Concurrent Hash Tables: Fast and General!

Tobias Maier¹, Peter Sanders¹, and Roman Dementiev².

¹Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
²Intel Deutschland GmbH

Migration with minimal Synchronization

- Circular linear probing
- Each entry can be changed atomically (CAS) \(|(\text{key}) + [\text{value}] = 128\) bit
- Bound capacity 2\(\times n\) size
- Reserved keys for \(\text{(empty)}\) and \(\text{(deleted)}\)
- Addressing using the most significant digits of \(h(\text{key})\)

\[
\text{insert}(\text{key}, \text{value}) \quad h(\text{key}) = 1010111 \quad \text{(empty)}
\]

Displaced elements can only occur within the first cluster of the block.

Displaced entries should be avoided, because they make atomic insertions necessary (into the destination table).
- Therefore, we move the block borders to free spaces, this can be done implicitly during the block migration.
- This eliminates displaced entries, because elements cannot be displaced over empty cells (insertions in the destination table can be done non-atomically).
- The expected size difference between a block and the corresponding implicit block is bound by a small constant.

We also implemented two different options to ensure atomicity in the source table.
- Marking copied elements \((m\text{\_ours})\)
- Using flags to ensure that no update can operate concurrently to the migration \((f\text{\_ours})\)

Architecture/Table Management

- **Global Object:** stores the current table and some data that do not offer any functions (except create Handle)
- **Handle Object:** stores threadlocal data and exposes the hash table functionality that cannot be shared between threads

\[
\text{approx}\_\text{count} \quad \text{version} \quad \text{table}\_\text{ptr}
\]

Experiments

Insertions (growing needed)

- Measured by inserting 100 000 000 elements (strong scaling)
- Our tables were initialized with 4096 cells.
- The competitors were initialized with 50 % of the target size.

Successful find (with contention)

- Measured by searching 100 000 000 keys.
- Searched keys are Zipf distributed \((P(\text{key} = k) = \frac{k}{H\cdot s})\)
- Every searched key was previously inserted. To make sure that the table size doesn’t shrink under high contention we inserted 100 000 000 additional elements.
- Using 48 threads (24 cores + hyperthreading)

Test Setup

- dual-socket 2\(\times 12\) cores with 2.3 GHz each
- Intel Xeon E5-2670 v3 (codenamed Haswell-EP)
- 128 GB RAM
- each measurement is the average of 5 runs