Future Generation Computer Systems 82 (2018) 127–141

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Concurrent hash tables on multicore machines: Comparison, evaluation and implications

Zhiwen Chen [a], Xin He [a], Jianhua Sun [a], Hao Chen [a],*, Ligang He [b]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
[b] Department of Computer Science, University of Warwick, Coventry CV47AL, United Kingdom

## HIGHLIGHTS

- A unified testing framework for concurrent hash tables is presented.
- Extensive evaluations are performed from a wide range of perspectives.
- The experiments are conducted on four major multi-core hardware platforms.
- Implications made in this paper could be used as guidelines for future research.

ARTICLE INFO

*Article history:*
Received 3 August 2017
Received in revised form 13 November 2017
Accepted 27 December 2017
Available online 1 January 2018

*Keywords:*
Concurrent hash table
Multiprocessors
Multi-core
Synchronization
NUMA
Hardware transactional memory

## ABSTRACT

Concurrent hash table has been an area of active research in recent years, and a wide variety of fast and efficient concurrent hash tables (CHTs) have been proposed to exploit the advantages of modern parallel computer architectures such as today's mainstream multi-core systems. As one of the fundamental data structures widely used in software systems, existing works on CHTs focus on either algorithmic improvements, or hardware-oriented optimizations, or application-specific designs. However, there is a lack of a comprehensive and comparative study on different implementations. In this paper, we conduct an experimental study on the state-of-the-art, and our goal is to critically review existing CHTs from wider aspects and with more detailed analysis. Concretely, we have conducted extensive evaluations of five CHTs using a unified testing framework on four multi-core hardware platforms, and implemented our HTM-based concurrent hash table. A variety of metrics such as throughput scalability, latency, impact of memory hierarchy, thread pinning strategies, synchronization mechanisms, and memory consumption, are measured in order to obtain the deep insights about performance impediments and good design choices. With this study, we hope to identify potential issues and pinpoint promising directions for future research of CHTs.

## 1. Introduction

As the number of cores has been increasing on modern computer architectures in recent years, challenges in inventing new or enhancing existent concurrency data structures to fully leverage the hardware advancements are emerging. Hash table is a well-known data structure, which provides simple interfaces to access the elements. *lookup*, *insert* and *delete* are the three main operations provided by hash tables. As it can offer constant time lookup and update operation, it is widely used in most software systems [1,2]. In spite of the fact that the study on sequential hash tables is relatively mature, the research on concurrent hash tables (CHTs) has attracted a lot of efforts in recent years, due to the promising performance, hardware advancement, and diversified requirements in different usage scenarios.

Ideally, CHTs should achieve high performance and scalability under different workloads and hardware settings. However, designing and implementing such CHTs is very challenging [3,4]. Hardware-conscious CHTs leveraging platform-specific features are often ineffective in obtaining portable performance [5]. On the other hand, hardware-oblivious CHTs often fails to achieve highest possible performance. Similarly, a CHT optimized for a specific type of workloads may exhibit poor performance under a slightly different workload. For example, the Read-Copy-Update (RCU) based hash table is one of the workload sensitive CHTs. It obtains high throughput and shows good scalability when dealing with read-only workloads. However, for workloads with a small fraction of update operation, it exhibits significant performance degradation. In [5], the authors present a complete picture of

* Corresponding author.
*E-mail address:* haochen@aimlab.org (H. Chen).

https://doi.org/10.1016/j.future.2017.12.054
0167-739X/© 2017 Elsevier B.V. All rights reserved.

how synchronization schemes behave in concurrent algorithms. In most cases, when a scalability issue is encountered, it is not straightforward to identify the root cause that may be the underlying hardware, synchronization algorithm, usage of specific atomic primitives, application context, or workloads.

Although both the industry and academia have proposed a range of CHTs with different target such as Threading Building Blocks (TBB) [6] and ConcurrentHashMap in Java [7], the performance of CHTs not only is related to the application requirements but also relies on the exploitation of lower-level hardware characteristics. When profiling CHTs, we need to perform the analysis by integrating many relevant elements and considering at different levels rather than evaluating based only on intuitive metrics such as throughput and latency. Furthermore, with a unified testing framework and a set of common performance metrics, it seems that no single CHT can outperform others in all aspects when handling a diversified set of workloads. On the other hand, from the perspective of users, the effective way to adopt a CHT is to clearly recognize all the potential performance obstacles. Unfortunately, these practical concerns are rarely mentioned in previous studies. Lacking a unified benchmark to evaluate CHTs, it is hard for the users to make a decision about which CHTs to employ in their software systems in order to attain desired performance. In summary, a comprehensive and in-depth analysis is of significance in using, designing, and optimizing CHTs. Inspired by the practices, we choose five state-of-the-art CHTs to conduct a comprehensive evaluation and analysis across a wide set of metrics. The five CHTs taken from the literature are listed in Table 1 with brief description.

CHTs in this study are written in C/C++, and they are evaluated on 4 multi-core platforms: AMD Opteron, Intel Xeon Phi 7120P (a many-core platform based on Intel MIC architecture), Intel Xeon E5-2630, and Intel Xeon E7-4850. To the best of our knowledge, it is the most comprehensive evaluation of concurrent hash tables to date. We make the following contributions in this paper.

- First, we present a framework, named **CHT-bench**, which provides a fair testing environment and unified interface for the experiments by hiding the discrepancies of hardware platforms, synthesized workloads, concurrency models, and compiler configurations. The source code of this work can be found at https://github.com/Gwinel/CHT-bench. In this way, we can guarantee the experimental results generated from our framework are fairly comparable between different CHTs.
- Second, the evaluations are explored from a wide range of perspectives including thread scalability, throughput, latency, memory hierarchy impact, low-level synchronization primitives, and memory usage. The inter-correlations between relevant metrics are also discussed when necessary. The experiments are conducted on four major hardware platforms including Intel MIC and three representative NUMA systems. We ported CHTs to the MIC platform, and to our knowledge, this is the first extensive study of concurrent hash tables on Intel MIC architecture.
- Third, implications about pitfalls, design trade-offs, and desirable optimizations are summarized for each evaluated metric, which can serve as guidelines for future research and practical development of CHTs.

The rest of the paper is organized as follows. Section 2 provides brief background on CHTs and modern computer hardware features. We present the experimental platforms and parameter configurations in Section 3. A comprehensive analysis of CHTs are made in Section 4. The related work is presented in Section 5. Section 6 concludes this paper.

## 2. Background

The explosive growth of commercial multiprocessor machines has brought about a revolution in the art of concurrent programming. The shared-memory programming model enforced by the underlying hardware and programming languages/runtime systems imposes much greater challenges in designing and verifying concurrent data structures than their sequential counterparts. In this section, we first introduce basic concepts and operations of concurrent hash tables and common metrics used to evaluate them. Inherent hardware features that have non-trivial impact on the performance of concurrent data structures are then presented, such as the intricacies of cache coherence on NUMA systems and the interplay between the cache coherent protocol and synchronization primitives.

A hash table (hash map) is a data structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. Hash collisions are unavoidable when hashing a random subset of a large set of possible keys. The chained and open addressing hashing are two common strategies to avoid hash collisions.

A chained hash table indexes into an array of pointers to the heads of linked lists. Each linked list cell has the key for which it was allocated and the value which was inserted for that key. To lookup a particular element from its key, the key's hash is used to work out which linked list to follow, and then that particular list is traversed to find the element. The disadvantage of chained hashing is that following pointers to search the linked list consumes more memory. The advantage is that the chained hash is only linearly slower as the load factor (the ratio of elements in the hash table to the length of the bucket array) increases, even if it is large than 1.

An open-addressing hash table indexes into an array of pointers to pairs of key/value. If there are hash collisions in the hash table, certain schemes are needed to find another slot instead. Open-addressing is usually faster than chained hashing when the load factor is low because it does not need to follow pointers between list nodes. However, it will become slower as the load factor is close to 1. In addition, the load factor of open addressing is always less than 1.

A **Concurrent Hash Table** (CHT) is a hash table that allows multiple readers and writers (or multiple readers and single writer) to access shared objects concurrently. Like its sequential counterpart, a CHT not only offers the same set of APIs, but can exert the performance of multiprocessors more efficiently. Arbitrating concurrent accesses is a necessity for all concurrent data structures. Lock-based and lock-free synchronization are two commonly used concurrency programming model. For lock-based CHTs, critical sections are protected by locks to ensure thread-safety. Coarse-grained locking is relatively easy to implement, while preventing more efficient utilization of computing resources. With fine-grained locking, multiple threads are allowed to operate on different partitions of the data concurrently. Finer granularity is beneficial to improve the overall performance but at the cost of more implementation endeavors. Lock-free is another concurrency programming paradigm without using explicit locks. Lock-free CHTs are also widely proposed [12,13].

In pursuing high performance concurrent data structures, hardware support for synchronization is also a main challenge. In multi-processor multi-core environments, to maintain data consistency, hardware cache-coherence is often needed to ensure the consistency of accessing shared data from different cores. A cache-coherence protocol maintains state transitions on load, store, and atomic instructions (i.e., CAS and FAI). For example, a protocol may choose different update and invalidation transitions such as update-on-read, update-on-write, invalidate-on-read, or