# Interprocessor Communication: Towards Cache Integrated Network Interfaces

Vassilis Papaefstathiou[1] and Michael Papamichael[1]

In collaboration with :
Stamatis Kavadias, George Kalokairinos, Dionisios Pnevmatikatos and Manolis Katevenis

*Institute of Computer Science (ICS),*
*Foundation for Research and Technology Hellas (FORTH),*
*Vassilika Vouton, P.O.Box 1385, GR-71110 Heraklion, Crete, Greece*
*E-mail: {papaef,papamix}@ics.forth.gr*

**ABSTRACT**

**Recent advances in silicon technology allow today's systems to host a few processor cores in the same chip. In the upcoming manycore era, parallel systems will depend on multi-core chips to allow their performance to scale. Scalability can only be achieved with a synergistic use of the available cores and thus the efficient communication between them is increasingly important. This *Interprocessor Communication* takes place in the processors' Network Interfaces (NIs) and thus requires low-cost and high performance NI architectures.**

**Our current research focus is on future on-chip NIs where the NIs are tightly coupled to the processors and the memory hierarchy. This paper introduces the on-chip environment for these NIs and discusses the scalability issues. We propose the integration of the NI inside the cache controller and a simple interface that allows only a few store/load instructions to send/receive messages at L1 cache rates.**

KEYWORDS: network interface; interprocessor communication; multi-core systems

## 1 Introduction

Networks originated in the seventies, with interconnection links of a few Kbits/s in throughput. Under those circumstances the network was treated as "just another" peripheral I/O device. Networking protocols were defined in that slow environment and everything was performed in software. Current systems and protocols, unfortunately, are still carrying elements inherited from those early choices. In traditional, non-virtualized NI's, an operating system call is needed to start up any network operation, resulting in very heavy overhead. Research in the nineties [MK96, MFHW96] led to virtualized NI's that allow user-level access to their control registers, thus avoiding the system call; still, the I/O bridge separates the NI from the processor and the memory, thus imposing a latency overhead of tens or hundreds

[1]The authors thankfully acknowledge the support of the EU FP6-IST program through the SIVVS, UNIsIX, SARC projects and the HiPEAC NoE.

of nanoseconds on operation start up [BAH+05]. In such *loosely-coupled* NI architectures, because of the above overheads, programmers consider that only large-block transfers are efficient. This, however, is an artifact of the traditional NI organization, and not an intrinsic necessity in interprocessor communications. Over time, conditions have radically changed: network throughput is no longer small, when compared to processor-memory throughput; and the network is no longer "just another" I/O device. The new situation dictates that the communication between compute engines located on the same chip will be achieved through a *Network-on-Chip (NoC)* that will replace the old-times "memory bus". Processors will communicate with each other, and with all memory levels but L1, through this new "network". Hence, the throughput of this new network is by definition equal to the memory throughput; and individual processors see no other I/O than this new network.

## 2 Scalability at low cost

In this new environment we have to take into serious consideration the cost and scalability of the architectural decisions for the NIs. Traditional NIs are expensive because they require extensive dedicated buffer memory in order to perform accesses to it when the processor accesses its own memory in parallel; the main-memory throughput is not sufficient for both of them in order to meet real-time latency guarantees. The future on-chip NIs will be tightly coupled to the processor, L1 caches, accelerators and their local memories. These *Level-1 (L1)* NIs are the interfaces between the individual cores and the NoC that is used for on-chip (short-range) communication.

However, each core should be able to communicate with off-chip (long-range) resources, which we believe will require a *Level-2 (L2)* NI; potentially more than one will exist in a chip. The challenge is to *unify the interprocessor communication protocols* either on-chip or off-chip to support a fully range of extensible systems (multi-chip multi-core processors / systems / clusters).

Our current focus is on L1 NIs and we consider that they should be low-cost enough in order to afford having them next to each and every processor in a chip. They should have a reasonable cost when compared to units that they connect to: processors, L1 caches, accelerators, local scratchpad memories. Thus, future NIs cannot afford to require extensive dedicated memory, but must instead *share* local memory with the processing engines. Sharing must be dynamic: when a core executes compute-intesive tasks, most of its memory must be allocatable to them; when core executes communication-intensive tasks, a larger portion of its memory can be used by the NI.

The low cost requirement dictates that the NI (L1) must use a very small number of dedicated registers or small FIFOs, and for the rest of its requirements must share the node memory. Within this node memory, NI tables, queues and buffer areas must occupy a space that is reasonably small, depending on the current intesity of communication, and also configurable at run-time.

On the other hand, the *scalability* requirement dictates that each NI must be able to operate in a system where potentially thousands of nodes exist to communicate with. At the same time, scalability with respect to virtualization must be provided: it must be possible to have multiple processes and threads, each with its own protected communications environment. For such systems to become feasible, we must avoid NI data structures that grow linearly with the number of nodes in the system, and we must restrict the space occupied by

NI data structures to be proportional to the current degree of virtualization or the current number of active connections.

# 3 Cache integration of the Network Interface

Our view in that tightly coupled environment is that the processor's interface with the NI can be as as simple and as fast as the local memory interface. This on-going work studies the aspects of such an architectural decision and presents some key issues that need to be addressed.

A load-store interface could allow a series of memory accesses (e.g. stores) in pre-configured (but run-time configurable) addresses to result in a message transmission to the NoC. If the local memory is configured as cache, we need to carefully map these addresses into cache-lines and conform with the cache mechanisms. The above restriction requires support by the cache controller so as to be able to allocate portions of the cache memory for special functions supporting messaging.

The allocation of a cache segment could be either coarse-grained (e.g. allocate one or a few ways from an N-way set associative cache) or fine-grained (e.g. allocate only a number of cachelines). The granularity of allocation is essential for the applications running on the processor. Computation-intensive applications can benefit from the use of this memory resource mostly as a cache/scratchpad and on the other hand communication-intensive applications can benefit from using a large portion of this memory for fast message transmition and reception.

This allocation will need support from the run-time system, possibly in conjunction with a user-level library to handle the low-level details. A few bits located in the tag part of each cacheline are enough to mark a specific cacheline for special use. Moreover, we need a few programmable registers inside the cache controller to control its behaviour on those special cachelines. In order to set up the cache controller and use the special cachelines – issue stores and loads – we have two choices: *(i)* we can reserve a bit in the address (shadow address space) or *(ii)* request for an additional bit in the TLB – set by the run-time system upon configuration – that will be supplied to the cache.

Moreover, we need to map the NI data structures into the data and tag parts of the cache. The actual message data are stored in the data part of the cacheline and the state in the tag part. The tags of a cacheline marked for NI use are can be ignored by the traditional matching mechanism and can always return a hit.

These special cachelines map very well with queues as a generic communication primitive. They can be used for composing and sending messages to many potential destinations as well as receiving messages atomically from potentially many sources. The multi-destination queues avoid the need for large number of queues that grow linearly with the number of communicating nodes. Likewise, the multi-source queues avoid the need for per-source buffers and can also serve as a synchronization mechanism. We can also support queues with multiple readers that are valuable for job dispatching. For each message a header is required, which carries at least a destination address and a size field.

The addressing scheme greatly depends on the system and the on-chip network and has not been yet finalized. We could follow a global address space approach where the addresses might have some virtual part to be translated by the network switches and/or the network interfaces – this approach could allow transparent process/thread migration.

The use of NI, as sketched above is based on the stores and loads that reach the cache. We have to carefully design the way addresses are used by software and reach the cache and how message completions are triggered; either for incoming or outgoing messages. We consider two alternatives: *(i)* all words can be written/read to/from the same address – signaling enqueues/dequeues – or *(ii)* each word in a message can have its own address – indicating a specific word inside the cacheline. Each of the two alternatives has its own implications. The first choice implies that the accesses must arrive in order to the cache interface – so as to correctly compose/receive a message – while in the second choice each operation is self contained. The message completion in the first alternative can be implemented with a simple counter while the second choice might require a completion bitmask. As far as the above issues are concerned, we have to carefully consider the optimizations performed by the modern processors to boost performance (out-of-order execution, weak memory ordering, speculative execution ) that might cause undesired behaviors.

All the above ideas on the cache integrated NIs are currently elaborated while the most mature of them are prototyped in FPGAs. We are also studying the NoC aspects and advanced network features for the NIs, such as flow-control and retransmissions, and their integration into the memory hierarchy.

# References


[BAH+05] J. Beecroft, D. Addison, D. Hewson, M. McLaren, D. Roweth, F. Petrini, and J. Nieplocha. QsNet2: Defining High-Performance Network Design. *IEEE-Micro*, 25(4):34–47, July-August 2005.

[MFHW96] S. Mukherjee, B. Falsafi, M. Hill, and D. Wood. Coherent Network Interfaces for Fine-Grain Communication. In *Proceedings of 23rd ACM Int. Symposium on Computer Architecture (ISCA 1996)*, pages 247–258, Philadelphia, PA USA, May 1996.

[MK96] E. Markatos and M. Katevenis. Telegraphos: High-Performance Networking for Parallel Processing on Workstation Clusters. In *Proc. 2nd IEEE Int. Symp. on High-Performance Computer Architecture (HPCA'96)*, pages 144–153, San Jose, CA USA, Feb. 1996.