N-Set Associative Cache Design Document

Trevor Lewis Problem:

Create a set associative cache that satisfies the following requirements:

1) Cache is entirely in memory.
2) Client interface should be:
    a. Type-safe for keys and values.
    b. Arbitrary type for keys and values.
    c. Each cache instance must contain same type for keys and values.
3) Design interface as library to be distributed to clients.
4) Provide LRU and MRU algorithms
5) Allow for alternative replacement algorithm.

Solution:

I created an array that is type specific based on the first set of key value pair that is inserted into the cache. I instantiate the array to the correct size needed by multiplying the number of sets in the cache by the number of cache blocks per set that has been specified by the client.

Note: Some of the structure and methods of this cache are taken from Guava Cache by Google. I have used this cache library in a project so I was familiar with how it acted and some of the methods that are implemented in it.

Design:

Cache Interface:

The cache interface contains the blueprint for any cache that is created by implementing it.

void put(Object key, Object value)

Insert key and mapped value into cache.

Object get(Object key)

Retrieve the value that is mapped to key.

long size()

Retrieve the number of cache blocks that are not empty.

void invalidate(Object key)

Invalidate cache block mapped to key.

void invalidateAll()

Invalidate all the blocks in the cache.

boolean isEmpty()

Determine is the cache is empty.

AlgoritmeTypes Enum:

Types of replacement algorithms that able to be used in cache.

Using enums to determine what replacement algorithm allows for control over what the client is able to enter. If say a string object was used the client would be able to enter whatever they felt like. The enum allows the client to know their options and allowed the code to have less validation or manipulation on that parameter.

LRU = Least Recently Used

Discards the least recently used cache block.

MRU = Most Recently Used

Discards the most recently used cache block.

CUSTOM = Client Alternative Replacement Algorithm

The client is able to override the customizedReplacementAlgorithm() with the replacement algorithm or their choice.

CachEntry Class:

The structure of the data that is used in the cache.

int tag

Is used to determine the cache set that the CacheEntry object will be inserted into.

Object data

The data that will be stored in the cache.

long lastHitTimestamp

The timestamp of when the object was inserted into the cache or when is was last retrieved from the cache.

boolean isEmpty

Indicator to weather the object contains data or not.

NSetAssociativeCache Class:

void put(Object key, Object value)

Insert key and mapped value into cache.

Validation is used here to provide a solution to the requirement of having each instance of the cache contain the same type of keys and values. I used two variables to act as master types for comparison in the validation. These variables are set on the first insertion to the cache. They are also erased if the cache is invalidated manually and contains no entries.

Logic is also present here to determine the replacement block index based on the replacement algorithm that is being used with the instance of the cache.

Object get(Object key)

Retrieve the value that is mapped to key.

There is logic here that is used to determine the set of blocks that the entry would be located in. The code checks those blocks to retrieve the entry. If the entry is present the last hit timestamp variable of the object is update to the current time in nanoseconds and then returned to the calling method. If the entry is not present in those blocks null is returned by default. The client has the option to override the cacheMiss() to have the code retrieve the entry from main memory or another location.

long size()

Retrieve the number of cache blocks that are not empty.

The cache array is populated with empty cacheEntry objects when it is created. Since I designed it this way to represent the cache blocks available I need to check each entry and count the entries that are not empty. *void invalidate(Object key)*

Invalidate cache block mapped to key.

Replaces the CacheEntry object mapped to the key with an empty cacheEntry object. If this was the only entry in the cache it will also reset the masterKey and masterValue variables so that the cache can be used with any type.

void invalidateAll()

Invalidate all the blocks in the cache.

Replaces all the CacheEntry objects in the cache with empty CacheEntry objects. Also, resets the masterKey and masterValue variables so that the cache can be used with any type.

boolean isEmpty()

Determine is the cache is empty.

Object cacheMiss()

Returns an object if there is a cache miss.

By default, this method returns null, meaning the entry was not found in the cache. The client can override this method and provide the instructions to look for the data elsewhere. Example locations would be main memory or database entry.

ReplacementAlgorithm Class:

Provides replacement algorithms for LRU, MRU, and a default customized replacement algorithm.

int *customizedReplacementAlgorithm()*

A method for a client to override to provide their own custom algorithm.