

# Introduction

- Reading
  - for today finish Chapter 10
  - for Thursday Active Messages & T3E paper
    - new link for T3E paper on web site
  - delete Chapter 9 & 11 from reading
- Program #1 was returned

# Cache Coherency (write through)

- Read only data cached
- Writeable values can be cached by one processor
  - a processor needs to gain write access
    - must force invalidation of other cached copies
  - all writes go back to main memory
  - reads can be served from cache for processor with write access
- Performance
  - good for
    - updates and reads by same processor
  - bad for
    - multiple updates by the same processor (many bus writes)

# How to Manage Caches

- **Snooping**

- each cache controller watches bus for “interesting” info
- may result in cache lines being invalidated if write seen
  - i.e. a write through cache
- limited by speed of cache controllers to watch the bus
  - must see everything to maintain correctness

- **Directories**

- memory stores information about cached copies
- does not require each cache controller to snoop
- permits more scaleable interconnect networks

# Directory Based Cache Controllers

- Requires additional circuits to maintain directories
- directories must be updated when a processors
  - starts caching a value
  - stops caching a value
  - changes from read to write caching (or back)
- each cache line has a directory entry
  - can use sparse schemes that only have entries for actively cached items
- can have several memory controllers in a machine
  - each manages a region of physical memory
  - bit vectors (one bit per processor)
  - addresses (several  $\log_2 n$  entries)

# Representing Directories

- bit vectors

- one bit per processor
- uses lots of space for a large machine
- permits each processor to cache a value

- addresses

- several entries for PE id (each entry is  $\log_2 n$  bits)
- what happens if a processor wishes to cache, and all entries are full?
  - use a linked list of directories (SCI uses this approach)
  - use a “wildcard” and force a broadcast to invalidate

# KSR-1

- COMA - Cache Only Memory Architecture
- second level cache replaces main memory
  - called the “all cache” design
  - cache line size is 128 bytes
- Interconnect is a “ring of rings”
  - first level ring
    - 32 processors
    - ARD - Allcache Routing and Directory cell
  - second level ring
    - 1,2,4 GB/sec
    - moves cache lines between rings
  - ARD
    - full directory of all lines on its ring
    - forwards requests from nodes in its ring
    - forwards requests for lines stored on its ring

# KSR-1 (cont.)

- Processor

- custom designed 64 bit microprocessor
  - a big mistake - never could keep pace with technology
  - needed 64 bits since memory is large
- 20 MIPS per processor, superscalar design
- FPU supports chained multiply/add instructions

- OS

- OSF-1
- uses shared memory so footprint is not as large as Paragon
- extensive used of thread based programming model

# Stanford Dash

- Structure

- collection of bus based multi-processors
- interconnect network and cache controller connect nodes

- Cache System

- snoopy protocol within in a single SMP node
- directory based cache controller between nodes
  - misses on local cluster go to home cluster of memory “owner”
  - owner may have current copy or could be cached on another cluster

- Processors

- 4 MIPS R3000 (33 Mhz) per node

- Interconnect

- 2 dimensional mesh



# Stanford Dash (cont.)

- Performance

- level 0 cache (1 clock)
- remote clutter load (132 clocks)

- New Directions

- FLASH
- use a full micro-processor for the cache controller
  - permits customization of cache protocols
  - makes the hardware simpler

# SGI Origin Servers

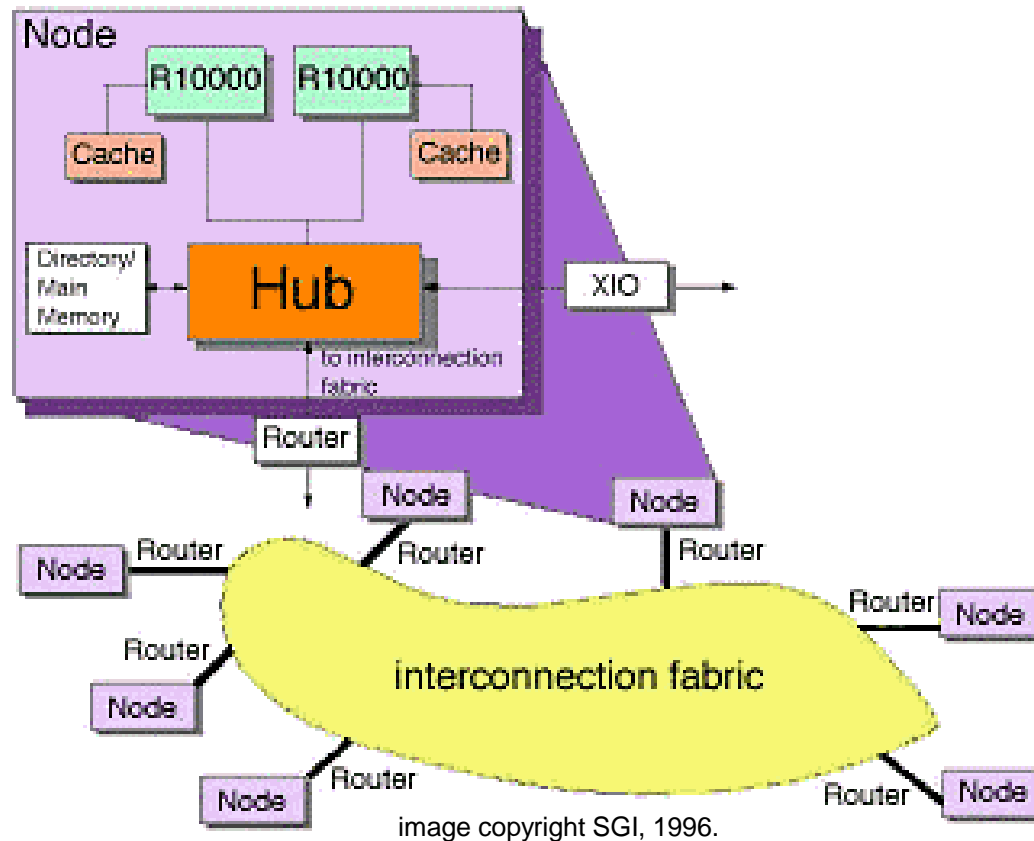
- Commercialization of Stanford DASH

- SMP nodes
- directory based cache controller

- Changes

- processors are R10000
- only 2 nodes per bus
  - slightly cheaper bus than DASH
  - faster processors require more bus bandwidth
- interconnection network
  - hypercube (to 32 nodes)
  - re-configurable routers beyond

# SGI Origin Structure



# Cray T3D

- Globally accesable memory
  - each node has is own private memory
  - can read/write the memory of other nodes
    - no cache coherency is provided
- Processors
  - Alpha: 64 bit processors
  - added extra prefetch hardware
    - just off processor circuit
    - permits requests for memory in advance of need
- Interconnect
  - 3 dimension torus
  - support scatter/gather operations with “address centrifuge”
- OS
  - slave micro-kernel on nodes

# Tera

- Uses threading to hide latency
  - descendent of HEP
  - rapid hardware context switching
    - if a cache miss occurs, switch threads
  - similar do dataflow
    - ready values drive execution
    - however, but programming model is threaded von Neuman
- Interconnect is 3-D mesh
  - processors, memory, and I/O devices are all on the mesh
- Processors
  - fully custom (using GaS technology)
  - superscalar
  - instructions include a count of the number insns that follow that do not depend on the current instruction