Hash tables suffer from
1. Poor core utilization
   - Main memory stall
   - LLC stall
   - L2 stall
   - L1 stall
   - Function unit stall
   - Fetch/Decode stall
   - Wrong path execution
   - Issued micro ops
   - Data-dependent branches with high misprediction rate & penalty
   - Poor backend utilization due to too many µops per operation
2. Poor spatial locality
   - Hash tables tend to spread keys evenly

**Background**

**HTA table format**

<table>
<thead>
<tr>
<th>Key 0</th>
<th>Key 1</th>
<th>Value 0</th>
<th>Value 1</th>
<th>Unused</th>
</tr>
</thead>
<tbody>
<tr>
<td>128b</td>
<td>128b</td>
<td>64b</td>
<td>64b</td>
<td>128b</td>
</tr>
</tbody>
</table>

Leaves overflows to software hash table
- Allows parallel probing after one memory access
- Avoids hard-to-predict branches
- Enables hardware acceleration

**HTA extensions**

- hta_lookup
- hta_update / hta_swap
- hta_delete

Adopt branch semantics
- Easy to predict
- Exploits branch prediction

**Flat-HTA**

- Adds simple changes to cores
- Reduces runtime overheads
- Improves hash-table-intensive applications by up to 2x
- Enables memoization of small code regions

**Hierarchical-HTA**

- Changes cores and caches
- Improves spatial locality

---

**Evaluation**

**Flat-HTA speedups**

<table>
<thead>
<tr>
<th>Application</th>
<th>Baseline</th>
<th>HTA-SW</th>
<th>Flat-HTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfc</td>
<td>1</td>
<td>1.5</td>
<td>1.7</td>
</tr>
<tr>
<td>hzw</td>
<td>1</td>
<td>1.8</td>
<td>2.0</td>
</tr>
<tr>
<td>hashjoin</td>
<td>1</td>
<td>1.4</td>
<td>1.6</td>
</tr>
<tr>
<td>ycsb-read</td>
<td>1</td>
<td>1.2</td>
<td>1.4</td>
</tr>
<tr>
<td>ycsb-write</td>
<td>1</td>
<td>1.2</td>
<td>1.4</td>
</tr>
</tbody>
</table>

**Flat-HTA on multithreaded apps**

<table>
<thead>
<tr>
<th>Application</th>
<th>Baseline</th>
<th>Flat-HTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ycsb-read</td>
<td>1</td>
<td>1.2</td>
</tr>
<tr>
<td>ycsb-write</td>
<td>1</td>
<td>1.4</td>
</tr>
</tbody>
</table>

**Hierarchical-HTA**

<table>
<thead>
<tr>
<th>Application</th>
<th>Baseline</th>
<th>HTA</th>
<th>Software Memoization</th>
</tr>
</thead>
<tbody>
<tr>
<td>ycsb-read</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
</tr>
<tr>
<td>ycsb-write</td>
<td>1</td>
<td>1.4</td>
<td>1.5</td>
</tr>
</tbody>
</table>

---

**HTA overview**

Adopts HTA table format that exploits caches
- Accelerates most operations with simple hardware
- Leaves rare cases to software
Uses HTA ISA extensions for hash table operations

- Accelerated by HTA function unit

**Software Hash Table**

- Accelerated by HTA function unit

**Two implementations**

- Flat-HTA
- Adds simple changes to cores
  - Reduces runtime overheads
  - Frees core backend from managing too many µops

- Hierarchical-HTA
  - Changes cores and caches
  - Improves spatial locality

---

**Hierarchical-HTA implementation**

Manages key-value pairs instead of cache lines
Maps hot pairs to same cache lines in small caches

**Execution**

- Fetch
- Decode
- Issue
- Execute
- Mem
- Commit

**HTA function unit**

- Address calculation
  - key → lineAddr
- Line comparison
  - lineValue → outcome