CHAPTER 7: DISTRIBUTED SHARED MEMORY

DSM simulates a logical shared memory address space over a set of physically distributed local memory systems.

Why DSM?

- direct information sharing programming paradigm (transparency)
- multilevel memory access (locality)
- wealth of existing programs (portability)
- large physical memory
- scalable multiprocessor system

Chapter outline

- NUMA architectures: similarity between multiprocessor cache and DSM systems
- Memory consistency models: why is memory consistency a more critical problem in multiprocessor and DSM systems? how is memory consistency defined?
- Cache coherency protocols: implementation of consistency models
- DSM Implementation: applying the consistency models and coherency protocols to a DSM system
Nonuniform Memory Access (NUMA) architectures

Generic NUMA architecture

- Processor
- Memory
- Memory coherence controller
- Buses, interconnection network, or communication network
Multiprocessor Cache and DSM architectures

(a) Multiprocessor cache architecture

(b) Distributed shared memory architecture
Common issues

Data consistency and coherency due to data placement, migration and replication

- Data Sharing Granularity
- Cache Miss Granularity
- Tradeoffs:
  - Transfer time
  - Administrative overhead
  - Hit rate
  - Replacement rate
  - False Sharing

What to do on cache miss?

- Locating block - owner/directory
- Block Migration - block bouncing
- Block Replication
- Push vs. Pull
Memory consistency models

These models apply consistency constraints to all memory accesses. Accesses may require multiple messages and take significant time.

Atomic consistency

All processors see the same (global) order. Respects real-time order.

Sequential consistency

All processors see the same (global) order and order respects all internal orders (not necessarily real time).

\[ P_1 : W(X) \]
\[ P_2 : W(Y) \]
\[ P_3 : R(Y) \]
\[ R(X) \]

"Global Time"

"Global Order"

Atomic Consistency – global total order respecting access intervals

Sequential Consistency – global total order (not necessarily respecting access intervals)
Causal consistency

Processors may see different order
all orders respect causal order (internal and r-w)

\[ P_1 : W(X)1 \quad W(X)3 \]
\[ P_2 : R(X)1 \quad W(X)2 \]
\[ P_3 : R(X)1 \quad R(X)3 \quad R(X)2 \]
\[ P_4 : R(X)1 \quad R(X)2 \quad R(X)3 \]

causal link that must be respected in any order

"P1’s View"

"P2’s View"

"P3’s View"

"P4’s View"

Causal Consistency – no global total order; causal partial order only
Each processor’s order respects internal order and Write–Read causality
Processor consistency

Writes from same processor are in order
Writes from different processors not constrained

\[ P_1 : W(X)_1 \]
\[ P_2 : R(X)_1 W(X)_2 \]
\[ P_3 : R(X)_1 R(X)_2 \]
\[ P_4 : R(X)_2 R(X)_1 \]

Processor Consistency – no global total order; partial order on writes by same processor
Each processor’s order respects internal order and order of writes by same processor
**Slow memory consistency**

Writes from same processor to same location are in order
Writes from different processors or locations not constrained

\[ P_1 : \quad W(X)_1 \quad W(X)_2 \quad W(Y)_3 \]
\[ P_2 : \quad R(Y)_3 \quad R(X)_1 \quad R(X)_2 \]

---

**“P1’s View”**

```
A1: W(X)_1
A2
A3: W(X)_2
A3: W(Y)_3
```

---

**“P2’s View”**

```
A4: R(Y)_3
A5: R(X)_1
A6: R(X)_2
```

---

Slow Memory Consistency – no global total order, no constraints across memory locations
Each processor’s order respects its internal order and order of writes to same memory by same processor
Synchronization Access Consistency Models

Accesses to *synchronization variables* distinguished from accesses to ordinary shared variables

**Weak consistency**

- **Accesses to synchronization variables** are sequentially consistent
- No access to a synchronization variable is issued by a processor before all previous *read/write* data accesses have been performed (i.e., synch waits until all ongoing accesses complete)
- No *read/write* data access is issued by a processor before a previous access to a synchronization variable has been performed (i.e., all new accesses must wait until synch is performed)
- in effect, system “settles” at synch.

**Release consistency**

The synchronization access (*synch*(S)) in the weak consistency model can be refined as a pair of *acquire*(S) and *release*(S) accesses. Shared variables in the critical section are made consistent when the release operation is performed. (i.e., S “locks” access to shared variables it protects, and release is not completed until all accesses to them are also completed).

**Entry consistency**

*acquire* and *release* are applied to general variables. Each variable has an implicit synchronization variable that may be acquired to prevent concurrent access to it.
(a) Weak consistency
like barrier sync
but local to process –
only sync when necessary

(b) Release consistency
Processor consistency
w.r.t. S
All vars in DSM system

c) Entry consistency
Consistency w.r.t.
memory object X
across all procs

---

No synchronization

Weak consistency

Release consistency
Taxonomy

- Atomic consistency
  - Real-time Order Weakening
    - Sequential consistency
    - Processor Relative Weakening
      - Causal consistency
      - Weak consistency
        - Access Type Weakening
          - Processor Relative Weakening
            - Processor consistency
            - Release consistency
              - Location Relative Weakening
                - Slow memory
                - Entry consistency
              - No system coherence support
Multiprocessor Cache Systems

Cache directory

<table>
<thead>
<tr>
<th>master copy</th>
<th>E</th>
</tr>
</thead>
</table>

P bits

<table>
<thead>
<tr>
<th>replicated block</th>
<th>V</th>
<th>E</th>
</tr>
</thead>
</table>

P: Number of processors

<table>
<thead>
<tr>
<th>replicated block</th>
<th>V</th>
<th>E</th>
</tr>
</thead>
</table>

V: Valid or invalid

<table>
<thead>
<tr>
<th>replicated block</th>
<th>V</th>
<th>E</th>
</tr>
</thead>
</table>

E: Exclusive or shared-read-only

V bit for validity (in replicas), E bit for exclusive access (in all)
May also include private (= not shared) bit and/or dirty (= modified) bit.

Cache coherency protocols

write-invalidate and write-update

Write-invalidate
- Read hit
- Read miss: transfer block, set P-, V-, and E-bit.
- Write hit: invalidate cache copies, write and set E-bit
- Write miss: like read miss/write hit

Hardware mechanisms

- Directory-based
- Snooping cache
DSM implementation

Memory management algorithms

READ:
1: Central server algorithm (SRSW)
2: Migration algorithm (SRSW)
3: Read-replication algorithm (MRSW)
4: Full-replication algorithm (MRMW)

WRITE:
1: Remote
2: Migrate
3: Replicate
4: Replicate

- Read-remote-write-remote: long network delay, trivial consistency
- Read-migrate-write-migrate: thrashing and false sharing
- Read-replicate-write-migrate: write-invalidate
- Read-replicate-write-replicate; full concurrency, atomic update

Considerations:
- Block granularity
- Block transfer communication overhead
- Read/write ratio
- Locality of reference
- Number of nodes and type of interaction
Distributed implementation of Directory

Locating Block Owner:

Maintaining Copy List:

(a) Spanning tree representation of copy set

(b) Linked list representation of copy set