One-Way Shared Memory

Martin Schoeberl

Department of Applied Mathematics and Computer Science
Technical University of Denmark
Email: masca@dtu.dk

***Abstract*—Standard multicore processors use the shared main memory via the on-chip caches for communication between cores. However, this form of communication has two limitations: (1) it is hardly time-predictable and therefore not a good solution for real-time systems and (2) this single shared memory is a bottleneck in the system.**

This paper presents a communication architecture for time-predictable multicore systems where core-local memories are distributed on the chip. A network-on-chip constantly copies data from a sender core-local memory to a receiver core-local memory. As this copying is performed in one direction we call this architecture a one-way shared memory.

With the use of time-division multiplexing for the memory accesses and the network-on-chip routers we achieve a time-predictable solution where the communication latency and bandwidth can be bounded. An example architecture for a 3x3 core processor and 32-bit wide links and memory ports provides a cumulative bandwidth of 29 bytes per clock cycle. Furthermore, the evaluation shows that this architecture, due to its simplicity, is small compared to other network-on-chip solutions.

I. Introduction

When using multicore processors in real-time systems the communication between threads, which are executing on different cores, needs to be time-predictable. Current multicore architectures support data structures allocated in shared main memory, protected by locks, for communication between cores. In practice this communication is performed via cache coherence protocols between first level caches, backed up by second and third level caches. This avoids going off-chip for most communication. However, this communication via cache coherence protocols is hardly time predictable.

For real-time systems, we need to bound execution time and communication time between threads. Therefore, we need to provide a solution where communication is more explicit. Message passing between core-local memories supported by a time-predictable network-on-chip (NoC) is one option for time-predictable communication [3], [5].

This paper presents a time-predictable architecture for on-chip communication between processing cores. The architecture consists of core-local, distributed, on-chip memories supported by a simple and time-predictable NoC. The NoC copies data from a region in the sender's core-local memory to the receiver's core-local memory. This is done continuously, which means that data is continually updated. This update is performed in one direction only, therefore we call it a one-way shared memory.

The main architectural features of the one-way shared memory are:

- Distributed core-local memories that exchange data via a time-predictable NoC in an autonomous way to provide a form of shared memory for multicore processors.
- The NoC uses time-division multiplexing (TDM) to enable worst case analysis of the network bandwidth and latency. Implementation of a NoC with TDM has a very low cost: there is no need for dynamic arbitration, buffering, or flow control.
- The proposed NoC stores the schedule in the router. Therefore, no header with routing information needs to be moved between routers.
- The granularity of a NoC packet is a single word. When no routing information needs to be moved along the packet it is efficient to use single word packets, which reduce the latency of short messages.
- The resulting router is so simple, just a multiplexer driven by a static table, that only a single pipeline register between routers is required. Therefore, the NoC implements single cycle hops between routers.
- The shared memory is connected via a counter, serving as the "network interface", to the NoC. This interface is very low cost and supports constant copy of data between processor cores.
- The simplicity of the whole one-way shared memory results in a very low resource usage and therefore in low power consumption.

Figure 1 shows the multicore processor, such as T-CREST [10], that uses the one-way shared memory. Additionally to the on-chip memory the cores are connected to a shared, external memory with a memory arbiter and a memory controller to support larger programs and larger data structures that would not fit into on-chip memories. Our general aim for this work is to build time-predictable architectures [9] to support real-time systems.

This work builds on top of the S4NOC project [11] and the scheduler for that NoC [2]. The rest of the paper is structured as follows: the paper is organized in 5 sections: The following section presents related work. Section III presents the design and architecture of the one-way shared memory. Section IV evaluates the design with respect to resource consumption and worst-case communication latency. Section V concludes.

978-3-9819263-0-9/DATE18/©2018 EDAA

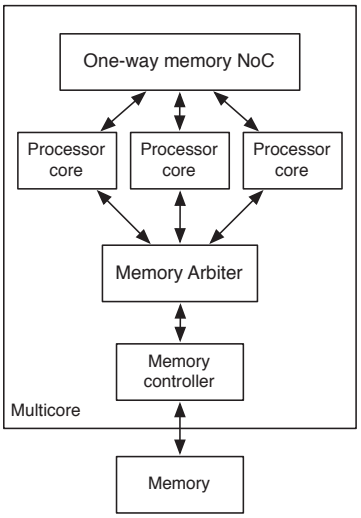

Fig. 1. The multicore architecture with several processor cores connected to: (1) the one-way memory NoC and (2) to an arbiter for the shared, external memory

II. Related Work

For time-predictable on-chip communication a NoC with TDM arbitration allows bounding the communication delay. Æthereal [3] is such a NoC that uses TDM where slots are reserved to allow a block of data to pass through the NoC router without waiting or blocking traffic. We are in line with the TDM approach of Æthereal, but provide a simpler network interface to the NoC.

Minimalistic NoCs are not a major topic in research and only a few projects exist. The paternoster NoC [8] aims for a simple design. It avoids flow control and complexity in the routers by restricting a packet to single standalone flits. Our NoC is a similar architecture, and uses just single word packets. However, we use statically scheduled TDM based arbitration to bound the maximum latency for packets and avoiding any buffering in the routers. Furthermore, we use a bidirectional torus instead of a unidirectional torus to achieve shorter routes.

The A Real-Time Capable Many-Core Model proposes many cores with a static switched NoC with TDM based arbitration [6]. The Reduced Complexity Many-Core architecture [7] proposes to avoid shared memory at all and to support timing analysis by using a fine-grain message passing NoC. We agree on this approach to prefer on-chip communication between local memories over shared memory communication and to use TDM based arbitration for the NoC.

Similar to the paternoster NoC, the design of the Hoplite architecture [4] uses routers without buffers, an unidirectional torus, and single flit packages that include the destination address. However, on an arbitration conflict, Hoplite uses deflection as a resolution mechanism. This design results in a very small hardware, not so different from our design, but cannot give any real-time guarantees.

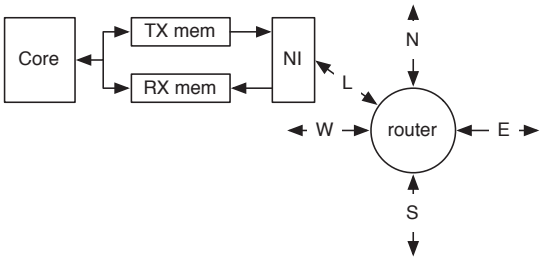

Fig. 2. One processing node consisting of a core, two local memories, a network interface, and a NoC router

Another NoC, with similar aims as our design, is the Argo NoC [5]. The Argo NoC aims for time-predictability. It uses TDM based arbitration in the routers, like our used NoC. It also uses TDM based DMA transfer of data from the local memory into the NoC.

The main difference of our work to all above described NoCs is the direct connection of a local memory to the network, which avoids any complex network interface, resulting in lower resource usage and better time predictability.

III. One-Way Shared Memory

We propose and present the design of a new form of shared memory: a one-way shared memory. The one-way shared memory provides automatic copy of data between core-local memories in a multicore processor. As this transfer is only in one direction, we call it one-way shared memory. The architecture provides one sharing path between each pair of cores.

A. Distributed Memory

The central components of the one-way shared memory are distributed dual-ported on-chip memories. These memories are connected with one port to the processor cores and with the other port to the NoC. Data is moved by the NoC (and NI) by continuously reading from a core-local memory, moving it through a pipelined structure of NoC routers, and writing it to the destination core-local memory. This transfer happens continuously without the need to setup any message passing transfer.

The NoC can read *and* write one word every clock cycle. Therefore, the core-local memory is split into two memories: one for the transmit (TX) channels and one for the receive (RX) channels. Figure 2 shows one processing node where a processor core is connected to the TX and RX memories. The second port of the dual-ported memories are connected to the NI. The NI is a simple address generator for the local memories accesses. That NI is connected to a router of the NoC through the local (L) port. The routers four additional ports (north (N), east (E), south (S), and west (W)) are connected to the neighbor nodes to build a bidirectional thorus.

Figure 3 shows the buffer configuration of two local memories for a 2x2 configuration: the TX memory of core 0 and the RX memory of core 2. Each memory is divided into blocks, which represent the communication channels with other cores.

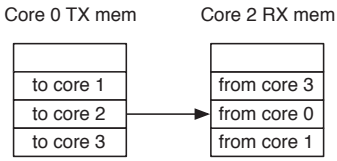

Fig. 3. Transmit (TX) memory of core 0 and receive (RX) memory of core 2

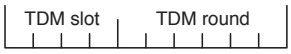

Fig. 4. Memory address generation with a TDM slot counter and a TDM round counter.

TABLE I
RESOURCE CONSUMPTIONS OF DIFFERENT COMPONENTS OF THE ONE-WAY SHARED MEMORY.

Component	LE	Register	Mem
2x2 NoC	415	392	-
3x3 NoC	2324	1448	-
4x4 NoC	5159	2568	-
2x2 one-way memory	958	799	-
3x3 one-way memory	3453	2374	-
4x4 one-way memory	7101	4241	-
Argo router	938	565	-
Argo single node	1775	933	1.3 KB
Argo 3x3 NoC	15146	8342	12.1 KB

The order of the blocks is a result of the static schedule. We can see that core 0 transmits in the first TDM slot to core 3 and core 2 receives in the first TDM slot from core 1. The transfer of data from core 0 to core 2 is scheduled in the second TDM slot.

B. Time-division Multiplexed NoC

The NoC is based on pipelined multiplexers [11]. Each router has five ports: north, east, south, west, and local. Local is connected to the core-local memory, the other ports are used to build a network of routers. We use a network organized as bidirectional torus. Each port consists of a 4:1 multiplexer for the inputs and a single pipeline register at the output. We use single word packets and the routing information is stored in the router.

We use TDM to arbitrate the access to the shared resource (pipeline register and link). That means time is divided into slots, each is a single clock cycle. A TDM schedule is n slots long, and is repeated forever. We call these n slots a TDM round.

The TDM schedule is computed in advance and built into the router hardware as ROM table. For the general case, we support an all-to-all schedule, which means that each core has one communication channel to each other core. In each TDM round one packet (word) is transmitted from each core to every other core. A TDM schedule to support all-to-all communication is very short, as a bi-torus NoC supports high overall bandwidth. E.g., for a 3x3 NoC it is 10 clock cycles, and for a 5x5 NoC 28 clock cycles [11].

C. Address Generation

The local port of the NoC is connected to one side of the core-local memory. Each clock cycle one word is written into the core-local RX memory and one word is read from the TX memory and injected into the NoC. This is done unconditionally. The address for the memory is generated by a combination of the TDM slot counter and a TDM round counter. The TDM slot counter is incremented every clock cycle and reset at the end of the TDM round. The TDM round counter is incremented each round.

Figure 4 shows an example of the address generation for the local memory. The lower bits are counting in TDM rounds, the upper bits count in TDM slots. In this example with 10 address bits the whole memory is 1024 32-bit words or 4 KB large. Each shared memory block is 64 words and the example supports up to 16 TDM slots, which is enough for a 9-core system.

In each cycle a word from a different core is read or written. This address generation is the same for the read address of the sender and the write address of the receiver. This means that it takes several TDM rounds to fill one memory block, for the above example 64 TDM rounds. We can call this time the hyper period. After one a hyper period we know that all memories in all cores are updated.

IV. EVALUATION

We have implemented the one-way shared memory in Chisel, a new hardware construction language [1]. We evaluate the one-way memory architecture in two forms: (1) exploring the functionality with Chisel based test benches and (2) with synthesize results for an FPGA.

We built unit tests for individual components, the network, and the whole system. We use the Chisel generated C++ hardware simulation and Scala based test benches for functional testing.

A. Resource Consumption

Table I shows resource consumptions of different components of the one-way shared memory. We use a 32-bit datapath for all experiments. The resource consumption is given in logic elements (LE) containing a 4-bit lookup table, registers (Reg.), and memory consumption in bits. We show only memory consumptions if it is used for tables. The size for the core-local memory is configurable and therefore not shown in the table. The synthesize results are for the Intel/Altera Cyclone IV EP4CE115 FPGA, which is used on the DE2-115 board. All NoC designs can be clocked above 200 MHz, which higher than a RISC processor can be clocked in that Cyclone IV FPGA (in the range of 100 MHz).

The first three row show the resource consumption of different NoCs with an all-to-all schedule. A 9-core configuration (3x3) consumes about 260 LEs and 160 registers per router. The 16-core configuration uses more often all four links and results in 320 LEs and 160 registers per router.

The next three rows show complete one-way shared memory architectures in different configurations. Those designs include the NoC, the NIs, and the core-local memories. A complete node (without a processor) consumes between 240 and 440 LEs and between 200 and 270 registers per node. This resource consumption is in the range of about 10 % of a typical RISC style processor core.

A TDM based NoC, also optimized for low resource consumption, the Argo NoC [5], is available in open source. Therefore, we can compare our one-way shared memory with the Argo NoC. We have synthesized a 3x3 core configuration for the same FPGA. The Argo NoC contains 3 stage pipelined routers and a network interfaces with TDM driven reading of data from a local memory. The last three rows give numbers for the Argo NoC. One 5-port router needs about 900 LEs and about 560 registers. A complete node with the network interface consumes 1780 LEs, 930 registers and 11000 bits of memory (for the schedule table). However, the Argo NoC, and the NI provide more functionality than our one-way shared memory, at a cost of almost 4 times the resources.

B. Bandwidth and Communication Latencies

Our aim for the proposed one-way shared memory architecture is to support real-time systems on a multicore processor. Therefore, we aim to bound the worst-case latency for exchanging messages. Using TDM scheduling on the NoC simplifies the worst-case analysis. Within each TDM round we can send one word from each core to every other core. The lengths of the TDM schedules (r), including the last hop to the local port, are as follows: 5 clock cycles for a 2x2 system, 10 clock cycles for a 3x3 system, and 19 clock cycles for a 4x4 system.

Those TDM schedules result in the bandwidth for all communication channels. For example, in a 3x3 system we can provide channels with a bandwidth of four bytes per 10 clock cycles. However, this is the bandwidth of a single channel. The whole system contains one channel from each of the 9 cores to each of the other 8 cores, therefore 72 channels. The cumulative bandwidth is then $4 \times 72/10 = 28.8$ bytes per clock cycle. The worst-case latency l of a single word transfer is: $l = r - 1 + h$ with TDM round r and h hops through the NoC.

C. Source Access

We strongly believe that reproducibility of results is a very important aspect in research. Providing the implementation of an architecture in open source simplifies or even enables the reproduction of the presented results. Furthermore, open source simplifies future research on top of the presented ideas. The source for the architecture presented in this paper is available at: https://github.com/schoeberl/one-way-shared-memory.

V. Conclusion

This paper presents a time-predictable and efficient on-chip communication architecture for multicore processors: the one-way shared memory. The one-way shared memory consists of distributed on-chip memory blocks that are connected to the processor cores and to a time-predictable network-on-chip. The network-on-chip continuously copies data from source core-local memories to destination core-local memories. As a copy of a single block of data is performed only in one direction, we call it one-way-shared memory.

We showed in example configurations the low resource consumption and the achievable and guaranteed bandwidth. For example, a 3x3 architecture with 32-bit wide memories supports a bandwidth of 29 bytes per clock cycle.

Acknowledgment

The work presented in this paper was partially funded by the Danish Council for Independent Research | Technology and Production Sciences under the project PREDICT (http://predict.compute.dtu.dk/), contract no. 4184-00127A.

References

[1] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avizienis, John Wawrzynek, and Krste Asanovic. Chisel: constructing hardware in a scala embedded language. In Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun, editors, *The 49th Annual Design Automation Conference (DAC 2012)*, pages 1216–1225, San Francisco, CA, USA, June 2012. ACM.

[2] Florian Brandner and Martin Schoeberl. Static routing in symmetric real-time network-on-chips. In *Proceedings of the 20th International Conference on Real-Time and Network Systems (RTNS 2012)*, pages 61–70, Pont a Mousson, France, November 2012.

[3] Kees Goossens and Andreas Hansson. The AEthereal network on chip after ten years: Goals, evolution, lessons, and future. In *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC 2010)*, pages 306 –311, 2010.

[4] Nachiket Kapre and Jan Gray. Hoplite: Building austere overlay nocs for fpgas. In *25th International Conference on Field Programmable Logic and Applications (FPL 2015)*, pages 1–8, Sept 2015.

[5] Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Sørensen, Christian T. Müller, Kees Goossens, and Jens Sparsø. Argo: A real-time network-on-chip architecture with an efficient GALS implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24:479–492, 2016.

[6] Stefan Metzlaff, Jörg Mische, and Theo Ungerer. A real-time capable many-core model. In *Proceedings of 32nd IEEE Real-Time Systems Symposium: Work-in-Progress Session*, 2011.

[7] Jörg Mische, Martin Frieb, Alexander Stegmeier, and Theo Ungerer. Reduced complexity many-core: Timing predictability due to message-passing. In Jens Knoop, Wolfgang Karl, Martin Schulz, Koji Inoue, and Thilo Pionteck, editors, *Architecture of Computing Systems - ARCS 2017: 30th International Conference, Vienna, Austria, April 3–6, 2017, Proceedings*, pages 139–151, Cham, 2017. Springer International Publishing.

[8] Jörg Mische and Theo Ungerer. Low power flitwise routing in an unidirectional torus with minimal buffering. In *Proceedings of the Fifth International Workshop on Network on Chip Architectures*, NoCArc '12, pages 63–68, New York, NY, USA, 2012. ACM.

[9] Martin Schoeberl. Time-predictable computer architecture. *EURASIP Journal on Embedded Systems*, vol. 2009, Article ID 758480:17 pages, 2009.

[10] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, André Rocha, Cláudio Silva, Jens Sparsø, and Alessandro Tocchi. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015.

[11] Martin Schoeberl, Florian Brandner, Jens Sparsø, and Evangelia Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proceedings of the 6th International Symposium on Networks-on-Chip (NOCS)*, pages 152–160, Lyngby, Denmark, May 2012. IEEE.