Evaluation of a Connectionless NoC for a Real-Time Distributed Shared Memory Many-Core System

Jochem H. Rutgers, Marco J.G. Bekooij, Gerard J.M. Smit
University of Twente, Department of EEMCS
P.O. Box 217, 7500 AE Enschede, The Netherlands
j.h.rutgers@utwente.nl

***Abstract*—Real-time embedded systems like smartphones tend to comprise an ever increasing number of processing cores. For scalability and the need for guaranteed performance, the use of a connection-oriented network-on-chip (NoC) is advocated. Furthermore, a distributed shared memory architecture is preferred as it simplifies software development for a multi-core system.**

In this paper, experimental evidence is provided, showing that replacing a connection-oriented NoC by a connectionless one in a distributed shared memory system reduces the hardware costs and improves the performance. We observed that our FPGA could only support an 8-core system with a connection-oriented NoC. We exchanged the NoC with our tree-shaped, connectionless network and a ring, allowing a 32-core system in the same FPGA, mainly because of a reduced number of physical connections. Although the analytical worst-case performance slightly decreased, measurements show that the latency of latency-critical memory reads was reduced by 52% on average.

I. Introduction

The current trend is that an ever increasing number of processing cores is integrated on a single chip for real-time embedded devices, such as smartphones and car-entertainment systems. Usually, expensive, off-chip memory communication is avoided by using distributed and non-uniform memory, such as caches and scratch-pad memories.

Because a traditional bus is unsuitable as infrastructure for such a many-core architecture, networks-on-chip (NoCs) have been developed. In particular, to give real-time guarantees, the use of a connection-oriented, guaranteed-service (GS) NoC is proposed [1–3]. This allows performance analysis of applications in isolation because communication of one application can only have a bounded influence on other running applications. Nevertheless, software development for such distributed-memory, many-core system is hard [4].

To reduce the programming effort, the hardware platform should preferably support a shared-memory programming model [5, 6]. Other programming models, such as streaming and message passing, can be emulated on such a system by means of a software middleware layer [5–8]. A connection-oriented NoC can be used in a distributed shared memory (DSM) architecture, but requires connections between every processor and (local) memory. This is expensive because a hardware cost is associated with every connection. A connectionless NoC would be less expensive, but the performance per connection is uncertain.

In this paper, experimental evidence is provided that confirms that substitution of a connection-oriented NoC by a connectionless one in a real-time DSM system can reduce hardware costs significantly. Furthermore, it can improve the processor utilization, but *does* compromise the analytically computed worst-case behavior. However, an increase in the uncertainty introduced by the connectionless interconnect is *not* confirmed by the experimental results.

We designed and built such a system with a connection-oriented NoC, tailored to streaming applications (Section II). It turned out that the size of a Virtex-6 LX240-T FPGA only allowed having 8 MicroBlaze cores (Section III). We replaced the connection-oriented NoC by a tree-shaped, connectionless network and a ring (Section IV). The new interconnect is smaller, and as a result, the same FPGA can contain 32 cores. Additionally, measurements show improved performance. Although the analytically calculated worst-case behavior is slightly worsened, it is bounded (Section V). This makes the use of a connectionless NoC viable.

A. Related Work

A connectionless NoC is also used in Intel's 48-core SCC, which implements both shared memory and message passing [9]. Another example of a DSM system is the Tilera TILE-Gx, which incorporates 100 cores and multiple NoCs for different data streams [10, 11]. For neither system, the NoC's influence on the real-time behavior has been analyzed.

MAGALI is a heterogeneous, reconfigurable system with 22 cores and a NoC for baseband processing with hard real-time properties [12]. Although the system is reconfigurable, it is optimized for a limited class of applications, where we target a more general-purpose system.

The CoMPSoC multiprocessor system [13] comprises three VLIW cores, a GS, connection-oriented NoC and a shared memory. A key property of this system is that applications cannot affect each others temporal behaviour. In contrast, in this paper we target real-time systems for which it is sufficient that interference is bounded by using starvation-free schedulers.

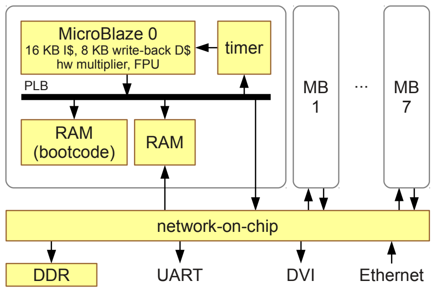


Figure 1. Structure of the system with eight identical MicroBlaze tiles, NOC, memory, and peripherals; arrows indicate master/slave relation

Table I
FPGA RESOURCE USAGE OF SYSTEM WITH ÆTHEREAL

	LUTS		FFS		BRAMS
master MicroBlaze	2,664	(3.2%)	2,239	(2.6%)	8
master tile[a]	2,372	(2.9%)	2,385	(2.8%)	9
7× slave MicroBlaze	2,461	(3.0%)	2,003	(2.3%)	8
7× slave tile[a]	1,184	(1.5%)	714	(0.8%)	5
interconnect	46,535	(57.0%)	56,274	(65.8%)	0
peripherals	4,542	(5.6%)	5,594	(6.5%)	10
total	81,628	(100.0%)	85,511	(100.0%)	118

[a] The tile includes a local memory, a timer, PLB and bridges

II. System Architecture

The characteristics of the system we target as mentioned in Section I, are: 1) a scalable distributed shared memory, many-core system; 2) streaming, *firm* real-time applications, where deadline misses are highly undesirable, but not catastrophic; and 3) the set of applications to be run at the platform—and thus the communication pattern—is unknown at design time, so the architecture must be flexible. We implemented a tool flow that generates such system for FPGA, with a configurable number of processing tiles, either a connection-oriented or a connectionless NoC, and one Xilinx DDR3 memory controller. The structure of an 8-core system is depicted in Figure 1. The processing tile comprises a MicroBlaze (not cache coherent) connected to a local bus with an SRAM and a timer. The MicroBlazes have a global address map and can access the SRAMs of other MicroBlazes via the NoC.

All MicroBlazes can execute code from main memory. Every core runs a small, custom, multi-threaded kernel using using Time-Division Multiplexing (TDM) scheduling and supports the newlib C library and implements the Pthread standard. Kernels communicate by message-passing over the direct connections between processors.

On top of this kernel, any application can be run. To our knowledge, there is no streaming or multi-core real-time application benchmark set. Hence, we use applications from the SPLASH-2 benchmark set [14] for our experiments; they are easy to port, and give a decent parallel workload. Because the lack of cache coherency, all shared application data is put in an uncached memory region; the caches only cover code, kernel data structures and the all process's stacks.

With this setup, we experiment with different NoCs.

III. Observations Connection-Oriented NoC

The first system that is evaluated is an 8-core system as of Section II with the connection-oriented Æthereal NoC [3]. In this NoC, the connections can be configured at run-time. Because we have a DSM system, the NoC should allow connections between any two processor tiles. Since it is not feasible to recompute and reconfigure the network configuration at run-time when a connection is (to be) used, a single general-purpose configuration is required. Therefore, we attempted to configure Æthereal to be fully connected.

Surprisingly, synthesis shows that the 8-core design does not fit in a Virtex-6 LX240-T FPGA because the fully connected interconnect is too big. As a—naive and non-generic, but simple—solution, Æthereal is configured such that every MicroBlaze can communicate with the DDR, one MicroBlaze that manages startup, one peripheral, and both neighbors. Æthereal is configured with very low bandwidth requirements to maximize configuration freedom[1] and the buffer sizes are set to contain one burst of the MicroBlaze's cache. This configuration is small enough to fit in the FPGA, and will be used as reference design in this paper.

A. FPGA Synthesis Results

The synthesis results of the 8-core reference design for a Xilinx Virtex-6 LX240-T at 100 MHz is shown in Table I. The table shows that the master MicroBlaze is slightly bigger than the slaves, which is caused by additional debug support. The peripherals include the memory controller and UART.

Still, the interconnect is the biggest part of the system. The NoC contains 1.5 times more look-up tables (LUTs) and 2.4 times more flip-flops (FFs) than all MicroBlaze tiles together. Two reasons for that result are that MicroBlazes are small and Æthereal does not map to an FPGA well.

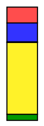
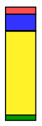
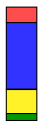
However, there is a more fundamental problem: every connection in a connection-oriented NoC has associated hardware costs. In case of Æthereal, most of the area is used by buffers. Because we need a fully-connected interconnect, a connection-oriented network becomes expensive since a quadratic number of buffers is required.

B. MicroBlaze Utilization

To analyze the performance of the platform, we ported the SPLASH-2 `raytrace`, `volrend`, and `radiosity` applications. Table II shows the distribution of cycles during the main, parallel application loop of each of the applications. In the table, *execution* indicates the time that the core executes instructions, or stalls on uncached data reads and data cache

[1]In fact, when realistic bandwidth requirements are set, a suitable configuration cannot be found. Forcing a different internal network structure or choosing the number of input/output ports differently does not help.

Table II
MicroBlaze utilization

event	raytrace		volrend		radiosity	
🟥 ICache mis	14.0%		6.3%		13.7%	
🟦 read data	16.7%		14.4%		58.7%	
⬛ write data	0.5%		0.6%		0.3%	
🟨 execution	63.4%		73.9%		21.2%	
🟩 other	5.3%		4.7%		6.1%	

misses (*read data*); uncached writes and data cache flushes (*write data*); and instruction cache misses (*ICache mis*). *Other* includes overlapping and indecisive events.

The performance is limited by the high memory read latency: a read takes 77 clock cycles on average, where 15 cycles are spent in the DDR controller and the rest in the NoC. Traversing the NoC is expensive, because: 1) one memory request packet waits for multiple TDM slots (which are non-contiguous) in the routers, even when the NoC is idle; and 2) the response has to wait for its slot too, because the arbitration of the request and response packets are unrelated.

We observe that we have two types of communication channels: *latency-tolerant* interprocess channels (typically FIFO communication, with bandwidth guarantees); and *latency-critical* channels (typically for caches). In the latter type of channel, the performance degrades immediately with a higher latency, where in the former, sensitivity for latency depends on the application. The profiling data shows that Æthereal does not perform well for latency-critical traffic.

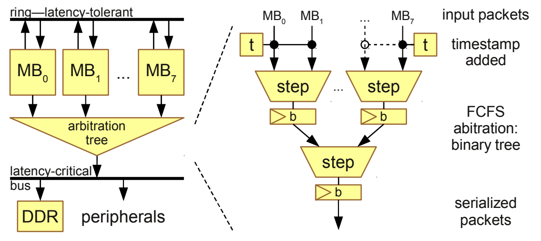
IV. Improvements by Connectionless NoC

The previous section identifies two problems: super-linear scaling of hardware resources and high latency for memory reads. We designed a connectionless NoC, of which Figure 2 shows its structure. Latency-tolerant traffic utilizes the interconnect at the top of the figure. We chose a ring for it, because of its simplicity, but since the applications hardly depend on it, a thorough discussion is omitted.

At the bottom, the latency-critical part connects the memory and peripherals. It must adhere to the following requirements: 1) starvation-free scheduling; 2) work conserving to optimize for latency; 3) scale linearly in hardware costs to the number of cores; and 4) pipelined and decentralized arbitration to avoid long wires for high performance.

A. Proposed NoC: Arbitration Tree

Figure 2b shows the structure of this new interconnect, having arbitration of 8 cores to a memory controller and peripherals. Every read and write request of a processor is packetized, containing one command, one address, and multiple data flits. A packet gets a timestamp and processor ID upon injection. The timestamp can be generated locally to every core, as long as they are (pseudo-)synchronized, i.e. during reset. When these timestamp generators are getting



(a) System structure (b) Arbitration tree structure

Figure 2. Structure of the system with replaced interconnect

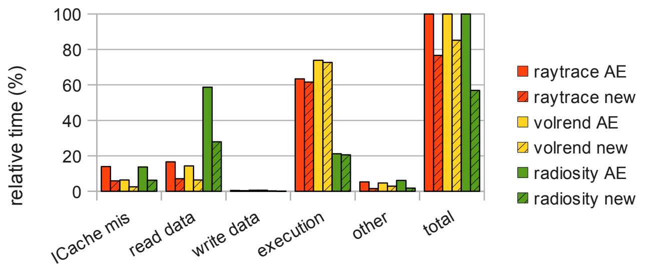


Figure 3. Comparison of processor utilization of Æthereal (measurements of Table II are labeled AE) and new arbitration tree

out of sync (by drifting clocks, for example), the First-Come-First-Served (FCFS)-property cannot be guaranteed.

Next, the packets are sent through a binary arbitration tree that multiplexes n processors to one bus master, where every *step* in the tree does local arbitration of two inputs. A step lets the packet with the lowest timestamp precede. Rearbitration only happens between packets. After every step, a small buffer can be placed for shorter wires or left out for lower latency. By means of back-pressure, requests can be stalled by subsequent steps and the bus slave. Because the network is a binary tree, the number of steps in the network equals $n - 1$. Hence, the hardware requirement scales linear to the number of processors.

Finally, the response will be sent back into a similar tree, but demultiplexes one to n without arbitration, based on the processor ID (not shown in the figure).

B. Evaluation

Synthesis of the arbitration tree in the 8-core system shows that the interconnect consumes 4,603 LUTs and 2,750 FFs. So, this system uses about half the resources of the one with Æthereal of Table I. With such resource usage, even a 32-core system fits in the FPGA.

On this 8-core platform, the same applications have been run as in Section III-B. Figure 3 compares the utilization of both experiments. As the chart shows, the total execution time is reduced for all applications, and the time the processor stalls at the instruction cache misses and reads from the memory is roughly halved. The read latency of the memory is reduced to 37 cycles under full load and 25 cycles when idle (where the memory controller still consumes 15 cycles).

Table III
MEASURED AVERAGE PACKET ISSUE INTERVALS (CLOCK CYCLES)

packet type	ℓ	`raytrace`	`volrend`	`radiosity`
read word[a]	2	47.6	60.4	10.4
read burst[b]	2	51.4	100.3	40.7
write word[c]	3	6197.0	4187.5	763.2
write burst[d]	10	7875.2	10049.1	18078.9

[a] Causes: uncached data read
[b] Causes: instruction cache mis; data cache mis
[c] Causes: uncached data write; data cache word flush
[d] Causes: data cache line flush

V. Bounded Temporal Behavior

FCFS, which is used in our new network, in general is not known to be fair and can be out-performed by other schedulers [15]. However, our interconnect has been designed such that FCFS can be used in a predictable system, which is proven in this section.

We define the *service time* $S(p)$ of a packet p with length $\ell(p)$, which is the time a packet resides in the network. This is the time packet p arrives at a leaf of the tree and gets its timestamp, till the last flit leaves the root and therefore is serviced. An initiator can only inject a new packet when the last flit of the previous one is injected in the tree.

A step in the arbitration tree can only process one flit per time unit, e.g. clock cycle. Traversing a buffer always takes at least one time unit, even if the buffer is empty. Then, the best-case service time S_{bc} of a packet p is $S_{bc}(p) = \ell(p) + \log_2 n$.

The maximum buffering in the tree is $(n-1)b$, where b denotes the depth of the buffer after a step. Therefore, the worst-case service time of p is that p must wait for all flits in the tree and the largest packet possible just being injected by all other initiators: $S_{wc}(p) = S_{bc}(p) + (n-1)b + (n-1)\max_{p' \in P} \ell(p')$, where P denotes the set of all possible packet types. Hence, the service time is bounded and the arbitration is starvation free.

In the system of Section IV, where $b = 2$, $S_{wc}(\text{read}) = 89$ cycles, where $S_{bc}(\text{read}) = 5$. In contrast, the worst-case service time in the system with Æthereal (as of Section III) is $S_{wc} = 84$ and the measured average case is $S_{ac} = 30.66$. S_{wc} for our interconnect is high, because a packet must wait for the largest packets to complete first. However, the largest packets are burst writes, which are rare; the measured average interval between packets are listed in Table III.

Although a single packet could stall according to S_{wc}, it is not possible that all packets must wait this long, because there are not enough interfering packets that can be waited on. For example, within a period τ, `raytrace` issues $q = \frac{1}{8}(\frac{\tau}{47.6} + \frac{\tau}{51.4})$ read packets. These packets could interfere with at most $\mathcal{I} = \frac{7}{8}(\frac{2\tau}{47.6} + \frac{2\tau}{51.4} + \frac{3\tau}{6197.0} + \frac{10\tau}{7875.2})$ other packet flits. So, on average every read packet waits at most for $\frac{\mathcal{I}}{q} = 14.3$ flits c.q. cycles, which is closer to the measurement.

VI. Conclusion

In this paper, a Virtex-6 FPGA implementation of an 8-core distributed shared memory system with a connection-oriented NoC is compared to one with a connectionless NoC. This comparison provides evidence that replacing a connection-oriented NoC by a connectionless one can significantly reduce hardware costs and can result in an improved average performance.

The higher cost of the connection-oriented NoC is mainly caused by super-linear scaling of the number of buffers with respect to the number of cores, where our network scales linearly to the number of cores. Experiments using SPLASH-2 applications show that the average read latency reduced from 77 to 37 clock cycles and the core utilization increased by 38%. Although the worst-case latency from core to memory increased from 84 to 89 cycles, the worst-case reduction in performance per core is more than compensated by having more cores for the same hardware costs.

The experimental evidence provided in this paper indicates that it can be attractive to use a connectless NoC in real-time multiprocessor systems with a distributed memory architecture.

References

[1] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv.*, vol. 38, no. 1, p. 1, 2006.

[2] G. De Micheli, C. Seiculescu, S. Murali *et al.*, "Networks on Chips: from Research to Products," in *Proceedings of the 47th Design Automation Conference (DAC 2010)*, vol. 1, 2010, pp. 300–305.

[3] K. Goossens and A. Hansson, "The Aethereal Network on Chip after Ten Years: Goals, Evolution, Lessons, and Future," in *Proc. Design Automation Conference (DAC)*, Jun. 2010.

[4] H. Sutter and J. Larus, "Software and the concurrency revolution," *Queue*, vol. 3, no. 7, pp. 54–62, Sep. 2005.

[5] L. Dagum and R. Menon, "OpenMP: An Industry-Standard API for Shared-Memory Programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[6] D. Kranz, K. Johnson, A. Agarwal *et al.*, "Integrating message-passing and shared-memory: early experience," *ACM SIGPLAN Notices*, vol. 28, no. 7, p. 63, 1993.

[7] P. van der Wolf, E. de Kock, T. Henriksson *et al.*, "Design and programming of embedded multiprocessors: an interface-centric approach," in *Proceedings of CODES+ISSS '04*. ACM, 2004, pp. 206–217.

[8] D. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, 1998, ISBN 1558603433.

[9] J. Howard, S. Dighe, Y. Hoskote *et al.*, "A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS," in *ISSCC, 2010 IEEE International*, 2010, pp. 108–109.

[10] Tilera TILE-Gx, http://www.tilera.com/products/processors/TILE-Gx_Family.

[11] A. Agarwal, "The Tile Processor: A 64-core Multicore for Embedded Processing," in *Proceedings of HPEC Workshop*, 2007.

[12] F. Clermidy, C. Bernard, R. Lemaire *et al.*, "A 477mW NoC-based digital baseband for MIMO 4G SDR," in *ISSCC, 2010 IEEE International*, 2010.

[13] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, "CoMPSoC: A template for composable and predictable multi-processor system on chips," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, 2009.

[14] S. Woo, M. Ohara, E. Torrie *et al.*, "The SPLASH-2 programs: characterization and methodological considerations," in *Proceedings of 22nd Annual Int. Symp. on Comp. Arch.*, 1995, pp. 24–36.

[15] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks," in *Proceedings of the IEEE*, 1995, pp. 1374–1396.