# Vertical Profiling: Understanding the Behavior of Object-Oriented Applications

Matthias Hauswirth, Amer Diwan
University of Colorado at Boulder

Peter F. Sweeney, Michael Hind
IBM Thomas J. Watson Research Center

Presented by: Irantha Suwandarathna

# Outline

- What is Vertical Profiling
- Motivations
- Implementation
- Case studies
- Conclusions

# Motivations

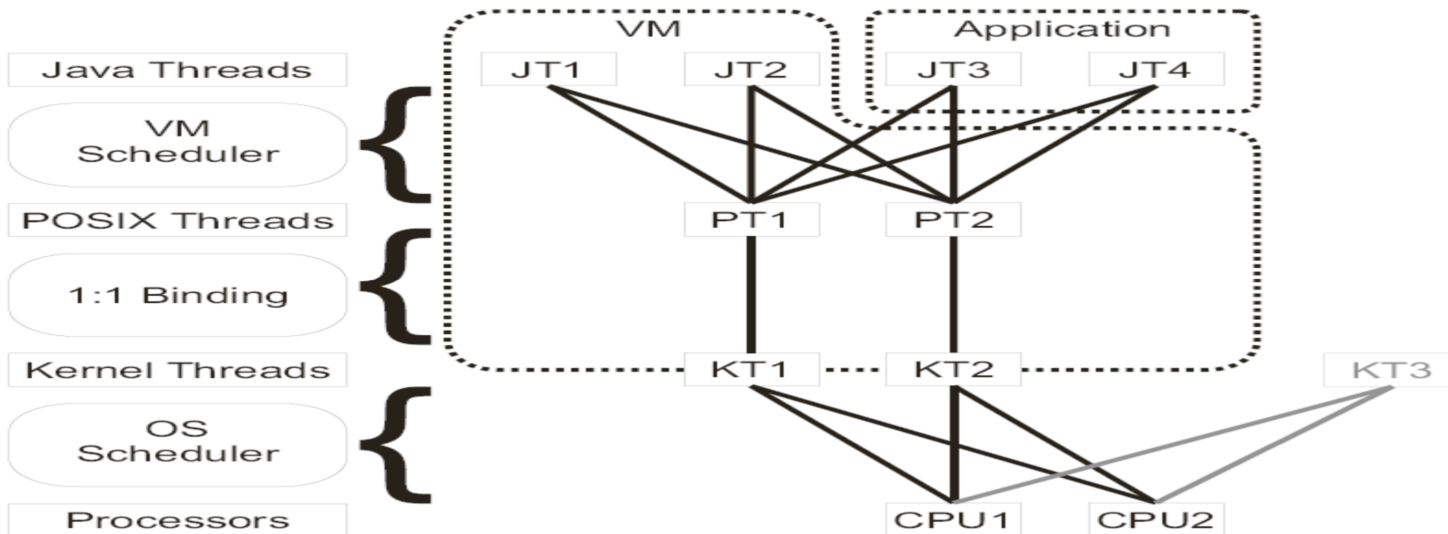
| **C** Program | **Java** |
| --- | --- |
| Application<br>Native Library<br>Operating System<br>Hardware | Application<br>Framework<br>Java Library<br>Virtual Machine<br>Native Library<br>Operating System<br>Hardware |

- Increased Virtualization
- Dynamic recompilation/garbage collection

# Implementation

- One trace file for JikesRVM Thread
  - Virtual Processor ID, Thread ID,
  - Real Time, Real Time duration ,
  - Compiled Method ID, Monitor Values
- Only 1 thread execute in measurement period
- Real time value to merge trace files & durations for non-VM Threads
- Meta file for ID → name mappings

# Where Monitor values are kept




- Native code Instrumentations
  - PThread specific storage
- Java level Instrumentations
  - Virtual Processor object

# Measurement Overhead

| Benchmark | Production | Vertical Profiling | |
|---|---|---|---|
| compress | 9.77 | 10.15 | 3.6% |
| db | 22.42 | 23.85 | 6.4% |
| jack | 13.38 | 14.41 | 7.8% |
| javac | 19.14 | 20.57 | 7.5% |
| jess | 8.23 | 8.64 | 5.0% |
| mpegaudio | 8.52 | 9.69 | 13.8% |
| mtrt | 7.64 | 7.99 | 4.6% |
| jbb | 27.17 | 31.84 | 17.2% |
| hsql | 19.19 | 19.39 | 1.1% |
| Average | | | 7.4% |

- With 148 Software performance monitors

# Perturbation Analysis

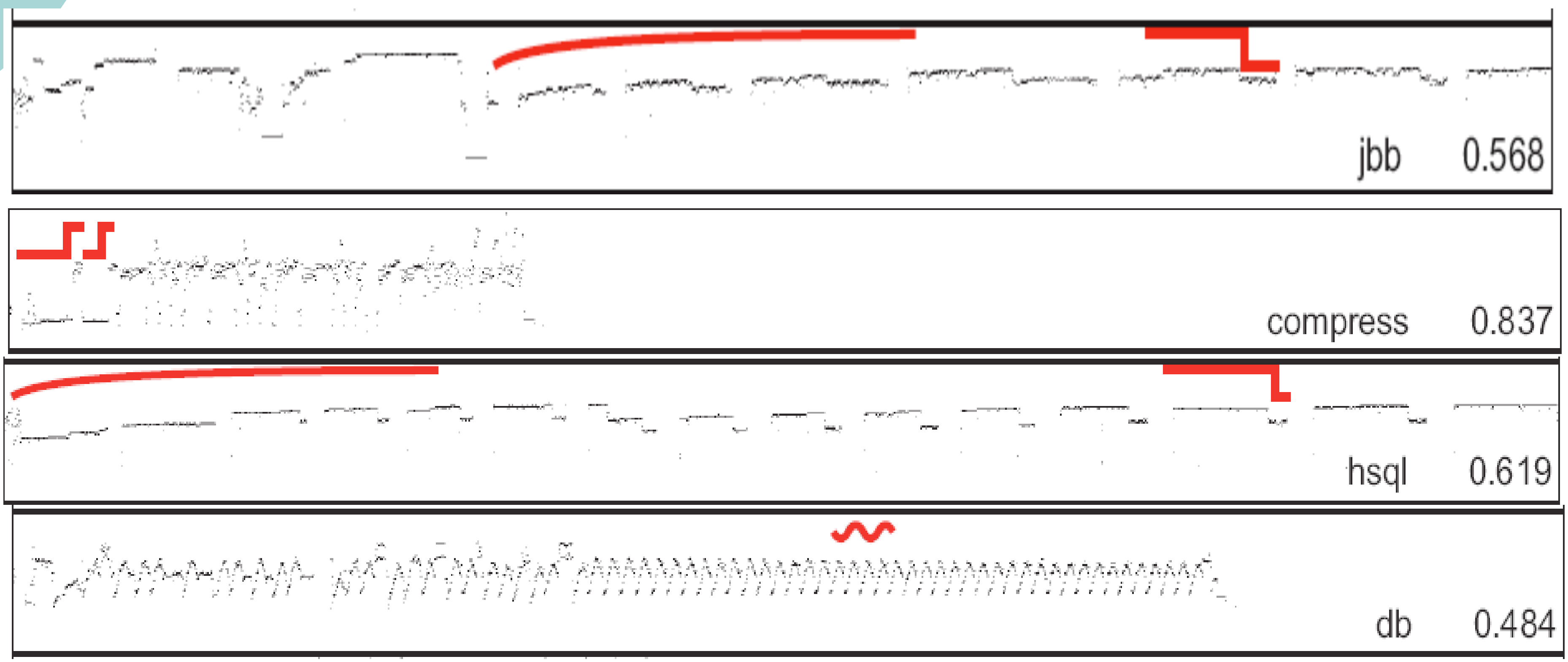
- End-to end perturbation for HPM
  - Runs with no collection during execution
  - 5 runs & taking the average
- Temporal impact of HPM
  - Qualitative analysis with their knowledge
- Impact of SPMs on HPMs
  - Runs with & without SPMs
- Impact of SPMs on SPMs
  - Qualitative analysis with their knowledge

# Validation

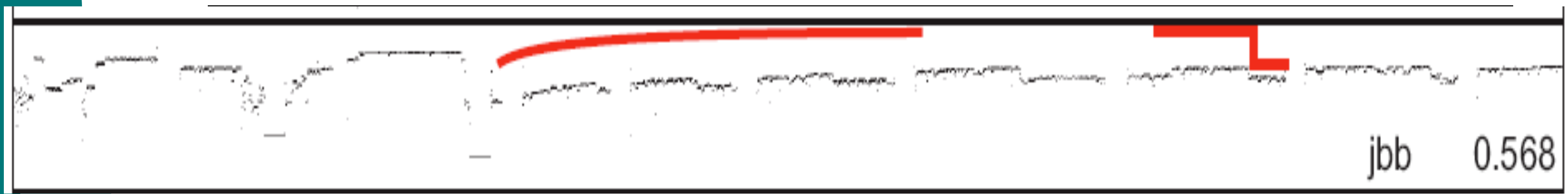
- Hypothesis about the cause
- Eliminate the cause
- See whether phenomenon is gone

# Case Studies

- Primary performance Metric
  - Instructions Per Cycle (IPC)

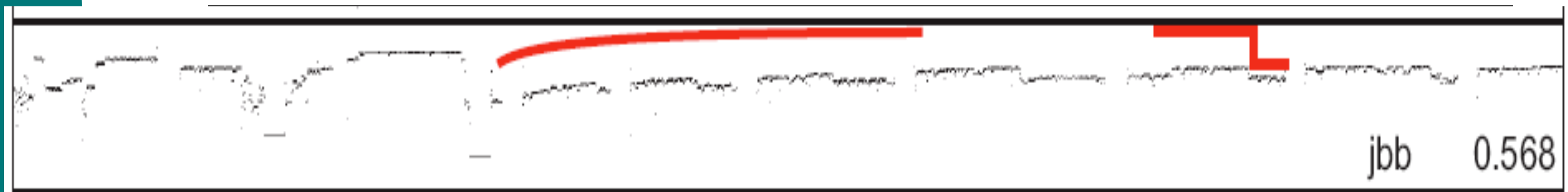


# Gradual Increase in ***jbb***



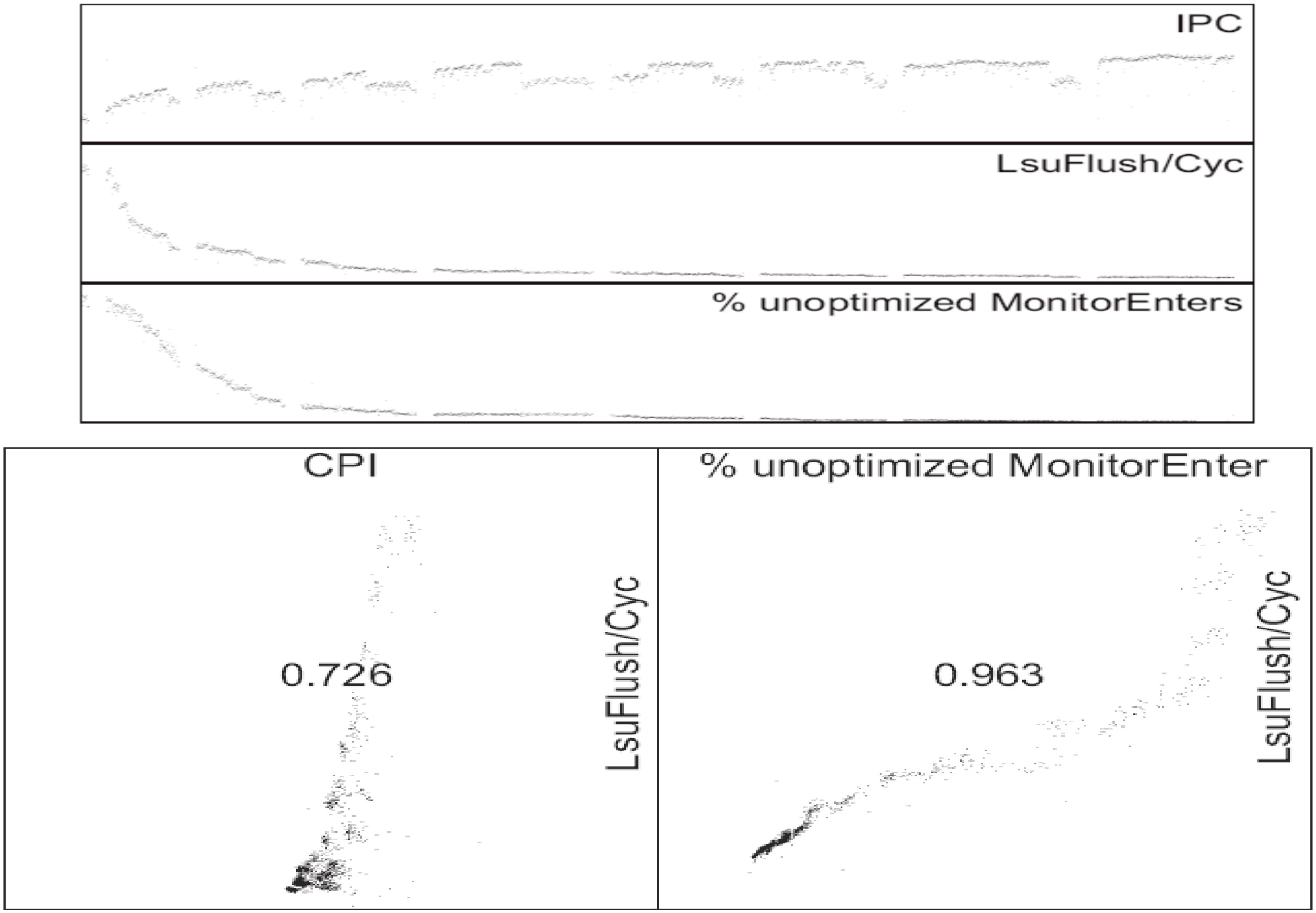

- 50 transactions per time slice
- From previous studies
  - Optimized code has 32% higher IPC
  - Increase IPC → low LSU flushes
    - 15.2% LSU flushes in optimized code
    - 0.1% LSU flushes in un-optimized code

# Gradual Increase in ***jbb*** …




- Measure time spent on optimized & un-optimized code
- Approximate- Number of synchronized methods executed
- Different synchronized method entry points for optimized & un-optimized code
- Validate with AOS disabled

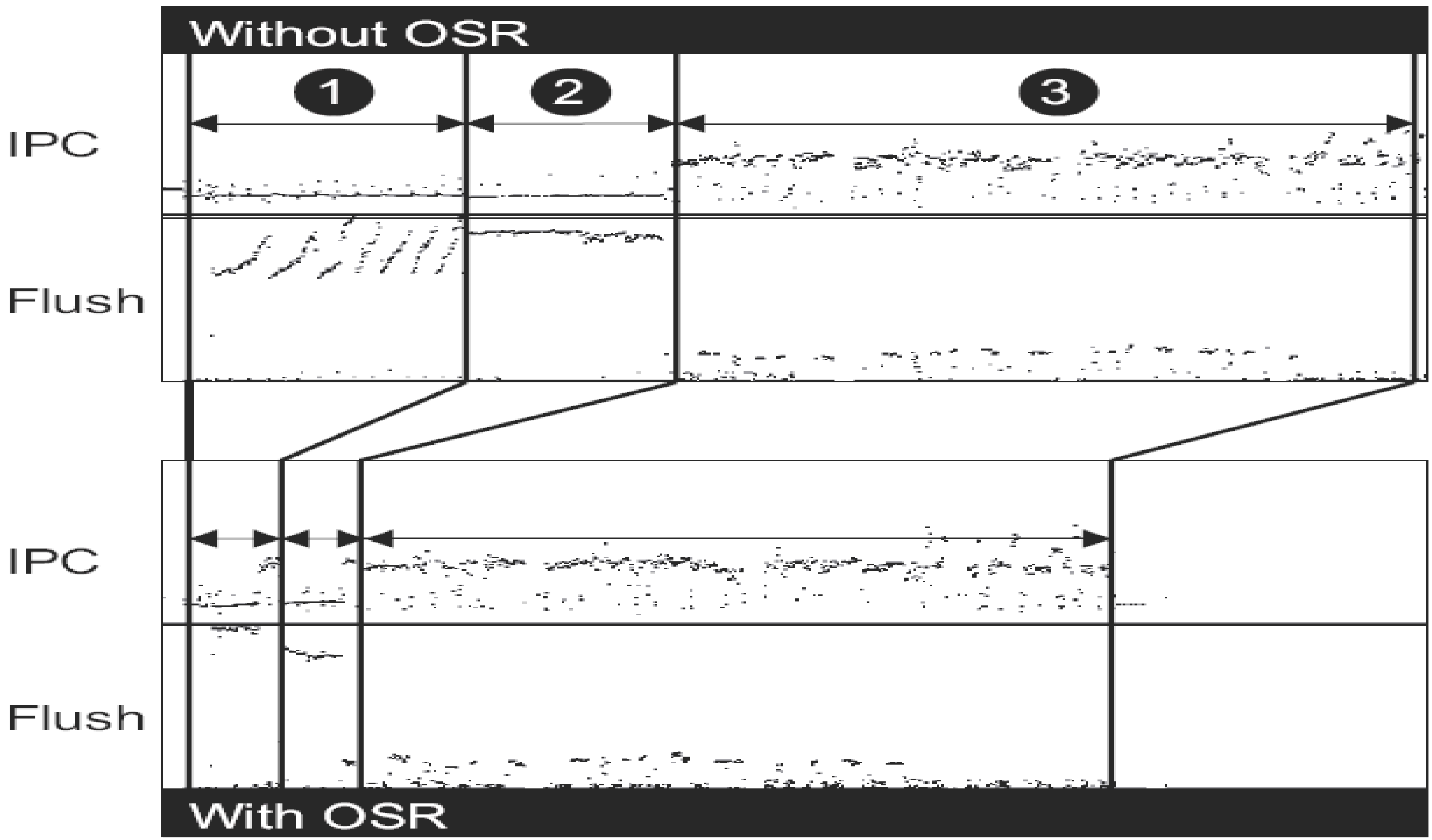
# Gradual Increase in ***jbb*** …



# Sudden Increase in ***compress***


- Two long running methods
  - Compress & decompress
- $1^{st}$ jump → compress optimized
- $2^{nd}$ jump → decompress optimized
- Instrumentation
  - Top of the stack method ID
  - Most recently optimized method

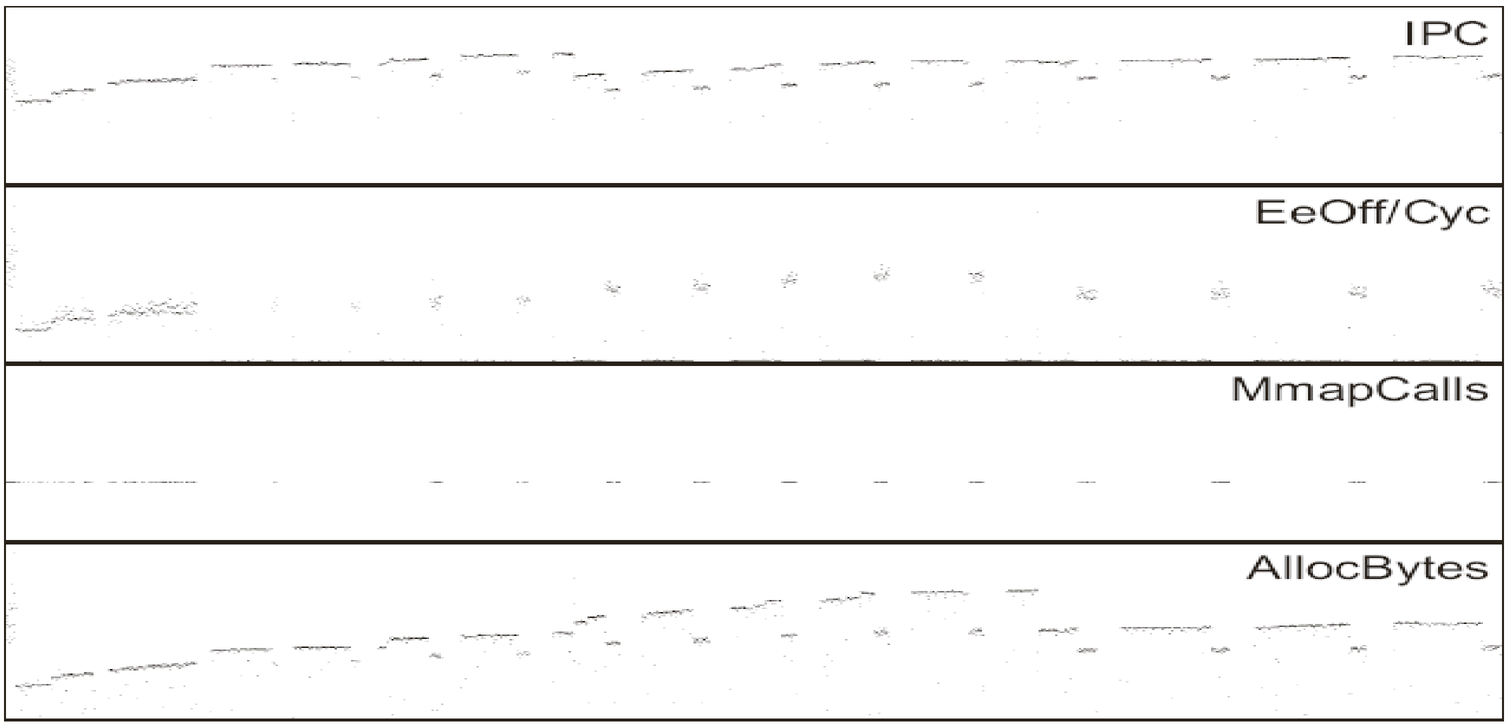
# Sudden Increase in ***compress …***



# Dip before GC in ***HSQL***

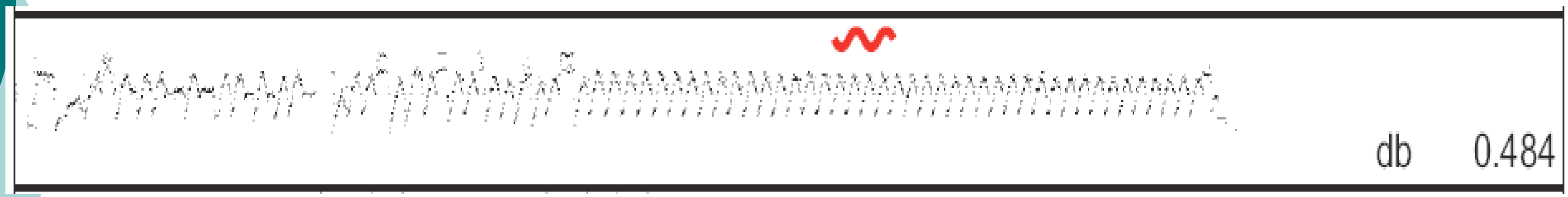

- Adaptive heap resizing
- App. runs out of memory→ Trigger GC
- page fault exceptions → low IPC
- Instrumentation
  - Number of virtual page requests
  - CPU cycles with exceptions disabled
  - Number of bytes allocated in JAVA

# Dip before GC in ***HSQL ...***



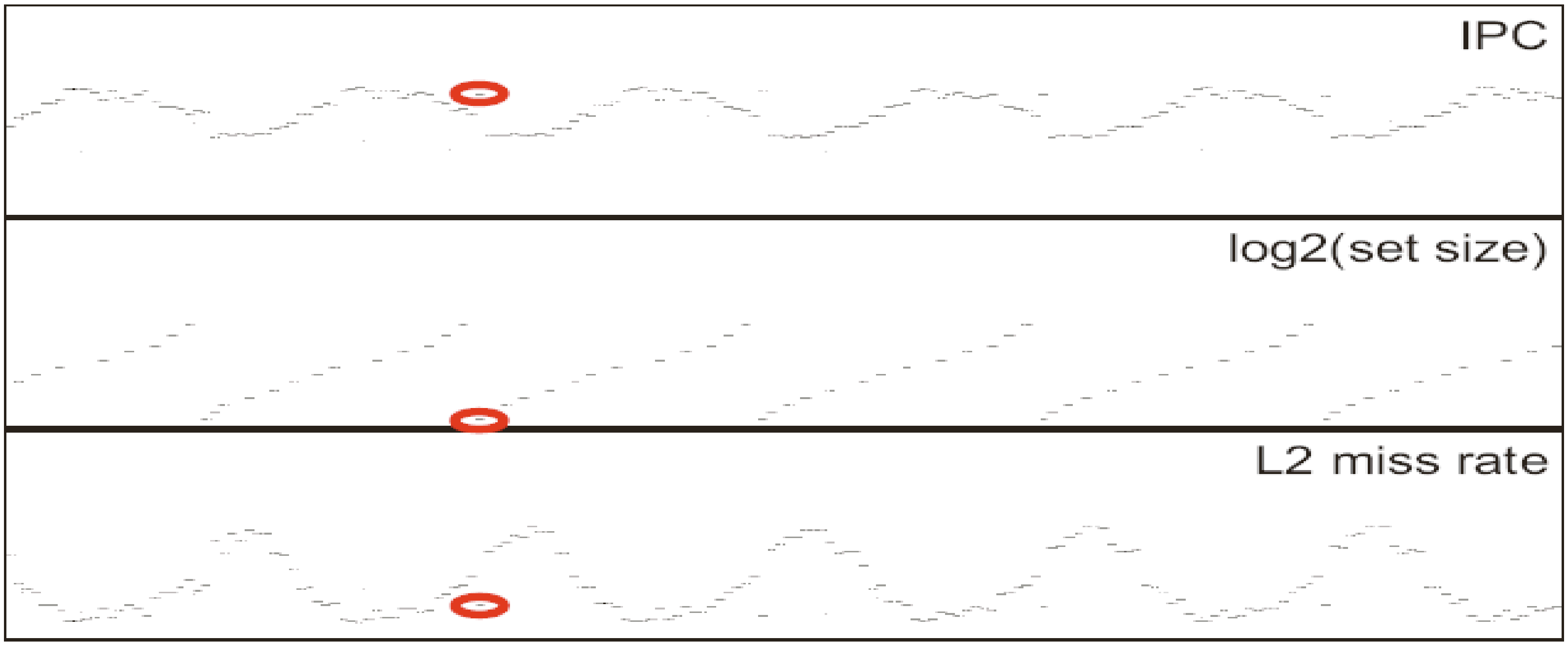

- Cross correlation
  - MmapBytes & AllocBytes → 0.9995
- Validation – disable adaptive heap resizing

# Periodic Pattern in ***Db***




- Each Pattern corresponds to a Shell sort run
- L2 cache not enough to keep the working set → drop in IPC
- -0.916 correlation between IPC & L2 cache miss rate
- Measure set size

# Periodic Pattern in ***Db*** ...




- Validation
  - Object in lining
  - IPC drop start at larger set size

# Why lot of small time slices in Multi-Threaded benchmarks?

- For ***jbb***
  - 1 worker thread – 2221 time slices
  - More than 1 – 10,441 time slices
- Small time slices
- High lock contention
  - Thread yield → end time slice prematurely

# Why lot of small time slices in Multi-Threaded benchmarks?

| Benchmark | Scale | Wall time (M,%) | | CPU time (M,%) | | Samples | Sample D. (M) | Lock Yields |
|---|---|---|---|---|---|---|---|---|
| mtrt | 1 on 1 | 1,070 | 100% | 828 | 100% | 575 | 1.440 | 0 |
| mtrt | 2 on 2 | 754 | 70% | 957 | 116% | 694 | 1.379 | 40 |
| mtrt | 4 on 4 | 666 | 62% | 1,224 | 148% | 1,129 | 1.084 | 302 |

- Decrease in wall clock time
- Increase in CPU time
- Samples – Lock Yields = constant

# Where Lock contention Happens

| Benchmark | Lock Yields | Library | VM | App |
|---|---|---|---|---|
| mtrt | 0 | 0 | 0 | 0 |
| mtrt | 40 | 40 | 0 | 0 |
| mtrt | 350 | 134 | 216 | 0 |
| jbb | 0 | 1 | 0 | 0 |
| jbb | 2,704 | 0 | 2,703 | 1 |
| jbb | 8,013 | 113 | 7,843 | 57 |
| hsql | 0 | 0 | 0 | 0 |
| hsql | 28,600 | 2 | 0 | 28,598 |
| hsql | 72,012 | 143 | 149 | 71,720 |

# Instrumentation & Layers

| Case Study / Layer | Gradual Increase in *jbb* | Sudden Increase in *compress* | Scalability in *mtrt*, *jbb*, *hsql* | Dip Before GC in *hsql* | Periodic Pattern in *db* |
|---|---|---|---|---|---|
| Application | | | | | SetSize |
| Framework | | | | | |
| Java Libraries | | | | | |
| Virtual Machine | OptMonitorEnter<br>UnoptMonitorEnter | TopOfStackMethodId<br>OptimizedMethodId | LockYieldCount<br>LockYieldTypeId | AllocBytes | |
| Native Libraries | | | | | |
| Operating System | | | | EeOff<br>MmapCalls<br>MmapBytes | |
| Hardware | Cyc<br>InstCmpl<br>LsuFlush | Cyc<br>InstCmpl | Cyc | Cyc<br>InstCmpl | Cyc<br>InstCmpl<br>L2Misses |

- Approach
  - Browsing
  - Searching

# Problems with statistical correlation

- Low even frequency
- No linear relationships
- Leverage points
- Direction of causality

# Difficulties of this approach

- Knowledge on all layers
  - H/W ,OS, VM ,Libraries , Application
- Required metrics not known
- Perturbation
- Not automated
  - Thousands of metrics to manually inspect

# Conclusions

- Vertical profiling can be used to understand performance phenomena in modern multi layers systems.