Transactional Memory

Instructor: Sandhya Dwarkadas
University of Rochester

26

Transactional Memory

- Borrowed from Databases
- Definition：***A transaction is a finite sequence of machine instructions executed by a single process, that satisfies the following properties***
  - Atomicity
  - Serializability
  - Essentially, ACI properties

Herlihy and Moss. "Transactional Memory: Architectural Support for Lock-free Data Structures, ISCA'93

27

Transactional memory

- Want multiple processes to access and possibly (try to) write to the same memory location concurrently
- Lock free
- Illusion of atomicity on multiple memory locations
- Guarantee that memory as seen by each processor always in a consistent state

28

Why use transactional memory

- Implement parallel programs without using locks
- Let underlying implementation take care of:
  - Priority inversion
  - Deadlock

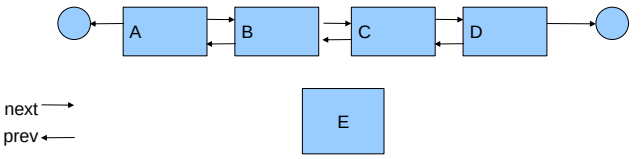
29

Programming Language Extensions?

- Atomic: delimits a transaction (region or block of code)
  - Can be nested/composed
- Requirements
  - Isolation
  - Conflict detection
  - Abort/rollback
  - Commit

30

Example of use: doubly linked list



P1 wants to insert E between B and C:

B.next=E
C.prev = E
E.prev = B
E.next=C

P2 want to delete A

B.prev = null

P3 want to delete C

B.next=D
D.prev=B

31

Definition of transaction (again)

- Finite sequence of machine instructions executed by a single processor that include read and write to memory where
  - Transactions appear to execute serially:
    - Steps of one transaction never appear to be interleaved with the steps of another
    - Committed transactions never appear for different processors to execute in different order
  - Transactions execute atomically: at the end all the changes are written in memory (COMMIT), or they are all discarded (ABORT)

32

In practice

- Programmer marks where transactions start and end
- Transactions are short in time and involve relatively few memory locations

33

Operations in TM

- Validation (not always necessary)
- Conflict detection
- Contention management

34

Conflict detection

- Recognize when committing two transactions would break the assumptions, which is when
  - Two transactions want to write to the same memory locations
  - One transaction wants to write to a memory location that has been read by another
- When a conflict occurs, one of the transaction may need to be aborted

35

Contention management

- How to decide which thread is aborted when a conflict occurs
- Usually not implemented as a separate process. More of a policy that everyone must follows
- Badly designed contention management can lead to bad performance, starvation, livelocks

36

Some contention management algorithms

- Among the many contention management policies (from Scherer et al, 2004)
  - Aggressive
  - Polite
  - Randomized
  - Karma
  - Eruption
  - Killblocked
  - Kindergarten

37

Validation

- Aim to prevent a transaction that would be aborted to execute with inconsistent data
- For instance:
  - Transactions T1 and T2 are in conflict
  - T1 commits, T2 does not abort until it tries to commit
  - T2 may be using some data read before T1 commits, some read after
- May not be necessary as a separate step with aggressive contention management

38

Transactional memory Implementations

- Can be implemented at the hardware or software level
- At the software level can be implemented at various granularity: word level, object level, etc.

39

Example hardware implementation (from Herlihy and Moss ISCA'93)

- Implemented by extension to multi-processor cache coherence
- Can change status of multiple cache lines in one bus cycle
- Separate caches for transactional and non-transactional operations
- Use "aggressive" snooping: a line is not written back to memory unless a cache is full

40

Memory Instructions

- Three types of memory instructions
  - Load Transactional (LT): read into private register
  - Load Transactional exclusive (LTX): read into private register with option to modify
  - Store Transactional (ST): tentatively write to memory

41

Sets

- Read set: set of locations read by LT
- Write set: set of locations accessed by LTX or ST
- Data set: Union of read set and write set

42

Cache line states and tags

Name	Access	Shared?	Modified?
INVALID	none	—	—
VALID	R	Yes	No
DIRTY	R, W	No	Yes
RESERVED	R, W	No	No

Table 1: Cache line states

Name	Meaning
EMPTY	contains no data
NORMAL	contains committed data
XCOMMIT	discard on commit
XABORT	discard on abort

Table 2: Transactional tags

- Cache line states are used by both caches
- Transactional tags used by the transactional cache

43

Operations on transactional cache

- During a transaction the cache keeps two copies for each entry involved:
  - XCOMMIT tag: contains last valid value
  - XABORT tag: can be modified during transaction
- On COMMIT:
  - XABORT → NORMAL
  - XCOMMIT → EMPTY
- On ABORT:
  - XABORT → EMPTY
  - XCOMMIT → NORMAL

44

About XCOMMIT tag

- Not required for correct operation – a performance optimization
  - Avoids the writeback of a modified copy prior to the start of a transaction

45

Bus cycles

Name	Kind	Meaning	New access
READ	regular	read value	shared
RFO	regular	read value	exclusive
WRITE	both	write back	exclusive
T_READ	trans	read value	shared
T_RFO	trans	read value	exclusive
BUSY	trans	refuse access	unchanged

Table 3: Bus cycles

- BUSY sent by process holding a line that is requested by another process
- Will cause process requesting the line to abort.

46

Processor actions

- A transaction aborts itself when it receives a BUSY signal
- If a transaction has not been aborted after its last instruction, then it can be committed
- COMMIT/ABORT operations performed by modifying the tags of the relevant cache entries atomically and in parallel

47