On Exploring Safe Memory Reclamation Methods with a Simplified Lock-Free Hash Map Design

Pedro Moreno[1,2], Miguel Areias[1] and Ricardo Rocha[1]

[1] CRACS/INESC TEC, Department of Computer Science, Faculty of Sciences, University of Porto, Portugal

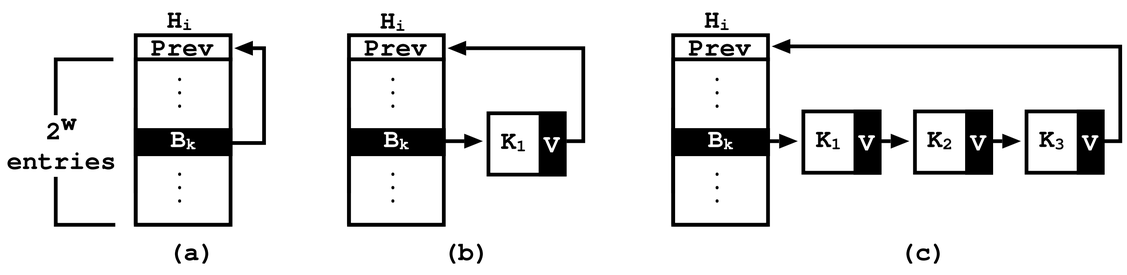
[2] Instituto de Astrofísica e Ciências do Espaço, University of Porto, Portugal

Introduction

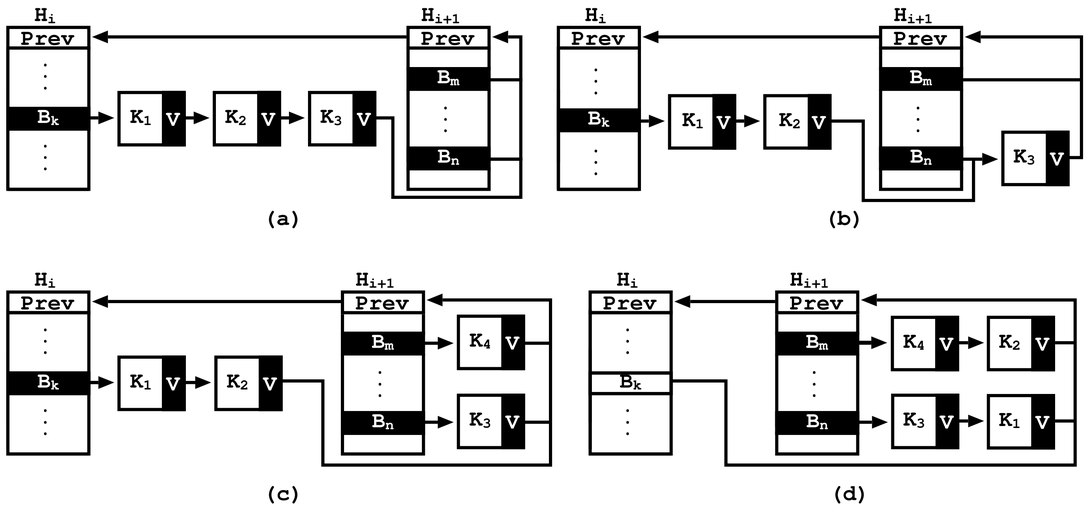
Hash maps are a very common and efficient data structure used to organize information that must be accessed frequently. Hash tries are a tree-based data structure with nearly ideal characteristics for the implementation of hash maps, which allows to efficiently solve the problems of setting the size of the initial hash table and of dynamically resizing it in order to deal with hash collisions.
In this work, we focus on simplifying a sophisticated implementation of a lock-free trie-based hash map, named *Lock-Free Hash Tries* (LFHT) by making it simpler, more cache friendly and compatible with most *safe memory reclamation* (SMR) methods.

Original Lock-Free Hash Tries

The original LFHT design has two kinds of nodes: hash nodes and leaf nodes. The leaf nodes store key/value pairs and the hash nodes implement a hierarchy of hash levels, each node with a fixed size bucket array of 2^w entries. To map a key/value pair (k,v) into this hierarchy, a hash value h is computed for k and then chunks of w bits from h are used to index the appropriate hash node, i.e., for each hash level H_i , the ith group of w bits of h are used to index the entry in the appropriate bucket array of H_i . To deal with collisions, the leaf nodes form a linked list in the respective bucket entry.



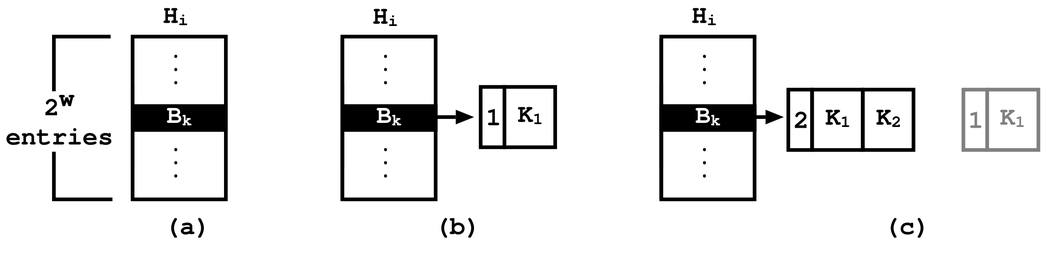
When a collision threshold is met an expansion operation updates the nodes in the linked list to a new hash level H_{i+1} , i.e., instead of growing a single monolithic hash table, the hash trie settles for a hierarchy of small hash tables of fixed size 2^w .



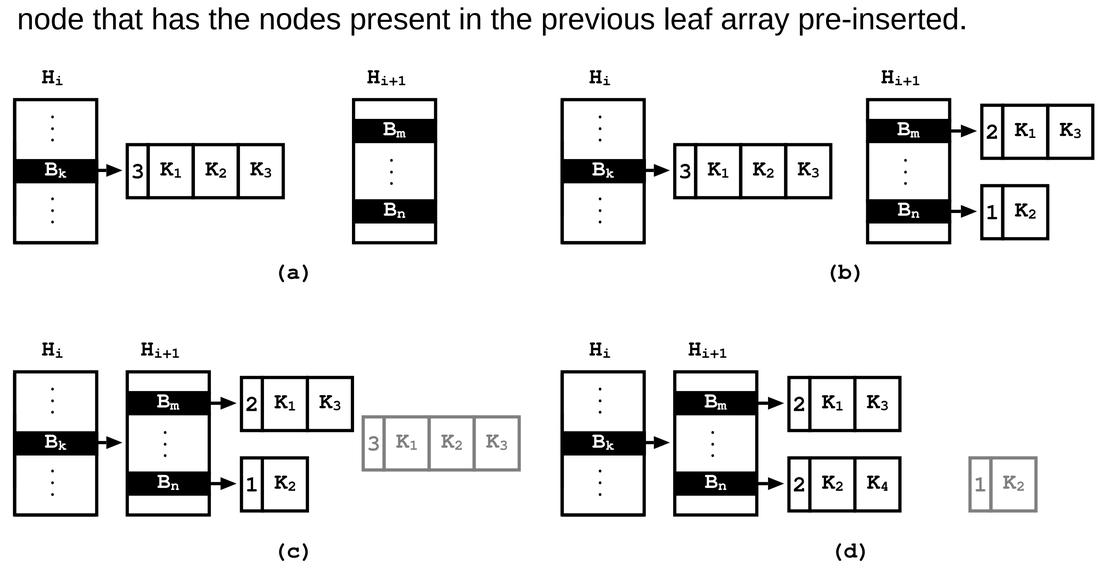
This expansion strategy renders the data structure Incompatible with most SMR methods.

Simplified Lock-Free Hash Tries

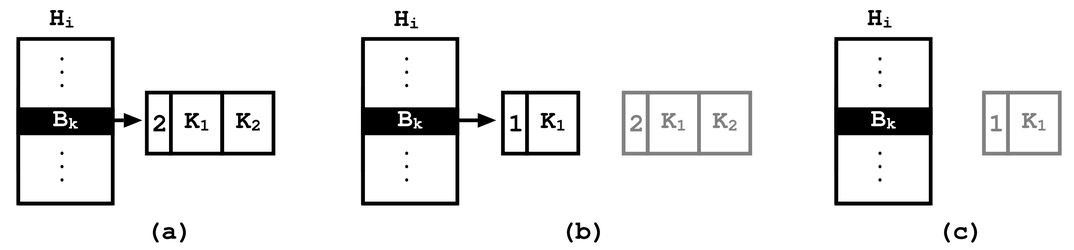
The key idea behind the new SLFHT design is to replace these collision chains with a specialized array of leaf nodes with a header that specifies the number of nodes in the array, followed by the nodes that collide in the corresponding bucket entry sequentially in memory. We call these arrays of leaf nodes as leaf arrays. The insertion procedure, instead of adding a node to the chain, replaces the entire leaf array with a new one containing all the previous leaf nodes plus the new node.



In the expansion procedure we simply replace the leaf array with a new hash node that has the nodes present in the previous leaf array pre-inserted.



Similarly, the removal procedure replaces the entire leaf array with a new one that contains all the previous nodes except the one being removed.

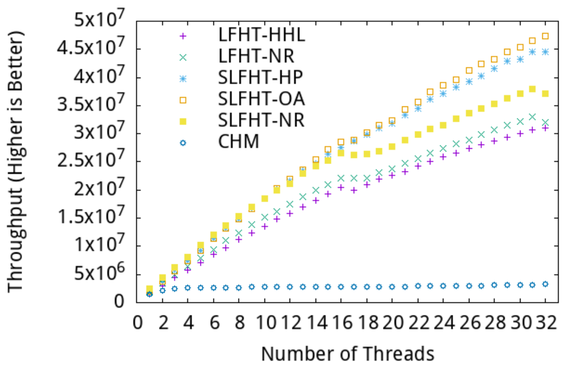


Safe Memory Reclamation

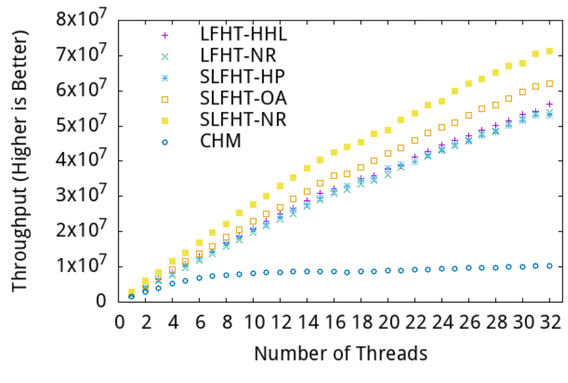
Safe memory reclamation (SMR) on lock-free data structures is a much harder problem, since exclusive access to any region of the data structure can not be expected without violating the lock-free properties.
Often, most lock-free data structure designs start by relying on garbage collection systems in order to reclaim memory, even though that such an option destroys the lock-freedom property of the system as a whole and can have a significant performance impact. Since garbage collection is not an option to ensure lock-freedom and there is no ideal SMR method, there is increasing pressure to design data structures such that they are easily integrated with more SMR methods.
We implemented two different SMR methods for the SLFHT design. We chose the Hazard Pointers (HP) method, as it is the most commonly used one and tends to achieve good performance in data structures with low depth with tight memory bounds. As a second option, we chose the Optimistic Access (OA) method that is one of the most efficient memory reclamation methods while being robust and simple to implement.

Experimental Results

Our experimental environment was a machine with 2 x AMD Opteron™ Processor 6274 with 16 cores each and a total of 32 GiB of DDR3 memory. The machine was running Ubuntu 22.04 with kernel 5.15.0-91 and all designs were compiled with GCC version 13.2.1.
For the performance analysis, we evaluate SLFHT against LFHT and the Concurrent Hash Map design (CHM) of Intel-TBB library version 2021.5.0. The scenarios in the figures specify the design and the memory reclamation method in use, e.g., LFHT-HHL means the LFHT design with the HHL memory reclamation method. The HHL (Hazard Hash and Level) method is the original memory reclamation method implemented for LFHT, the HP (Hazard Pointers) and OA (Optimistic Access) are the two SMR methods discussed for SLFHT, and NR are the versions without memory reclamation support.



50% Inserts and 50% Removes



90% Searches, 5% Inserts 5% Removes

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within projects UIDB/04434/2020, UIDP/04434/2020 and UIDB/50014/2020.
DOI 10.54499/UIDB/50014/2020