Study on the application of elementary number theory in conflict algorithm

Zhi Zhao, Jinde Huang*
School of Mathematics and Information Science, Guangxi College of Education, Nanning, China

*Corresponding author e-mail: h_jinde@tom.com

Abstract. Analyze the time complexity of the algorithm, for multiple threads to execute multiple transactions at the same time, and map the result of the remainder of the prime number to the hash address. After calculation, the formula is a convergent array. The larger the value of x, the smaller the calculated value, indicating that the algorithm can complete the task in a certain limited time. Applying the probability of random prime numbers to the design of conflict management algorithms can avoid endless waiting for each other between transactions, and also avoid the waste of computing resources caused by cancelling earlier transactions. Compared with the radical algorithm of direct termination, this algorithm gives the transaction a certain waiting and retry time, avoids the long execution of the long transaction, and guarantees the commit time of the transaction.

Keywords: Elementary number theory, prime number, conflict algorithm, time.

1. Introduction
To prevent endless wait for the completion of hostile transactions by some transaction, the upper limit of waiting time is available by Polite algorithm. In case of conflicts detected, the transaction will wait for a while politely. The unit time is 2n. When the retry time n reaches the maximum 8, the competitive transaction will be suspended unconditionally with Polite algorithm [1]. However, there is a disadvantage of Polite algorithm, that is, it is unclear which transaction should wait, and it is likely that the transaction with huge workload have to wait for the other with small workload, thus resulting in the back-off of the former, the wast of numerous calculation resources, and the problem of priority inversion[2].

Therefore, which one of the two transactions should be selected for continuous running without wasting calculation resources is the key to algorithms. According to the Timestamp algorithm, the starting time of transactions is taken as the standard of determining whether the transactions should be stopped. In case of conflicts between transactions, the earlier one should continue to be executed, while the later one will be suspended [3]. In this way, suspension of the transaction which has already done much calculation work can be prevented, thus avoiding the waste of calculation resources. However, if the later transaction is suspended all the time with no execution, problems such as deadlock and priority inversion will arise [4].
We will combine advantages of the Hash conflict detection algorithm with the function of repeated detection and the conflict evading algorithm with the function of high and low frequency distinguishing are combined, plus strengths of the Polti contention management algorithm, the transactional memory conflict management algorithm Full-Polti based on mutli-core processors will be constructed[5].

2. Algorithm design
The Full-Polti algorithm is described as follows:
Begin:
Input: The maximum thread number is defined, N random numbers are generated, the high and low frequency log sheet is maintained, and previously conflicted address value and confliction time n are recorded.
Output: Output parameters. The first column is the thread id, the second is the hash address, the third is the times for conflict occurrence, and the runtime (ms) is output.
Step 1: OMP_NUM_THREADS is used to define the maximum thread number in execution.
Step 2: A random number is selected from each thread as the key word. First, the address is searched from the high and low frequency log sheet, and the possibility of showing conflicts by the key word is predicted. If the possibility is low, the Hash operation can be conducted for obtaining remainder from primes.
Step 3: In the conflict detection period, the Hash conflict detection algorithm with the function of repeated detection is used to detect the reading and writing address conflict between two transactions, which will then be submitted to the contention management for verdict.
Step 4: First, the contention manager makes the earlier transaction continue to be executed and the later transaction wait for the unit time of 2n. n is the retry times, and n=n+1.
Step 5: If n<8, and the waiting transaction is detected to be not in conflict with the contention transaction, then the former can continue to be executed.
Step 6: If n=8, then the waiting transaction will no longer wait for the application submission by the contention transaction, instead, it will stop the contention transaction directly and start to be executed itself.
Next, the time complexity of the algorithm will be analyzed. For the execution of N transactions at the same time by P threads, the calculation result of obtaining remainder from primes is reflected to Hash address. The address space quantity in the Hash table is C, where C>N. The time needed for task execution by a transaction is Tr, and the time of waiting by a transaction is Tw. Thus, the time complexity of Polti algorithm obtained via deduction is:

\[ O\left( N \sum_{x=0}^{8} \left( \frac{P^x(C-Px)}{C^{x+1}T} + \frac{P^xT}{C^{x+1}T_r} \right) \right) \]  \hspace{1cm} (1)

Where, \( \frac{P^xT}{C^{x+1}T_r} \) is the probability of a transaction waited for x times, that is, the probability of being circulated for x times.
\( \frac{P^xT}{C^{x+1}T_r} \) is the waiting time before x times of circulation, and the upper limit of x is 8.

To simplify the model, we assume the waiting time and execution time of the transaction is 1:1, so the above formula can be written into:

\[ O\left( \sum_{x=0}^{8} \left( \frac{NP^x(C-Px + 1)}{C^{x+1}} \right) \right) \]  \hspace{1cm} (2)
It is known by calculation that the formula is a convergent array. When the value \( x \) is larger, the calculated value will be smaller, which indicates that the task can be completed within limited time with this algorithm.

The design of Polti algorithm can help to avoid the endless mutual wait by transactions as well as the calculation resource waste caused by the revocation of the earlier transaction. In contrast to the radical algorithm of direct suspension, certain waiting and retry time for transactions is provided with this algorithm, thus shortening the execution time of long transactions and ensuring their submission time.

3. Realization of the algorithm
OpenMP is used partially by this paper at the program parallel domain for compilation and guidance. Serial programs are parallelized. The OpenMP application programming port API is a programming model on the sharing memory system structure, which contains a compiler directive, runtime library, and environment variables. Fork-Join is taken by OpenMP as the parallel execution model based on threads. The part that cannot be executed in parallel in the main program is executed by the main thread, and the part that can be executed in parallel is run at the same time by several threads. Execution results are combined at last. Codes contained in the Fork-Join structure is divided to threads in the group for execution, or parallel sections can be set up via compilation and guidance. Different tasks are executed by different threads, or a single is set to make one thread execute code sections.

The process of the multithreading parallel program is: \( N \) random numbers are generated, and \( n \) threads are used to execute the operation of obtaining remainder from primes. The detected addresses in conflicts and time records are written in the log sheet as the basis for conflict evading. When the conflict thread enters to the contention management stage, one transaction will continue to run in accordance with the contention management algorithm, while the other will be suspended or wait. After completion of calculation by the running thread, data application will be submitted. The suspended thread will roll back, and all calculations completed before will be abandoned along with re-execution. Once \( N \) random numbers are all finished with calculation, all of the data will be collected by JION for output together.

The OpenMP guidance statement adopted by this experiment can achieve the parallel synchronization and merging of multithreading. Environmental variables can be set by using “omp_num_threads”, so as to define the maximum thread number in operation. When “#pragma omp parallel for” is added to the parallel program executed simultaneously by multi threads, parallel statements in execution will automatically be parallelized, and the loop statement following it closely will be executed in parallel by the thread group. Regarding the setting of thread synchronization, “#pragma omp critical (name)” can be used to make the code block in the parallel domain execute only one thread each time. Other threads are blocked. This statement is used for thread wait of the contention management part. Barrier guidance statement can be used to synchronize all threads in one thread group. The barrier of this experiment is a hidden bar. After all threads in the parallel zone are completed, the main thread will continue to be executed. Finally, the execution result will be merged, and the clause “reduction (operator: list)” is used to merge variables in the thread at the structure end, thereby making sure the output result is correct.

4. Conclusion
Contention management is an operation before the application submission of transactions, which is specific to the result of conflict detection. After conflicts of several transactions occur, it should be explored about how to manage transactions in conflicts, which transaction to be executed and which to be rolled back. First, several contention management algorithms are researched and summarized by this paper. In the experiment, it attempts to integrate two algorithms named Plite and Timestamp and construct the Polti contention management algorithm. The Hash conflict detection algorithm with the function of repeated detection is combined with it, and the conflict evading algorithm with the function of high and low frequency differentiation is used for conflict prediction. Experimental results show that advantages of the conflict detection algorithm and conflict evading algorithm have been integrated by
the “three in one” “Full-Polti” conflict management algorithm, plus the combination of strengths of Polite and Timestamp, the algorithm can achieve superior performance. The finally output conflicts can reduce by nearly 50% compared to that with the non-prediction and non-management algorithm. The mean runtime of Full-Polti is 43012ms which is close to the theoretically calculated value. In case of big data scale, the superiority of the algorithm can be reflected more significantly.

Acknowledgements
This research work was supported by Guangdong Higher Education Teaching Revolution Project: Experiencing Teaching and Mode Design of Cultivation of Internet Finance Talents under the Background of Financial Science Technology, project number: 2018SJJXGG01; Guangzhou College of Commerce Higher Education Teaching Revolution Project: Teaching Reform and Practice of Securities Investment under the Cultivation Method of Applied Talents, project number: 2018XJJXGG01. Foundation projects: Guangdong Undergraduate Higher Education Teaching Quality Engineering project: “Guangzhou College of Commerce -Applied Training Base of Mobile Commerce and E-commerce Logistics, project number: SJJJD201501; Guangdong College Characteristic Innovative Project (humanities and social sciences category): “Virtual Reality Experimental Teaching Center of Guangzhou College of Commerce”, project number: 2015WTSCX114.

References
[1] Dongxu. Shi, Dynamic Contention Management Strategy of Software Transaction Memory, J. Software Journal. Vol.11, No.4 (2012) 6-8.
[2] Fei. Lin, Dynamic Contention Management Strategy of Software Transaction Memory, J. Computer Engineering and Design. Vol.31, No.7 (2010) 1510-1512.
[3] Ramadan H E, Rossbach C J, Porter D E, et al. MetaTM/TxLinux: Transactional Memory for an Operating System, J. IEEE Micro. 28 (2008) 42-51.
[4] Sanchez D. Design and Implementation of Signatures for Transactional Memory Systems, J. Technical Report CS-TR-2007-1611, 2007.
[5] Bloom B H. Space/Time Trade-offs in Hash Coding with Allowable Errors, J. Communications of the ACM. Vol.13, No.7 (1970) 422-426.