Implementing Signatures for Transactional Memory

***Daniel Sanchez*, Luke Yen,**
Mark Hill, Karu Sankaralingam

University of Wisconsin-Madison

Executive summary

- Several TM systems use signatures:
  - ✓ Represent unbounded read/write sets in bounded state
  - ✘ False positives => Performance degradation
  - Use Bloom filters with bit-select hash functions
- We improve signature design:
  1. Use k Bloom filters in parallel, with 1 hash function each
     - Same performance for much less area (no multiported SRAM)
     - Applies to Bloom filters in other areas (LSQs…)
  2. Use high-quality hash functions (e.g. H_3)
     - Enables higher number of hash functions (4-8 vs. 2)
     - Up to 100% performance improvement in our benchmarks
  3. Beyond Bloom filters?
     - Cuckoo-Bloom: Hash table-Bloom filter hybrid (but complex)

Outline

- Introduction and motivation
- True Bloom signatures
- Parallel Bloom signatures
- Beyond Bloom signatures
- Area evaluation
- Performance evaluation
    - True vs. Parallel Bloom
    - Number and type of hash functions
- Conclusions

Support for Transactional Memory

- TM systems implement conflict detection
  - Find {**read-write**, **write-read, write-write**} conflicts among concurrent transactions
  - Need to track read/write sets (addresses read/written) of a transaction

- Signatures are data structures that
  - Represent an *arbitrarily large set* in *bounded* state
  - Approximate representation, with *false positives* but *no false negatives*

Signature Operation Example

Program:

xbegin

LD A

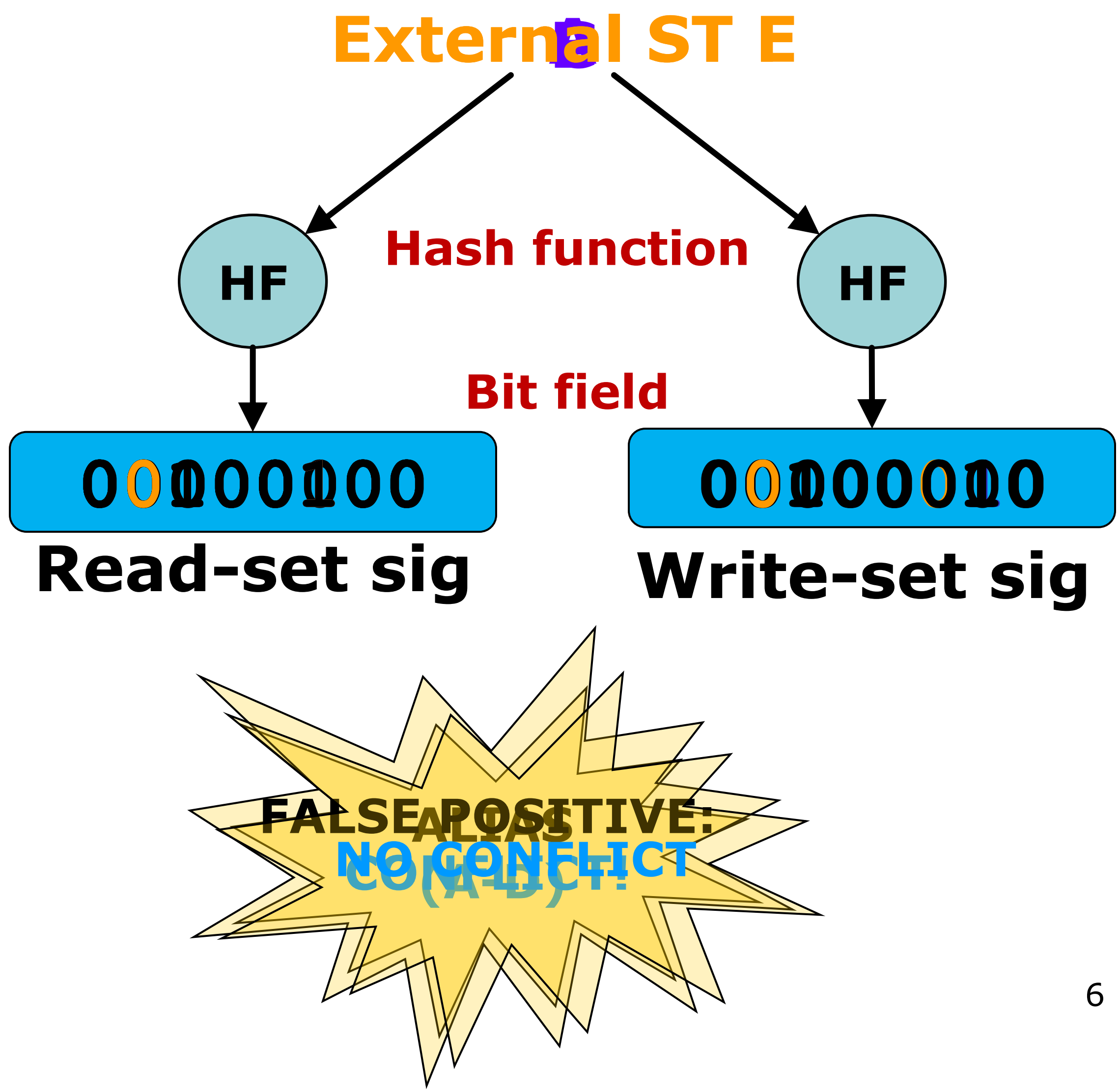
ST B

LD C

LD D

ST C

…



Motivation

- Hardware signatures concisely summarize read & write sets of transactions for conflict detection
  - ✓ Stores unbounded number of addresses
  - ✓ Correctness because no false negatives
  - ✓ Decouples conflict detection from L1 cache designs, eases virtualization
  - ✗ Lookups can indicate false positives, lead to unnecessary stalls/aborts and degrade performance
- Several transactional memory systems use signatures:
  - Illinois' Bulk [Ceze, ISCA06]
  - Wisconsin's LogTM-SE [Yen, HPCA07]
  - Stanford's SigTM [Minh, ISCA07]

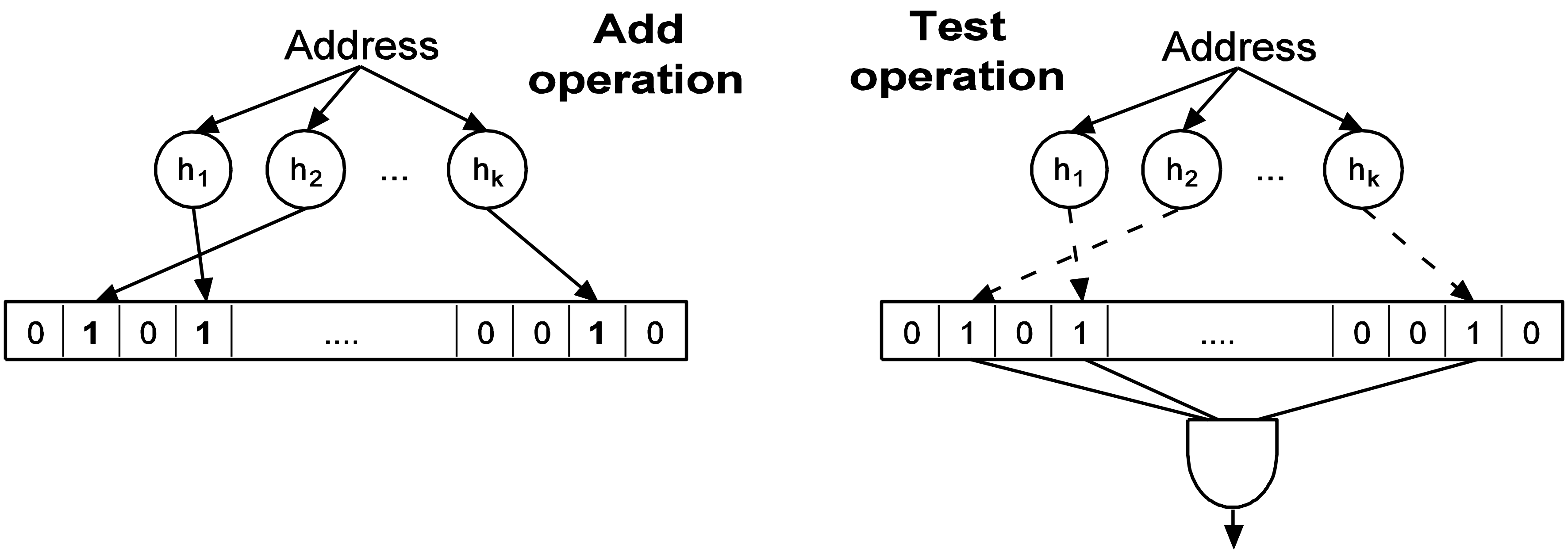
  - Implemented using (true/parallel) Bloom sigs [Bloom, CACM70]
- Signatures have applications beyond TM (scalable LSQs, early L2 miss detection)

Outline

- Introduction and motivation
- True Bloom signatures
- Parallel Bloom signatures
- Beyond Bloom signatures
- Area evaluation
- Performance evaluation
  - True vs. Parallel Bloom
  - Number and type of hash functions
- Conclusions

True Bloom signature - Design

- *Single* Bloom filter of *k* hash functions



True Bloom Signature - Design

- Probability of false positives (with independent, uniformly distributed memory accesses):

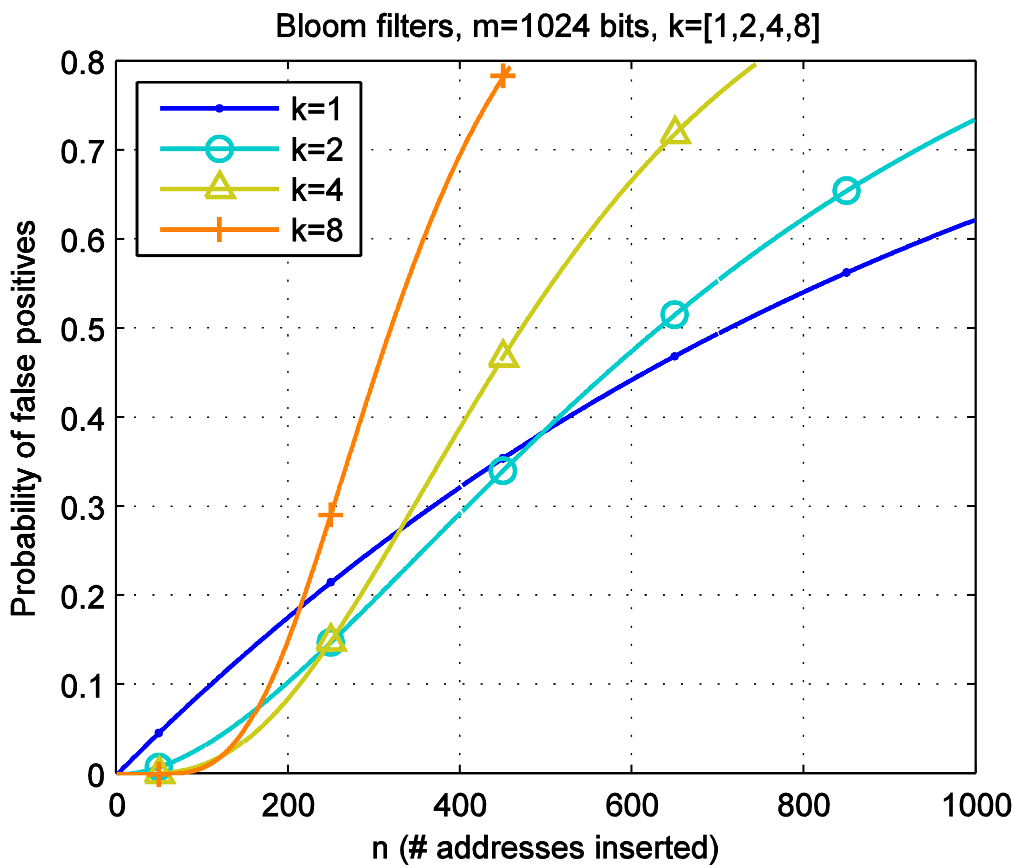
$$P_{FP}(n) = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k}$$

- Design dimensions
  - Size of the bit field (m)
  - Number of hash functions (k)
  - Type of hash functions

Larger is better

Examine in more detail

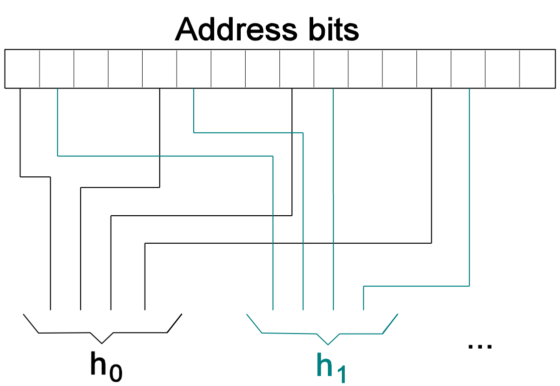
Number of hash functions



- High # elements => Fewer hash functions better
- Small # elements => More hash functions better

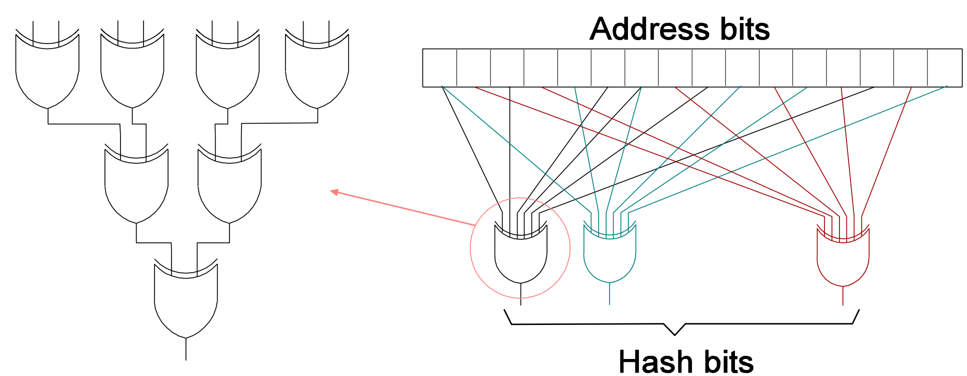
Types of hash functions

- Addresses not independent or uniformly distributed
- But can generate *almost* uniformly distributed and uncorrelated hashes with good hash functions
- Hash functions considered:



Bit-selection

(inexpensive, low quality)

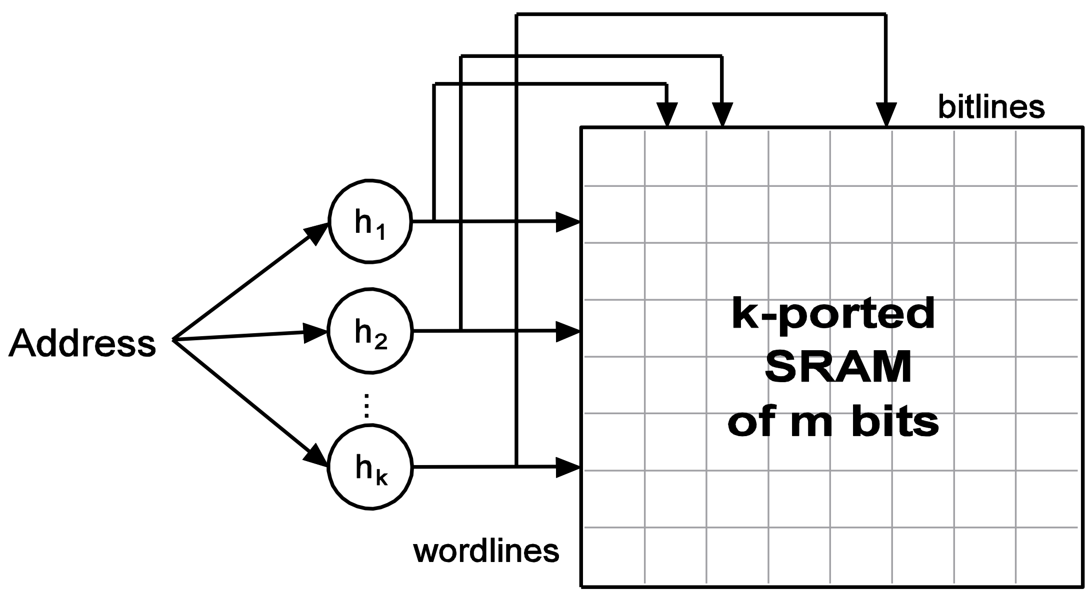


H_3 [Carter, CSS77]

(moderate, high quality)

True Bloom Signature – Implementation

- Divide bit field in words, store in small SRAM
  - Insert: Raise wordline, drive appropriate bitline to 1, leave rest floating
  - Test: Raise wordline, check value at bitline
- k hash functions => k read, k write ports



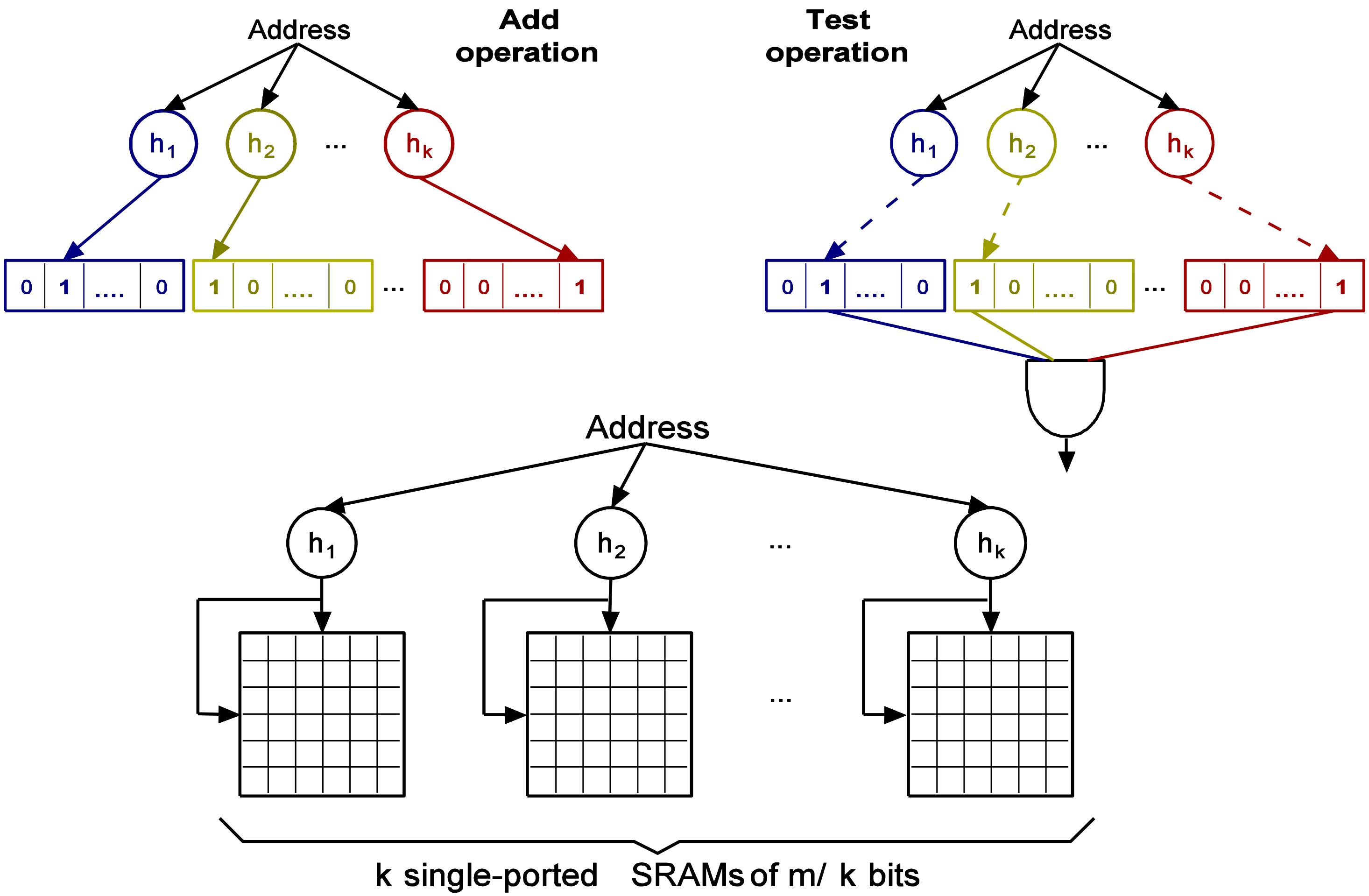
Problem
Size of SRAM cell increases quadratically with # ports!

Outline

- Introduction and motivation
- True Bloom signatures
- Parallel Bloom signatures
- Beyond Bloom signatures
- Area evaluation
- Performance evaluation
  - True vs. Parallel Bloom
  - Number and type of hash functions
- Conclusions

Parallel Bloom Signatures

- To avoid multiported memories, we can use k Bloom filters of size m/k in parallel



Parallel Bloom signatures - Design

- Probability of false positives:
  - True: $$P_{FP}(n) = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^k \cong \left(1 - e^{\frac{-nk}{m}}\right)^k$$
  - Parallel: $$P_{FP}(n) = \left(1 - \left(1 - \frac{1}{m/k}\right)^{n}\right)^k \cong \left(1 - e^{\frac{-nk}{m}}\right)^k$$

$$\left(\text{if } \frac{k}{m} << 1\right)$$

- Same performance as true Bloom!!
- Higher area efficiency

Outline

- Introduction and motivation
- True Bloom signatures
- Parallel Bloom signatures
- Beyond Bloom signatures
- Area evaluation
- Performance evaluation
  - True vs. Parallel Bloom
  - Number and type of hash functions
- Conclusions

Beyond Bloom Signatures

- Bloom filters not space optimal => Opportunity for increased efficiency
  - Hash tables are, but limited insertions [Carter,CSS78]
- Our approach: New Cuckoo-Bloom signature
  - Hash table (using Cuckoo hashing) to represent sets when few insertions
  - Progressively morph the table into a Bloom filter to allow an unbounded number of insertions
  - Higher space efficiency, but higher complexity
  - In simulations, performance similar to good Bloom signatures
  - See paper for details

Outline

- Introduction and motivation
- True Bloom signatures
- Parallel Bloom signatures
- Beyond Bloom signatures
- Area evaluation
- Performance evaluation
  - True vs. Parallel Bloom
  - Number and type of hash functions
- Conclusions

Area evaluation

- SRAM: Area estimations using CACTI
  - 4Kbit signature, 65nm

	k=1	k=2	k=4
True Bloom	0.031 mm^2	0.113 mm^2	0.279 mm^2
Parallel Bloom	0.031 mm^2	0.032 mm^2	0.035 mm^2
True/Parallel	1.0	3.5	8.0

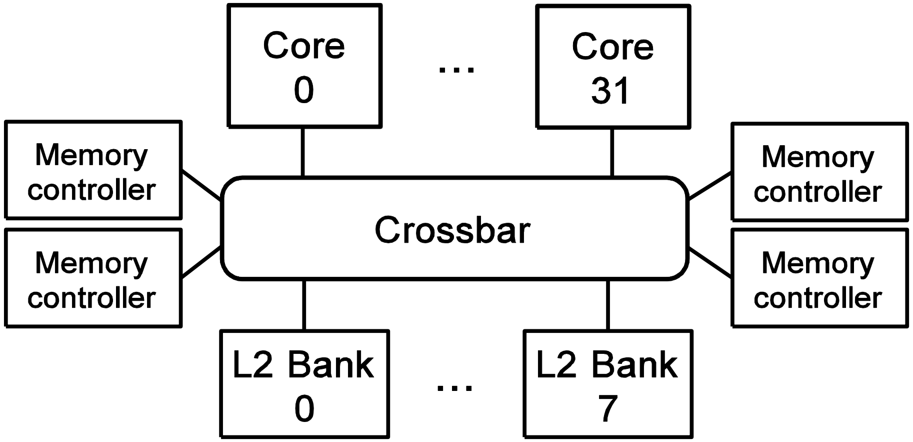
- 8x area savings for four hash functions!
- Hash functions:
  - Bit selection has negligible extra cost
  - Four hardwired H_3 require ≈25% of SRAM area

Outline

- Introduction and motivation
- True Bloom signatures
- Parallel Bloom signatures
- Beyond Bloom signatures
- Area evaluation
- Performance evaluation
  - True vs. Parallel Bloom
  - Number and type of hash functions
- Conclusions

Performance evaluation

- Using LogTM-SE
- System organization:
  - 32 in-order single-issue cores
  - 32KB, 4-way private L1s, 8MB, 8-way shared L2
  - High-bandwidth crossbar, snooping MESI protocol
  - Signature checks are *broadcast*
  - Base conflict resolution protocol with *write-set prediction* [Bobba, ISCA07]



Methodology

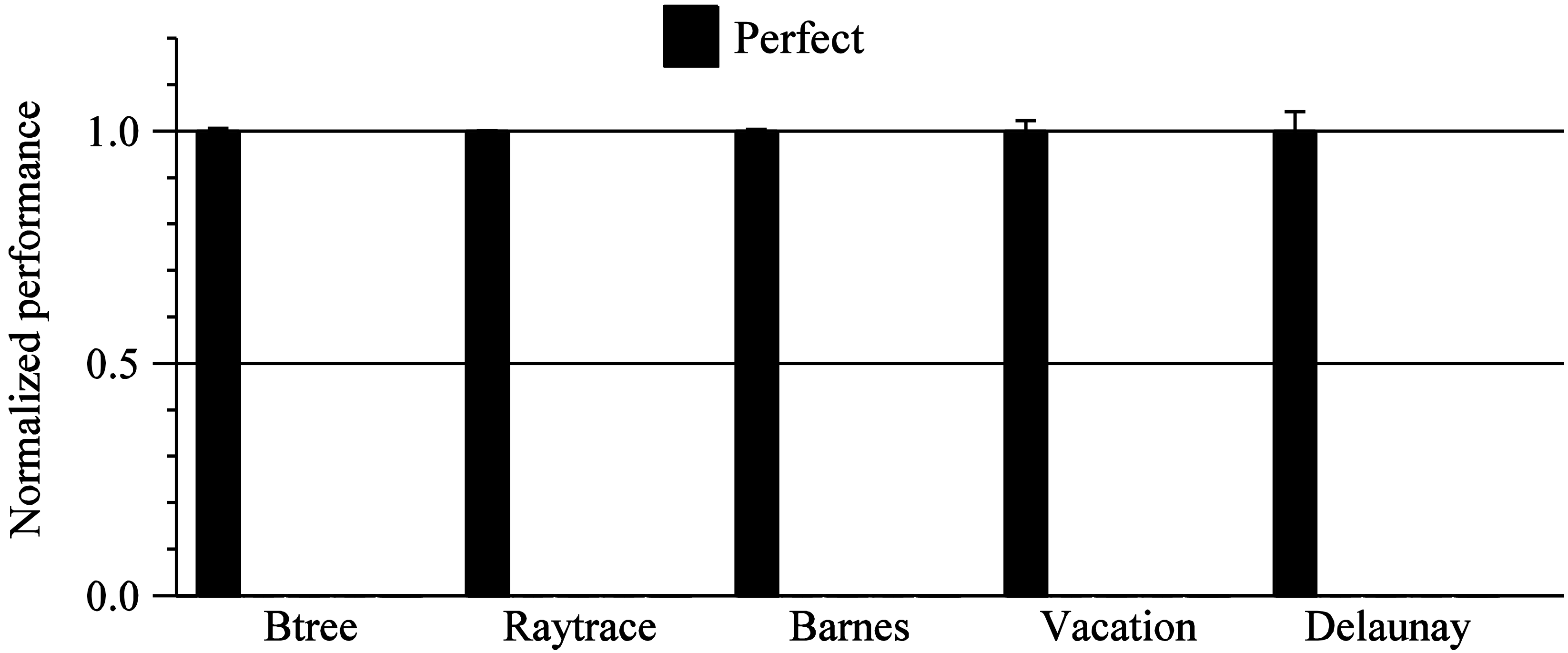
- Virtutech Simics full-system simulation
- Wisconsin GEMS 2.0 timing modules:

  www.cs.wisc.edu/gems

- SPARC ISA, running unmodified Solaris

- Benchmarks:
  - Microbenchmark: Btree
  - SPLASH-2: Raytrace, Barnes [Woo, ISCA95]
  - STAMP: Vacation, Delaunay [Minh, ISCA07]

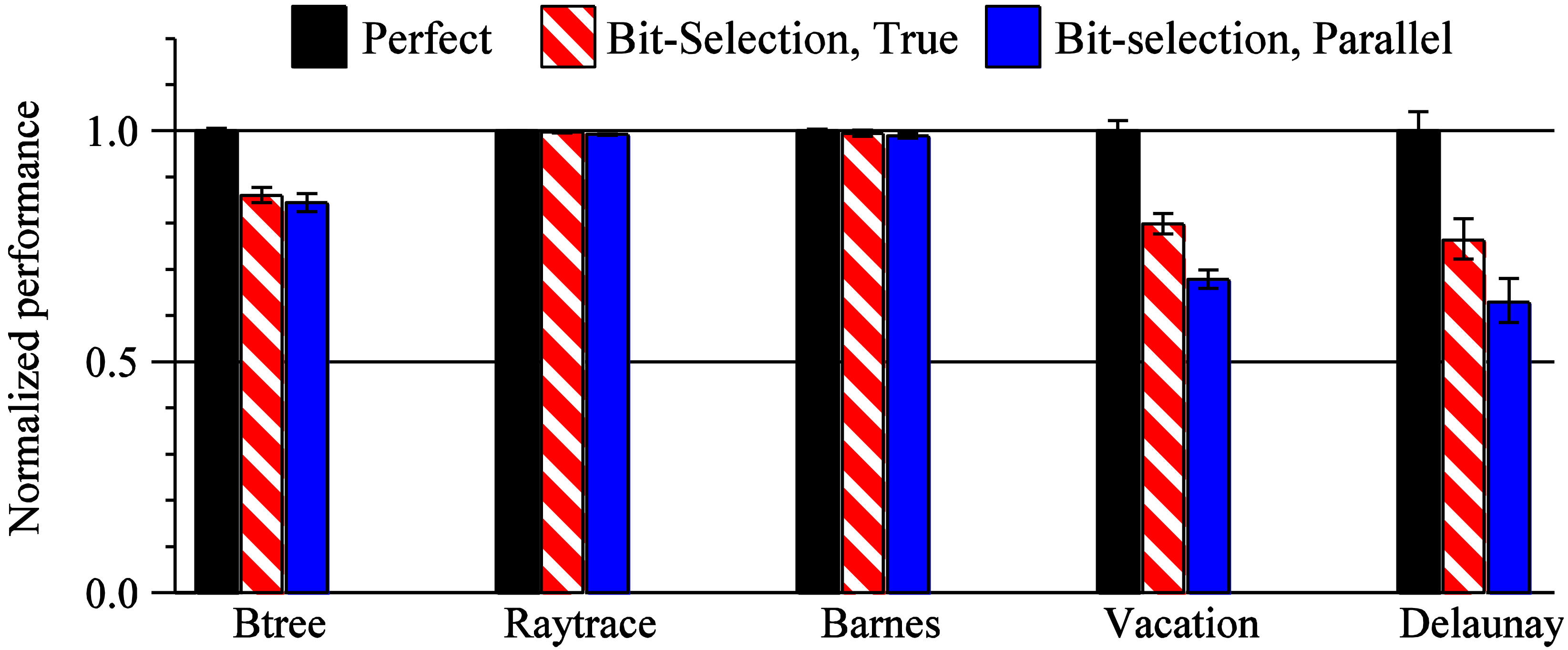
True Versus Parallel Bloom



2048-bit Bloom Signatures, 4 hash functions

- Performance results normalized to *un-implementable* Perfect signatures
- Higher bars are better

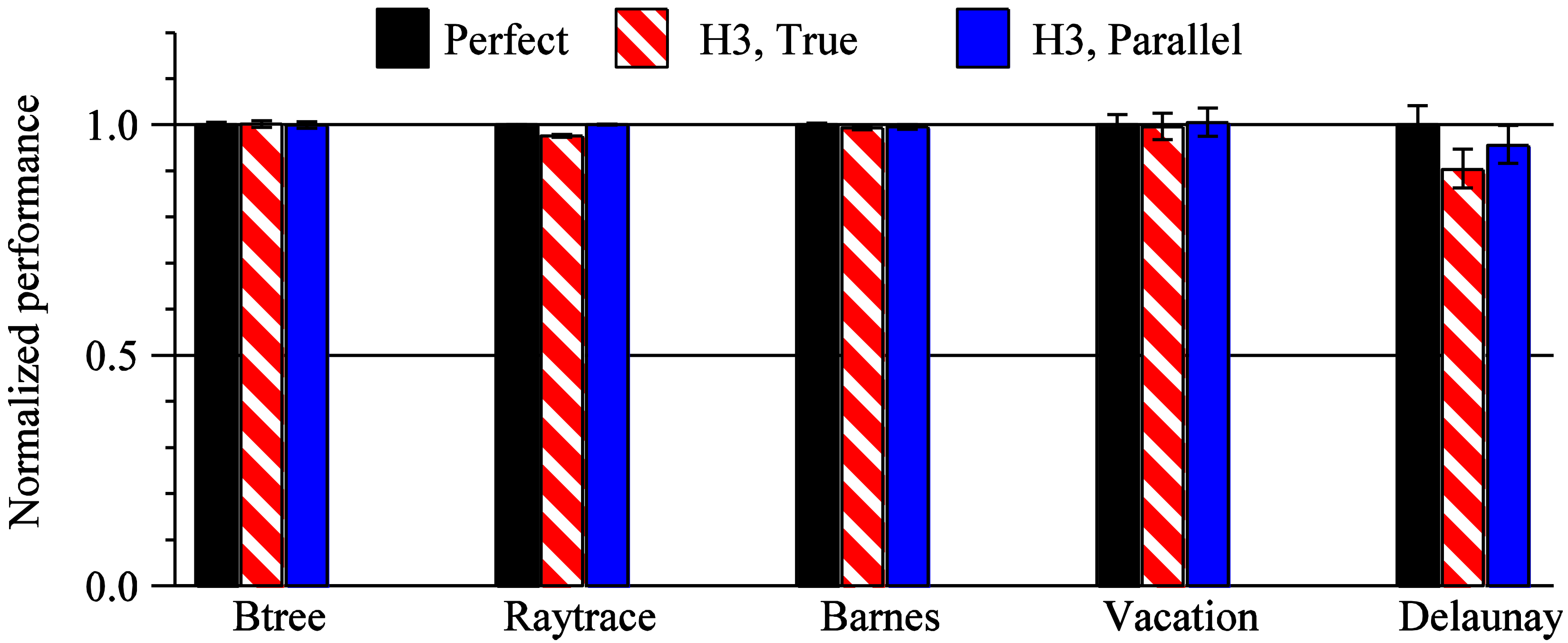
True Versus Parallel Bloom



2048-bit Bloom Signatures, 4 hash functions

- For Bit-selection, True & Parallel Bloom perform similarly
- Larger differences for Vacation, Delaunay – larger, more frequent transactions

True Versus Parallel Bloom



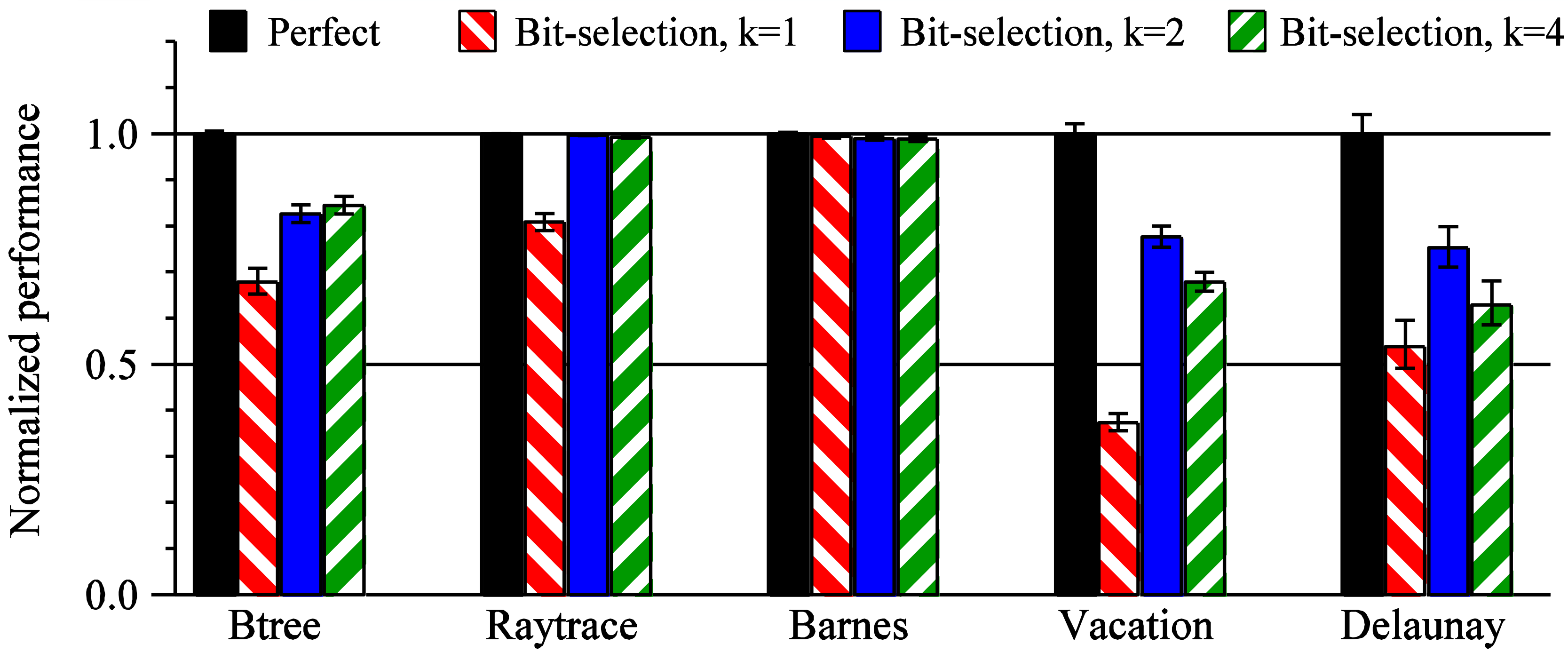
2048-bit Bloom Signatures, 4 hash functions

- For H_3, True & Parallel Bloom signatures also perform similarly (less difference than bit-select)
- **Implication 1**: Parallel Bloom preferred over True Bloom: similar performance, simpler implementation

Outline

- Introduction and motivation
- True Bloom signatures
- Parallel Bloom signatures
- Beyond Bloom signatures
- Area evaluation
- Performance evaluation
  - True vs. Parallel Bloom
  - Number and type of hash functions
- Conclusions

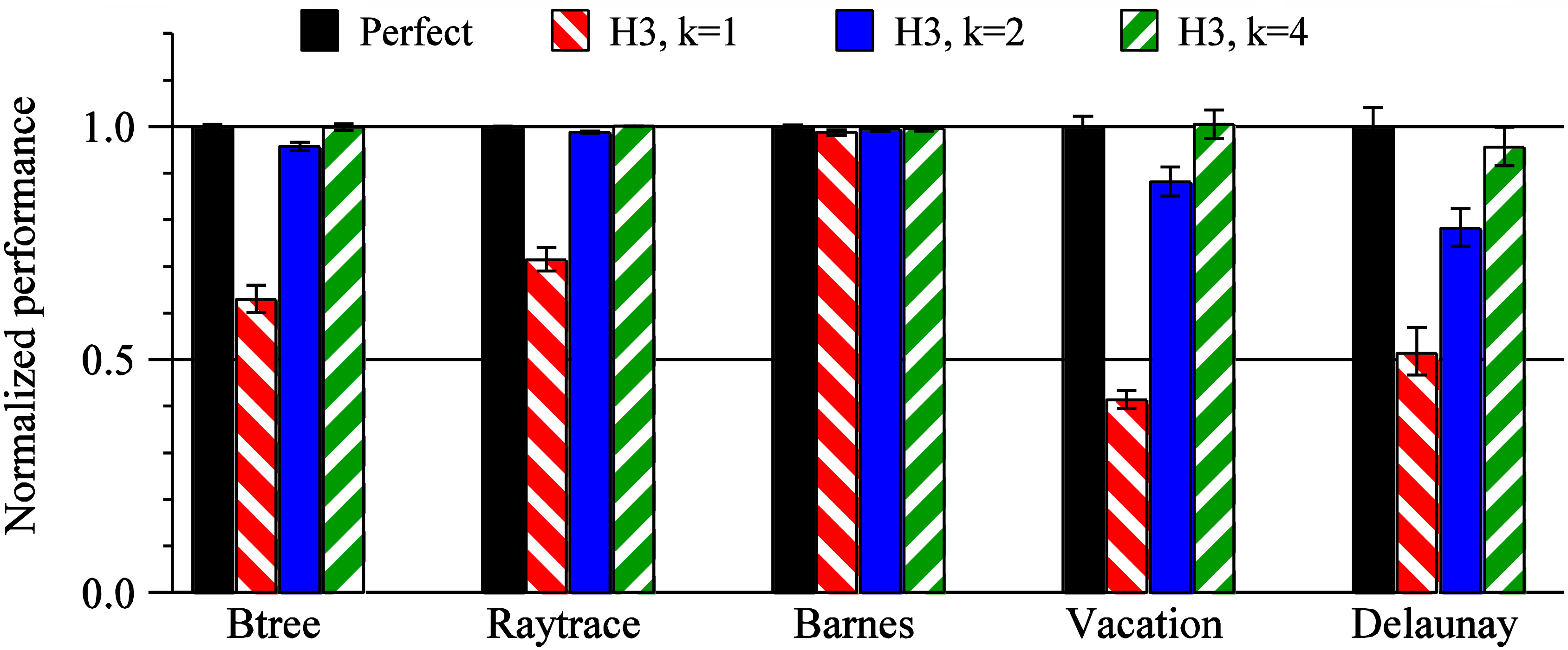
Number of Hash Functions (1/2)



2048-bit Parallel Bloom Signatures

- **Implication 2a**: For low-quality hashes (Bit-selection), increasing number of hash functions beyond 2 does not help
- Bits set are not uniformly distributed, correlated

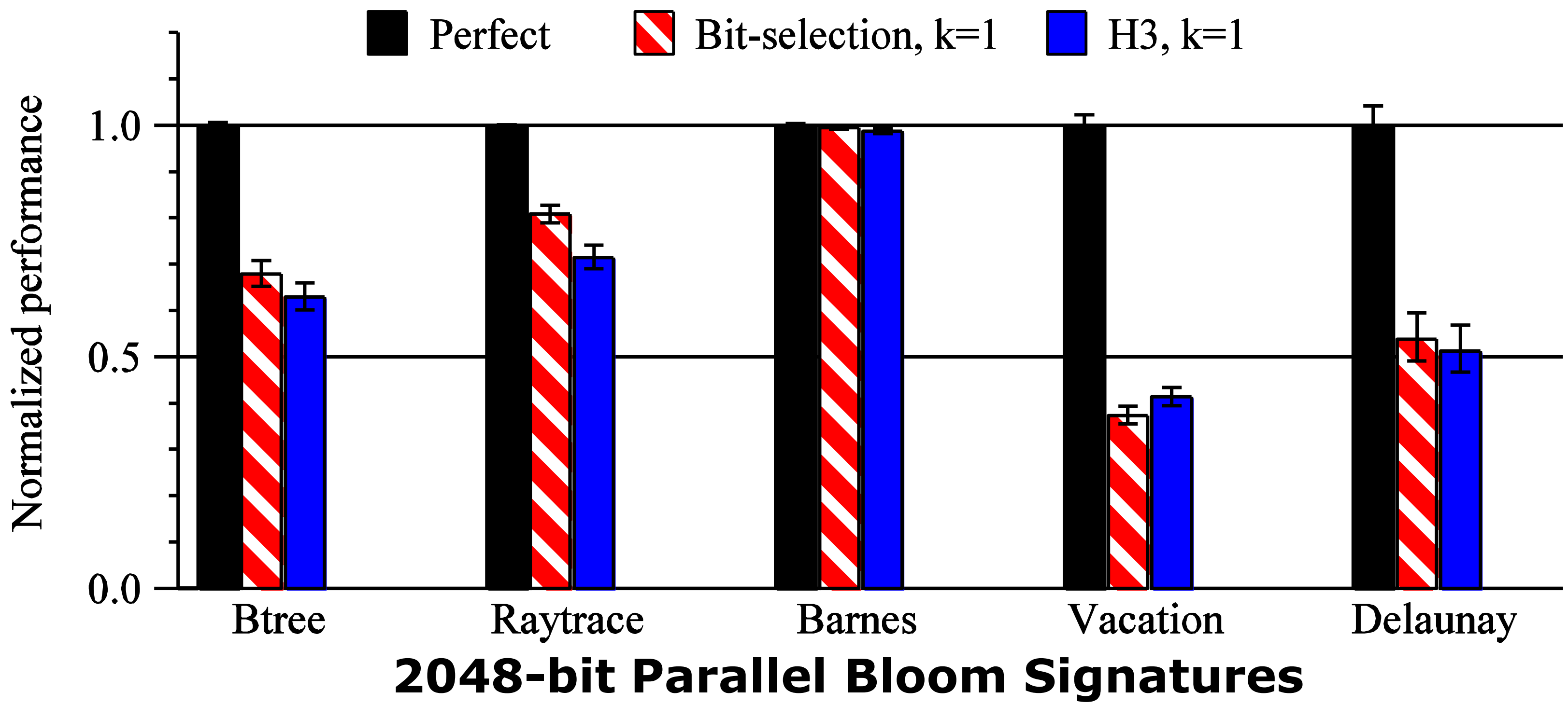
Number of Hash Functions (2/2)



2048-bit Parallel Bloom Signatures

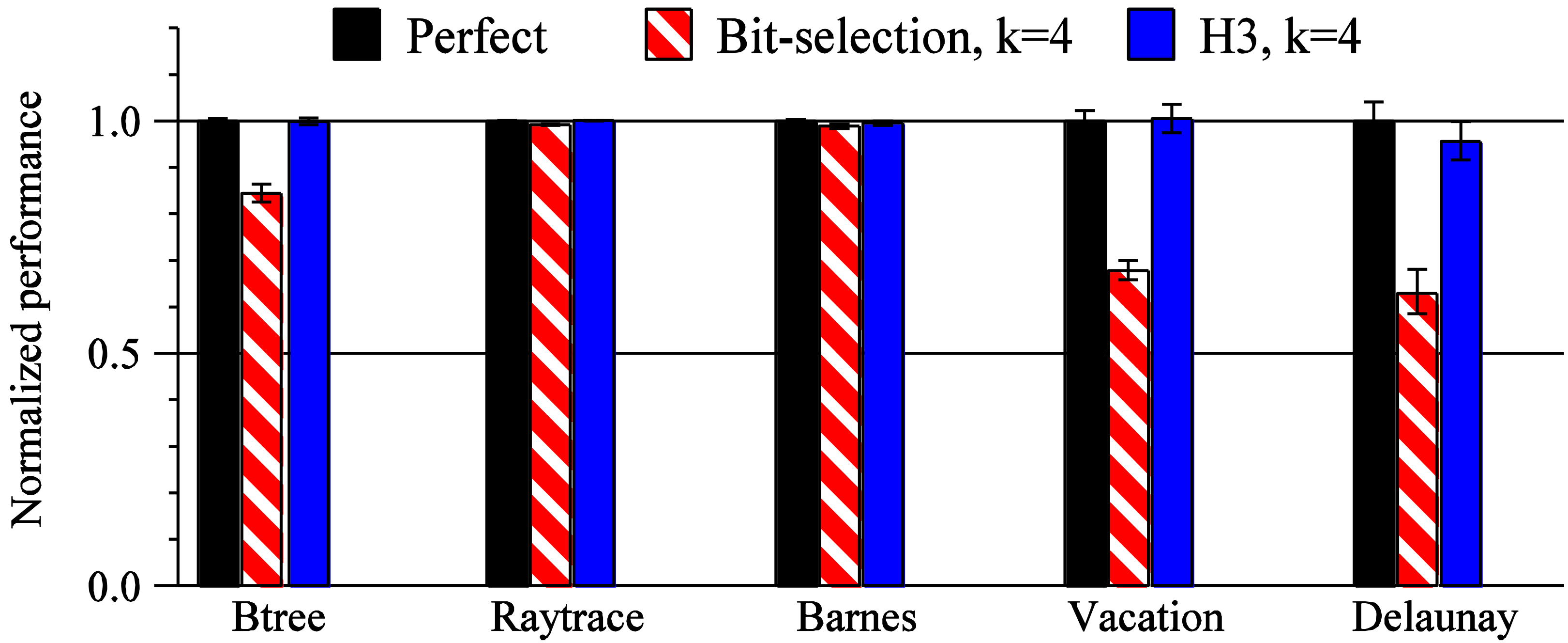
- For high-quality hashes (H_3), increasing number of hash functions improves performance for most benchmarks
- Even k=8 works as well (not shown)

Type of Hash Functions (1/2)



- 1 hash function => bit-selection and H_3 achieve similar performance
- Similar results for 2 hash functions

Type of Hash Functions (2/2)



2048-bit Parallel Bloom Signatures

- **Implication 2b**: For 4 and more hash functions, high-quality hashes (H_3) perform much better than low-quality hashes (bit-selection)

Conclusions

- Detailed design space exploration of Bloom signatures
  - Use Parallel Bloom instead of True Bloom
    - Same performance for much less area
  - Use high-quality hash functions (e.g. H_3)
    - Enables higher number of hash functions (4+ vs. 2)
    - Up to 100% performance improvement in our benchmarks
- Alternatives to Bloom signatures exist
  - Complexity vs. space efficiency tradeoff
  - Cuckoo-Bloom: Hash table-Bloom filter hybrid (but complex)
  - Room for future work
- Applicability of findings beyond TM

Thank you

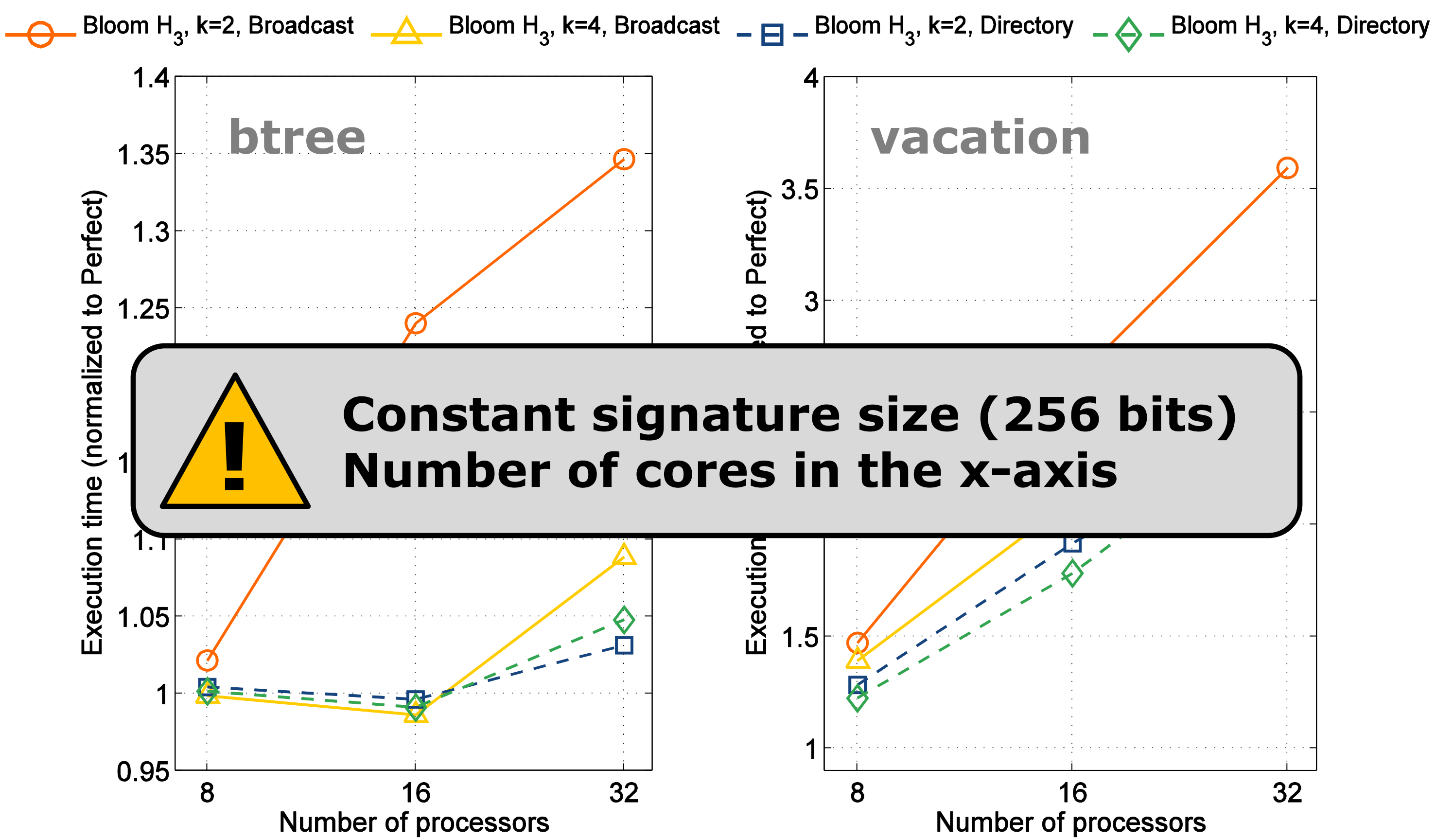
for your attention

Questions?

Backup – Why same performance?

- True Bloom => Larger hash functions, but uncertain who wrote what
- Parallel Bloom => Smaller hash functions, but certain who wrote what
- These two effect compensate
- Example:
  - Only bits {6,12} set in 16-bit 2 HF True Bloom => Candidates are (H1,H2)=(6,12) or (12,6)
  - Only bits {6,12} set in 16-bit 2 HF Parallel Bloom => Only candidate is (H1,H2) = (6,4), but each HF has 1 bit less

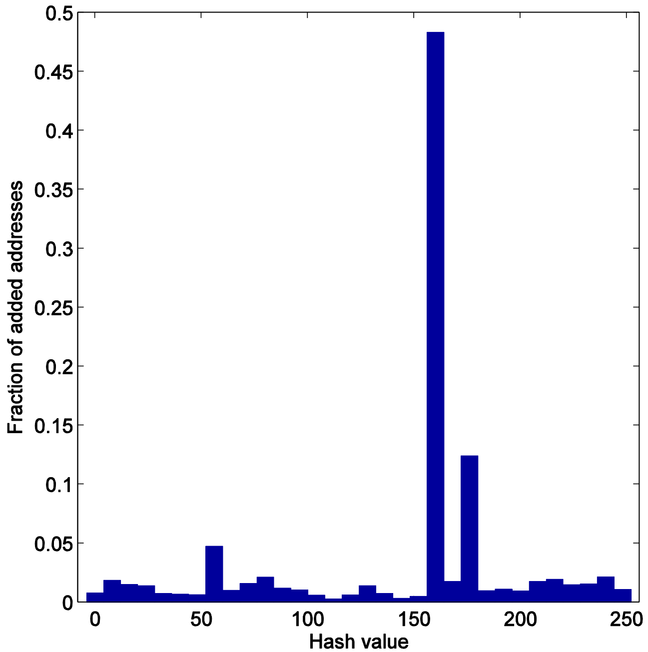
Backup - Number of cores & directory



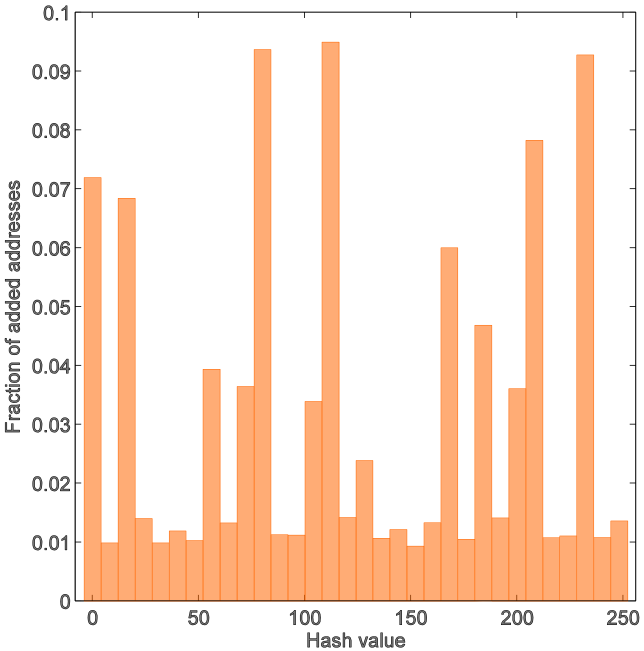
- Pressure increases with #cores
- Directory helps, but still requires to scale the signatures with the number of cores

Backup – Hash function analysis

- Hash value distributions for btree, 512-bit parallel Bloom with 2 hash functions



bit-selection

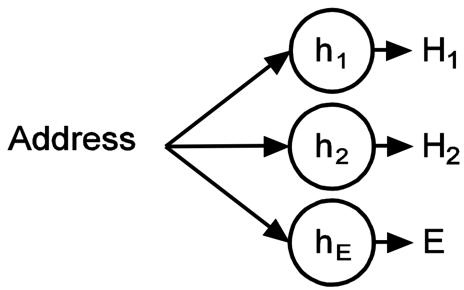


fixed H_3

Backup - Conflict resolution in LogTM-SE

- Base: Stall requester by default, abort if it is stalling an older Tx and stalled by an older Tx
- Pathologies:
  - DuelingUpgrades: Two Txs try to read-modify-update same block concurrently -> younger aborts
  - StarvingWriter: Difficult for a Tx to write to a widely shared block
  - FutileStall: Tx stalls waiting for other that later aborts
- Solutions:
  - Write-set prediction: Predict read-modify-updates, get exclusive access directly (targets DuelingUpgrades)
  - Hybrid conflict resolution: Older writer aborts younger readers (targets StarvingWriter, FutileStall)

Backup – Cuckoo-Bloom signatures



	bucket 0			bucket 1			
WC	H_1	H_2	E	H_1	H_2	E	
0				0	4	562	set 0
0							
0				6	2	453	
0	3	3	156	3	5	942	
0	4	0	244	5	4	671	
0				2	5	027	
1							
0	7	1	391	7	7	234	set 7