Stanford Dash Multiprocessing

Slides courtesy of
Sung Joon Yi, SNU CSE

Dash project overview

• Goals
  – high performance
  – wide applicability
    • Easy to programming

• Major components
  – High performance processors
  – Single address space with Directory based Caching
    • Shared address space
    • Cache coherence protocol with HW support
DASH Architecture

- Motivation for Distributed Directory Based Schemes
  - Centralized Directory structure were not scalable
    - Bottleneck
- Distributing the directory and main memory

```plaintext
<table>
<thead>
<tr>
<th>Processor</th>
<th>Cache</th>
<th>Snooping Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Cache</td>
<td>Snooping Bus</td>
</tr>
<tr>
<td>Processor</td>
<td>Cache</td>
<td>Directory</td>
</tr>
<tr>
<td>Processor</td>
<td>Cache</td>
<td>Directory</td>
</tr>
</tbody>
</table>
```

Scalability of the Dash approach(1)

- Scalable memory bandwidth
  - The use of directories removes the need for broadcasts and reduces coherent traffic
  - hot spot : highly contended synchronization objects or heavily shared object data
    - provide queue-based locks, caching shared writable data
- costs scale reasonably
  - Total amount of directory memory with bit-vector
    \[
    \text{(Number of Cluster)}^2 \times (\text{Megabits of memory per cluster}) + (\text{Cache Line size in bits})
    \]
  - limited broadcast of invalidations
    - Invalidations per shared write : 0.71(MP3d) 0.39(Pthor)
  - no need for directory that keeps whole memory block because most main memory blocks will not be present in any processor's cache (replace the complete directory bit-vector with pointers)
Scalability of the Dash approach (2)

- mechanisms are provided to deal with large memory latency
  - reduce latency
  - mechanisms that help tolerate it
- Dash Uses both approach
  - Caching shared data reduces average latency because of the spatial and temporal locality
  - Inter-cluster communication latency can be resolved by various mechanisms
    - support of a relaxed memory consistency
    - support of nonblocking prefetch operations

Cash Coherence protocol of DASH

- Partitioning and distributing the directory and main memory
- Using a new coherence protocol
- Memory hierarchy

Directory Based Cache Coherence(2)

- Key Issues
  - Scaling of memory and directory bandwidth
    - Can not have main memory or directory memory centralized
    - Need a distributed cache coherence protocol
  - Directory memory requirements do not scale well
    - Reason is that the number of presence bits needed grows as the number of PEs
    - In reality, there are many ways to get around this problem
      - limited pointer schemes of many flavors

Memory Hierarchy of Dash(1)

- State of cache block
  - uncached
  - Shared
  - dirty
- directory protocol
  - read

Memory Hierarchy of Dash (2)

- directory protocol
  - write

```
CPU

Processor's Cache
  Cache is dirty
  One of the cache owns the cache line

Local Cluster Cache
  Read-Exclusive is Serviced
  Location state is uncached or shared
  if shared, all cached copy must be in
  validated from local to remote
  No local cache owns the block
  Read-exclusive request
  Home Cluster Cache

Remote Cluster Cache
  Send writeback msg to
  the home level
```

Memory consistency of Dash

- Sequential consistency
  - require execution of parallel program to appear as an interleaving of the parallel processes on a sequential machine
  - imposes a large performance penalty on memory access

- release consistency (Dash use)
  - requires the operations to have completed before a critical section is released
  - provides the user with a reasonable programming model
  - permits reads to bypass writes and the invalidation of different write operations to overlap
  - disadvantage
    - programmer or compiler must identify all synchronization access
Memory access optimization

- Release consistency hides write latency.
  - Significant delay while the processor waits for remote cache
    - For poor cache behavior or extensive r/w sharing application
- Dash provides a variety of prefetch and pipelining op.
  - Prefetch operation
    - explicit nonblocking request to fetch data before issuing actual memory operation
    - allows pipelining of read misses
    - Not binding the value at the prefetching time
  - Update and deliver operations
    - update-write primitives updates the value of all existing copies of a data word
      - when consumer need same data (i.e.: broadcast)
    - deliver instruction explicitly specifies the destination clusters of the transfer

Support for synchronization

- Synchronization memory access may cause hot spot in memory system
  - In case of lock release, waiting nodes rush to grab the lock
  - cached test&set schemes are moderately successful for low contention locks, but fail for high contention lock
- Queue based lock
  - Directory
    - indicate which processors are spinning on the lock
    - choose one of the waiting processor randomly
    - reduce traffic and latency
- fetch-and-{increment/decrement} primitives
  - provide atomic increment/decrement operations on uncached memory location
  - low serialization
Dash Implementation

• Block Diagram of 2x2 Dash

Figure 2: Block diagram of a 2 x 2 DASH system.

Dash Implementation

• Dash prototype cluster
  – SGI Power station 4D/340
    • 4 MIPS R3000
      – 64K Instruction cache, write-through cache, data cache interface
      – 128K second-level writeback cache
  – Coherence Protocol
    • Illinois or MESI (C-to-C)
  – Mpbus
    • 32 bit address and 64 bit data bus
    • pipelined
    • support memory-to-cache and cache to cache transfer
    • Maximum bandwidth 64Mbyte/sec
  – Modification
    • To support directory memory and intercluster interface
      – Retry signal, Arbiter accept a mask, dc board
Dash Implementation

- Block Diagram of directory board

- Coherence example

Figure 4. Flow of Read Request to remote memory with directory in dirty-remote state.

Figure 5. Flow of Read-Exclusive Request to remote memory with directory in shared-remote state.
Dash Implementation

- Dash prototype cluster
  - Interconnection Network
    - use a pair of wormhole routed meshes (request, reply)
    - Total bandwidth 120 Mbytes/sec
    - use a pair of wormhole routed meshes
    - delivery of message without deadlocking but not guarantee.
      - Limited buffering on the directory board
      - message producer may full the buffer
  - Deadlock avoid mechanism
    - reply message can always be consumed because they are allocated a dedicated reply buffer
    - the independent request and reply meshes eliminate request reply deadlocks
    - back off mechanism break potential deadlocks due to request-req uests dependencies

Dash Implementation

- Software support
  - operating system
    - Unix
    - 4D/340 (Irix)
    - Adapt kernel to support special hardware feature
      - Prefetch, update write and queue-based lock
  - Compiler
    - developing parallelizing version
  - programming language
    - Jade
  - performance monitoring and debugging tool
Dash Performance

- Metric
  - the latency for memory accesses
  - virtual speed up
  - actual speed up
  - the ratio of access times between the 1st, 2nd cache, local memory, and remote memory is roughly 1:10:30:100

- the latency for memory accesses
Dash Performance

- speed up
  - tested program
    - Water, Mincut, MP3D
  - simulation program
    - tango allows parallel application to run on a uniprocessor and generates a parallel memory reference stream

- speed up result on simulation

Figure 10: Speedup of the three parallel applications on simulation of the DASH prototype with 1 to 64 processors.

Dash Performance

- speed up result on real experiment (16 cluster)

Figure 11: Speedup of the three parallel applications measured on the actual DASH prototype hardware with 1 to 16 processors.
Dash Performance

- speed up result on real experiment (48 processor)

Related Work

- Encore GigaMax and Stanford Paradigm
- IEEE, Scalable Coherent Interface
- MIT Alewife
  - each node single processor
  - Used SW to handle directory pointer overflow
  - Latency-hiding: multicontext processors
- Other Stuff
  - SGI Origin 2000
    - Origin CC protocol: enhancement Dash’s CC
  - HP/Convex Exemplar
  - Stanford FLASH Architecture
Conclusion

• Single Address Scalability
  – Directory Based Cache-Coherence
  – Distributed Memory
• Design Issue
  – Reducing memory latency