

Scalable Multi-Core Processor with Ring Network and Share Memory Space Support.

Shaomin Zhai

XiDian University

Computer Architecture Final Project Report

2016/3/24

### Abstract

Due to fundamental physical constraints, the rate of growth of central processing unit clock speeds is beginning to fall short of the predictions set forth by Moore's Law. As a result, researchers are now exploiting the performance gains possible from multi-core processing.

This project aims to introduce some of the difficulties presented by multi-core computing and what is needed to achieve an operational multi-core system.

**Key word:** scalable, reply-forwarding, ring network, credit-flow control, directory-based cache coherence, Shared address space.

## 0.Introduction

As you know, Processor design needs to consider power, area, and clock frequency. For parallel computer architecture, what should also be considered, is that delivery mechanism plurality of core network architecture, cache coherence protocols, network bandwidth, network message. All of these factors affect our decision to design processor architecture, and micro-architecture, which finally affects performance, price and power consumption of a processor. While this gives us a lot of design space, but also brings us a lot of problems that can be analysed and solved by processor model including software model and hardware model. These two models are both necessary for designing processor, here I need some hardwares to demonstrate my project.

**I will appreciate if you are willing to give financial assistance to me for my project! I'll need about 20 thousand RMB, which will be used for project devices, books, materials, consulting fee, service fee and so on, as the subvention of my project.**

In a shared memory multiprocessor system where each processor has its own data memory cache, care must be given to make sure that each processor receives correct data from its cache, regardless of how other processors may be affecting that memory address.

When the following two conditions are met, the cache correctly handles the memory accesses across the multiple processors, and is called cache coherent:

- 1, A value written by a processor is eventually visible to other processors
- 2, when multiple writes happen to the same location by multiple processor, all the processors see the write in the same order.

There are two general categories of cache coherency protocols:

- 1, Directory based protocols: where a single location keeps track of the status of memory blocks;
- 2, Snooping protocols: where each cache maintains the status of memory blocks and monitors activity from the other processors' caches. updating the status of the memory blocks if necessary.

A protocol is further specified as either write-invalidate, where a write to a memory location by one processor invalidates the corresponding cache line, if present, in the other processors' caches; or write-update, where the other processors update the data in their caches rather than invalidating them.

The protocols that I will implement and investigate for this project will be a MESI write-invalidate directory based protocols with write-back caches.

Agenda:

- 1, block diagrams of whole system
  - 2, Overview of Directory-Based Approaches, message format and interconnection network
  3. benchmarks
  4. future work and extensive
  5. references
- Appendix

# 1.Implementation Topology

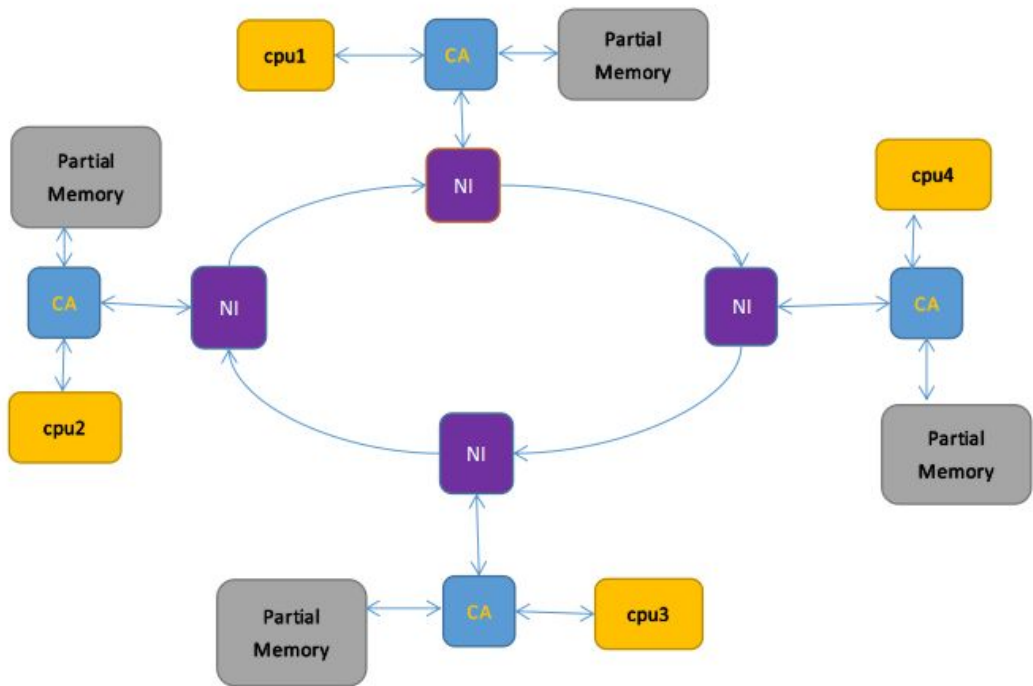


Fig. 1 Block diagram depiction of the hardware

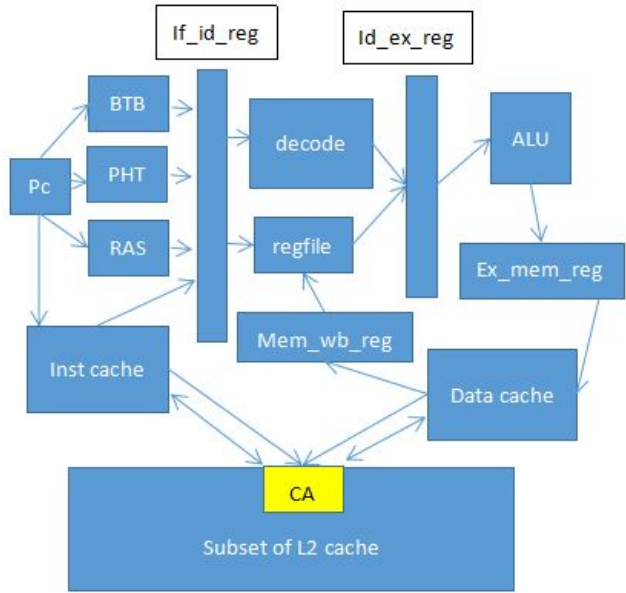


Fig.2 Block diagram depiction of cpu

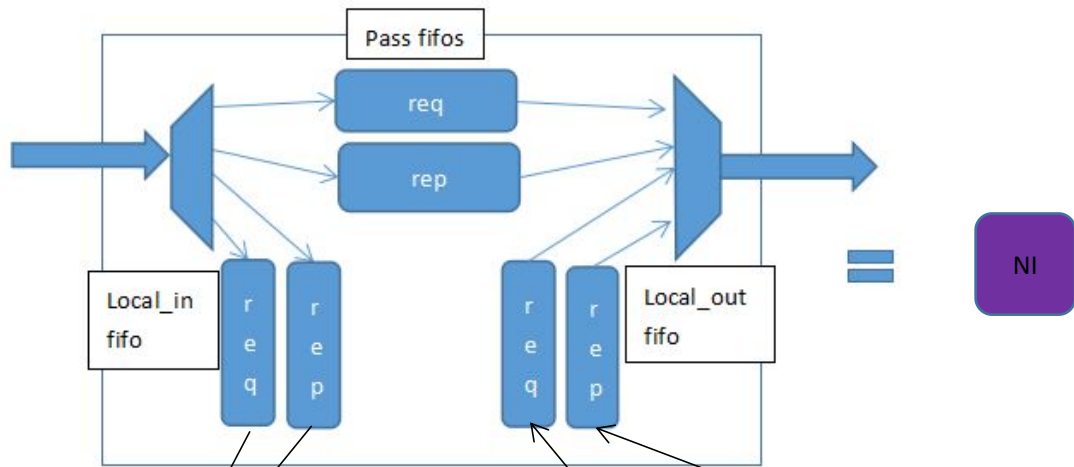


Fig 3 block diagram of network interface(NI).

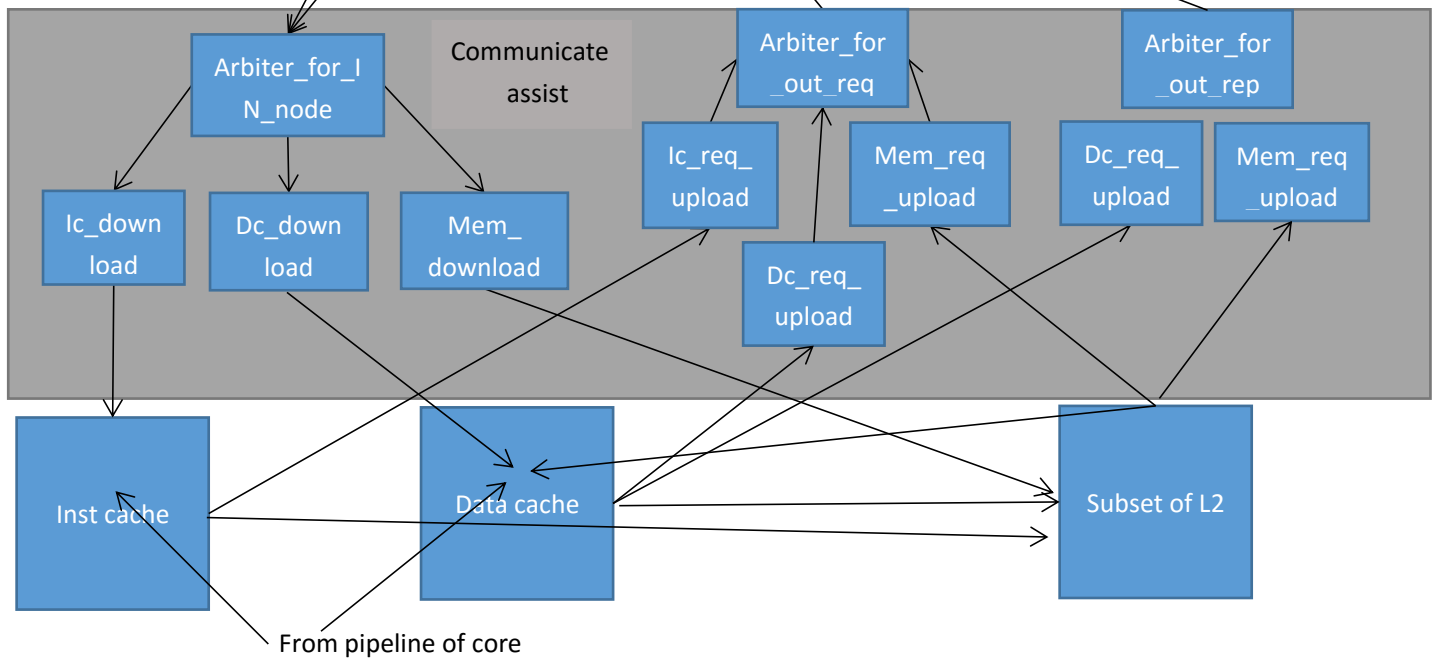


Fig 4 block diagram of communication assist(CA).

## 2 Overview of Directory-Based Approaches

This section begins by describing the directory scheme and how it might operate using cache states, directory states, and network transaction. It discusses the issues in state transition Diagram, directory structure and protocol level correctness one by one.

The following definitions will be useful throughout our discussion of directory protocols. For a Given cache or memory block:

The **home** node is the node in whose main memory the block is allocated.

The **dirty** node is the node that has a copy of block in its cache in modified (dirty) state. Note That the home node and the dirty node for a block may be the same node.

The **exclusive** node is the node that has a copy and that block in its cache in an exclusive state, either dirty or exclusive-clean. Thus, the dirty node is also the exclusive node.

The **local** node, or requesting node, is the node containing the processor that issues a request For the block.

The **owner** node is the node that currently holds the valid copy of a block and must supply the data when needed; this is either the home node, when the block is not in dirty state in a cache, or the dirty node.

Blocks whose home is **local** to the issuing processor are called locally allocated (or sometimes simply local) blocks, while all others are called remotely allocated (or **remote**).

### 2.1 some states of transition

**Cache states:** For each cache line, there are 5 possible states:

C-invalid (= Nothing): The accessed data is not resident in the cache.

C-shared (= Sh): The accessed data is resident in the cache, and possibly also cached at other sites. The data in memory is valid.

C-modified (= Ex): The accessed data is exclusively resident in this cache, and has been modified. Memory does not have the most up-to-date data.

C-transients (= read-Pending/write-pending): The accessed data is in a *transient* state (for example, the site has just issued a protocol request, but has not received the corresponding protocol reply).

**Home directory states:** For each memory block, there are 4 possible states:

$R(dir)$ : The memory block is shared by the sites specified in  $dir$  ( $dir$  is a set of sites). The data in memory is valid in this state. If  $dir$  is empty (i.e.,  $dir = \epsilon$ ), the memory block is not cached by any site.

$W(id)$ : The memory block is exclusively cached at site  $id$ , and has been modified at that site. Memory does not have the most up-to-date data.

$TR(dir)$ : The memory block is in a transient state waiting for the acknowledgements to the invalidation requests that the home site has issued.

$Tw(id)$ : The memory block is in a transient state waiting for a block exclusively cached at site  $id$  (i.e., in C-modified state) to make the memory block at the home site up-to-date.

### Base messages from MIT lecture notes( more details in reference).

Messages:

Cache to Memory requests: ShReq, ExReq

Memory to Cache requests: WbReq, InvReq, FlushReq

Cache to Memory responses: WbRep(v), InvRep, FlushRep(v)

Memory to Cache responses: ShRep(v), ExRep(v)

Operations on cache:

cache.state(a) -returns state s

cache.data(a) -returns data v

cache.setState(a,s), cache.setData(a,v), cache.invalidate(a)

inst = first(p2m); msg= first(m2c); mmsg = first(in)

No	Current State	Handling Message	Next State	Dequeue Message?	Action
1	C-nothing	Load	C-pending	No	ShReq(id,Home,a)
2	C-nothing	Store	C-pending	No	ExReq(id,Home,a)
3	C-nothing	WbReq(a)	C-nothing	Yes	None
4	C-nothing	FlushReq(a)	C-nothing	Yes	None
5	C-nothing	InvReq(a)	C-nothing	Yes	None
6	C-nothing	ShRep (a)	C-shared	Yes	updates cache with prefetch data
7	C-nothing	ExRep (a)	C-exclusive	Yes	updates cache with data
8	C-shared	Load	C-shared	Yes	Reads cache
9	C-shared	WbReq(a)	C-shared	Yes	None
10	C-shared	FlushReq(a)	C-nothing	Yes	InvRep(id, Home, a)
11	C-shared	InvReq(a)	C-nothing	Yes	InvRep(id, Home, a)
12	C-shared	ExRep(a)	C-exclusive	Yes	None
13	C-shared	(Voluntary Invalidate)	C-nothing	N/A	InvRep(id, Home, a)
14	C-exclusive	Load	C-exclusive	Yes	reads cache
15	C-exclusive	Store	C-exclusive	Yes	writes cache
16	C-exclusive	WbReq(a)	C-shared	Yes	WbRep(id, Home, data(a))
17	C-exclusive	FlushReq(a)	C-nothing	Yes	FlushRep(id, Home, data(a))
18	C-exclusive	(Voluntary Writeback)	C-shared	N/A	WbRep(id, Home, data(a))
19	C-exclusive	(Voluntary Flush)	C-nothing	N/A	FlushRep(id, Home, data(a))
20	C-pending	WbReq(a)	C-pending	Yes	None
21	C-pending	FlushReq(a)	C-pending	Yes	None
22	C-pending	InvReq(a)	C-pending	Yes	None
23	C-pending	ShRep(a)	C-shared	Yes	updates cache with data
24	C-pending	ExRep(a)	C-exclusive	Yes	update cache with data

Table 1,Cache State Transitions

No.	Current State	Message Received	Next State	Dequeue Message?	Action
1	$R(\text{dir}) \ \& \ (\text{dir} = \epsilon)$	ShReq(a)	$R(\{id\})$	Yes	ShRep(Home, id, data(a))
2	$R(\text{dir}) \ \& \ (\text{dir} = \epsilon)$	ExReq(a)	$W(id)$	Yes	ExRep(Home, id, data(a))
3	$R(\text{dir}) \ \& \ (\text{dir} = \epsilon)$	(Voluntary Prefetch)	$R(\{id\})$	N/A	ShRep(Home, id, data(a))
4	$R(\text{dir}) \ \& \ (id \notin \text{dir}) \ \& \ (\text{dir} \neq \epsilon)$	ShReq(a)	$R(\text{dir} + \{id\})$	Yes	ShRep(Home, id, data(a))
5	$R(\text{dir}) \ \& \ (id \notin \text{dir}) \ \& \ (\text{dir} \neq \epsilon)$	ExReq(a)	$Tr(\text{dir})$	No	InvReq(Home, dir, a)
6	$R(\text{dir}) \ \& \ (id \notin \text{dir}) \ \& \ (\text{dir} \neq \epsilon)$	(Voluntary Prefetch)	$R(\text{dir} + \{id\})$	N/A	ShRep(Home, id, data(a))
7	$R(\text{dir}) \ \& \ (\text{dir} = \{id\})$	ShReq(a)	$R(\text{dir})$	Yes	None
8	$R(\text{dir}) \ \& \ (\text{dir} = \{id\})$	ExReq(a)	$W(id)$	Yes	ExRep(Home, id, data(a))
9	$R(\text{dir}) \ \& \ (\text{dir} = \{id\})$	InvRep(a)	$R(\epsilon)$	Yes	None
10	$R(\text{dir}) \ \& \ (id \in \text{dir}) \ \& \ (\text{dir} \neq \{id\})$	ShReq(a)	$R(\text{dir})$	Yes	None
11	$R(\text{dir}) \ \& \ (id \in \text{dir}) \ \& \ (\text{dir} \neq \{id\})$	ExReq(a)	$Tr(\text{dir} - \{id\})$	No	InvReq(Home, dir - {id}, a)
12	$R(\text{dir}) \ \& \ (id \in \text{dir}) \ \& \ (\text{dir} \neq \{id\})$	InvRep(a)	$R(\text{dir} - \{id\})$	Yes	None
13	$W(id')$	ShReq(a)	$Tw(id')$	No	WbReq(Home, id', a)
14	$W(id')$	ExReq(a)	$Tw(id')$	No	FlushReq(Home, id', a)
15	$W(id)$	ExReq(a)	$W(id)$	Yes	None
16	$W(id)$	WbRep(a)	$R(\{id\})$	Yes	data $\rightarrow$ memory
17	$W(id)$	FlushRep(a)	$R(\epsilon)$	Yes	data $\rightarrow$ memory
18	$Tr(\text{dir}) \ \& \ (id \in \text{dir})$	InvRep(a)	$Tr(\text{dir} - \{id\})$	Yes	None
19	$Tr(\text{dir}) \ \& \ (id \notin \text{dir})$	InvRep(a)	$Tr(\text{dir})$	Yes	None
20	$Tw(id)$	WbRep(a)	$R(\{id\})$	Yes	data $\rightarrow$ memory
21	$Tw(id)$	FlushRep(a)	$R(\epsilon)$	Yes	data $\rightarrow$ memory

Table 2, Home Directory State Transitions, Messages sent from site id

**Note** content of section 2.1 is from Arvind Computer Science and Artificial Intelligence Lab M.I.T. November 14, 2005 lecture notes

## 2.2 The following rules is used in reply-forwarding (which is my effort)

### Background: protocol optimization

In general, there are three classes of techniques for improving performance:

- (i) protocol optimization ,
- (ii) high-level machine organization,
- (iii) hardware specialization to reduce latency and occupancy and increase bandwidth.

I want to use the first technique to reduce messages and critical transaction path.

Here are 3 kinds of options for me to try.

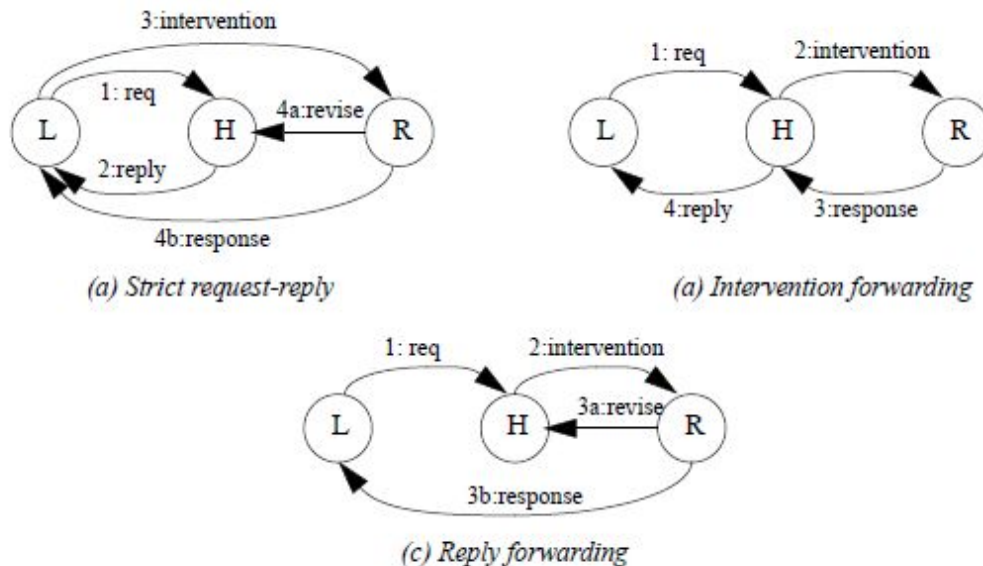


fig 5. Reducing latency through forwarding

The case shown is of a write request in node 1 to a block in shared state.let's assume node2,node3,node4 all share the data.L represent the local (requesting) node,H is the Home for block,and R is the remote owner node that has the exclusive copy of the block.

Because this system is a distributed-memory system,Every node has a partial memory ,so when a read-miss or write-miss arrives at communication assist,the CA will translate the physical address into a node id ,which can be used to find the destination node ,and a local address that will be used to look up information at home node,then use the node-id to check whether the miss will be handle locally or remotely. Of course, if the home node is the same as the requesting node, then no network transactions or messages are generated and it is a locally satisfied miss.

For (a) strict request-reply option,the CA in local node finds that it needs to generate a request (process 1)to the remote home,when the state in the cache line is invalid or shared,or the value indexed by address is not in the cache. When the request arrives at home node,the CA will decode the message to look up the directory states and find there are some other nodes sharing the data . Then the CA will change shared to TR(dir){dir={0,1,1,1}} waiting for invalidation responses returning of node2,3,4 and send reply including data back to local (requester) node with dir{0,1,1,1} information(process 2),Which will be used by local node to send 3 invalidation requests to NI,further to the corresponding node(process 3).note the the first request still in the home FIFO or other storage component used for the final state set for node1's write request.when the intervention(notd by 3) arrives at remote nodes ,the nodes immediately change the state in their cache from shared to invalid,in the meantime sending back invalidation responses back to local node and revision to home node.once the local node receive all the



needed responses ,it can actually write the data,and then local CA is able to process nest request.

When home node receive all revisions required,it is able to change the state from TR(dir) to W{1,0,0,0}.

For (b) intervention forwarding ,once the home node receive the request,it will forward 3 invalidation requests directly to the shared nodes to tell them to invalid their share data.when the invalidation requests get to shared nodes,the responses will be sent to the home node telling him that the shared data is already invalidated,then the home node can send response to local node telling it is able to do the real write.

For (c) reply forwarding ,refer to (b),once the remote node receives the intervention,it then send revise to home node to change state from TR{0,1,1,1} to W{1,0,0,0},also send reply including data directly to local node

In conclusion,it seems the third option can get best optimization.

(a)strict request-reply has 4 critical transactions and  $1+1+3+3+3=11$  messages totally.

(b)intervention forwarding has 4 critical transactions and  $1+3+3+1=8$  messages totally.

(c)reply forwarding has 3 critical transactions and  $1+3+3+1=8$  messages totally.

Since reply forwarding seems good,I will apply it to my project.

## 2.3 .Combine reply-forwarding with basic rules

### Cache side rules

#### Load -hit rule

If Load (a)==inst & cache.state(a)is sh or ex

Then p2m.deq

M2p.enq(cache.data(a))

#### Load -miss rule

If load(a)==inst & cache.state(a) is nothing

Then c2m.enq(msg(id,home,shreq,a))

Cache.setstate(a,pending)

//blocking cache left request in p2m FIFO  
//when the transactions done ,need to the  
// do the same thing like load-hit rule!  
// so must FIFO order to promise correct!

#### Store -hit rule

If store(a,v)==inst& cache.state(a)is EX

Then P2m.deq;

M2p.enq(ack)

Cache.setdata(a,v)

#### Store -miss rule

If store(a,v)==inst & cache.state(a) is nothing or sh

Then c2m.enq(msg(id,home,exreq,a));

Cache.setstate(a,pending)

```
// If store(a,v)==inst & cache.state(a) is sh
// Then c2m.enq(msg(id,home,upgrade,a));
Cache.setstate(a,pending)
//blocking cache left request in p2m FIFO
//when the transactions done ,need to the
// do the same thing like load-hit rule!
// so must FIFO order to promise correct!
```

## Processing shreq messages(at home node)

### Uncached or outstanding shared copies

```
if Msg(id,home,shreq,a)==mmsg & m.state(a) is R(dir)& id not in dir
Then inhome.deq; // don't need further requests to accomplish.
M.setstate(a,R(dir+id));
Out.enq(msg(home,id,shrep,a,m.data(a)))
```

### Outstanding Exclusive copy

```
If mmsg==msg(id,home,shreq,a) & m.state(a) is W(id') &(id!=id')
Then m.setstate(a,Tw(id'));
Outhome.enq(msg(home,id',wbreq,a,id)) //the node id' will directly send
// reply (with data) to the requester
//and revise to home .

//Then local cache will p2m.deq(inst)
//and home node will inhome.deq(mmsg)! While in the meantime the processor
//has to wait for completion and cache controller can't handle more inst,
//and in addition, inhome FIFO has to wait for the head msg to complete.
```

## Processing EXreq messages (at home node)

### Uncached or cached only at the requester cache

```
If mmsg==msg(id,home,exreq,a) & m.state(a) is R(dir) &(dir is empty or has only
id)
Then in.deq
M.setstate(a,W(id))
Outhome.enq(msg(home,id,exrep,a,m.data(a)))
```

### Outstanding shared copies

```
If mmsg==msg(id,home,exreq,a) & m.state(a) is R (dir) &!(dir is empty or has
```

only id)

Then `m.setstate(a,Tr(dir-id))`

`Outhome.enq(multicast(home,dir-{id},invreq,a,id))`

**\*\*\*\*\*** `note :Outhome.enq(msg(home,id,sh->exrep,a,m.data(a),  
INVRep_vector))`

*// sh->exrep should be treat differently in cpu side cache controler!*

*//note:when the reply\_forwarding msg*

*//arrives at node id,the id can't write data until all the needed*

*//invresps get to the id.this is actually **reply forwarding!***

### Outstanding Exclusive copy

If `mmsg==msg(id,home,Exreq,a) & m.state(a) is W(id') &(id'!=id)`

Then `m.setstate(a,Tw(id'))`

`Outhome.enq(msg(home,id',flushreq,a,id))` *//note: when this msg arrives at*

*//node id' ,the id' change its cache state to inv,and send Exrep msg with data to*

*//id.meanwhile send flushrep to home ,then home change state to W(id).*

## Processing reply messages (at cache)

### Shrep

If `msg=={msg(home,id,shreq,a,v) or msg(id',id,wb->shrep,a,v)}` `&(cache.state(a)`  
`must be pending or nothing)`

Then `m2c.deq`

`Cache.setstate(a,sh)`

`Cache.setdata(a,v)`

### Exrep

If `msg=={ msg(home,id,Exrep,a,v)`

`or msg({dir-id},id,sh->exrep,a,m.data(a))`

`or msg(id',id,flush->exrep,a,v) }` `&(cache.state(a) must be pending or`

`nothing)`

Then `m2c.deq`

`Cache.setstate(a,EX)`

`Cache.sedata(a,v)`

*// since one outstanding at any given time,i think i can drop address in the reply*

*//msg,which can cut down the number of flits.Assume flits has 16 bits,then we can*

*//reduce 2 flits average when need to rerply! I think it's a good idea to improve ipc.*

## Processing InvReq message (at cache)

### InvReq

If `msg==msg(home,id'',InvReq,a,id) & cache.state(a) is sh`

```

Then  m2c.deq
      Cache.invalidate(a)
      C2m.enq(msg(id'',id,InvRep,a))
      C2m.enq(msg(id'',home,InvRep,a))

```

```

If  msg(Home,id,InvReq,a) == msg & cache.state(a) is Nothing or Pending
Then  m2c.deq    //occurs when the shared data was evicted before!

```

## Processing WbReq message (at cache)

### wbReq

```

If  msg==msg(home,id',wbreq,a,id) & cache.state(a) is EX
Then  m2c.deq
      Cache.setstate(a,sh)
      C2m.enq(msg(id',home,wbrep,a,cache.data(a)))
      C2m.enq(msg(id',id,shrep,a,cache.data(a)))
If  msg==msg(home,id',wbreq,a,id) & cache.state(a) is sh or nothing or
pending
Then  m2c.deq

```

## Processing Flushreq message

### Flushreq

```

If  msg==msg(home,id',flushreq,a,id) & cache.state(a) is EX
Then  m2c.deq
      Cache.invalidate(a)
      C2m.enq(msg(id',home,flushrep,a,cache.data(a)))
      C2m.enq(msg(id',id,exrep,a,cache.data(a)))

```

**Note:because use NACKs to avoid multiple request on the same block ,the following two cases shouldn't happen in the system!**

```

/*****
If  msg==msg(home,id',flushreq,a,id) & cache.state(a) is sh
Then  m2c.deq
      Cache.invalidate(a)
      C2m.enq(msg(id',home,exrep,a)
      C2m.enq(msg(id',id,exrep,a))
// I think this case occur when ex_cache evict the cache line ,
//then read again in its cache during home send flushreq .

```

```

If msg==msg(home,id',flushreq,a,id) & cache.state(a) is nothing or pending
Then m2c.deq
    Cache.invalidate(a)
    C2m.enq(msg(id',home,flushrep,a,cache.data(a)))
    C2m.enq(msg(id',id,exrep,a,cache.data(a)))

```

*//after home sends flushreq,owner evicted the exclusive cache line  
 //so remain in nothing/then read it again but hadn't received data,  
 //so remain pending .*

\*\*\*\*\*/

## Processing Reply invRep message (at home)

### InvRep

```

If mmsg==msg(id,home,InvRep,a,id') & m.state(a) is Tr(dir)
Then deq mmsg
    If(dir!=id)
        M.settstate(a,Tr(dir-{id}))
    Else M.setstate(a,W(id'))

```

*// now dir doesn't include requester id! Here id'=requester id*

```

If mmsg==msg(id,home,InvRep,a) & m.state(a) is R(dir)
Then deq mmsg
    M.settstate(a,R(dir-{id}))

```

*// note: this can happen when a cache line in sh state is evicted.  
 // the cache controller should generate this InvRep msg.*

## Processing Reply WbRep message (at home)

### wbRep

```

If mmsg==msg(id,home,wbRep,a,v) & m.state(a) must be Tw(id) or W(id)
Then deq mmsg
    M.setstate(a,R(id))
    M.setdata(a,v)

```

### flushRep

```

If mmsg==msg(id,home,auto->flushRep,a,v) & m.state(a) must be W(id)
Then deq mmsg
    M.setstate(a,R(empty))

```

```

        M.setdata(a,v)
    If  mmsg==msg(id,home,flushRep,a,v,id') & m.state(a) must be Tw(id)
    Then  deq mmsg
        M.setstate(a,W(id'))
        M.setdata(a,v)

```

## 2.4 message format

Having understood rules about actions in home and cache, i need to encode these msgs for correctness. The msgs sent between nodes have following formats:

Changed message categories	encode	cmd 4'bxxxx	0	1	cache or home
Cache to Memory requests: ShReq: <i>msg(id,home,shreq,a)</i>		5'b00000		1	
ExReq: <i>msg(id,home,exreq,a)</i>		5'b00001		1	
ScExReq: <i>msg(id,home,ScExreq,a)</i>		5'b00010		1	
InstReq: <i>msg(id,home,InstReq,a)</i>		5'b00110		1	
Memory to Cache requests: WbReq: <i>msg(home,id',wbreq,a,id)</i>		5'b00011		0	
InvReq: <i>msg(home,id'',InvReq,a,id)</i>		5'b00100		0	
FlushReq: <i>msg(home,id',flushreq,a,id)</i>		5'b00101		0	
Sc->Invreq: <i>msg(home,id',scINVreq,a,id)</i>		5'b00110		0	
Cache to Memory responses: WbRep(v): <i>msg(id,home,wbRep,a,v)</i>		5'b10000		1	
invRep: <i>msg(id,home,C2HInvRep,a)</i>		5'b10001		1	
FlushRep(v) : <i>msg(id,home,flushRep,a)</i>		5'b10010		1	
flushfail_rep: <i>msg(id,home,flushfail,a)</i>		5'b10110		1	
//id of this kind is requester id,which launch the transition					
wbfail_rep: <i>msg(id,home,wbfail,a)</i>		5'b10111		1	
//id of this kind is requester id,which launch the transition					
Auto->flushRep: <i>msg(id,home,auto-&gt;flushRep,a,v)</i>		5'b10011		1	
cache/memory to Cache responses:					
Wb-> sh-> ShRep(v): <i>msg(id'/home,id,shrep,a,cache.data(a))</i>		5'b11000		0	
Flu-> R(id)-> ExRep(v): <i>msg(id'/home,id,exrep,a,cache.data(a))</i>		5'b11001		0	
Sh-> ExRep(v): <i>msg(home,id,exrep,a,cache.data(a),INV_vector)</i>		5'b11010		0	
scFlushrep: <i>msg(home,id,scFlushrep)</i>		5'b11100		0	
InstRep: <i>msg(home,id,Instrep,v)</i>		5'b10100		0	
Cache to cache responses:					
invRep: <i>msg(id',id,C2CInvrep,)</i>		5'b11011		0	

## Flit has 16 bits!

The head flit contains :dest\_id[15:14] ,  
cacheORhome1[13], //cache :0 ;home :1;  
src\_id[12:11],  
cacheORhome2[10],  
cmd[9:5],  
requester\_id(if needed)[4:3],  
[3:0] required INVREP vector.

Example:[3:0]={1,1,0,0}, meaning cache in node3 and cache in node2 need to reply the InvReps.

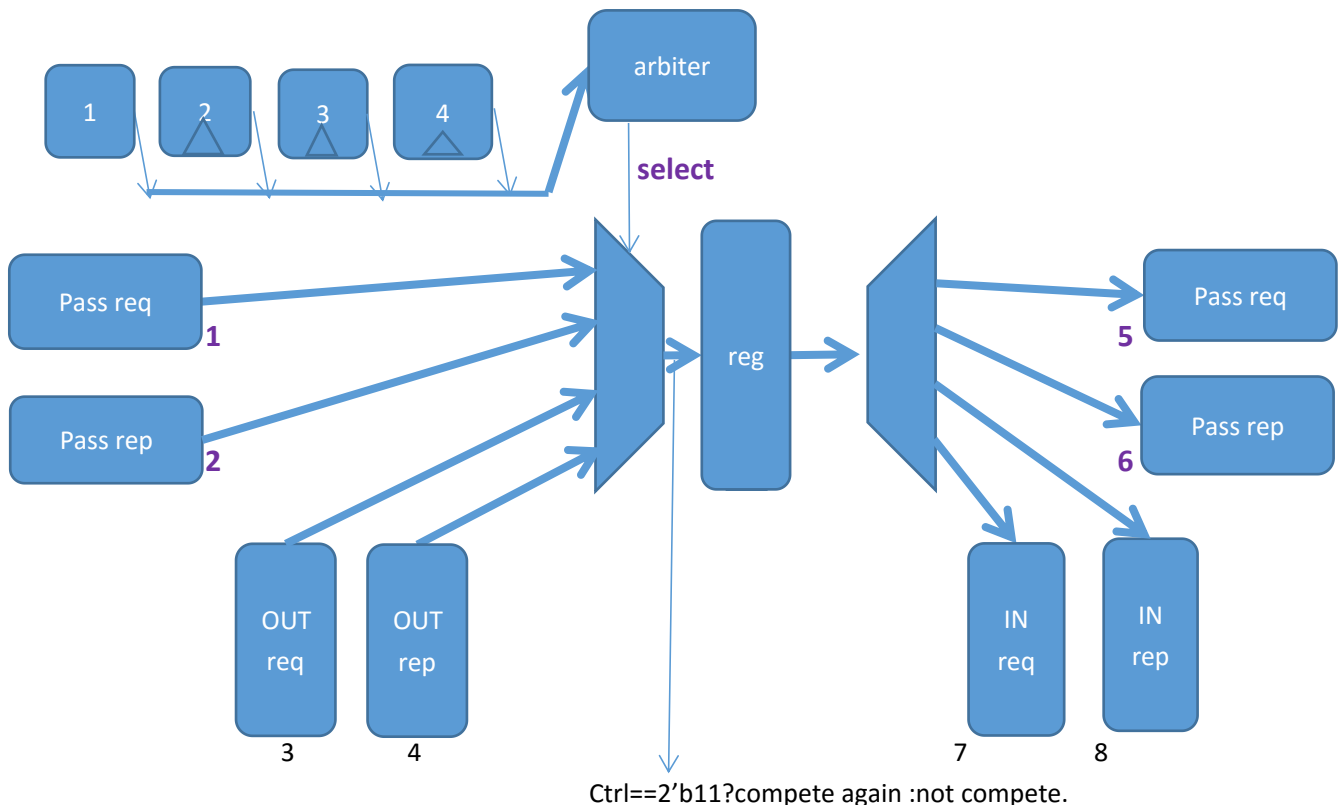
Since only 4 nodes 2 bits for id ,that's enough! cacheORhome indicate whether home id or cache id.

The body flits are address flits (If needed,determined by cmd)  
and data flits (If needed,determined by cmd).

Along with flit links, there will be some control signals to guide flits flow through the network.

The ring links are short,wide,and synchronous. Each unidirectional link is 16 data bits ,2 ctrl bits, 8 credit flow ctrl bits(reverse direction compared with others). Flits are 16 bits. 2 ctrl bits identify the flit type (00 :no info, 01:head flit, 10:body flit, 11:last body flit).8 credit flow ctrl bits are divided into two 4-bits flow ctrl signals, one for unused slots in downstream pass req fifo ,the other for pass rep fifo.

The credit flow ctrl bits are crucial for avoiding deadlock, In the meantime, avoiding overflow of donwstream fifos . Now let's study a case !



## 2.5. Deadlock free

Ti means time flow. **Can\_cmpt** means deadlock free and can compete with others.

The triangles noted by number stand for busy flag . When 1 is true ,it means pass req is currently busy transferring his flits to next node.

	1	2	3	4	Competition description
T1	1	1	1	0	1 win for 7 already. Pass Rep2 has been transferring flits to pass rep 6. Ctrl of flit from pass rep2 now is 2'b11. Next cycle pass rep 6 is free to be competed among pass rep 2 and OUT rep 4(if deadlock free).
T2	1	0	1	0	Now 2 and 4 both find that 6 free . Head flit tell 4 go to pass rep 6 instead of IN rep 8. And 4 finds there are enough slots for it to store its flits. Then 4 win for the right to access 6, according to some priority algorithm. Then arbiter set busy flag 4
T3	1	0	1	1	.....transferring flit according round robin algorithm ( At this time, 1,3,4 transfer flits in turn).
.....	1	0	1	1	.....transferring flit according round robin algorithm
T4	1	0	1	1	4 finishes transferring to 6, that's ctrl==2'b11. Then arbiter rst the busy flag 4.
T5	1	0	1	0	4 win for 8 and 2 win for 6(no competition between 4 and 2). And Ctrl of rep 3's flit == 2'b11. Then arbiter sets busy flag 2 and 4, rst 3.
T6	1	1	0	1	3 can't win for 5, because of lacking of deadlock free condition.(the first msg in OUT req3 have bigger number of flits than the number of unused slots in req 5).
.....	1	1	0	1	.....transferring flit according round robin algorithm
T7	1	1	0	1	Now 3 win for 5 after long time 5 pop some flits to next node of it. So there is enough unused slots now (deadlock free).
T8	1	1	1	1	1, 2, 3, 4 transfer flits in turn (RR algorithm).

### Think carefully

Once we let 3 win for 5, then 3 will fill up 5 with its flits while there are some flits still in 3. Assuming at this time, deadlock occurs between nodes like the figure below



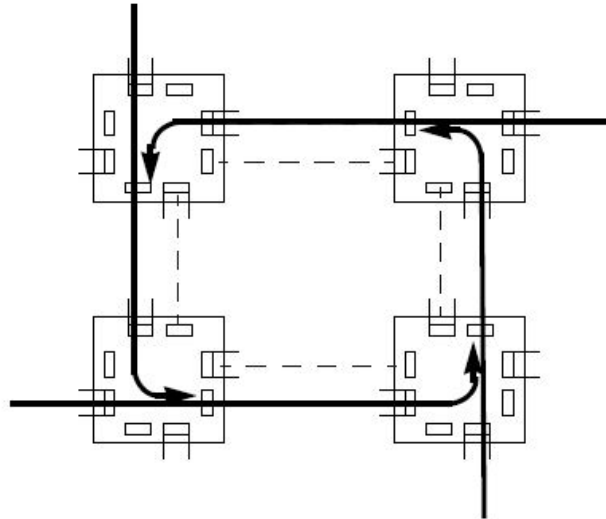


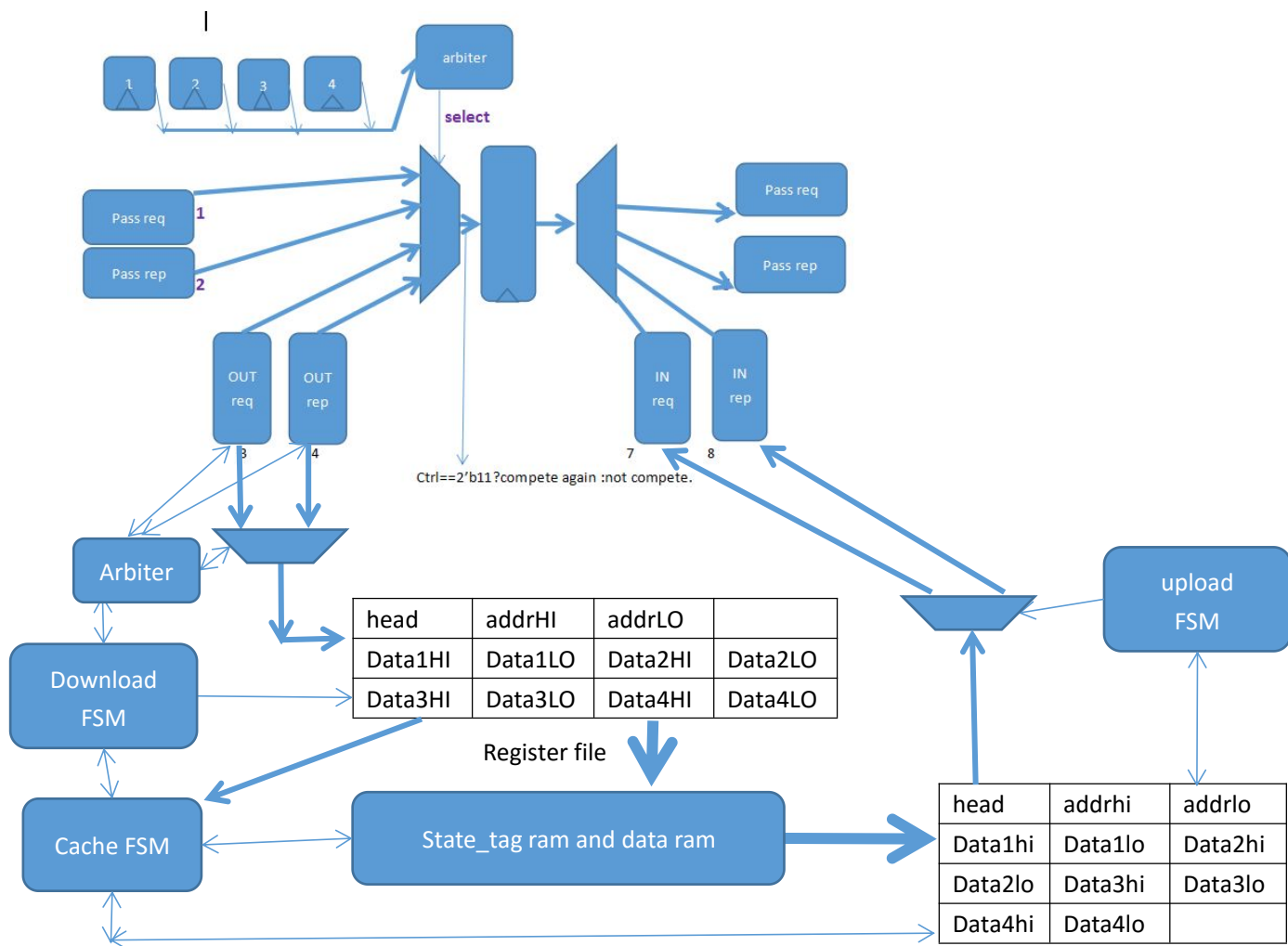
Figure 10-19 Examples of Network Routing Deadlock

**NOTE:** once there is possibility of deadlock between pass fifos ,at this time every pass fifo should have at least one unused slot for receiveing flit from previous node. we only allow pass fifos transfering between each other. So as long as flits flow to their dest node , the pass fifos will grandually become deadlock free for IN fifos to send out flits to pass fifos of next node.

**Reason** why T6 situation happens is that once we change transfer source from OUT fifos to pass fifos ,we may lose the correctness of the OUT fifo's message. Because msg will be divided into two part among nodes. The frame of the splited msg will be wrong for dest node's msg process logic, and the logic need to NACKs to src node to retry. In this project, i prefer to provide deadlock free to reduce NACKs to improve performance. While NACKs still might happen,when a node request a busy addr in the dest node, dest node NACK to src node to retry again after a short time depending on retry algorithm in the src node.

Since number of bits including head flit ,addr and data is more than 16 and cache or mem process whole mssage everytime, you may wonder how flits generate and transfer from node to ring network. There are some serial-parallel assist for donwloading flits from network to dest cache or mem, and parallel-serial assist for uploading flits from src cache or mem to network.

Figure below will tell you how it works!



The block diagram above still works for mem transaction!

Other than the use of reply forwarding, the most interesting aspects of the protocol are its use of **pending** states and **NACKs** to resolve race conditions and provide serialization to a location, and the way in which it handles race conditions caused by writebacks.

#### NOTE:

1, pending states: these imply that the home has received a previous request for that block, but was not able to complete that operation itself (e.g. the block may have been dirty in a cache in another node); transactions to complete the request are still in progress in the system, so the directory at the home is not yet ready to handle a new request for that block.

2, NACKs: when the CA receive a request to the block which is in progress due to previous request, the CA should drop the later requests and wait for them to retry.

### 3.benchmarks

In order to demonstrate performance improvement from single to multicore processor ,I will use several benchmark to feed the system.

**3.1** memory benchmark,which can be used to test the data transport performance.

**3.2** binary search algorithm (a classic divide and conquer algorithm) that searches for a value X in an sorted N-element array A and returns the index of matched entry,which is from Computer Organization and Design: The Hardware/Software Interface, Fifth Edition.

**3.3** sum of a array, which can be implemented on multicore easily.

**3.4** merge\_sort algorithm.

### 4. future work&extensive

Since the complexity of computer system is increasing rapidly, it force us to explore design space from many different points, we now need to build a powerful platform to support our explore. Here I want to make my project more useful ,which is scalable and parameterized.

Parallel computer architecture bring us additional exploration space ,such as network bandwidth,cache coherence protocol, message passing rules , Network router strategy and network interface fifo depth besides design space of single core design.

**network bandwidth (which can affects message format,)** : typically 16 bits, 32 bits, 48 bits etc.

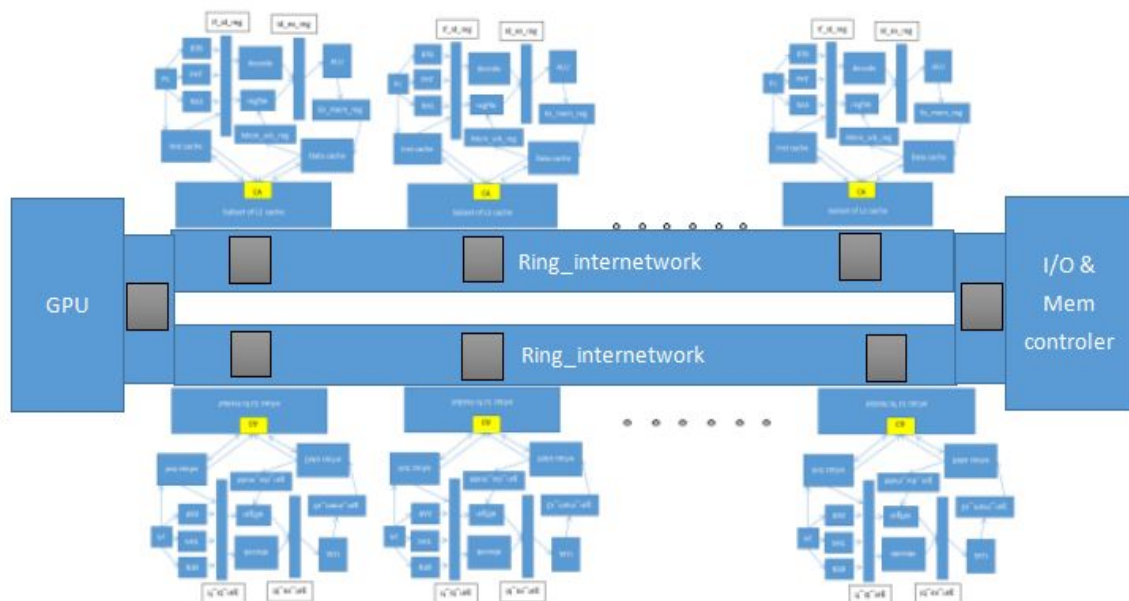
**cache coherence protocol** : snooping-based ; directory-base.


**message passing rules**:strict request-reply; intervention-forwarding; reply-forwarding.

**internetwork topology** : ring(I fucos on), mesh, linear array.

**Network router strategy** .

**Network interface fifo depth.**



**Note :**  : Network interface between node and network.

While the GPU can be replaced by other specific function unit. And you can see that you can put more cores on the network if you like easily!

Adding more cores will be supported by define some variable, for instance change core\_num from 4 to 8 , if you want to add another 4 cores to the original four-cores system.

You can change network\_bandwidth from 16 to 32 if you want to extend the bandwidth of inter\_network.

Or you can change some variable of fifo depth in the network to make a exploration for fifo depth.

If any time before I graduate from college or on my job ,I will try my best to make this project more useful for those who want to learn something about computer architecture!

Would you like to support me to do these things sometimes on the fpga platform?

.

**I think we should encourage those who want to be engaged in computer design to learn knowledge on the hardware platform instead of only doing some emulation on qemu, gem5 or simplescalar , which , i think , will be significant for our design experience, because Imagination has cooperate with Xilinx via MIPSfpga to teach students to learn real world processor design.**

**So,here is my suggestion that you can combine products and knowledge to attract your customers.**

## **5.References**

### **1.Parallel Computer Architecture A Hardware / Software Approach**

David Culler   University of California, Berkeley; Jaswinder Pal Singh; Princeton University with Anoop Gupta Stanford University.

### **2.Computer Organization and Design The Hardware / Software**

**Interface** David A. Patterson University of California,  
Berkeley John L. Hennessy Stanford University

### **3. 6-823-fall-2005 lecture notes from MIT.** Arvind and Krste Asanovic

### **4. CS 152 Computer Architecture and Engineering lecture notes from**

**UCB**   2014-3-6 John Lazzaro.

### **5. Parallel Computer Organization and Design**

Michel Dubois   University of Southern California (USC)  
Murali Annavaram   USC ;   Per Stenström   Chalmers University of Technology, Sweden

## Appendix:

My project code on github, [https://github.com/zhaishaomin/ring\\_network-based-multicore-](https://github.com/zhaishaomin/ring_network-based-multicore-)

A floorplan via quartus ii , which is powerful for designing complex digital system!

