Low-latency Explicit Communication and Synchronization in Scalable Multi-core Clusters

Christoforos Kachris, George Nikiforos, Vassilis Papaefstathiou,
Xiaojun Yang, Stamatis Kavadias, Manolis Katevenis
Institute of Computer Science (ICS), Foundation for Research and Technology - Hellas (FORTH)
Heraklion, Crete, Greece
e-mail: {kachris, nikiforg, papaef, yxj, kavadias, kateveni}@ics.forth.gr

Abstract— **One of the main challenges in the multi-core area is the communication and synchronization of the cores and the design of an efficient interconnection network that is scalable to multiple cores. In this paper we present an efficient implementation of a scalable system that is targeting multi-core systems. Each cluster node consists of 4 processors that support both explicit and implicit communication. Processor's cache is augmented with scratchpad and is merged with the network interface (NI) for reduced communication latency. All nodes are connected through a novel layer-2 switch that can support up to 20 nodes. The proposed system is designed and implemented using multiple FPGA boards and the performance evaluation presents the aggregate throughput of the system (with 16 processors) and the communication latency between that cluster nodes.**

Keywords: multi-cores systems, FPGAs, explicit and implicit communication, merged cache-network interface

I. INTRODUCTION

As the number of processing core per chip increases, so does the need for efficient and high-speed communication and synchronization support, so that applications can exploit the available cores. The memory hierarchies of these multi-core systems are based on one of the two dominant schemes -multilevel caches, or directly-addressable local *scratchpad* memories. Caches transparently decide on the placement of data, and use *coherence* to support communication, which is especially helpful in the case of *implicit* communication, i.e. when we do not know in advance which input data will be needed, or who last modified them. However, caches lack deterministic response time, and make it harder to the software to explicitly control and optimize data locality and transfers in the cases when it can intelligently do so. Furthermore, coherent caches scales poorly.

On the other hand, scratchpads offer predictable performance which is required in real-time applications. They also offer scalable general purpose performance by allowing explicit control and optimization of data placement and transfers. In the case of scratchpad, the interprocess communication is *explicit* meaning that the software (the application, or compiler, or runtime system) is able to indicate physical placement or transfers. The explicit communication is mainly based on remote direct memory accesses (RDMA) that is efficient and it becomes possibly in cases when the producer knows who the consumer will be. The main drawback of the scratchpad is that it reduces the programming efficiency, since extra effort must given to the synchronization and the consistency of the system.

Furthermore, the main challenges in these multi-processor systems is the design of an efficient inter-connection scheme that achieves high throughput and low latency communication. In the domain of Chip-Multi-Processors (CMP) several schemes have been proposed for the interconnection of the tiles such as mesh, concentrated mesh, torus, etc. These interconnection schemes provide high throughput but introduce significant latency. On the other hand, interconnections based on crossbars can achieve high throughput while also introducing lower latency [1]. The proposed interconnection scheme is targeting a multi-processor FPGA-based system that spans over several boards [2] [3]. It consists of multiple processing nodes (each consisting of 4 processors connected using a level-1 switch) and a level-2 off-chip interconnection network as shown in Figure 1.

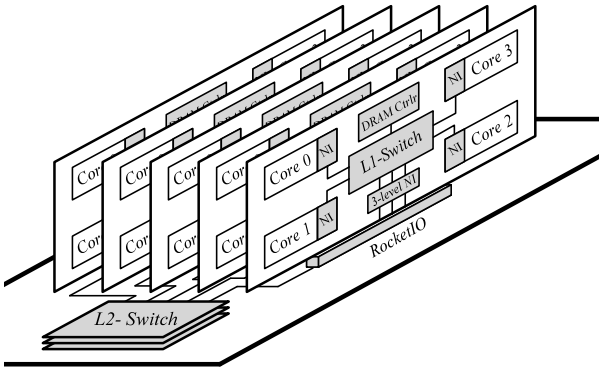

Fig. 1. Prototype block diagram

In this paper we focus on an efficient FPGA implementation of a scalable hierarchical system that consists of an efficient level-2 off-chip switch and each node consists of 4 processors that support both explicit and implicit communication and synchronization. The proposed system can support up to 20 nodes (80 processors) and the total aggregate throughput in the level-2 switch is 62.5Gbps. The main features of the proposed system are:

- Each cluster consists of 4 processors with explicit communication and synchronization
- Each processor's cache is merged with the scratchpad and the NI

- Switch size: up to 20x20 or 6x6(with 3planes) clusters
- Line speed rate: 3.125 Gbps
- Supports variable-size packet from 8 to 70 bytes.
- Flow control: 14 packet-based credits
- High performance: high sustainable throughput under a wide range of traffic models and low latency
- QoS support: fault-tolerant, traffic priority support
- Priority service: 3-level priority

The paper is organized as follows. Section II describes the explicit communication and synchronization of the processors across the scalable platform. Section III presents the level-2 switch, used to connect the clusters, which is based on a novel sequential iterative matching algorithm. Section IV presents the performance evaluation of the system using up to 4 clusters (16 cores). Finally, Section V concludes the paper.

II. Explicit Communication and Synchronization

This section describes the architecture of our local memory, which can be dynamically *configured* as partly-cache and partly-scratchpad, and also describes the software interface to the hardware mechanisms for explicit communication and synchronization. These are all based on common hardware actions, thus *unifying* the cache controller and the network interface: for each access, check the state and tag bits of the addressed line, and act accordingly.

We generalize the traditional approach where the processor uses I/O control and status registers to initiate operations and poll for status or wait for notifications. In order to increase parallelism, multiple pending operations are supported, by means of multiple control and status registers. To reduce overhead, these multiple registers are *virtualized*, so as to be accessible in user-mode. We bring these mechanisms close to the processor, into caches, to reduce latency. To allow parallel access, we target private - as opposed to shared- caches, and integrate our NI mechanisms into second-level (L2)-as opposed to L1-caches, in order to provide sufficient scratchpad space for application data without affecting the processor clock (however the ideas are general and independent of that choice).

A. Memory Access Semantics: Cache, Scratchpad, NI

To integrate cache, scratchpad, and communication/ synchronization, we extend memory *access semantics* using *address translation* to identify explicitly managed (scratchpad) address regions, and cache line state bits to indicate different access semantics at finer granularity.

Local and remote scratchpad regions are identified by their physical address provided via address translation. The address translation mechanism is augmented with a few extra bits that explicitly determine whether an address region contains cacheable or directly-addressed (scratchpad) data, as shown in Figure 2. This is important when remote scratchpad regions are addressed, so that the hardware accesses them remotely, rather than locally caching them. It also obviates tag bit comparison to verify that a memory access actually hits into a scratchpad line; hence, tag bits of scratchpad areas can be used for other purposes, such as implementing communication semantics for RDMA commands, counters, and queues. Regions marked as scratchpad occupy a set of blocks in the data portion of the memory "way" block. Each of the blocks in the region is marked as *non-evictable* in its state bits. This marking allows the distinction of memory access semantics at cache block granularity, and is used to ignore the actual tag-matching of the hit logic. This mechanism allows for *runtime-configurable partitioning* of the on-chip SRAM blocks between cache and scratchpad use, thus adapting to the needs of the application that is being run at each point in time.

Multiple, virtualized communication control/status *registers* are implemented in scratchpad region lines. Other than plain scratchpad memory, lines in these regions
can be marked, in their state bits, as having three types of such special semantics:

- *communication*: (RDMA or message) command/status;
- *counters*: used for synchronization and notification through atomic increment operations;
- *queues*: used to atomically multiplex or dispatch information from/to multiple concurrently executing tasks.

The virtualized nature of these three types of event sensitive lines (ESLs) results as follows: ESLs can be freely

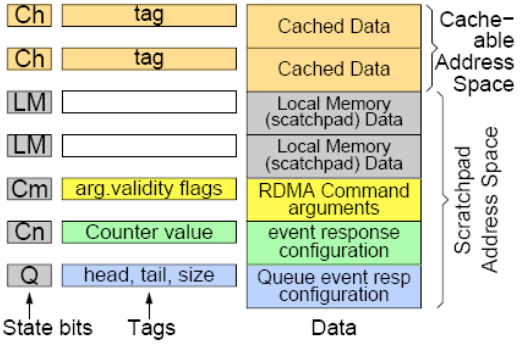

Figure 2. Cache line Types

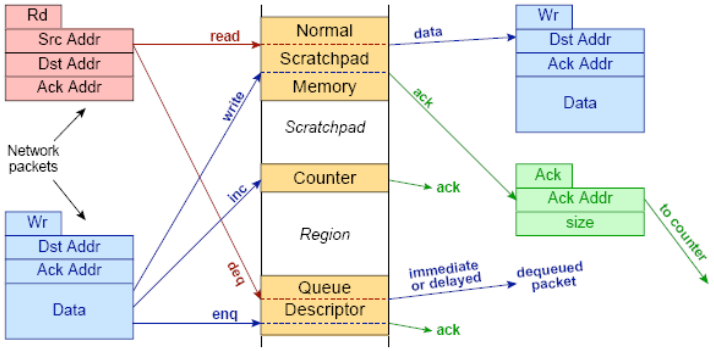

Figure 3. Event responses to RDMAs, Counters, and Queues

allocated in the (virtual) address space of any process. Each process can freely access locally allocated ESLs, in user-mode, independent of, and asynchronously to other processes. Virtualization also requires that address arguments passed to ESLs are given in virtual -rather than physical- space, and are protection-checked by hardware. Software can issue a remote DMA operation using 4 store instructions into a command buffer: (i) opcode and size; (ii) source (iii) destination, and (iv) acknowledgment addresses. Store instructions to scratchpad regions, identified by the processor TLB, should be processed in a pipelined fashion.

B. *EventResponses*

Event responses are hardware synchronization primitives, configurable by software and are provided by counter and queue event sensitive lines (ESLs). On every access, hardware checks the state and tag bits of the addressed line. When that is an ESL and a relevant condition is met, communication *response* is triggered, according to previous software configuration written in the ESL data block as shown in Fig. 3.

Counters are intended to provide software notification regarding the completion of an unordered sequence of operations, e.g. multiple transfer reception, or arrivals at a barrier. Counters perform atomic add-on-store. Software can write configuration information, and up to four notification addresses (which may correspond to other ESL's), in their data block. When the counter reaches zero, a pre-configured word is sent to the notification addresses.

Multiple-reader queues (mr-Q) accept asynchronous write (enqueue) and read (dequeue) accesses from any number of processors. Read requests arriving at an empty mr-Q are recorded, waiting until corresponding writes arrive, thus effectively *matching* read and write requests in time. When a write is matched with a read in the mr-Q, a response packet is triggered, with the data of the write sent to the response address of the read. Reads and writes to the mr-Q are buffered in scratchpad memory contiguous to the ESL, forming the queue body.

A lock can be implemented with an mr-Q, initialized to contain a single token-word (or a semaphore by initializing with multiple tokens); to acquire the lock, a task reads from the queue and waits for the response packet; to release the lock, the task writes the token back into the queue. Job dispatching can be implemented by writing task descriptors into an mr-Q; whenever processors becomes ready to run a new task, they read (dequeue) an ID from this queue.

III. SCLALABLE LEVEL-2 SWITCH

Four processors are connected using a on-chip level-1 switch forming a cluster. The clusters are connected using a level-2 switch fabric. The main challenge in designing a switch fabric is the switch arbitration, performed by a matching algorithm. Many matching algorithms have been proposed and investigated [4] - [11]. Among the matching algorithms, some of them are too complex to use in practice, such as the maximum weight matching (MWM). Some other algorithms, such as *i*SLIP [9], have lower complexity and are practical for hardware implementation. However, their performance is not as good as MWM when traffic pattern is bursty or non-uniform, especially at high loads.

For this system, an efficient matching algorithm, called special sequential iterative matching (SIM), is proposed for Combined Input Output Queue (CIOQ) switches. In order to make fast matching decision between the Input and Output Queues, several iterative matching algorithms have been proposed for CIOQ switches, such as PIM [6], *i*SLIP [9] and dual round robin arbitration scheme (DRR) [7]. In this system we designed a more efficient matching algorithm combining the efficiency of PIM2 and the performance of DRR, called special sequential iterative matching (SIM). The proposed scheme supports the following features:

- *Pipelined Iteration*

SIM works in a sequential mode; therefore it can be efficiently pipelined. In this case, the IQ can quickly scan, find and lock a matching pair between IQ and OQ. The iteration period is two clock cycles (one for request, another one for grant), thus a 2-stage pipelined request scheme can be achieved, which means that IQ can issue a request destined to the different OQ per clock.

- *Matching Arbitration under GSI*

In order to increase the probability of matching success, a set of registers are used in SIM. These registers are used to record all General State Information (GSI), e.g., IQs, OQs, request arbiter, grant arbiter, and so on. Subsequently, the gathered GSI is fed back to IQ request arbiters, OQ grant arbiters, and matching transfer controllers to supply necessary references for their arbitration and decision. The IQs that have at least a packet to send and the OQs that have enough space to receive a packet are forming the eligible queues group. The arbiter randomly selects a ready IQ number, whose destined OQ number should be also in this group too, to issue its request.

- *Exhaustive Transmission*

Furthermore, the SIM supports exhaustive matching; an IQ is served continuously until the IQ becomes empty or the corresponding OQ becomes full. This scheme can provide high throughput both in uniform and non-uniform traffic. The exhaustive transmission also benefits from GSI. GSI helps the transmitting controller to determine the number of packets to move from an IQ to their destined OQ, by obtaining the number of packets in an IQ and the free space of the corresponding OQ. Using exhaustive matching more packets will be transferred in burst mode per matching so as to avoid the frequent matching operation.

IV. PERFORMANCE EVALUATION

To evaluate the proposed scheme we set up a system that consist of 4 multi-core clusters and the level-2 switch. Each

Figure 4. Execution time

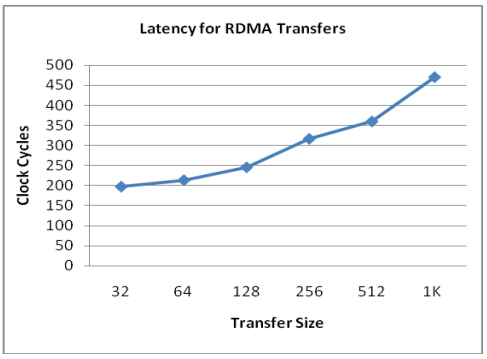

Figure 5. Latency vs. transfer size

cluster consists of a Xilinx ML505 board (Fig. 1) with an FPGA that incorporates 4 MicroBlaze processors supporting the explicit communication and synchronization described in Section 2. Each cluster is connected to the level-2 switch using 2 SATA interfaces configured at 3.125Gbps (thus providing 6.25Gbps per port). The level-2 switch is configured as a 4x4 store-and-forward switch with 2 planes (each port is 2 SATA interface; one for each plane) for a total aggregated throughput of 25 Gbps.

To evaluate the scalability of the proposed scheme we used the following benchmark. We measured the time to distribute evenly 64MB of data when the number of processors scales from 2 to 16 (i.e when the number of processors is 4, each processor sends 16MB to all the other processors, etc). Figure 4 shows the total execution time to distribute 64MB of data (compared with the execution using only two cores). As it is shown in the figure, the system scales efficiently as the number of cores increase from 2 to 16 (from 1 cluster to 4 clusters).

Figure 5 shows the latency for a data transfer between two clusters in clock cycles (at 75MHz). The latency between the processor and the SATA transceiver is around 15cc. The latency from the input to the output port of the level-2 switch is around 120 cc. This latency is due to the fact that the level-2 switch is configured as store-and-forward in order to increase the reliability of the switch (if a packet is lost, the switch can retransmit it). The rest of cycles (~65cc) are from the NI of the receiving cluster to the memory of the processor.

V. Conclusions

In this paper we have described and evaluated a multi-processor scalable cluster system with an efficient level-2 CIOQ switch that can scale efficiently up to 80 processors with up to 62.5Gbps aggregated bandwidth. The processors support both implicit and explicit communication and synchronization. The explicit communication can achieve high performance while the implicit communication increases the programming efficiency. Furthermore, the merged cache with the network interface (NI) and the hardware primitives for synchronization (RDMA, counters, queues) provide reduced interprocess communication and high bandwidth. The proposed system has been evaluated with 16 processors proving high bandwidth (25Gbps) and reduced interprocess communication latency between the clusters (200cc for 32 bytes).

Acknowledgment

This work is supported by the HiPEAC NoE and the SARC IP European union projects.

References

[1] Giorgos Passas, Manolis Katevenis, Dionisis Pnevmatikatos, "A 128x128x24Gb/s Crossbar, interconnecting 128 tiles in a single hop, and occupying less than 6% of their area," *The 4th ACM/IEEE International Symposium on Network-on-Chip (NOCS 2010)*, 2010.

[2] George Kalokerinos, et al, "FPGA implementation of a configurable cache/scratchpad memory with virtualized user-level RDMA capability," The 2009 IEEE International conference on embedded Computer Systems: Architecture, Modeling, and Simulation, July, 2009.

[3] Kavadias, S.,Katevenis, M., Zampetakis M., Nikolopoulos, D.,"On-Chip Communication and Synchronization Mechanisms with Cache-Integrated Network Interfaces", Proc. of IEEE International Conference on Computing Frontiers, May 2010

[4] Yoohwan Kim, H. Jonathan Chao, "Performance of exhaustive matching alogorithms for input-queued switches," *Conference on Communication (ICC 2003)*, vol. 3, pp. 1817-1822, 2003.

[5] Y. Li, S. Panwar, H. J. Chao, "The dual round-robin matching with exhaustive Service," *Proceedings of IEEE Workshop on High Performance Switching and Routing (HPSR 2002),* May 2002.

[6] Deng Pan, Yuanyuan Yang, "Pipelined two step iterative matching algorithms for CIOQ crossbar switches," *ANCS 2005*, pp. 41-50, October 2005.

[7] Jonathan Chao, "Saturn: a terabit packet switch using dual round-robin," *IEEE Communication*, pp. 78-84, December, 2000.

[8] Cyriel Johan Agnes Minkenberg, "On packet switch design," Ph.D. dissertation, Eindhoven University of Technology, 2001.

[9] Nick McKeown, "The *i*SLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol.7,no.2,pp.188-201, 1999.

[10] T. E. Anderson, S. S. Owicki, J. B. Saxie, C. P. Thacker, "High speed switch scheduling for local area networks," *ACM Trans. Computer Systems*, vol. 11, no. 4, pp. 319-352, Nov. 1993.

[11] Devavrat Shah, "Maximal matching scheduling is good enough," *IEEE GLOBECOM' 03*, vol. 6, pp. 3009-3013, 2003.