Mitigating Message Passing Interference in Trusted Embedded Platforms

Mattis Hasler, Sebastian Haas
Barkhausen Institut gGmbH, Dresden, Germany, Email: <first>.<last>@barkhauseninstitut.org

***Abstract*—Message passing (MP) hardware usually uses a fixed-size message buffer, which removes the need for resource consuming connection state. However, buffers may overflow and stall further incoming messages, jamming the communication fabric, and possibly impeding communication on the whole platform. The prevention of communication channel interference is especially important for the isolation of applications in a distributed operating system environment. By jamming the inevitably shared communication fabric one application may be able to obstruct the progression of another application (cf. denial-of-service attack), thus breaking isolation. In this work we propose a reject-and-resend mechanism for embedded MP hardware, increasing the robustness of the communication fabric by preventing jamming. It drastically lowers communication interference, while keeping the low cost and low latency, connectionless communication of common MP hardware solutions.**

I. Introduction

Message passing (MP) has many application scenarios in embedded systems. Increasing sizes of System-on-Chips make it necessary to partition systems into blocks or tiles and connect them with a generic communication fabric like a network-on-chip (NoC). Distributing applications over multiple tiles in such a fabric is usually implemented using MP as a communication primitive. MP is a widespread communication technique used on every computation scale from high-performance computing down to cache coherence protocols in CPUs [1]. MP can be implemented as a software library on top of remote direct memory access (RDMA) hardware [2] or be completely implemented in hardware [1]. The former version is preferred in large-scale scenarios where compatibility with existing hardware is essential and the software overhead does not weigh in too much, because of the large ratio of application calculation and communication. With shrinking application granularity the software overhead becomes a performance-degrading factor and full hardware implementations become favored [3]. However, realizing MP fully in hardware usually means not keeping connection states, because of its need for dynamic allocation, which is generally not possible in hardware. The main advantage of connection state is to have a transfer control, that manages buffer occupancy to prevent buffer overflows. Without transfer control, buffer overflows caused by one application may jam the NoC, hindering messages from an unrelated application, that happens to use the same NoC connection. In a healthy system, a jam may be short and cannot cause significant interference with other applications. However, a malicious application could exploit a message jam to deliberately deny usage of the NoC to other applications (denial-of-service (DoS) attack). The isolation of communication and as a result, especially in distributed operating systems, have been regarded as important in [4] and [5].

We propose the usage of a hardware reject-and-resend mechanic to remove the possibility of inter-application interference without introducing a resource-heavy connection state management.

II. Message Passing

RDMA-based MP can be implemented solely using the "send" method of the underlying hardware, which copies a local memory area to a remote one. To implement MP, a buffer and transfer management is needed that assures that messages arrive in allocated buffers and no read-after-write hazards occur (e.g. prevent overwriting of unread messages). The controller holds a connection state for each peer it exchanges messages with. A protocol of signaling messages between peers keeps these states in sync to assure safe message transfer.

Hardware implementions of MP do not hold connection states. Instead arriving messages are stored in a hardware buffer, regardless of their origin. The size of the buffer is fixed, usually a couple of slots, often only one. The CPU may be interrupted upon message arrival, but message polling by the CPU is also possible. When all message slots are occupied, no more messages can be received, and start jamming the NoC. Every time the CPU finishes reading a message it marks one slot as ready, allowing the next message from the NoC entering the buffer.

III. Reject And Resend

To overcome the jamming of the NoC and the consequences tied to it, we propose a mechanism that never stalls the NoC upon a buffer overflow. Instead of holding the message in the

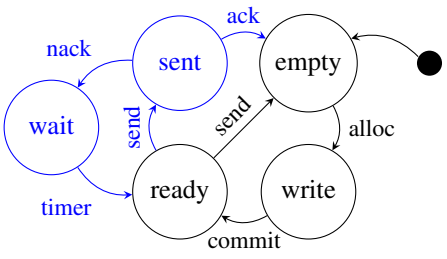

Fig. 1. Message buffer state diagram for message sending. The proposed extension introduces two new states (in blue) to implement the resending of a message based on a notification from the receiver.

NoC a message that cannot be stored into a buffer immediately is dropped. In any case, the sender of the message is notified about the arrival of the message with the information whether it could be stored or not. It is then the obligation of the sender to retry sending the message after some time. To implement this behavior, the sending message buffer is extended by two states, as shown in Fig. 1. When sending a message, the buffer is not freed immediately but awaits notification from the receiver. In case the receiver does not acknowledge the transfer (nack), the sender will wait a short time before returning the buffer to the "ready" state to be sent again.

IV. Evaluation

The proposed MP implementation —completely in hardware with a reject-and-resend extension— is compared in performance against a baseline variant without the extension as well as a software implementation based on an RDMA hardware unit. All three MP implementations are simulated in a system of 25 processors with $n_c = 5$ being consumers receiving messages and $n_p = 20$ being producers sending messages randomly to all consumers. All processors are connected by a circuit-switched orthogonal 5-by-5 mesh with bi-directional links. The consumers process a message in a constant time of $D = 1/\mu = 100$ cycles which resembles a realistic service call like it can be found in a distributed embedded OS [2]. Producers send messages randomly, following an exponential distribution with an average rate λ. The system utilization then becomes the ratio of total message production and consumption rates:

$$\rho_s = \frac{n_p \lambda}{n_c \mu}$$

In Fig. 2 expected message latency is shown depending on the system utilization. The maximum utilization $\rho_{\max}$ a system can handle is defined as the point where the message delay approches infinity, which is dependent on the used MP implementation. As expected the proposed extension does not affect the message latency and stays in the middle between the RDMA-based implementation and the theoretical maximum. The theoretical maximum is simulated by removing all MP implementation overhead, denoted as "null" in the figure. However, when purposely overloading one consumer, the proposed extension can limit the effects on the message latency to the overloaded consumer, keeping the other consumers unaffected. This is shown in another simulation with a fixed $\rho_s = 0.7$. Only the utilization of one specific consumer ρ_c is increased by adjusting the message rate of one producer only for one consumer. As can be seen in Fig. 3, when the ρ_c surpasses $\rho_{\max}$ the message latencies increase in the whole system when using a common hardware MP implementation. The isolation of communication is achieved by using the expensive RDMA-based MP or the proposed extension.

V. Conclusion

In embedded systems a hardware-based MP implementation may be favorable for resource and performance reasons.

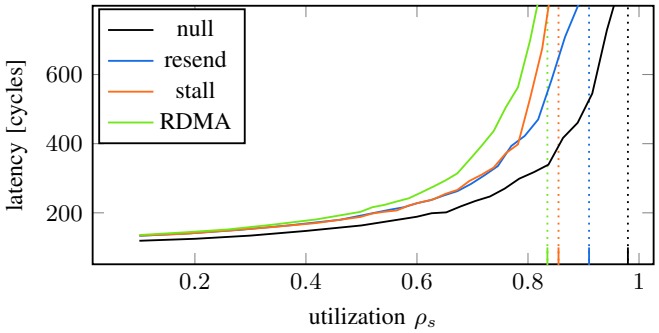

Fig. 2. Expected message latency depending on system utilization. Theoretical bound without message passing overhead (null) resembles a M/D/1-queue. The software-based (RDMA), baseline (stall) and extended (resend) implementations are compared.

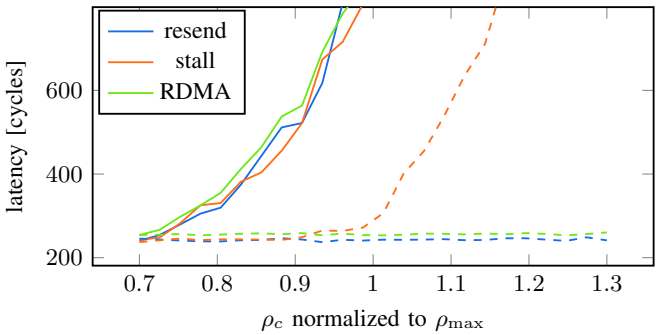

Fig. 3. Average message latency of an overloaded consumer (solid) and other consumers (dashed). Only in the baseline (stall) implementation an effect on all consumers can be observed.

However, common MP hardware implementations suffer from a vulnerability for DoS attacks, exploited by jamming the communication fabric with a message flood. We showed that a simple reject-and-resend extension can remove this vulnerability while preserving the resource and performance benefits of a hardware-based implementation.

Acknowledgment

This research is partly financed on the basis of the budget passed by the Saxon State Parliament in Germany.

References

[1] D. Petrović, T. Ropars, and A. Schiper, "Leveraging hardware message passing for efficient thread synchronization," *ACM Trans. on Parallel Computing*, vol. 2, no. 4, pp. 1–26, 2016.

[2] N. Asmussen *et al.*, "M3: A hardware/operating-system co-design to tame heterogeneous manycores," in *Proceedings of the 21st Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2016, pp. 189–203.

[3] G. Girão, D. Barcelos, and F. R. Wagner, "Performance and energy evaluation of memory organizations in noc-based mpsocs under latency and task migration," in *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*, Springer, 2009, pp. 56–80.

[4] M. Völp *et al.*, "Towards dependable cps infrastructures: Architectural and operating-system challenges," in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, Invited Paper, IEEE, 2015, pp. 1–8.

[5] A. A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *2008 The 28th International Conference on Distributed Computing Systems Workshops*, IEEE, 2008, pp. 495–500.