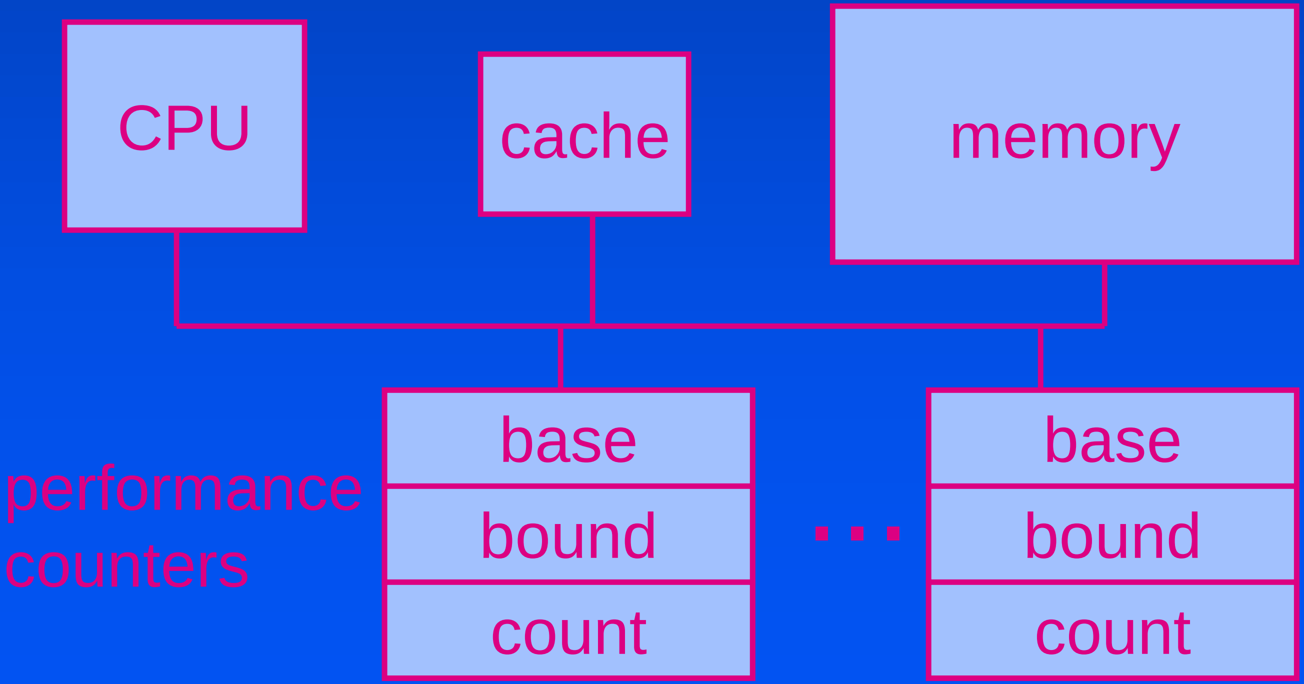
Toward Measuring Memory Hierarchy Effects by Region

Bryan Buck

Jeffrey K. Hollingsworth

Measuring Cache Effects by Region

- Simple base/bound register
  - Duplicate cache related performance counters
  - Each counter set collects info in own base/bounds
  - Difficult to convince chip makers to include



(assumes counters can use virtual, not physical addresses)

Approximating in Software

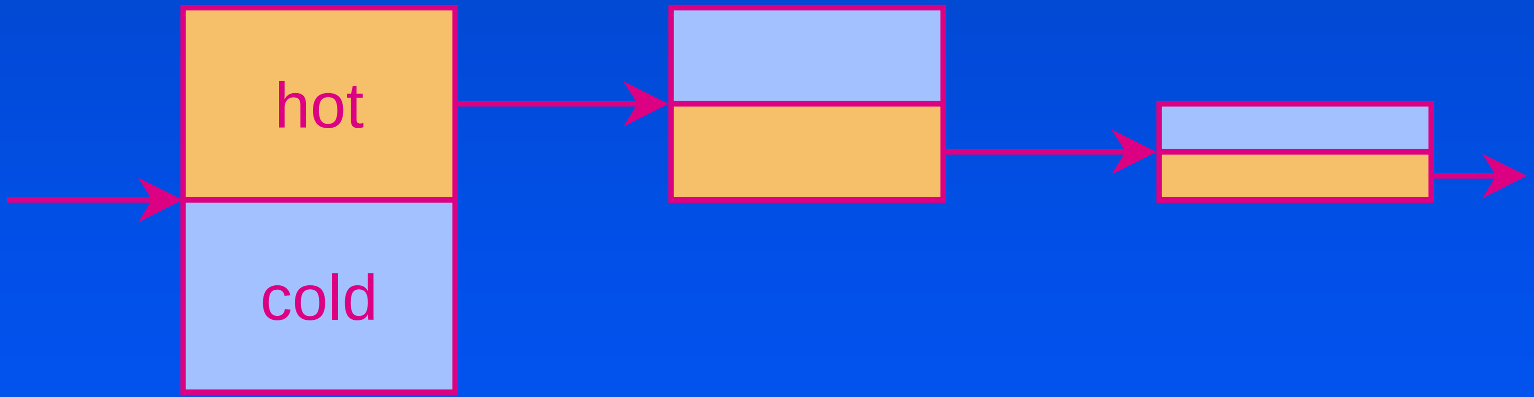
- Use a software cache simulator
  - Instrument applications to keep statistics
- Useful without hardware if fast enough
  - Need to instrument all load/stores, could be slow
- Demonstrates potential benefits
  - Even if slow, tells us if hardware would be useful

Cache Simulator

- Used existing simple cache simulator
  - Instruments application using ATOM
    - Calls cache simulator at loads/stores
    - Cycle count updated at each basic block
  - No need to change application source code
    - Optionally make a call to start measurement
- Added multiple performance counters
  - Each counter set has base/bounds

Memory Hotspot Search

- Goal: identify region causing most misses
- Use *n*-way search
  - Start with all memory split *n* ways and narrow down
  - Sample counters at regular intervals and readjust
  - Question: how does *n* affect the results?



- Tested on SPEC95 benchmark applications

Search Results

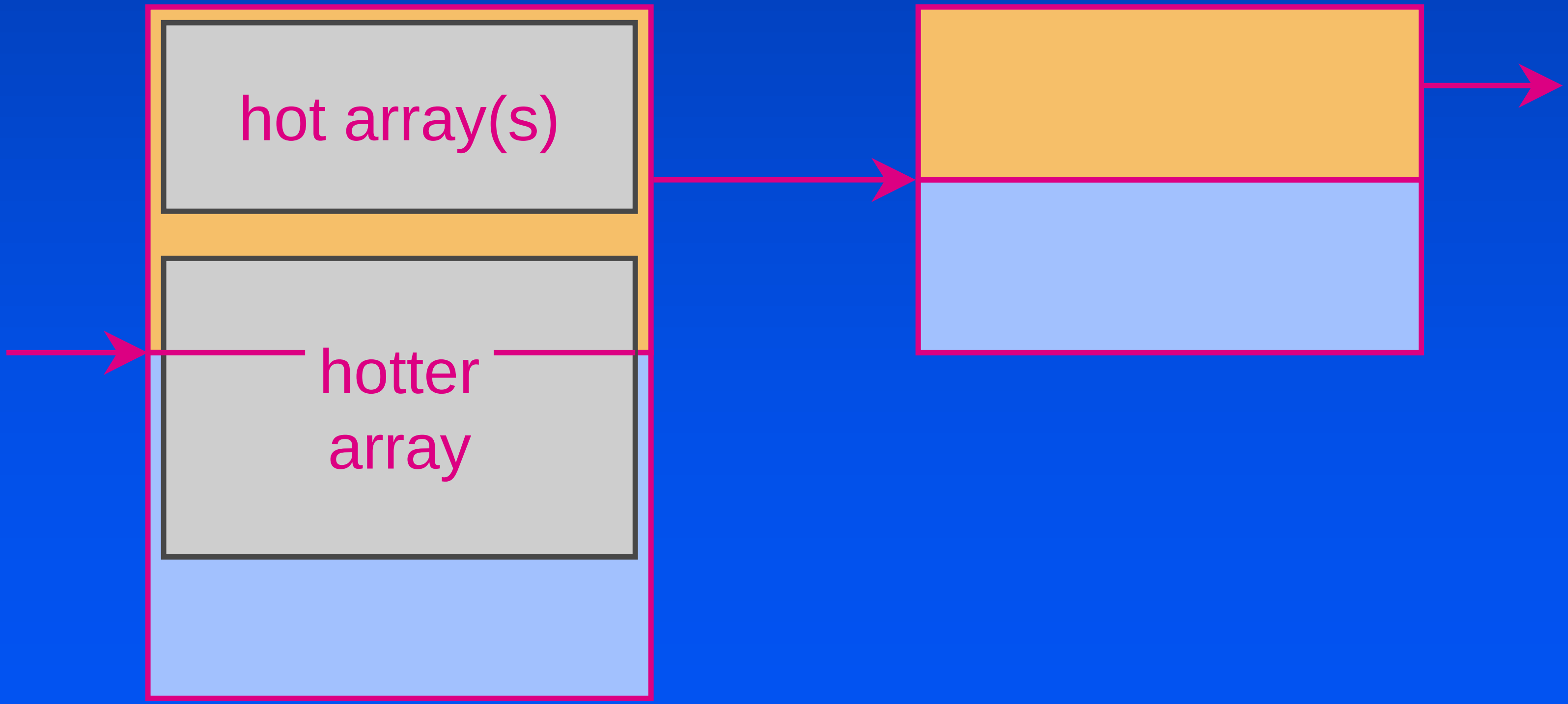
application	variable	% of misses	2-way	10-way
tomcatv	RX	23.59		x
	RY	23.57		x
	DD	9.71	x	
	D	9.60	x	
swim	H	7.72		
	UOLD	7.70		x
	VOLD	7.70		x
	UNEW	7.69	x	
	PNEW	7.69		x
	POLD	7.69		x
	P	7.64	x	

Search Results Continued

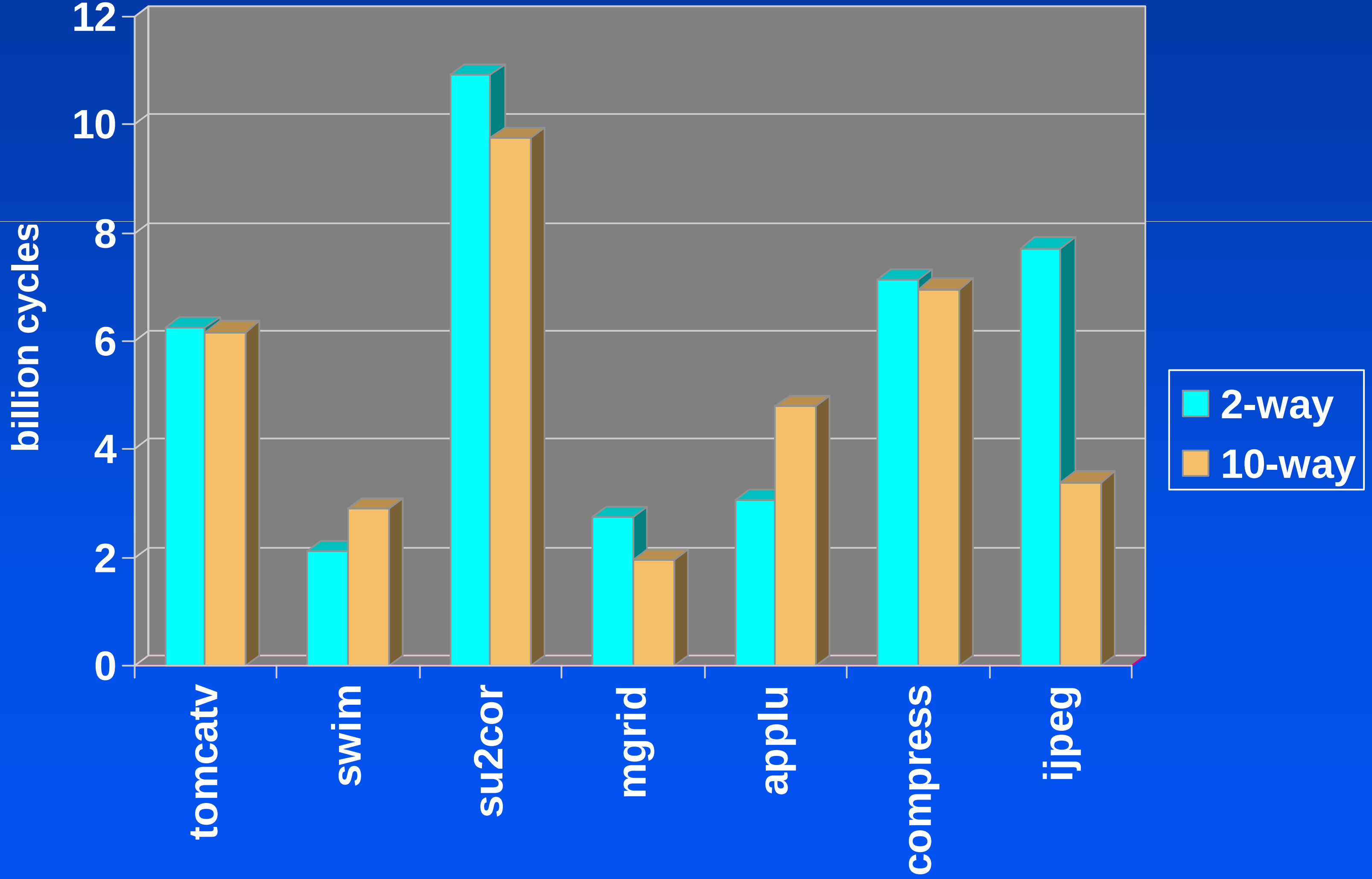
application	variable	% of misses	2-way	10-way
su2cor	U	43.35		
	W1	11.47	x	x
	B	9.82		x
	W2	9.04		x
mgrid	U	41.76		
	R	40.88	x	x
applu	B	21.23		
	A	21.22	x	
	C	21.22		x
compress	htab	66.49		x
	codetab	25.81	x	
ijpeg	jpeg_com…	3.92	x	x

Arrays Crossing Search Boundaries

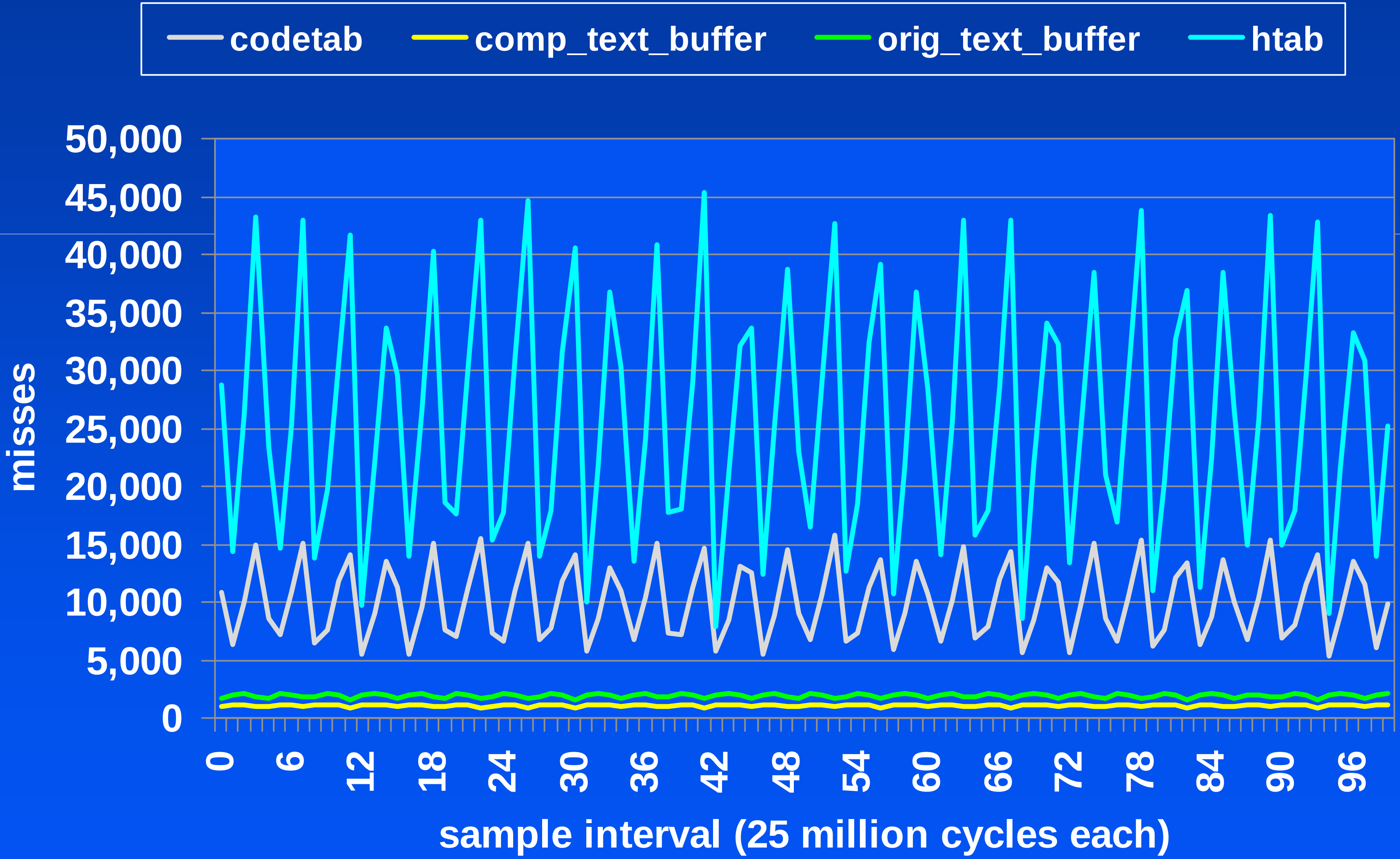
- An array may span two or more regions
  - Not enough misses in single region for detection
  - This is the problem with su2cor



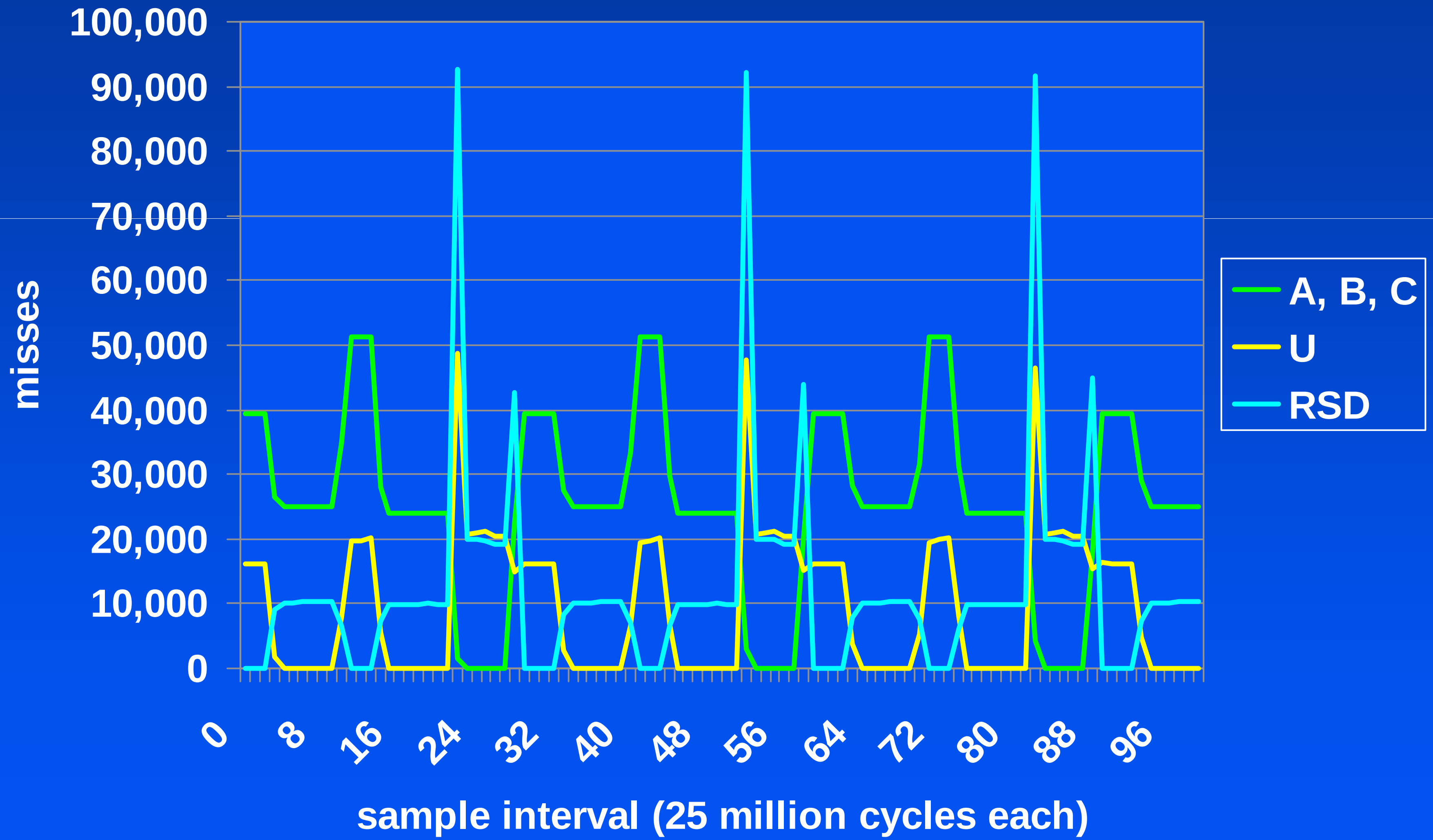
Search Time


billion cycles
0
2
4
6
8
10
12
tomcatv
swim
su2cor
mgrid
applu
compress
ijpeg
2-way
10-way

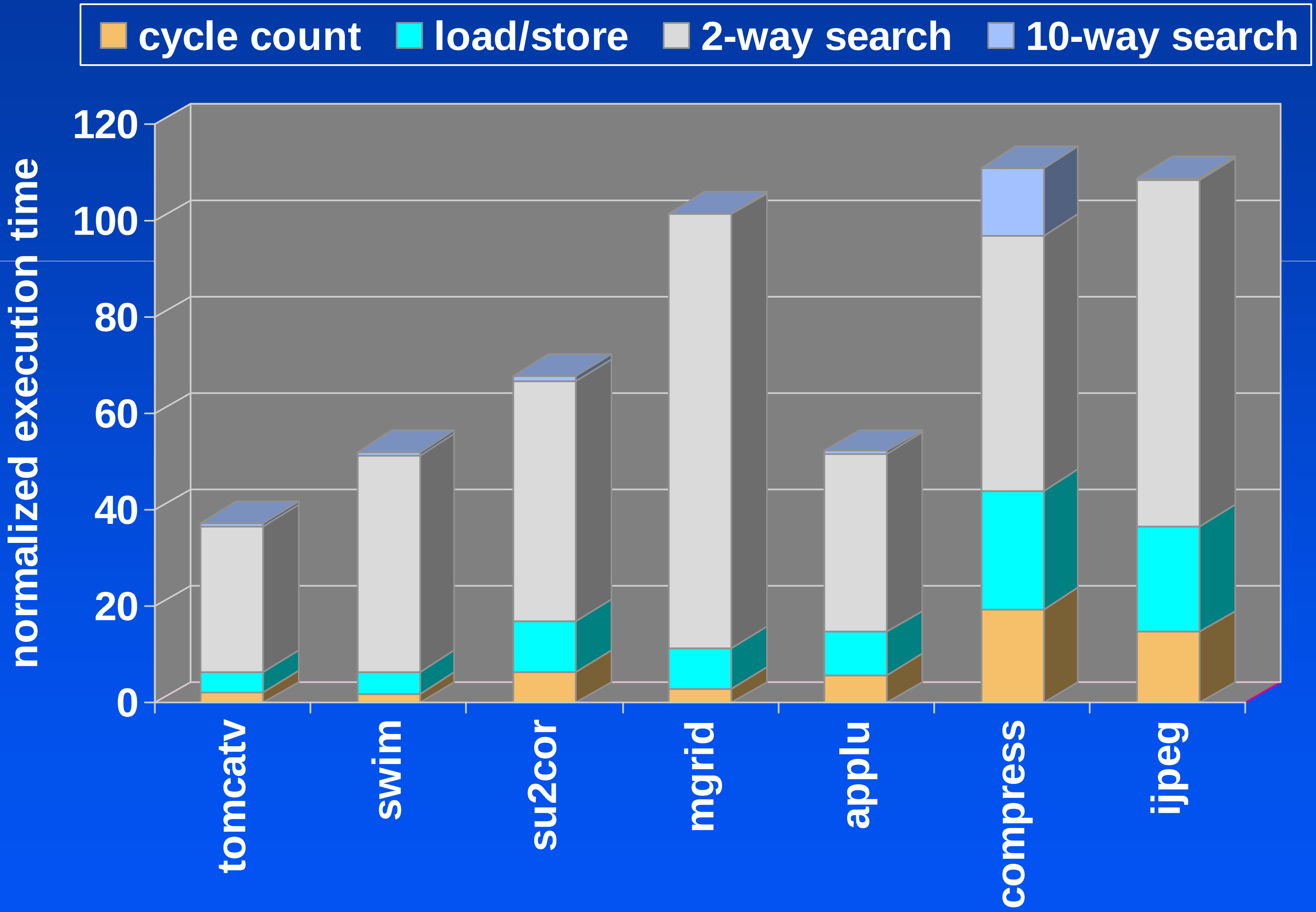
Misses vs. Time: Compress



Misses vs. Time: Applu



Instrumentation Costs



Implementing with the Dyninst API

- What we need
  - Determine instruction type (load/store)
  - Get effective address of load/store target
  - Basic block information
- What Dyninst will let us do
  - Allow application to initialize at normal speed
  - More selective instrumentation
  - Ability to add other axes (like Paradyn)
    - Show which function is causing cache problem

Conclusion

- Region miss information is useful
  - Automatic search can efficiently find arrays
- Simple algorithm has problems with...
  - Phases
  - Arrays spanning search regions
- More counters are more useful
  - 10-way search gets better results than 2-way
  - More counters doesn’t mean faster solution
- Cost of software instrumentation is high
  - Due to executing cache simulator every load/store
  - Much less instrumentation needed with hardware

Future Work

- Port simulator and search to Dyninst API
  - Need additional Dyninst features
- Asses value of counters in hardware
  - Simulate slowdown
- More sophisticated algorithms
  - Deal with phases
  - Better handling of dynamically allocated memory
    - Rearrange allocation for measurement
- More sophisticated instrumentation
  - Filter load/store instructions
  - Use hardware counters