International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-2 Number-4 Issue-6 December-2012

Defining Uniform Distributed Shared Memory Consistency Models on Unified Framework

Pankaj Kumar[1], Krishna Kumar[2]
[1]Assistant Professor, SRMCEM Lucknow
[2]Research Scholar, CMJ University, Meghalaya

Abstract

Distributed Shared Memory (DSM) is an architectural approach that allows processors to support a single shared address space that is implemented with physically distributed memory. The consistency model of a DSM system specifies the ordering constraints on concurrent memory accesses by multiple processors. Lots of Consistency Model are defined by a wide variety of source including architecture system, application programmer etc. Firstly the paper reviews and discusses the main Distributed Shared Memory Consistency Models and presents a Unified Framework and then defines each model separately on the basis of unified framework. This paper considers only the 'read' and 'write' memory operation to define the memory models.

Keywords

DSM, MCM, Uniform Memory Consistency model, Unified Framework.

1. Introduction

A Distributed Shared Memory (DSM) system provides application programmers the illusion of shared memory on top of massage passing distributed system, which facilitates the task of parallel programming in distributed system. DSM is technique for making multicomputers easier to program by simulating a shared address space on them. In simple way we can say that DSM represents a successful hybrid of two parallel computer classes i.e. shared memory and distributed memory. It provides the shared memory abstraction in system with physically distributed memories and consequently combine the advantages of both approaches [3, 4, 6, 9, 16]. A memory consistency model, or memory model, for a multiprocessor specifies how memory behaves with respect to read and write operations from multiple processors [3, 5]. With respect to the programmer's point of view, the model enables correct reasoning about the memory operations in a program. From the system designer's point of view, the model specifies acceptable memory behaviors for the system. As such, the memory consistency model influences many aspects of system design, including the design of programming languages, compilers, and the underlying hardware. In order to enhance performance, multiprocessors tend to implement sophisticated memory structures. These memories may replicate data through constructs such as caches and write buffers. Furthermore, the time required to access a data object may vary between processes and between objects. Any of these architectural features allow processes to have inconsistent views of memory, which, in turn, can result in unexpected program outcomes [5, 10, 15].

A memory consistency model is a set of guarantees describing constraints on the outcome of sequences of interleaved and simultaneous operations. Fewer guarantees allow more performance optimizations but yield machines that are very complex to understand and program. It is thus essential to provide multiprocessor programmers with a precise description of the memory model of the underlying machine. Several memory consistency models have been described in the literature. These descriptions arise from a wide variety of sources including architecture, system, and database designers, application programmers, and theoreticians. These descriptions use different types and degrees of formalism and hence are difficult to compare. Others are informal and sometimes ambiguous. There is no single unified formalization that describes the memory models addressed in the literature or provided by several existing machines.

2. Memory Consistency in DSM

The consistency model of a DSM system specifies the ordering constraints on concurrent memory accesses by multiple processors, and hence has fundamental impact on DSM systems' programming convenience and implementation efficiency [18]. DSM allows processes to assume a globally shared virtual memory even though they execute on nodes that do not physically share memory. The DSM software provide the abstraction of a globally shared memory in which each processor can access any data item without the programmer having to worry about where the data is or how to obtain its value In contrast in the native programming model on networks of workstations message passing the programmer must decide when a processor needs to

communicate with whom to communicate and what data to be send. For programs with complex data structures and sophisticated parallelization strategies this can become a daunting task [7, 19].

On a DSM system the programmer can focus on algorithmic development rather than on managing partitioned data sets and communicating values. The programming interfaces to DSM systems may differ in a variety of respects. The memory model refers to how updates to distributed shared memory are rejected to the processes in the system. The most intuitive model of distributed shared memory is that a read should always return the last value written unfortunately the notion of the last value written is not well defined in a distributed system. [3, 18, 19].
The memory consistency model can be categorized into parts one which is based on read and write memory operation called as uniform model and the other which is based on synchronization operation also called hybrid model. The synchronization operations are mapped to corresponding operations provided by concurrency control [2, 6, 12].

3. Proposed Unified Framework

The main component of our framework is the consistency models which considers only read & write memory operation to define consistency condition. We identify the characteristics that are inherent to all memory consistency models and the characteristics that are model-specific.

The framework is the combination of strong & relaxed memory model which proposes a simple and general definition of memory consistency models. The proposed framework is based on the models which come in to the category of uniform model. The unified framework is categorized by four properties, order of access; concurrency; atomicity and scope. The order of access defines the sequence in which accesses are seen by interested parties. The concurrency of access defines if nodes can concurrently access the data and the modes in which they can access it. The scope determines the set of data that is to be kept consistent and atomicity defines whether the propagation of updates is done on per access basis or whether several local updates can be done before a batched update is sent out [2].

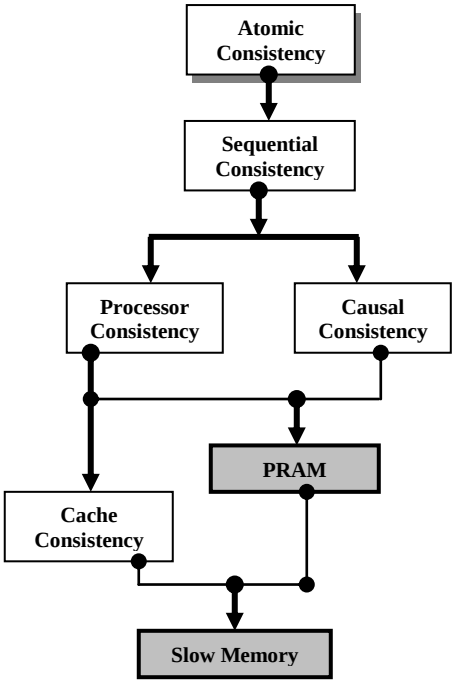

Figure 1: Structure of uniform Frameworks

We have taken Atomic consistency (AC), Sequential consistency (SC), Causal consistency (CC), Processor consistency (PC), PRAM, Cache consistency and Slow memory consistency models for our unified framework which is shown in figure 1 The first two AC & SC is the strong consistency whereas the other one are relaxed consistency. Atomic Consistency is the strict consistency among all the models of framework. If we follow the path from Top to Bottom, The sequential consistency is evolved from the atomic consistency so it inherits some property of atomic consistency. The processor consistency and the causal consistency i.e. defined by the sequential consistency. The PRAM model is evolved by combining the processor consistency as well as causal consistency. The cache consistency is defined by processor consistency and based on cache coherency. Slow memory model is the combination of PRAM and Cache Consistency. But if follow the path from Bottom to Top, the Sequential Consistency is the combination of Processor and Causal consistency and Processor Consistency is the combination of Pram and Cache consistency.

4. Defining Memory Consistency Model

To describe memory models in unified way, we propose a history-based system model that is related to unified framework. In our model, a parallel program is executed by a system. A system is a finite set of processors. Each processor executes a process that issues a set of operations on the distributed

shared global memory **M**. The distributed shared global memory **M** is an abstract entity composed by all addresses that can be accessed by a program. Each processor $\mathbf{P_i}$ has its own local memory $\mathbf{M_i}$. Each local memory $\mathbf{M_i}$ caches all memory addresses of **M**. A memory operation $\mathbf{O_{Pi}(x)v}$ is executed by processor $\mathbf{P_i}$ on memory address **x** with the value **v**. There are two basic types of operations on **M**: **Read (r)** and **Write (w)**. The execution history $\mathbf{H}$ of a process $\mathbf{P_i}$ is an ordered sequence of memory operation issued by the process $\mathbf{P_i}$. Figure 2 shows the execution history of processor $\mathbf{P_1}$ and $\mathbf{P_2}$.

P_1	W(x)2	W(x)1	W(y)1	
P_2			R(y)1	R(x)2

Figure 2: Execution History of P_1 and P_2

A read operation $\mathbf{R_{Pi}(x)v}$ is performed when a write operation on the same location **x** cannot modify the value **v** returned to $\mathbf{P_i}$. Read operations of $\mathbf{P_i}$ are always done on the local memory $\mathbf{M_i}$. A write operation $\mathbf{W_{Pi}(x)v}$ is in fact a set of memory operations $\mathbf{S} = \sum_{i=0}^{n-1} \mathbf{W_{Pi}(x)v}$ where n is the number of processors. A write operation $\mathbf{W_{Pi}(x)v}$ is performed with respect to processor $\mathbf{P_i}$ when the value **v** is written to the address **x** on the local memory $\mathbf{M_i}$ of $\mathbf{P_i}$. A write operation $\mathbf{W_{Pi}(x)v}$ is performed when it is performed with respect to all processors that compose the system.

An order relation that is used in the definition of all memory consistency models proposed is program order ($\xrightarrow{PO}$). An operation $\mathbf{O_1}$ is related to an operation $\mathbf{O_2}$ by program-order $\left(\mathbf{O_1} \xrightarrow{PO} \mathbf{O_2}\right)$ if:

1. Both operations are issued by the same processor $\mathbf{P_i}$ *a*nd $\mathbf{O_1}$ immediately precedes $\mathbf{O_2}$ in the code of $\mathbf{P_i}$ or
2. $\exists\, \mathbf{O_3}$ such that $\left(\mathbf{O_1} \xrightarrow{PO} \mathbf{O_3}\right)$ and $\left(\mathbf{O_3} \xrightarrow{PO} \mathbf{O_2}\right)$

A. Atomic Consistency (AC)

This is the strictest of all consistency models. With atomic consistency, operations take effect at some point in an operation interval. It is easiest to think of operation intervals as dividing time into non-overlapping, consecutive slots [12, 18]. AC, operations can take effect at any point in the operation interval; as long as the resulting history is equivalent to some serial execution. Atomic consistency is often used as a base model when evaluating the performance of an MCM.

Definition: A history $\mathbf{H}$ is ***atomically consistent*** if there is a legal linear sequence of $\mathbf{H}$ that respects the order $\xrightarrow{AT}$ which is defined as follows:

$\forall \mathbf{O_1, O_2}$: **if** $\mathbf{O_1} \xrightarrow{PO} \mathbf{O_2}$ **then** $\mathbf{O_1} \xrightarrow{AT} \mathbf{O_2}$ and

$\forall \mathbf{O_1, O_2}$: **if** $\mathbf{gt(performed(O_1)) < gt(issue(O_2))}$

then $\mathbf{O_1} \xrightarrow{AT} \mathbf{O_2}$

In above definition relation $\xrightarrow{AT}$ shows the order relation where all processors must perceive the same execution order of all shared memory accesses.

B. Sequential Consistency (SC)

Sequential consistency was first defined by Lamport in 1979. He defined a memory system to be sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program [1]. This is equivalent to the one-copy serializability concept found in work on concurrency control for database systems. In a sequentially consistent system, all processors must agree on the order of observed effects. Figure.3 shows a legal execution history for SC:

P_1	**W(x)1**				
P_2			**W(y)2**		
P_3		**R(y)2**		**R(x)0**	**R(x)1**

Figure 3: Execution history of P_1, P_2, and P_3 for SC

Note that R(y)2 by processor P3 reads a value that has not been written yet! Of course, this is not possible in any real physical system. However, it shows a surprising flexibility of the SC model. Another reason why this is not a legal history for atomic consistency is that the write operations W(x)1 and W(y)2 appear commuted at processor P3. Sequential consistency has been the canonical memory consistency model for a long time.

Definition: A history $\mathbf{H}$ is ***sequentially consistent*** if there is a legal linear sequence of $\mathbf{H}$ that respects the order $\xrightarrow{SC}$ which is defined as follows:

$\forall \mathbf{O_1, O_2}$: **if** $\mathbf{O_1} \xrightarrow{PO} \mathbf{O_2}$ **then** $\mathbf{O_1} \xrightarrow{SC} \mathbf{O_2}$

Like Dynamic Atomic Consistency, SC requires a total order on $\mathbf{H}$. The only difference between these two models is that preserving real-time order is no longer necessary in sequential consistency.

C. Causal Consistency (CC)

Causal Consistency means that all processor see all causally related shared access in the same order [6].

Mosberger describes Causal consistency as "A memory is causally consistent if all machines agree on the order of causally related events. Causally unrelated events (concurrent events) can be observed in different orders" [18]. A memory is causally consistent if all processors agree on the order of causally related events. Causally unrelated events (concurrent events) can be observed in different orders [1, 6, 10].

For example: the following is a legal execution history under CC but not under SC, Note that W(x)1 and W(x)2 are causally related as P2 observed the first write by P1.

P1	**W(x)1**		**W(x)3**		
P2	**R(x)1**	**W(x)2**			
P3	R(x)1			R(x)3	R(x)2
P4	R(x)1			R(x)2	R(x)3

Furthermore, P3 and P4 observe the accesses W(x)2 and W(x)3 in different orders, which would not be legal under SC.

Sufficient Conditions for CC

A history **H** is causally consistent if there is a legal linear sequence of **Hpi+w** that respects the order $\xrightarrow{CC}$ which is defined for each processor p_i follows the following condition:

1. atisfy the program order (po)

$$\forall O_1, O_2 : \text{if } O_1 \xrightarrow{PO} O_2 \text{ then } O_1 \xrightarrow{CA} O_2$$

2. atisfy the read by order (rb)

$$\forall O_1, O_2 : \text{if } O_1 \xrightarrow{rb} O_2 \text{ then } O_1 \xrightarrow{CA} O_2$$

3. Satisfy the transtivity relation

$$\forall O_1, O_2 : \text{if } O_1 \xrightarrow{CC} O_2 \text{ and } O_2 \xrightarrow{CC} O_3 \text{ then } O_1 \xrightarrow{CC} O_3$$

D. Processor Consistency (PC)

Processor Consistency is perhaps the clearest example of the problem that can arise if it is not defined the consistency model in formal way. In fact it is a family of memory consistency models that are based on the same idea but have small difference. These differences led to different memory behavior and consequently to different memory consistency models. The basic idea of these memory consistency models is to relax some conditions imposed by sequential consistency and to require only that write operation issued by the same processor are observed by all processor in the order they were issued [12]. Memory sub-operations must execute in a sequential order that satisfies the following conditions:

a) Sub-operations appear in this sequence in the order specified by the program order requirement as shown in the figure 5.5 and

$$\forall O_1, O_2 : \text{if } O_1 \xrightarrow{PO} O_2 \text{ then } O_1 \xrightarrow{PC} O_2$$

b) The order among sub-operations satisfies the write-update coherence (WUC) requirement, and

$$\forall O_1, O_2 : \text{if } O_1 \xrightarrow{PO} O_2 \text{ and } O_1 \xrightarrow{WUC} O_2 \text{ then } O_1 \xrightarrow{PC} O_2$$

c) A read sub-operation issued by R(i) returns the value of either the last write sub-operation W(i) to the same location that appears before the read in this sequence or the last write sub-operation to the location that is before the read in program order, whichever occurs later in the execution sequence.

A processor consistent DSM system with write-update coherence protocol and data replication consists of several processors each with their own copy of the entire memory. By modeling memory as being replicated at every processing node, we can capture the non-atomic effects that arise due to presence of multiple copies of a single memory location. Since the memory no longer behaves as a single logical copy, we need to extend the notion of read and write memory operations to deal with the presence of multiple copies. Write operations no longer appear atomic, however.

E. Pipelined RAM (PRAM)

The acronym PRAM is often used as a shorthand for Parallel Random Access Machine which has nothing in common with the Pipelined RAM consistency model. The reasoning that led to this model was as follows: consider a multi-processor where each processor has a local copy of the shared memory. Mosberger describes PRAM consistency is consistency in which "all processors (machines) observe the writes from a single processor (machine) in the same order while they may disagree on the writes by different processors (machines)" [18].

For the memory to be scalable, an access should be independent of the time it takes to access the other processors' memories. On a read, a PRAM would simply return the value stored in the local copy of the memory. On a write, it would update the local copy first and broadcast the new value to the other processors [6, 12, 18].

Definition: A history **H** is PRAM consistent if there is a legal linear sequence of **Hpi+w** that respects the order $\xrightarrow{PRAM}$ which is defined for each processor p_i follows:

$$\forall O_1, O_2 : \text{if } O_1 \xrightarrow{PO} O_2 \text{ then } O_1 \xrightarrow{PRAM} O_2$$

For the memory to be scalable, an access should be independent of the time it takes to access the other processors' memories. On a read, a PRAM would simply return the value stored in the local copy of the memory. On a write, it would update the local copy first and broadcast the new value to the other processors.

Assuming a constant time for initiating a broadcast operation, the goal of making the cost for a read or write constant is thus achieved. In terms of ordering constraints, this is equivalent to requiring that all processors observe the writes from a single processor in the same order while they may disagree on the order of writes by different processors. For example

P1	W(x)1		W(x)3		
P2		R(x)1	W(x)2		
P3				R(x)1	R(x)2
P4				R(x)2	R(x)1

P3 and P4 observe the writes by P1 and P2 in different orders, although W(x)1 and W(x)2 are potentially causally related.

F. Cache Consistency (Coherence)

Cache consistency and coherence are synonymous. Coherence is a location-relative weakening of SC. Recall that under SC, all processors have to agree on some sequential order of execution for all accesses. Coherence only requires that accesses are sequentially consistent on a per-location basis. Clearly, SC implies coherence but not vice versa. Thus, coherence is strictly weaker than SC [3, 6, 17, 18]. The example below is a history that is coherent but not sequentially consistent:

P1	W(x)1	R(y)0
P2	W(y)1	R(x)0

Clearly, any serial execution that respects program order starts with writing 1 into either x or y. It is therefore impossible that both read accesses return 0. However, the accesses to x can be linearized into R(x)0, W(x)1 and so can the accesses to y: R(y)0, W(y)1. The history is therefore coherent, but not SC. In essence, coherence removes the ordering constraints that program order imposes on accesses to different memory locations.

G. Slow Memory

Slow memory is a location relative weakening of PRAM. It requires that all processors agree on the order of observed writes to each location by a single processor. Furthermore, local writes must be visible immediately (as in the PRAM model). The name for this model was chosen because writes propagate slowly through the system. Slow memory is probably one of the weakest uniform consistency models that can still be used for intercrosses communication. m. However, this algorithm guarantees physical exclusion only. There is no guarantee of logical exclusion [6,18].

Definition: A history Ħ is Slow consistent if there is a legal linear sequence of **Ħpi+w** that respects the order $\xrightarrow{SL}$ which is defined for each processor $\mathbf{p_i}$ follows:

1. All processors must agree about the processor write order on the same memory location

$$\forall O_1, O_2 : \text{if } processor(O_1) = processor(O_2) = p_i \text{ and } O_1 \xrightarrow{PO} O_2 \text{ then } O_1 \xrightarrow{SL} O_2$$

and

2. All processors must eventually see all write operations issued by all processors since the order is defined on

$$\forall O_1, O_2 : \text{if } processor(O_1) = processor(O_2) = p_i \text{ and } address(O_1) = address(O_2)$$
$$\text{and } O_1 \xrightarrow{PO} O_2 \text{ then } O_1 \xrightarrow{SL} O_2$$

For example, after two processes P1 and P2 were subsequently granted access to a critical section and both wrote two variables a and b, then a third processP3may enter the critical region and read the value of as written by P1 and the value of b as written by P2. Thus, for P3 it looks like P1 and P2 had had simultaneous access to the critical section. This problem is inherent to slow memory because the knowledge that an access to one location has performed cannot be used to infer that accesses to other locations have also performed. Slow memory does not appear to be of any practical significance.

5. Conclusion

In this paper we presented a Unified Framework to describe different memory consistency models. The proposed framework considered only the read and write operation and it is not depend upon the synchronization operation so the models taken for defining the framework shows the uniformity property. The Atomic Consistency and Sequential Consistency is the strong consistency. The Atomic Consistency is also the strict consistency. Relations between different consistency models are also defined. A framework can also be designed by using the read and write memory operation as well as synchronization operation.

Reference

[1] L. Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs", IEEE Transaction Computers, vol. C-28, no. 9, pp. 690-691 September 1979.

[2] S. Weber, P.A. Nixon and B Tangney," A flexible Frame work for Consistency Management in Object Oriented Distributed Shared Memory", Department of Computer Science, Trinity College, Ireland, Oct. 13, 1998.

[3] Paul Krzyzanowski "Distributed Shared Memory and Memory Consistency Models" Rutgers University – CS 417: Distributed Systems ©1998, 2001.

[4] Z. Huang, C. Sun and M. Purvis ”Selection-based Weak Sequential Consistency Models for

Distributed Shared Memory" Departments of Computer & Information Science University of Otago, Dunedin, New Zealand , School of Computing & Information Technology Griffith University, Brisbane, Australia.

[5] Lisa Higham, Jalal Kawash and Nathaly Verwaal, "Define and Comparing Memory Consistency Model" ©1997 ISCA, Proceeding of PDCS'97.

[6] Abdelfatah Aref Yahya and Rana Mohamad Idrees Bader "Distributed Shared Memory Consistency Object-based Model", Journal of Computer Science 3 (1): 57-61, 2007 ISSN1549-3636© 2007 Science Publications.

[7] J. Silcock "A Consistency Model for Distributed Shared Memory on RHODOS among Shared Memory Consistency Models" Deakin University, 1997.

[8] John B.Carter, John K. Bennett and Willy Zwaenepoel "Techniques for Reducing Consistency-Related Communication in Distributed Shared Memory Systems"Rice University, TOCS95.

[9] Changhun Lee "Distributed Shared Memory" Proceedings on the 15th CISL Winter Workshop Kushu, Japan ¢ February 2002.

[10] Sarita V. Adve, Member, IEEE, Vijay S. Pai, Student Member, IEEE, and Parthasarathy Ranganathan, Student Member, IEEE "Recent Advances in Memory Consistency Models for Hardware Shared Memory Systems" proceedings Of The Ieee, Vol. 87, No. 3, March 1999.

[11] Albert Meixner and Daniel J. Sorin "Dynamic Verification of Memory Consistency in Cache-Coherent Multithreaded Computer Architectures" Duke University, Department of Electrical and Computer Engineering, Technical Report #2006-1, April 18, 2006.

[12] Alba Cristina Magalhães Alves de Melo "Defining Uniform and Hybrid Memory Consistency Models on a Unified Framework" Proceedings of the 32nd Hawaii International Conference on System Sciences -1999 IEEE.

[13] Jason F. Cantin, Student Member, IEEE, Mikko H. Lipasti, Member, IEEE, and James E. Smith, Member, IEEE "The Complexity of Verifying Memory Coherence and Consistency" IEEE Transactions On Parallel And Distributed Systems, Vol. 16, No. 7, July 2005.

[14] Robert C. Steinke and Gary J. Nutt "A Unified Theory of Shared Memory Consistency" Journal of the ACM, Vol. V, No. N, Month 20YY, Pages 1–47 2002.

[15] Z. Huang, C. Sun and M. Purvis "A View-based Consistency Model based on Transparent Data Selection in Distributed Shared Memory" Technical Report OUCS-2004-03.

[16] Ing. Thomes Seidmann," Distributed Shared memory in Modern Operating System" Ph.D. Thesis, Slovak University of Technology, January, 2004.

[17] Jalal Y. Kawash" Limitations and Capabilities of Weak Memory Consistency Systems" Ph.D. Thesis Calgary, Alberta January, 2000.

[18] D. Mosberger: "Memory consistency models", Operating Systems Review, 17(1):18-26, Jan. 1993.

[19] Benny Wang-Leung Cheung, Cho-Li Wang and Francis Chimoon Lau, "Migrating-Home Protocol for Software Distributed Shared Memory", Journal of Information Science and Engineering, 2000.

[20] Jerzy Brzezinski, Michal Szychowick, "Replication of Checkpoints in Recoverable DSM System", Proceedings of 21 IASTED, Feb 2003.

Dr. Pankaj Kumar is currently working as Assistant Professor in Sri Ramswaroop college of engineering & Management Lucknow. He received his PhD. degree in computer application in 2011 and MCA degree in 2001.His research interests are Parallel Computing, Memory Architecture of Parallel Computer and Distributed Computing. Many of the valuable research papers of Mr. Pankaj Kumar have been published in various national/international journals and IEEE proceeding publication in the area of "Parallel Computing". He is life member of Computer Society of India (CSI) and professional member of International Association of Engineers (IAENG), International Association of Computer Science and Information Technology (IACSIT) and Internet Society (ISOC).

Krishna Kumar (June 4th , 1974) is a research student in the Department of Computer Science & Engineering, CMJ University, Shillong, Meghalaya, India. He has got his Master Degree in Computer Applications (M.C.A.) in 1999 from Madan Mohan Malaviya Engineering Colloge, Gorakhpur which is affiliated to D.D.U. Gorakhpur University, Gorakhpur Uttar Pradesh, INDIA. He has more than 10 years teaching experience and 03 years research experience in the field of Memory Management & Software Engineering. Currently he is actively engaged in the research work on Designing and Defining of Memory Models of DSM System on Unified Framework. He has produced several outstanding publications on various research problems related to the Memory Models. He has published more than 04 International and National publications.