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
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 either 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
An E6000 Proc Board

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

288 signals = 256 data + ECC
An I/O Board

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

288 signals = 256 data + ECC

Address Controller

Data

Addr

I/O

I/O
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

---

**Data Signals**

<table>
<thead>
<tr>
<th>Addr Signals</th>
<th>Data Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address/CMD</td>
<td>Data</td>
</tr>
<tr>
<td>Address/CMD</td>
<td>Data</td>
</tr>
<tr>
<td>Address/CMD</td>
<td>Data</td>
</tr>
</tbody>
</table>

**Bus arbitration**

- A

---

“Mem Access Delay”

---

**Time**

- A → D
- D → D
EXAMPLE: MOESI snoop

RTS

Data

1: Mem A
   $: A(S)
   B(M) 
   Ld A

4: Mem
   $: A(S)
   St B

5: Mem B
   $: A(S)
   Ld B

RTW

1: Mem A
   $: A(S)
   B(M→I) 

4: Mem
   $: A(S)
   B(M→I) 
   St B

5: Mem B
   $: A(S)
   Ld B

RTS

1: Mem A
   $: A(S)
   B(I) 

4: Mem
   $: A(S)
   B(I) 
   owned

5: Mem B
   $: A(S)
   Ld B
Gigaplane Bus Timing (MOESI)

Example MOESI snoop:
CPU1 sends RTS (A), data in MEM1 \(\Rightarrow\) state S
CPU4 sends RTW(B), in MEM5, owned by CPU1
CPU5 sends RTS (B), owned by CPU4

MEM2 arbiting for Data
Earliest possible time (may also be later)
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

UPPSALA UNIVERSITY

AVDARK 2011

ctrl

addr arb aid did

OQ Prot Coh

IQ

S

S

S

Access-right Stat

Obligation State

S

$ P S

$ P S

$ P S

$ P S

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

© Erik Hagersten | user.it.uu.se/~eh
Timing of a single read trans
Board 1 reading from mem 2

= Fixed latency
= Shortest possible latency
Protocol tuned for timing

SRAM lookup

Address
State
Arbitration
DataID
Status
Data

Addr decode
DRAM access

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 IQ\textsubscript{trans} retired when data arrives
Later requests for A queued in IQ\textsubscript{CPU1} behind mRTO
IQ\textsubscript{CPU1} will eventually store: \langle m\text{RTW}_{d}, f\text{RTS}_{e}, f\text{RTW}_{f} \rangle
A cascade of "write requests"

A initially resides in CPU7’s cache

On the bus:

- CPU1: RTW, ID=a
- CPU2: RTW, ID=b
- ...
- CPU5: RTW, ID=f

<table>
<thead>
<tr>
<th>CPU tags</th>
<th>Snoop tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="I" /></td>
<td><img src="image" alt="I" /></td>
</tr>
<tr>
<td><img src="image" alt="I" /></td>
<td><img src="image" alt="I" /></td>
</tr>
<tr>
<td><img src="image" alt="I" /></td>
<td><img src="image" alt="M" /></td>
</tr>
<tr>
<td><img src="image" alt="S" /></td>
<td><img src="image" alt="I" /></td>
</tr>
</tbody>
</table>

IQ1 = <mRTW_{1Da}, fRTW_{1Db}>
IQ2 = <mRTW_{1Db}, fRTW_{1Dc}>
...
IQ5 = <mRTW_{1Df}>
...
IQ7 = < fRTW_{1Da}>
Implementing Sun’s SunFire 6800

Erik Hagersten
Uppsala University
Sweden
FirePlane, 24 CPUs

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

CPU board

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

CPU board


CPU

CPU

CPU

CPU

FirePlane, 24 CPUs
FirePlane, 24 CPUs
FirePlane, 24 CPUs
FirePlane, 24 CPUs

CPU board


Data, ID

ID = <CPU#, Uid>

Data Repeater

© Erik Hagersten | user.it.uu.se/~eh
Scalable Shared-Memory Implementations

Erik Hagersten
Uppsala University
Sweden
Cache-to-cache in snoop-based

A: Read A
Read A
...
...
Read A

Thread

B: Read B
...
Read A

Thread

BusRTS

My RTS → wait for data

Gotta answer

© Erik Hagersten | user.it.uu.se/~eh
"Upgrade" in dir-based

Who has a copy

A: Read A
Read A
... Read A
... Read A
Write A

B: Read B
... Read A

INV

ACK

ACK

Acknowledgments

Thread

Thread

Thread
Cache-to-cache in dir-based

Who has a copy

ReadRequest

ReadDemand

Ack

Forward

Thread

Thread

Thread

A: Read A
   Read A
   ...
   ...
   Read A

B: Read B
   Read A
   ...
   ...
   Read A

Read A
   Read A
   ...
   Write A

Who has a copy
Directory-based coherence: per-cacheline info in the memory

Directory Protocol

A: Cache access
B: Cache access

Directory state
Directory-based snooping: NUMA. Per-cacheline info in the home node
Why directory-based

- P2P messages → high bandwidth
- Suits out-of-the-box coherence
- Much more scalable!

Note:
- Dir-based can be used to build a uniform-memory architecture (UMA)
- Bandwidth will be great!!
- Memory latency will be OK
- Cache-to-cache latency will not!
- Memory overhead can be high (storing directory... )
Cache-to-cache in snoop-based

A: $  B: $  BusRTS

Thread

Read A
Read A
...  ...  ...  Read A
...  ...  ...  Write A

My RTS  \(\rightarrow\) wait for data

Gotta answer

Read B
...  ...  Read A
Cache-to-cache in dir-based

Who has a copy

ReadRequest

ReadDemand

Ack

Forward

Thread

Read A
Read A
...
...
...
Read A

Thread

Read A
...
...
Write A

Thread

Read B
...
...
Read 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 \([\log(N) \text{ bits}]\)
• 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 (DirB)**
  - broadcast bit turned on upon overflow
  - bad for widely-shared invalidated data

- **No-broadcast (DirNB)**
  - on overflow, new sharer replaces one of the old ones (invalidated)
  - bad for widely read data

- **Coarse vector (DirCV)**
  - change representation to a coarse vector, 1 bit per k nodes
  - on a write, invalidate all nodes that a bit corresponds to
Directory cache

```
A:  
  Dir$  
  Directory Protocol  
  State  
  Cache access  

Read A  
Read A  
…  
…  

B:  
  Dir$  
  Directory Protocol  
  State  
  Cache access  

Thread  
Thread  

Interconnect

INV-REQ

INV-DEM

ACK-REP

ACK-CMP
```
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
- SGI Origin 2000’s migration always turned off 😞
Three options for shared memory

- **COMA** (cache-only) (@SICS)
- **NUMA** (non-uniform)
- **UMA** (uniform, a.k.a. SMP)
Sun’s WildFire System

Erik Hagersten
Uppsala University
Sweden
### Sun’s WildFire System

- Runs **unmodified** SMP apps in a more scalable way than E6000
- Minor 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

ccNUMA

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

Links

Data Bus

Addr Bus
WildFire as a vanilla "NUMA"
NUMA -- local memory access

Interconnect

Access right OK?

Mem

Cache

Proc

I/F

Dir$

Mtag

Mem

Cache

Proc

I/F

Dir$

Mtag

Mem

Cache

Proc

I/F

Dir$

Mtag

Mem

Cache

Proc
NUMA -- remote memory access

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

Mod dir entry

Reply(Data)

Mem

I/F

Cache

Proc

...
NUMA -- local memory access

Interconnect

Mem

Cache

Proc

I/F

Dir$

Mtag

Access right OK? NO!!

Mem

Cache

Proc

I/F

Dir$

Mtag
Gigaplane Bus Timing

AVDARK
2011

Dept of Information Technology | www.it.uu.se
© Erik Hagersten | user.it.uu.se/~eh
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

### Diagram

```
            P
           /
        Cache          Cache
            /
       Interconnection Network
            /
        Memory          Directory
            |             |
        presence bits  dirty 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!!*

Dir$ 8b/line
Mtag 2b/line
Detecting a page for replication

Data w/ E-miss-bit

Associative Counters
OS Initializes a CMR page

Interconnect

Mem

I/F

AT

Dir$

Mtag

Mem

I/F

AT$

/page

Dir$

Mtag

Cache

Proc

Cache

Proc

VA→PA

New V→P mapping in this node

Address Translation Grey. <--> Yel..

Init acc right to INV

CL
An access to a CMR page
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

Mem

Cache

Proc

Change MTAG to "shared"
An access to a CMR page (hit)
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
Replication Issues Revisited

"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]

### Shallow

<table>
<thead>
<tr>
<th></th>
<th>No Initial Plac.</th>
<th>Initial Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No migr</td>
<td>Migr</td>
</tr>
<tr>
<td>No Repl</td>
<td>26</td>
<td>5.9</td>
</tr>
<tr>
<td>Repl</td>
<td>7.2</td>
<td>6.2</td>
</tr>
</tbody>
</table>

### BT

<table>
<thead>
<tr>
<th></th>
<th>No Initial Plac.</th>
<th>Initial Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No migr</td>
<td>Migr</td>
</tr>
<tr>
<td>No Repl</td>
<td>960</td>
<td>610</td>
</tr>
<tr>
<td>Repl</td>
<td>590</td>
<td>580</td>
</tr>
</tbody>
</table>

### MG

<table>
<thead>
<tr>
<th></th>
<th>No Initial Plac.</th>
<th>Initial Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No migr</td>
<td>Migr</td>
</tr>
<tr>
<td>No Repl</td>
<td>230</td>
<td>230</td>
</tr>
<tr>
<td>Repl</td>
<td>220</td>
<td>220</td>
</tr>
</tbody>
</table>

### CG

<table>
<thead>
<tr>
<th></th>
<th>No Initial Plac.</th>
<th>Initial Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No migr</td>
<td>Migr</td>
</tr>
<tr>
<td>No Repl</td>
<td>1060</td>
<td>700</td>
</tr>
<tr>
<td>Repl</td>
<td>300</td>
<td>280</td>
</tr>
</tbody>
</table>
WildFire’s Technology Limits

Interconnect

- Dir $ = 8 b/line
- Mtag = 2 b/line

SRAM size = DRAMsize/256
Snoop frequency

Hard to make busses faster

Dir $ reach >> sum(cache size)

Slow interconnect
Sun’s SunFire 15k/25k

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

Active Backplane

18x18 addr X-bar

Expander board

Data Repeater

18x18 addr X-bar

CPU board

Remote request

Glob-coh prot


MTAG Check!

Dir$

Data Rep.

Addr (snoop)

CPU

MTAG

Check!

Remoterequest

DATA

MTAG+

ECC =576 bits

Data Repeater

Remote request

DATA
StarCat, 72 CPUs

Active Backplane

18x18 addr X-bar

Expander board

Dir$

Glob-coh prot

CPU board

Allocate Dir$ entry only for write requests. Speculate on clean data on Dir$ miss

Data Rep.

Glob-coh prot


18x18 addr X-bar

Data Repeater

WildCat coherence w/o CMR & w/ faster interconnect
Directory cache, but no directory (broadcast on Dir$ miss)

A: Cache access

B: Cache access

Directory Protocol

State

Thread

Interconnect

Thread
StarCat Performance Data

Active Backplane
18x18 addr X-bar
18x18 addr X-bar

Expander board
Dir$
Glob-coh prot
Data Rep.

CPU board

Data Repeater

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

Up to 104 CPU
(trading for I/O)