Sun’s E6000 Server Family

Erik Hagersten
Uppsala University
Sweden
What Approach to Shared Memory

(a) Shared cache

(b) Bus-based shared memory

(c) Dancehall

(d) Distributed-memory

NUMA

UMA
Looks like a NUMA but drives like a UMA

- Memory bandwidth scales with the processor count
- One “interconnect load” per (2xCPU + 2xMem)
- Optimize for the dancehall case (no memory shortcut)
SUN Enterprise Overview

- 16 slots with CPUs or IO
- Up to 30 UltraSPARC processors (peak 9 GFLOPs)
- Gigaplane™ bus has peak bw 2.67 GB/s; up to 30GB memory
- 16 bus slots, for processing or I/O boards
Enterprize Server E6000

Interconnect

16 boards

MEM

$ P $ P

MEM

$ P $ P

MEM

$ P $ P

I/O

© Erik Hagersten | user.it.uu.se/~eh

Dept of Information Technology | www.it.uu.se

5
An E6000 Proc Board

80 signals = addr, uid, arb, ...

288 signals = 256 data + ECC

Address Controller

Mem

Data

Addr

Snoop tags

Proc tags

ctrl
An I/O Board

80 signals = addr, uid, arb, ...

288 signals = 256 data + ECC

Address Controller

Data

I/O

I/O

Addr

ctrl
Split-Transaction Bus

- Split bus transaction into request and response sub-transactions
  - Separate arbitration for each phase
- Other transactions may intervene
  - Improves bandwidth dramatically
  - Response is matched to request
  - Buffering between bus and cache controllers

Diagram:

```
<table>
<thead>
<tr>
<th>Bus arbitration</th>
<th>Addr Signals</th>
<th>Data Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Address/CMD</td>
<td>Address/CMD</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>Data</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>Data</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>Data</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>Data</td>
</tr>
</tbody>
</table>
```

- Mem Access Delay

Other transactions may intervene.
Gigaplane Bus Timing

Address
- Rd A
- Rd B

State
- Share
- Own

Arbitration
- 1
- 4,5
- 2
- 6
- 7

Tag
- uid1
- uid2
- uid1

Status
- OK
- Cancel

Data
- D0
- D1
- D0
Electrical Characteristics of the Bus

- At most 16 electrical loads per signal
- 8 boards from each side (ex. 15 CPU+1 I/O)
- 20.5 inches "centerplane"
- Well controlled impedance
- ~350-400 signals
- Runs at 90/100 MHz
Dual State Tags

Data

Obligation State

Access-right Stat

ctrl

addr arb aid did

ctrl

OQ

IQ

Prot

Coh

Data
Timing of a single read trans
Board 1 reading from mem 2

- Fixed latency
- Shortest possible latency
Protocol tuned for timing

11 cycles = ~110 ns
Foreign and own transactions queue in IQ
State Change on Address Packet

• Data “A” initially resides in CPU7’s cache
• CPU1: Issues a store request to “A”
• CPU1: Read-To-Write req, ID=d, (i.e., “write request”)
• CPU13: LD “A” -> Read-To-Shared req, ID=e
• CPU15: ST “A” -> RTW req, ID=f

mRTO stored in IQ\textsubscript{CPU1}
Own read IQtrans retired when data arrives
Later requests for A queued in IQ\textsubscript{CPU1} behind mRTO
IQ\textsubscript{CPU1} will eventually store: \(<mRTW_{1d}, fRTS_{1e}, fRTW_{1f}>\)
ctrl

addr       arb       aid       did

mRTW, IDd

Coh Prot

OQ

IQ

mRTW

Proc tags = I

S

S

S

S

S

S

$  P  $  P  $  P

$  P  $  P  $  P

Snoop tags => M

Proc tags = I

© Erik Hagersten | user.it.uu.se/~eh
A cascade of "write requests"

- A initially resides in CPU7’s cache
- CPU1: RTW, ID=a
- CPU2: RTW, ID=b
- ...
- CPU5: RTW, ID=f

\[
\begin{align*}
\text{CPU tags} & \quad \text{Snoop tags} \\
\begin{array}{c}
\text{I} \\
\text{I} \\
\text{I} \\
\text{S} \\
\end{array} & \quad \begin{array}{c}
\text{I} \\
\text{I} \\
\text{I} \\
\text{M} \\
\text{I} \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\text{IQ1} &= \langle \text{mRTW}_\text{IDa}, \text{fRTW}_\text{IDb} \rangle \\
\text{IQ2} &= \langle \text{mRTW}_\text{IDb}, \text{fRTW}_\text{IDc} \rangle \\
\text{...} \\
\text{IQ5} &= \langle \text{mRTW}_\text{IDf} \rangle \\
\text{...} \\
\text{IQ7} &= \langle \text{fRTW}_\text{IDA} \rangle
\end{align*}
\]
Implementing Sun’s SunFire 6800

Erik Hagersten
Uppsala University
Sweden
L2 cache = 8MB, snoop tags on-chip
CPU 1+GHz UltraSPARC III
Mem= 4+GB/CPU
FirePlane, 24 CPUs

ID = <CPU#, Uid>
FirePlane, 24 CPUs
FirePlane, 24 CPUs
FirePlane, 24 CPUs
FirePlane, 24 CPUs

CPU board


Here it is!!
FirePlane, 24 CPUs

ID = <CPU#, Uid>
Three options

- **COMA** cache-only (@SICS)
- **NUMA** non-uniform
- **UMA** uniform (a.k.a. SMP)
Directory-based snooping: NUMA. Per-cacheline info in the home node

A: Who has a copy
B: Who has a copy

Interconnect

Directory Protocol

State

Cache access

Thread

Thread
"Upgrade" in dir-based

Who has a copy

Who has a copy

A: Read A
Read A
...

B: Read B
Read A
...

Thread

Thread

Thread

Write A
Cache-to-cache in dir-based

A: Who has a copy

Thread
Read A
Read A
...
...
Read A

B: Who has a copy

Thread
Read B...
Read A

Forward

ReadRequest

ReadDemand

Ack

ReadA

Write A
Fully mapped directory

- $k$ Nodes
- Each node is the "home" for $1/k$ of the memory
- Dir entry per cacheline in home memory: $k$ presence-bits + 1 dirty-bit
- Requests are first sent to the home node’s CA
Reducing the Memory Overhead: SCI

--- Scalable Coherence Interface (SCI)

- home only holds pointer to rest of the directory info \([\text{log}(N) \text{ bits}]\)
- distributed linked list of copies, weaves through caches
  - cache tag has pointer, points to next cache with a copy
- on read, add yourself to head of the list (comm. needed)
- on write, propagate chain of invalidations down the list
- on replacement: remove yourself from the list
Cache Invalidation Patterns

Barnes-Hut Invalidation Patterns

Radiosity Invalidation Patterns
Overflow Schemes for Limited Pointers

- **Broadcast** (Dir\(_\text{iB}\))
  - broadcast bit turned on upon overflow
  - bad for widely-shared invalidated data

- **No-broadcast** (Dir\(_\text{iNB}\))
  - on overflow, new sharer replaces one of the old ones (invalidated)
  - bad for widely read data

- **Coarse vector** (Dir\(_\text{iCV}\))
  - change representation to a coarse vector, 1 bit per \(k\) nodes
  - on a write, invalidate all nodes that a bit corresponds to
cc-NUMA issues

- Memory placement is key!
- Gotta’ migrate data to where it’s being used
- Gotta’ have cache affinity
  - Long time between process switches in the OS
  - Reschedule processor on the CPU it ran last
- Origin 2000’s migration always turned off 😞
Sun’s WildFire System

Erik Hagersten
Uppsala University
Sweden
Sun’s WildFire System

- Runs **unmodified** SMP apps in a more scalable way than E6000
- Minot modifications to E6000 snooping required
- CPUs generate local address OR global address
- Global address --> no replication (NUMA)
- Coherent Memory Replication(\~Simple COMA@ SICS)
- Hardware support for detecting migration/replication pages
- Directory cache + address translation cache backed by memory
- Deterministic directory implementation (easy to verify)
WildFire:
One Solaris spanning four nodes

4 E6000 Systems
4 WFI boards
4 I/O boards
56 CPU/Memory boards
112 250 Mhz UltraSparc II
12 GB Memory 28 GFlops peak!
COMA: self-optimizing DSM

**COMA:**
- Self-optimizing architecture
- Problem at high memory pressure
- Complex hardware and coherence protocol
Adaptive S-COMA of Large SMPs

- A page may have space allocated in many nodes
- HW maintains memory coherence per cache line
- Replication under SW control --> simple HW (S-COMA)
- Adaptive replication algorithm in OS (R-NUMA)
- Coherent Memory Replication (CMR)
- Hierarchical affinity scheduler (HAS)
- Few large nodes -> simple interconnect and coherence protocol
A WildFire Node

- 16 slots with either CPUs, IO or...

- WildFire extension board ➔
  - Up to **28** UltraSPARC processors
  - Gigaplane™ bus has peak bw 2.67 GB/s
  - Local access time of 330ns (lmbench)
Sun WildFire Interface Board

- SRAM
- ADDR Controller
- Data Buffers
- Link Link Link

This space for rent
Sun WildFire Interface Board

Data Bus

Addr Bus

Links
WildFire as a vanilla "NUMA'
NUMA -- local memory access

Interconnect

Mem

Dir$

Mtag

Mem

Dir$

Mtag

Cache

Cache

Cache

Cache

Proc

Proc

Proc

Proc

Access right OK?
NUMA -- remote memory access

SRAM overhead = 10/512 = 2% (lower bound 2/512 = 0.4%)
Global Cache Coherence Prot.

Diagram showing the flow of data and interactions between components such as Memory (Mem), Interface (I/F), Directory (Dir$), Tag (Mtag), Cache, and Processor (Proc). The diagram highlights the process of modifying directory entries and replying with data. The text boxes mention 'Mod dir entry' and 'Reply(Data)', indicating the actions performed in the coherence protocol.

Access right changes is highlighted, indicating the importance of controlling access in the protocol.
NUMA -- local memory access

Interconnect

Mem

I/F

Dir$

Mtag

Access right OK? NO!!

Mem

I/F

Dir$

Mtag

Cache

Proc

...)

Cache

Proc

Cache

Proc

Cache

Proc

Cache

Proc
Gigaplane Bus Timing

Address
- Rd A
- Rd B
- Rd C
- Xxx

State
- Share
- Own

Arbitration
- 1
- 2
- 4,5

Tag
- A
- D
- A
- D

Status
- OK
- Cancel

Data
- D0
- D1

Share ~ Own

Own

Cancel
WildFire Bus Extensions

Ignore transaction squashes an ongoing transaction => not put in IQ
WildFire eventually reissues the same transaction
RTSF -- a new transaction sends data to CPU and memory
WildFire Directory -- only 4 nodes!!

- k nodes (with one or more procs).
- With each cache-block in memory: k presence-bits, 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit

**ReadRequest from main memory by processor i:**
- If dirty-bit OFF then { read from main memory; turn p[i] ON}
- If dirty-bit ON then { recall line from dirty proc (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to i;}

...
NUMA "detecting excess misses"

I thought you had the data!!* 😞
Detecting a page for replication

Data w/ E-miss-bit

Associative Counters

Mem

I/F

Dir$

Mtag

Mem

I/F

Dir$

Mtag

Cache

Proc

...
OS Initializes a CMR page

Interconnect

Mem

I/F

AT

Dir$

Mtag

Mem

I/F

AT$

Dir$

Mtag

VA->PA

New V->P mapping in this node

Proc

Cache

Proc

Cache

Proc

Cache

Proc

Cache

Proc

Init acc right to INV

/CL

Address Translation Grey. --> Yel..

OS Initializes a CMR page
An access to a CMR page

Interconnect

Mem

AT

Dir$

Mtag

AT$

Dir$

Mtag

Access right OK?

Cache

Proc

Cache

Proc

Cache

Proc

Cache

Proc

Proc

...
An access to a CMR page (miss)

Address Translation (AT) overhead = 8B/8kB = 0.1%
No extra latency added
An access to a CMR page (miss)

Interconnect

Change MTAG to “shared”
An access to a CMR page (hit)

Interconnect

Mem

AT

Dir$

Mtag

I/F

Cache

Proc

... 

Cache

Proc

AT$

Dir$

Mtag

I/F

Cache

Proc

Access right OK? YES!
Deterministic Directory

- MOSI protocol, fully mapped directory (one bit/node)
- Directory blocking: one outstanding trans/cache line
- Directory blocks new requests until completion received
- The directory state and cache state always in agreement (except for silent replacement...)

(a) Write: remote shared

(b) Read: Remote dirty

(c) Writeback
"Physical" memory

Only "promising" pages are replicated
OS dynamically limits the amount of replication
Solaris CMR changes in the hat_layer (=port)
Advantages of Multiprocessor Nodes

Pros:
- amortization of fixed node costs over multiple processors
- can use commodity SMPs
- fewer nodes to keep track of in the directory
- much communication may stay within node (NUCA)
- can share “node caches” (WildFire: Coherent Memory Replication)

Cons:
- bandwidth shared among processors and interface
- bus may increases latency to local memory
- snoopy bus at remote node increases delays there too
Memory cost of replication

Example: Replicate 10% of data in all nodes

- 50 nodes, each with 2 CPUs
  ==> 490% overhead

- 4 nodes, each with 25 CPUs
  ==> 30% overhead
**Does migration/replication help?**

NAS parallel Benchmark Study (Execution time in seconds)

[M. Bull, EPCC 2002]

<table>
<thead>
<tr>
<th></th>
<th>Shallow</th>
<th>BT</th>
<th>SP</th>
<th>MG</th>
<th>CG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Initial Plac.</td>
<td>Initial Placement</td>
<td>No Initial Plac.</td>
<td>Initial Placement</td>
<td>No Initial Plac.</td>
</tr>
<tr>
<td></td>
<td>No migr</td>
<td>Migr</td>
<td>NoMigr</td>
<td>Migr</td>
<td>No migr</td>
</tr>
<tr>
<td>No Repl</td>
<td>7.2</td>
<td>6.2</td>
<td>6.1</td>
<td>6.1</td>
<td>960</td>
</tr>
<tr>
<td>Repl</td>
<td>7.2</td>
<td>6.2</td>
<td>6.1</td>
<td>6.1</td>
<td>960</td>
</tr>
</tbody>
</table>

**Unopt.**

<table>
<thead>
<tr>
<th></th>
<th>FT</th>
<th>Unopt</th>
<th>HWopt</th>
<th>SWopt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Initial Plac.</td>
<td>Initial Placement</td>
<td>No Initial Plac.</td>
<td>Initial Placement</td>
</tr>
<tr>
<td></td>
<td>No migr</td>
<td>Migr</td>
<td>NoMigr</td>
<td>Migr</td>
</tr>
<tr>
<td>No Repl</td>
<td>520</td>
<td>330</td>
<td>380</td>
<td>260</td>
</tr>
<tr>
<td>Repl</td>
<td>250</td>
<td>260</td>
<td>190</td>
<td>200</td>
</tr>
</tbody>
</table>

**No Initial Plac.**

- Shallow: No migr 26, Migr 5.9, NoMigr 6.1, Migr 6.1
- BT: No migr 960, Migr 610, NoMigr 620, Migr 600
- SP: No migr 1540, Migr 780, NoMigr 760, Migr 780
- MG: No migr 230, Migr 230, NoMigr 240, Migr 230
- CG: No migr 1060, Migr 700, NoMigr 940, Migr 700
WildFire’s Technology Limits

- Dir$ = 8 b/line
- Mtag = 2 b/line
- Dir $ reach >> sum(cache size)
- SRAM size = DRAMsize/256
  Snoop frequency
- Hard to make busses faster
- Slow interconnect

Interconnect

Mem

Cache

Proc

Hard to make busses faster

SRAM size = DRAMsize/256
Snoop frequency

Slow interconnect
Sun’s SunFire 15k/25k

Erik Hagersten
Uppsala University
Sweden
StarCat
Sun Fire 15k/25k
(used at Lab2)
StarCat Coherence Mechanism

Active Backplane

18x18 addr X-bar

Expender board

Dir$

Glob-coh

CPU board

Remote request


Data Rep.

Addr (snoop)

$ CPU

MTAG Check!

DATA+

MTAG+

ECC = 576 bits

Data Repeater

MTAG

Check!

Remote request

DATA+

MTAG+

ECC = 576 bits
Allocate Dir$ entry only for write requests. Speculate on clean data on Dir$ miss.
StarCat Performance Data

Active Backplane

Expander board

CPU board

18x18 addr X-bar

18x18 addr X-bar

Dir$

Glob-coh

Data Rep.


Lat = 200-340ns
GBW=43GB/s
LBW=86GB/s

Up to 104 CPU (trading for I/O)