L07b. Distributed Shared Memory

Introduction:

- Distributed Shared Memory is an OS abstraction that provides an illusion of shared memory to applications, even though the cluster nodes in the Local Area Network don't physically share the memory.

Cluster as a Parallel Machine:

- Implicitly parallel program:
  - In this scenario, we use an automatic, user-assisted, parallelizing compiler to do the following:
    1. Identify opportunities for parallelization in the sequential program.
    2. Exploit these opportunities using the resources available in the cluster.
  - Compiler directives are used to distribute data/computation to the cluster resources.
  - This is efficient with Data Parallel Programs, in which the data accesses are fairly static and determinable at compile time.
  - Disadvantage: We're limited by the automatic parallelization directives; hence we cannot completely exploit the parallelism opportunities in the cluster.
  - High-Performance FORTRAN is an example of a programming language that facilitates user-assisted automatic parallelization using compiler directives.
- Explicit parallel program (Message passing):
  - In the scenario, the programmer writes an explicitly parallel program using low-level message-passing primitives to exchange messages between cluster nodes.
  - Each cluster node has its own computation and memory resources, and the distribution of resources is achieved by message passing instead of sharing the physical resources themselves.
  - Message passing approach is popular for writing scientific applications running on large-scale clusters.
  - Disadvantage: This approach requires explicit coordination between cluster nodes. This requires a radical change of thinking in terms of how to structure the program so that the activities in different processes can be coordinated.
  - Examples of message passing libraries are MPI, PVM, CLF, etc.
- Explicit parallel program (DSM):
  - The drawbacks of the two approaches explained earlier provided a motivation for the Distributed Shared Memory approach.
  - The transition from a sequential program to a parallel program in a DSM setting is easy and intuitive because it is natural to think of shared data structures among different threads of an application rather than thinking of explicit message passing between cluster nodes.

  - The DSM abstraction provides the same level of comfort to a programmer who is used to use primitives like locks and barriers for synchronization. The underlying DSM run-time will do the heavy-lifting of distribution of data/computation across the different cluster nodes.
- Notes on shared memory programming:
  - Mutex and Barrier are common primitives used in shared memory programming.
  - The memory accesses in a shared memory program are of 2 types:
    1. Normal Read/Write to Shared Data used by the application.
    2. Special Read/Write to Synchronization Variables provided by the operating system.
  - Cache Coherence vs. Memory Consistency:
    1. Cache Coherence is relevant in a multicore system where each processor has a private cache. It's about making sure that every private copy of the data is the same everywhere, and if any changes are made, those changes are propagated to every private cache.
    2. Memory Consistency defines how all the memory instructions in a multicore system will be ordered. It's not enough that a core sees the write happening to shared data, it's also important that it sees the writes in the same order as other cores. Otherwise, data will be *inconsistent* across cores.

Sequential Consistency – SC:

- From the programmer's perspective, all the shared memory accesses that are happening on a specific processor are in sequential (textual) order.
- The problem is, the interleaving between memory accesses on different processors are arbitrary and could lead to problems with regards to program order. This requires a coherence action.
- The SC model doesn't distinguish between data r/w and the OS synchronization information; hence a coherence action is needed for every memory access.
- Why might this be inefficient?
  - If we have multiple processors using the same lock, we're sure that all the data structures inside the critical section protected be the lock will not be accessed by any other processor till the current process releases the lock.
  - However, because the SC model doesn't know this information, it will perform the cache coherence action for each memory access inside the critical section, even though the coherence of these memory sections is not required till releasing the lock.
  - This means that the SC model is doing more work that required, which leads to unnecessary overhead and poor scalability.
  - This is the motivation for the Release Consistency Model.

Release Consistency – RC:

- The Release Consistency Model associates the protected data structures with their lock and defers the cache coherence action until the lock is released.

- This approach avoids blocking the processor for every memory access and instead block it at the lock release time to perform the cache coherence action for all the memory accesses inside the critical section.
- This results in more overlap between computation (memory accesses) and communication (cache coherence) which improves system performance.
- Advantages of the RC model:
  - No waiting for coherence actions on every memory access.
  - Overlap of computation and communication.
  - Better performance over SC.

Lazy RC:

- Since we're sure that no other processors will access the protected data structures before acquiring the lock, we can defer the cache coherence even more till acquiring the lock. This is called the Lazy RC model (considering the previous model as an Eager RC).
- Eager vs Lazy RC:
  - The Eager RC model performs the coherence actions by **broadcast-pushing** the updates at lock-release time to all the processors accessing the same data (single to multiple).
  - The Lazy RC model performs the coherence actions by **unicast-pulling** the updates at lock acquisition time from the single processor that performs the lock release (single to single). This improves system performance.
  - A disadvantage of the Lazy RC model is that at the lock acquisition time, the acquiring processor will incur some latency till the coherence actions are completed.

Software DSM:

- The DSM software partitions the globally shared virtual memory address space into chunks that are managed individually on different cluster nodes.
- The granularity of coherence maintenance in a single node is usually words of memory but communicating words of memory across cluster nodes for each memory access is expensive. Hence, the granularity coherence maintenance in software-DSM is an entire page in order to exploit spatial locality.
- The DSM software also handles maintenance of coherence by having Distributed Ownership for the different virtual pages that constitute the global virtual address space.
- The memory access pattern determines which node owns which pages. The page owner is responsible for keeping coherence information for the page, which means the page owner knows exactly which node to contact to get the latest update and performs the coherence actions for that page.
- The Software DSM systems use the Single-Writer Multiple-Readers Protocol for Cache Coherence maintenance. In this protocol, only a single writer is allowed to write to a page, but multiple readers can read that page at same time.

- The problem with using page level granularity with Single-Writer Multiple-Readers Protocol is that we'll potentially end up with false sharing.
  - Example: A single page X contains 10 different data structures, each protected by a different lock. If node A updates data structure 1 on page X repeatedly, while node B updates data structure 2 on page X repeatedly, then page X will continue to ping-pong between nodes A and B causing False Sharing and thereby decreasing overall system performance.

L-RC with Multi-writer Coherence Protocol:

- In a multi-writer coherence protocol, we'll create a diff of the pages that are being modified between the point before the critical section and the point after it.
- When the same lock is acquired by another process, the pages modified within the critical section of the former lock holder are invalidated.
- The acquiring processor will fetch the original copies of the pages from the page owners, in addition to the diffs from the former lock holder to construct the most updated copies of the pages.
- The advantage of this approach is that fine-grained diffs decrease the communication overhead (only the diffs are required to be communicated).
- The model ensures that the diffs from multiple nodes are obtained by the acquiring processor and applied in order so that the latest updated page can be constructed accurately. This allows the DSM to handle the scenario where only data structure 1 is updated by process 1 and only data structure 2 is updated by process 2 so that only the least number of diffs are acquired, thereby reducing the communication overhead.
- The same page could be modified simultaneously by several different threads so long as they use different locks.
- L-RC with Multi-writer implementation:
  - When a processor tries to write to a page, the page is copied. The original page is made writable while the twin page is not accessible since it's not mapped in the page table.
  - The processor updates the original page.
  - When the lock is released, diffs are computed between the modified original page and the twin page (Run-Length Encoding diffs).
  - The original page will be made write-protect again, and the twin page will be freed (since we already have the diffs by now).
  - If multiple writers are modifying the same page, different locks will be used to protect different parts of the page. Hence, different parts of the page can be modified concurrently by different processors, producing multiple diffs.
  - If multiple processes try to modify the same part of the page simultaneously, a data race would happen. This is a design problem, not an issue with the model itself.
  - If there are too many diffs related to a specific page, there will be space and access latency overhead. So, garbage collection (GC) of these diffs is triggered by applying these diffs in order to the original page and the page owner is updated with this latest copy of the page. All other copies of the page are invalidated.

  - GC is triggered when a particular threshold of pending diffs (space metric) is reached or a particular threshold of access time (time metric) has been exceeded.
  - This is how TreadMarks system implements L-RC with Multiple-writer Coherence protocol.
- Some DSM systems do not use the granularity of a page for coherence maintenance. For such cases, the DSM has to track individual reads and writes happening for each process.
- There are various ways to do that. One approach is called Library-based approach:
  - The program uses a library which provides the framework to do the tracking.
  - The library framework annotates shared variables that are used in the program.
  - Whenever a shared variable is touched, the library framework causes a trap at the point of access to the shared variable, which contacts the DSM software and completes the corresponding coherence actions.
  - The binary generated has appropriate code inserted in it to ensure that these coherence actions happen at the point of access of each shared variable.
  - The advantages of Library-based approach are:
    1. There is no OS support required since generating the trap is taken care of in the binary itself.
    2. Since the sharing is happening at the level of variables and not at the page-level, there is no False-sharing.
  - Examples: Shasta, Beehive.
- Another approach is the Structured DSM approach:
  - A programming library provides shared abstractions at the level of structures that are meaningful for an application and not at the level of memory locations or pages.
  - The shared abstractions are manipulated by the application program using API calls that are part of the language runtime.
  - The API calls complete all coherence actions required for the shared abstractions and do not require any OS support.
  - The advantages of Structured DSM approach are:
    1. No OS support required.
    2. No False Sharing.
  - Examples: Linda, Orca, Stampeded, PTS.

DSM Scalability:

- As more cluster nodes are added to the DSM system, the DSM has the benefit of increased parallelism, but at the cost of increased overhead due to implicit communication overheads on the LAN for coherence maintenance.
- However, overall, we do expect DSMs to scale well as the number of cluster nodes increase.
- The actual speedup may be less than the expected speed due to implicit communication overheads on the LAN for coherence maintenance.
- The amount of speedup will be almost 0 if the sharing is too fine-grained because there will be too much communication overhead.

- So, the basic principle to achieve good scalability is to make sure the computation to communication ratio is very high to achieve good speed-up.
- That is, the critical sections should be large enough so that the communication required for coherency actions is relatively low compared to the computation that happens within the critical section.
- In other words, Distributed Shared Memory scales well only when we share coarse-grained memory and not fine-grained memory.