Understanding Measurement Perturbation in Trace-Based Data

Peter F. Sweeney IBM Research, Hawthorne

Joint work with

Todd Mytkowicz U. of Colorado at Boulder

Amer Diwan U. of Colorado at Boulder

Matthias Hauswirth U. of Lugano, Switzerland

Motivation

- Instrument code to understand system behavior
  - Profile basic blocks, methods
  - Trace hardware & software metrics
- Instrumentation can perturb the system's behavior
- How does perturbation impact the ability to reason about system behavior?

Background

- NSF Grant "Understanding the Performance of Modern Systems"
  - Amer Diwan (U of Colorado Boulder)
  - Mike Mozer (U of Colorado Boulder)
  - Peter Sweeney (IBM Research)
- Vertical profiling
  - Trace-based data
  - Reason across software and hardware components
- General belief
  - Low overhead => low perturbation

    Overhead is instruction or cycle perturbation!

Methodology

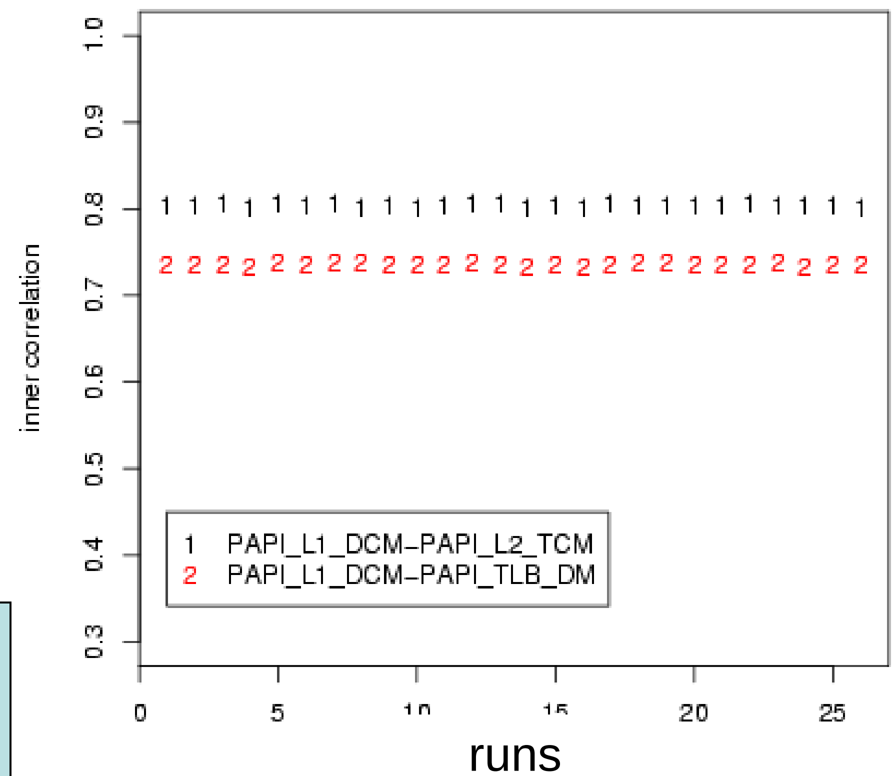
- Reason about metrics
  - Statistical correlation computes trend between two metrics
    - e.g. L1 and L2 misses
  - Compare correlation score before and after instrumentation

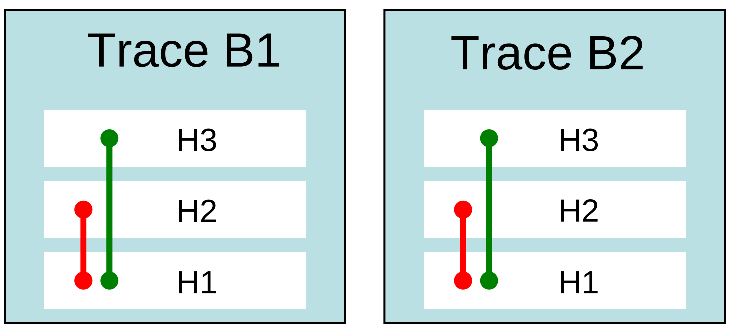
Infrastructure

- Extended CIL to instrument C programs
- Two types of instrumentation
  - Low level: hardware metrics
    - E.g. Cache misses, instructions executed, cycles
  - High level: software metrics
    - E.g. method calls, update global variable
- Periodically collect metric values
  - Use *settimer*
  - 10 to 100's millisecond intervals
  - reads counters and writes their values to disk

Reality Check

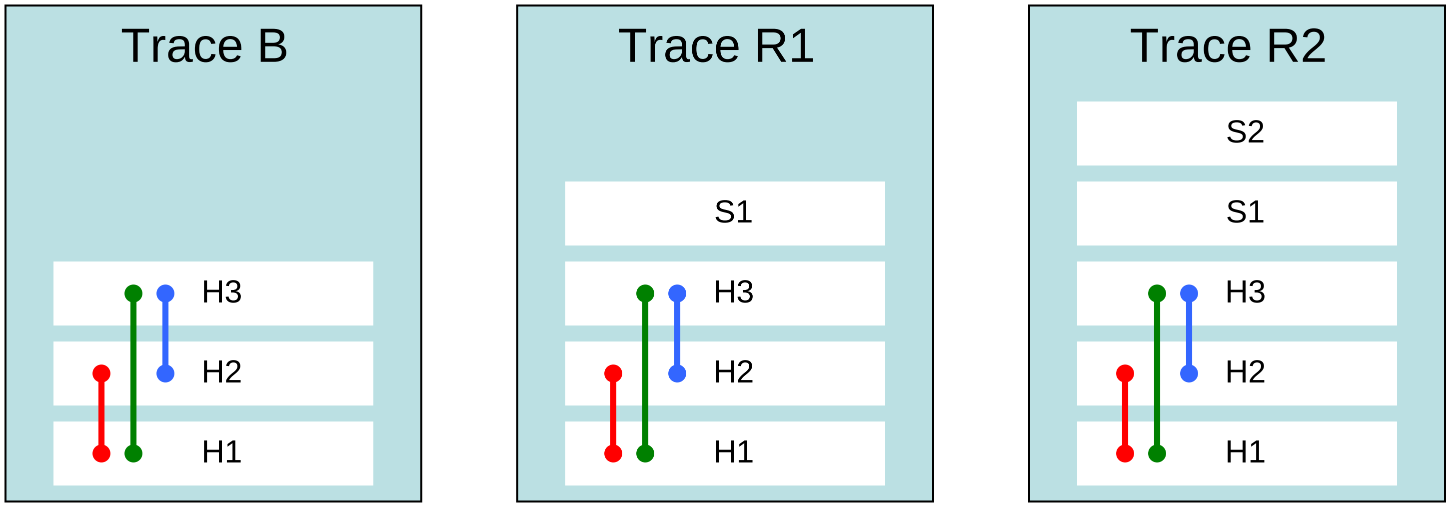
- sjeng (SPEC CPU2006)
- Multiple runs collecting same metrics
- Graph correlation of pairs of metrics within a trace
- Minimal perturbation across runs





…

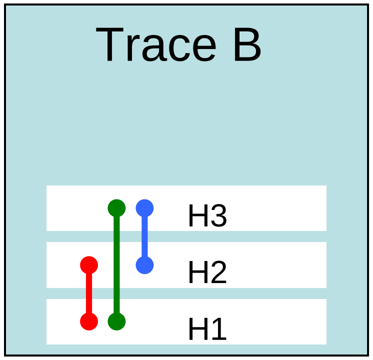
Inner Perturbation

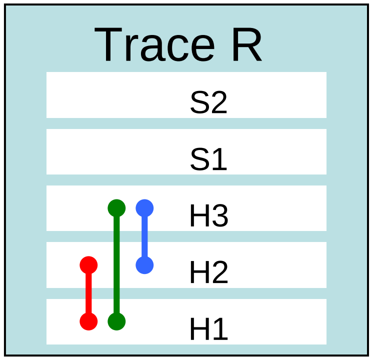


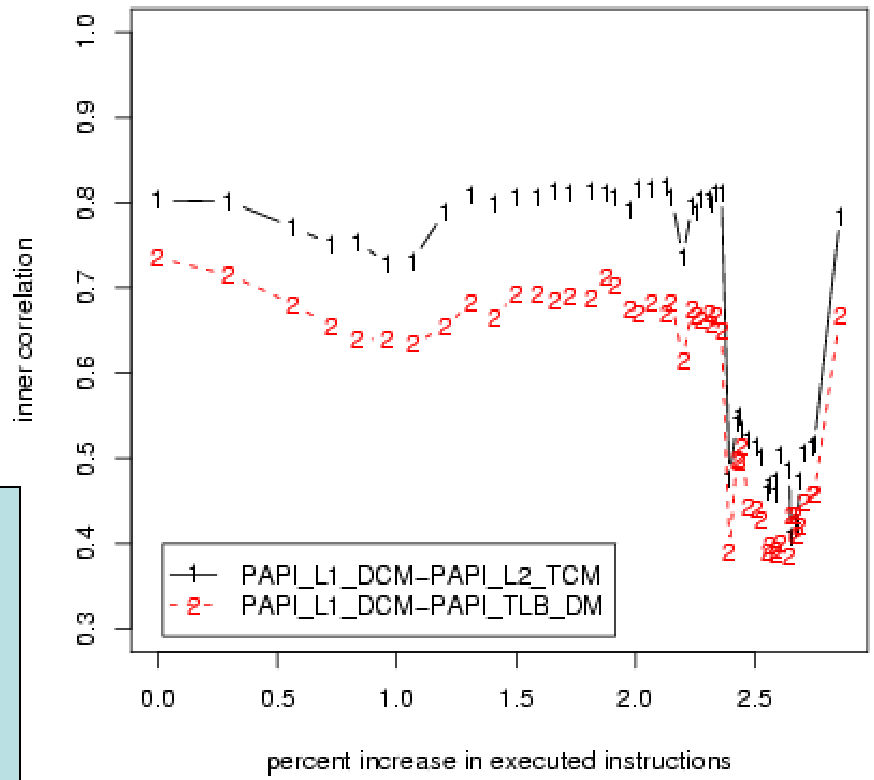
- Correlate pairs of metrics within a trace
- Compare *inner* correlation scores across traces
  - E.g. compare corr(B.H1, B.H2) with corr(R.H1, R.H2)
- Observe how correlation changes as additional instrumentation is added

Significant Inner Perturbation

- sjeng (SPEC CPU2006)
- Multiple runs collecting different software metrics
- Graph inner correlation scores of hardware metrics as instrumentation is added
- Same metrics as “Reality Check”
- Significant inner perturbation
- Small increase in instructions executed < 3%
- But recovers

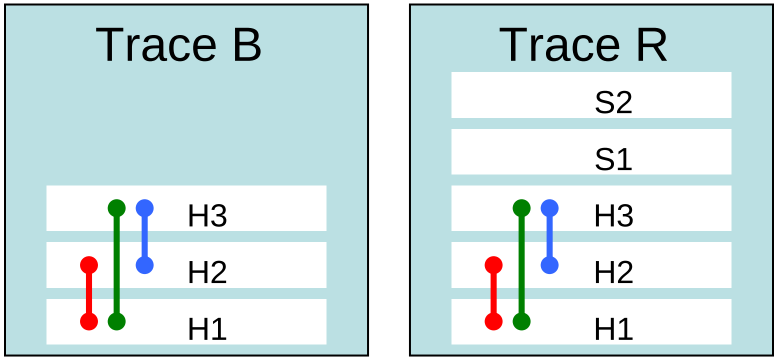


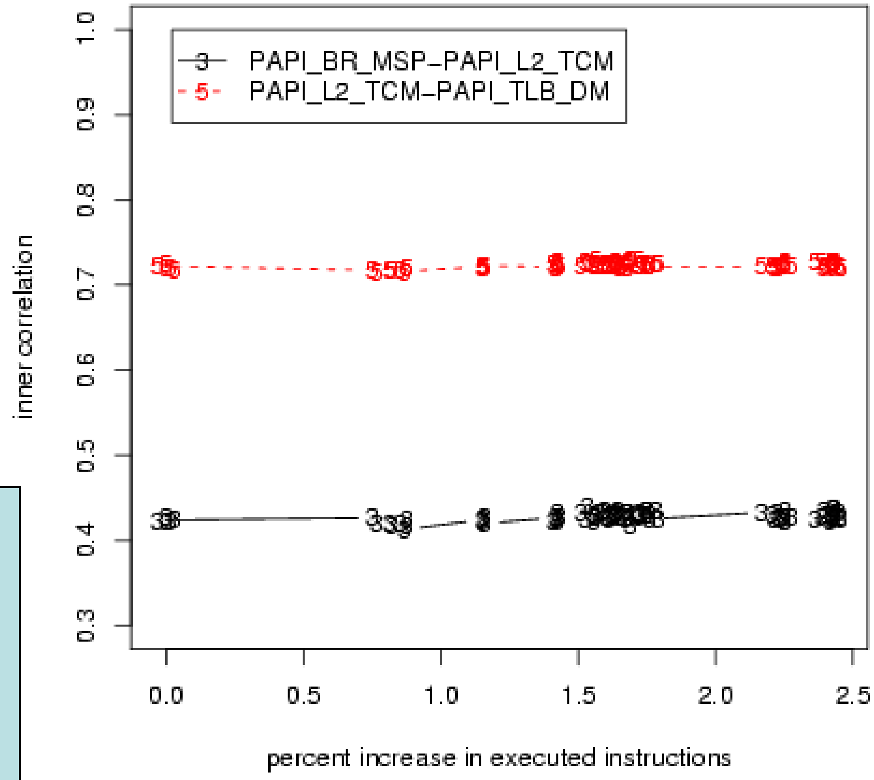




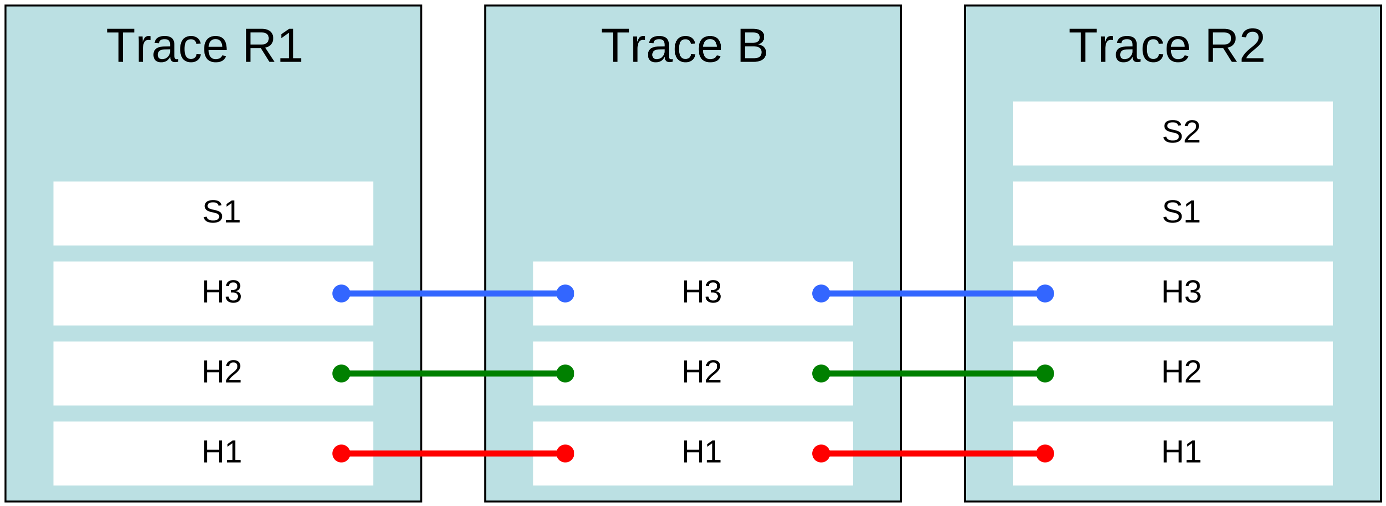
Minimal Inner Perturbation

- bzip (SPEC CPU2006)
- Multiple runs collecting different software metrics
- Graph inner correlation scores of hardware metrics as instrumentation is added
- Minimal inner perturbation
- Inner perturbation is benchmark specific





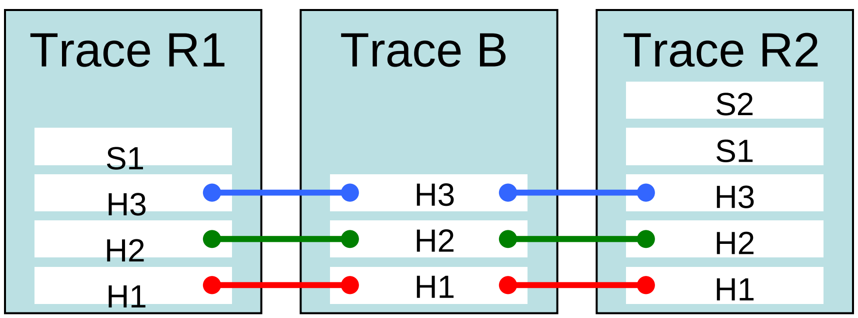
Outer Perturbation

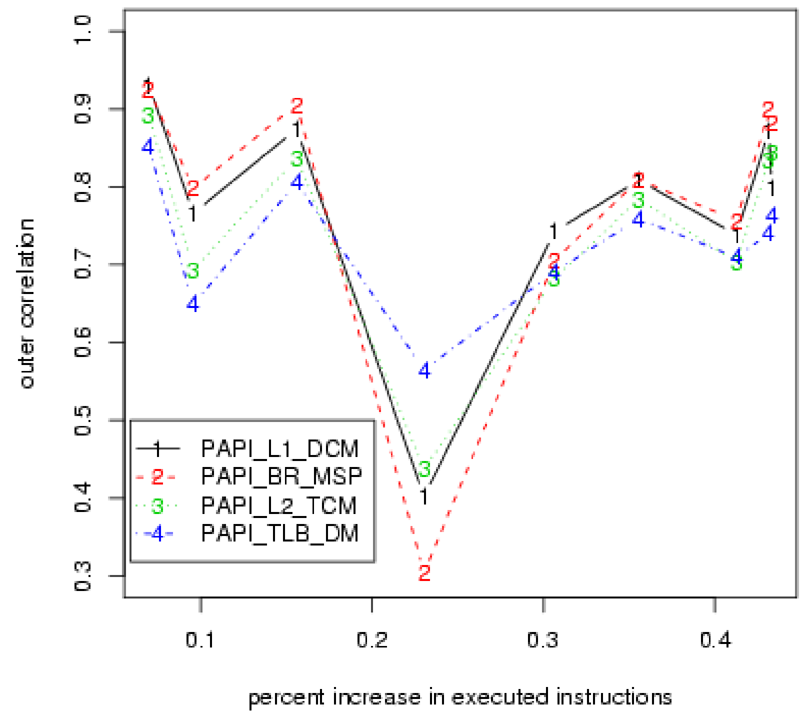


- Correlate same metric in baseline and in another trace
- Compare *outer* correlation scores across pairs of traces
  - E.g. compare corr(B.H1, R1.H1) with corr(B.H1, R2.H1)
- Observe how correlation changes as additional instrumentation is added
- Assumes technique to align traces
  - We use DTW

Significant Outer Perturbation

- Sim-outorder with gcc as input
- Multiple runs collecting different software metrics
- Graph outer correlation of hardware metrics as instrumentation is added
- Significant outer perturbation
- But recovers

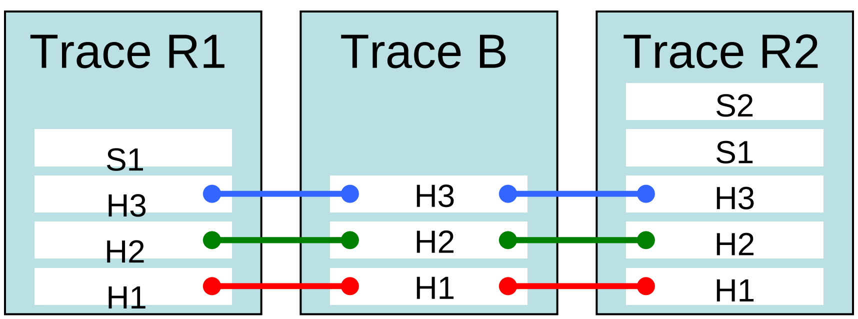


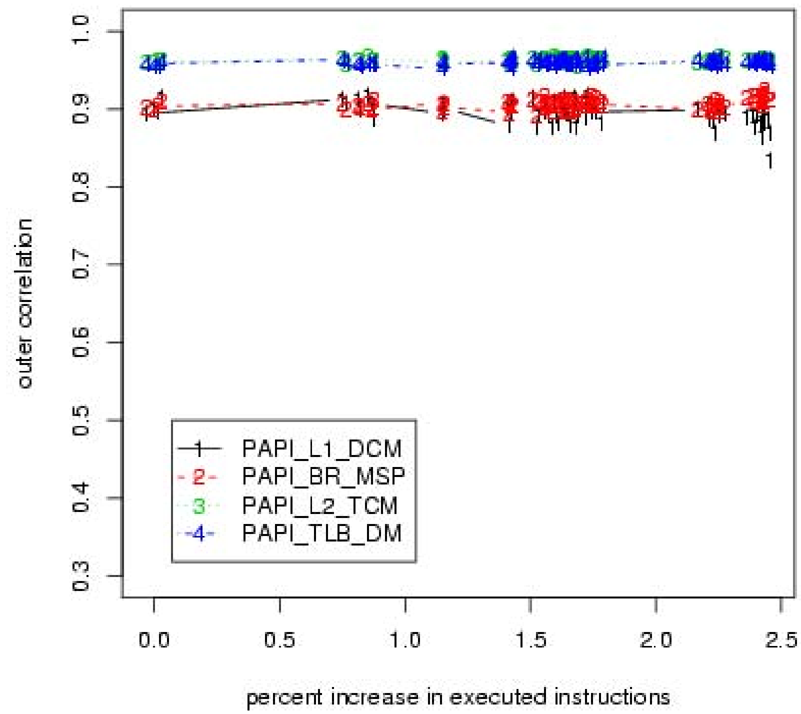


Minimal Outer Perturbation

- bzip (SPEC CPU2006)
- Multiple runs collecting with different metrics
- Graph outer correlation of hardware metrics as instrumentation is added
- Minimal outer correlation
- Outer perturbation is benchmark

  specific





Conclusions

- Low overhead !=> low perturbation
  - Minimal instrumentation overhead can result in significant perturbation
    - Less than 3% increase in executed instructions prevented reasoning about metrics within or across traces
- Perturbation is application specific
- Perturbation is not monotonic
  - Additional instrumentation may increase or decrease perturbation!
  - Makes impact of instrumentation hard to predict
- This is a starting point for a more in depth study!

Related Work

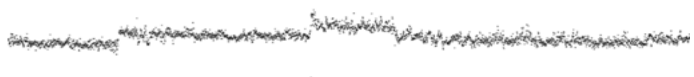
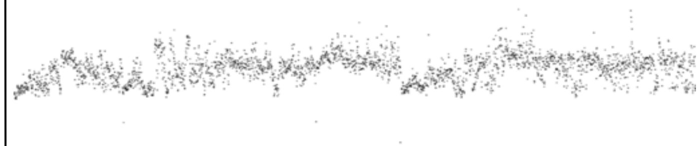
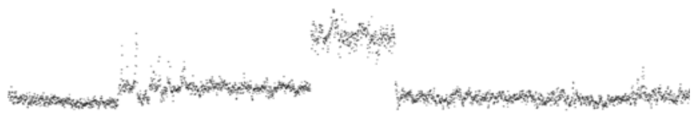
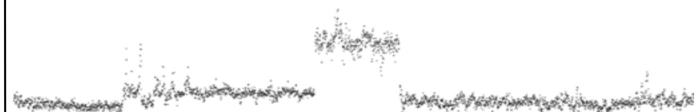
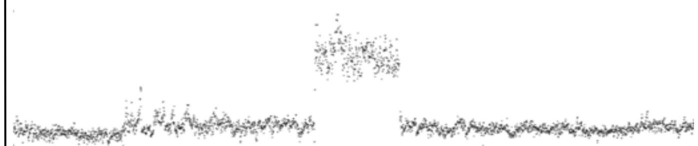
- Perturbation measurement [Daigle et al.]
  - Operational definition of perturbation
    - Aggregate runtime slowdown as function of instrumentation
- Perturbation management [Maloney]*
  - Use perturbation model to eliminate perturbation effects from a trace
    - Only as good as model
      - Difficult to model out-of-order superscalar machines
    - Overall program run time

Questions

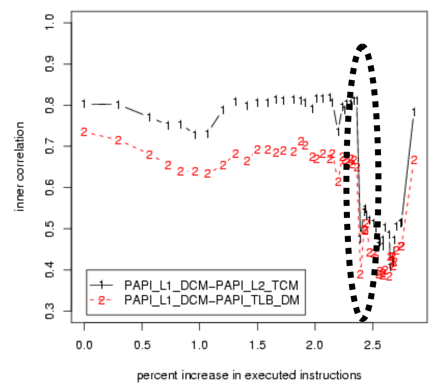
?

- How does perturbation impact our ability to reason about system behavior?

Inner Perturbation Details

	Good	Perturbed
L1_DCM		
L2_TCM		
S	*not measured*	

- Each row different metric
- “Good” is trace before perturbation
- “Perturbed” is trace after perturbation
- “S” is the software metric collected in “Perturbed” but not in “Good”



How to Evaluate Perturbation?

- Any instrumentation perturbs system behavior
- Count number of times a metric occurs
  - Hardware: no charge
  - Software: cost to increment
- Baseline trace
  - Only collect hardware metrics
    - Cost is to periodically collect metrics
  - Expect minimal perturbation, but no guarantee
  - Expect relationship between metrics are preserved

IPC over time for SPECjvm98