Towards Light-Weight Intra-CMP Network Interfaces

Manolis Katevenis

Foundation for Research & Technology - Hellas (FORTH)
Heraklion, Crete, Greece
Interprocessor Communication: End-to-End Performance

- As slow as the slowest component – the “bottleneck”
- Over decade-long periods, industry focuses on resolving the currently perceived bottleneck…
- … and then the next bottleneck is exposed!
70’s & 80’s: Bottleneck = Network Links

- Bottleneck = 10 Kb/s – 10 Mb/s WAN / LAN links, hence…
- Push for faster links; in the meanwhile…
- Networking protocols in software
  – we still “suffer” from them…
90’s: User-Level Communication

- 100 Mb/s – 1 Gb/s Network Links required…
- Hardware support for switching; in the meanwhile…
- Operating System Call surfaced as the next bottleneck
  – solution: User-Level (rather than kernel-mode) access to the network interface device
Today: Network Thruput \( \approx \) Memory (L1) Thruput

- 10 – 100 Gb/s Network or NoC – as fast as memory; hence…
- NI tightly-coupled to processor – no bridges or I/O buses
- NI side-by-side with Cache Controller – convergence?
  - send \( \approx \) write (store); receive \( \approx \) read (load)
In the Chip-Multiprocessor Environment

- Connect to all non-local memory (L2,…) via “the” network:
  - Memory speed \(\approx\) Network speed

- *Light-Weight Network Interface:*
  - NI cost must be quite less than processor & L1 cost
Thesis – Outline:

• NI cost & coupling to the processor
• Communication paradigm
  – Cache-coherent shared memory, versus
  – Message passing, versus
  – Common hardware support for both?
• Hardware primitives
  – Remote DMA
  – Remote Queues
• NI side-by-side with Cache Controller – convergence?
  – send ≈ write (store); receive ≈ read (load)
• Events triggering Actions
Thesis – Outline:

• **NI cost & coupling to the processor**
  • Communication paradigm
    – Cache-coherent shared memory, versus
    – Message passing, versus
    – Common hardware support for both?
  • Hardware primitives
    – Remote DMA
    – Remote Queues
  • NI side-by-side with Cache Controller – convergence?
    – send ≈ write (store); receive ≈ read (load)
• Events triggering Actions
Undesirable: NI requires Dedicated Memory of its own

- Partitioned memory can provide sufficient throughput, but…
- Promotes data copying
- Underutilizes the total memory space
NI should use the Processor’s Memory

- Space for the NI data structures (at least the large ones) should be dynamically allocated in the processor’s “local” memory
- Sufficient memory throughput provided through bank interleaving
Thesis – Outline:

• NI cost & coupling to the processor

• **Communication paradigm**
  – Cache-coherent shared memory, versus
  – Message passing, versus
  – Common hardware support for both?

• **Hardware primitives**
  – Remote DMA
  – Remote Queues

• NI side-by-side with Cache Controller – convergence?
  – send ≈ write (store); receive ≈ read (load)

• Events triggering Actions
Communication Primitives

• Cache Coherent Shared Memory
  – hope that data find their own way
  – hard problem to solve in hardware, and hard to scale

• Explicit movement of data in a Global Address Space
  – software guides the data to move
    • application program(mer), and/or
    • runtime system
  – pairwise bulk movement: Remote DMA
  – synchronization / collective op’s: Remote Queues
In-place Data Delivery: Remote DMA

- Allows zero-copy communication
- Allows adaptive (multipath) routing
- Requires buffer space allocation per producer-consumer pair
Multi-Party Synchronization: Remote Queues

Remote Queues differ from RDMA as follows:
- receive buffer space shared among many senders
- speeds up polling of multiple receive channels

Atomicity of multiplexing/demultiplexing: Synchronization Primitive
Thesis – Outline:

• NI cost & coupling to the processor

• Communication paradigm
  – Cache-coherent shared memory, versus
  – Message passing, versus
  – Common hardware support for both?

• Hardware primitives
  – Remote DMA
  – Remote Queues

• **NI side-by-side with Cache Controller – convergence?**
  – send ≈ write (store); receive ≈ read (load)

• Events triggering Actions
Queue Memory Similarities to Cache Memory

- Sender (tail end of queue): compose messages in local memory, then send
- Receiver (head end of queue): messages “appear” in local memory
Message Send Similar to Cache Writes

- A message is sent as fast as executing one or a few store instructions that hit in L1 cache
- Comparable to write-update coherent cache protocols
Sequential Accesses – Prefetching – Queues

- Sequential array accesses:
  - allow for optimized cache replacement – prefetching
- Queues – especially long ones – behave in the same way
Read Misses are like Remote Read DMA’s

- Unforeseen (hence, not pre-fetched) need to use an item from a given place in global address space
Multiple Readers from a Shared Queue

• Queue head must probably reside in a “nearby” memory, normally not the local memory of most (or all) readers
• Request one message to be forwarded into a private, local queue via a remote atomic operation
Thesis – Outline:

• NI cost & coupling to the processor
• Communication paradigm
  – Cache-coherent shared memory, versus
  – Message passing, versus
  – Common hardware support for both?
• Hardware primitives
  – Remote DMA
  – Remote Queues
• NI side-by-side with Cache Controller – convergence?
  – send ≈ write (store); receive ≈ read (load)
• Events triggering Actions
**Powerful Primitive: Events trigger Actions**

- **Action**: compose and send one or more packet(s)/message(s)
- **Example 1**: Remote Read DMA server: arrival of a request message in its queue triggers a remote write DMA in response.
- **Example 2**: Write DMA completion notification: count arriving bytes (per session); when counter expires, send (enqueue) notification message(s).
- **Example 3**: barriers (see next).
Barrier Synchronization Example

• Detection Phase:
  – action triggered when e.g. 3 messages received;
  – action notifies a specific queue;
  – notified queues form a tree.

• Notification Phase:
  – action triggered when one message is received;
  – action notifies e.g. 3 other queues;
  – notified queues form a tree.
Wait-on-Queues should replace most Interrupts

- Interrupts needed when (# ready tasks) > (# processors)
- CMP’s: (# processors) > (# ready tasks)
  - many processors sit idle, waiting for work to arrive
  - good for power consumption
  - good for task latency
- Processor waits on a set of empty queues, for any of them to become non-empty (like “select” system call)
  - variation: message arrival at a queue triggers an action to notify a central queue; processor waits on that central queue, alone.
Hardware-Assisted Software Cache Coherence

- Could be implemented using the above mechanism where events may trigger actions / notifications
- Dedicated processors where the OS / runtime system runs
Acknowledgements

• European Union Funding:
  – SARC
  – HiPEAC
  – UNiSIX

Crete:

• Angelos Bilas
• Dionisis Pnevmatikatos
• Sven Karlsson
• Manolis Marazakis

• Vassilis Papaefstathiou
• George Kalokerinos
• Angelos Ioannou

• Stamatis Kavadas
• Michalis Papamichael
• George Mihelogiannakis
• Evangelos Vlahos

• Georgi Gaydadjiev, Delft
• Paul Kelly, Imperial College
Conclusions

• High-Speed Interprocessor Communication
  – hardware primitives: few, simple, general-purpose

• Cache Controller – Network Interface Convergence
  – Message Send ≈ a few store hits into L1 cache
  – Message Receive ≈ a few load hits from L1 cache
  – Bulk transfers via RDMA
  – Synchronization via Remote Queues and automatic triggering of actions