com.tspowell.ttd.cache.associative.SetAssociativeCache<K, V>

N-Way Set Associative Cache

Travis Powell - tspowell@me.com

Design Approach

This implementation of a set-associative cache is not synchronized nor designed for concurrent applications. Two potential future approaches to concurrency:

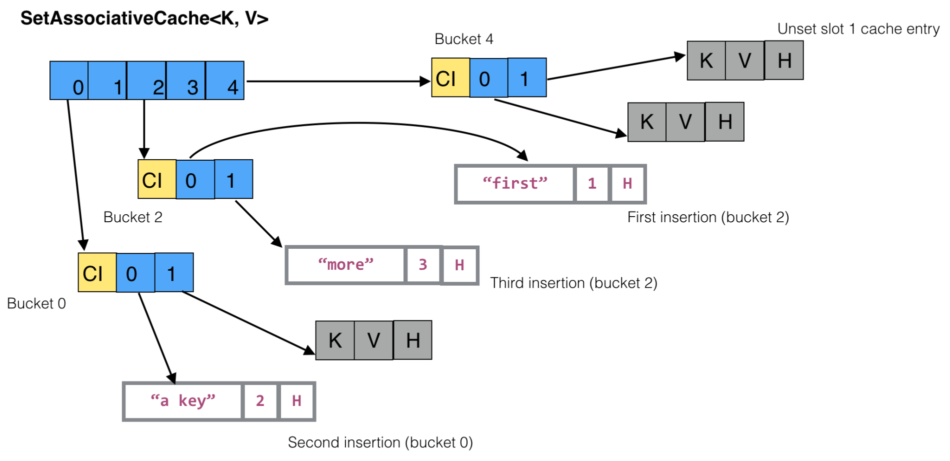
- Lock at the bucket level for reads, writes, and cache iterator traversal.
- Remove the “linked entry” interface and traverse over each slot in the bucket for every read. Writes would lock at the entry-level, and reads would be fully concurrent. Use a read/write timestamp to determine LRU.

As this is implemented in Java, cache locality isn’t as applicable (as there’s no contiguous allocation of non-primitive objects), and the CacheInvalidator may create other objects on the heap that are not allocated in coordination with the cache lifecycle. Therefore, the cache will only pre-allocate the entries themselves and allow the invalidators to maintain references to the slot, and request that the invalidator purge them on removal.

Memory Layout

To reduce the need to re-allocate cache entries, each bucket is pre-filled with cache slots pointing to null key-value pairs. When cache slots are released (either by calling the remove() API or calling put() into a particular bucket set once the bucket is full) the cache slot is unset(), flipping a set bit and removed from the linked list.

Example: 5 sets of 2-ways. 3 insertions in a String-Integer cache.
- Blue pointers are initialized
- Grey pointers are null
- Gold CI pointers point to a cache invalidator per bucket. (This invalidator class should maintain its own references to the cache entries on the heap)
- The hash is also stored with each cache entry for fast comparison.



API and Usage

The cache **SetAssociativeCache<K, V> implements Map<K, V>, Iterable<Cache.Entry<K, V>>**. Developers should prefer the *Map* interface. Ideally the cache would implement the full JCache API, but that is out of the scope of this initial release. We do, however, include the JCache dependency for the Cache.Entry interface and future JCache extension. The Maven project creates a fat JAR with all dependencies for client applications.

Build Environment

Requires JDK 8 and Apache Maven. (Developed with Maven 3.3.9)

To package:

```
$ mvn package
```

Extension: Alternative Cache Invalidation Algorithms

Three existing invalidation strategies are provided: *MRUInvalidator, LRUInvalidator* and *SmallestValueInvalidator*.

New invalidation strategies must implement the CacheInvalidator<K, V> interface. The implementation of invalidate() should make sure to call **.unset()** on the UnsettableEntry to ensure that dangling pointers don't keep references to invalidated values, but these values will likely be overwritten on the next call to put().

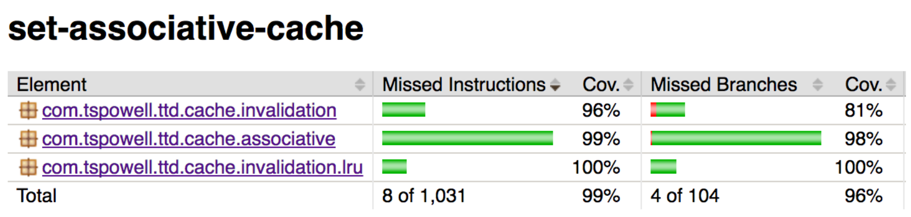
```
public interface CacheInvalidator<K, V> {

    void touch(UnsettableEntry<K, V> entry);
    void remove(UnsettableEntry<K, V> entry);

    /**
     * @return true if an item was removed, false if no items were removed.
     */
    boolean invalidate();
}
```

Code Coverage

99% / 98% coverage across unit and generative tests, and 100% coverage of the core implementation.



set-associative-cache

Element	Missed Instructions	Cov.	Missed Branches	Cov.
com.tspowell.ttd.cache.invalidation	🟩	96%	🟥🟩	81%
com.tspowell.ttd.cache.associative	🟩	99%	🟥🟩	98%
com.tspowell.ttd.cache.invalidation.lru	🟩	100%	🟩	100%
Total	8 of 1,031	99%	4 of 104	96%