Journal of Computer Science 5 (11): 778-782, 2009
ISSN 1549-3636
© 2009 Science Publications

An Improved Lazy Release Consistency Model

Chapram Sudhakar and T. Ramesh
Department of Computer Science and Engineering,
National Institute of Technology, Warangal-506004, India

Abstract: Problem statement: A network of workstations, viewed as a distributed shared memory system can be used to develop and test parallel algorithms. **Approach:** For implementing parallel algorithms on such DSMs shared memory consistency model plays a vital role. **Results:** However on a LAN, strict consistency models like Sequential Consistency model (SC) are not useful since the communication is slow. In such environments relaxed models like Entry Consistency (EC), Release Consistency (RC) or their variations such as Lazy Release Consistency (LRC) are generally used. **Conclusion/Recommendations:** In this study an Improved Lazy Release Consistency (ILRC) model is proposed. This model is studied with standard parallel algorithms. In many cases the ILRC model is proved to work better than the LRC model.

Key words: Distributed shared memory system, lazy release consistency model, memory consistency model, threads

INTRODUCTION

Usage of network of workstations for parallel processing is very common. Such an environment can be viewed by the programmer as a message passing environment or Distributed Shared Memory (DSM)[7] environment. The shared memory view makes parallel programming easier by using threads concept, where as the message passing view makes development of efficient parallel programs containing explicit messaging calls for remote data items.

Providing specific memory consistency models for a distributed shared memory system is necessary for developing parallel programs. Several models in the literature are proposed which are categorized in to two categories[11] based on the data being accessed. First category is uniform models which will treat all kinds of data accesses uniformly. Strict[5], Sequential, PRAM, Processor and Causal consistency models[6,9,10,13] are some examples for uniform memory consistency models. The second category is synchronization models that differentiate the memory accesses as synchronization related accesses and normal data accesses. Weak, Release[2] and Entry consistency[3] models[6,9,10,13] are some examples for synchronization models. Synchronization based models are more relaxed than the uniform consistency models. There is a variation of Release Consistency model known as Lazy Release Consistency (LRC) model which is used in TreadMarks[4] system. This model works better for parallel algorithms which manipulate small set of data items in a brief critical section that results in very little modifications in the page of those data items. As the data set size and the number of processes increases the total differences for an interval of time also increases and hence this LRC model cannot perform efficiently. ILRC overcomes these problems by some modifications to LRC model.

In the next part original LRC model, its drawbacks and proposed improvements are described. Implementation details of proposed modifications and test results with standard parallel algorithms are presented in subsequent parts of this paper.

Background: Lazy Release Consistency Model ensures that all programs without data races behave as if they were executing on a conventional Sequentially Consistent (SC) memory. Most parallel programs satisfy this condition and behave identically when executed on a multiprocessor system and DSM system with LRC model. But compared to Sequential Consistency model LRC has the advantage that it can be implemented more efficiently. The TreadMarks implementation of LRC[4] is described below.

LRC divides the execution of each process into logical intervals that begin at each synchronization access. Synchronization accesses are classified as release or acquire accesses. Acquiring a lock is an

Corresponding Author: Chapram Sudhakar, Department of Computer Science and Engineering, NIT, Warangal-506004, India, Tel: 091-870-2462731, 2468731

example of an acquire access and releasing a lock is an example of a release access. Waiting on a barrier can be modeled as a release followed by an acquire. LRC defines the relation corresponds on synchronization accesses as follows: A release access on a lock corresponds to the next acquire on the lock to complete and a release access on a barrier wait corresponds to the acquire accesses executed by all the processes on the same barrier wait.

Intervals are partially ordered according to the following two relations: (i) Intervals on a single process are totally ordered by program order and (ii) An interval x precedes an interval y, if the release that ends x corresponds to the acquire that starts y[1]. The partial order between intervals is represented by assigning a vector timestamp to each interval. TreadMarks implements LRC model by ensuring that if interval x precedes interval y (according to this partial order), all shared memory updates performed during x are visible at the beginning of y.

LRC data structures: Each process maintains the following data structures in its local memory:

PageArray:	Array with one entry per shared page
ProcArray:	Array with one list of interval records per process
DirtyList:	Identifiers of pages that were modified during the current interval
VC:	Local vector clock
Pid:	local process identifier

Each shared page entry has fields for twin page, write notices, page manager, copy set and current status of the page. The twin page of a shared page is used for storing old contents of the corresponding page before attempting any modifications and is used later at the release time (or delayed till the next page difference request) to compute the page differences. The write notices field in the page entry describes modifications to the page. The entry for process i in the write notices array contains a list with all the write notices created by i for the page, that are known to the local process. Each of these write notice records describes the updates performed by i to the page in a given interval. The write notice record contains a pointer to the interval record describing that interval and a pointer to the differences containing the words of the page that were updated in the interval. The interval records contain a back pointer to a list with one write notice for each page that was modified during the interval. Whenever an interval record is created, it is tagged with the vector time and the identity of its creator. The page manager is the process which has the ownership of the page and acts like a home machine for that page. Page ownership might be changed dynamically depending on which process has recently modified the page. The copy set indicates the list of processes that are having copy of the shared page and used to recall the page from other processes when exclusive access to the page is required. The status field of a page entry is the operating system protection status for the page, i.e., if the status is no-access then any access to the page triggers a page fault and if the status is read-only a write access to the page triggers a page fault.

The procArray has an entry for each process. The entry for process i contains a list of interval records describing the intervals created by i that the local process knows about. This list is ordered by decreasing order of interval logical times. We refer to the value of VC_i as i's vector time and to the value of $VC_i[i]$ as i's logical time. Similarly, the vector time of an interval created by i is the value of VC_i when the interval is created and the logical time of the interval is the value of $VC_i[i]$.

The storage for the differences, the write notice records and the interval records is not freed until garbage collection is performed, i.e., a process effectively maintains a log of all shared memory accesses since the last garbage collection. This is necessary because any other process that requires a page which was referenced long back, may ask for entire history of differences for that page. In that case all the differences for that page from the oldest interval must be given in the reply for the page difference request.

MATERIALS AND METHODS

Weaknesses of LRC model: There are several drawbacks for the original Lazy Release Consistency implementation, when applied to real parallel applications. If the data of the application is very large and is modified frequently, then the difference representation for one interval can exceed the size of original page itself. Even if smaller differences are there, collectively for a number of processes, which might acquire the lock in sequence before the current process, the total size of differences together may exceed the size of original page itself. The original LRC implementation is useful if one or few scalar variables are present in a page that is modified in critical section, where the resultant differences are smaller. It is not suitable for large data arrays that are modified frequently by large number of processes. The amount of space utilized by twin pages, which may be maintained for longer periods, if differences are not requested, is additional overhead in addition to the write notice records, interval records and page differences.

Computing page differences, when requested by other processes, is time consuming and requests get delayed. If in advance page differences are computed, then it may become wastage of time if those differences are not requested in the future. The improved implementation of LRC, overcomes these memory space and CPU time problems.

Improved LRC implementation: Improved LRC implementation takes annotated input source program. All the variables of the source program are divided into two categories, small data items and larger data items. Synchronization wise related set of small data items are kept together in one page. Different such sets are placed in to separate pages. The compiler includes the size element for each related data item set. This indicates used amount of memory for the variables in that page. Even though most of the page is wasted, for any parallel application usually very few such type of smaller data items exist. So the overall wastage is very less. For larger data items that may span multiple pages, any additional information is not required. Those pages are treated normally with write-invalidation protocol.

When a lock is acquired the process gets the list of current managers of the recently modified pages. The acquired process modifies its data structures to indicate the managers for each such modified page. For the pages which have not been modified since they are requested for the last time, those are still up to date. If an older page is referenced, then a page fault occurs. The page fault handler determines whether the page is small data item set page or normal page. It sends a partial page request or full page request to the manager of that page depending on page type. It can also request for ownership if the faulted instruction is write operation.

This method eliminates the need for computing page differences and maintaining write notice records, page differences and interval records. But for each page it maintains type of the page, page manager, last acquired interval and state information which occupies very little space. This is really advantageous to real parallel applications that require huge amounts of memory. As the related small variables are maintained in one page those can be requested at once with a single partial page request and thus communication can be reduced. Implementation details of this method with the data structures are given below.

ILRC data structures: Each process maintains the following data structures:

Array of NormalSharedPages	:	{manager, last interval, state}
Array of SmallDataItemSet	:	{Page number, size, manager, last interval, state}
IntervalRecord	:	Interval record holding write notices for small data item set pages
VectorClock	:	To order the intervals

A NormalSharedPage entry contains information about the page such as which process is the current manager, interval when the page is last requested and its state indicating read-only, writable and dirty. A SmallDataItem set entry contains the page number of the data items allocated to and their size in addition to the NormalSharedPage information. IntervalRecord contains for a given interval of vector time, the write notices for small data item set pages that are modified. The vector clock is used for partial ordering of the intervals in all processes.

The prototype system is designed to support any type of memory consistency model. It includes a generic memory consistency manager that supports any model with a common interface. The common interface contains set of operations for all memory consistency related events. Some of the operations of the memory consistency model interface are given below:

Interface of memory consistency model:
InitConsistencyData: This function initializes the memory consistency model specific data structures. It is called automatically when the process is being created or if the process explicitly calls set_mctype(…) to set the memory consistency model to a specific model.

PageFault: This performs memory consistency model specific operations such as invalidation, updation, or request a page copy, when a page fault occurs.

LockAcquire: This is called when a lock/semaphore is locked by the current process.

LockRelease: This is called when a lock/semaphore is released by the current process. In the case of a barrier when joining, LockRelease is called and when leaving LockAcquire is called.

InvalidatePage: This is called when invalidation request for a page is received from other processes.

UpdatePage: This is called when update request is received.

EnlargeConsistencyData: This is called for expansion/shrinking of memory consistency specific data structures in the case of process address space size is changed.

ProcessMCRequest: This is called when any other type of memory consistency request is received from another process. This is provided for extending the support for any other type of memory consistency model specific events.

ProcessMCReply: This is also provided for extending the support for handling any consistency model specific reply message.

ProcessExit: This is called when a process is terminating, to upload the locally made changes to the data items to their respective home locations.

Environment and software: The prototype system is currently running with a set of X86 based 32-workstations connected with high-speed Ethernet LAN. All workstations execute the same copy of the operating system. The application can be developed using standard pthread library. Any pthread based parallel application can be executed without any modifications. But for using improved features of LRC and Entry consistency models source program must be annotated with the required information, which is a minor change. An application can be started from any workstation. When it creates threads, correspondingly processes are spawned on other workstations, which can run in parallel. All those processes use same selected memory consistency model or a default consistency model assigned by the system.

RESULTS AND DISCUSSION

The developed memory consistency model has been tested with well known parallel algorithms such as reduction, sieve (finding prime numbers), matrix multiplication, merge sort, quick sort[8] , bucket sort and a branch and bound algorithm for travelling salesperson problem[12] and SPLASH-2 Benchmark programs. All of the programs are tested using 8-workstations. The data set sizes for each one of the algorithms and the comparative execution times using old LRC implementation and improved LRC implementation are shown in Table 1.

Merge sort algorithm changes most of the contents of the data frequently. So, as expected, ILRC is showing considerable performance improvement in this case. In Sieve algorithm, actually marking is reduced as the current prime number increases,

Table 1: Performance ILRC and LRC compared

Application	Data set size of the App	LRC (m.sec)	ILRC (m.sec)	Percentage of Impr.
Reduction	8388608	234.26	183.08	21.8
Sieve	4194304	301.19	238.81	20.7
Matmult	512×512	878.45	652.74	25.7
Merge	5242880	516.86	350.25	32.2
Quiksort	5242880	831.59	813.93	2.1
TSP	20	236.01	225.02	4.6
Bucketsort	4194304	10566.47	8278.61	21.7
LU	512×512	1955.85	1633.67	16.5
Radixsort	2097152	1800.39	1545.95	14.1
Barnes	16384	1631.40	1600.00	1.9
FMM	16384	2484.75	2276.62	8.4
Ocean-cont	258×258	301.19	246.77	18.1
Ocean-non	258×258	343.01	270.65	21.1
Water-nsquared	512	418.31	294.53	29.6
Water-spatial	512	368.11	262.69	28.6
FFT	65536	41.83	35.82	14.4

which effectively reduces the amount of change in the flag array. But even then ILRC is showing slight improvement over the original LRC model. This is due to high amount of change in the flags array, in the initial stages. The reduction algorithm is very simple in which contention for lock (Global sum) occurs only at the end. But twin page creation for very short period usage, which is done sequentially one after the other processor, makes comparatively much difference. Hence some difference in execution times can be observed in this case also. Matrix multiplication problem is actually not affected by any particular model, because all computations are independent computations. Propagation of changes in the data arrays does not happen in this case. So there should not be any difference in execution time. But in ILRC improved communication primitives are used for propagation of matrix rows and hence the difference is shown in the table. In the case of travelling sales person problem, as the algorithm follows lexicographic search order, almost all processors search independently. The only data item shared both for reading and writing, is current known bound value. As it is simple small data item, there is no considerable difference between the two consistency models. In other SPLASH-2 applications and kernels also similar improvements can be observed.

CONCLUSION

The Improved Lazy Release Consistency model has been implemented and tested with standard parallel algorithms. For many cases ILRC model has shown better performance compared to the original LRC model. The only drawback is program needs to be annotated, to indicate small data item sets and large data item sets. It can be improved to identify dynamically small data item sets and large data item sets by prediction. Another possible improvement that can be done is adaptively using either whole page approach or incremental modification approach dynamically depending on the amount of changes.

ACKNOWLEDGEMENT

Researchers thank MHRD for their financial support through the research project entitled “implementation of distributed shared memory system using page based memory management system”.

REFERENCES

1. Tanenbaum, A.S., 1994. Distributed Operating Systems. US Edn., Prentice Hall, ISBN: 10: 0132199084, pp: 648.
2. Bennett, J.K., J.K. Carter and W. Zwaenpoel, 1990. Munin: Distributed shared memory based on type-specific memory coherence. Proceeding of the 2nd ACM Symposium on Principles and Practice of Parallel Programming, Mar. 14-16, ACM Press, Seattle, Washington, United States USA., pp: 168-176. http://portal.acm.org/citation.cfm?id=99182
3. Bershad, B.N., M.J. Zekauskas and W.A. Sawdon, 1993. The midway distributed shared memory system. Proceeding of the IEEE Conference on COMPCON Spring, Feb. 22-26, IEEE Xplore Press, San Francisco, CA., USA., pp: 528-537. DOI: 10.1109/CMPCON.1993.289730
4. Amza, C., A. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu and W. Zwaenepoel, 1996. TreadMarks: Shared memory computing on networks of workstations. IEEE Comput., 29: 18-28. DOI: 10.1109/2.485843
5. Fleisch, B. and G. Popek, 1989. Mirage: A coherent distributed shared memory design. Proceedings of the 14th ACM Symposium on Operating System Principles, Dec. 3-6, ACM Press, New York, USA., pp: 211-223. http://portal.acm.org/citation.cfm?id=74851.74871
6. Protic, J., M, Tomasevic and V. Milutinovic, 1995. A survey of distributed shared memory systems. Proceedings of the 28th Annual Hawaii International Conference on System Sciences, Jan. 04-07, IEEE Computer Society Washington, DC., USA., pp: 74-74. http://portal.acm.org/citation.cfm?id=798090
7. Li and Hudak, 1989. Memory coherence in shared virtual memory systems. ACM Trans. Comput. Syst., 7: 321-359. http://portal.acm.org/citation.cfm?id=75105
8. Quinn, M.J., 1993. Parallel Computing Theory and Practice. 2nd Edn., Tata McGraw-Hill Companies, USA., ISBN: 10: 0070512949, pp: 446.
9. Ramachandran, U. and M.Y.A. Khalidi, 1989. An implementation of distributed shared memory. Proceeding of the 1st Workshop Experiences with Building Distributed and Multiprocessor Systems, (EBDMS’89), USENIX Association, pp: 21-38.
10. Steinke, R.C. and G.J. Nutt, 2004. A unified theory of shared memory consistency. J. ACM., 51: 800-849. http://portal.acm.org/citation.cfm?id=1017464
11. Adve, S.V. and K. Gharachorloo, 1996. Shared memory consistency models: A tutorial. Computer, 29: 66-76. DOI: 10.1109/2.546611
12. Ramesh, T. and C. Sudhakar, 2006. A linear space, deterministic, parallelizable, algorithm for travelling sales person problem. Proceedings of NCIOM, Mar. 3-4, Allied Publishers, pp: 273.
13. Zhou, S., M. Stumm and T. McInerney, 1990. Extending distributed shared memory to heterogeneous environments. Proceedings of the 10th International Conference on Distributed Computing Systems, May 28-June 01, IEEE Xplore Press, Paris, France, pp: 30-37. DOI: 10.1109/ICDCS.1990.89329