Methodology

Lizy John, 2007 ISCA

- Similar programs in benchmark suites don’t add info
- Need to statistically analyze benchmarks
- Begin by using PAPI to log dynamic instruction counts

SPEC Int Characteristics

Name – Language	Inst. Count (Billion)	Branches	Loads	Stores
CINT 2006				
400.perlbench –C	2,378	20.96%	27.99%	16.45%
401.bzip2 – C	2,472	15.97%	36.93%	12.98%
403.gcc – C	1,064	21.96%	26.52%	16.01%
429.mcf –C	327	21.17%	37.99%	10.55%
445.gobmk –C	1,603	19.51%	29.72%	15.25%
456.hmmer –C	3,363	7.08%	47.36%	17.68%
458.sjeng –C	2,383	21.38%	27.60%	14.61%
462.libquantum-C	3,555	14.80%	33.57%	10.72%
464.h264ref- C	3,731	7.24%	41.76%	13.14%
471.omnetpp- C++	687	20.33%	34.71%	20.18%
473.astar- C++	1,200	15.57%	40.34%	13.75%
483.xalancbmk- C++	1,184	25.84%	33.96%	10.31%

SPEC FP Characteristics

Name – Language	Inst. Count (Billion)	Branches	Loads	Stores
CFP 2006				
410.bwaves – Fortran	1,178	0.68%	56.14%	8.08%
416.gamess – Fortran	5,189	7.45%	45.87%	12.98%
433.milc – C	937	1.51%	40.15%	11.79%
434.zeusmp–C,Fortran	1,566	4.05%	36.22%	11.98%
435.gromacs-C, Fortran	1,958	3.14%	37.35%	17.31%
436.cactusADM-C, Fortran	1,376	0.22%	52.62%	13.49%
437.leslie3d – Fortran	1,213	3.06%	52.30%	9.83%
444.namd – C++	2,483	4.28%	35.43%	8.83%
447.dealII – C++	2,323	15.99%	42.57%	13.41%
450.soplex – C++	703	16.07%	39.05%	7.74%
453.povray – C++	940	13.23%	35.44%	16.11%
454.calculix –C, Fortran	3,041	4.11%	40.14%	9.95%
459.GemsFDTD – Fortran	1,420	2.40%	54.16%	9.67%
465.tonto – Fortran	2,932	4.79%	44.76%	12.84%
470.lbm – C	1,500	0.79%	38.16%	11.53%
481.wrf - C, Fortran	1,684	5.19%	49.70%	9.42%
482.sphinx3 – C	2,472	9.95%	35.07%	5.58%

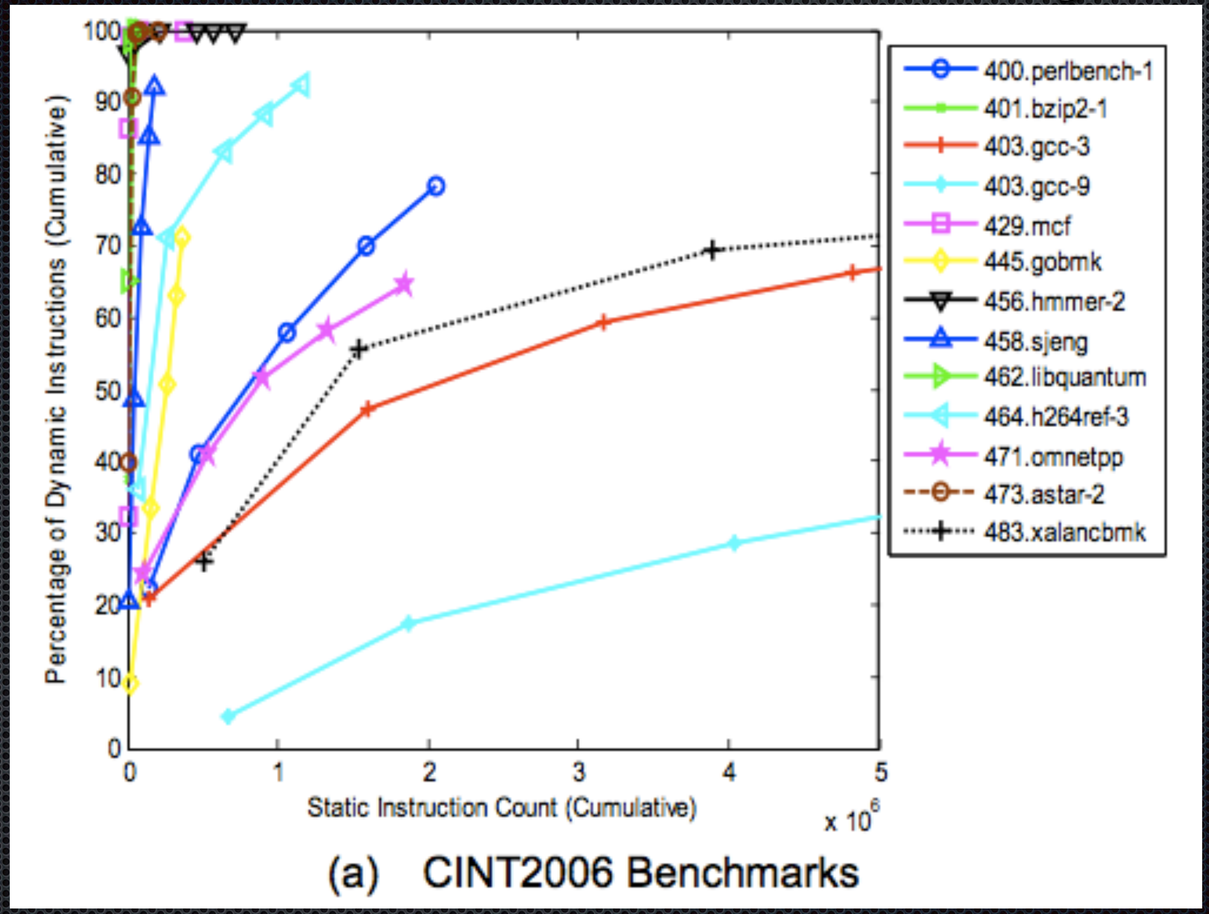
Comparison

- Higher rate of branches in integer codes
  - 20% on average vs. 5%
- Slightly lower load rate but similar store rate vs. FP

Instruction Locality

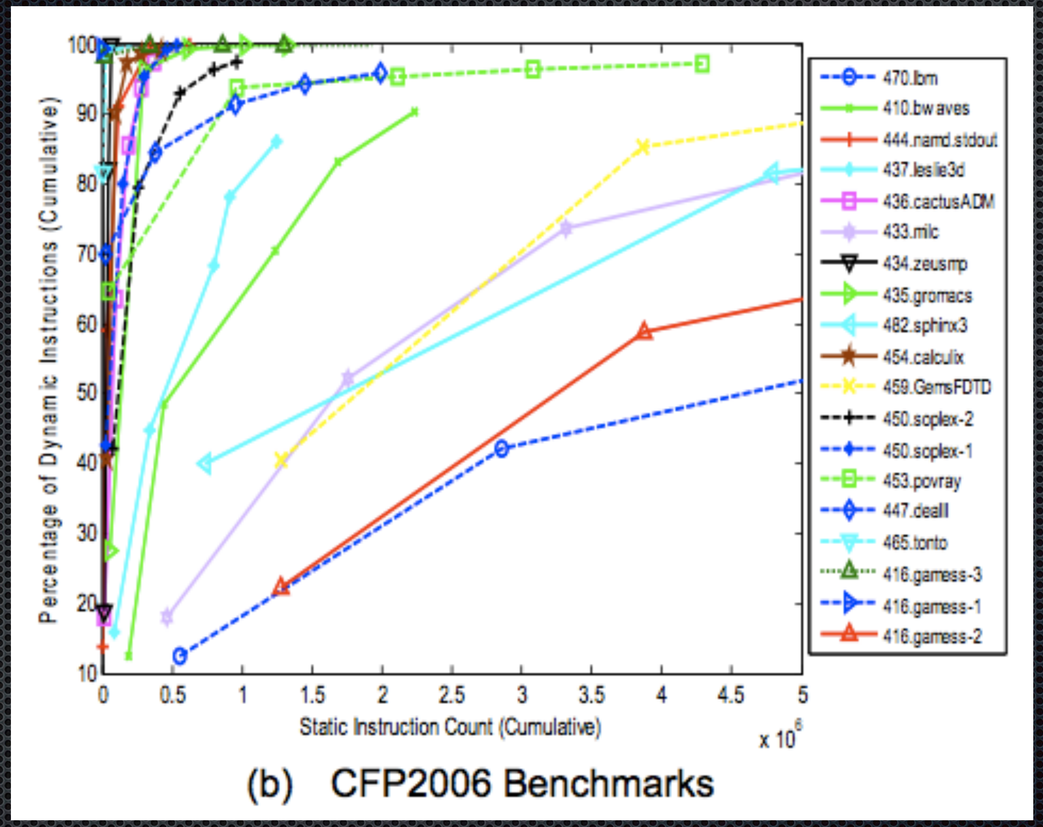
- Use PIN to measure fraction of dynamic (executed) instructions in top N hottest subroutines
- Plot against static instructions
- If a high fraction of dynamic instructions are covered by a small fraction of the static instructions, then the code has good locality
  - Good locality may also imply that the data set doesn’t exercise a large portion of the application

SPEC Int Instruction Locality



bzip, mcf, hmmer, libquantum, astar have high locality

SPEC FP Instruction Locality



(b) CFP2006 Benchmarks

namd, cactus, zeusmp, gromacs, calculix, soplex1, tonto, gamess3, gamess1 high in locality

Other Gross Characteristics

Table 2: Range of important performance characteristics of SPEC CPU2006 benchmarks

Metric	Min	Max
I-cache miss ratio	~ 0	1.7%
L1 D-cache miss ratio	6.3%	33%
L2 cache misses per instruction (per L2 access)	~0 (0.01%)	2.4% (49%)
DTLB miss ratio	0.2%	8.4%

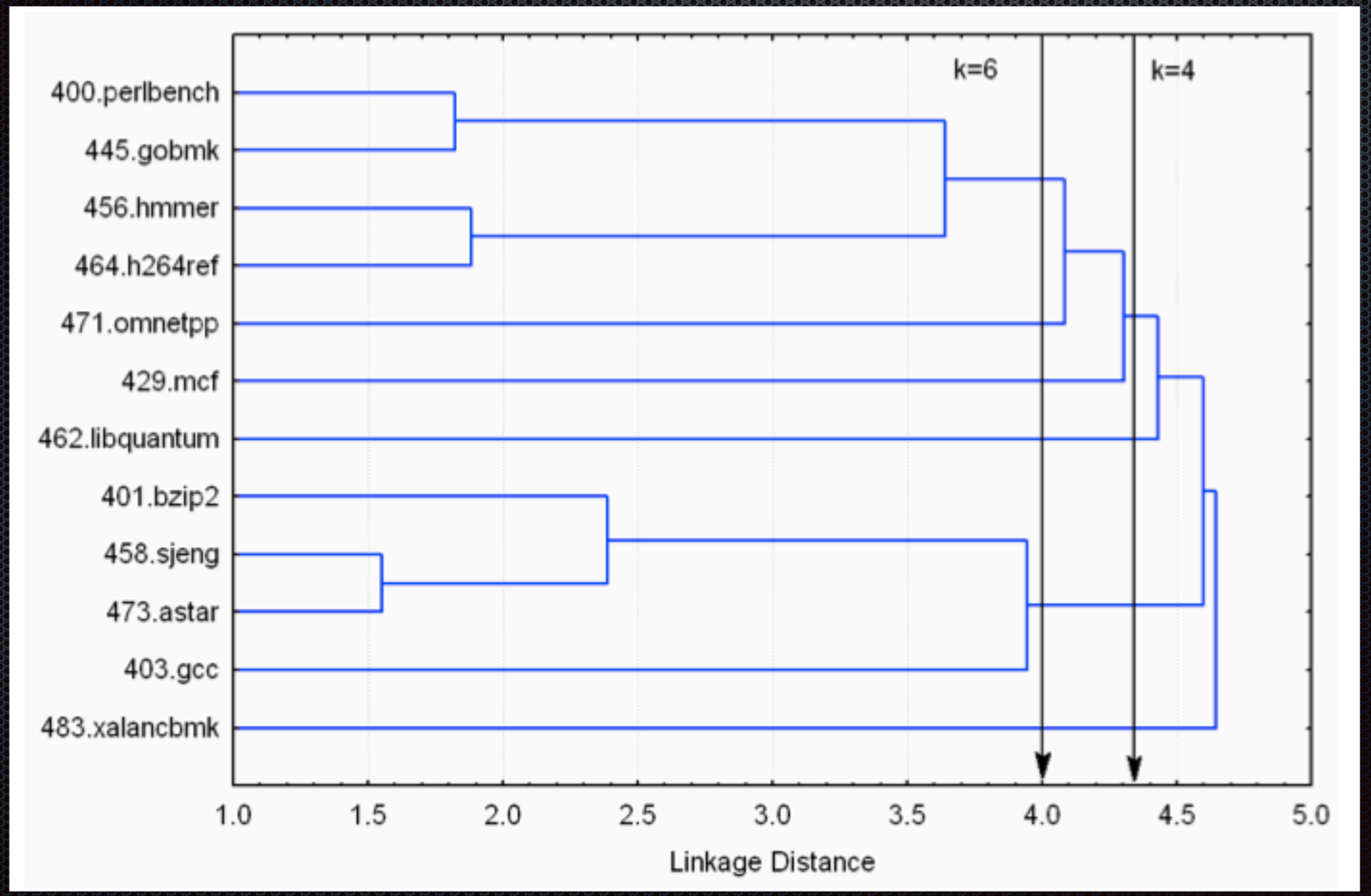
Data Source and Processing

- Six performance counter metrics X five machines = 30 variables
- Obtained from vendors under NDA
- Need to eliminate redundant info
- Principle Component Analysis identifies most significant factors
- Discard those that do not contribute much info

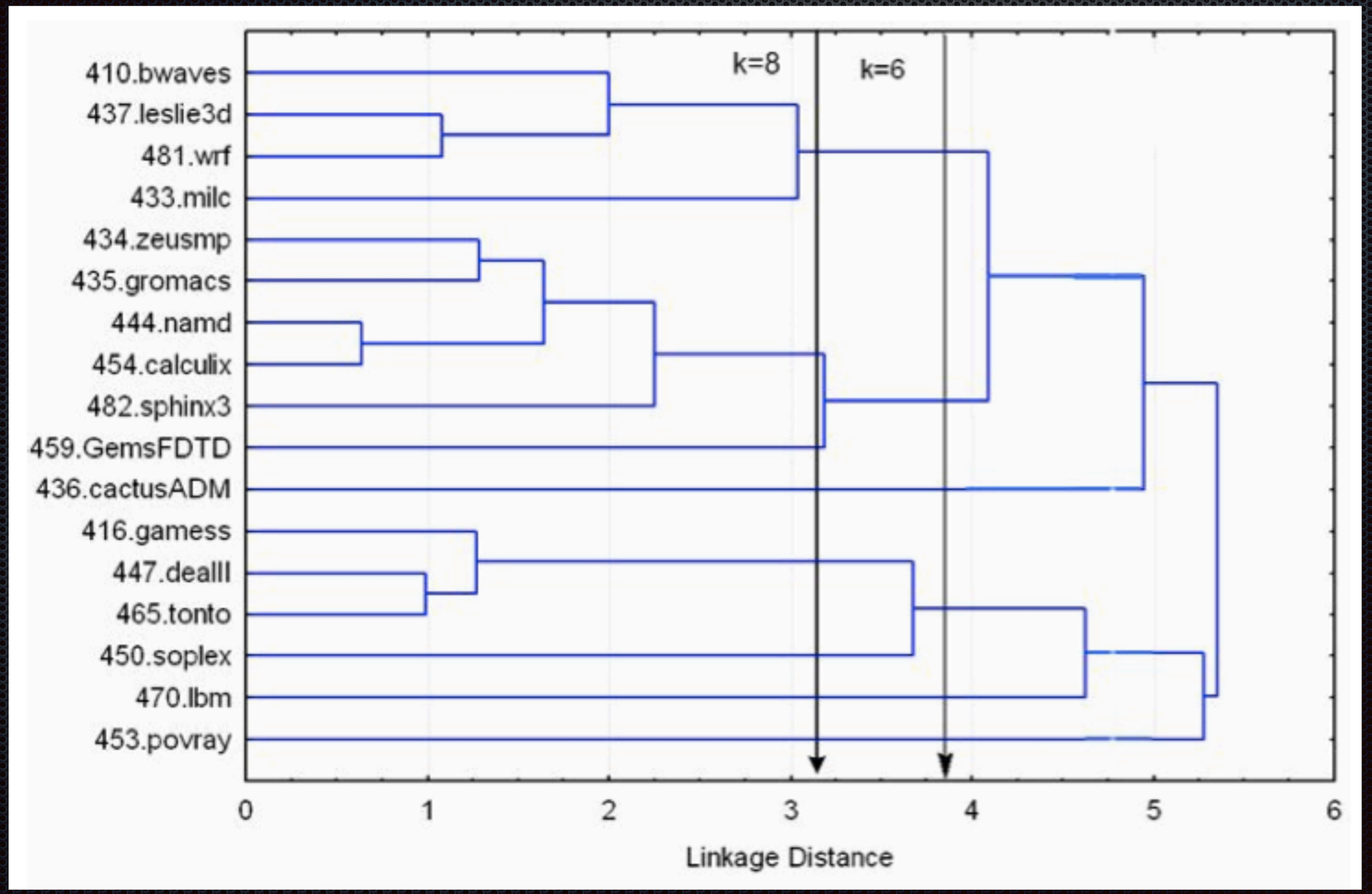
Clustering

- Hierarchical clustering groups two items with most similar characteristics
- Recomputes distances for new cluster
- Repeats until there is just one cluster
- Produces a dendrogram -- tree indicating similarity relationships

SPEC Int Dendrogram



SpecFP Dendrogram



For a Given Cluster

- Calculate distance from center
- Choose benchmark closest to center
- Fewer clusters reduces work, but may lump together too many benchmarks and reduce overall variance
- This analysis just shows best distribution within the set but doesn’t tell whether the set is representative

Suggested Subsets

Table 4. Representative subset of SPEC CINT2006 programs.

Subset of Four Programs	400.perlbench, 462.libquantum, 473.astar, 483.xalancbmk
Subset of Six Programs	400.perlbench, 471.omnetpp, 429.mcf, 462.libquantum, 473.astar, 483.xalancbmk

Table 5. Representative subset of SPEC CFP2006 programs.

Subset of Six Programs	437.leslie3d, 454.calculix, 436.cactusADM, 447.dealII, 470.lbm, 453.povray
Subset of Eight Programs	437.leslie3d, 454.calculix, 459.GemsFDTD, 436.cactusADM, 447.dealII, 450.soplex, 470.lbm, 453.povray

Validation

- Pick some SPEC reports
- Use figures for individual benchmarks to compare subsets to entire set
- Errors for 4 Int programs average 5.8%
- Errors for 8 FP programs average 7%

SPEC Int Validation

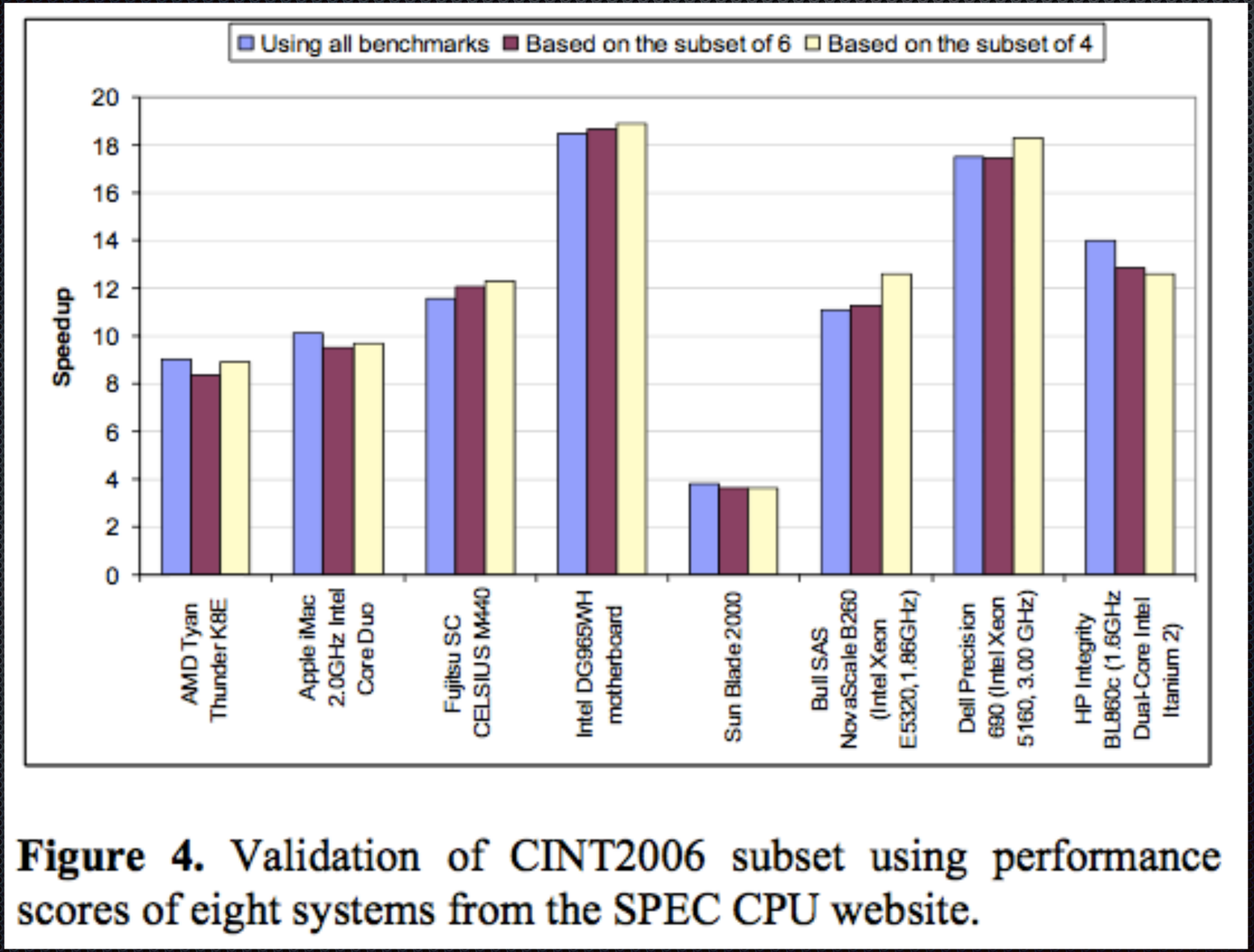


Figure 4. Validation of CINT2006 subset using performance scores of eight systems from the SPEC CPU website.

SPEC FP Validation

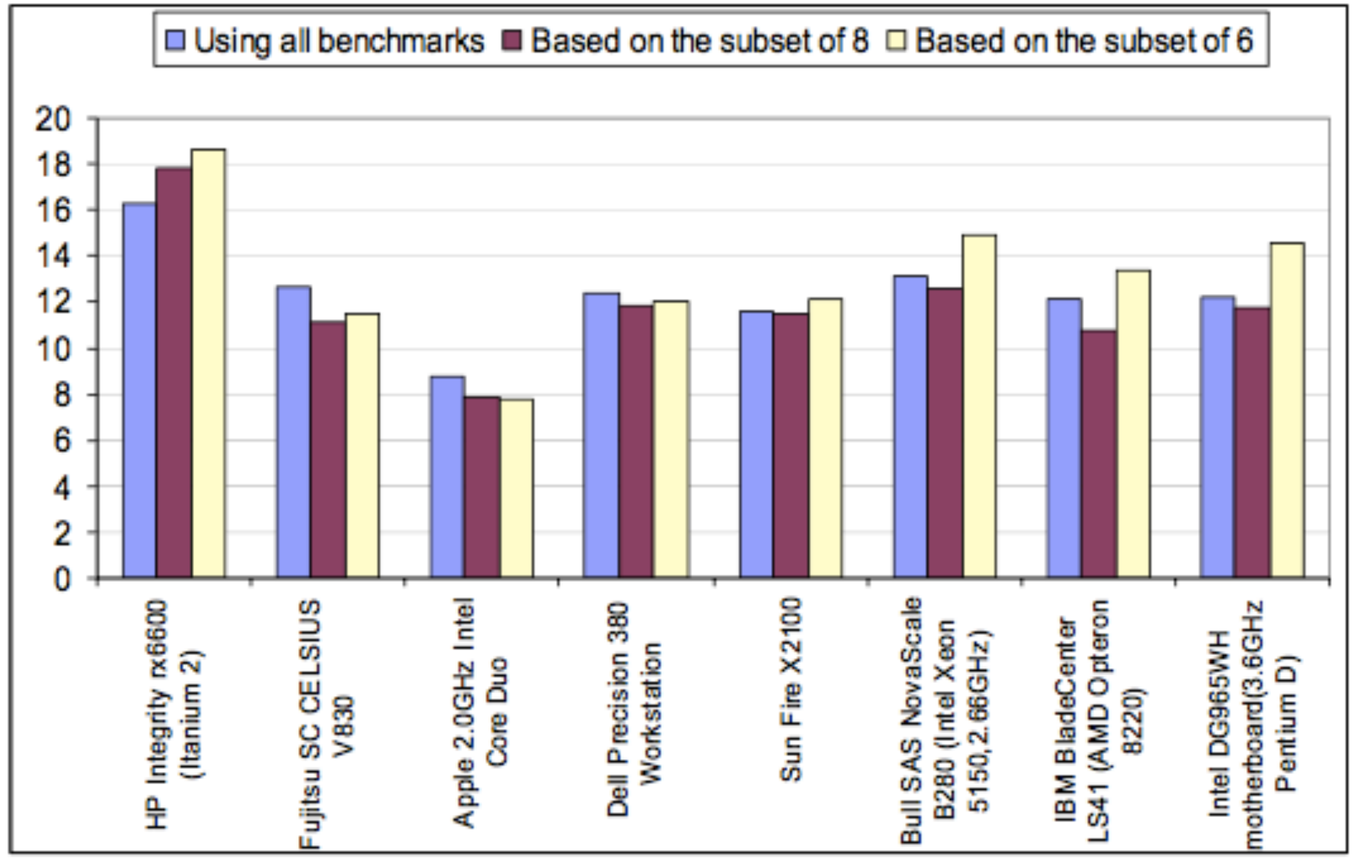
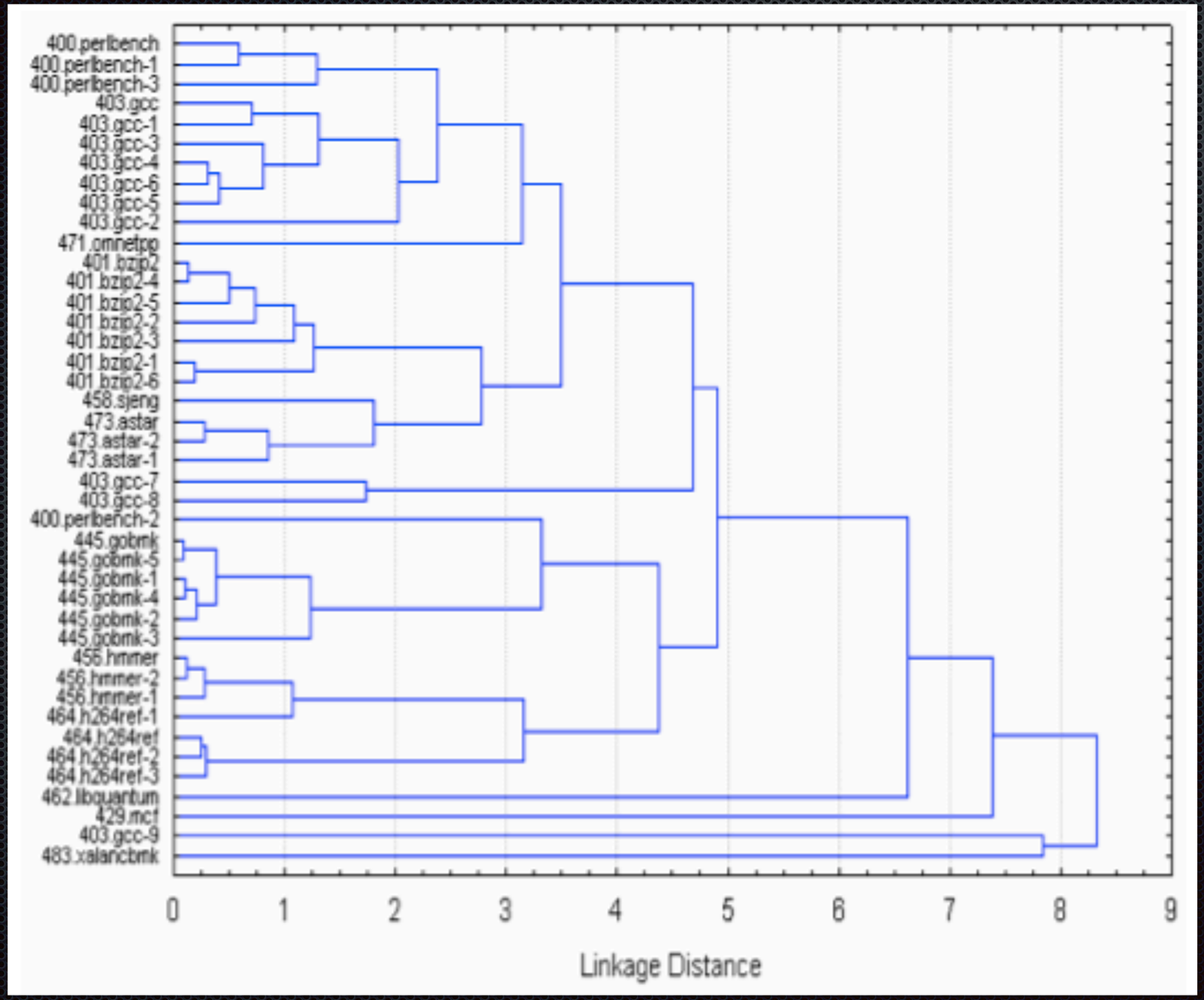


Figure 5. Validation of CFP2006 subset using performance scores of eight systems from the SPEC CPU website.

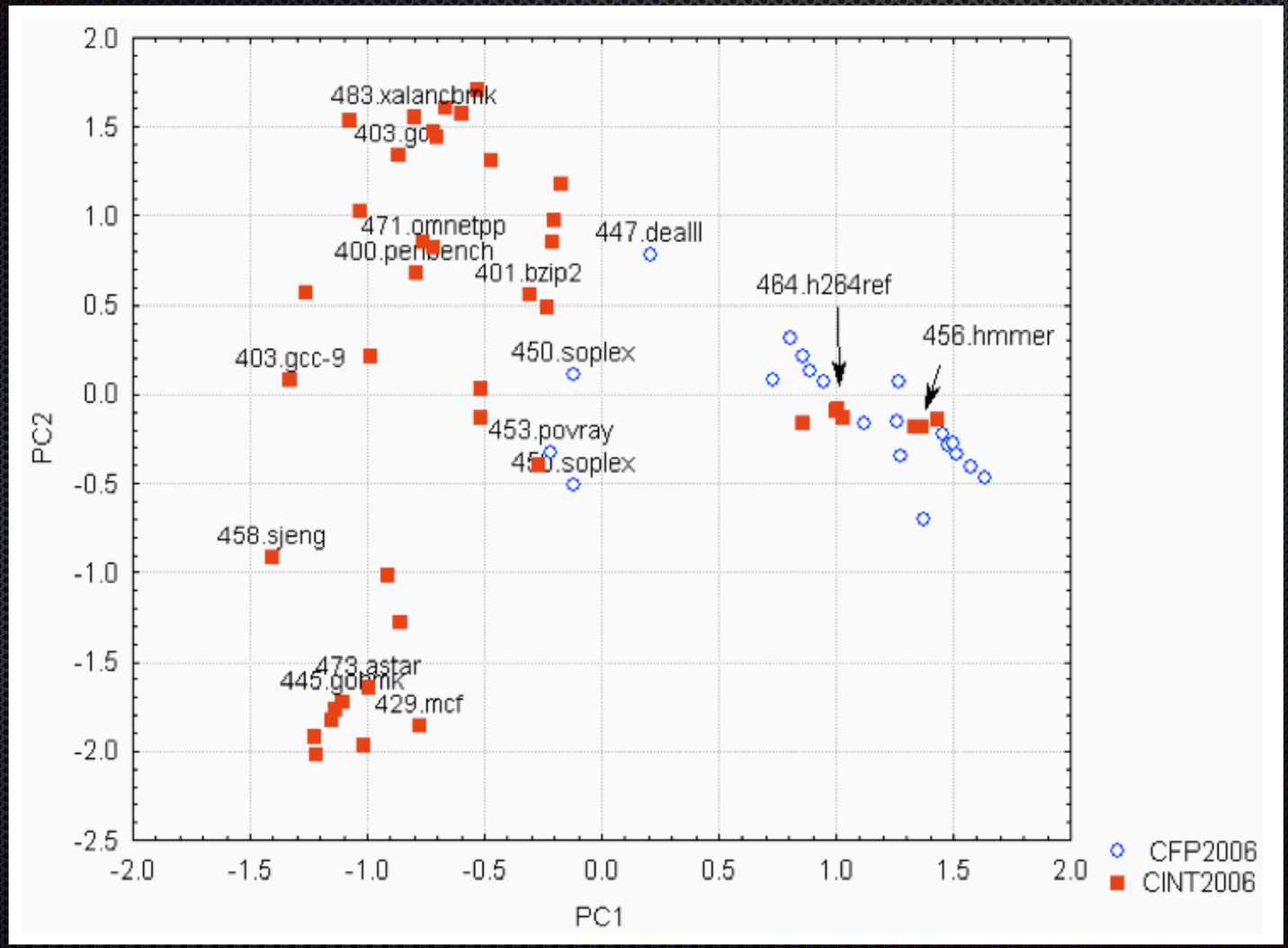
Data Set Variation

- Can be greater than program variation
- Need to analyze PCA clustering by data set

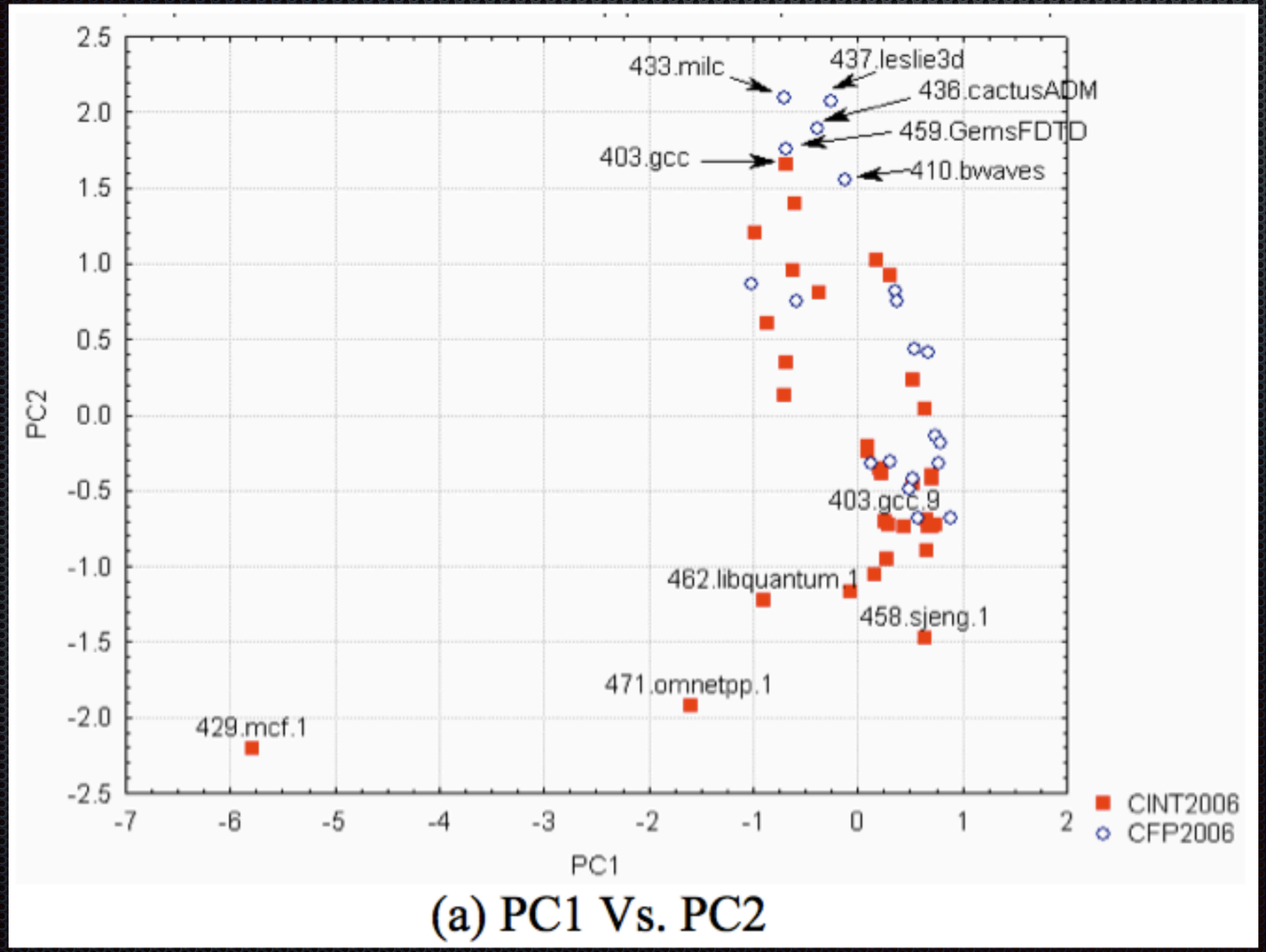
SPEC Int Data Set Clusters



Branch Prediction Clustering

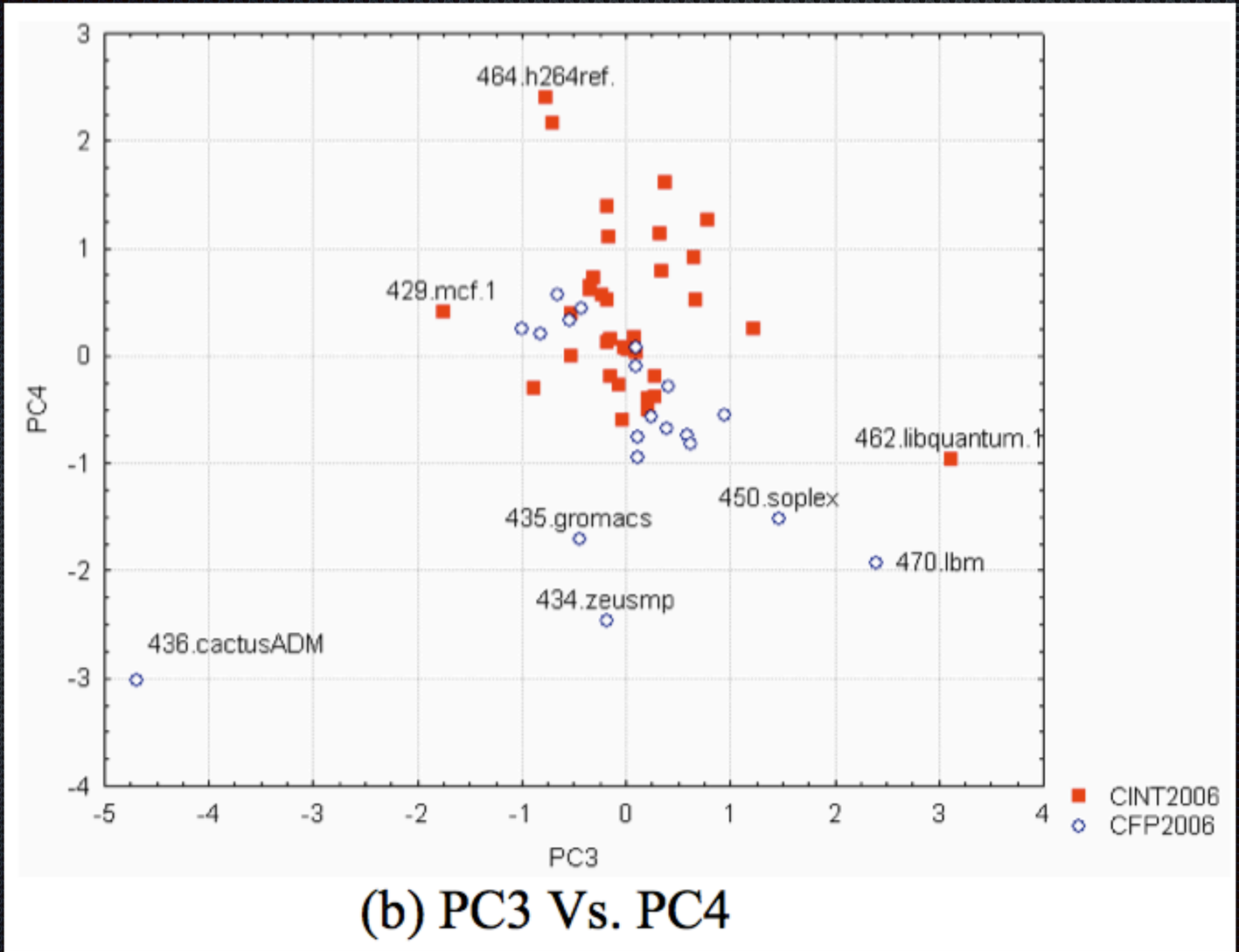


Data Access Clustering



(a) PC1 Vs. PC2

Data Access Clustering



(b) PC3 Vs. PC4

Further Analysis

- The EDA programs are replaced by others with similar coverage
- The 2006 benchmarks have more diversity than 2000
- Some of the applications from the same area are redundant, but others are not
- Some are very sensitive to architecture, while others (like gcc) are not

Discussion

Yang ISCA 2015

Computer Performance Microscopy with SHIM

Performance Measurement

- Enables deeper optimization of code
- Can lead to new compiler optimizations
- Can provide insights for architects, OS designers
- Enables comparison of systems

Performance Counters

- Designed for engineering studies
- Limited set of hardware-oriented events
- Fewer counters than event types - multiple passes
- Not always accurate, can overflow
- Privileged, requires OS access overhead
- Different on each model, not well documented

Software Monitoring

- Replace instructions with calls to logging functions
- Very flexible in level of event granularity and type
- Disrupts instruction flow
  - Significant overhead (increased time, memory)
  - Significant observer effect (disturbed behavior)
- No access to hardware and other asynchronous events

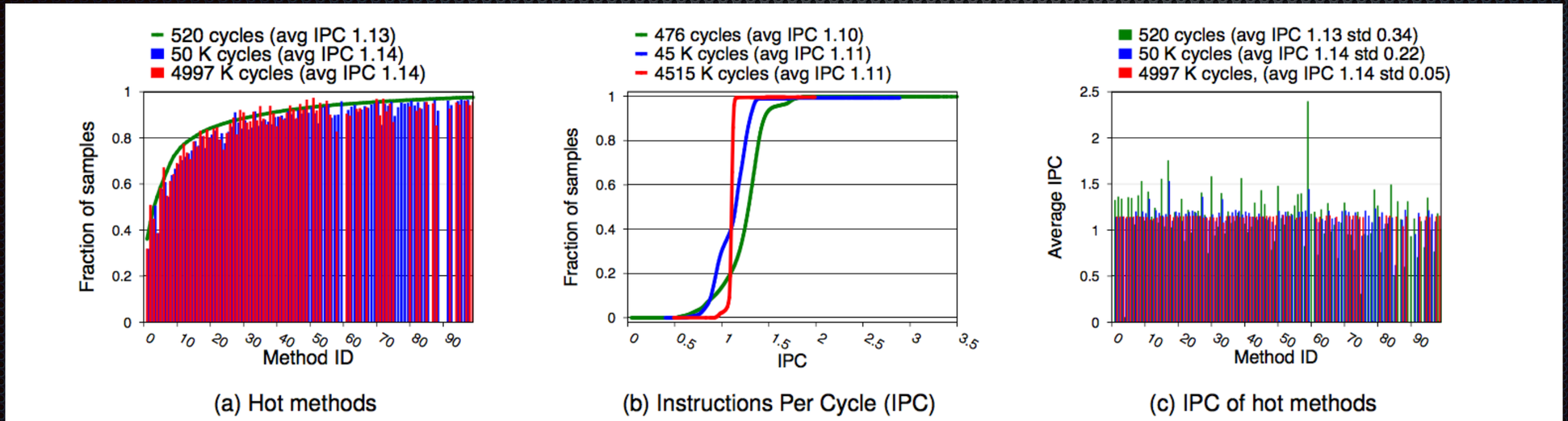
SHIM - Pros

- Runs in separate thread or on separate core
  - Software event signals monitor thread
- Reduces overhead and observer effect
- Reads both software and hardware events
- Operates at high frequency, sampling short periods
- Constant overhead/effects can be accounted for

SHIM - Cons

- Still disrupts hardware state, up to 61% overhead
- Fine grained data cannot be fully validated
- Runs in Java JIT environment
  - Needs lengthy runs to reach stability
  - Automatic GC limits ability to fine tune
  - Java isn’t a language of choice for high performance
- Not clear how it works with highly parallel code

Sample Rate



(a) Hot methods

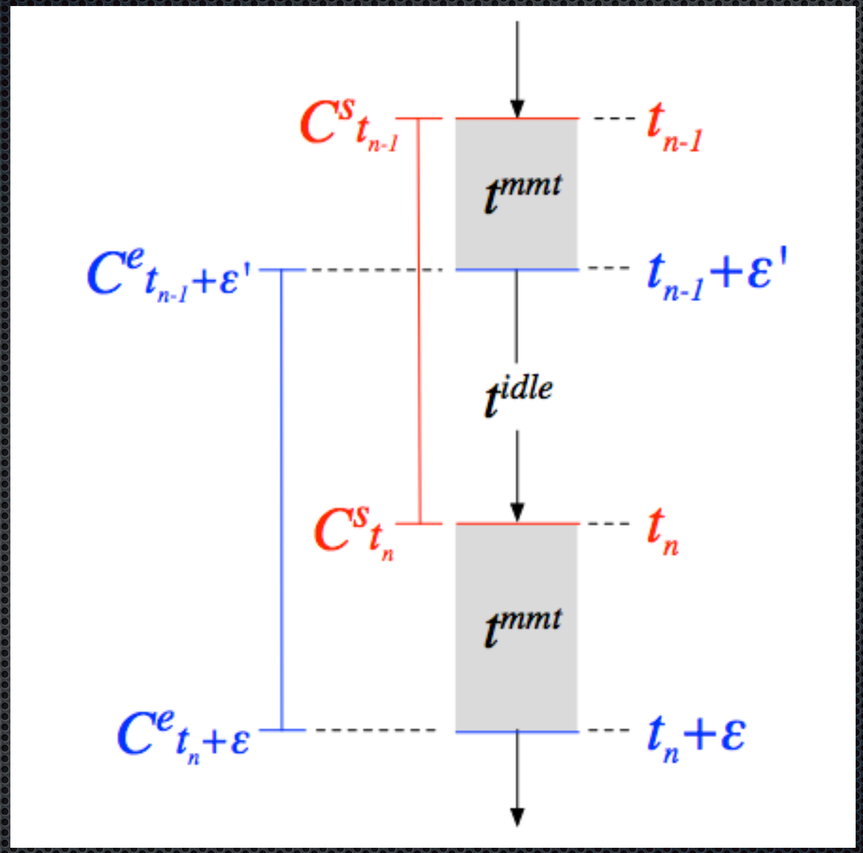
(b) Instructions Per Cycle (IPC)

(c) IPC of hot methods

Figure 1: The impact of sample rate on lusearch. (a) Varying the sample rates identifies similar hot methods. The green curve is the cumulative frequency distribution of samples at high frequency. Medium (red) and low (blue) frequency sampling cumulative frequency histograms are permuted to align with the green. (b) Sample rate significantly affects measures such as IPC. (c) Sample rate significantly affects IPC of hot methods. Each bar shows average IPC for a given hot method at one of three sample rates.

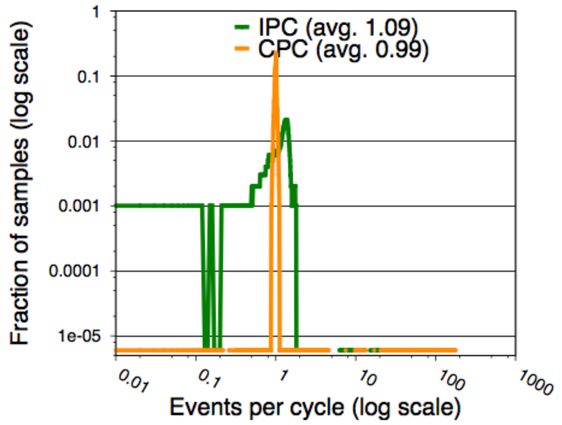
Coarse sampling blurs some effects

Reducing Noise

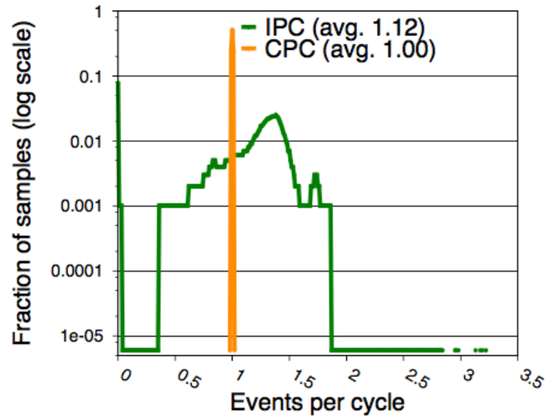


Overlapping sample periods helps identify disrupted behaviors, when measurements are inconsistent

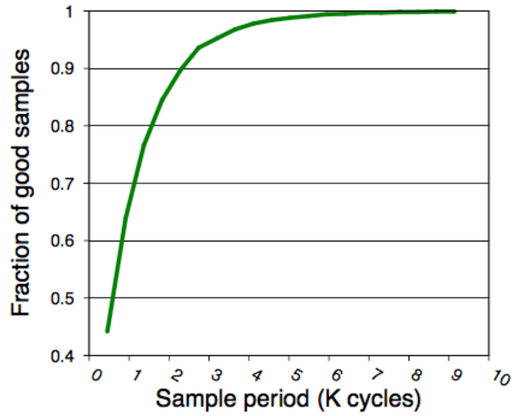
Effects of Overlapping



(a) Unfiltered IPC and CPC distortions, e.g., IPC> 4 on a 4-way superscalar processor.



(b) DTE correction with ±1% CPC error eliminates all distorted IPC values.



(c) Fraction of good samples with ±1% CPC error as a function of sampling frequency.

Figure 4: DTE filtering on SMT keeps samples for which ground truth CPC is 1.±0.01, eliminating impossible IPC values. At small sample periods, DTE discards over half the samples. At sample periods >2000, DTE discards 10% or fewer samples.

Eliminates most bad signal values,
but at fine grained rates, over 50% are bad

Randomness

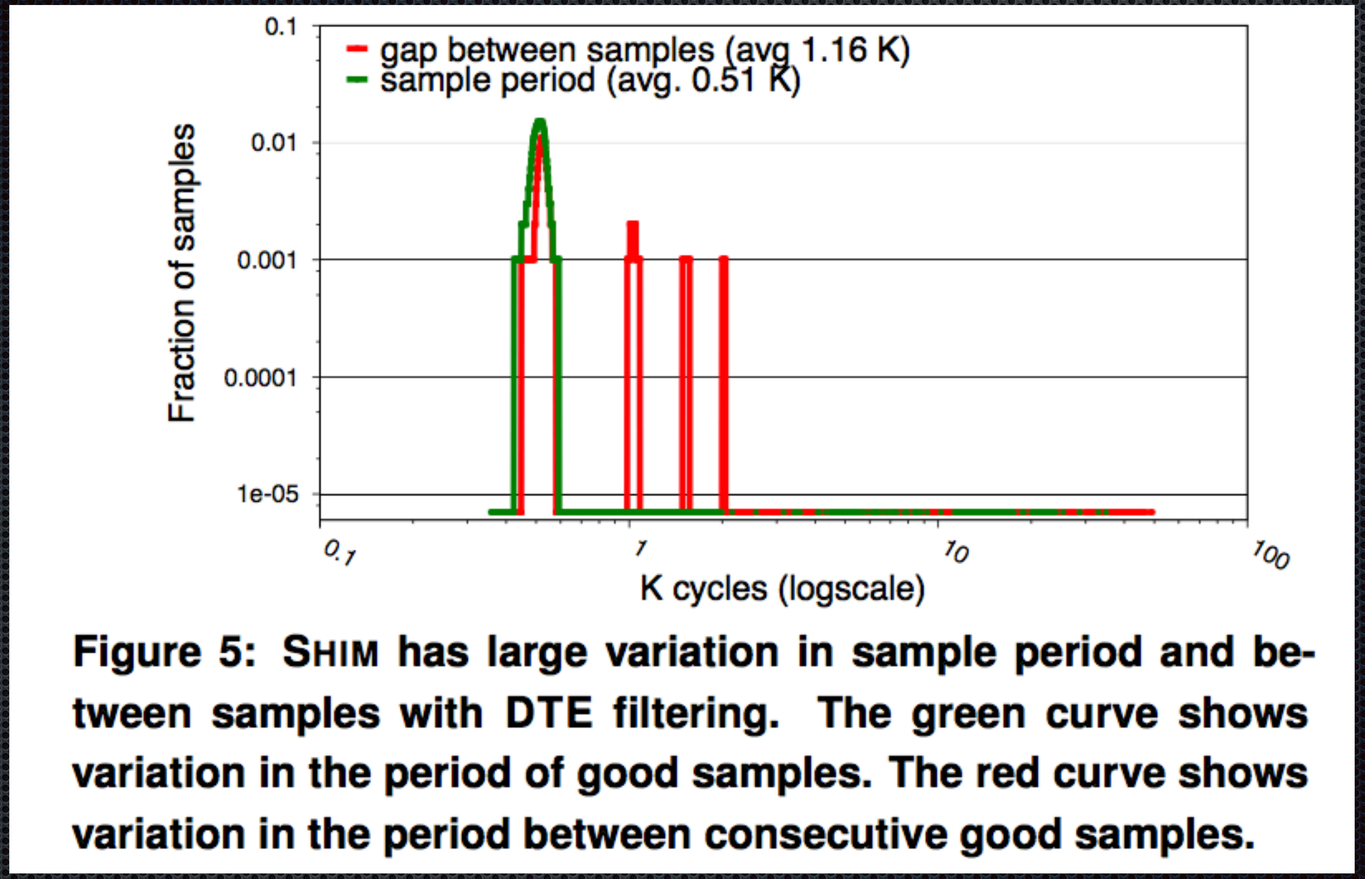


Figure 5: SHIM has large variation in sample period and between samples with DTE filtering. The green curve shows variation in the period of good samples. The red curve shows variation in the period between consecutive good samples.

Randomizing good samples is claimed to reduce errors related to periodicity

Observer Effect

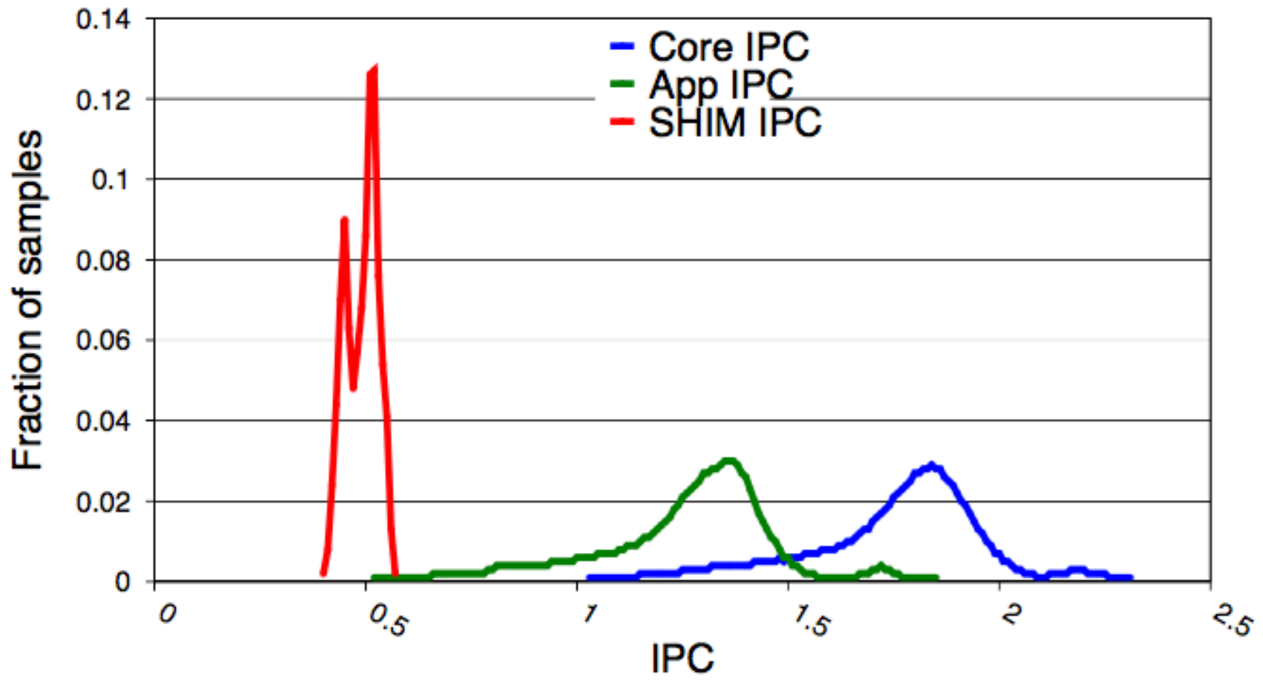
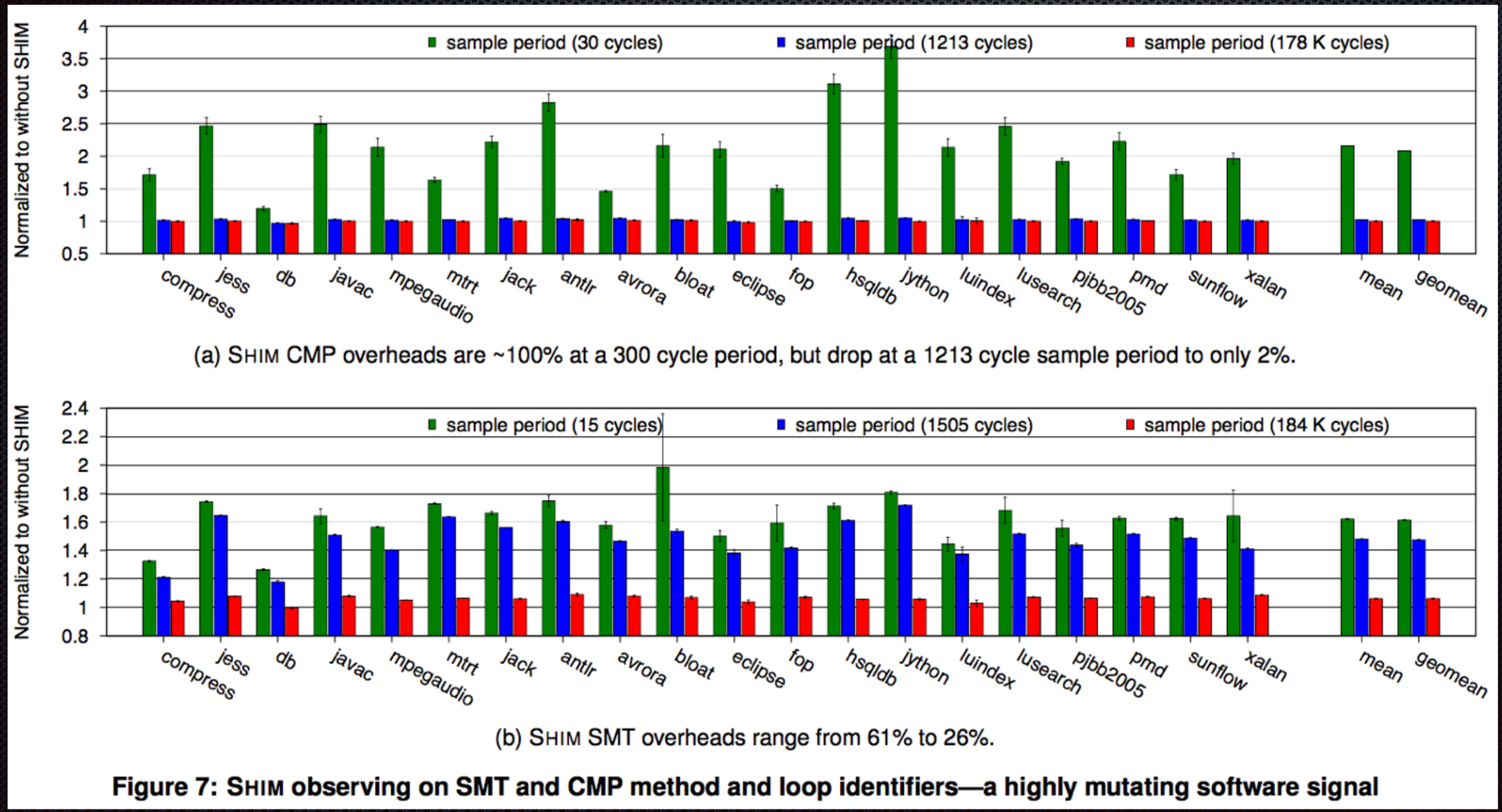


Figure 6: SHIM SMT observer effect on IPC for 476 cycle sample period with lusearch. The green curve shows lusearch IPC, red shows SHIM's IPC, and blue shows IPC for the whole core.

Overhead Core vs. SMT



Example: Garbage Collection

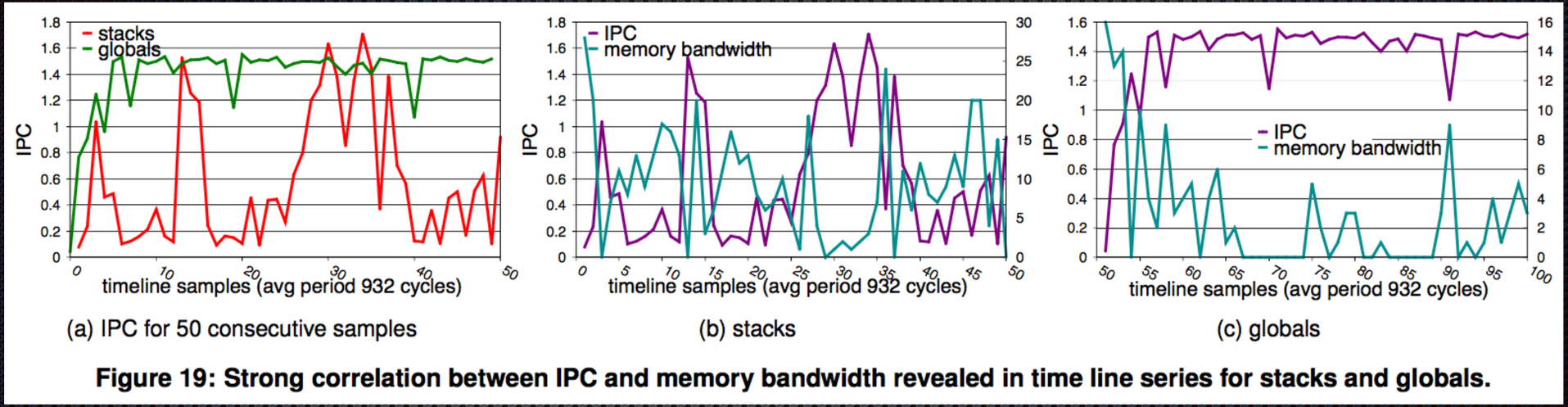


Figure 19: Strong correlation between IPC and memory bandwidth revealed in time line series for stacks and globals.

Memory bound task

Discussion