Continuous Path & Edge Profiling

Michael Bond, UT Austin
Kathryn McKinley, UT Austin

Why profile?

- Inform optimizations
  - Target hot code
  - Inlining and unrolling
  - Code scheduling and register allocation
- Increasingly important for speculative optimization
  - Hardware trends → simplicity & multiple contexts
  - Less speculation in hardware, more in software

Speculative optimization

Less speculative — *More speculative*

- **Superblocks** [Hwu et al. '93]
- **Hyperblocks** [Mahlke et al. '92]
- **Dynamo fragments** [Bala et al. '00]
- **rePLay frames** [Patel & Lumetta '99]
- **MSSP tasks** [Zilles & Sohi '02]

Speculative optimization

Less speculative — *More speculative*

- **Superblocks** [Hwu et al. '93]
- **Hyperblocks** [Mahlke et al. '92]
- **Dynamo fragments** [Bala et al. '00]
- **rePLay frames** [Patel & Lumetta '99]
- **MSSP tasks** [Zilles & Sohi '02]

Profiling requirements

- Predict future with representative profile
  - Accurate
  - Continuous

Profiling requirements

- Predict future with representative profile
  - Accurate
  - Continuous
- Other requirements
  - Low overhead
  - Portable
  - Path profiling
- Previous work struggles to meet all goals

PEP: continuous path and edge profiling

- Predict future with representative profile
  - Accurate: **94-96%**
  - Continuous: **yes**
- Other requirements
  - Low overhead: **1.2%**
  - Portable: **yes**
  - Path profiling: **yes**

PEP: continuous path and edge profiling

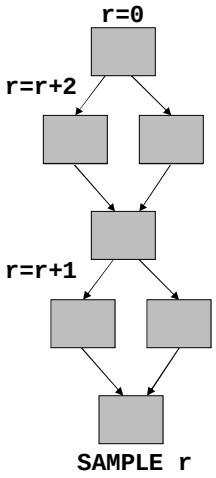
- Combines **all-the-time instrumentation** & **sampling**

New!

PEP: continuous path and edge profiling

- Combines **all-the-time instrumentation** & **sampling**
  - Instrumentation computes path number
  - Sampling updates profiles using path number

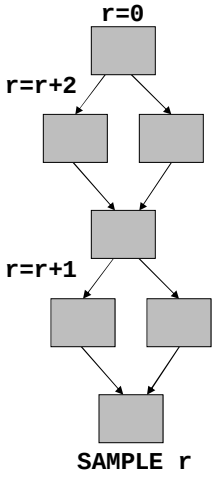
New!



PEP: continuous path and edge profiling

- Combines **all-the-time instrumentation** & **sampling**
  - Instrumentation computes path number
  - Sampling updates profiles using path number
  - Overhead: 30% → 2%

New!

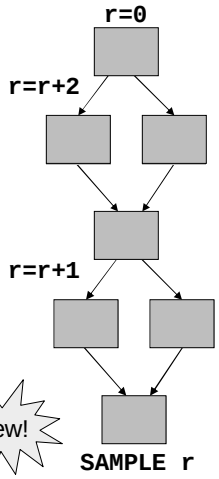


PEP: continuous path and edge profiling

- Combines **all-the-time instrumentation** & **sampling**
  - Instrumentation computes path number
  - Sampling updates profiles using path number
  - Overhead: 30% → 2%
- Path profiling as efficient means of edge profiling

New!

New!



Outline

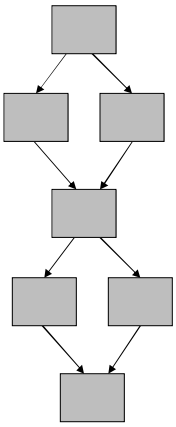
- Introduction
- Background: Ball-Larus path profiling
- PEP
- Implementation & methodology
- Overhead & accuracy

Ball-Larus path profiling

- Acyclic, intraprocedural paths
- Instrumentation maintains execution frequency of each path
  - Each path computes unique integer in **[0, N-1]**

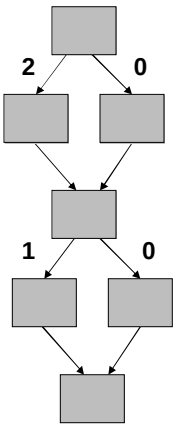
Ball-Larus path profiling

- 4 paths → **[0, 3]**



Ball-Larus path profiling

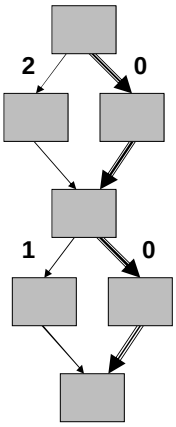
- 4 paths → **[0, 3]**
- Each path sums to unique integer



Ball-Larus path profiling

- 4 paths → **[0, 3]**
- Each path sums to unique integer

Path 0

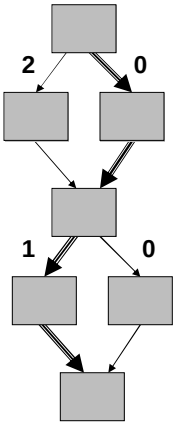


Ball-Larus path profiling

- 4 paths → **[0, 3]**
- Each path sums to unique integer

Path 0

Path 1



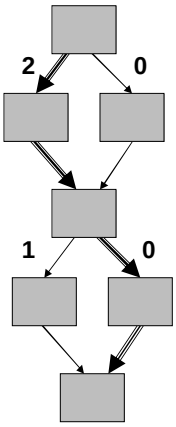
Ball-Larus path profiling

- 4 paths → **[0, 3]**
- Each path sums to unique integer

Path 0

Path 1

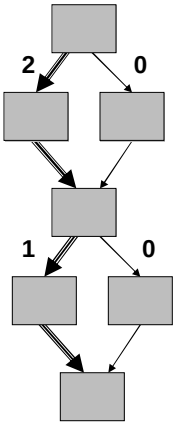
Path 2



Ball-Larus path profiling

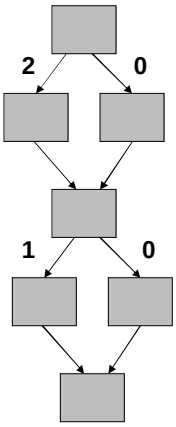
- 4 paths → **[0, 3]**
- Each path sums to unique integer

Path 0
Path 1
Path 2
Path 3



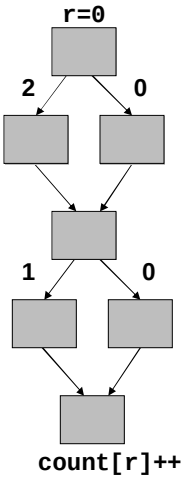
Ball-Larus path profiling

- **`r`**: path register
  - Computes path number
- **`count`**: frequency table
  - Stores path frequencies



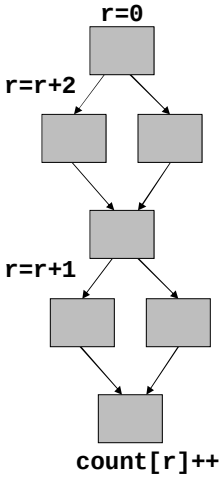
Ball-Larus path profiling

- **`r`**: path register
  - Computes path number
- **`count`**: frequency table
  - Stores path frequencies



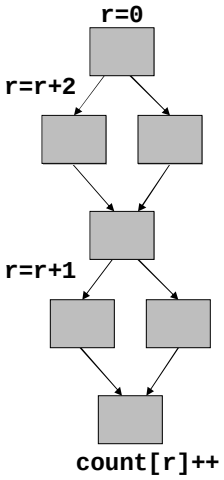
Ball-Larus path profiling

- **`r`**: path register
  - Computes path number
- **`count`**: frequency table
  - Stores path frequencies



Ball-Larus path profiling

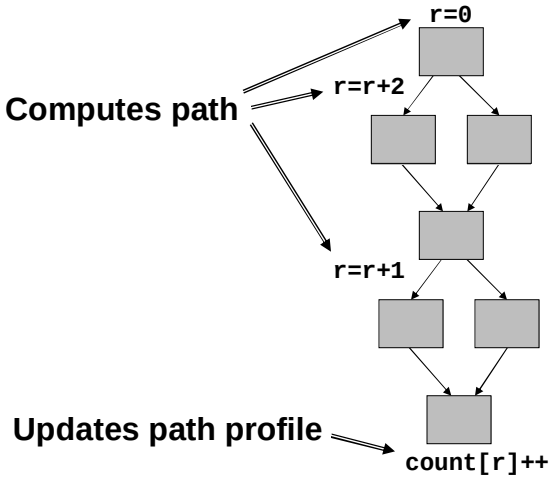
- **`r`**: path register
  - Computes path number
- **`count`**: frequency table
  - Stores path frequencies
  - Array by default
  - Too many paths?
    - Hash table



Outline

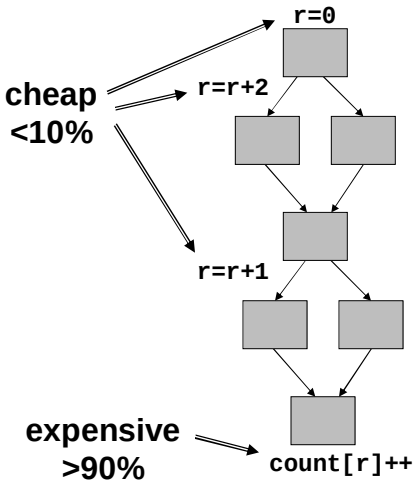
- Introduction
- Background: Ball-Larus path profiling
- PEP
- Implementation & methodology
- Overhead & accuracy

Motivation for PEP

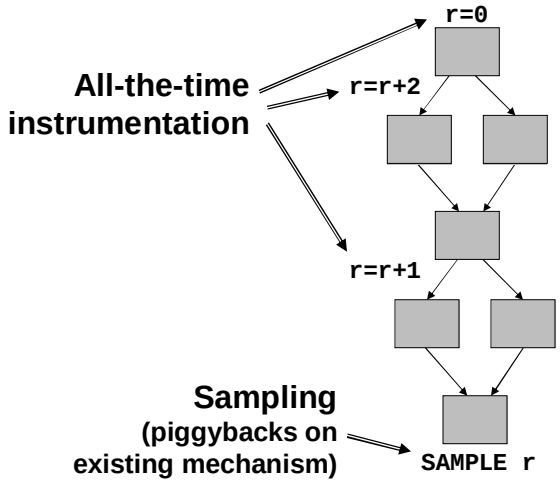


Motivation for PEP

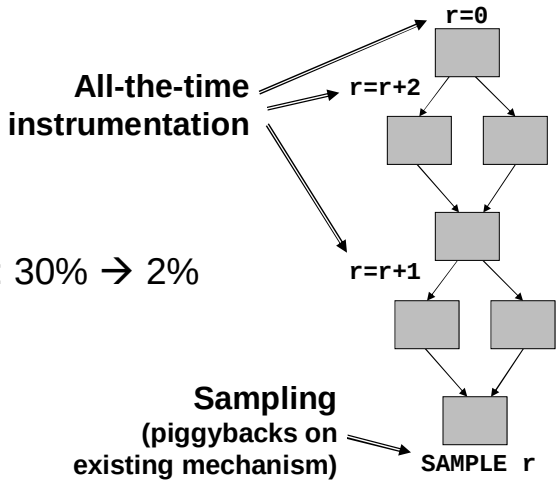
- Where have all the cycles gone?



What PEP does



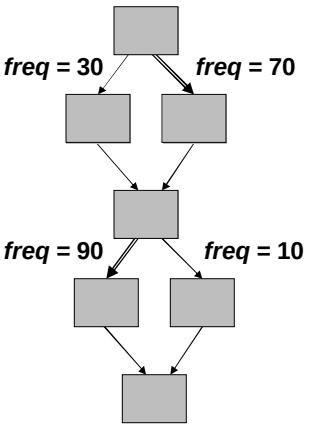
What PEP does



Overhead: 30% → 2%

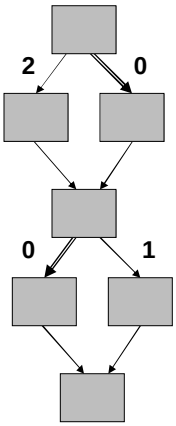
Profile-guided profiling

- Existing edge profile informs path profiling [Joshi et al. ’04]



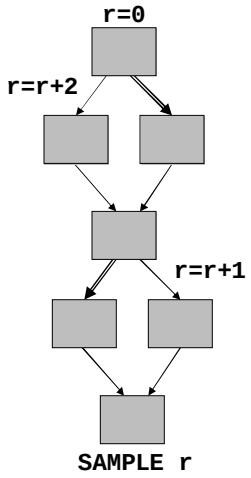
Profile-guided profiling

- Existing edge profile informs path profiling [Joshi et al. ’04]
- Assign zero to hotter edges



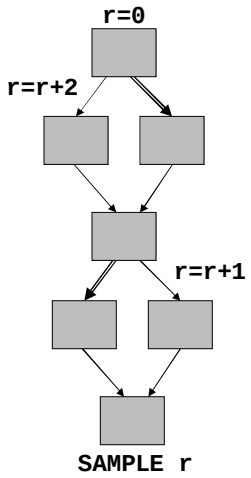
Profile-guided profiling

- Existing edge profile informs path profiling [Joshi et al. ’04]
- Assign zero to hotter edges
  - No instrumentation



Profile-guided profiling

- Existing edge profile informs path profiling [Joshi et al. ’04]
- Assign zero to hotter edges
  - No instrumentation
- From *practical path profiling* [Bond & McKinley ’05]



Outline

- Introduction
- Background: Ball-Larus path profiling
- PEP
- Implementation & methodology
- Overhead & accuracy

Implementation

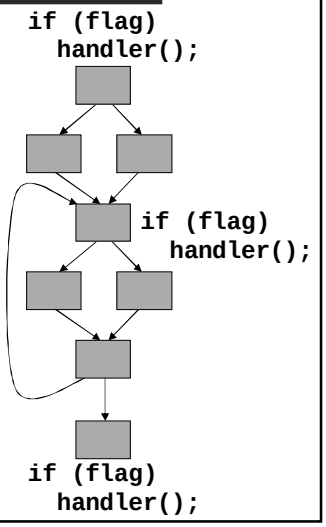
- Jikes RVM
  - High performance, Java-in-Java VM
  - Adaptive compilation triggered by sampling
- Two compilers
  - *Baseline* compiles at first invocation
    - Adds instrumentation-based edge profiling
  - *Optimizer* recompiles hot methods
    - Three optimization levels
- PEP implemented in optimizing compiler

PEP sampling

- Piggybacks on existing thread-switching mechanism

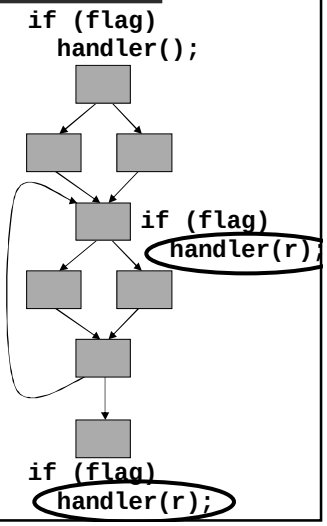
PEP sampling

- Piggybacks on existing thread-switching mechanism
- Jikes RVM inserts *yieldpoints* at loop headers and method entry & exit
  - Yieldpoint handlers switch threads and update profiles



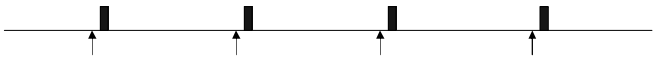
PEP sampling

- Piggybacks on existing thread-switching mechanism
- Jikes RVM inserts *yieldpoints* at loop headers and method entry & exit
  - Yieldpoint handlers switch threads and update profiles
- PEP samples **r** at headers and method exit
  - Updates path and edge profiles



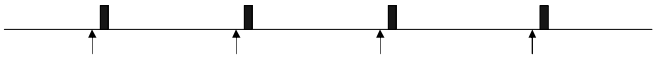
Sampling methodology

- Classic



Sampling methodology

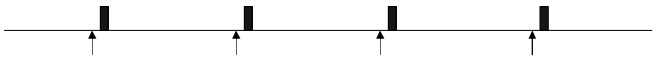
- Classic



- Arnold-Grove sampling: invaluable for PEP
  - Multiple samples per timer tick
  - Strides to reduce timer-based bias

Sampling methodology

- Classic



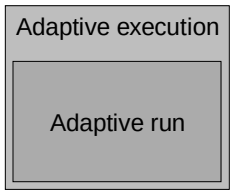
- Arnold-Grove sampling: invaluable for PEP
  - Multiple samples per timer tick
  - Strides to reduce timer-based bias

- PEP strides *before* first sample only, not between samples

  - Timer goes off every 20 ms
  - Stride 0 -16 samples, then 64 consecutive samples
  - PEP sampling rate: 3200 paths/second

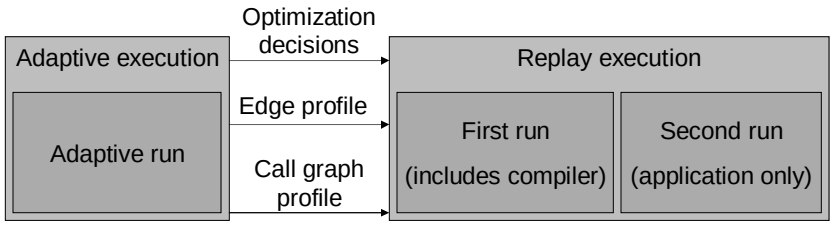
Execution methodology

- *Adaptive*: normal adaptive run
  - Different behavior from run to run



Execution methodology

- *Adaptive*: normal adaptive run
  - Different behavior from run to run
- *Replay*: deterministic compilation decisions
  - First run includes compilation
  - Second run is application only [Eeckhout et al. 2003]



Benchmarks and platform

- SPEC JVM98
- **`pseudojbb`**: SPEC JBB2000, fixed workload
- DaCapo Benchmarks
  - Exclude **`hsqldb`**

- 3.2 GHz Pentium 4 with Linux
  - 8K DL1, 12Kμop IL1, 512K L2, 1GB memory

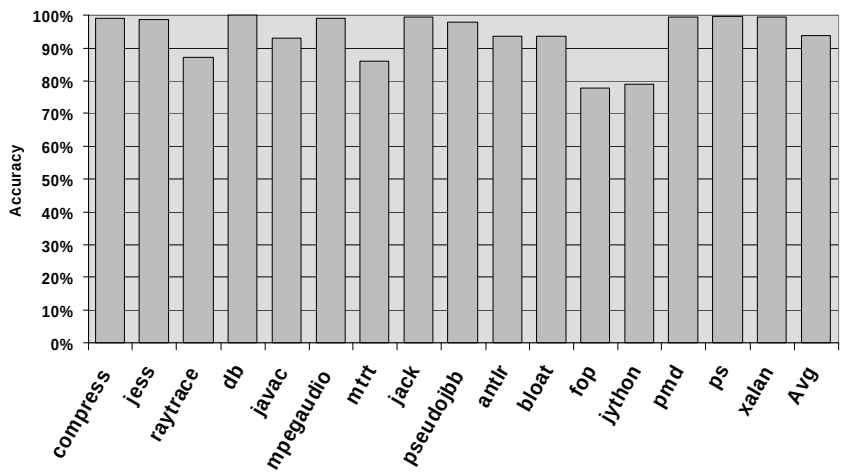
Outline

- Introduction
- Background: Ball-Larus path profiling
- PEP
- Implementation & methodology
- Accuracy & overhead

Path accuracy

- Compare PEP's path profile to perfect profile
- Wall weight-matching scheme [Wall '91]
  - Measures how well PEP predicts hot paths
- Branch-flow metric [Bond & McKinley '05]
  - Weights paths by their lengths

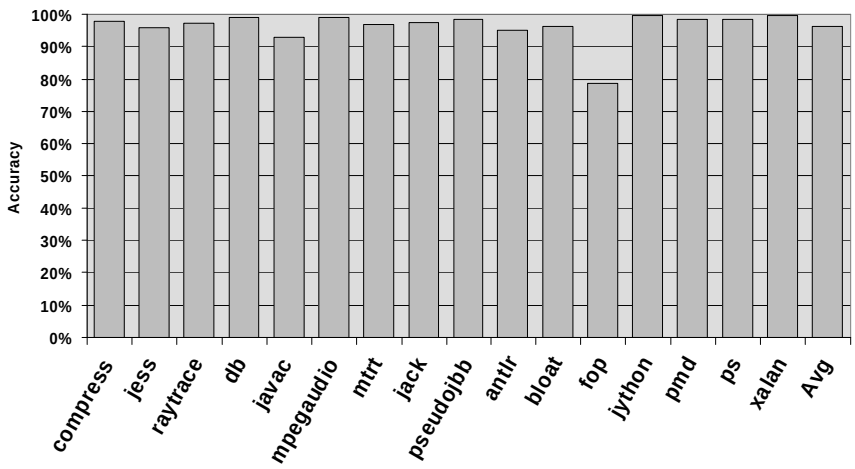
Path accuracy



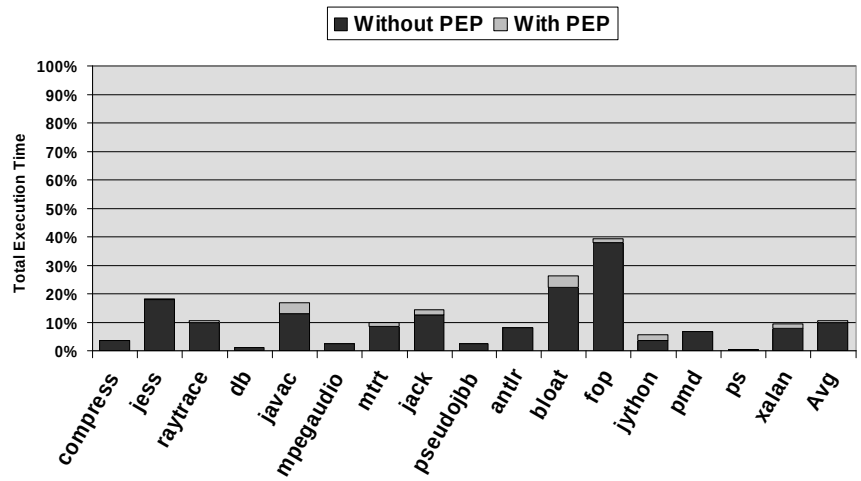
Edge accuracy

- Compare PEP's edge profile to perfect profile
- Relative overlap
  - Measures how well PEP predicts edge frequency relative to source basic block
  - Jikes RVM uses relative frequencies only

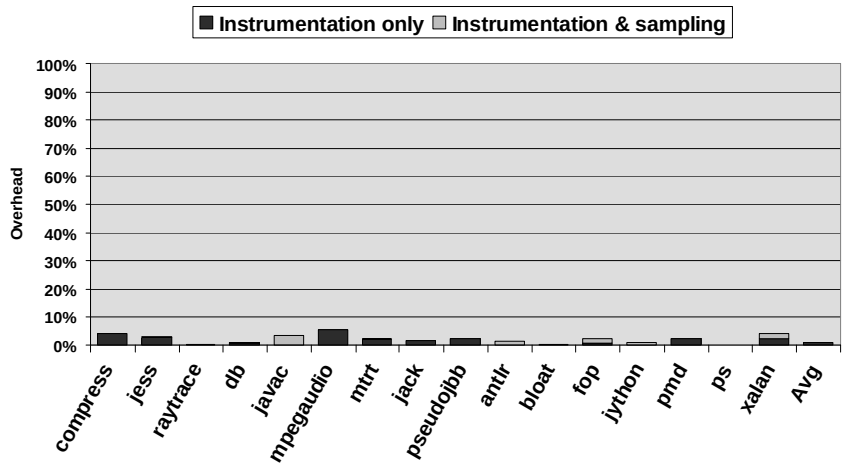
Edge accuracy



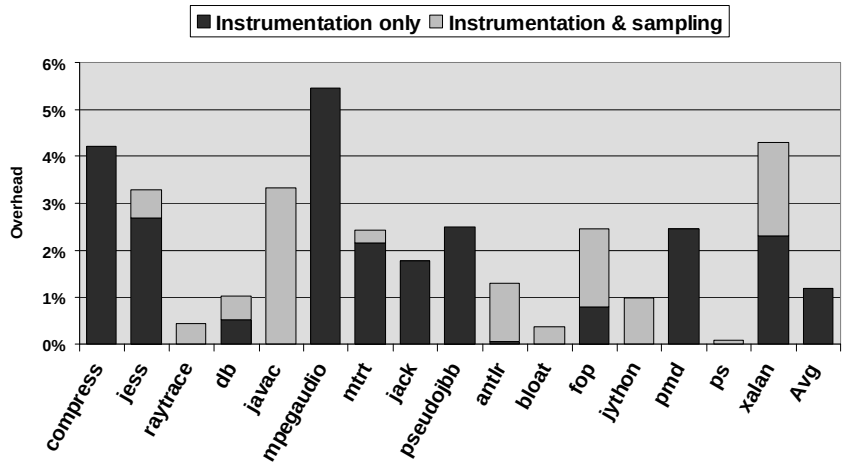
Compilation overhead



Execution overhead



Execution overhead



Related work

- Instrumentation [Ball & Larus '96]
  - High overhead
- One-time profiling [Jikes RVM baseline compiler]
  - Vulnerable to phased behavior
- Sampling
  - Code sampling [Anderson et al. '00]
    - No path profiling
  - Code switching [Arnold & Ryder '01, dynamic instrumentation]
- Hardware [Vaswani et al. '05, Shye et al. '05]

Summary

- Continuous & accurate profiling needed for aggressive, speculative dynamic optimization
- PEP: continuous, accurate, low overhead, portable, path profiling

Thank you!

- Questions?