Design of Large-Scale Cache-Coherent Multiprocessors

- Hierarchical Coherence Protocols
- Directory-Based Coherence Schemes
- Invalidation Patterns
- Memory vs Cache based schemes
- Full vs Partial, Dense vs Sparse
- Software Coherence

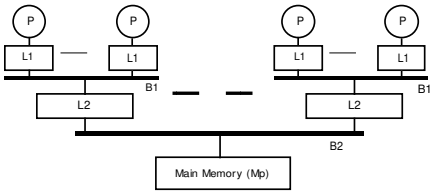
© Culler 1995

Scaling of C.C. Designs with P

Issue:	Bus-based
• Support a large number of processors	no
• Memory Access Latency and Bandwidth	low, fixed
• Coherency protocol bandwidth	1
• Locality / Placement	NA (Uniform)

© Culler 1995

Hierarchy of Busses

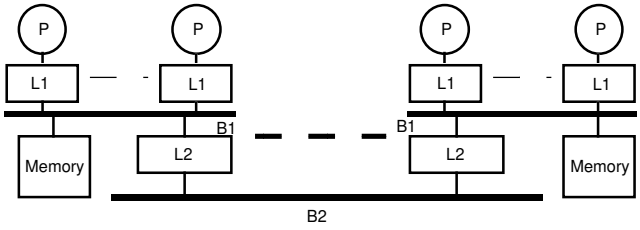


- Snoopy cache protocol on B1 with L2 acting as memory
- Snoopy cache protocol on B2 with L2 acting as processor

- Number processors: limited by packaging
- Memory Latency and Bandwidth: single traversal to root
  - **local, as well as shared data**
- Coherency protocol bandwidth: many B1, one B2
- Locality / Placement: NA for memory
  - **localized sharing avoids B2 broadcasts**

© Culler 1995

Cluster-based Hierarchies (Gigamax, Sparc Center)



- Main memory distributed among clusters

- Number processors: limited by packaging
- Memory Latency and Bandwidth: multiple, fast local access
- Coherency protocol bandwidth: many B1, one B2
- Locality / Placement: important

- L2 cache can be replaced by tag-only router-coherence switch

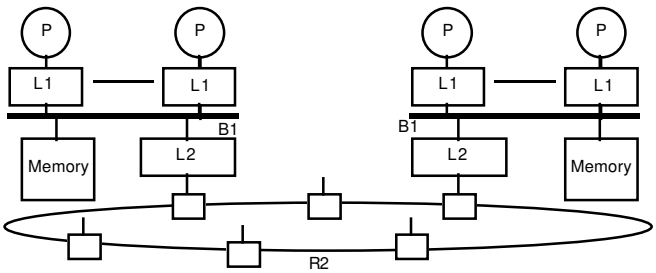
© Culler 1995

Cache Coherence in Encore Gigamax

- Router-Coherence switch must know about:
  - **Local Mp words in remote caches and their state (clean/dirty)**
  - **Remote Mp words in local caches and their state (clean/dirty)**
- A write to a local-bus is passed to global-bus if:
  - **reference belongs to remote Mp**
  - **belongs to local Mp but is present in some remote cache**
- A read to a local-bus is passed to the global-bus if:
  - **reference belongs to remote Mp (and not in cluster cache)**
  - **belongs to local Mp and is dirty in some remote cache**
- A write on global-bus is passed to the local-bus if:
  - **reference belongs to local Mp**
  - **data belongs to remote Mp, but the block is dirty in local cache**
- **...**
- Many race conditions possible
  - **(write-back going out as request coming in)**

© Culler 1995

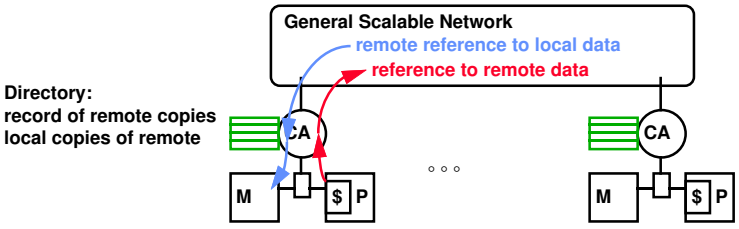
Ring-based Cache Coherence (e.g. KSR)



- Number processors: many
- Memory Latency and Bandwidth linear in P
- Coherency protocol bandwidth
  - **broadcast medium, but no global serialization**
  - **many concurrent transactions, out of phase**
    - » **many, many states**
- Locality / Placement all same on ring

© Culler 1995

Directory Based Cache Coherence

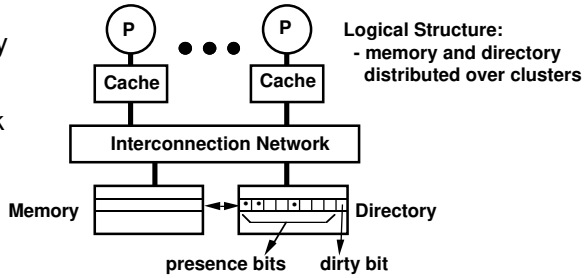


- Snoopy schemes do not scale because they rely on broadcast
- Hierarchical snoopy schemes have the root as a bottleneck
- Directory-based schemes allow scaling.
  - **they avoid broadcasts by keeping track of all PEs caching a memory block, and then using point-to-point messages to maintain coherence**
  - **they allow the flexibility to use any scalable point-to-point network**

© Culler 1995

Full Bit-vector (Censier & Feautrier)

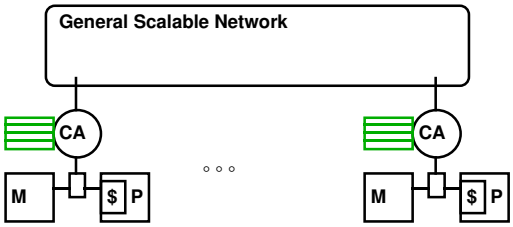
•P presence bits + 1 dirty bit per memory block
• valid & dirty per $ block



- Read from main memory by PE-i:
  - **If dirty-bit is OFF then { read from main memory; turn p[i] ON }**
  - **if dirty-bit is ON then { recall line from dirty PE (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to PE-i; }**
- Write to main memory:
  - **If dirty-bit OFF then { supply data to PE-i; send invalidations to all PEs caching that block; turn dirty-bit ON; turn P[i] ON; ... }**

© Culler 1995

Key Issues



- Number processors: many
- Memory Latency and Bandwidth scalable network
- Coherency protocol bandwidth ?
  - **how many concurrent invalidates (updates)?**
  - **number of copies?**
  - **broadcasts?**
  - **storage requirements: [P • (P • M / B)], where P is # of PEs, M is mem per PE, and B is cache-block size.**
- Locality / Placement moderate

© Culler 1995

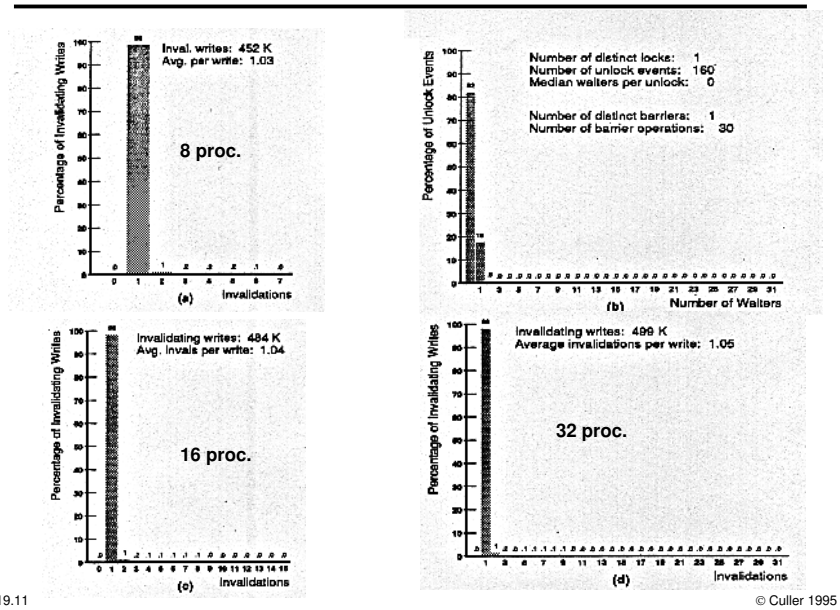
Cache Invalidation Patterns

- Hypothesis:
  - **On a write to a shared location, with high probability only a small number of caches need to be invalidated.**

- If the above were not true, directory schemes would offer little advantage over snoopy schemes.

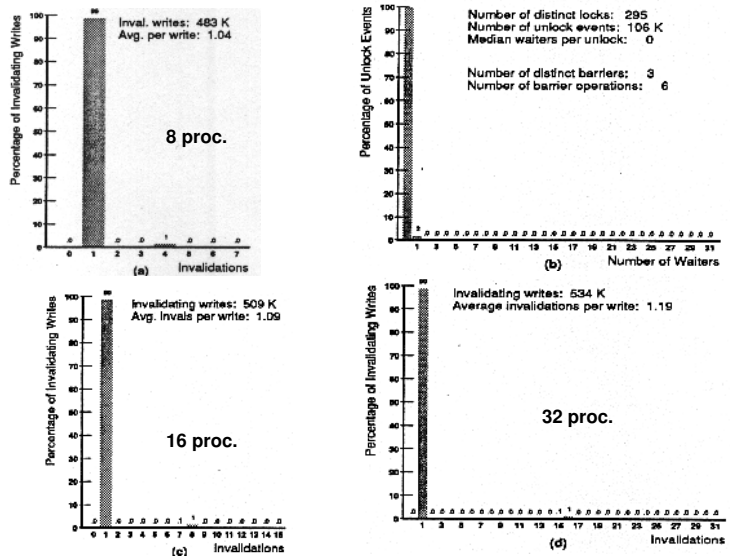
© Culler 1995

Behavior of MP3D



© Culler 1995

Behavior of Wator



© Culler 1995

Invalidation Pattern Summary

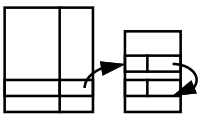
- Code and read-only objects (e.g, distance matrix in TSP)
  - **no problem, rarely written**
- Migratory objects (e.g., particles in MP3D)
  - **even as # of PEs scale, only 1-2 invalidations**
- Mostly-read objects (e.g., bound in TSP)
  - **invalidations are large but infrequent, so little impact on performance**
- Frequently read/written objects (e.g., task queue data structures)
  - **invalidations usually remain small, though frequent**
- Synchronization objects
  - **low-contention locks result in small invalidations**
  - **high-contention locks need special support**
    - » **(SW trees, queueing locks)**

Gupta and Weber, IEEE TOC, July 1992

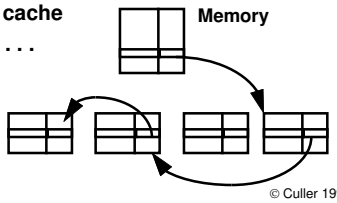
© Culler 1995

Directory Organizations

- Memory based schemes (DASH, Alewife, Flash)
  - **Full map (Dir-n-NB) vs. Partial map (limited pointers)**
    - » **storage scales with total amount of memory x potential sharers**
  - **Dense vs. Sparse**
    - » **fixed vector of potential sharers vs list of actual sharers**
  - **Operates at mem. speed**



- Cache based schemes (Aurora, SCI, . . .)
  - **Use cache blocks to link together sharing chain**
    - » **storage scales with total amount of cache**
    - » **insertion, deletion, communication, . . .**
    - » **single vs. double link**
  - **Built out of SRAM**
  - **More messages**
    - » **clean data read ?**



© Culler 1995

Analysis of Memory Based Schemes

- Full bit vector
  - **Storage**
    - » **One bit of directory memory per main-mem block per PE**
    - » **Memory requirements are [P • (P • M / B)], where P is # of PEs, M is main memory per PE, and B is cache-block size.**
  - **Invalidation traffic**
    - » **less than limited pointer schemes**
  - **One way to reduce overhead is to increase B**
    - » **Can result in false-sharing and increased coherence traffic**
  - **Overhead not too large for medium-scale multiprocessors.**
    - » **Example: 256 PEs organized as sixty four 4-PE clusters**
    - » **64 byte cache blocks ==> ~12% memory overhead**

© Culler 1995

Limited Pointer Schemes

- Few sharers most of the time, so keep few pointers
- Overflow scheme?
- DIR-i-B
  - **Beyond i-pointers, set inval-broadcast bit ON**
  - **Storage needed [i • log(P) • PM / B]**
  - **Expected to do well since widely shared data is not written often**
- DIR-i-NB
  - **When sharers exceed "i", invalidate one of existing sharers**
  - **Significant degradation expected for widely shared mostly-read data**
- DIR-i-CV-r
  - **When sharers exceed "i", use bits allocated to "i" pointers as a coarse-resolution-vector (each bit points to multiple PEs)**
  - **Always results in less coherence traffic than Dir-i-B**
- Limitless directories: Handle overflow using software traps

© Culler 1995

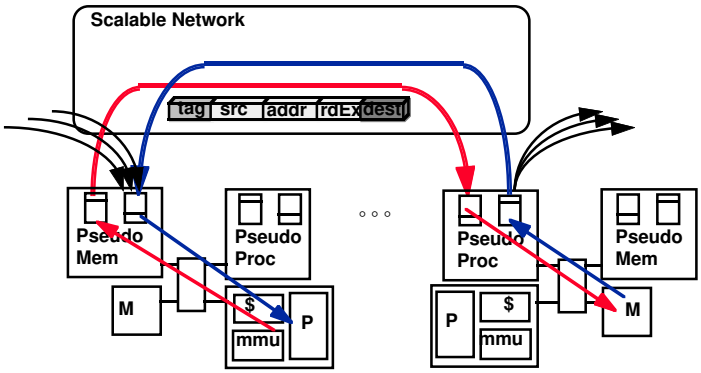
Sparse Directories

- Since total # of cache blocks in machine is much less than total # of memory blocks, most directory entries are idle most of the time
  - **Example: 256 Kbyte cache, 16 Mbyte memory per PE**

    ==> >98% idle
- Sparse directories reduce memory requirements by:
  - **using single directory entry for multiple memory blocks**
  - **dir-entry can be freed by invalidating cached copies of a block**
  - **main problem is the potential for excessive dir-entry conflicts**
  - **conflicts can be reduced by using associative sparse directories**

© Culler 1995

Implementing C.C. Shared Memory



- Remote Pseudo-processor performs directory update as well as memory access
  - **Write: Pseudo mem send RdEx to remote home**
  - **Pseudo proc. sends invalidates to sharers and replies with block and count**
  - **Acks returned to requestor**

© Culler 1995

Flash Directory Structure

- Use a dynamic pointer scheme (Simoni)
  - **dense array with single pointer per memory block, plus next ptr**
- pointers for other sharers are allocated out of free pool
- with replacements, memory usage is proportional to cache in machine

- pointer management in FLASH is handled by a fully programmable but specialized processor

© Culler 1995

Directory-based Coherency Summary

- Directories offer the potential for scalable cache coherence
  - **no broadcasts**
  - **arbitrary network topology**
  - **tolerable hardware overheads**
- Memory:
  - **directory consumes 8-10% of main memory**
  - **amount wasted in msg-passing machines in replication ??**
- Logic Gates:
  - **multiported memory access plus protocol processor**

  - **Paragon: NIC + msging coprocessor**
  - **Meiko CS-2: buffers + DMA engine + SPARC core**
  - **CM-5: NIC + fair portion of SPARC**

© Culler 1995

Compiler based Cache Coherence

- A variety of schemes of differing aggressiveness have been proposed.
- Key observation has been:
  - **"Contents of private caches and shared-memory can differ as long as stale data is not used by the program."**

  ⇒ **One can have lazy or eager schemes.**
- Simplest scheme:
  - **Do not cache read-write shared data, and**
  - **flush caches on process migration.**
- Support needed:
  - **ability to distinguish between private and shared references**
  - **ability to bypass and flush cache when desired**

© Culler 1995

More Advanced Schemes

- Scheme 2:
  - **While within a critical section, shared data being modified can be cached, and then flushed before exiting.**
  - **Shared data in other caches also has to be invalidated, or they have to know when to read-through (bypass the cache).**
- Scheme 3a: (Veidenbaum extensions)
  - **Think of computation as task graphs**
  - **Levelize graph so no communication between tasks at same level**
  - **Flush caches between each level**
- Scheme 3b: (cache lines stay across task boundaries)
  - **Assumes one word cache lines and write-through caches**
  - **Mark references: (i) cache read, (ii) mem read (potentially unsafe)**
- Hardware: change bit; valid bit; invalidate instr. resets all change bits
- Mem reads that find change bit and valid bit set are cache hits

© Culler 1995

Software Based Coherency

- Recent work using OS and runtime systems rather than compilers
- All are based on relaxed consistency models
  - **lazy versus eager schemes (when to communicate)**
- Some are based on page-based consistency
  - **Shared Virtual Memory (Li)**
  - **Lazy Release Consistency (Munin)**
  - **Treadmarks**
- Some also rely on object structure (what to communicate)
  - **Entry Consistency (Midway)**
  - **JADE**

© Culler 1995

Major Research Projects

- DASH
  - **demonstrate scalable directory by integrating SGI clusters at the L2 level**
  - **relaxed consistency models and prefetching**
- Alewife
  - **demonstrate integration of SM and msg passing**
  - **software assist in directory**
  - **limited multithreading**
- WWT / Typhoon
  - **minimal HW support for fine-grain access control**
- Flash
  - **demonstrate integration of SM and msg passing**
  - **flexible protocol processing**
- KSR
  - **COMA**

© Culler 1995