Remarks on Memory Consistency Description

Ludwik Czaja

Vistula University, Warsaw
and
Institute of Informatics, The University of Warsaw
lczaja@mimuw.edu.pl

Abstract. Two observations in the matter of pictorial as well as formal presentation of some consistency in distributed shared memory are made. The first concerns geometric transformation of line segments and points picturing read/write operations, the second - converting partial order of the operations into linear order of their initiations and terminations. This allows to reduce serialization of the read/write operations as a whole to permutations of their beginnings and ends. Some draft proposals are introduced.

1 Introduction

Description of a consistency model for a distributed shared memory (DSM) requires determining what partial orders between constituents of sequential processes are admissible. The constituents need duration, some of them possibly long - like the update of all replicas of variables. Customarily, a process is pictured as a sequence of line segments or points corresponding to read/write operations, and parallel run as a number of such sequences put on rightwards directed parallel straight lines representing passage of global time. But a certain vagueness and difficulty yields such presentation. What is the right end of a segment corresponding to write operation? Is it a time instant when update of all replicas of a variable being written is completed, or memory manager's response that write request was accepted? Not always the partial order of operations can be converted into a linear order retaining the result of execution, required by some kinds of consistency. For these reasons finer granularity of constituents of processes is proposed: instead of read/write operations as a whole, timeless events of their initiations (invocations) and terminations. In such semantic model, each parallel execution may be represented as a sequence of events, in which independent events may occur in arbitrary order. Furthermore, decision on whether or not a parallel execution can be converted into equivalent sequential, reduces to finding a suitable permutation of the sequence. Another observation is that arrangement of partially ordered executions of read/write operations into a linear order, in geometrical terms means a transformation of line segments representing the operations: shifting them along the time axis.

A schematic image of a system of two computers with DSM is in Fig. 1.1 This virtual memory - a common address space - is here seen as a union of local memories of computers.

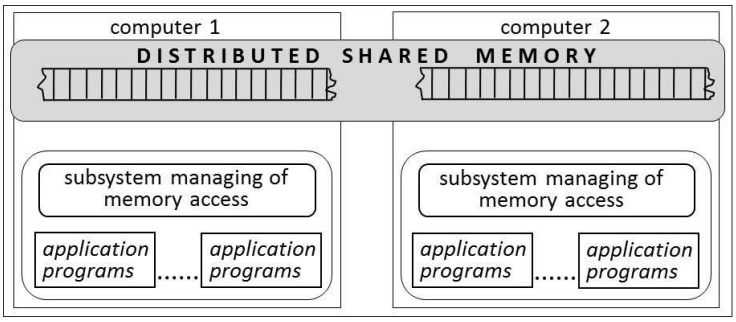


Fig. 1.1

2 Parallel Executions as Succession of Global States

Consider the following system of two computers running programs in parallel with initial values of variables $a = b = 0$. Reading their values is associated with sending them to print.

computer 1	computer 2
$a := 1;$	$b := 1;$
$print(b);$	$print(a);$

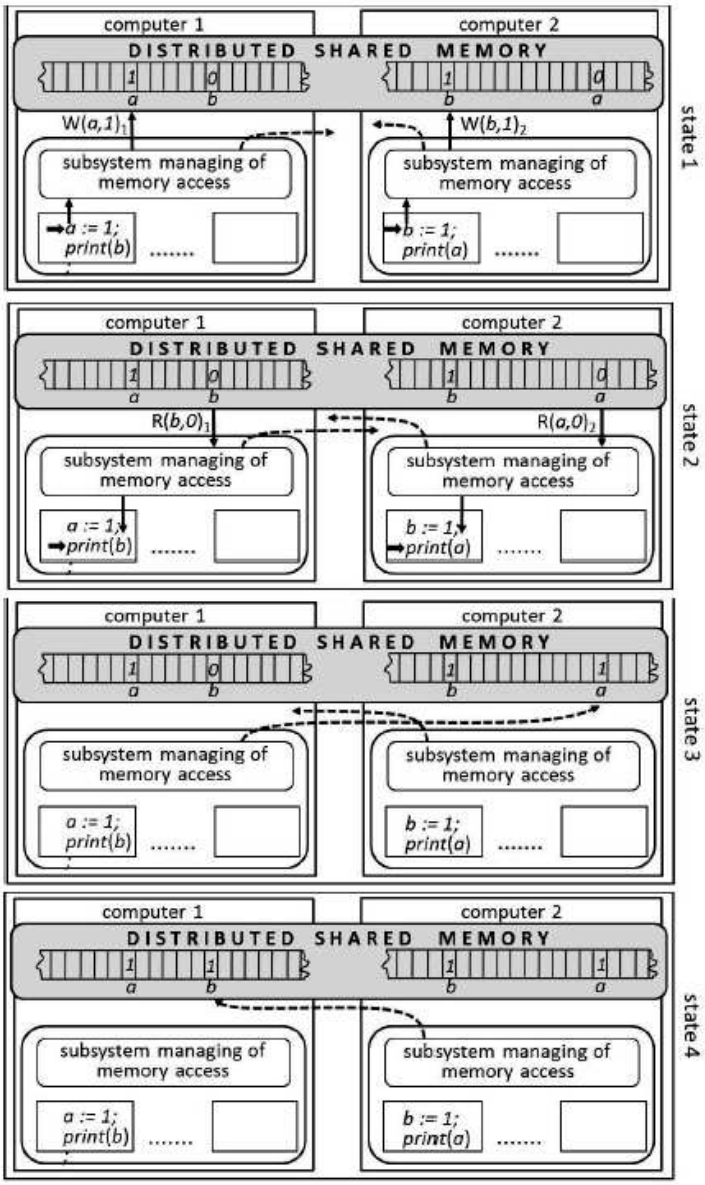
Fig. 2.1

Neither the entire program statements, nor the access operations to the shared variables a, b are assumed to be atomic (indivisible). One of possible executions of the system, presented as a succession of its global states 1 - 4, is shown in Table 1. The states are chosen to display a progress of the system activity. Thus, in a sense, the choice is arbitrary: adapted to a generality level suitable for presentation of a considered problem. Here, writing to shared variables a, b (state 1) and reading their values (along with sending to print - state 2), is performed in parallel by both computers. The dashed arrows stretching across successive states represent the progress of data transmission from one computer to the other.

Two remarks on this execution:

- A variable has a location in the local memory of each computer using this variable: a paging (or cache memory) mechanism ensuring a local accessibility of the variable is assumed (its functioning is not displayed in Table 1). That is why replicas of one variable have locations in local memories of computers using this variable.
- Every computer reads value of a variable before it is updated in all replicas. That is why printouts are $a = 0, b = 0$, but not $a = 1, b = 1$ - what would be the case if progress of data transmission outdistanced reading of values of a and b (transmissions racing)

Table 1. State 1: local update of values of variables a, b and start of their transmission. State 2: local readout of values of variables a, b with sending them to print and their transmission in progress. State 3: end of transmission of the value of variable a and transmission of the value of variable b in progress. State 4: end of transmission of the value of variable b.



3 Geometric Presentation of Parallel Executions

A set of read/write operation occurrences in a single computer process is called a *history of memory access* of this process (*history of the process* in short). Customarily, it is pictured as a set of line segments on a rightwards directed straight line - a global time axis. Each segment represents a time interval between initiation and termination of a read or write execution. Initiation, termed an invocation of the memory access, is shown as the left-most point of the segment. It represents an instant of global time in which request for the operation occurs in the process. Similarly, the right-most point of the segment shows the termination of a read or write and for a read it represents an instant of global time when a value being read is fully fetched from memory to processor. However for a write, the right-most point of the segment may be understood as an instant of global time in which either:

(1) all replicas of a shared variable's value are updated by the value being written, or
(2) a response (acknowledgment of invocation) from the subsystem managing memory access is received by the process.

A placement of the line segments on the global time axis depends on the case taken into account. In the case (1), the history of process is not always a *sequence* but sometimes merely a *set* of segments, i.e. some segments may overlap - they are *partially* ordered in time. This happens because transmission of data through the network not always makes the process suspend: the subsystem managing memory access may proceed in parallel with another activity of the process. In the case (2), the history of process is always a sequence of segments - they are *totally* ordered. Therefore, in the case (1), the execution shown in Table 1 is graphically represented in Fig.3.1, while in the case (2) - in Fig.3.2.

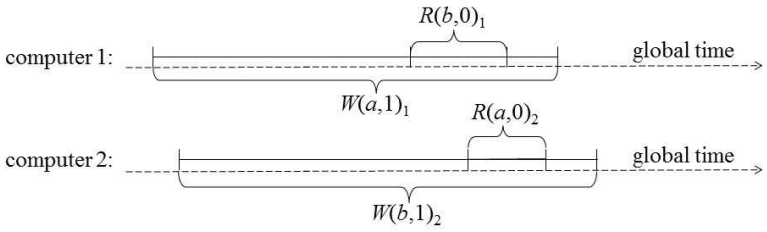


Fig.3.1

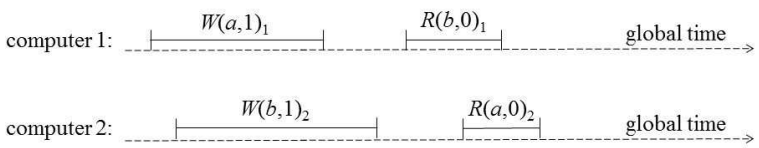


Fig.3.2

4 Sequential and Strict Consistency/Inconsistency - Informal

The system {computer 1, computer 2} in Fig.2.1 may be executed in many ways: a succession of its R/W operation instances in global time may be arbitrary unless restrictions on the activity of subsystems managing of memory access is imposed. Various executions may render various outcomes, i.e. printouts and memory content. Consider the following three examples:

- The DSM managed by subsystems permitting of execution like in Table 1 (geometrically Figures 3.1, 3.2) is a *sequentially inconsistent memory*. The four read/write operations cannot be arranged in one sequence preserving their order of execution in individual programs and result of computation: these two requirements are in contradiction! Indeed, in such arrangement, $R(b,0)_1$ should precede $W(b,1)_2$ and $R(a,0)_2$ should precede $W(a,1)_1$ which violates the order of their execution specified by programs. The segments representing the operations cannot be arranged on the global time axis so that the two requirements are fulfilled.
- Let us modify the above execution replacing $R(b,0)_1$ with $R(b,1)_1$. Now, the operations can be arranged in one sequence fulfilling the two above requirements: $W(b,1)_2\ R(a,0)_2\ W(a,1)_1\ R(b,1)_1$. The corresponding segments have been suitably shifted (a geometric transformation). The DSM managed by subsystems which permit such executions but prohibit such as in the latter example, is a *sequentially consistent memory*.
- Finally, let us modify the latter execution replacing $R(a,0)_2$ with $R(a,1)_2$. The sequence of operations fulfilling the two requirements is now, e.g. $W(a,1)_1\ W(b,1)_2$ $R(b,1)_1\ R(a,1)_2$.Moreover, the sequence preserves succession of operation invocations in global time. The DSM managed by subsystems which permit such executions but prohibit such as in the first example, is a *strictly consistent memory*.

In the first example, no requirement on conformity (order similarity) between a succession of read/write execution specified by individual programs and order of these operations during the system concurrent run, is imposed. In the second, such conformity is required as well as identical outcome of sequential and concurrent executions. In the third example, apart from requirements such as in the latter, the conformity between the global time order of invocations of read/write operations and order of their instances in sequential arrangement should be possible.

Bearing in mind the examples, one may express not too formally both properties of DSM:

Sequential consistency. For any parallel execution there exists a sequential execution i.e. a sequence of read/write operations where their succession is the same as specified by individual programs; furthermore, each readout of a variable fetches a value assigned to this variable by a write operation, which precedes the readout in this sequence, and assignment of different value to this variable does not occur inbetween.

Strict consistency. In any parallel execution, each readout of a variable fetches a value assigned to this variable by a write operation performed before this readout in global time, and assignment of different value to this variable has not been performed inbetween.

Therefore, the meaning of words "precede", "inbetween" etc. is relative to a context of their usage: they may refer to the global time or to a position in a sequence. This must be specified whenever such words are used.

Remarks

(1). Sequential and strict consistency or inconsistency happens also if arrangement of access operations to one variable only is considered. For example, the system in Fig.4.1 may be executed as in Fig.4.2. But the read/write operations cannot be arranged in one sequence satisfying property of sequential consistency. Here too, reading a value of variable a is associated with sending it to print.

computer 1	computer 2	computer 3	computer 4
$a := 1;$	$a := 2;$	$print(a);$	$print(a);$
....	....	.....	....
....	....	$print(a);$	$print(a);$

Fig.4.1

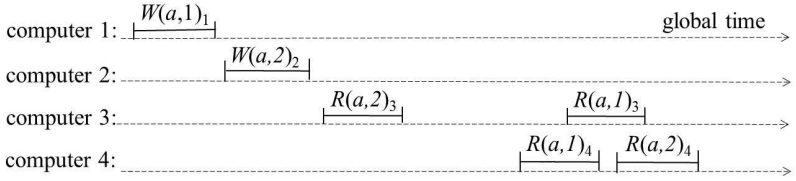


Fig.4.2

However in another execution, which differs from the one in Fig.4.2 so that $R(a,2)_4$ is replaced with $R(a,1)_4$, the read/write operations can be arranged in one sequence: $W(a,2)_2, R(a,2)_3, W(a,1)_1, R(a,1)_4, R(a,1)_3, R(a,1)_4$. Such execution satisfies property of sequential consistency. Execution in which all read operations fetch value 2 would allow to arrange all the read/write operations in one sequence in which their succession is the same as their order of execution in global time. Such sequence satisfies property of strict consistency.

(2) Geometrically, arrangement of read/write operations in a sequence satisfying property of sequential consistency causes a transformation of line segments: a shift along the global time axis. For instance in the modified Fig.4.2, the segment $W(a,1)_1$ would be shifted rightwards, so that this write is performed later than $R(a,2)_3$ and earlier than $R(a,1)_4$: a delay of execution of $W(a,1)_1$ took place.

(3) Arrangement of read/write operations in one sequence boils down to extension of their partial order to linear. The examples demonstrate that this is not possible for some executions when restrictions defining sequential or strict consistency are imposed. Without any restrictions it is always possible, also for infinite history of processes. This

is widely known due to a very general theorem: for any partial order relation $\preceq$ on a set **X** there exists a linear order $<$ on **X** such that $\preceq \subseteq <$ [KM 1976], [Mar 1996]

(4) The unpredictable duration of operations on memory ("coarse granularity") and their presentation as line segments, makes images of parallel execution somewhat counter intuitive, especially when right-most points of the segments represent acknowledgment of invocations (case 2 in Section 3) - not update of all replicas. Worse, not every parallel execution can be described in terms of interleaving of sequences of read/write operations: not always the interleavings exist for some consistency models. The segments are partially ordered, so, one cannot speak of their permutations, what facilitates formal description of some consistency models proposed in Section 6. That is why the (timeless) events of initiation and termination of read/write operations ("fine granularity"), not the operations at the whole length, will be taken as atomic (indivisible) building units of processes in the next Section. With such fine granularity, all events can be ordered into global sequences - interleavings of local histories of memory operations. Parallel executions will be represented by such interleavings, where independent events may occur in the arbitrary order, thus one execution may be represented by several interleavings (note that such approach, in the abstract setting, was a basis for Mazurkiewicz traces [Maz 1987], [TBT 1995] THE BOOK OF TRACES, edited by V. Diekert and G. Rozenberg, World Scientific Publ. Co. 1995).

5 Events of Initiation and Termination of Read/Write

Let us admit the following denotations and assumptions:

(1) Initiation of reading value α of variable x by computer number $j = 1, 2, ..., N$ is denoted by $\overline{R}(x,\alpha)_j$ and termination by $\underline{R}(x,\alpha)_j$; similarly of writing: $\overline{W}(x,\alpha)_j$, $\underline{W}(x,\alpha)_j$. Events $\overline{R}(x,\alpha)_j$ and $\overline{W}(x,\alpha)_j$ are interpreted as invocations for performing the operations. Event $\underline{R}(x,\alpha)_j$ is interpreted as reception of value α of variable x and $\underline{W}(x,\alpha)_j$ as completion of updating of all replicas of x with the value α. A computer initiates and terminates the reading of a variable with the same value, and this concerns also the writing.

(2) Subsystems managing of memory access ensure serialization of initiation and termination events, thus their linear order. In any such sequence, initiation of an operation must precede its termination. Such model clearly exhibits arrangement of access operations on the global time axis. For example, the following sequence of events:

$\overline{W}(a,1)_1\ \overline{W}(a,2)_2\ \overline{R}(a,2)_3\ \underline{R}(a,2)_3\ \underline{W}(a,1)_1\ \overline{R}(a,1)_4\ \overline{R}(a,1)_3\ \underline{R}(a,1)_4\ \underline{W}(a,2)_2$
$\overline{R}(a,2)_4\ \underline{R}(a,1)_3\ \underline{R}(a,2)_4$

shows succession in global time of initiations and complete terminations of read/write operations during parallel execution depicted in Fig.4.2. No sequence of read/write operations as a whole may yield the same result.

Figure 5.1 shows the same execution as Figures 3.1, 3.2, but with events of initiation and termination of read/write. Events $\underline{W}(a,1)_1$, $\underline{W}(b,1)_2$, denote completion of updating all replicas of a and b with the value 1.

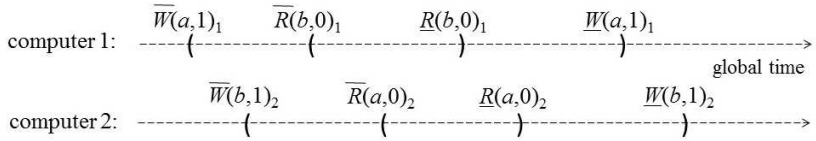


Fig.5.1

The nested bracket structures represent both histories of memory access. Parallel execution progresses e.g. in the following interleaving of the histories:

$$\overline{W}(a,1)_1\ \overline{W}(b,1)_2\ \overline{R}(b,0)_1\ \overline{R}(a,0)_2\ \underline{R}(b,0)_1\ \underline{R}(a,0)_2\ \underline{W}(a,1)_1\ \underline{W}(b,1)_2$$

with the nested bracket structure $((((()))))$. For the memory coherence, the sequence of events should be rearranged so that its fragment from $\overline{R}(a,0)_2$ to $\underline{R}(a,0)_2$, thus the entire operation $R(a,0)_2$, should precede entire operation $W(a,1)_1$ and $R(b,0)_1$ should precede $W(b,1)_2$. On the other hand, for not violating the order of the individual programs run, $W(a,1)_1$ should precede $R(b,0)_1$ and $W(b,1)_2$ should precede $R(a,0)_2$ in this rearranged sequence. This is not possible, thus the execution in Fig.5.1 does not fulfil requirements of sequential consistency.

Modifying the execution in Fig.5.1 by replacing $\overline{R}(b,0)_1$, $\underline{R}(b,0)_1$ with $\overline{R}(b,1)_1$, $\underline{R}(b,1)_1$ accordingly, the sequential consistency is obtained by arrangement:

$$\underbrace{\overline{W}(b,1)_2\ \underline{W}(b,1)_2}_{W(b,1)_2}\ \underbrace{\overline{R}(a,0)_2\ \underline{R}(a,0)_2}_{R(a,0)_2}\ \underbrace{\overline{W}(a,1)_1\ \underline{W}(a,1)_1}_{W(a,1)_1}\ \underbrace{\overline{R}(b,1)_1\ \underline{R}(b,1)_1}_{R(b,1)_1}$$

i.e. permutation of the given sequence. The permuted sequence has the flat bracket structure $()()()()$.

Taking timeless events (points), but not time consuming entire operations, (line segments) as elementary indivisible objects means usage of a linear instead of partial order for description of global system behaviours: the so-called "true concurrency" is simulated by nondeterminism. The example shows that a decision whether or not a given event-sequence can be transformed into a sequence fulfilling the property of sequential or strict consistency, reduces to finding of its suitable permutation. This suggests using this observation in a certain formalization of some consistency models presented in the next Section.

6 Some Consistency Models in the Interleaving Semantics

Let us admit the following denotations:

(1) $\boldsymbol{S} = \{P_1, P_2, \ldots, P_N\}$ – a system of sequential programs with DSM, performing in parallel by computers numbered $1, 2, \ldots, N$.

(2) $\boldsymbol{V}$ – set of variables used by the programs and allocated in DSM.

(3) $\boldsymbol{D}$ – set of values, the variables may assume.

(4) $\boldsymbol{Ev} = \{\overline{W}(x,\alpha)_j, \underline{W}(x,\alpha)_j, \overline{R}(x,\alpha)_j, \underline{R}(x,\alpha)_j\}$ with $x \in \boldsymbol{V}$, $\alpha \in \boldsymbol{D}$, $j \in \{1,2,\ldots,N\}$; this is the set of all events of initiations and terminations of read/write events that may occur during activity of the system $\boldsymbol{S}$.

(5) $Q = q_1 q_2 \ldots q_n \in \boldsymbol{Ev}^*$ is sequentially simulated parallel activity of the system $\boldsymbol{S}$ if every event $\underline{W}(x,\alpha)_j$ and $\underline{R}(x,\alpha)_j$ occurring in Q is preceded by respective $\overline{W}(x,\alpha)_j$ and $\overline{R}(x,\alpha)_j$ in this sequence called a *global history* of $\boldsymbol{S}$. If every $\overline{W}(x,\alpha)_j$ and $\overline{R}(x,\alpha)_j$ is followed on by respective $\underline{W}(x,\alpha)_j$ and $\underline{R}(x,\alpha)_j$ then Q is called *closed* with respect to the memory access operations.

Sequential and strict consistency: A global history $Q = q_1 q_2 \ldots q_n$ fulfills the property of sequential consistency if Q is closed and there exists a permutation $\pi(Q) = q_{\pi(1)} q_{\pi(2)} \ldots q_{\pi(n)}$ such that:

(i) If $\underline{W}(x,\alpha)_j$ or $\underline{R}(x,\alpha)_j$ occurs in $\pi(Q)$ then it is adjacently preceded by respective $\overline{W}(x,\alpha)_j$ or $\overline{R}(x,\alpha)_j$.

(ii) Events $\overline{W}(x,\alpha)_j$ and $\overline{R}(x,\alpha)_j$ occur in the sequence $\pi(Q)$ in the same order as in the local history of program P_j; by virtue of (i) operations $W(x,\alpha)_j$ and $R(x,\alpha)_j$ are performed in the system $\boldsymbol{S}$ in the same global time order as in program P_j.

(iii) If an event $\overline{R}(x,\alpha)_j$ occurs in $\pi(Q)$ then α is an initial value or $\overline{R}(x,\alpha)_j$ is preceded in $\pi(Q)$ by a certain event $\overline{W}(x,\alpha)_k$ with no event $\underline{W}(x,\beta)_i$ inbetween, where $\beta \neq \alpha$, $k,i = 1,2,...,N$.

The system $\boldsymbol{S}$ obeys the principle of sequential consistency of DSM iff $\boldsymbol{S}$ admits only global histories Q fulfilling the property of sequential consistency. In short: the memory managed by $\boldsymbol{S}$ in sequentially consistent.

The strict consistency is obtained if (i), (ii), (iii) are satisfied for the non-rearranging (identity) permutation $\pi(Q) = Q$.

Remarks

1. The global history $Q = q_1 q_2 \ldots q_n \in \boldsymbol{Ev}^*$ uniquely determines the memory state (content) and printouts yielded by Q. The memory state is a set of pairs $\sigma[Q] = \{\langle x,\alpha\rangle : x \in \boldsymbol{V}, \alpha \in \boldsymbol{D}\}$ thus a relation $\sigma[Q] \subseteq \boldsymbol{V} \times \boldsymbol{D}$. If for each closed Q relation $\sigma[Q]$ is a function $\sigma[Q] : \boldsymbol{V} \rightarrow \boldsymbol{D}$ (i.e. $x = y$ implies $\sigma[Q](x) = \sigma[Q](y)$: values of all replicas of each variable are identical), then the memory is *coherent*. Sequentially consistent memory is coherent. Memory incoherence (lack of integrity) arises e.g. in effect of parallel execution shown in Fig.4.2. The execution yields the DSM content: in memory of computers 1,2,4, the value of a is 2 and of computer 3 is 1. The memory state is then $\sigma[Q] = \{\langle a,1\rangle, \langle a,2\rangle\}$, which is not a function.
2. It is known that in general a decision whether or not a given finite execution fulfils the sequential consistency property is intractable. It is evident in the model presented here: a search for a permutation satisfying (i),(ii),(iii) is required. A good many research tackled this task with the same answer, unless additional information is supplied about the system behaviour. Some examples of this research are [GK 1992], [CLS 2005], [WYTC 2011].

3. A challenge is to extend the consistency concept to infinite executions. For such executions some problems, e.g. starvation are explored, but (to my knowledge) not memory consistency models. Note that space of infinite executions is non-enumerable.
4. Strict (unacceptable for efficiency) and sequential consistency bring nearer the DSM system to the multiprocessor with one physical shared memory with direct access. Nonexistence of such memory is transparent for the DSM users. Applications where efficiency is more crucial than preservation of some execution order in individual programs, may tolerate more liberal models than sequential consistency, the natural model for users and more acceptable than strict consistency.

Causal consistency

In any execution, if an effect of an update (write) operation depends on another update (in the same or different process and of the same or different variable), then in every process, the global time order of readouts of the updated variables should be the same as of these updates.

Let us define a relation of *causal dependency* in the set $\boldsymbol{Ev}: \rightsquigarrow \ \subseteq \boldsymbol{Ev} \times \boldsymbol{Ev}$. For events p, q two auxiliary primary relations are admitted:

1. If p precedes q in the same process then $p \xrightarrow{process} q$
2. If event $q = \underline{R}(x,\alpha)_i$ terminates reading value α of variable x and α was assigned to x by a write operation completed with event $p = \underline{W}(x,\alpha)_j$ then $p \xrightarrow{readout} q$

Causal dependency $\rightsquigarrow$ is the least (wrt. $\subseteq$) relation such that:

(i) if $p \xrightarrow{process} q$ or $p \xrightarrow{readout} q$ then $p \rightsquigarrow q$

(ii) if $p \rightsquigarrow q$ and $q \rightsquigarrow r$ then $p \rightsquigarrow r$

If $p \rightsquigarrow q$ then p is a *cause* of q and q is an effect of p. The events are *independent* if neither $p \rightsquigarrow q$ nor $q \rightsquigarrow p$, written $p \parallel q$.

Let $\underline{\boldsymbol{R}} = \{\underline{R}(x,\alpha)_j\}$, $\underline{\boldsymbol{W}} = \{\underline{W}(x,\alpha)_j\}$, $j \in \{1,2,\ldots,N\}$, $x \in \boldsymbol{V}, \alpha \in \boldsymbol{D}$.

Let $Q = q_1 q_2 \ldots q_n \in \boldsymbol{Ev}^*$ be a closed history of execution of the system $\boldsymbol{S}$. Q is *causally consistent* iff for any $q_i, q_j \in \underline{\boldsymbol{W}}$ in Q with $q_i \rightsquigarrow q_j$, and for any $q_k, q_l \in \underline{\boldsymbol{R}}$ in Q, the following holds: **if** $q_k \xrightarrow{process} q_l$ (q_k, q_l are in the same process) and $q_i \xrightarrow{readout} q_k$ and $q_j \xrightarrow{readout} q_l$ **then** $q_k \xrightarrow{process} q_l$; but **if** for some $q_i, q_j \in \underline{\boldsymbol{W}}$, $q_k, q_l \in \underline{\boldsymbol{R}}$ in Q, the relations $q_i \rightsquigarrow q_j$, $q_k \xrightarrow{process} q_l$, $q_i \xrightarrow{readout} q_k$ and $q_j \xrightarrow{readout} q_l$ $q_l \xrightarrow{process} q_k$ hold **then** Q is *causally inconsistent*. In symbols:

$\forall q_i, q_j \in \underline{\boldsymbol{W}}\, [q_i \rightsquigarrow q_j \Rightarrow \forall q_k, q_l \in \underline{\boldsymbol{R}}\, ((q_k \xrightarrow{process} q_l \ \wedge \ q_i \xrightarrow{readout} q_k \wedge$
$q_j \xrightarrow{readout} q_l) \Rightarrow q_k \xrightarrow{process} q_l)]$

The system $\boldsymbol{S}$ obeys the principle of causal consistency of DSM iff $\boldsymbol{S}$ admits only global causally consistent histories Q. In short: the memory managed by $\boldsymbol{S}$ is causally consistent.

Example. Let

$$Q = ...\underbrace{\underline{W}(x,9)_1}_{q_1}...\underbrace{\underline{R}(x,9)_2}_{q_2}...\underbrace{\underline{W}(y,3)_2}_{q_3}...\underbrace{\underline{R}(x,9)_3}_{q_4}...\underbrace{\underline{R}(y,3)_3}_{q_5}...\underbrace{\underline{R}(y,3)_4}_{q_6}...\underbrace{\underline{R}(x,9)_4}_{q_7}$$

This causally inconsistent execution is presented graphically in Fig.6.1. By definition of causality relation $q1 \rightsquigarrow q3$ holds (suppose, for instance, that process P_2 after having read value 9 of variable x, computes $\sqrt[2]{9} = 3$ and writes 3 to y; initiations of read/write operations are not shown for simplicity).

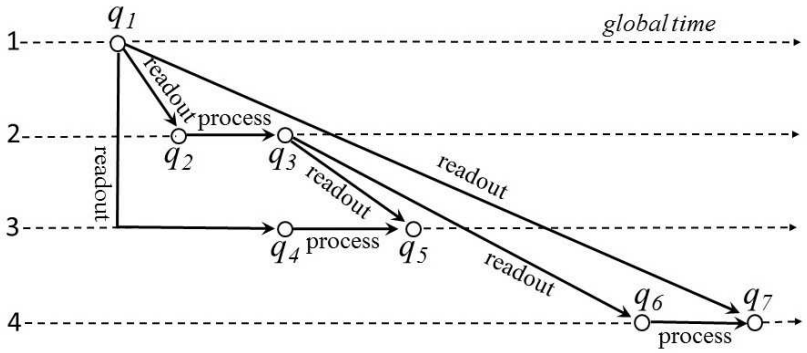


Fig.6.1

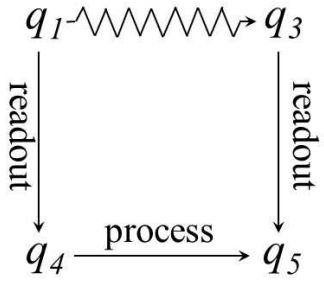


Fig.6.2a

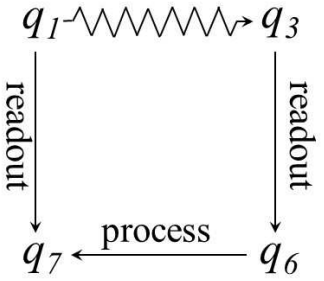


Fig.6.2b

Conditions for causal consistency of a history Q may be schematically presented by diagram in Fig.6.2a. For demonstrating inconsistency in suffices finding events q_1, q_3, q_6, q_7 such that there exists a diagram in Fig.6.2b. The diagrams show relations between events in Q.

PRAM (*Pipelined Random Access Memory*) consistency

In any execution, if two update (write) operations are in the same process, then in every process, the global time order of readouts of these updated variables should be the same as of the updates.

Let $Q = q_1 q_2 \dots q_n \in \boldsymbol{Ev}^*$ be a closed history of execution of the system $\boldsymbol{S}$. Q is *PRAM-consistent* iff for any $q_i, q_j \in \underline{\boldsymbol{W}}$ in Q with $q_i \xrightarrow{process} q_j$, and for any $q_k, q_l \in \underline{\boldsymbol{R}}$ in Q, the following holds: **if** $q_k \xleftarrow{process} q_l$ (q_k, q_l are in the same process) and $q_i \xrightarrow{readout} q_k$ and $q_j \xrightarrow{readout} q_l$ **then** $q_k \xrightarrow{process} q_l$; but **if** for some $q_i, q_j \in \underline{\boldsymbol{W}}$, $q_k, q_l \in \underline{\boldsymbol{R}}$ in Q,

the relations $q_i \xrightarrow{process} q_j$, $q_k \xleftrightarrow{process} q_l$, $q_i \xrightarrow{readout} q_k$ and $q_j \xrightarrow{readout} q_l$ $q_l \xrightarrow{process} q_k$ hold **then** Q is *PRAM-inconsistent*.

In symbols:

$\forall q_i, q_j \in \underline{\boldsymbol{W}} \, [q_i \xrightarrow{process} q_j \Rightarrow \forall q_k, q_l \in \underline{\boldsymbol{R}} \, ((q_k \xleftrightarrow{process} q_l \ \wedge \ q_i \xrightarrow{readout} q_k \wedge$
$q_j \xrightarrow{readout} q_l) \Rightarrow q_k \xrightarrow{process} q_l)]$

The system $\boldsymbol{S}$ obeys the principle of PRAM-consistency of DSM iff $\boldsymbol{S}$ admits only global PRAM-consistent histories Q. In short: the memory managed by $\boldsymbol{S}$ is PRAM-consistent.

Note that causally inconsistent execution shown in Fig.6.1 is PRAM-consistent. The formal definitions of the four memory consistency models clearly set up them into the known hierarchy: $strict \subset sequential \subset causal \subset PRAM$.

References

[CLS 2005] Cantin, J.F.; Lipasti, M.H.; Smith, J.E**.** *The complexity of verifying memory coherence and consistency*, Parallel and Distributed Systems, IEEE Transactions on, On page(s): 663 - 671 Volume: 16, Issue: 7, July 2005

[GK 1992] Gibbons, P.B. ; AT&T Bell Lab., Murray Hill, NJ, USA ; Korach, E. *The complexity of sequential consistency* , Published in: Parallel and Distributed Processing, 1992. Proceedings of the Fourth IEEE Symposium on Date of Conference, Page(s): 317 - 325 : 1-4 Dec 1992

[KM 1976] Kuratowski K., Mostowski A., *Set Theory, with an Introduction to Descriptive Set Theory* (Studies in Logic and the Foundations of Mathematics - Vol 86) Hardcover – February 26, 1976, second edition

[Mar 1996] Marczewski (Spilrajn) K.: *Sur l'extension de l'ordre partiel, FM 16 (1930)*, in Edward Marczewski, Collected Mathematical Papers, IM PAN 1996, pp. 386-389

[Maz 1987] Mazurkiewicz A., *Trace theory*, in W. Brauer et al. Editors, Petri Nets, Application and Relationship to other Models of Concurrency, n. 255, Lecture Notes in Computer Sciences", s. 279-324, 1987.

[TBT 1995] THE BOOK OF TRACES, (edited by V. Diekert and G. Rozenberg), World Scientific Publ. Co. 1995.

[WYTC 2011] Weiwu Hu; Yunji Chen; Tianshi Chen; Cheng Qian; Lei Li, *Linear Time Memory Consistency Verification*, Computers, IEEE Transactions on, On page(s): 502 - 516 Volume: 61, Issue: 4, April 2011