Predictable Embedding of Large Data Structures in Multiprocessor Networks-on-Chip

Sander Stuijk, Twan Basten, Bart Mesman, Marc Geilen

Predictable embedding of large data structures in multiprocessor networks-on-chip

(extended abstract)

Sander Stuijk[a], Twan Basten[a], Bart Mesman[a,b] and Marc Geilen[a]

[a] Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands.

[b] Philips Research Laboratories, Prof. Holstlaan 4, NL-5656 AA Eindhoven, The Netherlands.

s.stuijk@tue.nl

Abstract - This extended abstract presents models to derive timing and resource usage numbers for an application when distant, shared memories are used in an important class of future embedded platforms, namely network-on-chip-based multiprocessors.

1. Introduction

Real-time requirements in media systems have put the main focus on predicting the *timing* behavior of complex media systems. This has led to the development of predictable, tile-based multiprocessor networks-on-chip. Each *tile* contains one or a few processors and local memories [3]. These memories are typically too small to store large data structures (e.g. a video frame). A solution to this is to add tiles with large memories to the architecture. The memories will be *distant* and *shared* among potentially many computational resources. We use the terms *local* and *remote* to refer to the local processing tile or remote memory tile and their components.

2. Approach

To get a system with predictable timing properties, we need an appropriate design flow. The starting point of this flow is an SDF model of the application and a predictable architecture. Synchronous dataflow (SDF) and Boolean dataflow (BDF) [2] are often used to specify multiprocessor applications. Nodes in the graph, called *actors*, communicate with each other via *tokens* carried over the edges in the graph. An actor *firing* typically models a data transformation or some other activity and it consumes tokens from its inputs and produces tokens on its outputs. An actor may have an edge to itself (*self-edge*), modeling the state saving of an actor.

We use stepwise refinement of the application SDF model to include mapping decisions and to model the effects of architecture details such as buffer sizes. During this refinement, actors are annotated with appropriate execution times. The result is a combined SDF graph of the application and the architecture with predictable behavior wrt timing, memory usage etc. See [4] for an example of such a flow.

The mapping of data structures onto memories is one of the decisions that should be taken into account in this flow. To be able to reason about the timing properties of this mapping decision, we need a model of the memory accesses to the remote memory. Such a model is presented in Sec. 3. The communication between the remote memory and a local actor is handled by the communication assist (CA). Its role is to provide (pre)fetching of data from a remote memory in a transparent manner to a programmer. Support for prefetching is important as it allows hiding of large delays experienced when data is fetched over the network. The CA cannot be ignored when we want to build a predictable system with acceptable timing. Therefore, we need to refine the memory access model with an SDF model, presented in Sec. 4, of the (pre)fetching functionality of the CAs for accessing data in a remote tile memory. Timing annotations are omitted because they are application and architecture specific.

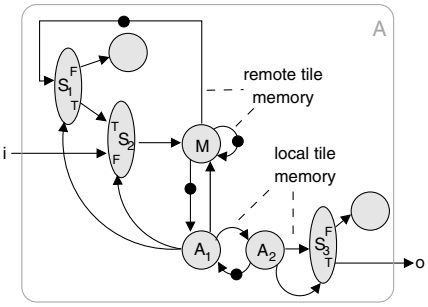

Figure 1. BDF model for remote memory accesses to state and input data.

3. Memory access model

Assume an actor A with a single input i, a single output o and a self-edge for its state. Furthermore, assume that input and state are stored in remote memory, and output in local memory. The following approach can be used to model this mapping decision (other mappings can be handled similarly). The basic idea is that the remote tile memory is modeled through a separate actor M. An actor can send a token to M to request a read or write of data stored in memory M. On its turn, M returns the requested data elements to the requesting actor. A BDF model based on this idea is shown in Fig. 1. The state of A stored in the remote tile memory is modeled via the self-edge on actor M. The input data for A, also stored in the remote tile memory, is modeled via the loop going through the switch S_1, the select S_2 and the actor

M. The switch S_1 and select S_2 are used to keep the existing input token (both control tokens true) or to read a new input token from i into the remote tile memory (both control tokens false). This approach allows for (pre)fetching of data elements to be incorporated later.

The functionality of actor A is split over two actors A_1 and A_2 which allows tighter bounds on the time at which tokens are consumed/produced because A_2 and M can operate in parallel. When actor A_1 is executed, it reads the data elements it requested from the remote tile memory into the local memory. The local tile memory is modeled by the edges from A_1 to A_2 and from A_2 to switch S_3. A_1 simultaneously sends a new request to the remote tile memory to read and/or write data. It further outputs control tokens for S_1 and S_2, both with the same value. As soon as A_1 finishes its firing, it hands over control to A_2. Actor A_2 reads the local state and performs a transformation on it, which is equal to the transformation performed by the original actor A. At the end of its firing, it outputs a control token to switch S_3. This switch is used to control the production of an output token for the original actor A on its output o. The actor A_2 indicates to the switch whether the produced data, stored in local memory, is valid or not. If it is not valid, the token is discarded; if it is valid, the token is put on output o. Actor A_2 hands back control to A_1 as soon as it has to read or write data that is not stored in the local tile memory.

Remote memory tiles will be shared among many processing tiles. To get a predictable system, each processing tile must get guarantees on the response time of the remote memory. This can be realized by using a TDMA scheme to control access to the remote memory [1]. Using such a scheme, different processing tiles using the same remote memory can be considered independently. For each actor A whose data is mapped to a remote memory we can use the model shown in Fig. 1 (or variants of it for different mapping decisions). The sharing of the remote memory is taken into account via the timing behavior of actor M (memory).

4. Prefetching model

(Pre)fetching functionality is implemented in the CAs in our architecture template. Consider the ith firing of actors A_1 and A_2 in the memory access model of Fig. 1, called $A_{1,i}$ and $A_{2,i}$ in the remainder. Actor firing $A_{1,i}$ produces data for firing $A_{2,i}$ and it requests data from the remote memory that is needed for firing $i+1$ of A_2. The local CA sends this request to the remote CA. On its turn, the remote CA will return the requested data. Next, the local CA has to copy the data into the local tile memory. However, it might be that not all requested data can be stored in the local memory when firing $A_{2,i}$ is being executed (and thus part of the local memory is still occupied). For that reason, the local CA might have to break the request to the remote CA into two steps. First, the local CA requests and receives the data

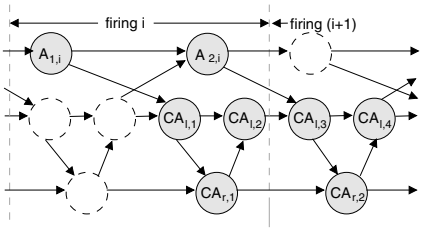

Figure 2. SDF model for (pre)fetching.

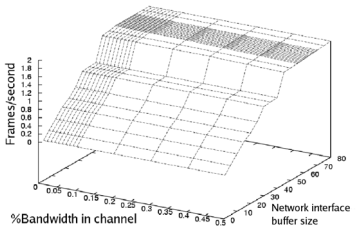

Figure 3. H.263 design space.

that can be *prefetched* during firing $A_{2,i}$. Next, it *fetches* the data that could not be stored in the local memory. An SDF model for the behavior of the CAs is shown in Fig. 2. The various firings of A_1 and A_2 are turned into separate actors. The actors $CA_{l,1}$, $CA_{r,1}$ and $CA_{l,2}$ model the prefetching of data and the actors $CA_{l,3}$, $CA_{r,2}$ and $CA_{l,4}$ model the fetching of data. These last three actors can be omitted if all data can be prefetched. Note that the remote tile memory actor M has been abstracted away. Its (timing) behavior is included in the CA_r actors. It is interesting to observe that the untimed (pre)fetching model of Fig. 2 is independent of the prefetching strategy. Prefetching strategies only influence the execution time annotations of the model.

5. Results

We used our approach to determine the communication bandwidth and network interface buffer sizes that must be allocated to guarantee the performance of a motion compensator in an H.263 decoder. For this we used, besides the models presented here, an SDF model of the network. The resulting design space is shown in Fig. 3. The experiment shows that the number of frames per second is not influenced by the allocated bandwidth (which is always sufficient), but that the network interface buffers need to be large (64 elements) to obtain the maximal throughput.

References

[1] M. Bekooij, et al. Predictable multiprocessor system design. In *SCOPES'04*, p. 77–91. Springer, 2004.

[2] J. Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model.* PhD thesis, University of California, Berkeley, CA, 1993.

[3] D. Culler, et al. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[4] P. Poplavko, et al. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *CASES'03*, p. 63–72. ACM, 2003.