学校代号	10532	学　　号	S18100001
分 类 号	TP391	密　　级	普通

湖南大學
HUNAN UNIVERSITY

硕士学位论文

基于某某技术的某某研究与实现

学位申请人姓名	郭靖
培　养　单　位	信息科学与工程学院
导师姓名及职称	洪七公 教授
学　科　专　业	计算机科学与技术
研　究　方　向	并行分布式系统
论文提交日期	二〇二 x 年 x 月 xx 日

学校代号: 10532

学　　号: S18100001

密　　级: 普通

湖南大学硕士学位论文

基于某某技术的某某研究与实现

学位申请人姓名: 郭靖

培　养　单　位: 信息科学与工程学院

导师姓名及职称: 洪七公 教授

专　业　名　称: 计算机科学与技术

论文提交日期: 二〇二 x 年 x 月 xx 日

论文答辩日期: 二〇二 X 年 x 月 xx 日

答辩委员会主席: 待定

Study on XX.

By

JING Guo

B.E. (Hunan University)201x

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of engineering

in

Computer Science and Technology

in the

Graduate School

of

Hunan University

Supervisor

Professor QIGONG Hong

June, 2020

湖 南 大 学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖南大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1、保密 □，在_____ 年解密后适用本授权书。

2、不保密 □。

（请在以上相应方框内打“✓”）

作者签名：　　日期：　　年　　月　　日

导师签名：　　日期：　　年　　月　　日

摘　要

关键词：关键字 1；关键字 2；关键字 n

Abstract

Key Words: Keyword1; Keyword2; Keywordn

目　录

学位论文原创性声明和学位论文版权使用授权书 ····· **I**
摘　要 ····· **II**
Abstract ····· **III**
插图索引 ····· **VI**
附表索引 ····· **VII**
第 1 章　绪　论 ····· **1**
1.1　多核系统概述 ····· 1
1.1.1　多核体系结构 ····· 1
1.2　选题背景与意义 ····· 1
1.3　本文主要工作 ····· 1
1.4　本文组织结构 ····· 1
第 2 章　第二章标题 ····· **2**
2.1　2.1 节 ····· 2
2.1.1　2.1.1 子节 ····· 2
2.1.2　2.1.2 子节 ····· 2
2.2　2.2 节 ····· 2
第 3 章　基于多核系统的并发哈希表的评估与分析 ····· **3**
3.1　实现并发哈希表的同步方法比较 ····· 3
3.2　典型的并发哈希表 ····· 3
3.2.1　缓存行哈希表 ····· 3
3.2.2　Cuckoo 哈希表 ····· 3
3.2.3　线程扩展性 ····· 3
第 4 章　章标题 ····· **4**
4.1　节标题 ····· 4
第 5 章　章标题 ····· **5**
5.1　节标题 ····· 5
5.1.1　子节标题 ····· 5
5.1.2　最优哈希函数个数 ····· 6
5.1.3　加锁与解锁 ····· 6
总结与展望 ····· **8**

参考文献 9
附录 A　读学位期间所发表的学术论文 16
附录 B　读学位期间所参加的科研项目 17
致　谢 18

插图索引

附表索引

第 1 章　绪　论

1.1　多核系统概述

1.1.1　多核体系结构

多处理技术的重要性体现在以下几个方面:

- 2000 年至 2005 年期间，研究人员在寻找和利用更高的指令级并行期间发现，功耗和硅成本的增长速度远超性能增长的速度，更高的指令级并行理论意义大于实际意义。除了指令级并行之外，另一条为人熟知的可能比基础技术具有更高性能的方法便是通过多处理 (Multiprocessing)。
- 云计算和软件即服务 (saas) 对高端服务器的需求越来越大;
- 互联网上的海量数据刺激数据密集型应用的增长;
- 有观点认为提升桌面电脑性能相比之下不再那么重要 (至少在图形处理方面)，要么是因为当前的桌面电脑满足性能需求，要么是因为高强度的计算密集型和数据密集型应用可以通过云计算完成。
- 人们对于如何有效的使用多处理器的认识的加深。

1.2　选题背景与意义

1.3　本文主要工作

本文主要着眼主流多核处理器架构上的并发哈希表的优化、设计与应用研究。

1.4　本文组织结构

本文分五个章节展开，各章大体内容安排如下:

第 1 章

第 2 章

第 3 章

第 4 章

第 2 章　第二章标题

2.1　2.1 节

2.1.1　2.1.1 子节

2.1.2　2.1.2 子节

2.2　2.2 节

第 3 章　基于多核系统的并发哈希表的评估与分析

3.1　实现并发哈希表的同步方法比较

3.2　典型的并发哈希表

3.2.1　缓存行哈希表

3.2.2　Cuckoo 哈希表

3.2.3　线程扩展性

(1)

(2)

第 4 章　章标题

4.1　节标题

第 5 章　章标题

5.1　节标题

假阳性 (false positives) 也叫误判是指当前元素不在集合内，但由于哈希冲突的缘故存在其它元素被映射到部分相同 bit 位上，从而有一定的概率导致在判定该元素时认定其对应的所有位置都为 1，从而判定其在集合内，造成一次误判。这个概率本文称为误判率，误判率用假阳性 (false positives) 也叫误判是指当前元素不在集合内，但由于哈希冲突的缘故存在其它元素被映射到部分相同 bit 位上，从而有一定的概率导致在判定该元素时认定其对应的所有位置都为 1，从而判定其在集合内，造成一次误判。这个概率本文称为误判率，误判率用假阳性 (false positives) 也叫误判是指当前元素不在集合内，但由于哈希冲突的缘故存在其它元素被映射到部分相同 bit 位上，从而有一定的概率导致在判定该元素时认定其对应的所有位置都为 1，从而判定其在集合内，造成一次误判。这个概率本文称为误判率，误判率用假阳性 (false positives) 也叫误判是指当前元素不在集合内，但由于哈希冲突的缘故存在其它元素被映射到部分相同 bit 位上，从而有一定的概率导致在判定该元素时认定其对应的所有位置都为 1，从而判定其在集合内，造成一次误判。这个概率本文称为误判率，误判率用

5.1.1　子节标题

定义 5.1　假阳性 (false positives) 也叫误判是指当前元素不在集合内，但由于哈希冲突的缘故存在其它元素被映射到部分相同 bit 位上，从而有一定的概率导致在判定该元素时认定其对应的所有位置都为 1，从而判定其在集合内，造成一次误判。这个概率本文称为误判率，误判率用ϵ 表示。

$$p_1 = 1 - 1/m \tag{5.1}$$

则 k 个哈希函数中都没有一个对其置位的概率 p_2：

$$p_2 = {p_1}^k \tag{5.2}$$

如果插入 n($n \leqslant m$) 个元素，但都未对其置位的概率 p_3:

$$p_3 = {p_2}^n = {p_1}^{kn} \tag{5.3}$$

则此位被置位的概率 p_4 为:

$$p_4 = 1 - (1 - 1/m)^{kn} \tag{5.4}$$

现在考虑判定阶段，若对应某个判定元素的 k 个位全部置位为 1，则可判定其在集合中。因此将某元素误判的概率为:

$$\epsilon = \left(1 - (1 - 1/m)^{kn}\right)^k \tag{5.5}$$

由 $(1 + x)^{1/x} \approx e$ 可知，但 m 很大时，满足 $-1/m \to 0$，可将公式5.4转化为:

$$\epsilon = \left(1 - (1 - 1/m)^{-m\frac{-kn}{m}}\right)^k \approx \left(1 - e^{-\frac{nk}{m}}\right)^k \tag{5.6}$$

由公式5.6可以初步断定ϵ 与元素的个数 n 和位数组的长度 m 决定，增大 n 或者减小 m 都会导致ϵ 的升高。m 和 n 的比值对应第 **??**的空间开销。这种计算方法不严格，因为前面假设哈希函数和散列后值的分布是相互独立的。但是，这个假设随着 m 和 n 的增大误判率更接近真实的误判率。Mitzenmacher 证明无假设情况下的误判率的期望值相同[118]。

5.1.2 最优哈希函数个数

5.1.3 加锁与解锁

接下来的内容介绍如何在论文中插入伪代码，仅供参考。

算法 5.1: 基于 Intel RTM 的 MCS 锁算法

```
1  struct {
2      mcs_lock_t lock;
3      uint32_t pad1[128/4 - sizeof(mcs_lock_t)];
4  } spec_mcs_lock_t;
5  struct {
6      mcs_lock_t mcs;
7      uint8_t mode;
8      char padding[];
9  } locklib_mutex_t;
10 int locklib_mutex_lock (locklib_mutex_t  *mutex, uint8_t mode){
11     spec_mcs_lock_t  *lock  =  (spec_mcs_lock_t  *)  mutex;
12     uint32_t reason = 0;
13 speculative_path:
14     XBEGIN(fallback_path, reason);
15     if  (lock->lock)  XABORT(1);
16         return 0;
17 fallback_path:
18     retries++;
19     while  (lock->lock)
20         cpu_relax();
21     if (retries < MAX_RETRIES)
22         goto speculative_path;
          /*  标准方式申请锁                                                    */
23      my_node.locked = true;
24     qnode_t *prev = __sync_lock_TAS(&lock->lock, &my_node);
25     if  (unlikely(prev != NULL)) {
26         prev->next = &my_node;
27         while  (my_node.locked)
28             cpu_relax();
29     }
30     mutex->mode = mode;
31     return 0;
32 }
```

总结与展望

1. 本文工作总结

1.

2.

3.

4.

2. 下一步工作展望

参考文献

[1] Hennessy J L, Patterson D A. Computer architecture: a quantitative approach. Elsevier, 2011

[2] Geer D. Chip makers turn to multicore processors. Computer, 2005, 38(5):11–13

[3] Marr D, Binns F, Hill D, et al. Hyper-threading technology in the netburst® microarchitecture. 14th Hot Chips, 2002.

[4] Samson E C, Machiroutu S V, Chang J Y, et al. Interface Material Selection and a Thermal Management Technique in Second-Generation Platforms Built on Intel® Centrino™ Mobile Technology. Intel Technology Journal, 2005, 9(1)

[5] Lempel O. 2nd Generation Intel® Core Processor Family: Intel® Core i7, i5 and i3. In: Proc of Hot Chips 23 Symposium (HCS), 2011 IEEE. IEEE, 2011, 1–48

[6] Chang J, Huang M, Shoemaker J, et al. The 65-nm 16-MB shared on-die L3 cache for the dual-core Intel Xeon processor 7100 series. IEEE Journal of Solid-State Circuits, 2007, 42(4):846–852

[7] Kongetira P, Aingaran K, Olukotun K. Niagara: A 32-way multithreaded sparc processor. IEEE micro, 2005, 25(2):21–29

[8] Shi L, Chen H, Sun J, et al. vCUDA: GPU-accelerated high-performance computing in virtual machines. IEEE Transactions on Computers, 2012, 61(6):804–816

[9] Bolla R, Bruschi R. An effective forwarding architecture for SMP Linux routers. In: Proc of Telecommunication Networking Workshop on QoS in Multiservice IP Networks, 2008. IT-NEWS 2008. 4th International. IEEE, 2008, 210–216

[10] Giesen F. Cache coherency primer. Website. https://fgiesen.wordpress.com/2014/07/07/cache-coherency/

[11] Liu Y, Zhang K, Spear M. Dynamic-sized nonblocking hash tables. In: Proc of Proceedings of the 2014 ACM symposium on Principles of distributed computing. ACM, 2014, 242–251

[12] David T, Guerraoui R, Trigonakis V. Asynchronized Concurrency: The Secret to Scaling Concurrent Search Data Structures. SIGARCH Comput. Archit. News, 2015, 43(1):631–644

[13] Shalev O, Shavit N. Split-ordered lists: Lock-free extensible hash tables. Journal of the ACM (JACM), 2006, 53(3):379–405

[14] Li X, Andersen D G, Kaminsky M, et al. Algorithmic Improvements for Fast Concurrent Cuckoo Hashing. In: Proc of Proceedings of the Ninth European Conference on Computer Systems. New York, NY, USA: ACM, 2014, 27:1–27:14

[15] Herlihy M, Shavit N, Tzafrir M. Hopscotch hashing. In: Proc of Distributed Computing. Springer, 2008: 350–364

[16] Metreveli Z, Zeldovich N, Kaashoek M F. Cphash: A cache-partitioned hash table. In: Proc of ACM SIGPLAN Notices, volume 47. ACM, 2012, 319–320

[17] Baumann A, Barham P, Dagand P E, et al. The multikernel: a new OS architecture for scalable multicore systems. In: Proc of Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009, 29–44

[18] David T, Guerraoui R, Trigonakis V. Everything You Always Wanted to Know About Synchronization but Were Afraid to Ask. In: Proc of Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. New York, NY, USA: ACM, 2013, 33–48

[19] Desnoyers M, McKenney P E, Stern A S, et al. User-level implementations of read-copy update. Parallel and Distributed Systems, IEEE Transactions on, 2012, 23(2):375–382

[20] Herlihy M P, Wing J M. Linearizability: A correctness condition for concurrent objects. ACM Transactions on Programming Languages and Systems (TOPLAS), 1990, 12(3):463–492

[21] Intel. Intel® transactional synchronization extensions. Website. https://software.intel.com/en-us/node/524022

[22] McKenney P E. Kernel korner: using RCU in the Linux 2.5 kernel. Linux Journal, 2003, 2003(114):11

[23] McKenney P E, Sarma D, Soni M. Scaling dcache with RCU. Linux Journal, 2004, 2004(117):3

[24] McKenney P E, Boyd-Wickizer S, Walpole J. RCU usage in the linux kernel: One decade later. Technical report, 2013.

[25] Yu M, Fabrikant A, Rexford J. BUFFALO: Bloom filter forwarding architecture for large organizations. In: Proc of Proceedings of the 5th international conference on Emerging networking experiments and technologies. ACM, 2009, 313–324

[26] Dharmapurikar S, Krishnamurthy P, Taylor D E. Longest prefix matching using bloom filters. In: Proc of Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. ACM, 2003, 201–212

[27] Bonomi F, Mitzenmacher M, Panigrah R, et al. Beyond bloom filters: from approximate membership checks to approximate state machines. In: Proc of ACM SIGCOMM Computer Communication Review, volume 36. ACM, 2006, 315–326

[28] Song H, Dharmapurikar S, Turner J, et al. Fast hash table lookup using extended bloom filter: an aid to network processing. ACM SIGCOMM Computer Communication Review, 2005, 35(4):181–192

[29] Jokela P, Zahemszky A, Esteve Rothenberg C, et al. LIPSIN: line speed publish/subscribe inter-networking. ACM SIGCOMM Computer Communication Review, 2009, 39(4):195–206

[30] Broder A, Mitzenmacher M. Network applications of bloom filters: A survey. Internet mathematics, 2004, 1(4):485–509

[31] Fan L, Cao P, Almeida J, et al. Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking (TON), 2000, 8(3):281–293

[32] Bonomi F, Mitzenmacher M, Panigrahy R, et al. An improved construction for counting bloom filters.

In: Proc of ESA, volume 6. Springer, 2006, 684–695

[33] Bender M A, Farach-Colton M, Johnson R, et al. Don't thrash: how to cache your hash on flash. Proceedings of the VLDB Endowment, 2012, 5(11):1627–1637

[34] Haberman J. State of the hash functions. Website. http://blog.reverberate.org/2012/01/state-of-hash-functions-2012.html

[35] Appleby A. MurmurHash. Website. https://sites.google.com/site/murmurhash/

[36] Geoff Pike J A. Cityhash. Website. https://opensource.googleblog.com/2011/04/introducing-cityhash.html

[37] Jenkins B. Hash functions. Dr Dobbs Journal, 1997, 22(9):107–+

[38] Jenkins B. Function for producing 32bit hashes for hash table lookup, 2006

[39] Jenkins B. Spookyhash: a 128-bit noncryptographic hash, 2012

[40] Knuth D E. The art of computer programming: sorting and searching, volume 3. Pearson Education, 1998

[41] Heileman G L, Luo W. How Caching Affects Hashing.. In: Proc of ALENEX/ANALCO. 2005, 141–154

[42] Pagh R, Rodler F F. Cuckoo hashing. Journal of Algorithms, 2004, 51(2):122–144

[43] Erlingsson U, Manasse M, McSherry F. A cool and practical alternative to traditional hash tables. In: Proc of Proc. 7th Workshop on Distributed Data and Structures (WDAS'06). 2006

[44] Ross K A. Efficient hash probes on modern processors. In: Proc of Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. IEEE, 2007, 1297–1301

[45] Black J R, Martel C U, Qi H. Graph and Hashing Algorithms for Modern Architectures: Design and Performance.. In: Proc of Algorithm Engineering. 1998, 37–48

[46] Agarwal A, Cherian M. Adaptive backoff synchronization techniques, volume 17. ACM, 1989

[47] Anderson T E. Performance implications of spin-waiting alternatives for shared-memory multiprocessors. In: Proc of Proceedings of the 1989 International Conference on Parallel Processing. Publ by IEEE, 1989

[48] Graunke G, Thakkar S. Synchronization algorithms for shared-memory multiprocessors. Computer, 1990, 23(6):60–69

[49] Mellor-Crummey J M, Scott M L. Algorithms for scalable synchronization on shared-memory multiprocessors. ACM Transactions on Computer Systems (TOCS), 1991, 9(1):21–65

[50] Craig T. Building FIFO and priorityqueuing spin locks from atomic swap. Technical report, Technical Report TR 93-02-02, University of Washington, 02 1993.(ftp tr/1993/02/UW-CSE-93-02-02. PS. Z from cs. washington. edu), 1993

[51] Magnusson P, Landin A, Hagersten E. Queue locks on cache coherent multiprocessors. In: Proc of Parallel Processing Symposium, 1994. Proceedings., Eighth International. IEEE, 1994, 165–171

[52] Scott M L. Non-blocking timeout in scalable queue-based spin locks. In: Proc of Proceedings of the

twenty-first annual symposium on Principles of distributed computing. ACM, 2002, 31–40

[53] Scott M L, Scherer W N. Scalable queue-based spin locks with timeout. In: Proc of ACM SIGPLAN Notices, volume 36. ACM, 2001, 44–52

[54] Michael M M, Scott M L. Nonblocking algorithms and preemption-safe locking on multiprogrammed shared memory multiprocessors. journal of parallel and distributed computing, 1998, 51(1):1–26

[55] Mellor-Crummey J M, Scott M L. Scalable reader-writer synchronization for shared-memory multiprocessors. In: Proc of ACM SIGPLAN Notices, volume 26. ACM, 1991, 106–113

[56] Krieger O, Stumm M, Unrau R, et al. A fair fast scalable rea, der-writer lock. In: Proc of Parallel Processing, 1993. ICPP 1993. International Conference on, volume 2. IEEE, 1993, 201–204

[57] Brooks E D. The butterfly barrier. International Journal of Parallel Programming, 1986, 15(4):295–307

[58] Hensgen D, Finkel R, Manber U. Two algorithms for barrier synchronization. International Journal of Parallel Programming, 1988, 17(1):1–17

[59] Mellor-Crummey J, Scott M. Fast, Contention-Free Combining Tree Barriers. 1992.

[60] Tseng Y L, Huang K H, Lai B C C. Scalable mutli-layer barrier synchronization on NoC. In: Proc of VLSI Design, Automation and Test (VLSI-DAT), 2016 International Symposium on. IEEE, 2016, 1–4

[61] Dijkstra E W, Scholten C S. Termination detection for diffusing computations. Information Processing Letters, 1980, 11(1):1–4

[62] Solihin Y. Fundamentals of Parallel Multicore Architecture. CRC Press, 2015

[63] Lamport L. A new solution of Dijkstra's concurrent programming problem. Communications of the ACM, 1974, 17(8):453–455

[64] Herlihy M. Wait-free synchronization. ACM Transactions on Programming Languages and Systems (TOPLAS), 1991, 13(1):124–149

[65] Herlihy M, Luchangco V, Moir M. Obstruction-free synchronization: Double-ended queues as an example. In: Proc of Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on. IEEE, 2003, 522–529

[66] Herlihy M, Luchangco V, Moir M, et al. Software transactional memory for dynamic-sized data structures. In: Proc of Proceedings of the twenty-second annual symposium on Principles of distributed computing. ACM, 2003, 92–101

[67] Herlihy M P. Impossibility and universality results for wait-free synchronization. In: Proc of Proceedings of the seventh annual ACM Symposium on Principles of distributed computing. ACM, 1988, 276–290

[68] Fich F, Hendler D, Shavit N. On the inherent weakness of conditional synchronization primitives. In: Proc of Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing. ACM, 2004, 80–87

[69] Kogan A, Petrank E. Wait-free queues with multiple enqueuers and dequeuers. In: Proc of ACM

SIGPLAN Notices, volume 46. ACM, 2011, 223–234

[70] Michael M M, Scott M L. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In: Proc of Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing. ACM, 1996, 267–275

[71] Kogan A, Petrank E. A methodology for creating fast wait-free data structures. In: Proc of ACM SIGPLAN Notices, volume 47. ACM, 2012, 141–150

[72] Timnat S, Petrank E. A practical wait-free simulation for lock-free data structures. In: Proc of ACM SIGPLAN Notices, volume 49. ACM, 2014, 357–368

[73] Brecht T. On the importance of parallel application placement in NUMA multiprocessors. In: Proc of Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS IV). 1993, 1–18

[74] Lachaize R, Lepers B, Quéma V. MemProf: a memory profiler for NUMA multicore systems. In: Proc of ATC-USENIX Annual Technical Conference. 2012

[75] Dashti M, Fedorova A, Funston J, et al. Traffic management: a holistic approach to memory placement on NUMA systems. In: Proc of ACM SIGPLAN Notices, volume 48. ACM, 2013, 381–394

[76] Tam D, Azimi R, Stumm M. Thread clustering: sharing-aware scheduling on SMP-CMP-SMT multiprocessors. In: Proc of ACM SIGOPS Operating Systems Review, volume 41. ACM, 2007, 47–58

[77] Tang L, Mars J, Zhang X, et al. Optimizing Google's warehouse scale computers: The NUMA experience. In: Proc of High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on. IEEE, 2013, 188–197

[78] Bull J M, Johnson C. Data distribution, migration and replication on a cc-NUMA architecture. In: Proc of Proceedings of the fourth European workshop on OpenMP. 2002

[79] Blagodurov S, Zhuravlev S, Fedorova A, et al. A case for NUMA-aware contention management on multicore systems. In: Proc of Proceedings of the 19th international conference on Parallel architectures and compilation techniques. ACM, 2010, 557–558

[80] Lepers B, Quéma V, Fedorova A. Thread and Memory Placement on NUMA Systems: Asymmetry Matters.. In: Proc of USENIX Annual Technical Conference. 2015, 277–289

[81] Moir M, Shavit N. Concurrent Data Structures., 2004

[82] Kung H T, Robinson J T. On optimistic methods for concurrency control. ACM Transactions on Database Systems (TODS), 1981, 6(2):213–226

[83] Herlihy M, Moss J E B. Transactional memory: Architectural support for lock-free data structures, volume 21. ACM, 1993

[84] Rajwar R, Goodman J R. Speculative lock elision: Enabling highly concurrent multithreaded execution. In: Proc of Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture. IEEE Computer Society, 2001, 294–305

[85] Rajwar R, Goodman J R. Transactional lock-free execution of lock-based programs. In: Proc of ACM SIGOPS Operating Systems Review, volume 36. ACM, 2002, 5–17

[86] Spear M F. Lightweight, robust adaptivity for software transactional memory. In: Proc of Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures. ACM, 2010, 273–283

[87] Saha B, Adl-Tabatabai A R, Hudson R L, et al. McRT-STM: a high performance software transactional memory system for a multi-core runtime. In: Proc of Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming. ACM, 2006, 187–197

[88] Shavit N, Touitou D. Software transactional memory. Distributed Computing, 1997, 10(2):99–116

[89] Yen L, Bobba J, Marty M R, et al. LogTM-SE: Decoupling hardware transactional memory from caches. In: Proc of High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on. IEEE, 2007, 261–272

[90] Moore K E, Bobba J, Moravan M J, et al. LogTM: log-based transactional memory.. In: Proc of HPCA, volume 6. 2006, 254–265

[91] Dalessandro L, Carouge F, White S, et al. Hybrid norec: A case study in the effectiveness of best effort hardware transactional memory. ACM SIGARCH Computer Architecture News, 2011, 39(1):39–52

[92] Cain H W, Michael M M, Frey B, et al. Robust architectural support for transactional memory in the power architecture. In: Proc of International Symposium on Computer Architecture. 2013, 225–236

[93] Wang A, Gaudet M, Wu P, et al. Evaluation of Blue Gene/Q hardware support for transactional memories. 2012. 127–136

[94] Intel R. Intel R 64 and IA-32 Architectures. Software Developer's Manual. 2015.

[95] Afek Y, Levy A, Morrison A. Software-improved hardware lock elision. 2014, 212–221

[96] Wang Z, Qian H, Li J, et al. Using restricted transactional memory to build a scalable in-memory database. In: Proc of Proceedings of the Ninth European Conference on Computer Systems. ACM, 2014, 26

[97] Wei X, Shi J, Chen Y, et al. Fast in-memory transaction processing using RDMA and HTM. In: Proc of Proceedings of the 25th Symposium on Operating Systems Principles. ACM, 2015, 87–104

[98] Chen Y, Wei X, Shi J, et al. Fast and general distributed transactions using RDMA and HTM. In: Proc of Proceedings of the Eleventh European Conference on Computer Systems. ACM, 2016, 26

[99] Wang X, Zhang W, Wang Z, et al. Eunomia: Scaling Concurrent Search Trees under Contention Using HTM. In: Proc of Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM, 2017, 385–399

[100] Wang Z, Qian H, Chen H, et al. Opportunities and pitfalls of multi-core scaling using hardware transaction memory. In: Proc of Proceedings of the 4th Asia-Pacific Workshop on Systems. ACM, 2013, 3

[101] Bloom B H. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7):422–426

[102] 吴军. 数学之美, volume 5. 人民邮电出版社, 2012

[103] Fan L, Cao P, Almeida J, et al. Summary cache: A scalable wide-area web cache sharing protocol. In: Proc of ACM SIGCOMM Computer Communication Review, volume 28. ACM, 1998, 254–265

[104] Putze F, Sanders P, Singler J. Cache-, hash-and space-efficient bloom filters. Experimental Algorithms, 2007. 108–121

[105] Mitzenmacher M D, Vocking B. The asymptotics of selecting the shortest of two, improved. 1999.

[106] Fan B, Andersen D G, Kaminsky M. MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing.. In: Proc of NSDI, volume 13. 2013, 385–398

[107] Intel. Threading Building Blocks. Website. https://www.threadingbuildingblocks.org

[108] Oracle. Java 7 EE. Website. https://www.oracle.com

[109] R.Pagh, E.F.Rodler. Cuckoo Hashing. Journal of Algorithms, 2004, 51(2):122–144

[110] McKenney P E, Slingwine J D. Read-copy update: Using execution history to solve concurrency problems. In: Proc of Parallel and Distributed Computing and Systems. 1998, 509–518

[111] liburcu. User-level Read-copy Update. Website. http://liburcu.org

[112] Intel. Intel 64 and IA-32 architectures software de- veloper's manual. 2016.

[113] Trigonakis V. SSPFD. Website. https://github.com/trigonak/sspfd

[114] Mazouz A, Touati S A A, Barthou D. Performance evaluation and analysis of thread pinning strategies on multi-core platforms: Case study of spec omp applications on intel architectures. In: Proc of High Performance Computing and Simulation (HPCS), 2011 International Conference on. IEEE, 2011, 273–279

[115] Dice D, Herlihy M, Lea D, et al. Applications of the adaptive transactional memory test platform. Applications of the Adaptive Transactional Memory Test Platform Researchgate, 2008.

[116] Intel 64 and IA-32 Architectures Optimization Reference Manual. Order Number, 2011.

[117] Bobba J, Moore K E, Volos H, et al. Performance pathologies in hardware transactional memory. In: Proc of International Symposium on Computer Architecture. 2007, 81–91

[118] Mitzenmacher M. Compressed bloom filters. IEEE/ACM transactions on networking, 2002, 10(5):604–612

[119] Fan B, Andersen D G, Kaminsky M, et al. Cuckoo filter: Practically better than bloom. In: Proc of Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies. ACM, 2014, 75–88

附录 A　读学位期间所发表的学术论文

1.
2.
3.

附录 B　读学位期间所参加的科研项目

1. A A A A A A A A A
2. A A A A A A A A A
3. A A A A A A A A A

致　谢

TODO：1 页