Acta Technica 62 (2017), No. 5A, 63–72 © 2017 Institute of Thermomechanics CAS, v.v.i.

Repeat-hash collision detection algorithm and map log sheet conflict evading algorithm based on multi-core pc

Ting Zhang[1], Cong-Cong Li[2], Ming Qiu[3,4]

Abstract. Against the hash collision detection algorithm is easy to read and write different address mapping to the same hash address, produce the problem such as "false conflict", puts forward a Repeat-Hash collision detection algorithm, using hash method to calculate multiple hash address, reduce unnecessary costs of rollback by the misjudgment in transaction. This paper analyzes consistency check of transaction and threads, gives the parallel algorithm and the application example. The experimental results show that the algorithm can well reflect the transaction memory collision detection in the actual operation process, it is an effective method to control the transaction memory system parallelly. The traditional transactional memory system can only deal with conflicts, but it is without the prevention of conflicts in advance. Thus, a conflict evading algorithm based on MAP log sheet is proposed. Before the transaction starts, the possibility of conflict occurrence can be predicted in accordance with the historic conflict situations, and the transaction can be regulated according to the prediction result, so as to reduce its failure rate. The evading for the read-write conflict between the transaction and thread is conducted along with the provision of parallel algorithm and application examples. Experimental results show that the algorithm can well reflect the actual operating procedure of the conflict evading of transaction memory, and it is considered effective in realizing the parallel control and operation of the transactional memory system.

Key words. Conflict detection algorithm, multicore pc, transactional memory, conflict evading algorithm.

[1]School of Computer Science and Engineering, Xiangsihu College of Guangxi University for Nationalities, Nanning 530008, China; e-mail: MAME2013@163.com

[2]College of mechanical and electrical engineering, Agricultural University of Hebei, Baoding, 071001, China; e-mail: hebaulcc@126.com

[3]Educational technology and information management center, Guangxi College of Education, Nanning 530023, China

[4]Corresponding author: Ming Qiu; e-mail: 1040168586@qq.com

1. Introduction of Repeat-Hash Collision Detection Algorithm

Currently, data sharing is realized by the multithreading parallel programming design mainly through locks or semaphore. The inter-multithreading operation becomes slow due to competitive locks, or it is likely to cause deadlock, priority inversion or other mistakes. To resolve shortcomings of the sharing data lock mechanism, the replacement of the lock mechanism with software transactional memory (STM) is initially proposed by Literature [1] in 1995. Later STM researches include DSTM and RSTM models. In 2006, Ceze et al. come up with an optimization scheme [2] which turns the computation of the transactional read-write operation address into a value similar to Hash in order to carry out collision detection through value contrast. In this way, the broadcast information amount on the bus is significantly reduced, but it is possible to lead to fake collision. The BSTM model is put forward by Literature [3]. In collision detection, the Hash table is taken as the data structure when detecting the consistency of read-write data, so as to decrease the search time in the data buffer. The simulation test indicates that the structure of Hash table can help to reduce the search time in case there are plenty of shared variables, so its performance is relatively advantageous. However, it is easy to result in performance worsening due to the fake collision caused by collision detection of the data structure. Thus, it is of great significance to improve the Hash collision detection algorithm, so as to settle the parallel programming lock mechanism issue of the multi-core PC effectively and give full play to the potential of multi-core PC.

2. Proposal of Hash collision detection algorithm and problems in transactional memory

2.1. Ideology of the Hash collision detection algorithm in transactional memory

Multi-items in transactional memory can be executed in parallel and visit the sharing object simultaneously. If two transactions visit the same object at the same time with the occurrence of the write-write operation or read-write operation, then collision will take place because the consistency of data finally submitted is destroyed.[4] The completion management strategy needs to be adopted in collisions in order to select a transaction for further execution with the termination of other transactions. Stopped transactions will be re-dispatched for execution at some subsequent time point [5].

Collision detection of transactional memory is divided into the intra-transaction data consistency detection and inter-transaction data consistency detection. Intra-transaction data is consistent, because the address in the write-buffer is detected first through the read-shared data operation, for example, the write-buffer address is detected by the read-buffer in transaction 1 of Figure 1. In case there is the same address, it will directly be read from the write-buffer, so as to ensure the intra-transaction data consistency. Regarding the inter-transaction consistency detection,

it is only needed to check the write-operation of submitted transactions and read-operation of transactions that haven't been submitted, as shown in Figure 2. The specific flow of collision detection is:

(1)Read operation

It is searched whether the address of variable x to be read is contained in the intra-transaction write-buffer. If the same address does exist, the latest data will be read from the write-buffer. In Figure 1, the variable x has not been read or written before, so its value 0 can be read from the corresponding actual memory address.

(2) Write operation

Before the write operation, it should be checked whether there is the same address in the shared data address of the detection buffer in the bottom transaction manager. If there is, the data should be updated, otherwise, new nodes should be inserted in the buffer. After modification, the modified value 1 and the variable data address &Adress should both be preserved in the write buffer, and they should not be submitted until transactions are completed.

(3)Submission of the transaction application

After the read-write operation of shared memory, the submission operation will be conducted for transactions. The submission right will be applied for from the transaction manager which will then make the arbitration and allow the submission of one transaction. Other transactions submitted will be in the waiting status, and transactions that have not been applied for submission can continue to be executed.

(4)The transaction submission stage

Data in the write buffer is written into the corresponding actual physical memory address by transaction 1 which has got the submission right, and the write-buffer is copied to the detection buffer of the transaction manager at the same time, so that the consistency check can be carried out for other transactions. In Figure 2, the value of the variable x address in the shared memory is updated to 1, meanwhile, the address and data of the variable x are also contained in the detection buffer of the transaction manager.

read-write address. The data address and data content need to be preserved in the read-write buffer, so the structure of each unit in Hash table includes three elements: data address, data value, and pointer of the next element among synonyms. Among them, the read-write address is the keyword of Hash table. By calculating some Hash function, the read-write address can be mapped on the Hash address which helps to compare the read-write addresses of transactions or threads. As shown in Figure 3, the address R2 for read operation maps the same Hash address as the address W2 for write operation, so the same address 1 is read and written. Long address data has turned to short Hash addresses via the Hash function computation with this algorithm. Thus, the comparison time is shortened, and it is verified by experiments of Literature [3] that the system performance can be improved significantly with this algorithm in contrast with others.

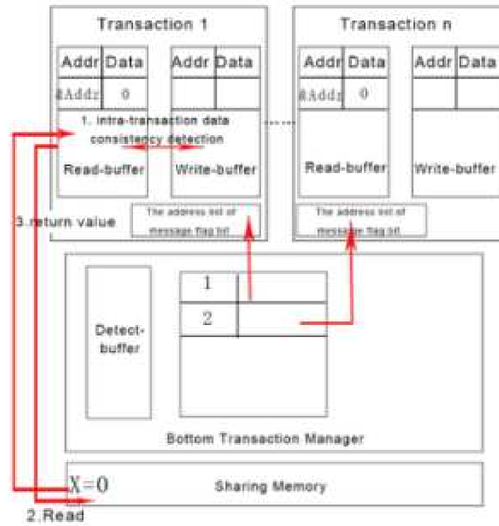

Fig. 1. The read operation flow when the variable x=1

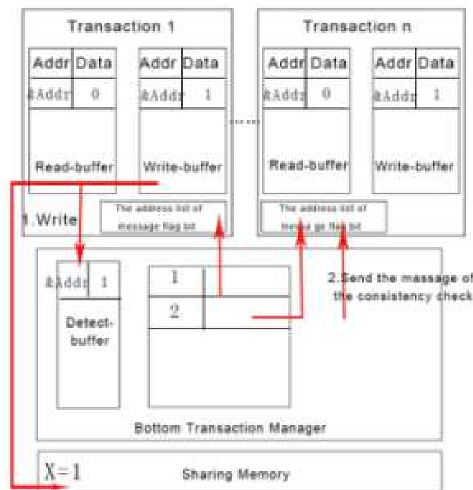

Fig. 2. The write operation flow when the variable x=0

2.2. Proposal of problems

However, in contrast to the method of comparing actual addresses one by one, the Hash collision detection way proposed by Literature [3] has collision risk, because the same Hash address may be a result of calculating different keywords. According to Figure 4, &Add1 &Add2, but f(&Add1)= f(&Add2). In other words, there is collision, thus resulting in "fake collision" between the read operation that has no

address collision before and the submitted write operation. In this way, transactions that should not have collision will roll back, causing unnecessary expenditure and affecting the system efficiency. Hash collision cannot be totally avoided, but its probability can be lowered as far as possible. For this, an improvement is made by us in accordance with the Hash data structure.

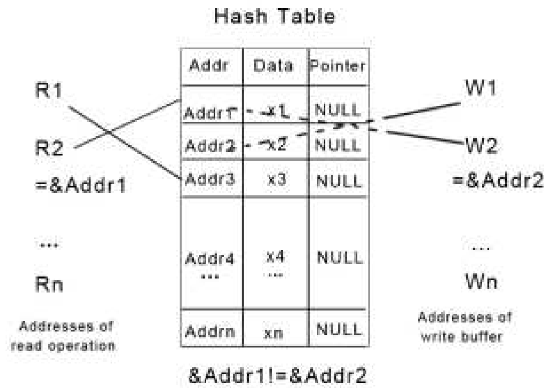

Fig. 3. The principle of hash collision detection

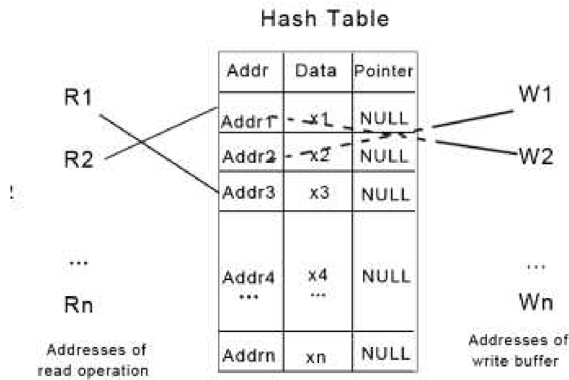

Fig. 4. "Fake collision" in hash collision detection

3. Repeated Hash collision detection in transactional memory

3.1. The mechanism of repeated Hash collision detection

Although the address comparison time is prolonged by the repeat-Hash algorithm, the time and expenditure on re-execution is rather insignificant in contrast to the rollback caused by misjudgment of numerous transactions.

In actual operation, the data size is large, the time N of multi-time Hash increases

as well, but the time N cannot be enlarged infinitely, otherwise, the processing rate can be affected. If it is too small, the load capacity will be decreased. Researches show that the search performance of the algorithm can be improved significantly by the multi-function Hash table.

3.2. Repeat-Hash algorithm design

Hash collision detection refers to the fast comparison of read-write addresses of various transactions (read-write addresses of threads) by taking them as the index. If it turns out the read-write address of two transactions is the same, it indicates there is collision and the competition management needs to be used to determine the transactional execution and rollback. On this basis, the Hash function time is introduced by Repeat-Hash algorithm, thus increasing the accuracy of data consistency check.

It is assumed that N random integers are produced randomly, the address space is m, the remainder use from prime method is taken to construct Hash function, and the prime database includes all prime sets smaller than m. The repeat-Hash algorithm is shown below:

Begin:

Input: define the maximum thread number, and generate N random numbers randomly.

Output: the first list of output parameters is the thread id, the second list is the Hash address, and the third is the collision times, and the operation time (millisecond) is output as well.

Step1: Use OMP_NUM_THREADS to define the maximum thread number in the execution process.

Step 2: Take one random number as the keyword in each thread, then the prime in the prime database is divided by this random number, and the obtained remainder will be the Hash address.

Step3: In the debug output process, the Hash address positioned by each thread is displayed. If the Hash address obtained by each thread differs, the keyword will be written in the Hash table address.

Step4: If two or more calculated Hash addresses are the same in the thread, continue to use the remainder of the prime number.

Step5: If the re-calculated Hash address is different, return to Step 3 and write the keyword in the Hash table address.

Step6: If the hash address obtained via N times of prime remainder use still differs, it is considered the read-write address is the same in the thread and there is collision, so there will be 1 more for the collision times.

4. Experimental results and comparative analysis

4.1. Experimental platforms and parameters

The hardware platform of this experiment is: nanometer Core i5-3450 4-core processor of Intel 22, 3.7GHz of CPU dominant frequency, and 8G memory. The software platform is: Microsoft Windows 7 operation system, Microsoft Visual Studio 2010 (OpenMP) +Intel Parallel Studio XE 2013, C++ language programming. To reflect advantages of multi-core PC, an Intel N280 mononuclear process is selected for comparative tests.

4.2. Experimental results and analysis

4.2.1. Test results of linear Hash (LH), repeated-Hash by linear (RHL), and repeated-Hash by Random (RHR) The method of linear Hash (LH) is adopted for linear detection rehash. There are two methods of obtaining primes through the repeated-Hash by linear, namely, the orderly prime repeated-Hash by linear (RHL) and the random prime repeated-Hash by random (RHR). According to following test results, each data is a mean value calculated by averaging the test results for 20 times, and test results are shown in Table 1:

Table 1. Test results of LH, RHL and RHR programs

4-core PC operation result					**Mononuclear PC operation result**
Algorithm name	**Average circulation time**	**Maximm circulation time**	**Minimum circulation time**	**Operation time / ms**	**Operation time / ms**
Linear detection Hash (LH)	3190596	3258442	3123751	4	17.3
Orderly prime repeated-Hash (RHL)	37911624	22256228	552926	7	72.5
Random prime repeated-Hash (RHR)	366065	408111	189297	13	104.7

In the above three algorithms of linear tests, the operation on multi-core PC is obviously faster than that on the mononuclear PC, embodying the advantage of multi-core. The operation time of linear Hash (LH) is the shortest, but the average circulation time of the random prime (RHR) is less than 1 magnitude order compared to the linear hash (LH), that is, it is 10 times of circulation less. It suggests that the average access-memory time can be clearly reduced with the RHR algorithm, thus reflecting the advantage of addressing. As for the order prime (RHL)

method that constructs Hash function via orderly prime extraction, there are two magnitude orders between the maximum circulation time and the minimum one, indicating the positioning performance of the orderly prime is unstable. When it is lucky, the circulation time will be small, on the contrary, 100 times of circulation may be needed. Better Hash performance is manifested with the random prime (RHR) algorithm with conducts further random selection on the basis of random calculation, so as to improve the overall algorithm performance significantly. Next, the random prime (RHR) method will be parallelized.

First, through the build-in performance analysis tool of Microsoft Visual Studio 2010, the performance analysis for RHR algorithm is conducted. Figure 5 shows the CPU occupation analysis report of the random prime (RHR) algorithm, and the function call time is one of the standards to evaluate the performance. The more time it needs for call, it is more likely to become the performance bottleneck. The code section with more time consumption in the program is analyzed in Table 2, and the next step of parallelized rewriting is conducted.

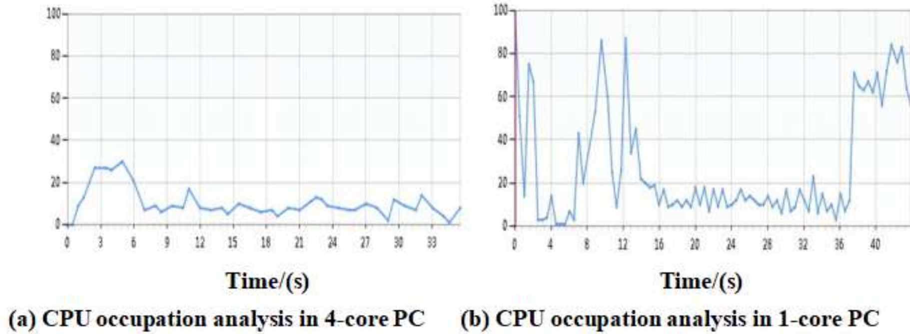

Fig. 5. The analytical report of cpu occupation (use rate) with the random prime (rhr) algorithm

Table 2. Performance analysis of the random prime (rhr) algorithm

Function name	**Non-monopolized sample number**	**Monopolized sample number**	**Non-monopolized sample percentage**	**Monopolized sample percentage**
_main	21	0	100	0
_rand	2	2	9.52	9.52
hash(int *,int)	4	0	19.5	0
hashMap1(int(*))	7	0	33.3	0
ran(int)	3	2	14.29	9.52

4.2.2. Test results of parallelized repeated-hash (rhr) OpenMP is used to compile the guiding statement and add "#pragma omp parallel for" before the paral-

leled statement. It is defined omp_get_thread_num(), and the used thread number is output. Through omp_set_num_threads (THREAD_NUM), the thread number is set in the subsequent paralleled zone. The paralleled part is executed by n threads, and the size n is realized by setting the value of the environmental variable omp_num_threads.

Here, n=4, n=8, and n=16 are set in turn, and 3, 7 and 15 threads are used at most with the paralleled random prime (RHR) algorithm, as shown in Figure 6. The utilization rate is higher than 75%.

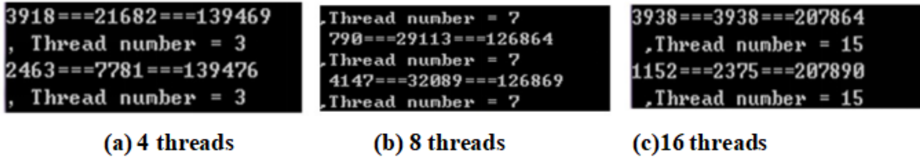

Fig. 6. Thread use conditions

When the thread number is 2, 4, 8 and 16, the operation time of the paralleled random prime (RHR) will be shown as Table 3:

Table 3. Performance analysis of the paralleled random prime (RHR) algorithm

Thread number	**Operation time of the paralleled random prime algorithm / ms**	**Operation time of the linear random prime algorithm / ms**	**Speed-up ratio**
2 threads	12.3	13	1.06
4 threads	11	13	1.18
8 threads	10.1	13	1.29
16 threads	9.7	13	1.34

When the thread number is n=2, 4, 8 and 16, the thread collision of paralleled RHR algorithm is shown in Figure 4:

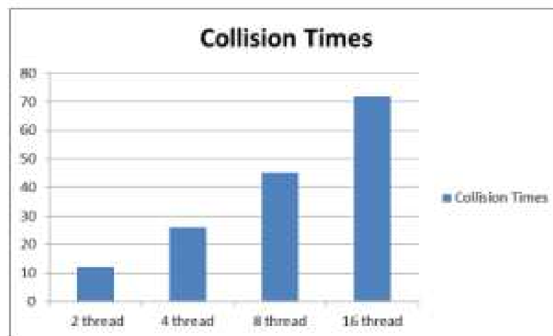

Fig. 7. Test results of the paralleled RHR program

4.2.3. Result analysis Seen from Table 3 and Table 4, when the thread number increases from 2 to 16 after parallelizing the random prime (RHR), the more threads there are, the shorter the operation time will be, but the collision times will increase dramatically. It implies that the inter-thread collision occurrence probability will be larger with the increase of the thread number. The algorithm advantage is reflected after parallelization, as shown in Table 3. The average operation time of the paralleled random prime (RHR) is 4ms larger than that of linear random prime (LH), and the maximum speed-up ratio is up to1.34, indicating great rate enhancement.

By making use of shorter Hash values to express the read-write address set, Hash collision detection algorithm is a very promising collision detection design scheme in transactional memory system. With the introduction of multiple Hash functions for many times, the accuracy of comparing multi-thread read-write addresses can be improved via the increase of re-constructing Hash function time N with the repeat-Hash algorithm. In this way, less collision can be produced when detecting large data sizes, thus reaching the purpose of parallelized settlement of application problems.

References

[1] M. F. Spear, V. J. Marathe, W. N. Scherer, M. L. Scott: *Conflict Detection and Validation Strategies for Software Transactional Memory, Distributed Computing.* Lecture Notes in Computer Science *4167* (2006).

[2] B. H. Bloom: *Time Trade-offs in Hash Coding with Allowable Errors.* Communications of the ACM (1970), 422–426.

[3] Y. P. Kang, Y. L. Tan, C. X. Liu, Y. R. Yu: *The optimization design of collision detection for barrage games.* Journal of Communication and Computer *6* (2009), No. 11, 8–11.

[4] G. Abbas, N. Asif, H. Grahn: *Performance Tradeoffs in Software Transactional Memory.* Master Thesis Computer Science Thesis (2010).

[5] J. C. Frank: *Adaptive software transactional memory: dynamic contention management.* Clinical Pediatrics *21* (2008), No. 11, 4427–4437.

[6] A. Barros, L. M. Pinho, P. M. Yomsi: *Non-preemptive and SRP-based fully-preemptive scheduling of real-time Software Transactional Memory.* Journal of Systems Architecture Sciences *61* (2015), No. 10, 553–566.

[7] M. M. Waliullah, P. Stenstrom: *Removal of Conflicts in Hardware Transactional Memory Systems.* International Journal of Parallel Programming *42* (2014), No. 1, 198–218.

[8] W. M. Yan, W. M. Wu: *Data Structure (C-Language).* JTsinghua (2013).

[9] C. S. Ananian, K. Asanovic, B. C. Kuszmaul: *Unbounded Transactional Memory.* IEEE Micro (2006).

[10] K. Moore, J. Bobba, MoravanM, LogTM: *log-based transactional memory.* High-Performance Computer Architecture (2006), 254–265.

Received November 16, 2017