A Link Arbitration Scheme for Quality of Service in a Latency-Optimized Network-on-Chip

Jonas Diemer and Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig, Germany
Email: {diemer,ernst}@ida.ing.tu-bs.de

***Abstract*—Networks-on-chip (NoC) for general-purpose multiprocessors require quality of service mechanisms to allow real-time streaming applications to be executed along with latency-sensitive general purpose processing tasks. In this paper, we propose a NoC link arbitration technique that supports bandwidth guarantees along with best effort latency optimizations. In contrast to many existing quality of service mechanisms, our technique prioritizes best effort over guaranteed bandwidth traffic for optimal latency. Distributed traffic shaping is used to offer bandwidth guarantees over previously reserved connections, which are established dynamically using control messages. Initial simulation results show that our arbitration scheme can provide tight bandwidth guarantees for streaming traffic under network overload conditions. At the same time, the average latency of best effort traffic is improved compared to a simple prioritization of streaming traffic.**

I. Introduction and Related Work

Driven by power efficiency and thermal constraints, mainstream general-purpose processors are integrating an increasing number of processing cores on a single die. These many-core architectures are facing challenges similar to embedded MPSoC architectures [1], [2], especially the on-chip communication complexity. Networks-on-chip (NoC) have been used as an efficient communication architecture in this domain for years and are now also being considered for general-purpose and high performance computing, e.g. [3], [4], [5], [6]. One of the challenges for these NoCs is the wide variety of traffic patterns and communication requirements which cannot be narrowed down at design time: traffic generated by processor cores running general-purpose applications is very latency-sensitive and bursty, but can be treated as best effort (BE) without timing guarantees. Streaming applications such as multimedia or augmented reality on the other hand require a guaranteed bandwidth (GB) with bounded latency.

Without quality of service (QoS) mechanisms supporting these traffic types in the network-on-chip, it is very difficult, if not impossible, to execute soft or firm real-time streaming applications in coexistence with other general-purpose applications. Most existing communication architectures for general-purpose systems are optimized for best effort latency and throughput, offering no guarantees to individual traffic streams. In contrast, communication architectures in embedded (real-time) streaming systems give strong guarantees, usually by connection-based preallocation of time slots (e.g. [7], [8]), by prioritization of virtual channels (e.g. [9], [10], [11]) or a combination of these [12]. While some approaches (e.g. [11]) offer the ability to separately control latency and bandwidth guarantees, all of the existing solutions have treat BE traffic, i.e. traffic for which no service requirements or resource bounds are known, as low priority traffic, which may only consume excess bandwidth. This results in higher latencies for the BE traffic, which is unfavorable in a general-purpose multiprocessor, where most of the best effort traffic is latency-sensitive.

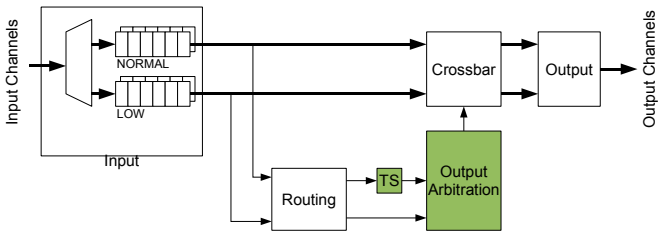

Fig. 1. Proposed router architecture including traffic shapers (TS).

Therefore, this paper presents a NoC architecture that is optimized for latency but allows bandwidth guarantees for specific traffic streams. The goal is to provide the best possible latency to the regular BE traffic while meeting the bandwidth guarantees for the streaming (GB) traffic. We achieve this by modifying the link arbitration to prioritize BE traffic, but limit its rate by means of traffic shapers to guarantee bandwidths for GB traffic. In this way, our approach is similar to the scheduling in several recent memory controllers with QoS support (e.g. [13], [14], [15]), which use prioritization to control latencies and install traffic shapers at the input queues to regulate bandwidth. As a consequence, these memory controllers allow bandwidth guarantees independent of priorities. While memory controllers regulate traffic at a single point, our scheme uses distributed regulators at the network routers. Coordination of these shapers is achieved by a configuration protocol, which allows dynamic adjustments to traffic shaper settings.

II. QoS-Aware Link Arbitration Scheme

Our baseline NoC is a packet switched mesh network with dimension ordered XY-routing for deterministic in-order packet delivery. The routers use input buffering and switching is non-blocking. Inter-tile packet flow control follows the

virtual cut-through scheme. Buffers and links are arbitrated at packet granularity, i.e. packets cannot be preempted during transmission.

A. Priorities and Traffic Shaping

As discussed above, we assume two different traffic classes. The "regular" traffic generated by processors and caches is *best effort (BE)* and is latency-sensitive. Traffic resulting from streaming applications requires a *bandwidth guarantee* (GB), but can tolerate (bounded) latencies. Hence, we define two priorities: NORMAL for BE traffic and LOW for GB traffic. Separate buffers (virtual channels) are assigned to priorities. Link arbitration between traffic is strictly prioritized. NORMAL priority packets will always be handled before LOW priority ones. Scheduling within the same priority is round-robin.

Traffic shapers limit the rate of high priority traffic to leave a guaranteed bandwidth to the low priority traffic. Traffic shaping usually refers to regulating the rate of packet injection, which implies that the shapers are placed at the injection links. However, to regulate the best effort "background" traffic at individual routes regardless of its source, the shapers are placed within each router, more specifically after the routing and before the arbitration stage. Here we can control the rate of NORMAL priority traffic for each direction (north, east, south, west, and two clients), allowing us to reserve distinct GB paths. Fig. 1 depicts our proposed router architecture, with our variations from a standard router shaded.

We propose to use token bucket shapers, which implement a bucket for up to b tokens, to which c tokens are added after every T clock cycles. As long as the bucket is full, additional tokens are discarded. Whenever a packet is to be sent over an output link, tokens are taken out of the corresponding bucket. Each token is worth one cycle on the link. If there are not enough tokens in the bucket to transfer a complete packet, it is stalled until enough new tokens have accumulated. This implies that the bucket must be large enough for the longest packet. The maximum average rate (as a fraction of the link capacity) for best effort traffic is $r^{BE}_{max} = \frac{c}{T}$, which results in a minimum guaranteed rate for GB traffic of $r^{GB}_{min} = 1 - r^{BE}_{max}$.

The longest interval t_{block} that a single GB packet is blocked can be derived from the shaper parameters b, T and c as follows: The worst case occurs when there are unlimited NORMAL priority packets injected, the bucket is filled and the next c tokens are added after c cycles, which yields the maximum number of additions. Because at most one token can be consumed per time unit, the maximum time that a single low priority packet is delayed is (see Fig. 2 for an example):

$$t_{block} = b + \left\lceil \frac{t_{block} - c}{T} \right\rceil \cdot c \qquad (1)$$

This equation can be solved by iterating from $t_{block} = b$. A fixpoint exists if $c < T$, i.e. $r^{BE}_{max} < 1$.

During a BE burst, GB traffic does not receive its guaranteed average rate. The corresponding buffer must be at least $r^{GB}_{min} \cdot t_{block}$ in size so it can absorb enough GB traffic arriving at the rate r^{GB}_{min} for t_{block} (worst-case BE burst), to ensure that there will be enough GB data to send during future BE idle periods in which tokens accumulate for another burst.

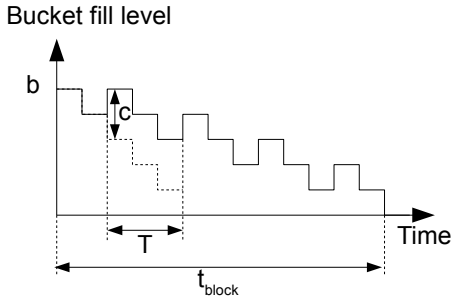

Fig. 2. Shaper bucket fill level for $b = 5, T = 3, c = 2$, resulting in $t_{block} = 13$.

B. Bandwidth Allocation Protocol

In order to provide a bandwidth guarantee on a network route, the corresponding traffic shapers need to be set up accordingly. This can be done by a distributed protocol using control messages sent as best effort traffic to set-up and tear-down a GB connection, which is also done in e.g. [7]. We assume that T and b are kept constant throughout the NoC, all shapers are disabled at initialization (e.g. by setting $c = T$) and traffic defaults to NORMAL priority.

To establish a guaranteed bandwidth channel from node A to node B, node A sends a control message to reserve a guaranteed rate $r_{request}$.This message contains a requested number of tokens per period $c_{request} = r_{request} \cdot T$ to reserve for this channel. The control message is routed deterministically on the same route as any other traffic between A and B. On its way, it is processed by every router. With T being constant, each router *decrements* c for the traffic shaper of the corresponding output port by $c_{request}$. This effectively limits the rate of BE traffic potentially blocking the GB traffic stream to $r_{BE} = \frac{T - c_{request}}{T}$ (after the first reservation). When the reservation message successfully reaches B, an acknowledge message is sent back to A. After that, the GB channel is established and may be used by sending packets on the LOW priority. If a router cannot free enough bandwidth ($c < c_{request}$), it discards the request and sends a not-acknowledge message back to A. Note that this message may take a different route than the setup packet, so it cannot directly free resources. To free reserved tokens – either as a response to a not-acknowledge message or when a connection is no longer needed – the requester A sends a control message to B or to the router that sent the not-acknowledge to free the previously reserved tokens $c_{free} = c_{request}$. Routers that see this message *increment* c in the corresponding shapers accordingly. The bandwidth allocation protocol has an overhead of two packet round trips on the BE traffic class, which amortizes quickly if large amounts of data need to be streamed.

C. Overlapping GB Streams

We have only discussed a single GB connection so far, but multiple independent reservations can be made along overlapping routes. Since LOW priority streams are indistinguishable from each other, they can interfere, i.e. "steal" bandwidth from each other. Hence, we require GB streams to conform to their requested average bandwidth, which can be done by a system policy or by additional shapers at the corresponding injection points. When GB streams overlap, there are three cases to consider: (1) Streams enter and leave the router over the same ports. Here, they can share both buffers and shapers. The blocking time reduces compared to a single stream due to reduced c, which also reduces buffering requirements. (2) Streams diverge, i.e. they enter at the same port, but leave over different ports. Naturally, they use separate shapers, but they must also use separate buffers to avoid that head-of-line blocking prevents streams from using their guaranteed bandwidth when it becomes available (i.e. when the corresponding BE shaper is depleted). The allocation of buffers to GB connections is done dynamically during the setup of GB connections. (3) When streams converge, they use different buffers, but share the same traffic shaper. Blocking time (and hence buffer size) becomes more complicated to calculate, since individual GB streams are not only blocked by BE traffic, but also by other GB streams (for the maximum packet length s per contending GB stream). Assuming the BE shaper does not saturate during that time (i.e. $b > (N-1) \cdot s/T \cdot c$), the blocking time for N converging GB streams is given by Eq. 2, which can be used along with the corresponding rates to calculate the minimum buffer sizes.

$$t_{block,conv} = b + (N-1) \cdot s + \left\lceil \frac{t_{block,conv} - c}{T} \right\rceil \cdot c \quad (2)$$

D. Discussion

We have employed virtual cut-through flow control for its favorable overload behavior. However, we acknowledge the fact that this requires large packet buffers compared to flit-sized buffers in the more commonly used wormhole flow control scheme. This may make our scheme seem infeasible for a current NoC. However, in a general purpose architecture, larger caches are located on every node anyway, which could be used to buffer packets, so virtual cut-through becomes viable again. Furthermore, we believe that our scheme can be extended to be used with wormhole flow control when each GB connection uses its own virtual channel (VC), so that stalled flits do not interfere with other streams. Multiple VCs for BE traffic are beneficial for the same reason.

With the arbitration scheme presented in this paper, bandwidth that is reserved but not consumed by GB traffic is wasted. BE traffic cannot be allowed to send with the corresponding traffic shaper being depleted without potentially violating the bandwidth guarantees, because link arbitration is done on packet level and thus a BE packet cannot be preempted when a GB packet arrives. This drawback can be solved by arbitrating on flit level (by using wormhole routing), so that the transmission of a BE packet can be preempted to transmit a GB packet in that case. Flit-level arbitration would also improve BE latency.

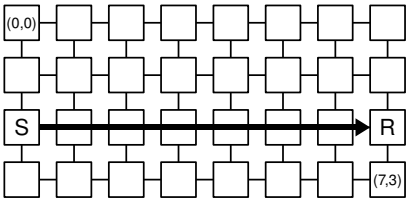

Fig. 3. Mesh with streaming application (Sender and Receiver).

Our arbitration scheme has additional hardware costs for the traffic shapers, improved arbitration logic, and configuration logic. We believe these are feasible, but no thorough studies have been conducted so far.

III. Experimental Evaluation

For a initial evaluation of the concepts, we have implemented the NoC outlined above in a SystemC model. The routers have two separate, relatively large buffers (one for each priority), each of which holds 8 packets. Routing delay is one cycle (very optimistic), and each router can transfer 4 bytes per cycle and output on bidirectional links. Network traffic is generated by packet generators with configurable randomized packet size, injection rate and destination distribution. These are used to model both background traffic as well as streaming traffic. All experiments are performed on a 4x8 mesh network. A packet generator modeling a streaming application is located on node (0,2) and sends packets along the x-axis to node (6,2), as illustrated in Fig. 3. All other nodes inject background traffic. Test cases were randomized (e.g. packet injection jitter) and executed for 100k cycles multiple times with different random seeds to obtain standard deviations and min/max values shown in the plots.

Our first experiments demonstrate the effect of network overload to the streaming application. The streaming application sends a packet of 32 bytes (including packet overhead) every 12 to 52 cycles, averaging 1 byte/cycle. Hence, it consumes 25% of the available bandwidth on that route (with a link bandwidth of 4 bytes/cycle). All other nodes inject background traffic. The left bar represents the streaming application running under medium network load: Each background tile injects a packet of 32 bytes every 10 to 22 cycles to a random destination. As a result, the streaming application achieves full throughput (left bar in Fig. 4), even though both traffic classes share the same priority and buffers.

To model an overload situation, background tiles inject 32 bytes every 8 to 12 cycles only to destinations in row 2. This results in a 65% drop in the streaming application bandwidth when the same priorities are used (second bar in Fig. 4), which could lead to severe degradation of service quality (e.g. deadline misses) for that application. The naïve solution to restore the throughput of the streaming application is to send the background traffic on LOW priority (without any

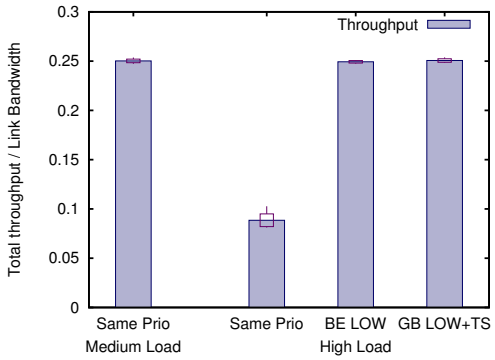

Fig. 4. Streaming traffic throughput.

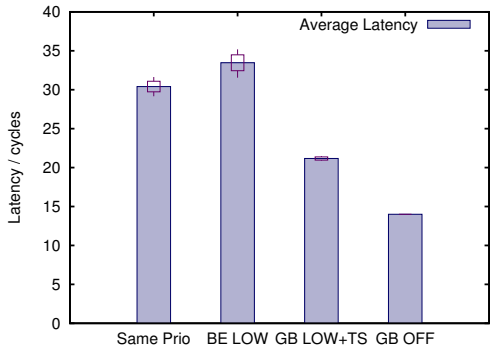

Fig. 5. Best effort traffic latency.

traffic shaping, third bar in Fig. 4). This, however, has negative effects on the latency, but can be seen as the best case for the streaming traffic. Alternatively, in our arbitration scheme, the streaming traffic is sent on LOW priority and traffic shaping of the NORMAL priority is enabled (with $b = 64$, $T = 64$ and $c = 48$, resulting in $r_{BE}^{max} = 75\%$) in the east-bound shapers of nodes (0,2) through (5,2) and the reception shaper of (6,2). This also restores the throughput of the streaming traffic to its original value (rightmost bar in Fig. 4).

To see the effect of these adjustments on the best effort traffic, we have measured the average latencies for tile (1,2) east of "S" in Fig. 3, sending bursts of 1 to 3 packets to row 2 every 44 to 84 cycles (1 byte/cycle average). The background load is low (32 bytes every 44 to 84 cycles) to reduce the impact of background traffic on the measured latency.

Fig. 5 shows the average latencies for different scenarios. The left bar constitutes the case where all traffic shares one priority and no guarantees are given. With the naïve solution of BE traffic being sent on LOW priority, the average latency increases slightly. When streaming traffic is sent on LOW priority with traffic shaping enabled (shaper settings as above), we can improve the latency by over 30%. This is still about 50% more than the best case latency (with streaming traffic OFF) due to the lack of packet preemption.

IV. Conclusion

In this paper, we have presented a link arbitration scheme for a Network-on-Chip targeted to general purpose computing, supporting two traffic classes: guaranteed bandwidth and latency-optimized best effort. Our approach allows bandwidth guarantees while maintaining much of the latency of the best effort traffic, which is essential for the performance of general purpose applications. This is achieved by prioritizing the latency-sensitive BE traffic over GB traffic and integrating traffic shapers in the link arbitration to maintain bandwidth guarantees. We have presented a distributed model to configure this mechanism and discussed the overheads and possible extensions of the scheme. Initial simulation results have demonstrated the effectiveness of the arbitration scheme to give tight bandwidth guarantees to streaming traffic under high load. They have also shown the positive impact on best effort latency under low load conditions compared to a simple prioritization of streaming traffic.

Acknowledgment

This work is supported by Intel Corporation. The authors would like to thank the researchers at Intel Germany Research Center for their valuable input and support.

References

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," UC Berkeley, Tech. Rep. UCB/EECS-2006-183.

[2] M. Azimi, N. Cherukuri, D. Jayashima, A. Kumar, P. Kundu, S. Park, I. Schoinas, and A. Vaidya, "Integration Challenges and Tradeoffs for Tera-scale Architectures," *Intel Technology Journal*, August 2007.

[3] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: Built for speed," *IEEE Micro*, vol. 26, no. 3, pp. 10–23, 2006.

[4] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob *et al.*, "An 80-Tile 1.28 TFLOPS Network-on-Chip in 65nm CMOS," *ISSCC 2007*.

[5] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. Miao, J. Brown III, and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[6] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: a many-core x86 architecture for visual computing," in *SIGGRAPH '08*, 2008.

[7] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE DESIGN & TEST*, pp. 414–421, 2005.

[8] T. M. Marescaux, *Mapping and management of communication services on MP-SoC platforms*. Technische Universiteit Eindhoven, 2007.

[9] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *J. Syst. Archit.*, vol. 50, no. 2-3, pp. 105–128, 2004.

[10] N. Kavaldjiev, G. Smit, P. Jansen, and P. Wolkotte, "A Virtual Channel Network-on-Chip for GT and BE traffic," *IEEE ISVLSI*.

[11] T. Bjerregaard, *The MANGO clockless network-on-chip: concepts and implementation*. IMM, Danmarks Tekniske Universitet.

[12] M. A. A. Faruque, G. Weiss, and J. Henkel, "Bounded arbitration algorithm for QoS-supported on-chip communication."

[13] S. Heithecker and R. Ernst, "Traffic shaping for an FPGA based SDRAM controller with complex QoS requirements," *DAC '05*, pp. 575–578.

[14] S. Whitty and R. Ernst, "A bandwidth optimized SDRAM controller for the MORPHEUS reconfigurable architecture," *IPDPS 2008*, 2008.

[15] B. Akesson, K. Goossens, and M. Ringhofer, "Predator: A predictable SDRAM memory controller," *CODES+ISSS '07*, pp. 251–256, 2007.