Design and Implementation of Signatures in Transactional Memory Systems

Daniel Sanchez

August 2007

University of Wisconsin-Madison
Outline

- Introduction and motivation
- Bloom filters
- Bloom signatures
- Area & performance evaluation
- Influence of system parameters
- Novel signature schemes (brief overview)
- Conclusions
Signature-based conflict detection

- Signatures:
  - Represent an *arbitrarily large set* of elements in *bounded* amount of state (bits)
  - Approximate representation, with *false positives* but *no false negatives*

- Signature-based CD: Use signatures to track read/write sets of a transaction
  - Pros:
    - Transactions can be *unbounded* in size
    - Independence from caches, eases virtualization
  - Cons:
    - False conflicts -> Performance degradation
Motivation of this study

- Signatures play an important role in TM performance. Poor signatures cause lots of unnecessary stalls and aborts.
- Signatures can take significant amount of area
  - Can we find area-efficient implementations?
  - Adoption of TM much easier if the area requirements are small!
- Signature design space exploration incomplete in other TM proposals
Previously proposed TM signatures are either true Bloom (1 filter, k hash functions) or parallel Bloom (k filters, 1 hash function each).

- Performance-wise, True Bloom = Parallel Bloom
- Parallel Bloom about 8x more area-efficient

New Bloom signature designs that double the performance and are more robust

Pressure on signatures greatly increases with the number of cores; directory can help

Three novel signature designs
Outline

- Introduction and motivation
- Bloom filters
- Bloom signatures
- Area & performance evaluation
- Influence of system parameters
- Novel signature schemes (brief overview)
- Conclusions
Bloom filters

- **Hash functions**: $h_1$ and $h_2$
- **Address**: Positioned between $h_1$ and $h_2$
- **Hash values**: $\{0, \ldots, m-1\}$
- **Bit field (m bits)**: A sequence of bits indicating membership: 000000000000000000000000
Bloom filters

Add 0x2a83ff00

h₁

3

0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0

h₂

8
Bloom filters

Add 0x2a8ab3f4

h₁

12

h₂

2

0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0
Bloom filters

Test 0x2a8a83f4

\[ h_1 \quad h_2 \]

\[ 10 \quad 2 \]

\[0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0]\n
False
Bloom filters

Test 0x2a83ff00

h₁
3
0 0 1
1
h₂
8
0 0 0 0 0 1
1 0 0 0 1
0 0 0 0

True
Bloom filters

Test 0xff83ff48

h₁
2

h₂
8

0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0

True (false positive!)
Outline

- Introduction and motivation
- Bloom filters
- Bloom signatures
  - True Bloom signatures
  - Parallel Bloom signatures

Area & performance evaluation
- Influence of system parameters
- Novel signature schemes (brief overview)
- Conclusions
True Bloom signature - Design

- True Bloom signature = Signature implemented with a single Bloom filter
- Easy insertions and tests for membership
- Probability of false positives:

\[
P_{FP}(n) = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^k \approx \left(1 - e^{-\frac{nk}{m}}\right)^k \quad \text{(if } \frac{k}{m} \ll 1)\]

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

- Addresses neither independent nor uniformly distributed (key assumptions to derive $P_{FP}(n)$)
- But can generate hash values that are almost uniformly distributed and uncorrelated with good (universal/almost universal) hash functions
- Hash functions considered:

**Bit-selection**

- Inexpensive, low quality

**H_3**

- Moderate, higher quality
True Bloom signature – Implementation

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

Problem
Size of SRAM cell increases quadratically with # ports!
- Use $k$ Bloom filters of size $m/k$, with independent hash functions

- Probability of false positives:

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

Same as true Bloom!
- Highly area-efficient SRAMs
- Same performance as true Bloom! (in theory)
Outline

- Introduction and motivation
- Bloom filters
- Bloom signatures
- Area & performance evaluation
  - Area evaluation
  - True vs. Parallel Bloom in practice
  - Type of hash functions
  - Variability in hash functions
- Influence of system parameters
- Novel signature schemes (brief overview)
- Conclusions
SRAM: Area estimations using CACTI

- 4Kbit signature, 65nm

<table>
<thead>
<tr>
<th></th>
<th>$k=1$</th>
<th>$k=2$</th>
<th>$k=4$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>True Bloom</strong></td>
<td>0.031 mm$^2$</td>
<td>0.113 mm$^2$</td>
<td>0.279 mm$^2$</td>
</tr>
<tr>
<td><strong>Parallel Bloom</strong></td>
<td>0.031 mm$^2$</td>
<td>0.032 mm$^2$</td>
<td>0.035 mm$^2$</td>
</tr>
</tbody>
</table>

- **8x area savings** for four hash functions!

- Hash functions:
  - Bit selection has no extra cost
  - Four hardwired $H_3$ require $\approx 25\%$ of SRAM area
- **System organization:**
  - 32 in-order single-issue cores
  - Private split 32KB, 4-way L1 caches
  - Shared unified 8MB, 8-way L2 cache
  - High-bandwidth crossbar
  - Signature checks are *broadcast* (no directory)
  - Base conflict resolution protocol with *write-set prediction*

- **Benchmarks:** btree, raytrace, vacation
  - barnes, delaunay, and full set of results in report
True vs. Parallel Bloom signatures

- Bottom line: True $\approx$ parallel if we use good enough hash functions

Graph format

- Solid lines = Parallel Bloom
- Dashed lines = True Bloom
- Different colors = Different number of hash functions
- Execution times are always normalized
- $H_3$ clearly outperforms bit-selection for $k \geq 2$
- Only 2Kbit signatures with $4+ H_3$ functions cause no degradation over all the benchmarks
The benefits of variability

- **Variable H₃**: Reconfigure hash functions after each commit/abort
  - Constant aliases -> Transient aliases
  - Adds robustness
The benefits of variability

- **Variable $H_3$:** Reconfigure hash functions after each commit/abort
  - Constant aliases $\rightarrow$ Transient aliases
  - Adds robustness
Conclusions on Bloom signature evaluation

- Parallel Bloom enables high number of hash functions “for free”
- Type of hash functions used matters a lot (but was neglected in previous analysis)
- Variability adds robustness
- Should use:
  - About four $H_3$ or other high quality hash functions
  - Variability if the TM system allows it
  - Size... depends on system configuration
Outline

- Introduction and motivation
- Bloom filters
- Bloom signatures
- Area & performance evaluation
- Influence of system parameters
  - Number of cores
  - Conflict resolution protocol
- Novel signature schemes (brief overview)
- Conclusions
- Pressure increases with #cores
- Directory helps, but still requires to scale the signatures with the number of cores
Effect of conflict resolution protocol

- Protocol choice fairly orthogonal to signatures
- False conflicts *boost* existing pathologies in btree/raytrace -> Hybrid policy helps even more than with perfect signatures

Constant signature type ($H_3$, $k=2$)
Execution times not normalized
Overview of novel signature schemes

- Cuckoo-Bloom signatures
  - Adapts cuckoo hashing for HW implementation
  - Keeps a hash table for small sets, morphs into a Bloom filter dynamically as the size grows
  - Significant complexity, performance advantage not clear

- Hash-Bloom signatures
  - Simpler hash-table based approach
  - Morphs to a Bloom filter more gradually than Cuckoo-Bloom
  - Outperforms Bloom signatures for both small and write sets, in theory and practice

- Adaptive Bloom signatures
  - Bloom signatures + set size predictors + scheme to select the best number of hash functions
Conclusions

- Bloom signatures should always be implemented as parallel Bloom
  - with $\approx 4$ good hash functions, some variability if allowed
  - Overall good performance, simple/inexpensive HW
- Increasing #cores makes signatures more critical
  - Hinders scalability!
  - Using directory helps, but doesn’t solve
- Hybrid conflict resolution helps with signatures
- There are alternative schemes that outperform Bloom signatures
Thanks for your attention

Any questions?
Backup – Hash function analysis

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

<table>
<thead>
<tr>
<th>WC</th>
<th>$H_1$</th>
<th>$H_2$</th>
<th>$E$</th>
<th>$H_1$</th>
<th>$H_2$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>562</td>
<td>0</td>
<td>4</td>
<td>562</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>2</td>
<td>453</td>
<td>3</td>
<td>5</td>
<td>942</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>156</td>
<td>3</td>
<td>5</td>
<td>942</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>244</td>
<td>5</td>
<td>4</td>
<td>671</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>244</td>
<td>5</td>
<td>4</td>
<td>671</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>027</td>
<td>7</td>
<td>7</td>
<td>234</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>1</td>
<td>391</td>
<td>7</td>
<td>7</td>
<td>234</td>
</tr>
</tbody>
</table>

**Graphs:**
- **btree**
- **vacation**
Backup – Hash-Bloom signatures

![Diagram of Hash-Bloom signatures](image)

Bloom and Parallel Hash-Bloom signatures of m=1Kbit

 Execution time (normalized to Perfect)

Signature size

n (# addresses inserted)

Probability of false positives

1. Bloom, k=1
2. Bloom, k=2
3. Bloom, k=4
4. Bloom, k=8
5. Hash-Bloom 4b/row
6. Hash-Bloom 8b/row

vacation
Backup – Adaptive Bloom signatures

Prediction table

<table>
<thead>
<tr>
<th>PC</th>
<th>RS size</th>
<th>WS size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12f74</td>
<td>25</td>
<td>7</td>
</tr>
<tr>
<td>0x13410</td>
<td>138</td>
<td>2</td>
</tr>
<tr>
<td>0x104d8</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Update logic

Hash function selection logic

Address

\[ h_s \rightarrow h_1 \rightarrow h_2 \rightarrow \ldots \rightarrow h_k \]

0 1 .... 0

\[ 1 \rightarrow \quad 1 \rightarrow \quad 0 \]

\[ 0 0 .... 0 \]

Execution time (normalized to Perfect)

- raytrace

- vacation