Fakultät für Informatik
Technische Universität München

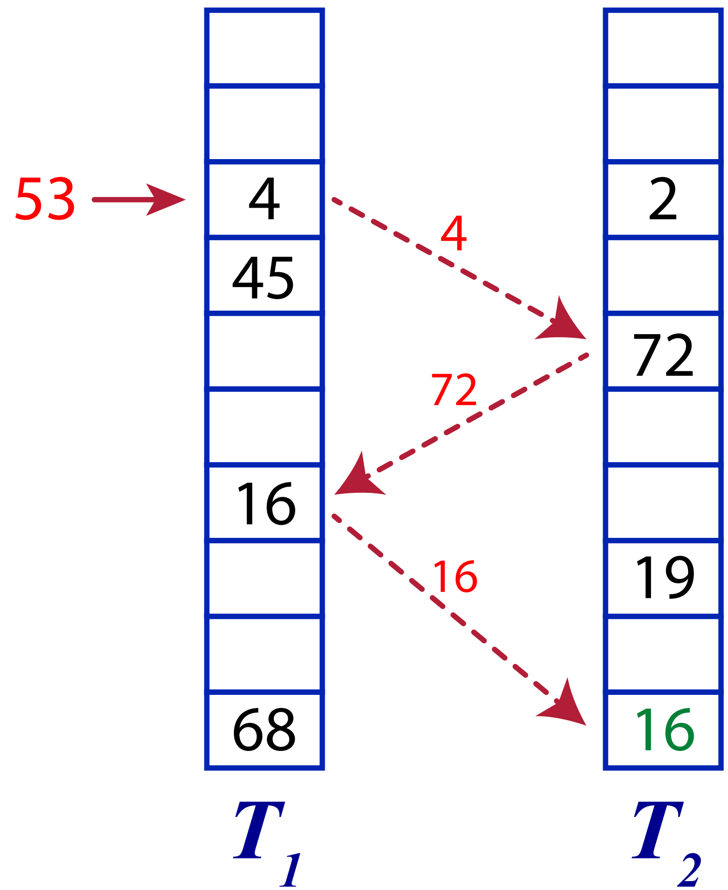
Seminar: Techniques for implementing main memory database

SIMD-Accelerated Hash Tables: Cuckoo Hashing

Rafid Ahmed

Cuckoo Hashing[1] :

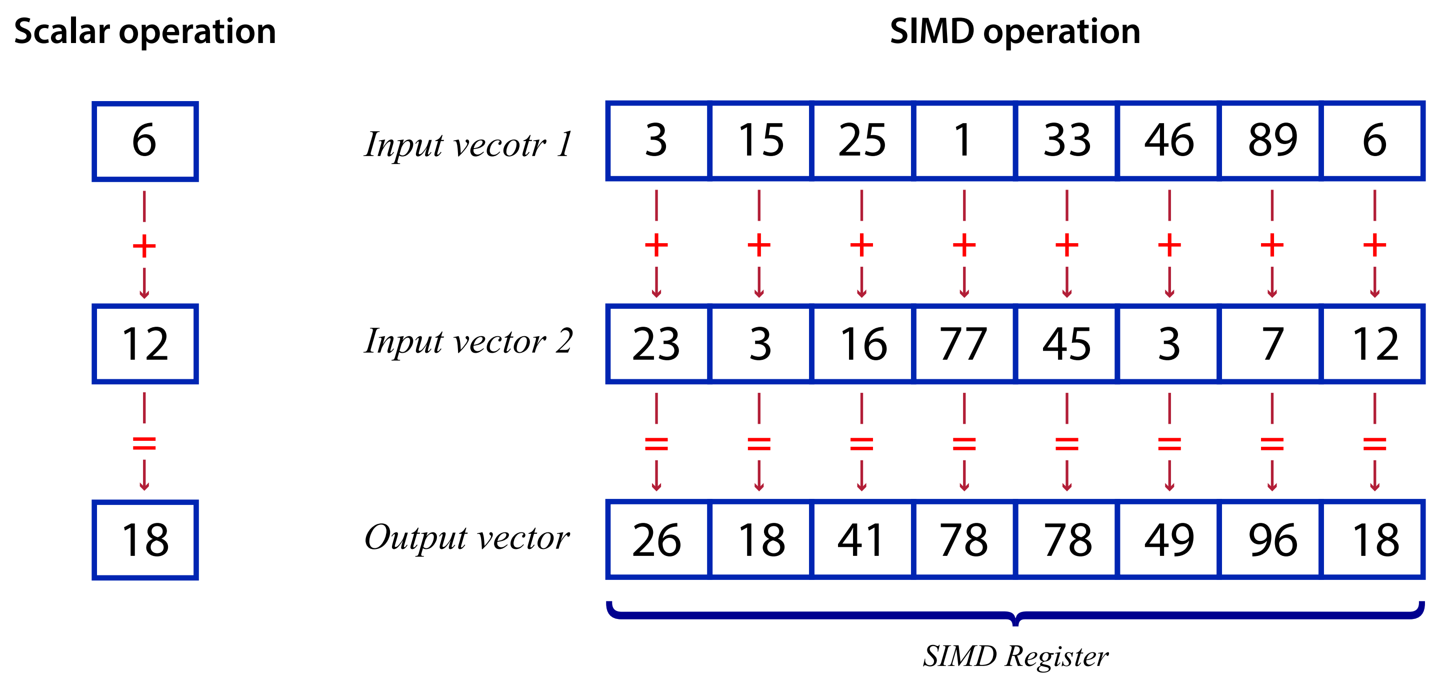
- Two locations must be checked for lookup
- Two locations must be checked for deletion
- Inserting a new key may push an older key to another position:
  - alternate position for older key is vacant
  - if not the procedure recurses until an empty slot is found or enters a cycle



Worst-Case Time Complexity of Hash Tables:

Operations	Linear Probing	Double Hashing	Random Hashing	Separate Chaining	Cuckoo Hashing
Search	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
Deletion	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
Insertion	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

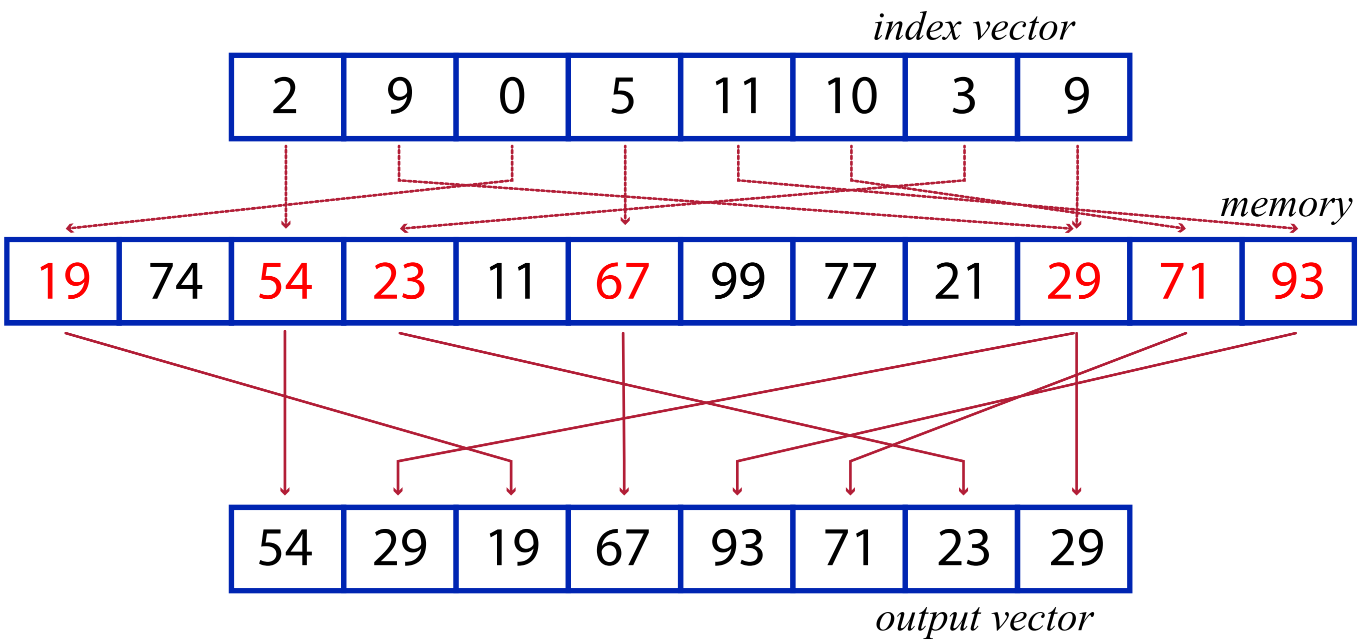
SIMD (Single Instruction Multiple Data):



$$O(f(n)) \qquad O(f(n)/W)$$

Fundamental Vector Operations:

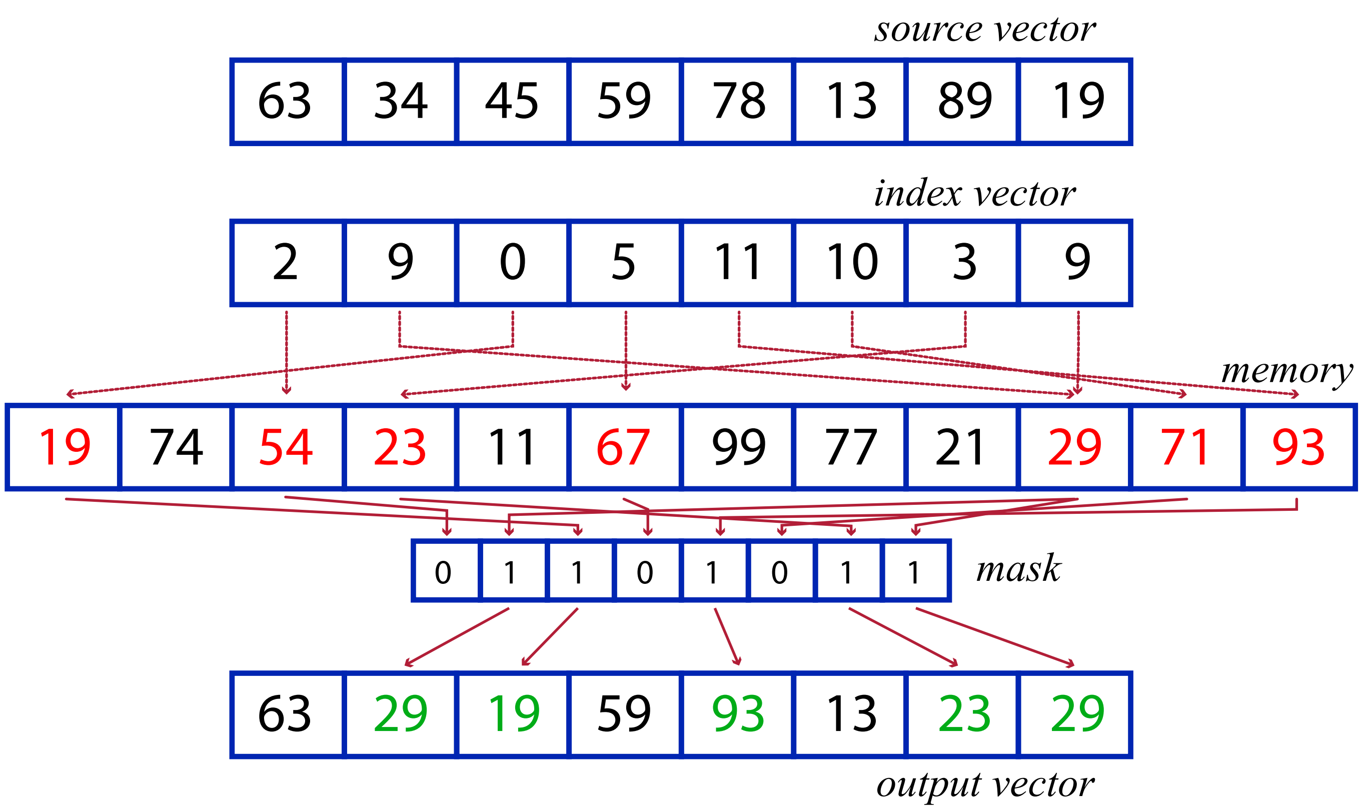
- **Gather:**




```
__m256i _mm256_i32gather_epi64 (__int64 const* base_addr, __m128i vindex, const int scale)
```

Fundamental Vector Operations:

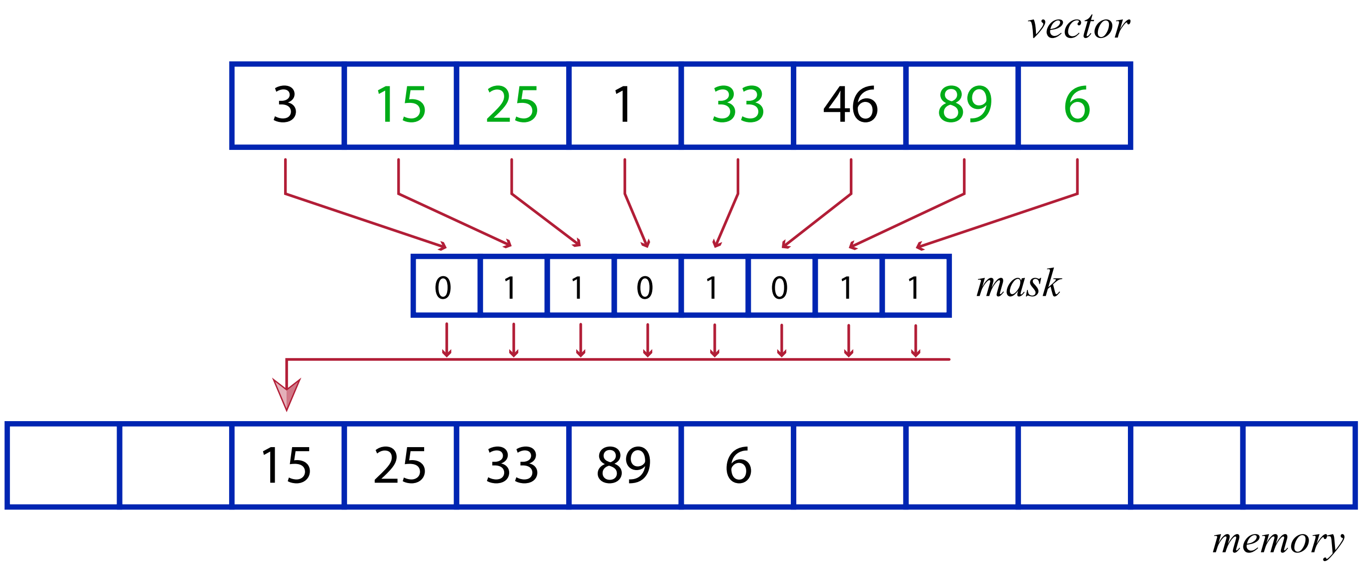
- **Selective Gather:**



__m256i _mm256_mask_i32gather_epi64 (__m256i src, __int64 const* base_addr, __m128i vindex, __m256i mask, const int scale)

Fundamental Vector Operations:

- **Selective Store :**



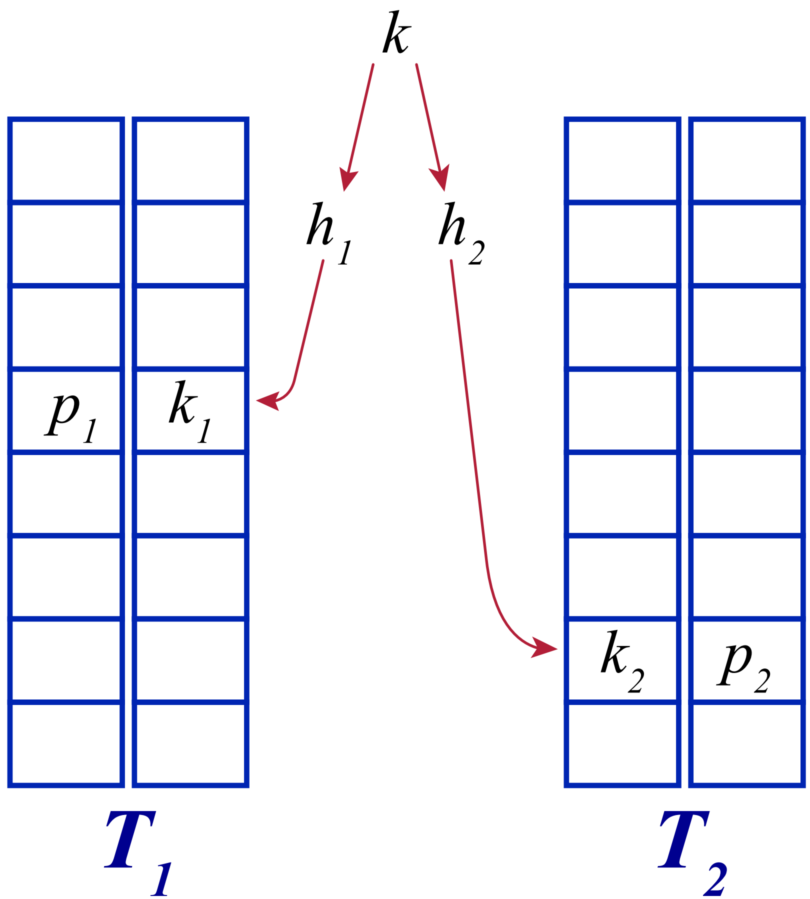
`__m256i _mm256_permutevar8x32_epi32 (__m256i a, __m256i idx)`

`void _mm256_maskstore_epi32 (int* mem_addr, __m256i mask, __m256i a)`

Cuckoo Hash tables:

Scalar Hash tables :

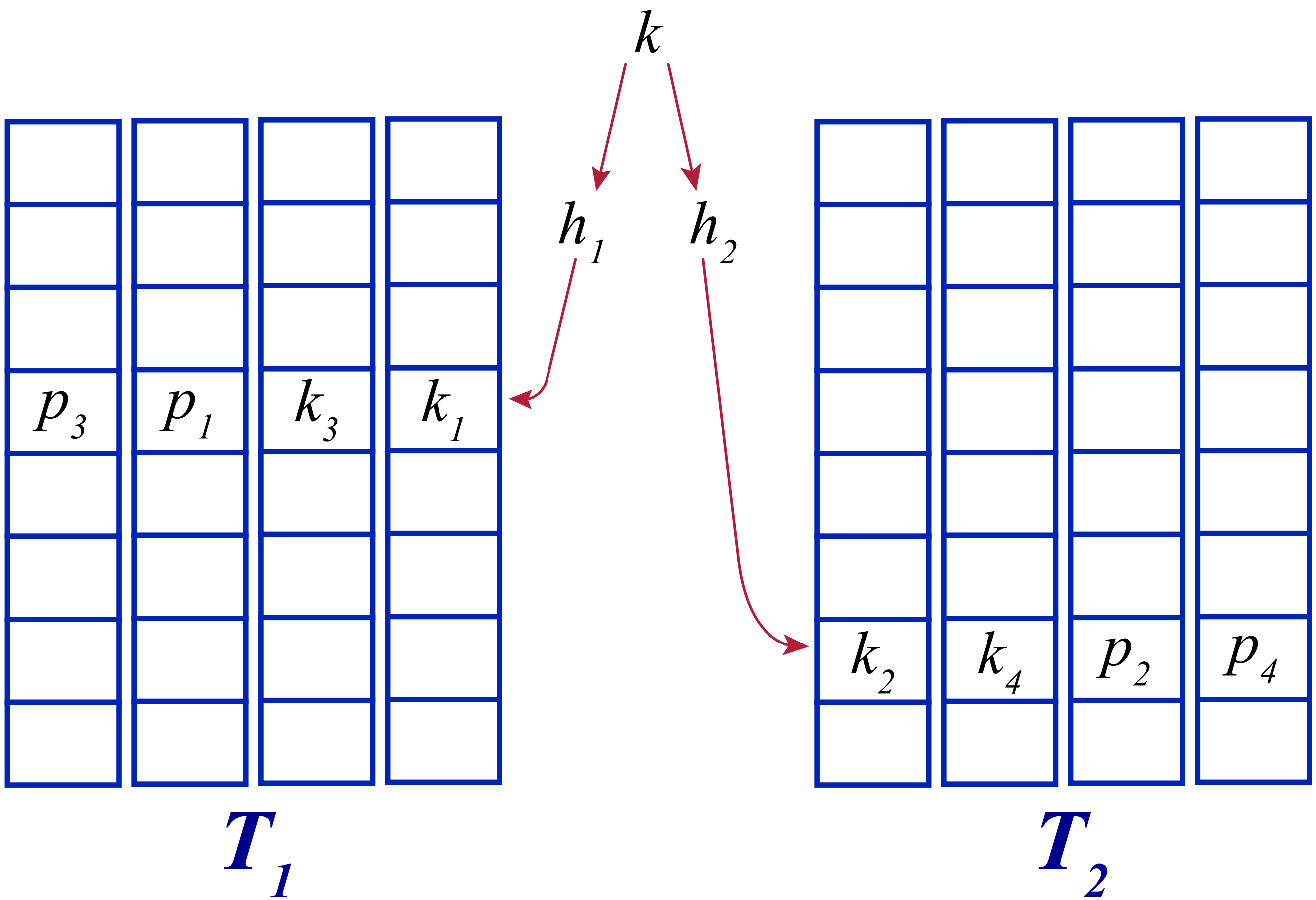
- Branching
  - conditionally access second location
- Branchless [2]
  - access both locations



Cuckoo Hash tables:

- **Vector Hash tables (Horizontal)[3] :**

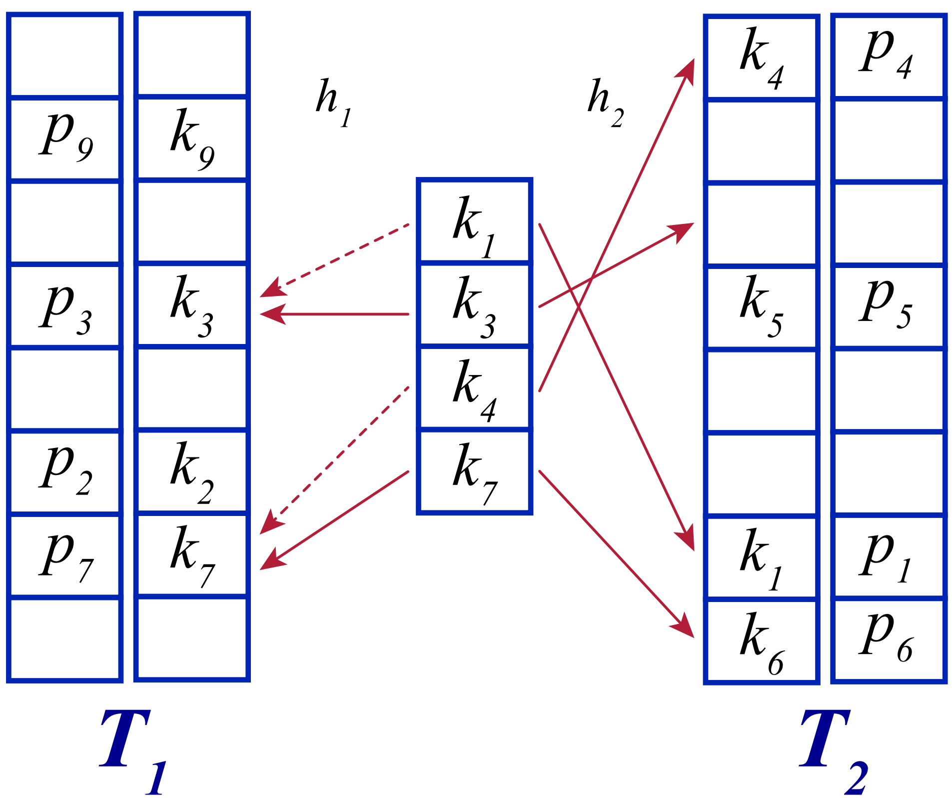
- Building
  - Hash buckets
  - Contiguous Keys
  - Contiguous Payloads
- Probing techniques
  - loop unrolling
  - SIMD masking



Cuckoo Hash tables:

- **Vector Hash tables (Vertical)[4] :**

- Blending
  - gather all buckets
  - generate masks from comparing keys
- Selective
  - gather first bucket per key
  - selectively gather second bucket for keys that did not match



Vertical (Selective) Cuckoo Hashing - Probing[4] :

$j \leftarrow 0$

for $i \leftarrow 0$ **to** $|S|$ - 1 **step** W **do**

$\vec{k} \leftarrow S_{keys}[i]$ — load input tuples

$\vec{v} \leftarrow S_{payloads}[i]$

$\vec{h}_1 \leftarrow (\vec{k} \cdot f_1) >> |T|$ — calculate 1st hash value

$\vec{h}_2 \leftarrow (\vec{k} \cdot f_2) >> |T|$ — calculate 2nd hash value

$\vec{k}_T \leftarrow T_{keys}[\vec{h}_1]$ — gather 1st hash bucket

$\vec{v}_T \leftarrow T_{payloads}[\vec{h}_1]$

$m \leftarrow \vec{k} \neq \vec{k}_T$ — compare hash bucket key with probe key

$\vec{k}_T \leftarrow_m T_{keys}[\vec{h}_2]$ — selectively gather 2nd hash bucket

$\vec{v}_T \leftarrow_m T_{payloads}[\vec{h}_2]$

$m \leftarrow \vec{k} = \vec{k}_T$ — again compare hash bucket key with probe key

$RS_{keys}[j] \leftarrow_m \vec{k}$ — selectively store matches

$RS_{S\_payloads}[j] \leftarrow_m \vec{v}$

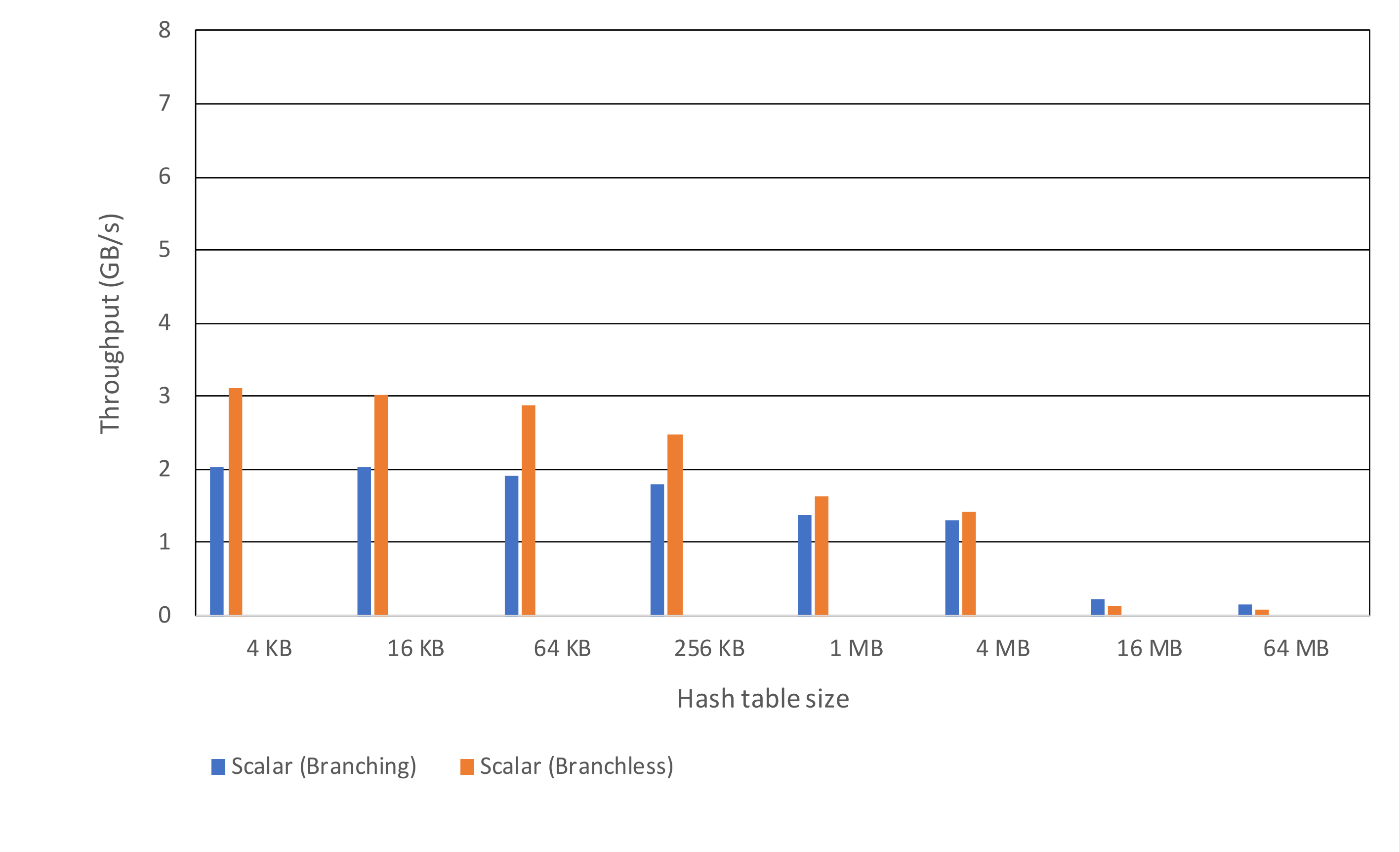
$RS_{R\_payloads}[j] \leftarrow_m \vec{v}_T$

$j \leftarrow j + |m|$

end for

Results:

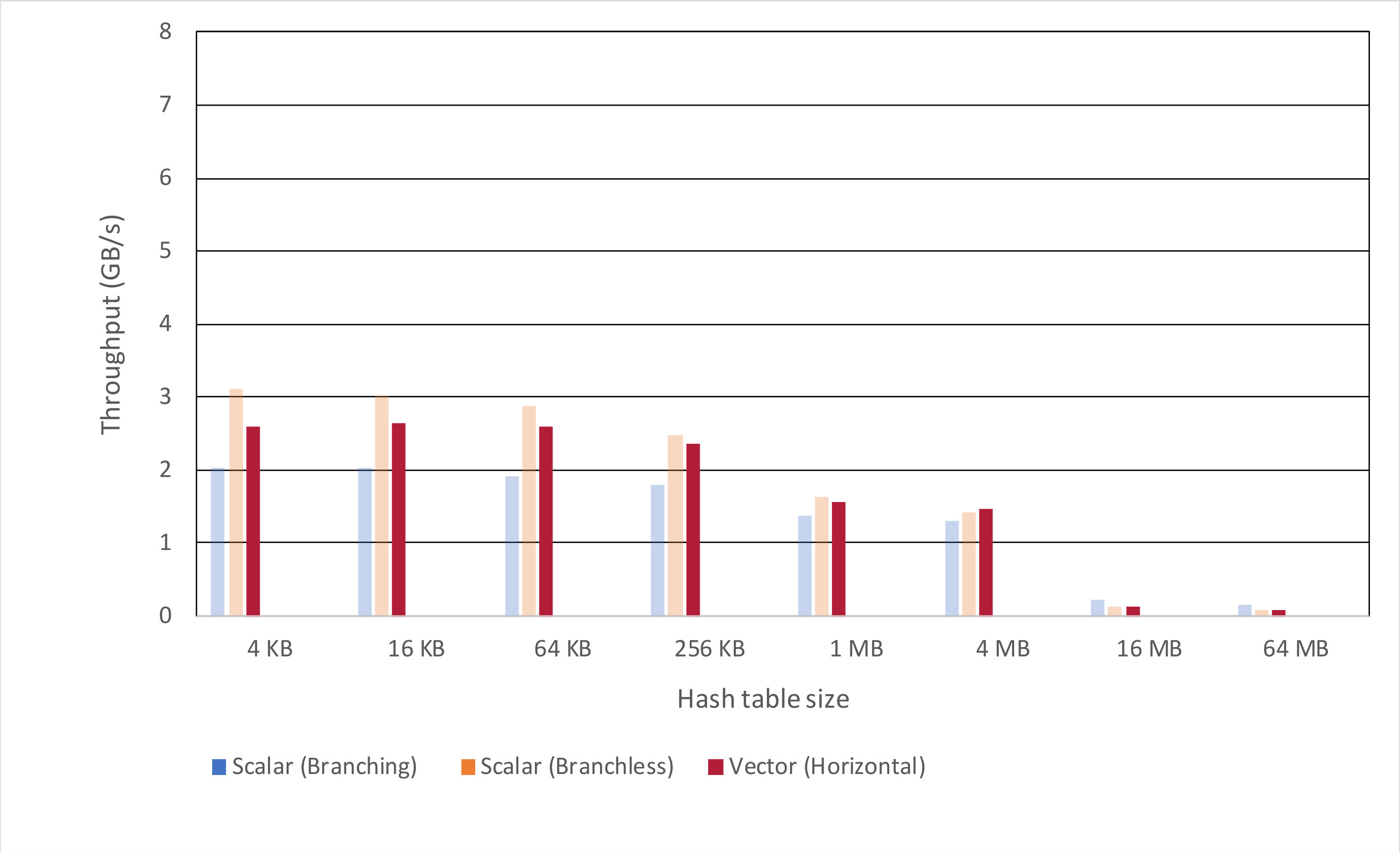
- **Cuckoo Hashing - Probing (30% Selectivity) Benchmark:**



Run on (4 X 3500 MHz CPU s)
CPU Caches:
L1 Data 32K (x4)
L1 Instruction 32K (x4)
L2 Unified 256K (x4)
L3 Unified 6144K (x1)

Results:

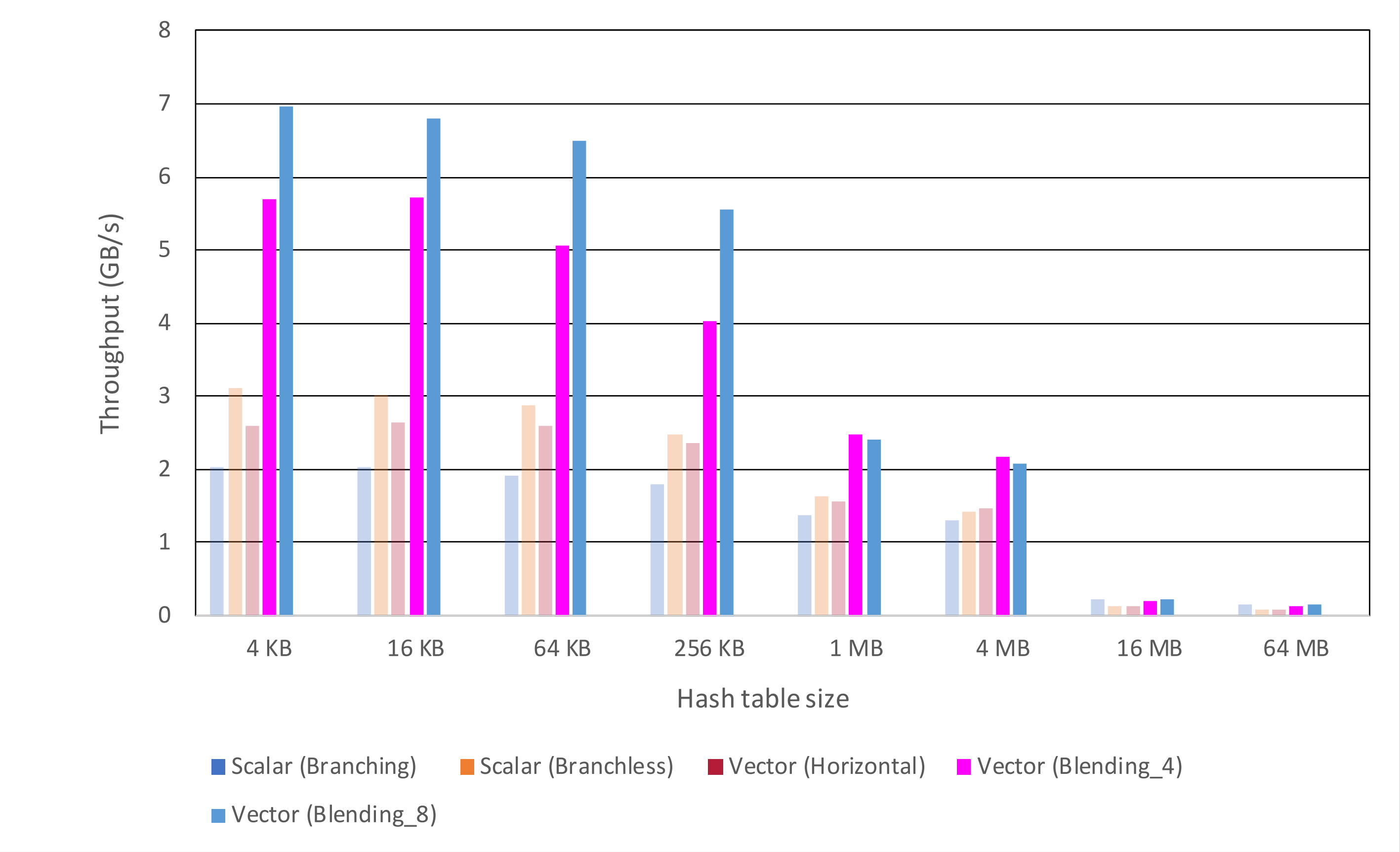
- **Cuckoo Hashing - Probing (30% Selectivity) Benchmark:**



Run on (4 X 3500 MHz CPU s)
CPU Caches:
L1 Data 32K (x4)
L1 Instruction 32K (x4)
L2 Unified 256K (x4)
L3 Unified 6144K (x1)

Results:

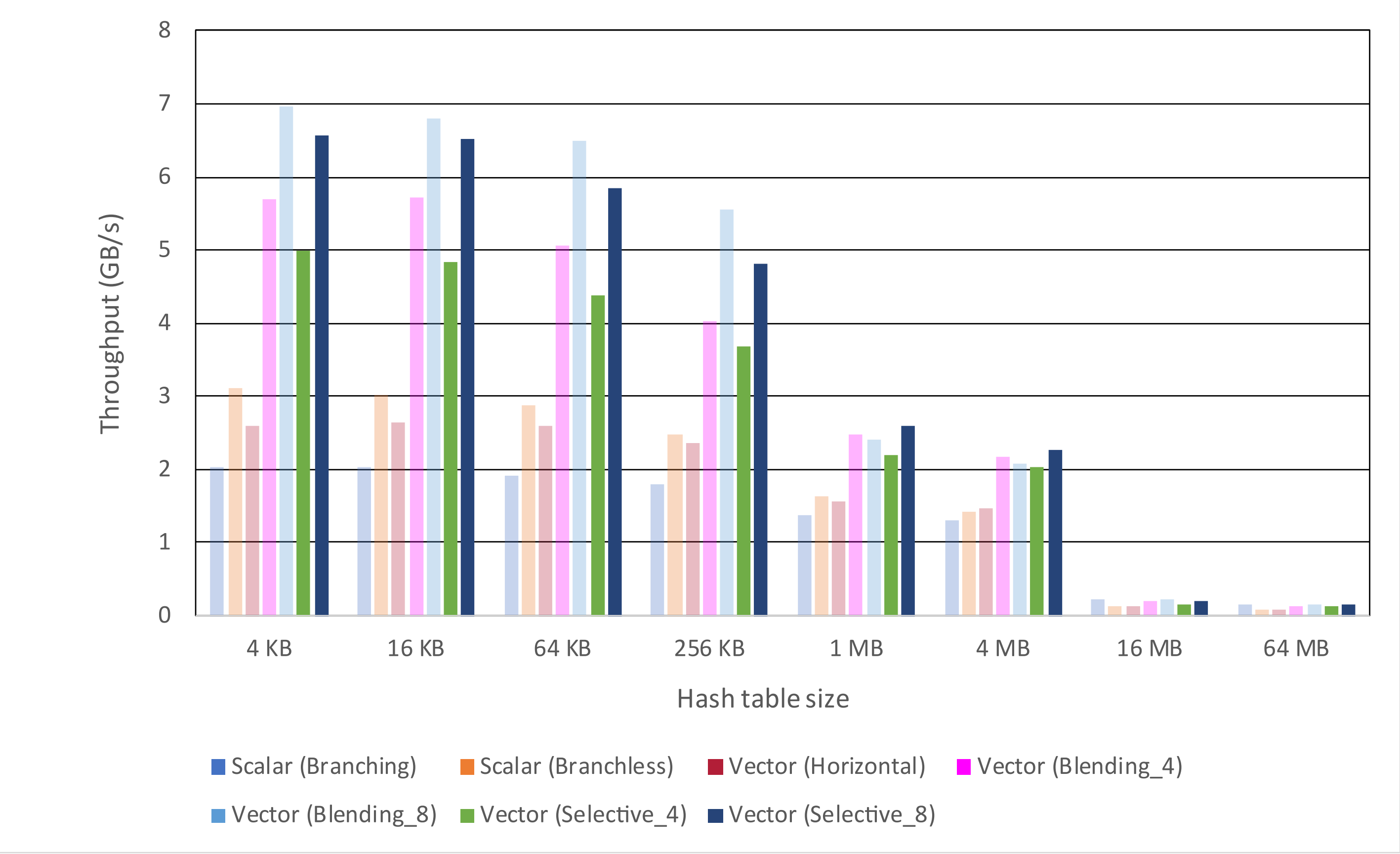
- **Cuckoo Hashing - Probing (30% Selectivity) Benchmark:**



Run on (4 X 3500 MHz CPU s)
CPU Caches:
L1 Data 32K (x4)
L1 Instruction 32K (x4)
L2 Unified 256K (x4)
L3 Unified 6144K (x1)

Results:

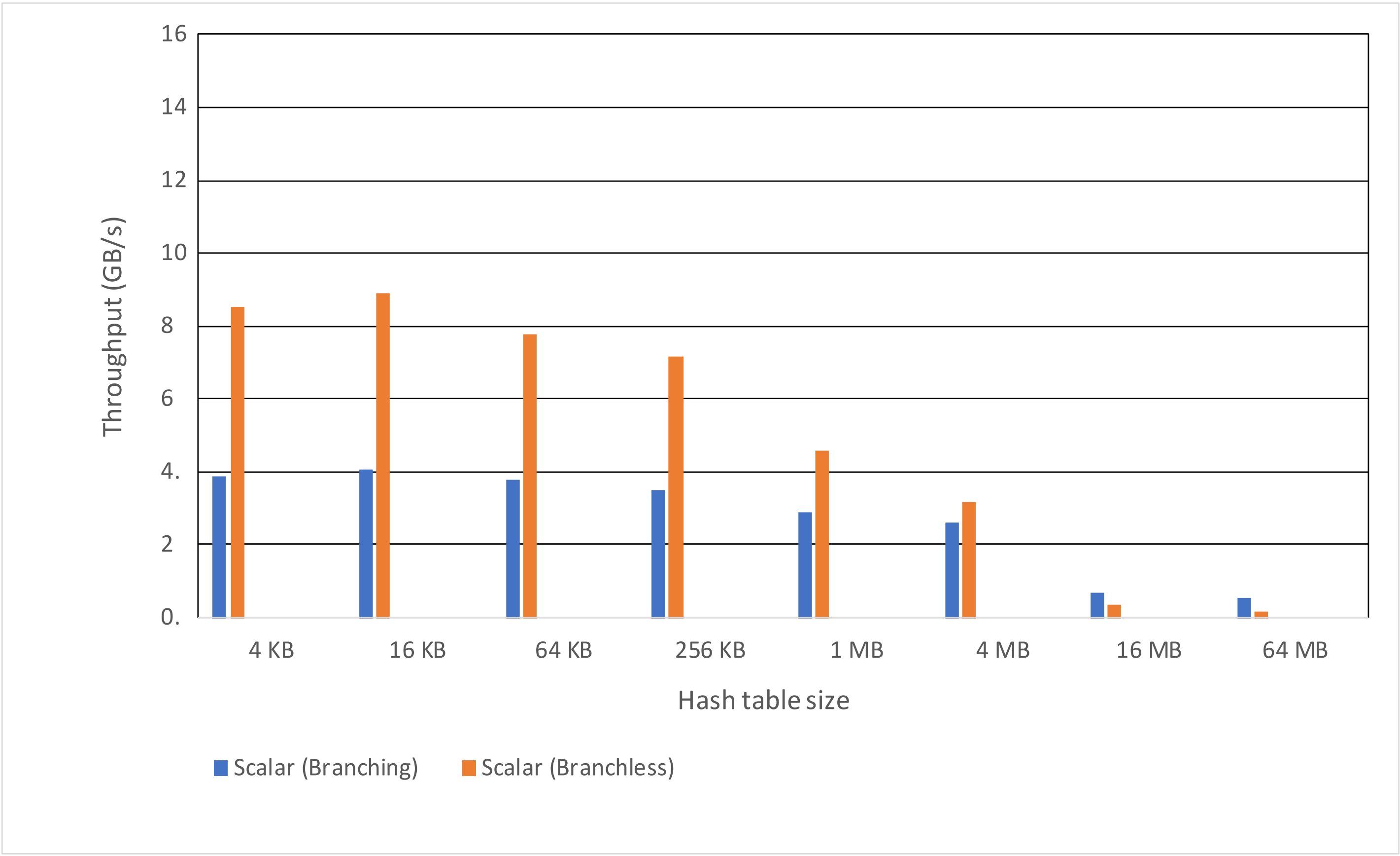
- **Cuckoo Hashing - Probing (30% Selectivity) Benchmark:**



Run on (4 X 3500 MHz CPU s)
CPU Caches:
L1 Data 32K (x4)
L1 Instruction 32K (x4)
L2 Unified 256K (x4)
L3 Unified 6144K (x1)

Results:

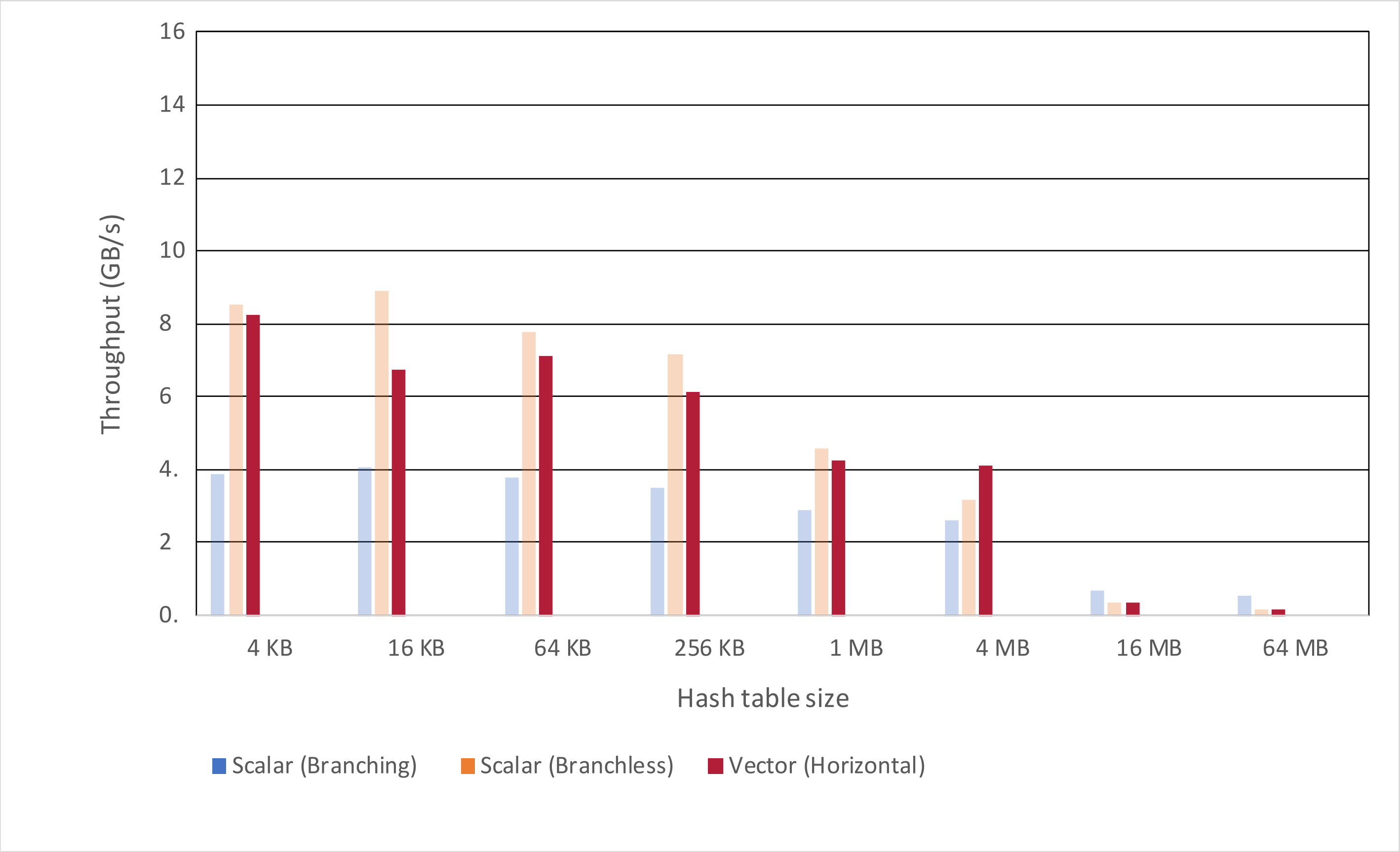
- **Cuckoo Hashing - Probing (90% Selectivity) Benchmark:**



Run on (4 X 3500 MHz CPU s)
CPU Caches:
L1 Data 32K (x4)
L1 Instruction 32K (x4)
L2 Unified 256K (x4)
L3 Unified 6144K (x1)

Results:

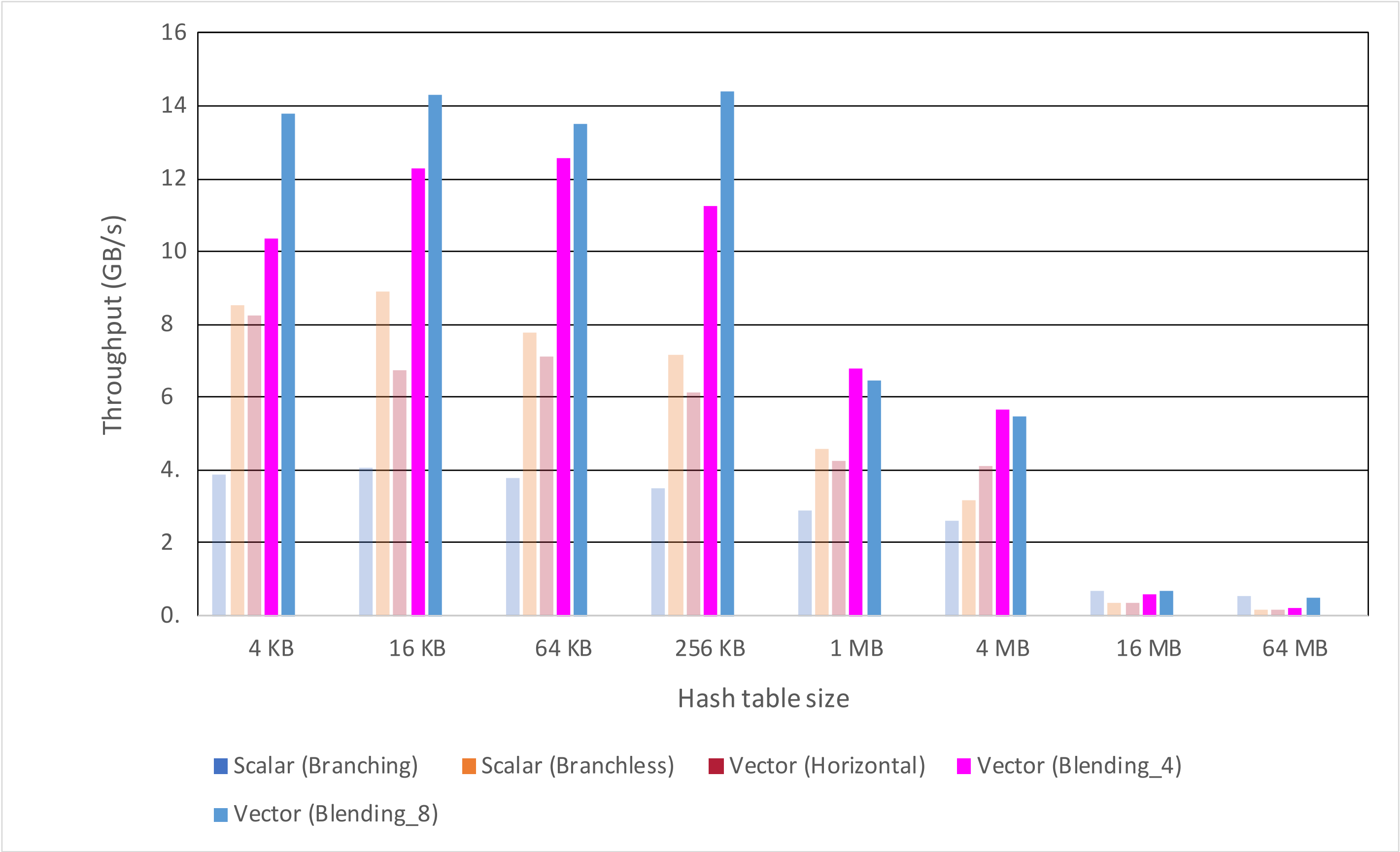
- **Cuckoo Hashing - Probing (90% Selectivity) Benchmark:**



Run on (4 X 3500 MHz CPU s)
CPU Caches:
L1 Data 32K (x4)
L1 Instruction 32K (x4)
L2 Unified 256K (x4)
L3 Unified 6144K (x1)

Results:

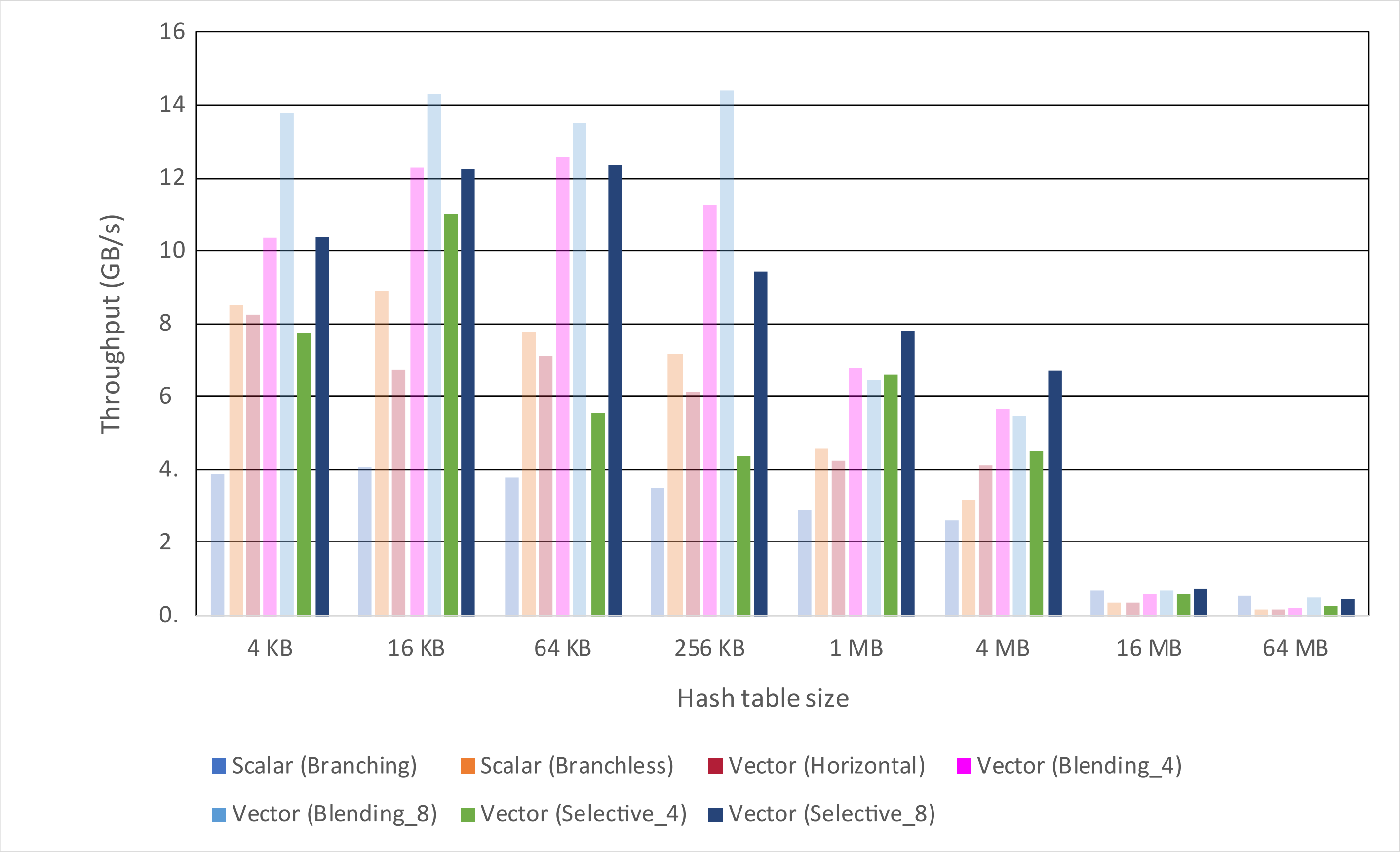
- **Cuckoo Hashing - Probing (90% Selectivity) Benchmark:**



Run on (4 X 3500 MHz CPU s)
CPU Caches:
L1 Data 32K (x4)
L1 Instruction 32K (x4)
L2 Unified 256K (x4)
L3 Unified 6144K (x1)

Results:

- **Cuckoo Hashing - Probing (90% Selectivity) Benchmark:**



Run on (4 X 3500 MHz CPU s)
CPU Caches:
L1 Data 32K (x4)
L1 Instruction 32K (x4)
L2 Unified 256K (x4)
L3 Unified 6144K (x1)

Summary:

- For both selectivity rate, vertically vectorized code performs about 1.8x faster than horizontal vector and scalar implementation

Sources:

[1] R. Pagh et al. Cuckoo hashing.

[2] M. Zukowski et al. Architecture-conscious hashing.

[3] K. A. Ross. Effcient hash probes on modern processors.

[4] O. Polychroniou, A. Raghavan, K. A. Ross. Rethinking SIMD Vectorization for In-Memory Databases