Dedicated Message Passing Hardware

Motivation:

- Memory sharing is slow
- Assumes that you want read and write
- If you only need (Read | Write) performance can be better

Strategy

- Introduce new coprocessor registers which have the following API:

Register 23: Write: Set Target   Read: Target Busy

Register 24: Write: Send to Target Read: Read from target

Register 25: Write: deq from target Read: Recieve Empty

Where

CoreIndex target sets who we are going to enq/deq/read from.

Demos

Implementation

- We make a:

```
Vector#(NumCores,Vector#(NumCores, Fifo#(2, Data))) ipc_write
<- replicateM(replicateM(mkCFFifo));
Vector#(NumCores,Vector#(NumCores, Fifo#(2, Data))) ipc_read
= transpose(ipc_write);
```

And pass ipc_*[core_id] to each core

(* synthesize *) goes away ☹

But if synthesize didn't go away, where we deq/enq would always block so this isn't so bad!

Part || Bugs

Using the response Register data in the NBCache FSM

- Most of the data should come from the entry in stq/ldbuff

Not Bypassing to stq in NBCache FSM

Out of Order Stq resp

- Might receive out of order stq response

Debugging Tips

Don’t use ?

- We said in class use unpack(0), but I’d instead advise using funny-constants (0xdeadbeef, 0x1337l337, 0xBAADF00D, etc http://en.wikipedia.org/wiki/Hexspeak)
- This will let you see where data you don’t care about is getting injected more specifically

Print out Cycles using Cop

- Let’s you more easily look at the sim output and see where exactly problems start

Learn Bluespec

- We didn’t use a lot of the really cool fun features of BSV too heavily in class

ie, print rule from Stq

```
 rule debugInfo( isValid(core_id) );
     function Action zw (Integer x, Reg#(a) data) provisos (FShow#(a));
     return (action
            let i = fromInteger(x);
            if (enqP > deqP && i < enqP && i >= deqP)
               $fwrite(stdout, " ", fshow(data), " ");
           else if( enqP < deqP && ( i >= deqP || i < enqP ) )
               $fwrite(stdout, " ", fshow(data), " ");
           else if( full_reg )
               $fwrite(stdout, " ", fshow(data), " "); endaction);endfunction
      $fwrite(stdout, "StQ: Core ID: ", fshow(core_id) );
      joinActions(zipWith(zw, genVector, data));
      $fwrite(stdout, "\n");
endrule
```

Look for unconditional Behaviors

- If you are always doing something, be sure you always do actually want to do it
  - in the rule handling response, were unconditionally upgrading the data line, not just on I -> (msi > I)
  - We caught this by using the hexspeak codes suggested earlier

Use assertions:

```
if (! trueThing)     begin
        $display(“invariant x broken”);
        $finish;
end
```