ms.**
- **Re-enable Ratio=0.05, History Table 2048 entries.**

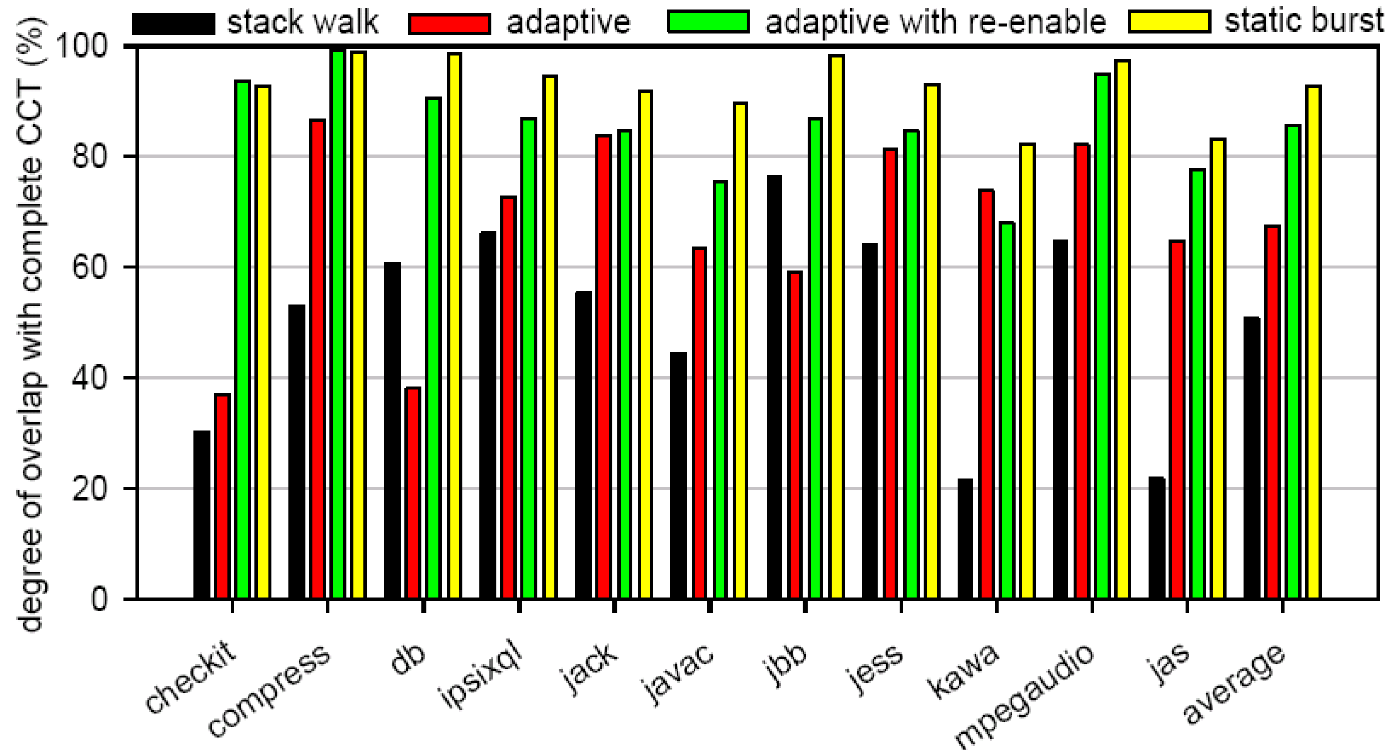
© 2006 IBM Corporation

Measuring the Accuracy of Calling Context

- **Degree of Overlap**
  - **Focus on measuring the completeness of a CCT against the complete CCT**
- **Hot-edge Coverage**
  - **Focus on the coverage of hot edges (above a threshold)**
- **Formal definitions explained in the paper.**

© 2006 IBM Corporation

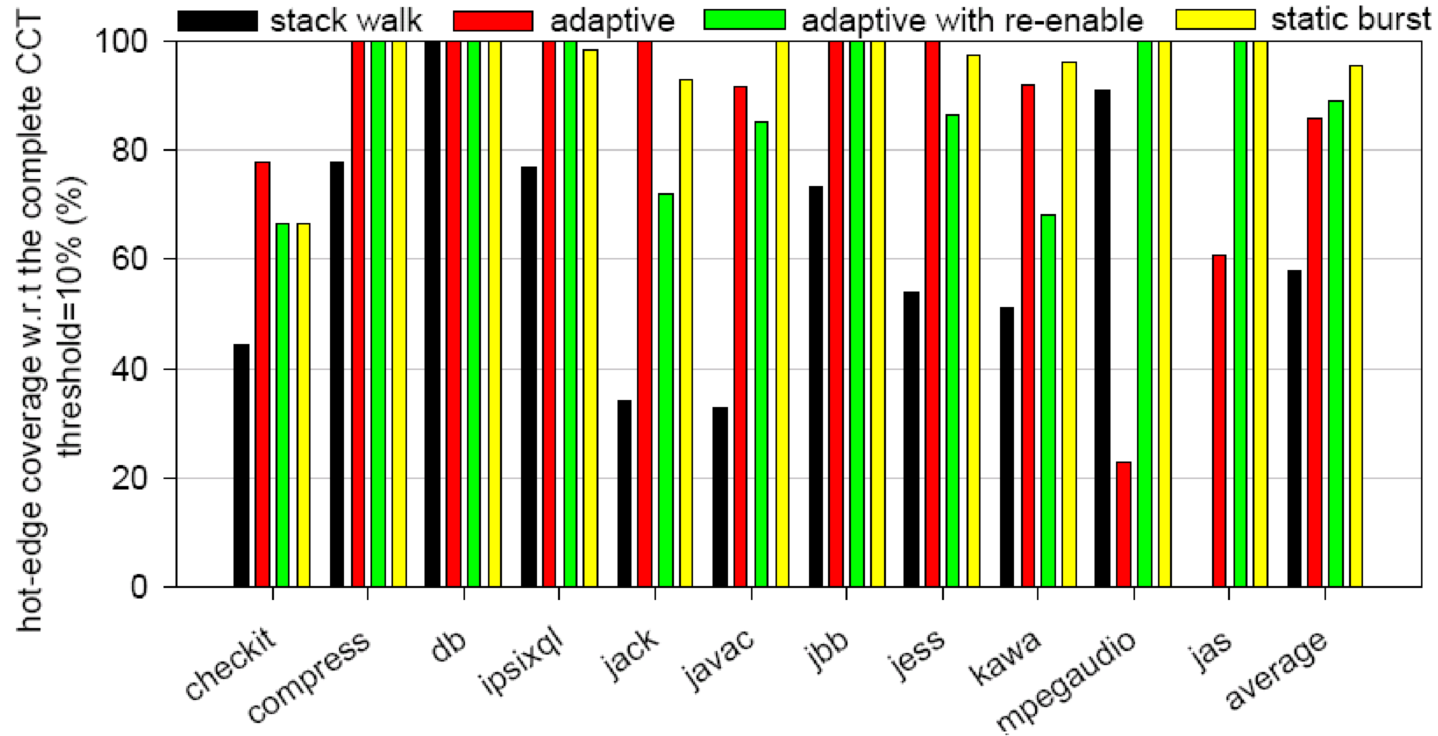
Results—Degree of Overlap



- **Average: stack walk (49.8%), adaptive (68.8%), adaptive w/ reenable (85.2%), static burst (91.4%).**

© 2006 IBM Corporation

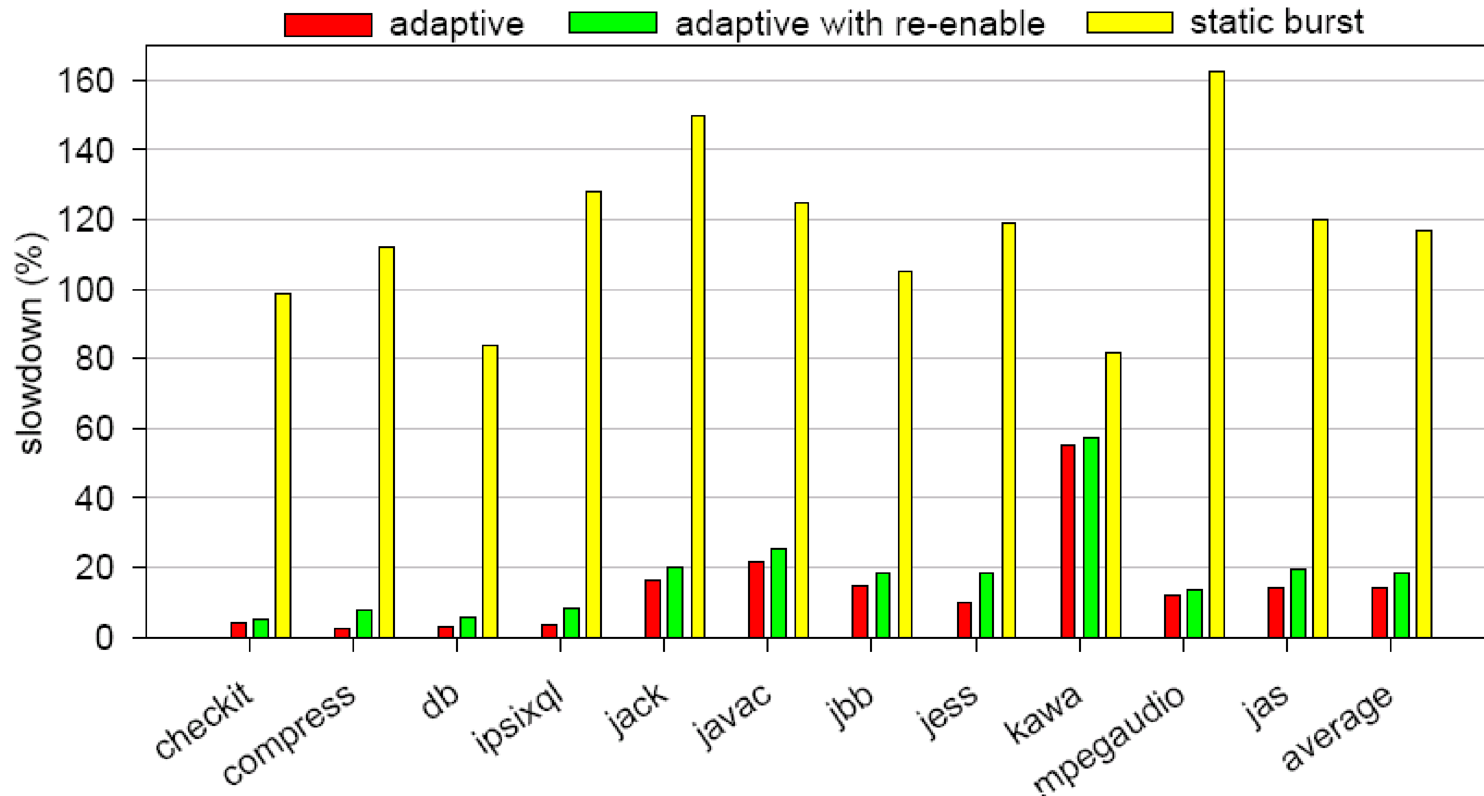
Results—Hot-edge Coverage



- **Average: stack walk (52.9%), adaptive (79.1%), adaptive w/ reenable (88.2%), static burst (88.1%).**

© 2006 IBM Corporation

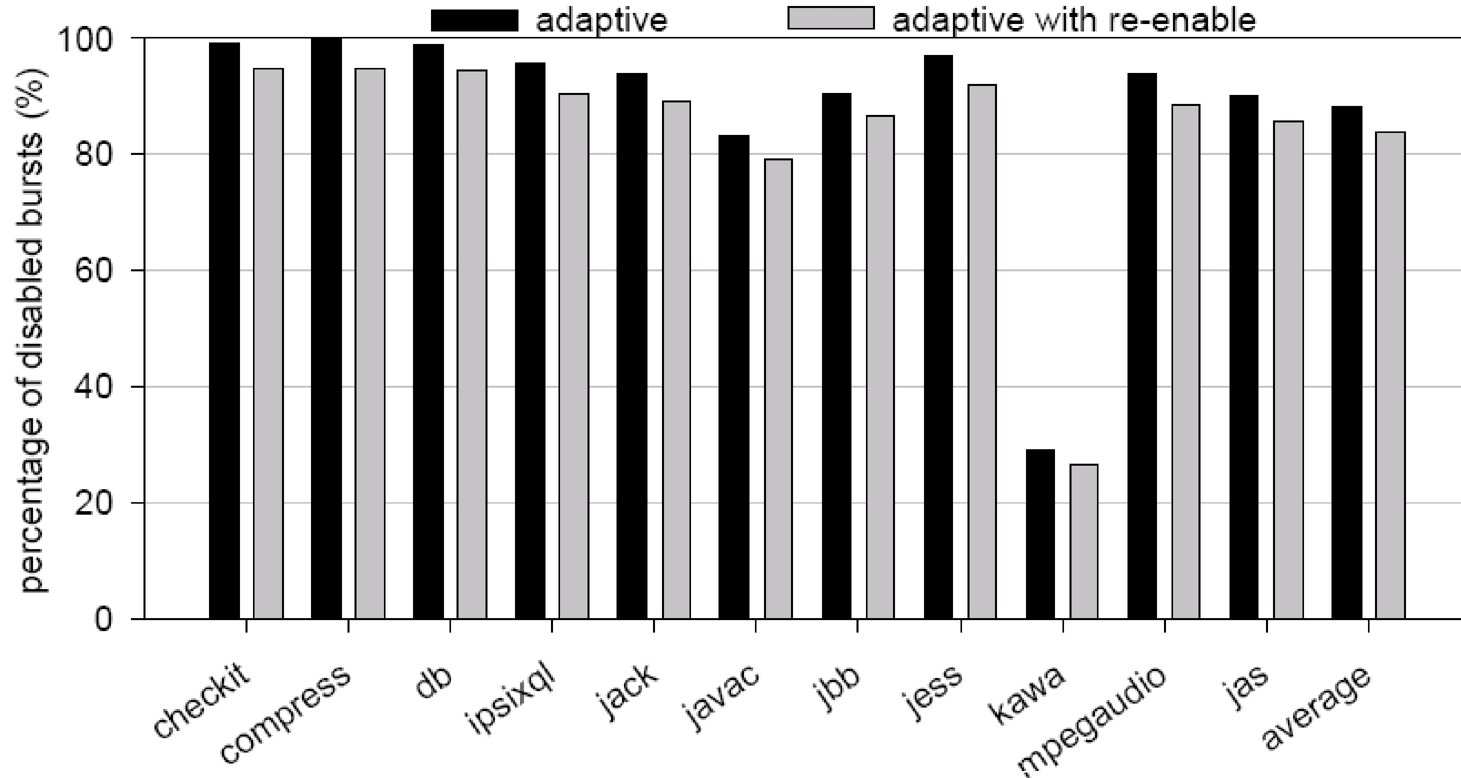
Results—Slowdown



- **Average: stack walk (<1%), adaptive (14.8%), adaptive w/ reenable (18.8%), static burst (117%)**
- **JVMPI is inefficient**

© 2006 IBM Corporation

Results—Percentage of Disabled Bursts



- **Both approaches disabled most bursts**
- **Reenablement only adds small % of bursts (*RR* = 5%)**

© 2006 IBM Corporation

Summary of Results

- **JVMPI-based adaptive bursting**
  - **A modest slowdown**
  - **85% degree of overlap**
  - **88% hot-edge coverage**

- **Sampled stack walking**
  - **Negligible slowdown**
  - **Around 50% degree of overlap and hot-edge coverage**
  - **Bad for large server benchmark JAS (*0% coverage*)**

- **Static bursting**
  - **Accuracy is close to adaptive bursting (<6%)**
  - **Slowdown 6 times higher**

© 2006 IBM Corporation

Outline

- Introduction
- Existing Approaches
- Our Approach: Adaptive Bursting
- Results
- Related Work
- Conclusion and Future Work

© 2006 IBM Corporation

Related Work

- **Exhaustive approach: Ammons et. el. [PLDI-97], Spivey [SPE-04],**

- **Sampling-based approach: Arnold & Sweeney [IBM TR-00], Froyd et. el. [ICS-05], Whaley [Java Grande-00]**

- **Context Sensitive Inlining: Hazelwood & Grove [CGO-03]**

© 2006 IBM Corporation

Outline

- Introduction
- Existing Approaches
- Our Approach: Adaptive Bursting
- Results
- Related Work
- Conclusion, Future Work, Criticism/Discussion

© 2006 IBM Corporation

Conclusion

- **Novel, efficient construction of accurate CCT**
  - **Accuracy: 80% to 90%.**
  - **Moderate overhead with JVMPI**
  - ***~6% overhead observed with JVM-based instrumentation.***
- **Formal definitions of two metrics for evaluating CCT accuracy**
  - **Degree of overlap**
  - **Hot-edge coverage**
- **Extensive measurements using a large number of benchmark programs, including a very large commercial J2EE Java application**

© 2006 IBM Corporation

Future Work

- **Further reduce the overhead**
  - **Better instrumentation (alternatives to JVMPI)**
  - **M. Bond suggested using PCC to identify history**
- **Call site information**
- **Applications: context sensitive optimizations**
  - **Lock contention analysis?**
  - **Object allocation analysis**
  - **Method inlining**

© 2006 IBM Corporation

Criticism/Discussion

- **Cold path coverage?**
  - **Insufficient cold path coverage**
  - **Rare bugs can't be discovered**

- **Overlap vs. Hot Edge Coverage: which is better?**

© 2006 IBM Corporation

Discussion

© 2006 IBM Corporation