Data-Flow Deadlock Avoidance for Streaming Applications Mapped on Network-on-Chips

Vittorio Zaccaria
Dipartimento di Elettronica e Informazione
Politecnico di Milano - ITALY
Email: zaccaria@elet.polimi.it

ABSTRACT

In this paper, we tackle the problem of deadlocks within streaming applications running on network-on-chip (NoC)-based architectures. First, we introduce a motivating example which shows that even a deadlock-free streaming pipeline, when mapped onto a deadlock-free network-on-chip, can produce a data-flow deadlock. Second, we analyze state-of-the-art deadlock-avoidance techniques to understand their scalability versus increasingly complex application scenarios and multi-core architectures. Finally, we provide some potential solution from a combined compiler/OS/architecture point of view.

1. A MOTIVATING EXAMPLE

Let us consider Figure 1, which shows a pipeline of streaming filters. We assume the general case of variable-rate filters. Data is generated by filter A and elaborated by the chain composed of filters B, C and D. Each filter is a program running on a core of our target many-core system; the program contains an input queue from which data can be extracted with *get* and *peek* primitives exposed by the filter run-time architecture. Once elaborated, data is sent to the receiving filter's input queue by using *put* operations. In this paper, we consider that each *put* streaming primitive tightly interacts with the network interface (NI) of the core itself in order to send data across the network on chip connecting the processors. We assume that the *put* primitive blocks whenever the NoC router queue associated to the corresponding NI is full.

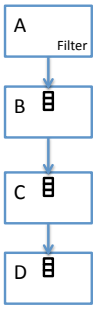

Figure 1: A deadlock-free pipeline of streaming filters.

We briefly show that the pipeline in Figure 1 is susceptible of *data-flow* deadlock whenever it is deployed into a low-level deadlock-free NoC. Let us assume to use a set of mesh routers for connecting the cores associated to the filters of Figure 1. The routers are organized in a row (see Figure 2). Packet routing between routers happens in the X-direction, which is provably low-level deadlock free. In fact, this is an instance of the more generalized XY routing which avoids circular dependencies between the buffers associated to the routers of a mesh.

Figure 2 shows that, since paths a and b within router 2 share the same physical channel (*channel* 2.3), they create a circular dependency between buffers involved in the communication between A-B and B-C. As can be noted, this situation is prone to a *data-flow* deadlock. Assume, for instance, that C is slow in consuming data produced by B. In this situation, the buffers associated to the B $\rightarrow$ C path (i.e., 2.C, 3.2, 4.3 and B.4) are filled back-to-back, forcing B to be often blocked into a *put* operation. The delay is accumulated over channels on the A $\rightarrow$ B path (4.B, 3.4 and 2.3). Those channels are, pessimistically, filled all with data coming from A towards B. Now, nothing prevents B from blocking again into a *put* (waiting for C to consume) while C is blocked waiting that channel 2.3 is freed for at least one slot (being all occupied by data directed to B). This is a circular dependency since C is waiting, essentially, on an event that only itself can produce; a *data-flow* deadlock has been generated.

2. STATE-OF-THE-ART

Although the general problem of *deadlock-avoidance* is a very investigated topic since decades, the interaction between NoCs and filter-mapping has been rarely taken into account in the recent research on streaming architectures.

Ideally, the problem can be solved by a suitable static filter mapping technique that avoids that streams belonging to the same pipeline branch do not intersect within the NoC topology. This would ensure that no circular dependencies are created within the streaming graph, for both static and variable rate filters.

One of the most notable and explicit contributions in this direction has been developed within the StreamIt framework [1, 2, 3]. The solution proposed by the StreamIt ideators deals specifically with the RAW architecture, a communication exposed multi-processing system where the NoC is composed by a 2D mesh. The nature of the solution combines the ability to orchestrate the communication between filters by using static information derived from the pipeline graph with a sofisticated simulation technique. This implies the

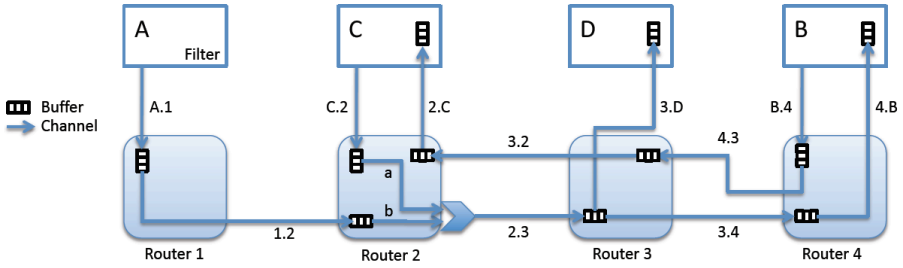

Figure 2: A set of streaming filters connected by buffered channels. Routing of packets between routers is free from low-level deadlocks. However, a data-flow deadlock is present.

implementation of a *communication scheduler* which statically schedules the communication by identifying deadlock conditions and serializing the communication to avoid buffer full scenarios.

3. A POTENTIAL SOLUTION - SYNERGY BETWEEN THE OS, THE COMPILER AND THE ARCHITECTURE

We must note that, in the literature, very few insights have been given for answering the following questions:

- How the static communication scheduling is put in place? Is it either enforced by the distributed action of the routers or by a single master entity regularly orchestrating the behavior of the single filters?
- What is the overall overhead due to the communication scheduling? How much the actual performance of the system are far from an ideal deadlock-free and scheduling-free scenario?
- Is static scheduling scalable and generalizable to different NoC-based streaming systems and future many-core architectures?

The above questions are key in order to understand the scalability and operability of the streaming paradigm towards NoC-based many-core architectures. In this scenario, several independent streaming applications can exploit the NoC fabric in a way that is hardly predictable by a static communication scheduler.

This situation can be easily identified in the IBM CELL^{TM} architecture where the OS is in charge of scheduling tasks (filters) on the available tiles (or SPUs). In this case, the compiler could not even access physical information about the actual NoC fabric, which is embedded in the OS kernel (as of version 2.6.23 of the Linux kernel for IBM Blade Servers). The current API allows for specifying an abstract 'affinity' property associated with tasks that should run on neighorhood SPUs, but this is not enough for implementing a 'traditional' static communication scheduler. On the other hand, the OS kernel is completely blind about the communication patterns of the tasks running on the SPUs. The four uni-directional buses constituting the CELL's Element Interconnect Bus might be used to construct 'virtual networks' to avoid deadlocks, but an inspection of the current Linux OS kernel and the API shipped with the CELL processor does not confirm any design decision taken in this direction.

The deadlock scenario is just the surface of an iceberg-sized problem which is essentially due to the missing synergy between the OS, the streaming compiler and the architecture. We believe that this gap should be filled in order to build performant and safe (working) streaming applications in the future many-core era.

The gap can be filled by: 1) virtualizing the network resources at the OS level and 2) allowing information about the actual data-flow to be passed from the compiler to the OS, in some standardized form (maybe just embedded into the ELF image associated with the application).

Concerning the first point, we believe that exposing the architecture/NoC features to the OS, such as the presence of alternative paths or virtual networks, is of utmost importance in improving the overall system efficiency. This would enable the OS to virtualize the streaming network resource into safe streaming channels to be exploited by the filters. In this sense, the *put, get* and *peek* operations would be implemented as actual OS system calls operating on virtualized resources.

Cocerning the second point, we note that a streaming-aware OS would open a wide-range of run-time optimization possibilities by allowing for safe communication among the streaming filters and optimized, ad-hoc filter mapping for performance and power consumption. We believe that the quest for such optimized run-time techniques is sound and should be taken into account by the OS developers in order to take full advantage of the streaming paradigm. This paradigm shift is comparable with the introduction of run-time binary optimization techniques for statically compiled programs.

4. REFERENCES

[1] M. Gordon, W. Thies, and S. Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. *ACM SIGPLAN Notices*, 41(11), Nov 2006.

[2] M Gordon, D Maze, S Amarasinghe, W Thies, and et al. A stream compiler for communication-exposed architectures. *ACM SIGARCH Computer Architecture News*, Jan 2002.

[3] J. Chen, M. Gordon, W. Thies, M. Zwicker, K. Pulli, and Frédo Durand. A reconfigurable architecture for load-balanced rendering. *HWWS '05: Proceedings of the ACM conference on Graphics hardware*, Jul 2005.