Parallel and Distributed Systems

Instructor: Sandhya Dwarkadas
Department of Computer Science
University of Rochester

- What is a parallel computer?
  - "A collection of processing elements that communicate and cooperate to solve large problems fast"
- What is a distributed system?
  - "A collection of independent computers that appear to its users as a single coherent system"

Why Parallel or Distributed Computing?

- Fundamentally limited by the speed of a sequential processor
- Resource sharing
- Information exchange
- Collaboration

Programming Models

- Standard models of parallelism
  - shared memory (Pthreads)
  - message passing (MPI)
  - data parallelism (Fortran 90 and HPF)
  - shared memory + data parallelism (OpenMP)

Why is Parallel Computing Hard?

- Amdahl's law – insufficient available parallelism
  - Speedup = 1/(fraction_enhanced/speedup + (1-fraction_enhanced)
- Overhead of communication and coordination
- Portability – knowledge of underlying architecture often required

Steps in the Parallelization Process

- Decomposition into tasks
- Assignment to processes
- Orchestration – communication of data, synchronization among processes

Basics of Parallelization

- Dependence analysis
- Synchronization
  - Events
  - Mutual exclusion
- Parallelism patterns

Parallel Processing Issues

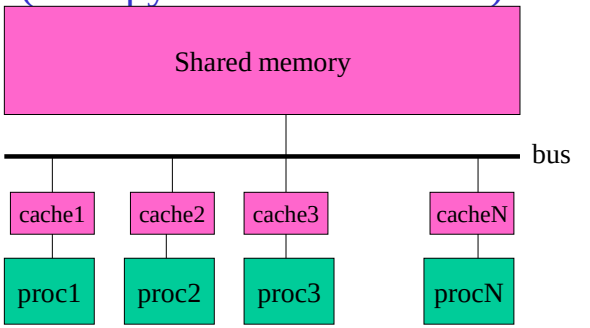
- Data sharing
- Process coordination
- Distributed (NUMA) vs. centralized (UMA) memory
- Connectivity
- Fault tolerance

Correctness and performance issues: deadlock, livelock, starvation

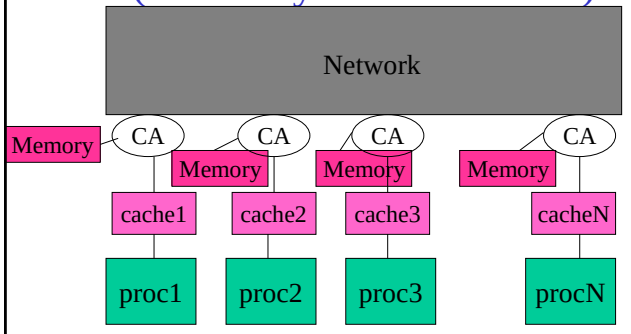
Shared Memory Implementation

- Coherence - defines the behavior of reads and writes to the same memory location
  - ensuring that modifications made by a processor propagate to all copies of the data
  - Program order preserved
  - Writes to the same location by different processors serialized
- Synchronization - coordination mechanism
- Consistency - defines the behavior of reads and writes with respect to access to other memory locations
  - defines when and in what order modifications are propagated to other processors

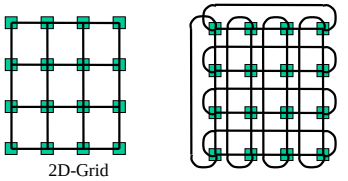
Shared Memory Hardware (Snoopy-Based Coherence)

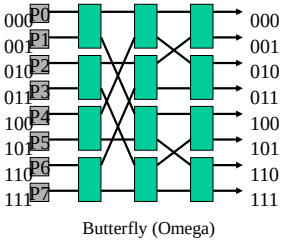


Shared Memory Hardware (Directory-Based NUMA)



Example Interconnect Topologies





- Criteria
  - Bandwidth (per link, bisection, total)
  - Latency (average diameter)
  - Cost (total links, ports/switches)

Basic Hardware Mechanisms for Synchronization

- Test-and-set – atomic exchange
- Fetch-and-increment – returns value and atomically increments it
- Load-locked/store conditional – pair of instructions – deduce atomicity if second instruction returns correct value

Software Synchronization Algorithms

- Locks - test&test&set, ticket, array-based queue, MCS (linked list)
- Barriers – centralized/sense-reversing, software combining trees, tournament, dissemination
- Lock-free, non-blocking, wait-free properties

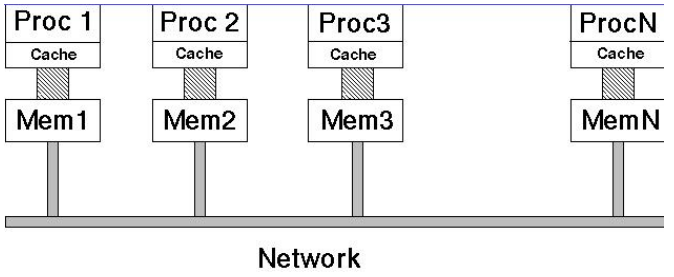
Sequential Consistency

- ``A 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." [Lamport 79]
  - In practice, this means that every write must be seen on all processors before any succeeding read or write can be issued

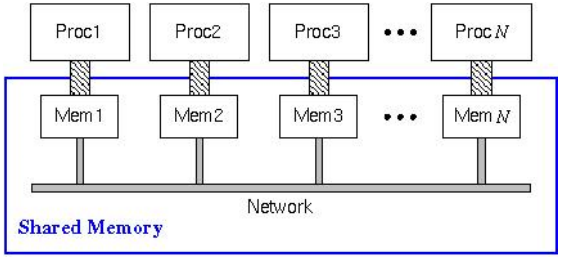
Consistency Model Classification

- Models vary along the following dimensions
  - Program order (for each processor)
  - Write atomicity

Distributed Memory Hardware



Software Distributed Shared Memory (S-DSM)



Cluster-Based Servers

- Large Internet data centers (Google, Microsoft, …)
  - A form of parallel computing
  - Customized parallel programming model (MapReduce)
  - A distributed system
    - Consistency, reliability, scalability, availability
  - Power management, network topology, error monitoring

Dependable Systems

- Availability – most likely working at any instant
- Reliability – run continuously without failure (increase time interval between failures)
- Safety – nothing catastrophic happened on temporary failures
- Maintainability – ease of repair

Faults and Failures

- Fault – cause of an error that might lead to failure
  - Transient
  - Intermittent
  - Permanent
- Failure models
  - Crash (fail-stop)
  - Omission
  - Timing
  - Response (value or state is incorrect)
  - Byzantine or arbitrary

Agreement in Faulty Systems

- A hard problem
  - Perfect processes, unreliable communication
    - E.g., the two-army problem
  - Unreliable processes, perfect communication
    - E.g., the Byzantine generals' problem

Distributed Systems Issues

- Concurrent access – simultaneous access by multiple users to multiple resources
  - Scalability – size, geography, administration
- Reliability – time interval between failures
  - Fault transparency

Distributed Systems Implementation

- Logical versus physical clocks
  - Lamport and vector timestamps
- Distributed snapshots
  - Consistent global state
- Coordinator election
- Mutual exclusion

Logical Clocks

Agreement on ordering of events rather than what time it is is what matters

- Lamport Timestamps: partial order
  - "happened-before" relation "→", causal ordering, or potential causal ordering
    - Transitive relation
  - Assign every event a time value C(a) such that if a → b then C(a) < C(b) – can be captured numerically through a monotonically increasing software counter
  - Lamport's solution for message ordering
    - Each message carries sending time
    - Receiver's clock is set to the greater of its own clock or the sender's clock and then incremented by 1
    - Between every two events, the clock must tick at least once

Vector Timestamps

- Lamport timestamps do not imply causality
- Solution: vector timestamps: captures the notion of causality in addition to concurrency
  - Maintain a vector clock with n integers for n processes

Global State Determination

- Example uses: deadlock or termination detection
- Solution: distributed snapshot (Chandy and Lamport 1985)
  - Important property: consistent global state/cut
  - Termination detection: snapshot in which all channels are empty

Transactions

- Modify multiple data items potentially at multiple locations/by multiple processes as a single atomic operation
- Transaction properties (ACID) –
  - Atomic – happens indivisibly to the outside world
  - Consistent – does not violate system invariants – must hold before and after but not necessarily during
  - Isolated (or serializable) – refers to multiple simultaneous transactions – the final result must appear as if each transaction occurred in some sequential order
  - Durable – once committed, the results become permanent – no failure can undo the results

Distributed Transactions

- ACID properties
- Concurrency control
  - Two-phase locking
  - Pessimistic vs. optimistic timestamp ordering
- Distributed commit
  - Two-phase vs. three-phase commit

Group Communication

- Virtual synchrony – message multicast to group view G is reliably delivered to each non-faulty process in G or to none.
- Unordered vs. FIFO-ordered vs. causally-ordered vs. totally-ordered message delivery
- Scalable reliable multicast

Fault Tolerance

- Checkpointing with rollback recovery
  - Independent checkpointing
  - Coordinated checkpointing
    - Two-phase blocking vs. distributed snapshots (non-blocking)
  - Communication-induced checkpointing
- Log-based recovery
  - Execution of a failed process during recovery is identical to its pre-failure execution

Course Recap

- Basics of parallelization: dependences, synchronization, patterns of parallelism
- Parallel languages/models: PGAS, GPGPU, MPI, OpenMP, Cilk, pthreads, map-reduce
- Coherence, synchronization, and consistency in hardware and software
- Transactional memory
- Fault tolerance and recovery
- Scalability and group communication