Networks on chip

Luca Benini
DEIS-Universita' di Bologna
lbenini@deis.unibo.it

Outline

- Introduction and motivation
  - Physical limitations of on-chip interconnect
  - Communication-centric design
- On-chip networks and protocols
- Software aspects of on-chip networks

Application requirements

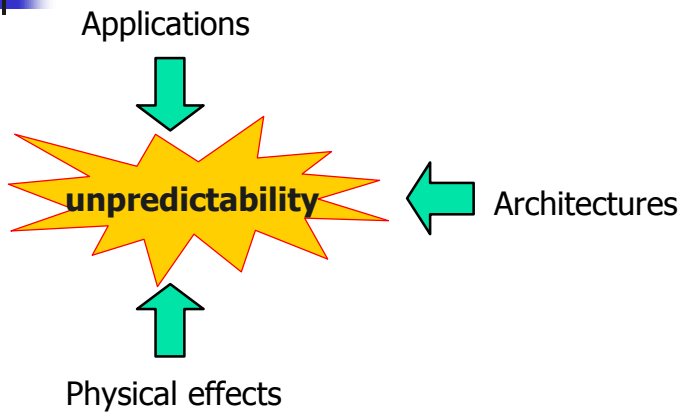
Signal processing	Media processing	Multi-media
hard real time very regular load	hard real time irregular load	soft real time irregular load
high quality	high quality	limited quality
worst case **typically on DSPs**	average case **SoC/media processors**	average case **PC/desktop**

- Very challenging!

Systems on chip

- Moore's law provides exponential growth of resources
- But design does not become easier
  - → Deep submicron problems (DSM)
    - Wire vs. transistor speed, power, signal integrity
  - → Design productivity gap
    - IP re-use, platforms, NoCs
    - Verification technologies

The challenge of unpredictability



- We still need to build predictable systems!

The differentiation challenge

- Wireless terminal platforms
  - Micro + DSP
- Digital video platforms
  - Micro + VLIW + Coprocessors
- Why?
  - IP components are similar (often the same!)
  - Developing new cores is expensive, and software…
- But: a big "?" on how communication is performed
  - Opportunity for differentiation!

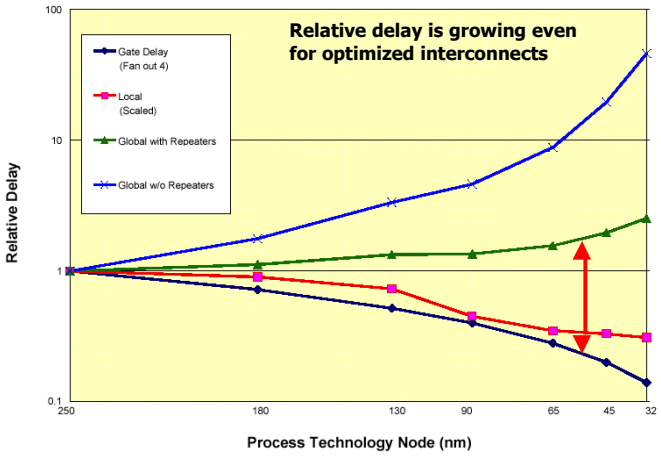
Outline

- Introduction and motivation
  - Physical limitations of on-chip interconnect
  - Communication-centric design
- On-chip networks and protocols
- Software aspects of on-chip networks

Qualitative roadmap trends

- Continued gate downscaling
- Increased transistor density and frequency
  - Power and thermal management
- Lower supply voltage
  - Reduced noise immunity
- Increased spread of physical parameters
  - Inaccurate modeling of physical behavior

Interconnect delay trend



Wire delay vs. logic delay

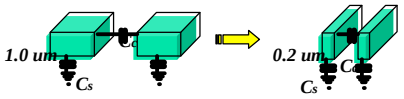
Operation	Delay (0.13um)	(0.05um)
32b ALU Operation	650ps	250ps
32b Register Read	325ps	125ps
Read 32b from 8KB RAM	780ps	300ps
Transfer 32b across chip (10mm)	1400ps	2300ps
Transfer 32b across chip (20mm)	2800ps	4600ps

2:1 global on-chip comm to operation delay
9:1 in 2010

Taken from W.J. Dally presentation: Computer architecture is all about interconnect (it is now and it will be more so in 2010) HPCA Panel February 4, 2002

Communication Reliability

- Information transfer is inherently unreliable at the electrical level, due to:
  - Timing errors
  - Cross-talk
  - Electro-magnetic interference (EMI)
  - Soft errors



- The problem will get increasingly more acute as technology scales down

Noise and transient errors

- SoC will operate in presence of noise
  - Data may be delayed or corrupted
  - Malfunctions modeled as single/multiple upsets
- Present design methods reduce noise
  - Physical design (e.g., sizing, routing)
- Future methods must tolerate noise
  - Push solution to higher abstraction levels

Outline

- Introduction and motivation
  - Physical limitations of on-chip interconnect
  - Communication-centric design
- On-chip networks and protocols
- Software aspects of on-chip networks

Network design objectives

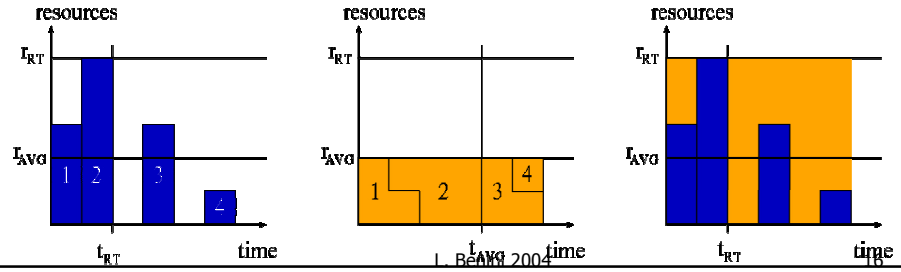
- Low communication latency
  - Streamlined control protocols
- High communication bandwidth
  - To support demanding SW applications
- Low energy consumption
  - Wiring switched capacitance dominates
- High system-level reliability
  - Correct communication errors, data loss

Theme 1: QoS or Best effort?

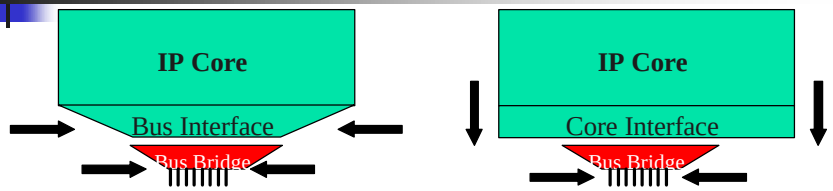
- Essential to recover global predictability
  - NOTE: pplications require it
  - It fits well with protocol stack concept
- What is QoS?
  - Requester poses the service request (negotiation)
  - Provider either commit to or reject your request
  - renegotiate when requirements change
- After negotiation. steady states that are predictable

The cost of QoS

- Best-effort services have better average resource utilisation at the cost of unpredictable/unbounded worst-case behaviour
- The combination of best-effort & guaranteed services (c) is useful!



Theme 2: Modularity vs. efficiency



- Bus-Centric Protocol Interface:
  - Forces core interface to the specs of a particular bus
- Core-Centric Protocol Interface:
  - Provides an abstract communication channel for the core
  - Enables unconstrained interface bridge to any interconnect structure

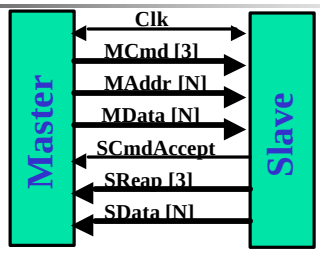
[SonicsInc]

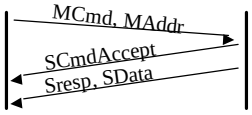
Basic OCP Concepts

- Point-to-point, unidirectional, synchronous
  - Easy physical implementation
- Master/slave, request/response
  - Well-defined, simple roles
- Extensions
  - Added functionality to support cores with more complex interface requirements
- Configurability
  - Match a core's requirements exactly
  - Tailor design to required features only

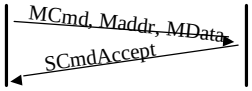
Reference: [SonicsInc]

Basic OCP protocol





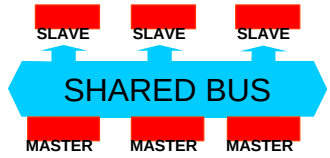
Read:
Command, Address
Command Accept
Response, Data



Write:
Command, Address, Data
Command Accept

Theme 3: Communication Architectures

- SHARED BUS (broadcast);
  - Low area
  - Poor scalability
  - .High energy consumption



- NETWORK on CHIP (point-to-point)
  - Scalability;
  - Low energy consumption;
  - High area.

Do we need a network?
Can we do with dedicated wiring?

Dedicated wires vs. Network

Dedicated Wiring	On-Chip Network
Spaghetti wiring	Ordered wiring
Variation makes it hard to model crosstalk, returns, length, R & C.	No variation, so easy to exactly model XT, returns, R and C.
Drivers sized for 'wire model' – 99% too large, 1% too small	Driver sized exactly for wire
Hard to use advanced signaling	Easy to use advanced signaling
Low duty factor	High duty factor
No protocol overhead	Small protocol overhead

Networks on chip: advantages

- differentiated services
  - offer different kinds of communication with one network
- scalable
  - add routers and network interfaces for extra bandwidth (at the cost of additional latency)
- compositional
  - add routers/NIs without changing existing components e.g. timing, buffers
- efficient use of wires
  - statistical multiplexing/sharing (average vs. worst-case) ⇒ fewer wires ⇒ less wire congestion
  - point to point wires at high speed
- communication becomes re-usable, configurable IP

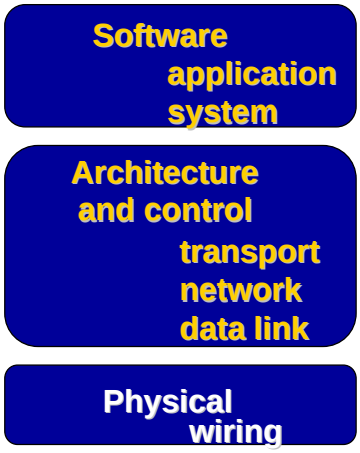
Outline

- Introduction and motivation
  - Physical limitations of on-chip interconnect
  - Communication-centric design
- On-chip networks and protocols
  - Designing NoCs
  - NoCs case studies
- Software aspects of on-chip networks

Outline

- Introduction and motivation
  - Physical limitations of on-chip interconnect
  - Communication-centric design
- On-chip networks and protocols
  - Designing NoCs
  - NoCs case studies
- Software aspects of on-chip networks

Physical layer

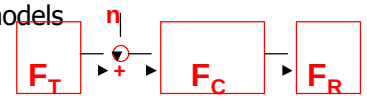


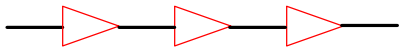
Physical design:

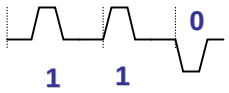
- Voltage levels
- Driver design
- Sizing
- Physical routing

Physical layer: the channel

- Channel characteristics
  - Global wires: lumped → distributed models
    - Time of flight is non-negligible
  - Inductance effects
    - Refelections, matching issues
- Designing around the channel's transfer function
  - Current mode vs. voltage mode
  - Low swing vs. rail-to-rail
  - Repeater insertion
  - Wire sizing
  - Pre-emphasis / post-filtering
  - Modulation

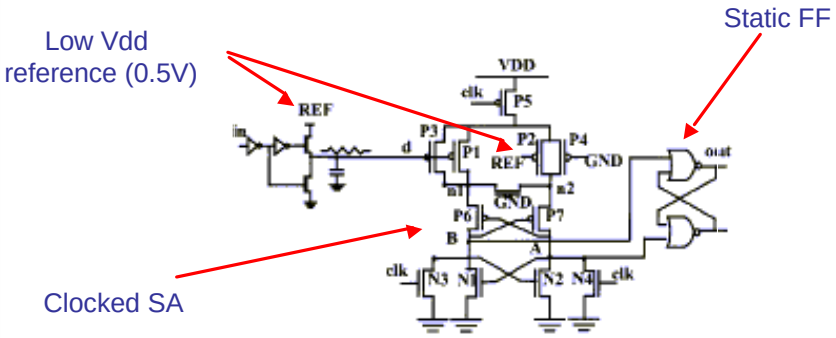






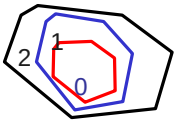
Low Swing signalling circuit

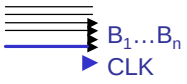
- Pseudodifferential interconnect [Zhang et al., JSSC00] (x6 energy reduction vs. CMOS Vdd=2V)

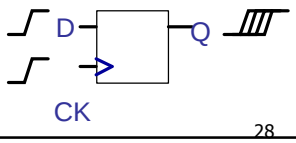


Physical layer: synchronization

- Single, global timing reference is not realistic
  - Direct consequence of non-negligible tof
  - Isochronous regions are shrinking
- How and when to sample the channel?
  - Avoid a clock: asyncronous communication
  - The clock travels with the data
  - The clock can be reconstructed from the data
- Synchronization recovery has a cost
  - Cannot be abstracted away
  - Can cause errors (e.g., metastability)

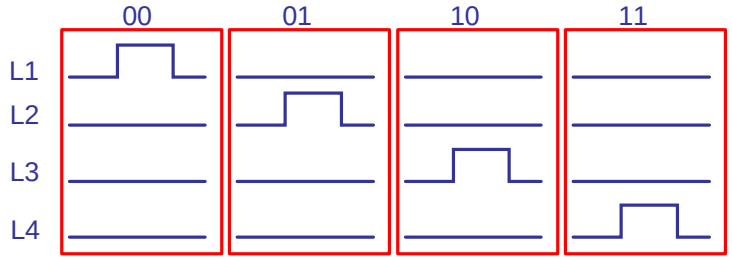




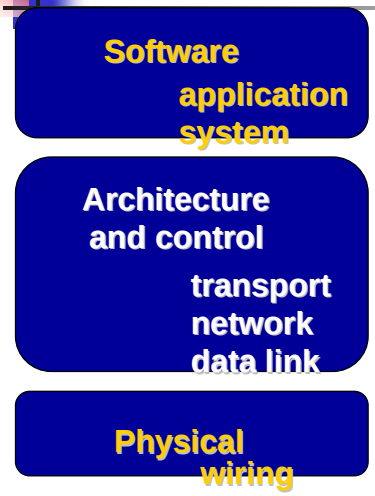


Case study: Asyncronous Bus

- MARBLE SoC Bus
  - 1-of-4 encoding (4 wires for 2 bits)
  - Delay insensitive - No bus clock - Wire pipelining
  - High crosstalk immunity
  - Four-phase ACK protocol



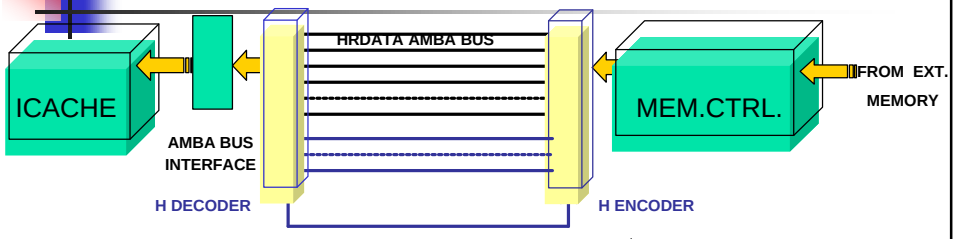
Architecture and control



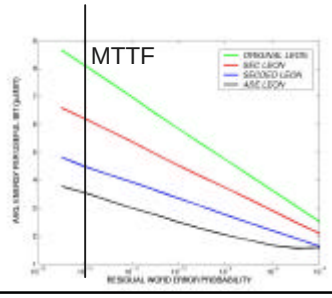
Data link layer

- Provide reliable data transfer on an unreliable physical channel
- Access to the communication medium
  - Dealing with contention and arbitration
- Issues
  - Fairness and safe communication
  - Achieve high throughput
  - Error resiliency

Data-link protocol example: error-resilient coding



- Compare original AMBA bus to extended bus with error detection and correction or retransmission
  - SEC coding
  - SEC-DED coding
  - ED coding
- Explore energy efficiency



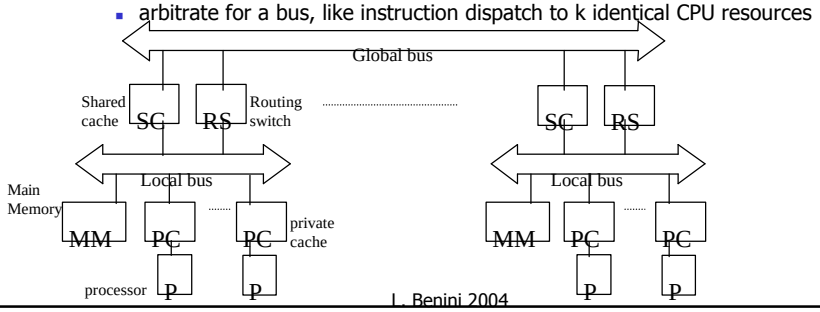
Network layer: topology

- Buses:
  - Pro: simple, existing standards
  - Contra: performance, energy-efficiency, arbitration
- Other network topologies:
  - Pro: higher performance, experience with MP
  - Contra: physical routing, need for network and transport layers

Challenge: exploit appropriate network architecture and corresponding protocols

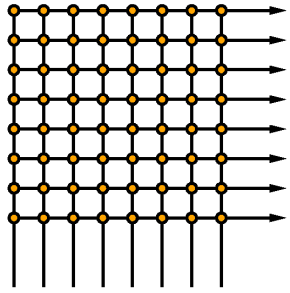
Multiple Busses

- Simple way to increase bandwidth
  - use more than one bus
- Can be static or dynamic assignment to busses
  - static
    - A->B always uses bus 0
    - C-> always uses bus 1
  - dynamic
    - arbitrate for a bus, like instruction dispatch to k identical CPU resources



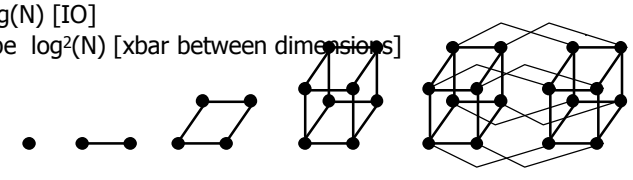
Crossbar

- No bandwidth reduction
  - (except receiver at endoint)
- Easy routing (on or offline)
- Scales poorly
  - N^2 area and delay
- No locality



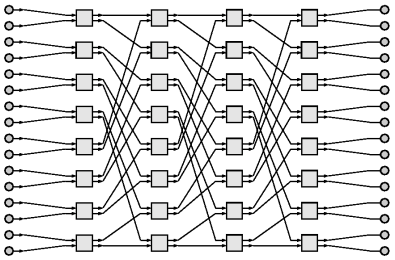
Hypercube (direct)

- Arrange 2^n nodes in n-dimensional cube
- At most n hops from source to sink
  - log(number of nodes)
- High bisection bandwidth
  - good for traffic (but can you use it?)
  - bad for cost [$O(n^2)$]
- Exploit locality
- Node size grows
  - as log(N) [IO]
  - Maybe $\log^2(N)$ [xbar between dimensions]



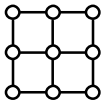
Multistage (indirect)

- Unroll hypercube vertices so log(N), constant size switches per hypercube node
  - solve node growth problem
  - lose locality
  - can be blocking, but it can be made non-blocking with more stages



K-ary N-cube

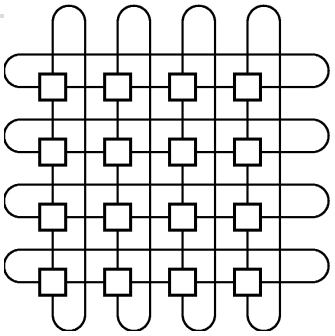
- Alternate reduction from hypercube
  - restrict to N<log(N) dimensional structure
  - allow more than 2 ordinates in each dimension
- E.g. mesh (2-cube), 3D-mesh (3-cube)
- Matches with physical world structure
- Bounds degree at node
- Has Locality
- Even more bottleneck potentials

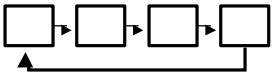


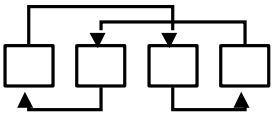
2D Mesh

Torus

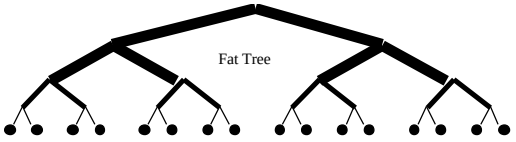
- Wrap around n-cube ends
  - 2-cube → cylinder
  - 3-cube → donut
- Cuts worst-case distances in half
- Can be laid-out reasonable efficiently
  - maybe 2x cost in channel width?

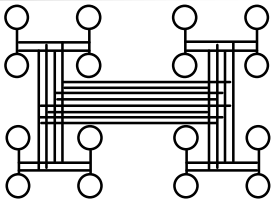






Fat-Trees





- Fatter links (really more of them) as you go up, so bisection BW scales with N

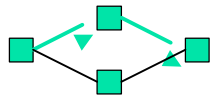
Network layer: control

- Switching
  - Connection-oriented switching
    - A path from source to destination is reserved prior to communication
    - Useful for traffic with infrequent and long messages
    - Circuit switching, virtual circuits
  - Connection-less switching
    - The communication path is determined dynamically
    - Datagram
- Routing
  - Unicast, multicast
  - Source routing, distributed routing
  - Deterministic, adaptive

Challenge: which models and what parameter values are effective for on-chip communication?

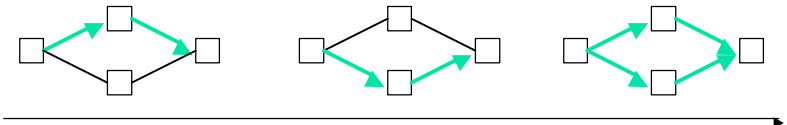
Connection-oriented schemes

- Communication path is fixed before data transmission starts
- Network types: **circuit switched, virtual circuit**
  - Basic operations
    - Connection setup
    - Data transfer
    - Connection teardown
  - Advantages
    - QoS (e.g. bandwidth, delay, jitter) guarantee through resource reservation over the fixed path
    - Suited to real-time, constant BW communication
  - Disadvantages
    - Resource utilization is worse than connection-less communication (i.e. datagram).
    - Connection setup overhead



Connection-less schemes

- Communication path is determined dynamically during data (packet) transmission



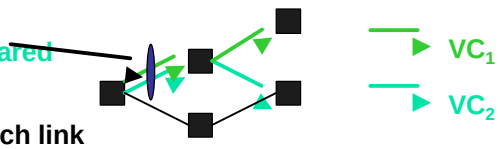
- Network type: datagram
- Advantages
  - Better adaptation to the varying network traffic
  - Better utilization of network resource
  - Suited to variable bit rate communication
    - (e.g. encoded voice, MPEG2,4, etc.)
- Disadvantage
  - Poor QoS support (← no resource reservation)

Packet-based communication: Virtual circuit

- Fixed end-to-end communication path
  - Packets are transferred over the VC

Each link can be shared by several VC's

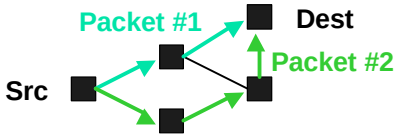
*** Multiplexing on each link**



- QoS guarantee:
  - Through resource reservation when the connection is set up

Packet-based communication: Datagram

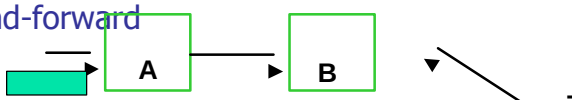
- Connection-less
- Routers route packets independently.
  - Packets can take any paths to the destination.



  - Routers manage routing tables to find paths to destinations
- Non-deterministic communication delay due to communication resource (buffer and link) contention
- No QoS guarantee!
- Flow and congestion control is needed

Packet Forwarding Schemes

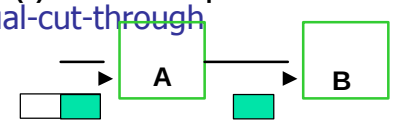
- Store-and-forward



(1) After A finishes receiving the entire packet, A asks B if B is ready for the entire packet.
(2) B → A, ack
(3) A sends the packet to B.

The same buffer space is needed

- Virtual-cut-through

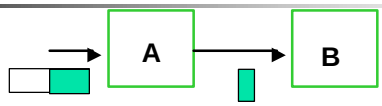


Pipelining!

(1) While A receives a part of the packet, A asks B if B is ready for the entire packet.
(2) B → A, ack
(3) A starts to send the packet to B even when A has not yet received the entire packet.

Packet Forwarding Schemes II

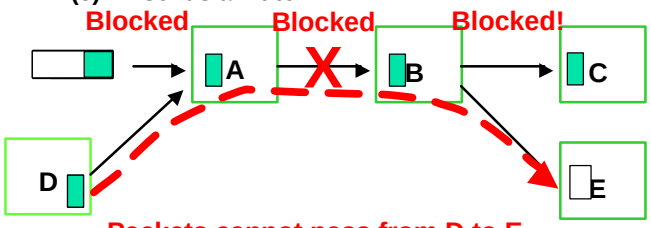
- Wormhole



Pipelining on a flit (flow control unit) basis

(1) After A receives a flit of the packet, A asks B if B is ready to receive a flit
(2) B → A, ack
(3) A sends a flit to B.

flit size < packet size
Smaller data space is needed than store-and-forward



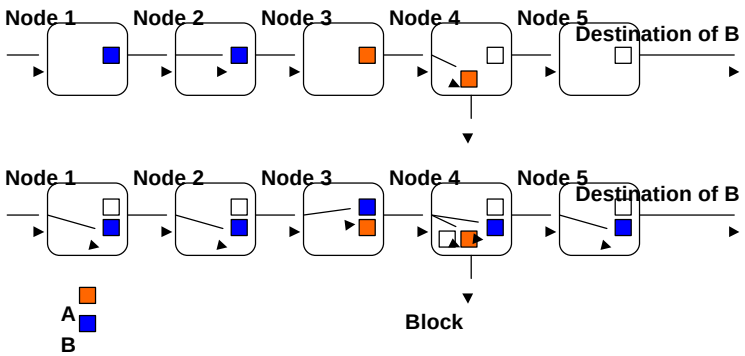
C cannot send it and has no enough space for a new flit

Head-of-line blocking problem

Packets cannot pass from D to E due to the blocked link between A and B

Virtual Channels

- Performance improvement using virtual channels



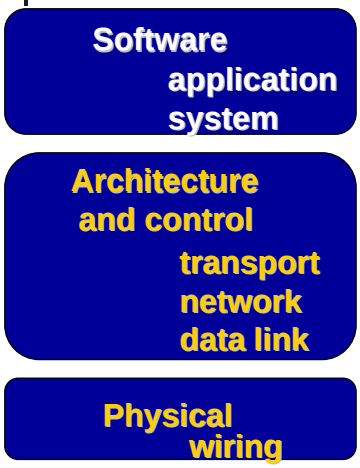
Transport layer

- Decompose and reconstruct information
- Important choices
  - Packet granularity
  - Admission/congestion control
  - Packet retransmission parameters
- All these factors affect heavily energy and performance
- Application-specific schemes vs. standards

Benefits of packets

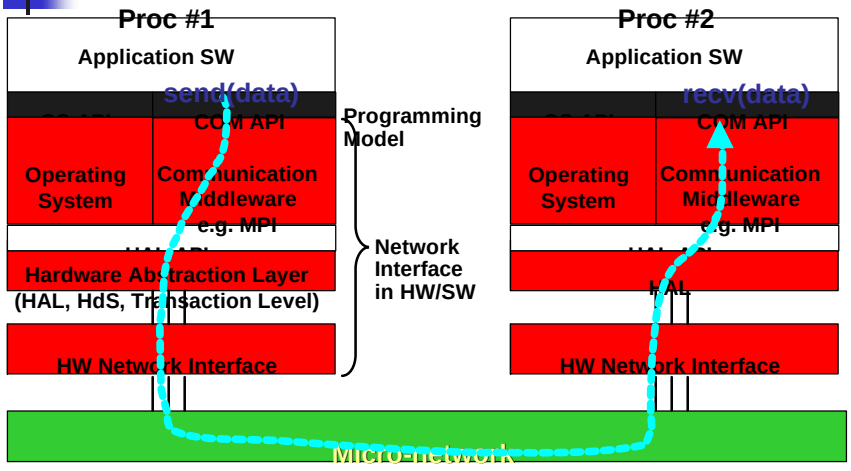
- Reliable error-control mechanism
  - With small overhead
- Exploit different routing paths
  - Spread information to avoid congestion
- Several user-controllable parameters
  - Size, retransmission schemes, …
- Use retransmission rate for calibrating parameters

Software layers



- System software
  - OS, RTOS, run-time scheduler
  - Component and network dependent
- Application software
  - User and standard applications

System view of communication



System software

- Programming paradigms
  - Shared memory
  - Message passing
- Middleware:
  - Layered system software
    - Should provide low communication latency
    - Modular, scaleable, robust ….

Application software development tools

- Software synthesis
  - Source-code generation
  - Source-level optimizing transformations
- Application-specific compilation
  - Choice of instructions, registers, schedule
- Software design tools need network awareness to be effective
  - Balance computation, storage and communication

Outline

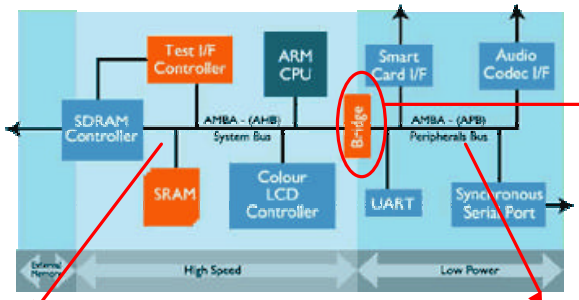
- Introduction and motivation
  - Physical limitations of on-chip interconnect
  - Communication-centric design
- On-chip networks and protocols
  - Designing NoCs
  - NoCs case studies
- Software aspects of on-chip networks

Standard "Bus" Architecture

- Large semiconductor firms
  - CoreConnect (IBM)
  - STBUS (STMicroelectronics)
- Core vendors
  - AMBA (ARM Ltd.)
- Interconnect IP vendors
  - CoreFrame (Palmchip)
  - WishBone (Silicore)
  - SiliconBackPlane (Sonics)
- Many others!

AMBA bus

Amba is a bridged bus



System-Peripheral Bridge

AHB: *high-speed high-bandwidth multi-master bus*

APB: *Simplified processor for general purpose peripherals*

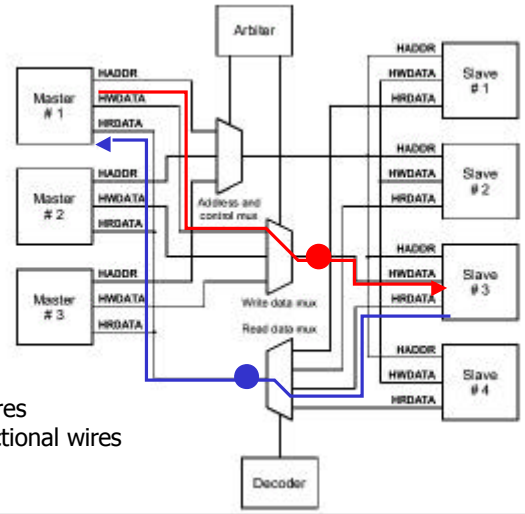
Bus Components: terminology

- Initiator
  - The FU that initiates transactions
- Target
  - Responds to incoming transaction

Bus actors

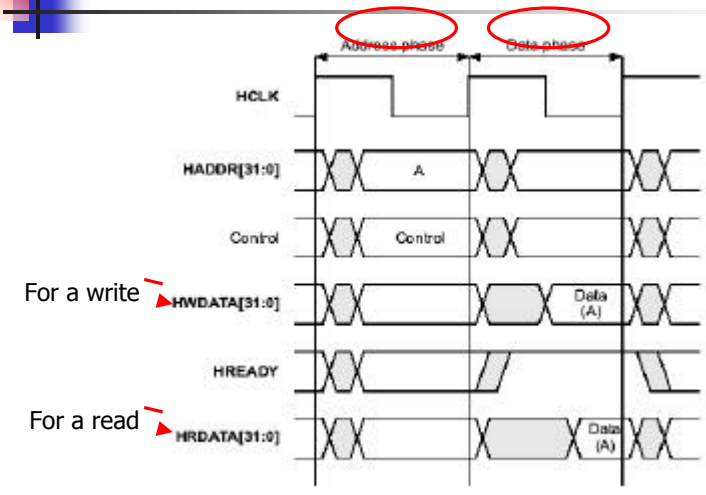
- Master/Slave
  - The initiator/target side of the bus interface
- Arbiter
  - Controls the access to the bus
- Bridge
  - Connects two buses
  - It acts as an initiator on one side and a target on the other

AHB Bus architecture



Different wires
⇒No bidirectional wires

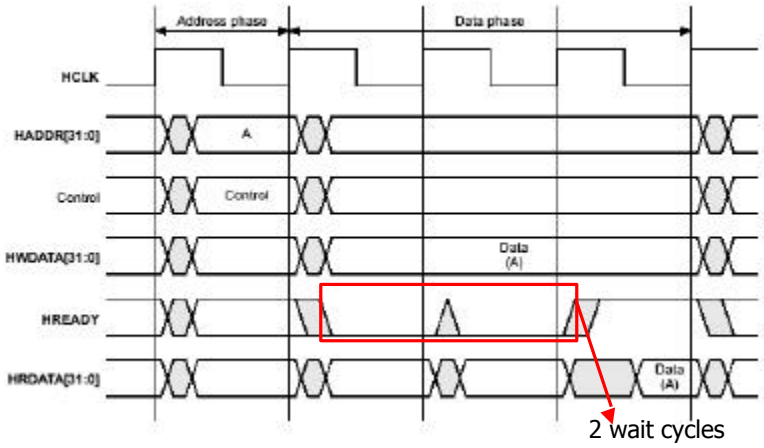
AMBA basic transfer



For a write

For a read

Transfer with WAIT states

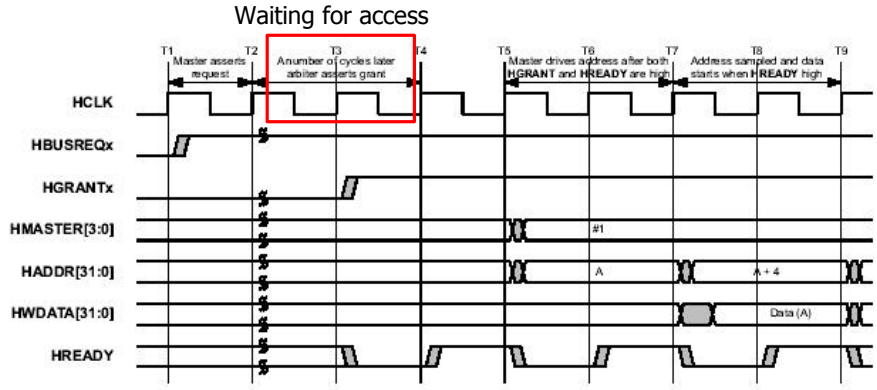


Bus Arbitration

- Buses can support multiple initiators
- Need a protocol for allocating bus resources and resolving conflicts
  - Bus arbitration
- Need a decision procedure to choose
  - Arbitration policy

NOTE: Do not confuse Arbitration Protocol with arbitration policy

Granting bus access



Burst transfers

- Arbitration has a significant cost
- Burst transfers amortize arbitration cost
  - Grant bus control for a number of cycles
  - Help with DMA and block transfers
- Requires safeguards against starvation

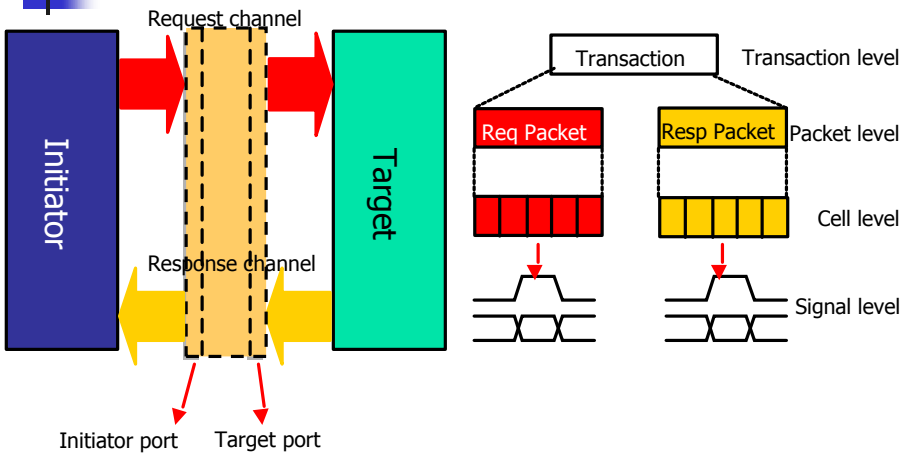
Critical analysis: bottlenecks

- Protocol
  - Lacks parallelism
    - In order completion
    - No multiple outstanding transactions: cannot hide slave wait states
  - High arbitration overhead (on single-transfers)
  - Bus-centric vs. transaction-centric
    - Initiators and targets are exposed to bus architecture (e.g. arbiter)
- Topology
  - Scalability limitation of shared bus solution!

STBUS

- On-chip interconnect solution by ST
  - Level 1-3: increasing complexity (and performance)
- Features
  - Higher parallelism: 2 channels (M-S and S-M)
  - Multiple outstanding transactions with out-of order completion
  - Supports deep pipelining
  - Supports Packets (request and response) for multiple data transfers
  - Support for protection, caches, locking
- Deployed in a number of large-scale SoCs in STM

STBUS Protocol (Type 3)



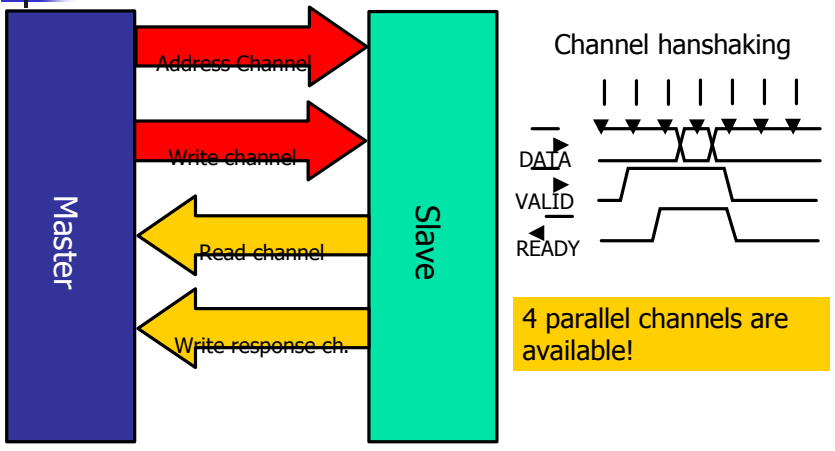
STBUS bottlenecks

- Protocol is not fully transaction-centric
  - Cannot connect initiator to target (e.g. initiator does not have control flow on the response channel)
- Packets are atomic on the interconnect
  - Cannot initiate nor receive multiple packets at the same time
  - Large data transfers may starve other initiators

AMBA AXI

- Latest (2003) evolution of AMBA
  - Advanced eXtensible Interface
- Features
  - Fully transaction centric: can connect M to S with nothing in between
  - Higher parallelism: multiple channels
  - Supports bus-based power management
  - Support for protection, caches, locking
- Deployment: ??

Multi-channel M-S interface

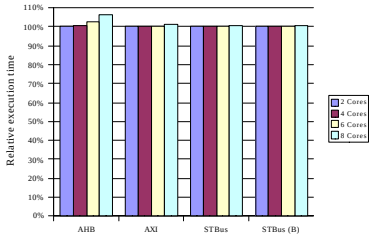


Multiple outstanding transactions

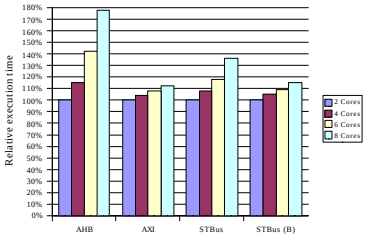
- A transaction implies activity on multiple channels
  - E.g Read uses the Address and Read channel
- Channels are fully decoupled in time
  - Each transaction is labeled when it is started (Address channel)
  - Labels, not signals, are used to track transaction opening and closing
  - Out of order completion is supported (tracking logic in master), but master can request in order delivery
- Burst support
  - Single-address burst transactions (multiple data channel slots)
  - Bursts are not atomic!
- Atomicity is tricky
  - Exclusive access better than locked access

Scalability: Execution Time

- Highly parallel benchmark (no slave bottlenecks)

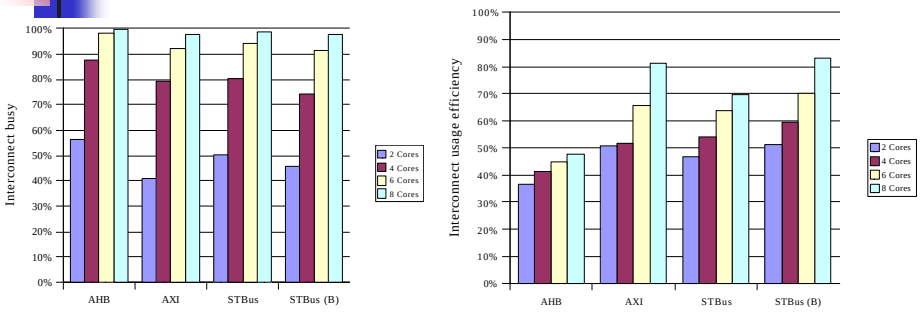


- 1 kB cache (low bus traffic)



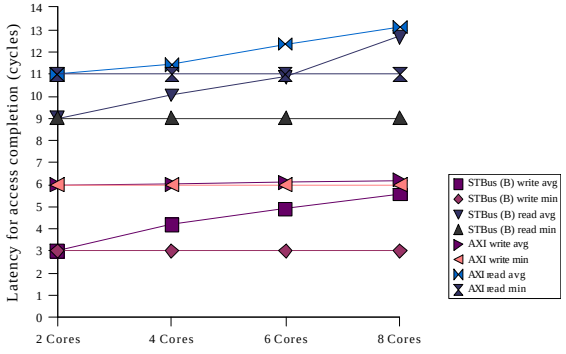
- 256 B cache (high bus traffic)

Scalability: Protocol Efficiency



- Increasing contention: AXI, STBus show 80%+ efficiency, AHB < 50%

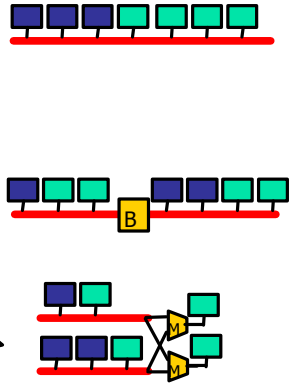
Scalability: latency



- STBus management has less arbitration latency overhead, especially noticeable in low-contention conditions

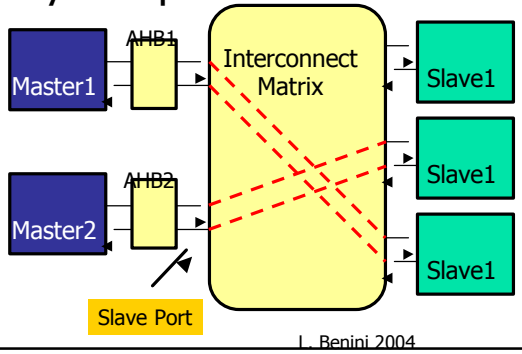
Topology

- Single shared bus is clearly non-scalable
- Evolutionary path
  - "Patch" bus topology
- Two approaches
  - Clustering & Bridging
  - Multi-layer/Multibus



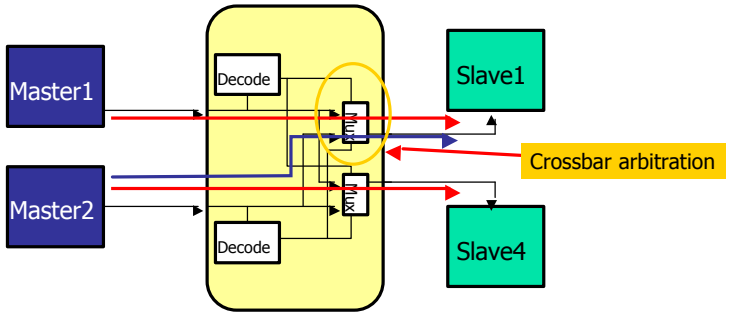
AMBA Multi-layer AHB

- Enables parallel access paths between multiple masters and slaves
- Fully compatible with AHB wrappers



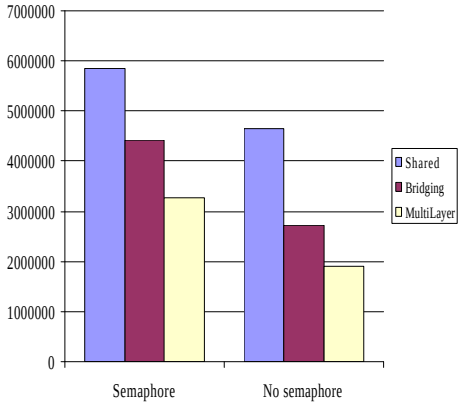
Multi-Layer AHB implementation

- The matrix is made of slave ports
  - No explicit arbitration of slaves
  - Variable latency in case of destination conflicts



Topology speedup (AMBA AHB)

- Independent tasks (matrix multiply)
- With & without semaphore synchronization
- 8 processors (small cache)

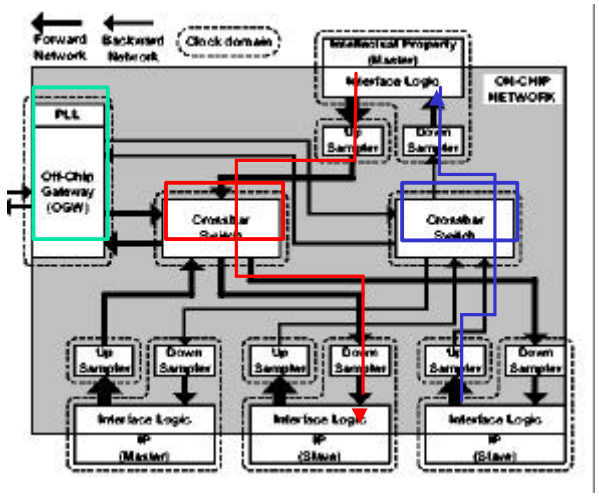


Lee's Star network [ISSCC'03]

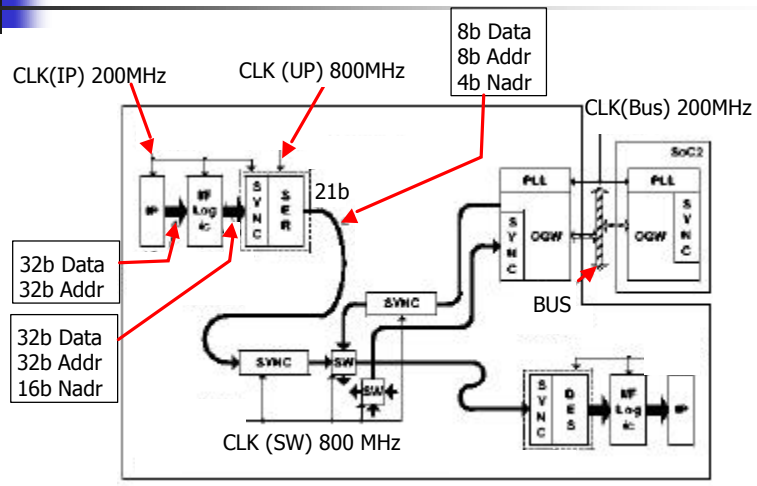
- Two n×m crossbar-based networks
  - Forward: master to slave (m masters)
  - Backward: slave to master (n slaves)
- Reduce area by serializing packets
  - Up-sampling @ transmitter
  - Down-sampling @ receiver
- Plesiochronous clocking of NoC regions
  - Synchronizers compensate for phase differences @ clock region boundaries

Network architecture

Backward
Forward
Interface to off-chip synchronous bus

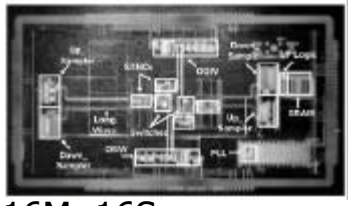


Packet flow

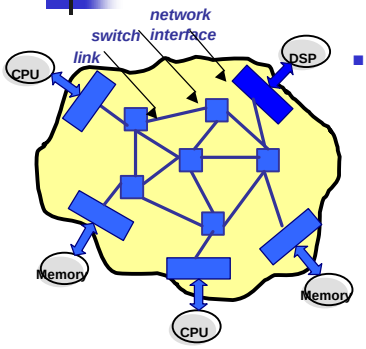


Implementation

- Area
  - 64.8mm^2 (.16µm DRAM tech)
- Power
  - 264mW at 2.3V
  - savings of 40% w.r.t. bus for 16M, 16S
- Serialization reduces area by 57% (crossbar becomes 1/16!)
- Performance
  - Max end-to-end latency 24.5ns (20 CLKs)
  - Each crossbar switch port sustains 1.6GB/s



NoCs



- More radical solutions in the long term
  - Nostrum
  - HiNoC
  - Linkoeping SoCBUS
  - SPIN
  - Star-connected on-chip network
  - Aethereal
  - Proteo
  - Xpipes
  - … (at least 15 groups)

NOCs vs. Busses

STBUS and AXI

- Packet-based
  - No distinction address/data, only packets (but of many types)
  - Complete separation between end-to-end transactions and data delivery protocols
- Distributed vs. centralized
  - No global control bottleneck
  - Better link with placement and routing
- Bandwidth scalability, of course!

The "power of NoCs"

Design methodology

Clean separation at the **session layer**:

1. Define end-to-end transactions
2. Define quality of service requirements
3. Design transport, network, link, physical

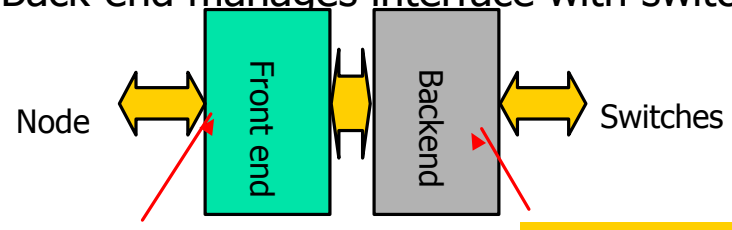
Modularity at the HW level: only 2 building blocks

1. Network interface
2. Switch (router)

Scalability is supported from the ground up (not as an afterthought)

Building blocks: NI

- Session-layer interface with nodes
- Back-end manages interface with switches



Standardized node interface @ session layer.
Initiator vs. target distinction is blurred

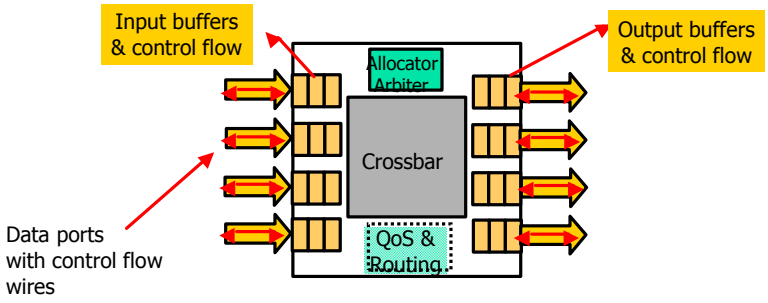
1. Supported transactions (e.g. QoSread…)
2. Degree of parallelism
3. Session prot. control flow & negotiation

NoC specific backend (layers 1-4)

1. Physical channel interface
2. Link-level protocol
3. Network-layer (packetization)
4. Transport layer (routing)

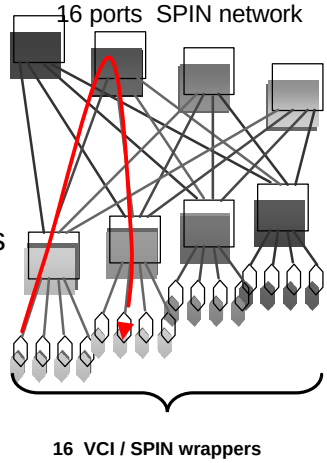
Building blocks: Switch

- Router: receives and forwards packets
  - NOTE: Packet-based does not mean datagram!
- Level 3 or Level 4 routing
  - No consensus, but generally L4 support is limited (e.g. simple routing)



The **SPIN** Network

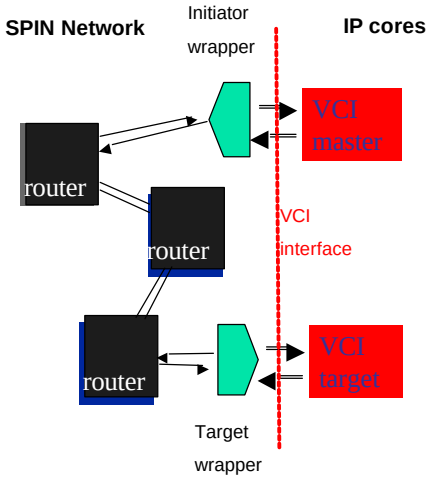
- Packet switching network
- Wormhole routing
- Multi-level Fat-Tree topology
- Point-to-point bidirectional links
- Credit-based flow control
- Adaptative routing



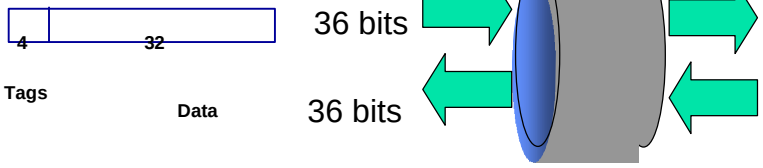
VCI / SPIN Wrappers

VCI master wrapper
- 16 concurrent requests
- 1- 3 cycles latency
- unconstrained packet length
- fully asynchronous mode

VCI target wrapper
- only one request
- 1- 3 cycles latency
- fully asynchronous mode



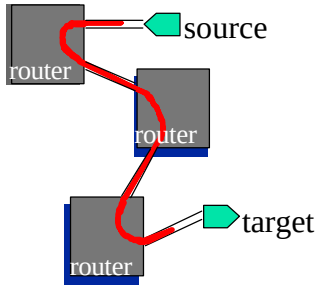
The bidirectional link



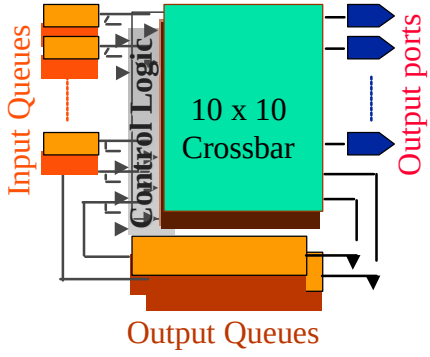
- 2 * 32-bits data links @ 200 MHz
  - ⇨peak link Bandwidth = 12.8 Gbit/s
- Asynchronous, credit based flow control
  - ⇨easy floorplan routing & timing in DSM process

Multistage Packet Switched Network

- Packet Switching
  - No circuit reservation
  - Atomic transaction = packet
- Wormhole routing
  - routers forward packets ASAP
  - packets span several routers
  - contention is a problem

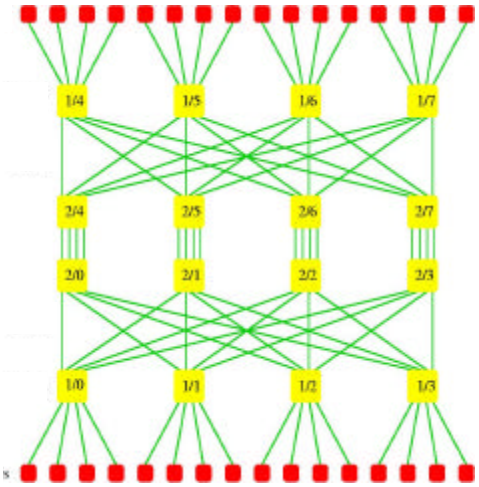


The RSPIN router



- Pipelined routing logic
- 2.5 cycles per routing hop
- Randomized, adaptive routing
- Elementary output queuing,
  - tuned for 64-byte payloads

Example : 32 ports SPIN

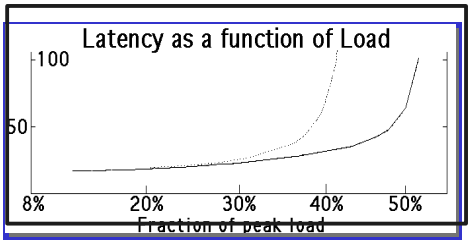


Performance evaluation

- Bit-true, cycle-true simulation for multi-million cycles for a 32 ports «raw» SPIN network:
- Worst-case workload (random, non-local)

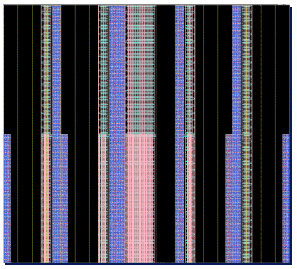
50% of peak bandwidth

3+3 Gbit/s per port



The SPIN hard macrocells

- 10 x 10 partial crossbar of 36-bit busses
- 2 kbits of memory
- Peak bandwidth = 50 Gbit/s
- Latency = 2.5 cycles
- Router area = 0.25 mm^2 (for STMicro 0.13µ)



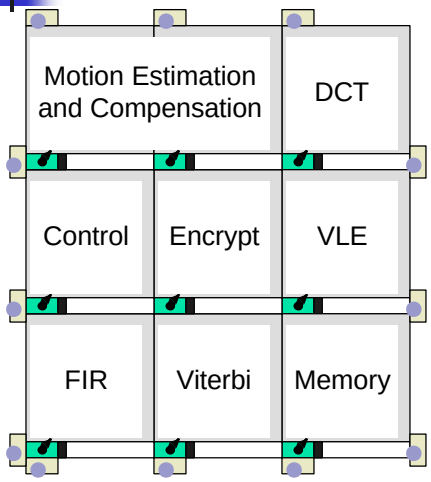
Router floorplan

- 32 ports SPIN network : 16 routers
- 1 * 4 mm^2 (for STMicro 0.13µ)
- All routing wires on top of the 16 routers
- Symbolic layout technology ⇒ fully portable

The **aSoC** network

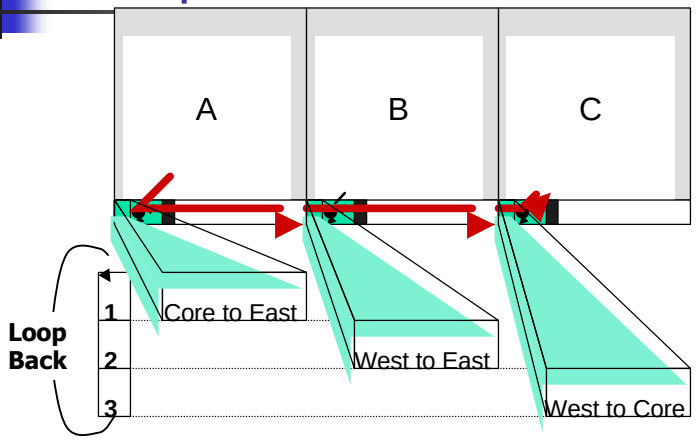
- High throughput
  - Fast and predictable interconnect
  - Very simple and fast routing
- Guaranteed QoS: deterministic scheduling, no contention
- Flexible
  - Runtime reconfiguration of cores and interconnect
- Power consumption
  - Implement power saving features in both cores and interconnect
  - Use reconfiguration to dynamically control power consumption

aSoC: adaptive System on a Chip



- Tiled SoC architecture
- More than one tile can be allocated to a single core
- Direct network
  - Near neighbor communication
- Routing logic in corners

Example: Stream



Critical analysis

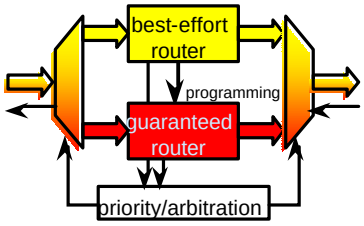
- Tile-based architecture not well-suited to heterogeneous platforms
- Static scheduling implies very rigid communication patterns
  - Underutilization of resources
  - Sporadic traffic is hard
- Not really for general purpose computation & communication

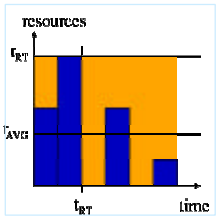
Æthereal: context

- Consumer electronics
  - reliability & predictability are essential
  - low cost is crucial
  - time to market must be reduced
- NoC offer differentiated services
  - to manage (and hence reduce) resources
  - to ease integration (and hence decrease TTM)

Æthereal: features

- Conceptually, two disjoint networks
  - a network with throughput+latency guarantees (GT)
  - a network without those guarantees (best-effort, BE)
- Several types of commitment in the network
  - combine guaranteed worst-case behaviour with good average resource usage





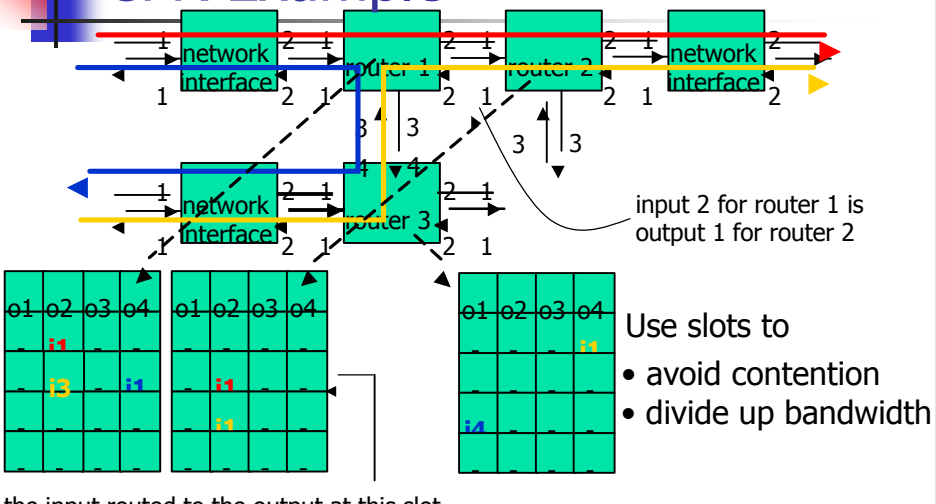
Router architecture

- Best-effort router
  - Worm-hole routing
  - Input queueing
  - Source routing
- Guaranteed throughput router
  - Contention-free routing
    - synchronous, using slot tables
    - time-division multiplexed circuits
  - Store-and-forward routing
  - Headerless packets
    - information is present in slot table

Contention-free routing

- Latency guarantees are easy in circuit switching
- Emulate circuits with packet switching
- Schedule packet injection in network such that they never contend for same link at same time
  - in space: disjoint paths
  - in time: time-division multiplexing
  - or a combination

CFR Example

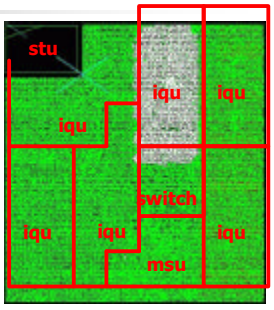


CFR setup

- Use best-effort packets to set up connections
  - set-up & tear-down packets like in ATM (asynchronous transfer mode)
- Distributed, concurrent, pipelined
- Safe: always consistent
- Compute slot assignment compile time, run time, or combination
- Connection opening is guaranteed to complete (but without a latency guarantee) with commitment or rejection

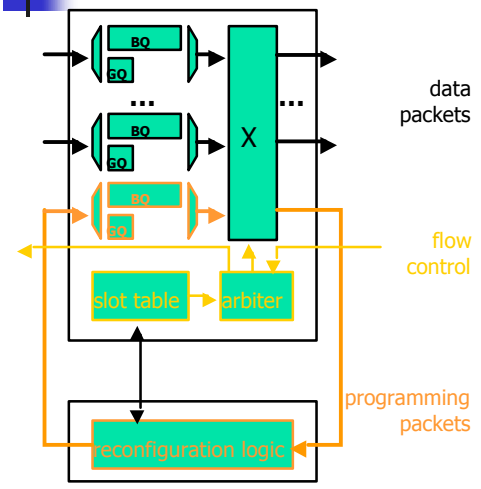
Router implementation

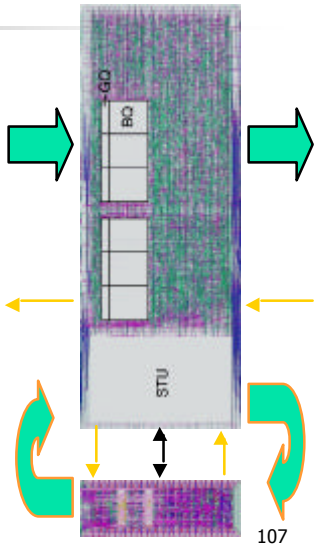
- Memories (for packet storage)
  - Register-based FIFOs are expensive
  - RAM-based FIFOs are as expensive
    - 80% of router is memory
  - Special hardware FIFOs are very useful
    - 20% of router is memory
- Speed of memories
  - registers are fast enough
  - RAMs may be too slow
  - Hardware FIFOs are fast enough



routers based on register-file and hardware fifos drawn to approximately same scale (1mm2, 0.26mm2)

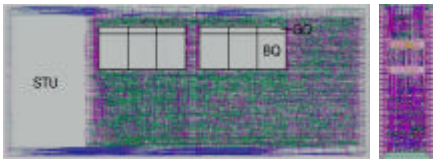
Layout





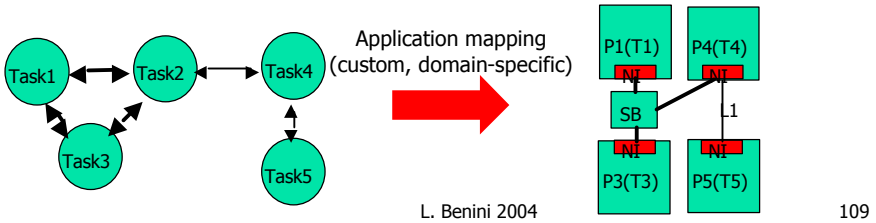
Results

- 5 input and 5 output ports (arity 5)
- 0.25 mm2 CMOS12
- 500 MHz data path, 166 MHz control path
- flit size of 3 words of 32 bits
- 500x32 = 16 Gb/s throughput per link, in each direction
- 256 slots & 5x1 flit fifos for guaranteed-throughput traffic
- 6x8 flit fifos for best-effort traffic



Xpipes: context

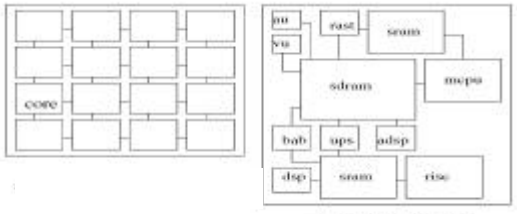
- Typical applications targeted by SoCs
  - Complex
  - Highly heterogeneous (component specialization)
  - Communication intensive
- Xpipes is a synthesizable, high performance, heterogeneous NoC infrastructure



Heterogeneous topology

SoC *component specialization* leads to the integration of *heterogeneous cores*

Ex. MPEG4 Decoder



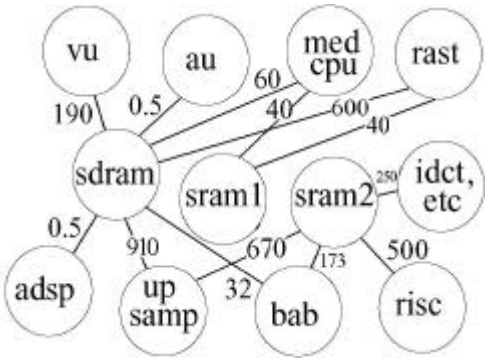
- Non-uniform block sizes
- SDRAM: communication bottleneck
- Many neighboring cores do not communicate

On a homogeneous fabric:

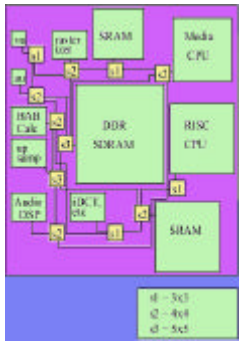
- Risk of under-utilizing many tiles and links
- Risk of localized congestion

Example: MPEG4 decoder

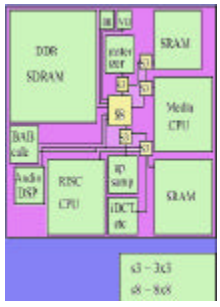
- Core graph representation with annotated average communication requirements



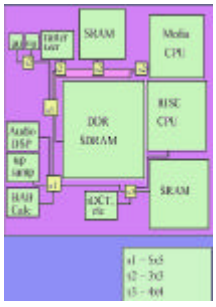
NoC Floorplans



General purpose: mesh

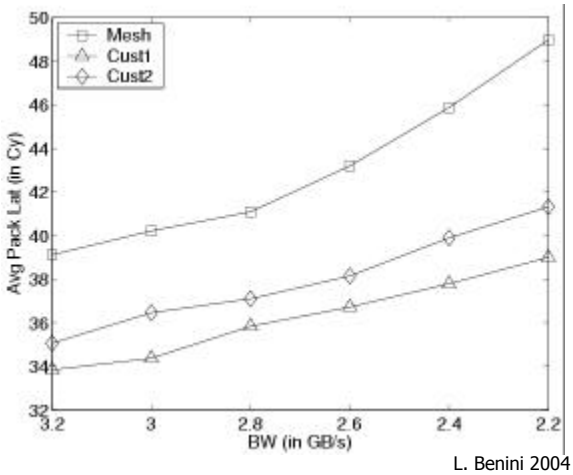


Application Specific NoC1 (centralized)



Application Specific NoC2 (distributed)

Performance, area and power



Less latency and better Scalability of custom NoCs

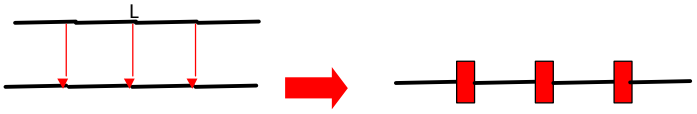
- Relative link utilization (customNoC/meshNoC): 1.5, 1.55
- Relative area (meshNoC/customNoC): 1.52, 1.85
- Relative power (meshNoC/customNoC): 1.03, 1.22

Xpipes: features

- Source based routing
  - Very high performance switch design
- Wormhole switching
  - Minimize buffering area while reducing latency
- Pipelined links
  - Link data introduction interval is not bound by wire delay
  - Link-latency (# of repeater stages) insensitive operation
- Parameterizable network building blocks
  - Plug-and-play composable for arbitrary network topology
  - Design time tunable buffer size, link width, virtual channels, # of switch I/Os
- Standard OCP interface

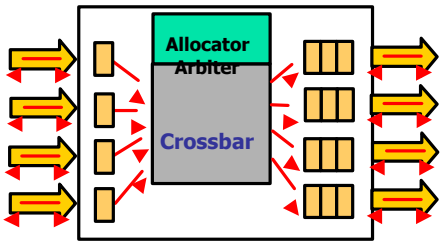
Link delay bottleneck

- Wire delay is serious concern for NoC Links
  - If NoC "beat" is determined by worst case link delay, performance can be severely limited
- ⇨ Pipeline links
  - Delay is transformed in Latency
  - Data introduction speed is not bound by link delay any longer!



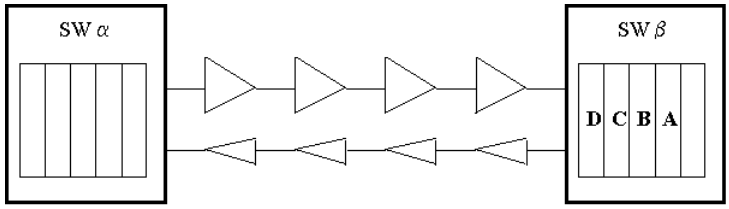
Xpipes Switch soft IP

- Output Buffering
  - ✓ Dual ported memory bank, purposes:
    1. Buffering for performance (tunable area/speed tradeoff)
    2. Error recovery on NACK
- Tuned for pipelined unreliable links



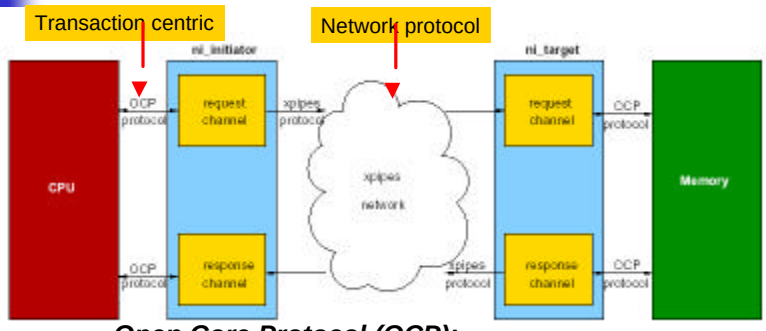
- ACK/NACK flow control
- 2-stages pipeline
- High speed (1GHz @ 130nm)
- Wormhole switching
- Arbitration: fixed priority, RR
- Source-routing

Flow control



- Transmission
- ACK and buffering
- NACK
- ACK/NACK propagation
- Memory deallocation
- Retransmission
- Go-back-N

Xpipes Network Interface Soft IP

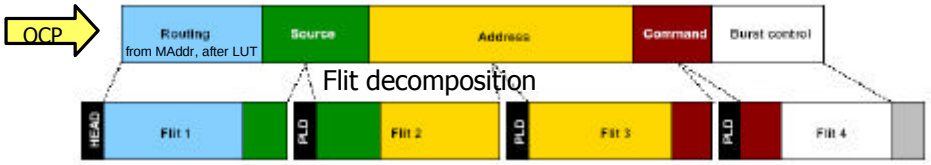


Open Core Protocol (OCP):

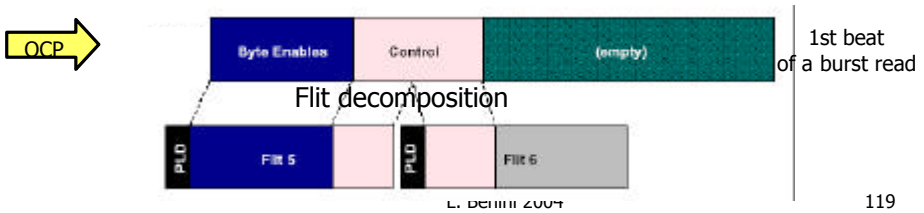
- End-to-end communication protocol
- Independence of request/response phases
- Can be tailored to core features
- Support for sideband signals (e.g., interrupts)
- Efficient burst handling
- Supports threading extensions

Xpipes packeting mechanism

Header register (about 50 bits): one for every transaction



Payload register: one for every burst beat

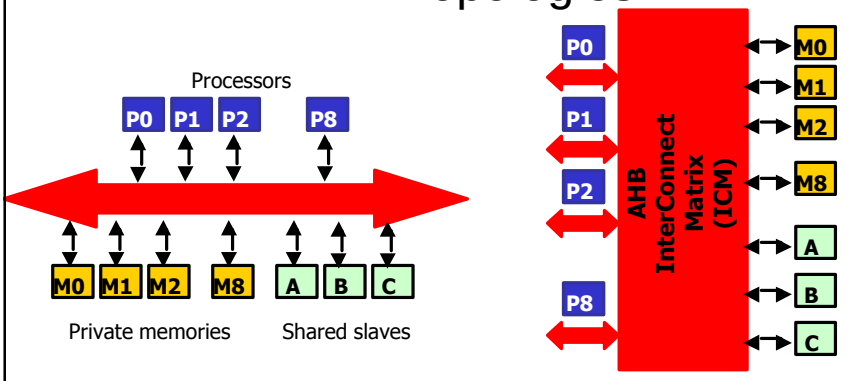


Xpipes design challenges

- The fight against latency: multi-hop topologies are at a disadvantage
  - Low number of stages per hop
  - Overclock the network
- Minimize the price for flexibility
  - Synthesis-aware design
  - Use specialized leaf cells

NoCs at work: cross-benchmarking

AMBA Topologies

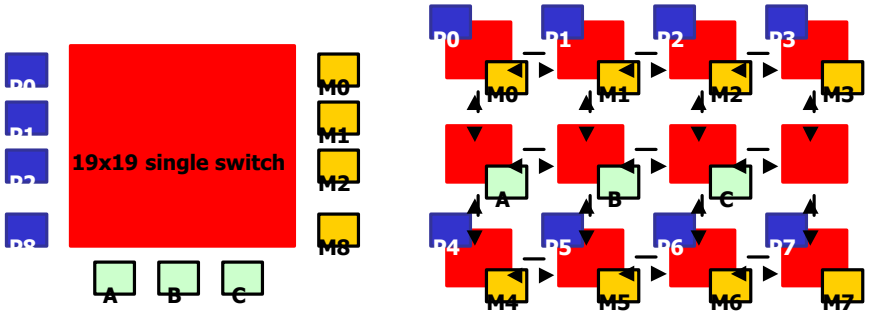


Shared bus

Multilayer (crossbar)

Topologies under test

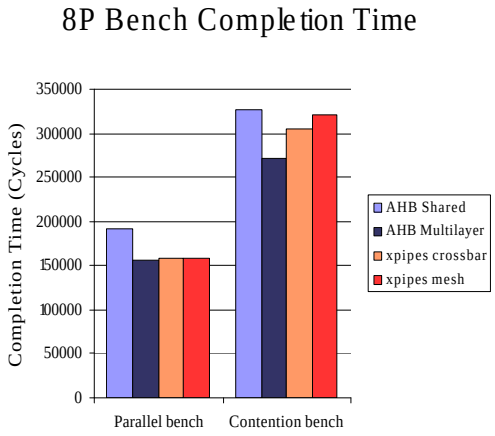
Xpipes topologies



Crossbar-like

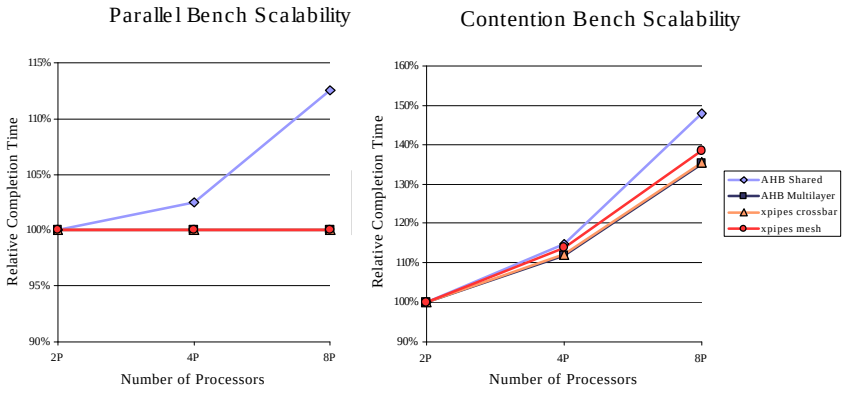
Mesh

Benchmark execution time



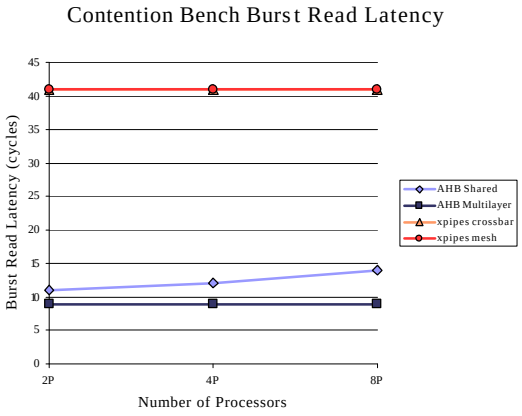
- AHB Shared: saturated with 8P
- AHB ML: best case (full crossbar, no arbitration latency)
- Xpipes: good performance due to available bandwidth, despite packeting latency penalty

Scalability results



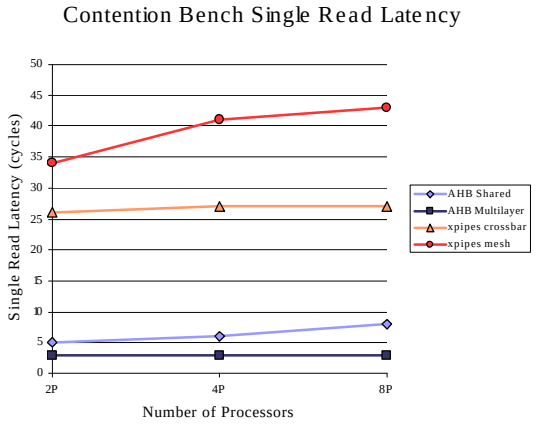
- Xpipes crossbar scales as AHB ML
- Xpipes mesh scales almost as well (yet, distributed topology!)

Latency analysis



- Burst reads: to private memories
- Xpipes latency is the target for performance optimization!!

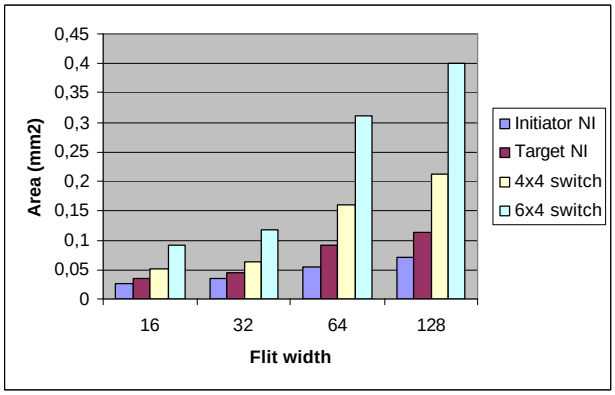
Latency analysis



- Single reads: to shared memory
- Xpipes mesh scales worse than the crossbar due to link congestion and more hops

Xpipes area/frequency

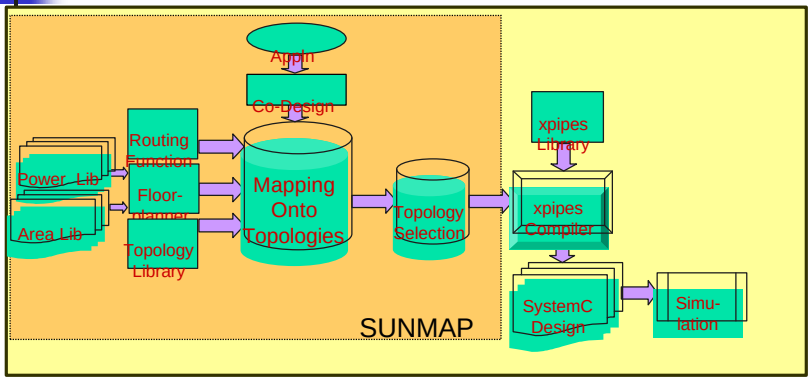
0.13 um technology, 4-flit output buffers



- Initiator NI: 1 GHz
- Target NI: 1 GHz
- 4x4 switch: 1 GHz
- 6x4 switch: 875-980 MHz

A 3x4 Xpipes mesh with 8 processors and 11 slaves consumes ~2,6 mm2

NoC synthesis flow



In cooperation with Stanford Univ.

Topology mapping (SUNMAP)

Heuristic approach with several phases

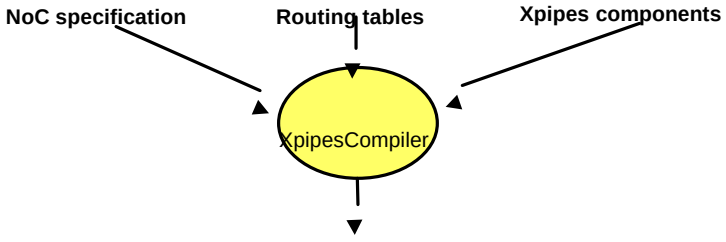
Initial mapping using a greedy algorithm (from communication graph)

1. Compute optimal routing (using flow formulation)
2. Floorplan solution
3. Check area and bandwidth constraints
4. Compute mapping cost

Iterative improvement loop (Tabu search)

Allows manual and interactive topology creation

Topology instantiation (XpipesCompiler)

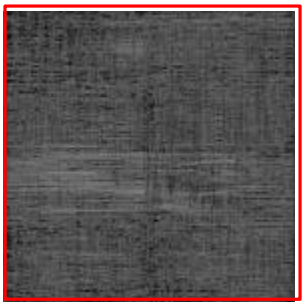


- Parsing specification (tree structure)
- Create a class template for each type of network component
- Components optimizations (I/O ports, buffer sizing)
- Hierarchical instantiation of the system (sc_main)
  - Synthesis view
  - Simulation view

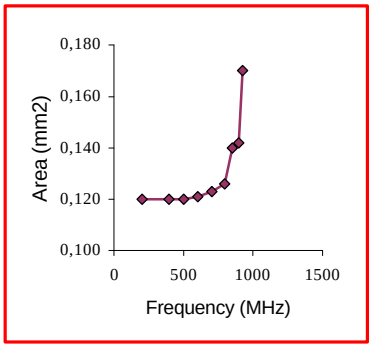
Synthesis back-end

- Fully synthesizable soft IPs
  - Compatible with state-of-the art design implementation flows

Switch layout



Area vs. frequency tradeoff



Outline

- Introduction and motivation
  - Physical limitations of on-chip interconnect
  - Communication-centric design
- On-chip networks and protocols
  - Designing NoCs
  - NoCs case studies
- Software aspects of on-chip networks
  - Programming abstractions, OS, software development toolkits
  - Quantitative analysis

Programming for NoCs

- The programmer model
  - Sequential: no parallelism is exposed to the programmer
  - Parallel: multiple threads/tasks
    - Shared memory: communication is "implied" by shared memory access
    - Message passing: communication is explicit in messages
- Parallelism extraction vs. parallelism support

Sequential Programming Model

- Naming: Can name any variable (in virtual address space)
  - Hardware (and perhaps compilers) does translation to physical addresses
- Operations: Loads, Stores, Arithmetic, Control
- Ordering: Sequential program order
- Performance Optimizations
  - Compilers and hardware violate program order without getting caught
    - Compiler: reordering and register allocation
    - Hardware: out of order, pipeline bypassing, write buffers
  - Retain dependence order on each "location"
  - Transparent replication in caches

SM Programming Model

- Naming: Any process can name any variable in shared space
- Operations: loads and stores, plus those needed for ordering
- Simplest Ordering Model:
  - Within a process/thread: sequential program order
  - Across threads: some interleaving (as in time-sharing)
  - Additional ordering through explicit synchronization
- Can compilers/hardware weaken order without getting caught?
  - Different, more subtle ordering models also possible

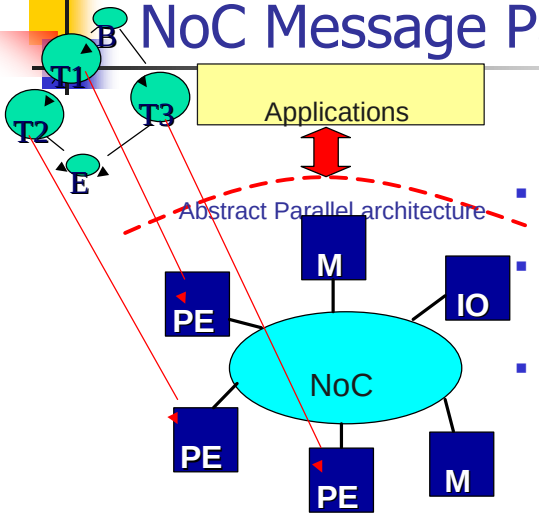
MP Programming Model

- Naming: Processes can name private data directly.
  - No shared address space
- Operations: Explicit communication through send and receive
  - Send transfers data from private address space to another process
  - Receive copies data from process to private address space
  - Must be able to name processes
- Ordering:
  - Program order within a process
  - Send and receive can provide pt to pt synch between processes
  - Mutual exclusion inherent + conventional optimizations legal
- Can construct global address space:
  - Process number + address within process address space
  - But no direct operations on these names

The Case for Message Passing

- MP exposes communication
  - Eases taking into account communication costs at the application level
  - Better control and predictability
- MP is scalable
- Current specification styles (e.g., DSP) match MP abstraction
- Achieves higher performance
- Programming/porting harder starting from single-thread computation

NoC Message Passing

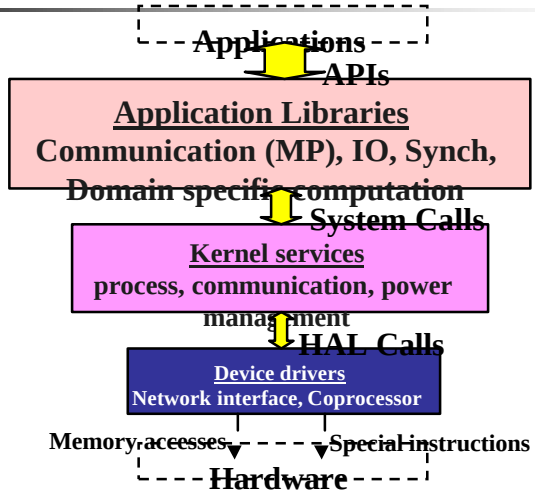


- How much HW detail?
  - E.g. exposing network latency?
- How much control?
  - E.g. Guaranteed QoS of Best effort?
- Implicit vs. explicit traffic
  - E.g. Instruction cache misses

Middleware

- Traditional RTOSes
  - Single-processor master
  - Limited support for complex memory hierarchies
  - Focused on performance
- The NoC Middleware
  - Natively parallel & scalable
  - Supports heterogeneous memory, computation, communication
  - Energy/power aware

Middleware Architecture



NoC API design

- Supporting explicit message passing
- High degree of control on performance
  - Real-time requirements RT/MPI
  - QoS, resource reservation
- Tailored for the target application & hardware platform (generality vs. performance)
  - Low memory usage
  - High level of hardware support (software emulation is slow)

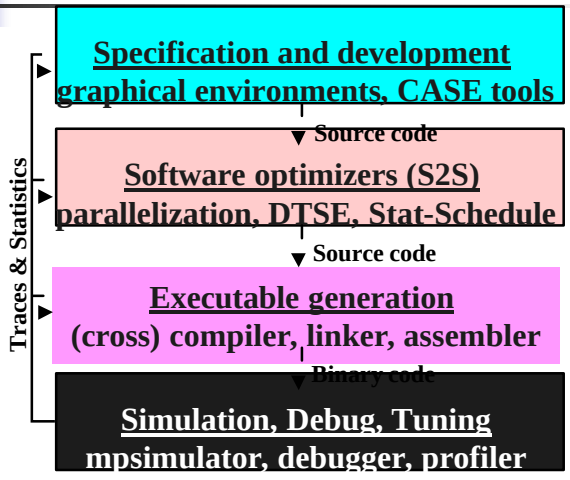
Kernel Design

- Task management (static and dynamic)
  - Allocation
  - Scheduling
  - Context switches
- Resource management
  - Synchronization
  - Fair allocation
  - Power management
- Automatic application specific kernel generation
  - Lightweight (low memory & resource usage)
  - Fast & predictable

Development Environment

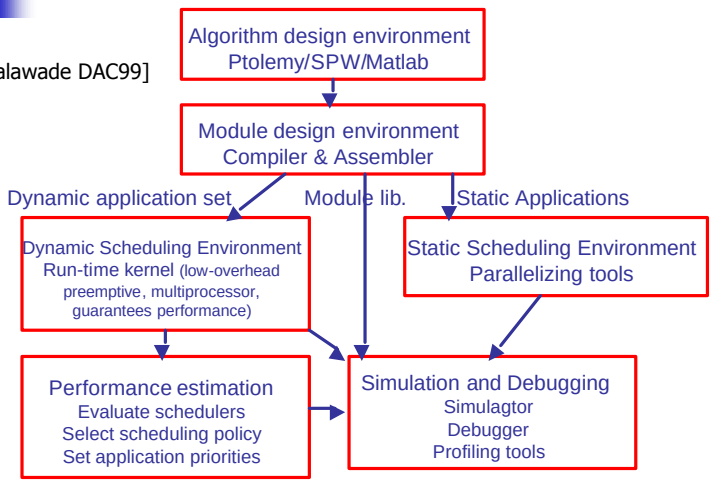
- Software development environment
  - Development tools
  - Debug tools
- Target platform
  - Simulation
  - Emulation
- Access to middleware libraries
- Flexible for many hardware platforms

Architecture of DD tools



Example: Daytona SDE

[Kalawade DAC99]



Outline

- Introduction and motivation
  - Physical limitations of on-chip interconnect
  - Communication-centric design
- On-chip networks and protocols
  - Designing NoCs
  - NoCs case studies
- Software aspects of on-chip networks
  - Programming abstractions, OS, software development toolkits
  - Quantitative analysis

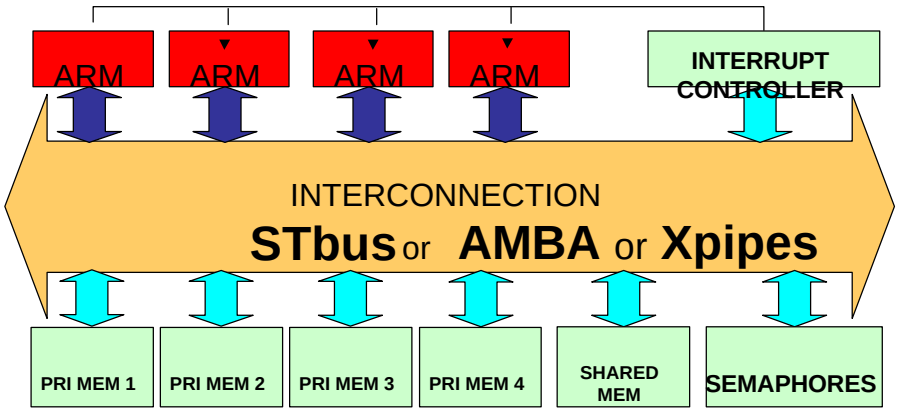
Traffic Modeling

Traditionally used traffic models trade-off accuracy with complexity:

- ***Stochastic traffic models***
  - Analytical distributions
  - Easily parameterizable
- ***Trace-based models***
  - Higher accuracy
  - Does not consider dynamic traffic-dependent effects (e.g. inter-processor communication)
- ***Functional traffic***
  - Traffic directly generated by running applications
  - Requires OS support

Complexity
Accuracy

MPARM Architecture

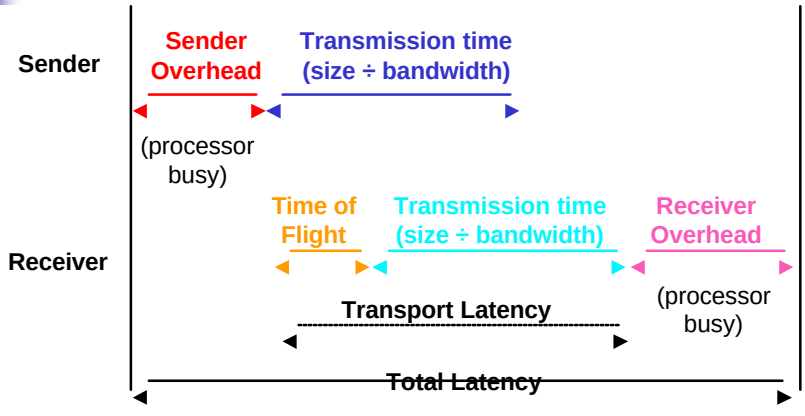


Approach to design space analysis

- Exploring of meaningful points in the design space
  - Modelling accuracy emphasized

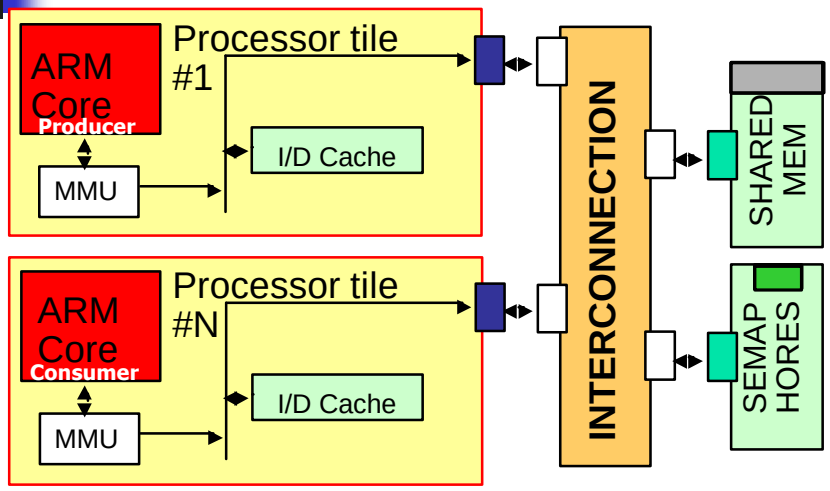
- Power and performance analysis for different:
  - Classes of applications (computation vs communication dominated)
    - Software architectures (stand-alone vs OS supported appln)
      - System configurations (cache size, memory latency, ..)

Latency breakdown

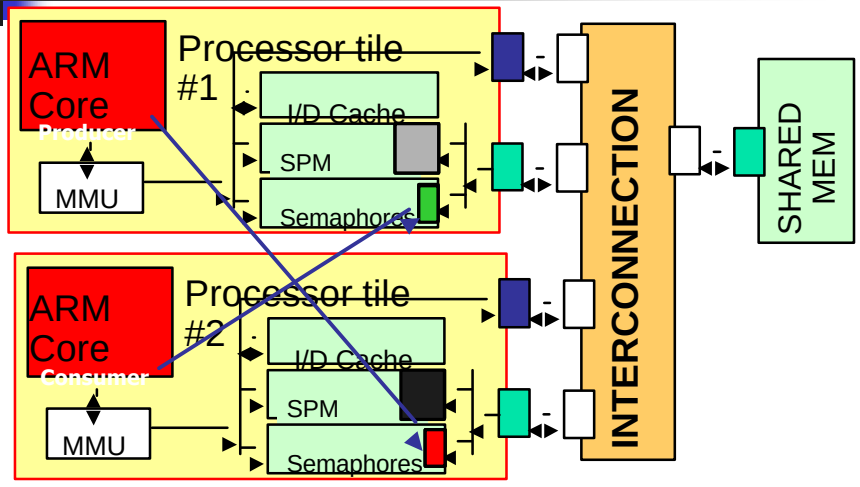


Total Latency = Sender Overhead + Time of Flight + Message Size ÷ BW + Receiver Overhead

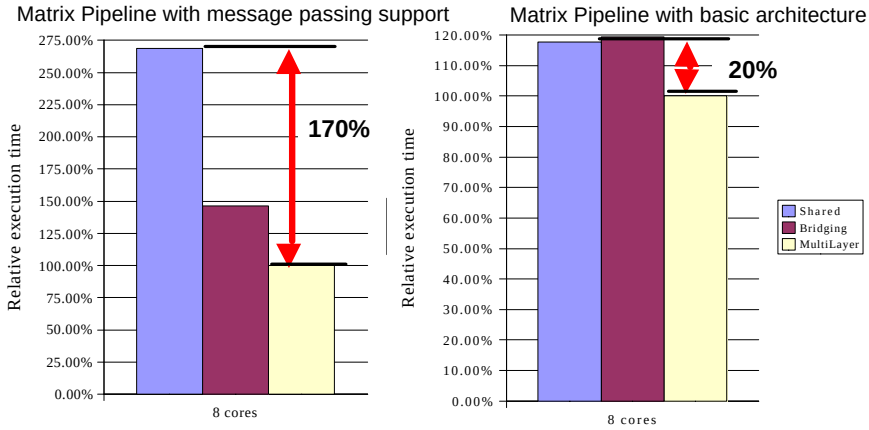
Basic architecture



Support for message passing

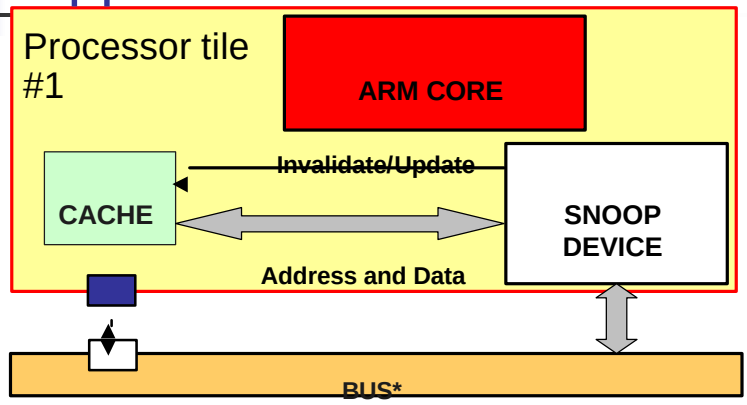


HW support for MP: results



Send+Receive cost: 35KCycles (basic architecture) vs. 4KCycles (MP support)
Configuration: 4 Processors, Shared bus

Support for UMA



*cannot be a generic interconnect!

Summary

- Paradigm shift towards NoCs
- Designing NoCs
  - Huge design space
  - Unclear if there is a winner
  - Research will solidify soon
- Hardware/software interfaces is currently the main bottleneck