Message Passing & Cache-Only Memory Architectures
The Cosmic Cube

- 64 small computers (8086/8087 processor)
- Point-to-point communication network
- Binary 6-cube
- Hardware Simulation of future VLSI implementation with single-chip nodes
- Suggests scalability into 1000s of nodes
- A message passing machine
  - Compare to cc-NUMA: Pros and Cons
N-cube Architecture

- Also called hypercube (paper figure 1)
- Internode communication scales well to large number of nodes (paper figure 2)
- High aggregate bandwidth
- High bisection bandwidth
Process Programming

- Hardware structure of Cosmic Cube is difficult to target for programming
- Resident operating system a more flexible machine-independent environment for concurrent computations
- Process model of computation is quite similar to the hardware structure but is usefully abstracted from it
- Programmer formulates problems in terms of processes and virtual channels between them
- Each process has a unique global ID
- Messages have headers containing src and dest IDs and message info (e.g. type, length)
- A node can have one process or multiple processes
Programming Model

- Message passing: communication and synchronization through messages
  - Explicitly seen by the programmer
- Programming model is reflected in the hardware and operating system
- Operating system kernel in each node
  - schedule processes within node
  - Provides system calls for processes to send and receive
- Single-program, multiple-data (SPMD)
  - N-body problem (Paper figures 3 & 4)
Non-Uniform Memory Access (NUMA) Architectures

- Physical address space is statically partitioned among nodes
  - Access to local memory much faster than remote memory
- For fast execution
  - Program should try to distribute work such that each processor uses mostly data from its local memory
- Optimizing programs for NUMA machines needs:
  - Knowledge of static memory partitioning
  - Migrating part of data used by a processor to its local memory (unless cache is big enough)
- Would be easier if data can migrate automatically to local memory
Cache-Only Memory Architecture (COMA)

- Programming model: Shared memory
- Physical design: Distributed Shared Memory
  - Each node holds a portion of the address space
- Key feature: Partitioning of data is dynamic
  - There is no fixed association between an address and a physical memory location
- Each node has “cache-only” memory
  - Acts like a big cache for the node
  - Holds a subset of the physical address space
- Figure 1 shows UMA, NUMA and COMA
“Attraction” Memory

- Memory of the local node is organized as another cache level, called attraction memory.
- Coherence protocol “attracts” data used by a processor to its attraction memory.
- Virtual address is translated to a cache block or “item” identifier.
- There is no mapping between the identifier and a physical memory.
- But there is an association of the identifier to one or more items.
- Association is determined using the cache tag.
COMA Features

- Does not require static distribution of execution and memory usage
- Migration of data to the processor accessing the data is automatic
- For optimized NUMA programs, COMA runs with comparable speed
- Optimally runs non-optimized NUMA programs
- UMA programs runs well on COMA, but not necessarily on NUMA
- Performs better than NUMA when data migrates according to usage
- No need for programmer to migrate data to local memory
COMA Design Issues

- To avoid increasing memory cost, attraction memory needs to be built with ordinary DRAM
  - Where do we put tags?

- Needs to find a home for blocks evicted from a cache
  - Complicates coherence protocol
  - Could result in deadlocks or livelocks
  - Needs total memory size to be bigger than “item” address space (otherwise, paging will be needed)

- Possible Implementation
  - Extend coherence protocol to find data on a read miss, and to handle replacement
  - Use directory protocol, where state is kept in home directory and data can move freely
Data Diffusion Machine (DDM)

- An implementation of COMA
- Relies on hierarchical network structure
- Large hierarchical machines can be built from small single-bus DDM machines
- Cache coherence protocol modeled after the Write-Once protocol (paper figure 2)
Minimal DDM

- A single bus connects the attraction memories (paper figure 3)
- Use split transaction bus
  - The bus is released between request and response
  - Requests are queued and tagged
- Single-bus DDM coherence protocol
  - Similar to write-once coherence protocol
    - But contains temporary state due to split transaction bus
  - Added support for replacement
    - Handles the attraction (read) and replacement when a set is full
DDM Cache Coherence States

- **Invalid**: Item not present (or has invalid data value)
- **Exclusive**: Only copy of item
- **Shared**: Attraction memory contains item, and other memories could possibly contain it
- **Reading**: Waiting for data after issuing read request
- **Waiting**: Waiting to become exclusive after issuing erase request (i.e., invalidate)
- **Reading and Waiting**: Waiting for data value, will become exclusive later
- **Answering**: Attraction memory has promised to answer a read request
DDM Bus (Network) Transactions

- **Erase**: Invalidate all other copies of item
- **Exclusive**: Acknowledge an erase request
- **Read**: Read a copy of item
- **Data**: Reply to earlier read with data value
- **Inject**: Carry the only copy of data looking for a home
- **Out**: Carry a data out of the subsystem and terminate when another copy is found

State transition diagram: paper figure 4
Replacement

- Attraction memory may run out of space (no more invalid blocks)
- Replacement policy: try Shared items first
- Oldest *Shared* item may be selected for replacement
  - Generates an *Out* transaction
    - If *Out* sees an item in S, R, W, or RW, it does nothing
    - Otherwise, it is converted to Inject transaction
- If an item in *Exclusive* state is to be replaced
  - Protocol generates an Inject transaction
- Inject transaction places item in any available I or S locations in other memories
Hierarchical DDM

- Removes the scalability limitation of a single shared bus
- Replaces top protocol with a directory, which:
  - Interfaces the local bus to a higher level bus
  - Tracks all items in the attraction memories below
- Block diagram (paper figure 5)
- Directory architecture (paper figure 6)
- Multi-level read: Try first in local subsystem, then next higher directory etc. until a directory contains item
- Multi-level write: Erase all copies in local subsystem and the next higher directory etc. until all directories containing item are erased
- Replacement: Out and Inject transactions can move up the directory hierarchy
DDM Prototype

- Block diagram of a node: paper figure 10
- Remote delays in system: paper table 1
- For each item, we need to store
  - Address tag
  - State
- Memory overhead for a 32-processor DDM: 6%
- Memory overhead for a 256-processor DDM: 16%
  (due to multi-level directory hierarchy)
- R-NUMA (Falsafi & Wood) combines both ccNUMA and COMA to choose the best protocol for each memory page
Reading Assignment

- **Tuesday:**
- HW3 due today
- Midterm on Thursday