Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 9, September 2014, pg.196 – 201

RESEARCH ARTICLE

CONSISTENCY MODELS IN DISTRIBUTED SHARED MEMORY SYSTEMS

Radhika Gogia[1], Preeti Chhabra[2], Rupa Kumari[3]

Dronacharya College Of Engineering, Gurgaon, India

gogia.radhika13@gmail.com [1], chhabra.preeti28@gmail.com [2], rupasingh252@gmail.com [3]

Abstract - A Distributed Shared Memory (DSM) combines the advantage of shared memory parallel computer and distributed system. The value of distributed shared memory depends upon the performance of Memory Consistency Model. The consistency model is responsible for managing the state of shared data for distributed shared memory systems. Lots of consistency model defined by a wide variety of source including architecture system, application programmer etc. In this paper, we explore shared memory, memory consistency models and mechanisms for differentiating memory operations.

Keywords- Memory consistency, performance, programming language, parallelism

I. INTRODUCTION

In the early days of distributed computing, everyone implicitly assumed that programs on machine with no physically shared memory obviously ran in different address spaces. In 1986, Li proposed a different scheme, known as distributed shared memory.

Shared memory systems cover a broad spectrum, from systems that maintain consistency entirely in hardware to those that do it entirely in software. Distributed shared memory (DSM) is a form of memory architecture where the (physically separate) memories can be addressed as one (logically shared) address space distributed shared memory (DSM) is a form of memory architecture where the (physically separate) memories can be addressed as one (logically shared) address space.

Software DSM systems can be implemented in an operating system (OS), or as a programming library and can be thought of as extensions of the underlying virtual memory architecture. When implemented in the OS, such systems

are transparent to the developer; which means that the underlying distributed memory is completely hidden from the users. Software DSM systems also have the flexibility to organize the shared memory region in different ways. The page based approach organizes shared memory into pages of fixed size.

Shared memory architecture may involve separating memory into shared parts distributed amongst nodes and main memory; or distributing all memory between nodes.

A. ON-CHIP MEMORY

Although most computers have an external memory, self-contained chips containing a CPU and all the memory also exist. Such chips are produced by the millions, and are widely used in cars, appliances, and even toys. In this design, the CPU portion of the chip has address and data lines that directly connect to the memory portion.

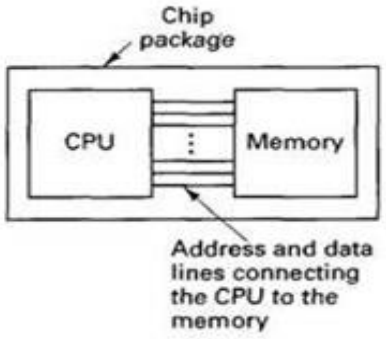

Fig.1 Single-Chip Computer Microprocessor

B. BUS BASED MULTIPROCESSOR

A bus is a collection of parallel wires having connection between CPU and memory, some holding the address the CPU wants to read or write, some for sending or receiving data, and the rest for controlling the transfers. This bus is on-chip, but in most systems, buses are external and are used to connect printed circuit boards containing CPUs, memories, and I/O controllers. On a desktop computer, the bus is typically etched onto the main board (the parent-board), which holds the CPU and some of the memory, and into which I/O cards are plugged. On minicomputers the bus is sometimes a flat cable that wends its way among the processors, memories, and I/O controllers.

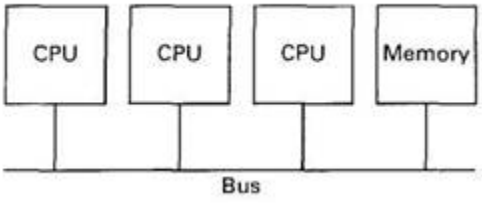

Fig.2 A Bus based Multiprocessor

C. RING BASED MULTIPROCESSOR

In this, a single address space is divided into a private part and a shared part. The private part is divided up into regions so that each machine has a piece for its stacks and other unshared data and code. The shared part is common to all machines (and distributed over them) and is kept consistent by a hardware protocol roughly similar to those used on bus-based multiprocessors. Shared memory is divided into 32-byte blocks, which is the unit in which transfers between machines take place.

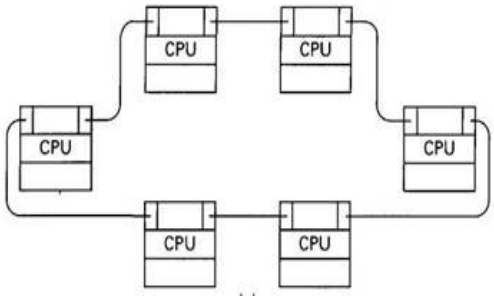

Fig.3 Ring based multiprocessors having six CPU

D. NUMA MULTIPROCESSOR

It is a non-uniform memory access. A NUMA machine has a single virtual address space that is visible to all CPUs. When any CPU writes a value to location a, a subsequent read of a by a different processor will return the value just written.

On a NUMA machine, access to a remote memory is much slower than access to a local memory, and no attempt is made to hide this fact by hardware caching. The ratio of a remote access to a local access is typically 10:1, with a factor of two variations either way not being unusual. Thus a CPU can directly execute a program that resides in a remote memory, but the program may run an order of magnitude slower than it would have had it been in local memory.

❖ *Properties of NUMA Multiprocessors:*

NUMA machines have three key properties that are of concern to us:

1. Access to remote memory is possible.
2. Accessing remote memory is slower than accessing local memory.
3. Remote access times are not hidden by caching.

II. MEMORY CONSISTENCY MODEL

The memory consistency model of a shared-memory multiprocessor provides a formal specification of how the memory system will appear to the programmer, eliminating the gap between the behavior expected by the programmer and the actual behavior supported by a system.

A consistency model is essentially a contract between the software and the memory. It says that if the software agrees to obey certain rules, the memory promises to work correctly. This model determines the order in which memory operations will appear to execute to the programmer. It affects the process of writing parallel programs and forms an integral part of the entire system design including the architecture, the compiler, and the programming language.

Its specification is required for every level at which an interface is defined between the programmer and the system. At the machine code interface, the memory model specification affects the designer of the machine hardware and the programmer who writes or reasons about machine code. At the high level language interface, the specification

affects the programmers who use the high level language and the designers of both the software that converts high-level language code into machine code and the hardware that executes this code.

A. STRICT CONSISTENCY

The most stringent consistency model is called strict consistency. It is defined by the following condition: "Any read to a memory location x returns the value stored by the most recent write operation to x."

To understand this consistency, take an example: In Fig.4,consider two processes P1 and P2. The operations done by each process are shown horizontally, with time increasing to the right. Straight lines separate the processes. The symbols W(x)a and R(y)b mean that a write to x with the value a and a read from y returning b have been done, respectively.

P_1 does a write to location x, storing the value 1. Later, P_2 reads x and sees the 1and this behavior is correct for a strictly consistent memory.

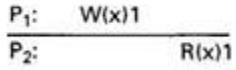

Fig.4 Strictly Consistency Memory

B. SEQUENTIAL CONSISTENCY

It is the most intuitive model greatly restricts the use of many performance optimizations commonly used by uniprocessor hardware and compiler designers, thereby reducing the benefit of using a multiprocessor.

A multiprocessor system is 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.

There are two aspects to sequential consistency:

1) Maintaining program order among operations from individual processors

2) Maintaining a single sequential order among operations from all processors.

The latter aspect makes it appear as if a memory operation executes atomically or instantaneously with respect to other memory operations.

Example for sequential consistency:

```
Initially Flag1 = Flag2 = 0
Flag1 = 1              Flag2 = 1
if (Flag2 == 0)        if (Flag1 == 0)
critical section       critical section
```

C. CAUSAL CONSISTENCY

The causal consistency model represents a weakening of sequential consistency in that it makes a distinction between events that are potentially causally related and those that are not.

For a memory to be considered causally consistent, it is necessary that the memory obey the following condition:

Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.

Example: Consider an event sequence that is allowed with a causally consistent memory in Fig.5, but which is forbidden with a sequentially consistent memory or a strictly consistent memory. The thing to note is that the writes W(x)2 and W(x)3 are concurrent, so it is not required that all processes see them in the same order. If the software fails when different processes see concurrent events in a different order, it has violated the memory contract offered by causal memory.

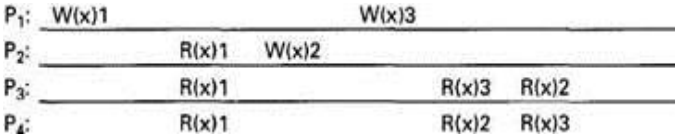

Fig.5 Sequence of events for causally consistent memory

D. PRAM CONSISTENCY AND PROCESSOR CONSISTENCY

It is defined by following condition: Writes done by a single process are received by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes.

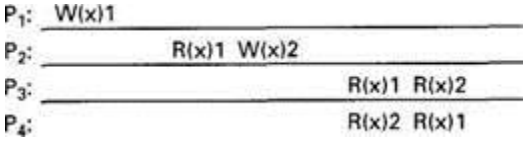

Fig.6 Sequence of events for PRAM consistency

PRAM consistency is interesting because it is easy to implement. It says that there are no guarantees about the order in which different processes see writes, except that two or more writes from a single source must arrive in order, as though they were in a pipeline. In this model all writes generated by different processes are concurrent.

Processor consistency is close enough to pram consistency in such a way that some authors have regarded them as being effectively the same. It is also known as Goodman's model.

E. WEAK CONSISTENCY

This model is defined by stating the following properties:

1. Accesses to synchronization variables are sequentially consistent.
2. No access to a synchronization variable is allowed to be performed until all previous writes have completed everywhere.
3. No data access (read or write) is allowed to be performed until all previous accesses to synchronization variables have been performed.

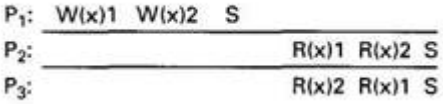

Fig.7 Sequence of events for weak consistency

F. RELEASE CONSISTENCY

This model provide access which are used to tell the memory system that a critical region is about to be entered. Release accesses say that a critical region has just been exited. These accesses can be implemented either as ordinary operations on special variables or as special operations. In either case, the programmer is responsible for putting explicit code in the program telling when to do them.

For example- By calling library procedures such as acquire and release or procedures such as enter_critical_region andleave_critical_region.

There are two flavors of release consistency that differ based on the program orders they maintain among special operations. The first flavor maintains sequential consistency among special operations (RCsc), while the second flavor maintains processor consistency among such operations (RCpc).

III. MECHANISMS FOR DISTINGUISHING MEMORY OPERATIONS

In this, we describe several possible mechanisms for conveying the information which are as follows:

A. CONVEYING INFORMATION AT THE PROGRAMMING LANGUAGE LEVEL

We consider programming languages with explicit parallel constructs. The parallel programming support provided by the language may range from high level parallelism constructs such as do all loops to low level use of memory operations for achieving synchronization. Therefore, the mechanism for conveying information about memory operations depends on the support for parallelism provided by the language.

B. CONVEYING INFORMATION TO THE HARDWARE

The information conveyed at the programming language level must ultimately be provided to the underlying hardware. Therefore, the compiler is often responsible for appropriately translating the higher level information to a form that is supported by the hardware.

REFERENCES

[1] K. Gharachorloo, "Memory consistency models for shared memory multiprocessors," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1995.
[2] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: A tutorial," IEEE Comput., vol. 29, pp. 66–76, Dec 1996.
[3] Sarita V. Adve. Designing Memory Consistency Models for Shared-Memory Multiprocessors. PhD thesis, Computer Sciences Department, University of Wisconsin Madison,December 1993. Available as Technical Report#1198.
[4] http://en.wikipedia.org/wiki/Distributed_shared_ memory

