Mapping streaming applications on a reconfigurable MPSoC platform at run-time

Philip K.F. Hölzenspies, Gerard J.M. Smit, Jan Kuper
University of Twente, the Netherlands

***Abstract* — In this paper we present a method for mapping streaming applications, with real-time requirements, onto a reconfigurable MPSoC. In this method, the performance of the hardware architecture (the reconfigurable Processing Element, the Network Interface and the Network-on-Chip) is integrated in the performance models of the applications. In this way the performance of the mapped application can be determined at run-time. A predictable NoC (guaranteed bandwidth and bounded latency), a predictable Network Interface and a predictable Processing Element are key requirements for our approach.**

I. Introduction

In the EU-FP6 project 4S[*] we propose an energy-efficient reconfigurable Multi-Processor System-on-Chip (MPSoC) architecture with run-time software and tools. The MPSoC architecture consists of a heterogeneous set of processing elements (PEs) (also denoted as tiles) interconnected by a Network-on-Chip (NoC). By exploiting the available parallelism of the PEs, they can run at a relatively low frequency (below 500 MHz) to achieve sufficient performance at an acceptable energy cost. The architecture, including the run-time software, can replace inflexible ASICs for future embedded systems.

A. Streaming applications and MPSoCs

In this paper we focus on the domain of streaming applications [1]: e.g. wireless baseband processing for audio/video broadcast (DAB, DRM, DVB) and multi-media processing (MPEG-2, MPEG-4). Streaming applications show a predictable temporal and spatial behavior. Some characteristics are:

- Relatively simple local processing on huge amounts of data.
- Throughput guarantees (in data items per sec.) are required for the processing as well as for the communication.
- The life-time of a streaming application is semi-static, which means that its processes and communication streams are fixed for a relatively long time (seconds to hours).

Current standard processing architectures are not suitable for adaptive streaming applications because:

- General Purpose Processors (GPPs) have a large overhead; therefore they are by far not efficient enough for high-performance streaming applications,
- Streaming implies a very high I/O rate (mega samples/sec) that cannot be handled by von Neuman style processors,
- Data caching, one of the pillars of modern processors, is not very effective for streaming applications and therefore needs to be reconsidered.

It is expected that devices with hundreds of cores interconnected by a NoC [6] can be designed and will become available in the coming years [10]. We expect that, due to energy and performance constraints, these cores will be reconfigurable and heterogeneous.

We believe reconfigurable MPSoC architectures will, to a large extend, replace ASICs especially for the streaming part of energy-efficient embedded systems. This trend leads to a number of challenges. For example: we have learned in the last decades that mapping an arbitrary program on a MPSoC is a tough problem; the available parallelism is often difficult to exploit. However, for an extremely important sub-class of programs, streaming applications mentioned above, we believe this mapping problem is more manageable. The reason is that streaming applications are inherently parallel and that memory can be distributed. This is beneficial for the memory wall, the energy wall, and the ILP wall [3].

B. SR-SDF graphs and MCM analysis

To map streaming applications on a MPSoC, we assume the applications can be represented as communicating parallel processes. In this paper, we use a Single Rate Synchronous Data Flow (SR-SDF) graph model [7] which is a directed graph with nodes representing sequential processes (also called tasks or actors) and edges representing FIFO communication between processes. The vertices of an SR-SDF graph are called actors; they model some activity. The actors are characterised by an execution time given as a label of the actor. The graph's edges represent dependencies between the actors. The actors interact by exchanging tokens over the connecting edges. An edge behaves like a FIFO buffer where the tokens are stored.

When there is at least one token preset on each input edge of an actor, the actor is executed (also called fired). After a time period equal to its execution time, the actor produces one token on each of its output edges. To prevent the start of a second execution before the first one has finished, the actor can also be connected with a self-edge with a single token.

Figure 1 shows an example SR-SDF graph that models a bounded FIFO buffer with a capacity of two data items. The number of tokens on the cycle between the two actors corresponds to the buffer capacity. Each token on the upper

[*] This work is part of the 4S project [4] that has been supported by the Sixth European Framework Programme under project number IST 001908

edge corresponds to an empty buffer space and each token on the lower edge corresponds to a full buffer space. When a token arrives on the edge IN, the actor A_1 is executed, consuming an empty buffer space and producing a full buffer space. Subsequently, executing A_2 consumes a full buffer space and produces an empty buffer space and a data item on the OUT edge.

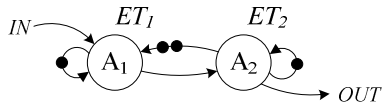


Figure 1: SR-SDF model of a FIFO buffer of capacity two data items

Given an SR-SDF graph, we can derive its throughput in terms of the number of tokens per time unit by applying a standard analysis technique for single-rate synchronous data flow models called Maximum Cycle Mean (MCM) analysis [8]. MCM analysis examines all cycles in an SR-SDF graph and determines their *cycle mean*. The cycle mean of a cycle is defined as the ratio between the sum of the execution times of all the actors on the cycle and the number of tokens on the cycle.

SR-SDF graphs have two important properties [8]: periodicity and monotonicity. The periodicity property means that after a transient period after the start of an application, the execution of a strongly connected SR-SDF graph will exhibit periodic behavior. The monotonicity property means that the throughput of a SR-SDF graph is a non-decreasing function of the execution time of the actors. In other words, decreasing these execution times may only lead to equal or higher throughput. In our models, the execution times may vary, but we label the actors always with the worst case execution times. Hence, according to the monotonicity property, by applying the MCM analysis we derive the worst case throughput.

C. Run-time mapping

The MPSoC architecture of our system is controlled by a Central Coordinating Operating System (CeCOS) that runs on one of the GPPs (General Purpose Processors) of the MPSoC. The main task of the CeCOS is to manage the system resources. When applications are started, the CeCOS tries to satisfy the Quality of Service (QoS) requirements, to optimize the resources usage and to minimize the energy consumption. To reduce the energy consumption, each process is mapped on the PE that can execute it most efficiently. This so called *spatial mapping* of processes is performed at run-time by the spatial mapping tool [5]. Run-time mapping offers a number of advantages over design-time mapping. For example:

- to adapt to the available resources. In general it is only known at run-time what mix of applications will run in parallel;
- to enable unforeseeable upgrades after first product release;
- to avoid defective parts of a SoC. Larger chips mean lower yield. The yield can be improved when the mapper is able to avoid faulty parts of the chip. Also, aging can lead to faulty parts that are unforeseeable at design-time.

The CeCOS determines when the spatial mapping tool is called; in principle only when a new streaming application is started, but sometimes applications should be remapped, because other applications have freed resources that will make the applications to be remapped run more efficiently. The objective of the spatial mapping tool is to minimize the energy consumption of an entire application: processing as well as the inter-process communication while preserving timing guarantees. To be able to perform the mapping of actors to PEs at run-time the mapping algorithm needs:

1. A model of the application (in our case an SDF graph),
2. A model of the SoC platform (number and type of tiles, and NoC structure),
3. The constraints of the application (e.g. throughput and/or latency requirements),
4. The expected worst case execution time (and resource usage, e.g. energy consumption) of the process implementations on specific tiles,
5. The performance model (time as well as energy consumption) of the inter-process communication.

In this paper, we present a method for estimating the performance of mappings of streaming applications on a MPSoC. This can be used by the run-time mapping tool, to estimate whether a certain mapping is timing wise feasible.

We use the following phases in our approach:

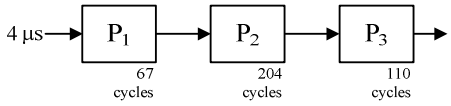
- Phase 1: (design-time) model the application as a SR-SDF graph
- Phase 2: (run-time) map all the actors to an available and suitable Processing Element (PE)
- Phase 3: (run-time) extend the SR-SDF graph with the response time of the actor on the selected PE
- Phase 4: (run-time) extend the SR-SDF graph with the performance of the selected route through the NoC
- Phase 5: (run-time) perform MCM analyses to obtain the performance estimates of the entire mapped application
- Phase 6: (run-time) depending on the result of step 5, go back to step 2 (re-map the application), or accept the current mapping. The critical cycle can be determined and can be fed-back to the mapping algorithm.

II. CASE STUDY HIPERLAN/2

We illustrate our method with a simple example of a number of processes connected in a pipelined fashion. We use the HiperLAN/2 [1] receiver which we map on a MPSoC. How the HiperLAN/2 receiver is partitioned is discussed in [2]. The tasks are compiled for Montium processing tiles [2] and the processing times reported by the compiler are given in the figure (in clock cycles).

The HiperLAN/2 receiver processes information received on a wireless communication channel. Every 4 μs a new data item (OFDM symbol = 64 32-bit words) arrives on the input of the receiver and the receiver must be ready to process it. Therefore, the data inter-arrival period of 4 μs defines the real-

time constraint on the receiver operation. To be able to process all arriving data items, the receiver must have throughput of at least $TH_R = 1/4\ \mu s = 250$ data items/ms.



P1 - Frequency offset correction
P2 - Inverse OFDM
P3 - Equalization, Phase offset correction and De-mapping

Figure 2: Pipeline of a HiperLAN/2 receiver

One of the tasks of the mapping tool is to evaluate the performance of the mapping (timing as well as energy performance). The performance depends on the hardware architecture of the PE, the Network Interface (NI) and the NoC. In this section we show how the PE organization and NoC latency can be incorporated in the SR-SDF model of the application to predict the overall (time and energy) performance of the streaming applications.

A. MPSoC and NoC architecture model

The application pipeline consists of *3* tasks, denoted as P_1 to P_3, and we assume they are mapped onto separate PEs of the MPSoC. The processed data items are transported between the PEs by the NoC. The NoC and a PE exchange data through the PE's local memory (MEM); the received data items are loaded in the MEM and after processing they are read from the MEM and transmitted to the next PE. The data exchange between a PE and the NoC is handled by a network interface (NI). The communication uses back-pressure. For example, when the input data buffer, reserved in the MEM, for arriving data items is full, the NI stops receiving and in turn blocks the next data item in the network.

The performance of the application is determined by the time that the tasks need to process a data item and by the time needed to communicate a data item between two PEs. To predict the application's performance, all the worst case processing and communication times of the application must be known before the mapping takes place.

Besides the processing and communication times, the application performance also depends on whether processing and communication in a PE can be performed simultaneously. When the PE is used in "block mode" [9], the parallelism between processing and communication is restricted by the MEM. Since each of the three operations (receiving, processing and transmitting a data item) requires access to the MEM, these operations can be performed in parallel only if the MEM supports parallel access. Based on the simultaneous access supported by the MEM, we consider the following three cases of parallelism in a PE (more options are possible): (1) single port access: the MEM allows only one access at a time, so only one operation can be performed simultaneously, (2) dual port access: the MEM can be accessed by two actors simultaneously and (3) triple port access: the MEM can be accessed by three actors at the same time.

For this paper we assume that the PE organization supports single access only. In a heterogeneous MPSoC a mixture of all three mechanisms mentioned above might be present. Therefore, at run-time, when a task is mapped on the MPSoC, depending on which particular PE is selected, one of the three access methods is used.

For predicting the application's performance in a system, we build an application model that captures all the aspects influencing the application's performance – the processing and communication times, the effect of blocking due to the back-pressure and the parallelism enabled by the PEs memory organization.

B. Predicting the throughput of a single process

To model a process mapped on a PE as an SR-SDF graph we need three actors: one for receiving a data item, one for processing and one for transmitting. We refer to these actors as receiving, processing and transmitting actors.

In a single-port memory only one of the three actors can be active at a time. Following the data dependencies, the natural order in which the actors are executed is: receiving, processing and transmitting. For each data item, this cyclic pattern is repeated. We model this task behavior with the SR-SDF graph in Figure 3. The graph contains one cycle of length three with one token. The token in the cycle is circling around the actors allowing only one of them to execute at a time. The token can be interpreted as a grant for memory access: the actor that currently has the token has access to the memory. The three self edges guarantee that actors only fire after their previous execution finishes. Because we only have one token circling in the loop, the self edges are redundant in this case.

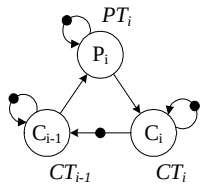


Figure 3: An SR-SDF model of a task running on a PE with a single-port data memory

Applying MCM analysis we derive the throughput of the graph. Through MCM analysis we find the MCM and the throughput of the graph in tokens per second:

$$MCM_1 = \max\left[CT_{i-1}, PT_i, CT_i, \left(CT_{i-1} + PT_i + CT_i\right)\right] = CT_{i-1} + PT_i + CT_i \quad (1)$$

$$TH_1 = \frac{1}{CT_{i-1} + PT_i + CT_i} \quad (2)$$

Since a grant for memory access is given for the time of processing and/or communicating a single data item, this result is interpreted as the worst case throughput in words per second of the task *i* running on a PE with single-port data memory. TH_R is the requested throughput of the application. From (2) it follows that to guarantee a lower bound TH_R on the throughput ($TH_1 \geq TH_R$) we must have that:

(3) $$CT_{i-1} + PT_i + CT_i \leq \frac{1}{TH_R}$$

C. Throughput of the whole application

The model of the whole application is constructed by combining the separate models of the application's tasks. Since in our example the application has a pipeline structure, the tasks are concatenated in a pipeline. The throughput of the application is determined by the throughput of the slowest task in the pipeline.

In Phase 2 of our mapping algorithm a suitable tile is chosen. The concatenation of the task models is done by merging the sending and receiving actors of consecutive stages of the pipeline, which assumes that the sending and receiving start and finish at exactly the same time. Such an assumption neglects the network delay due to the network buffering. The performance characteristics can now be filled in.

After Phase 4 the communication parameters should be taken into account. To express the NoC delay and throughput in the application SR-SDF graph explicitly, the graphs of the separate tasks are connected by introducing a new actor(s) between the sending and receiving actors. Figure 4 shows an SR-SDF model of an application of two tasks (P_1 and P_2) running on PEs with a single-port data memory and communication actor N_2. For simplicity self-edges are omitted. The execution time of the new actor N_2 has to be set to the delay (in cycles) of the communication channel between the tasks. The throughput of the NoC can be modeled with an actor with a self-cycle with a single token. This is largely simplified when the NoC has a predictable performance: i.e. the NoC has a guaranteed throughput and an upper bound on the latency. A NoC as proposed in [6] has such properties.

The throughput of the overall application TH_G is found by applying MCM analysis for all cycles in the graph. In our simple example equation (2) for each of the three tasks in the pipeline determine the throughput:

(4) $$TH_G = \frac{1}{\max_{i\in[1,3]}\left[CT_{i-1} + PT_i + CT_i\right]}$$

The throughput bound TH_R is derived by applying inequality (3) for each of the three tasks:

(5) $$CT_{i-1} + PT_i + CT_i \leq \frac{1}{TH_R}, \text{ for } i \in \{1,2,3\}$$

By providing that all the inequalities (5) are satisfied we guarantee that the application throughput is greater than or equal to TH_R.

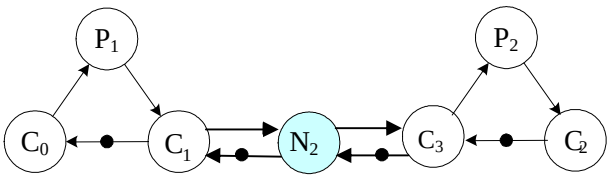


Figure 4: Two tasks running on PEs with a single-port data memory and communication latency actor N_2

D. Throughput of sample application (see Figure 2)

In our case, the Montium tiles have single port memories, so we can derive the throughput of the sample application by combining three inequalities (5).

The Montium tiles are set to run at a fixed clock frequency of 100 MHz. Hence, the actual task execution times are PT_1=0.67 µs, PT_2=2.04 µs and PT_3= 1.1 µs.

Any solution of the system of inequalities (5) gives a set of worst case communication times for which the required application throughput TH_R is guaranteed. One possible solution is: CT_0=2.35µs, CT_1=0.98µs, CT_2=0.98µs, CT_3=1.92µs. Having the communication times and the size of the communicated data items, we can verify the throughput of the entire application including the throughput of the communication channels of the NoC.

III. Conclusion

In this paper we study run-time mapping of streaming processes on PEs of a MPSoC. The performance of applications running on a MPSoC is influenced by the memory organisation of the PEs, the clock frequencies of the PEs and the NoC. In our approach, we extend the application graph with performance figures of the PEs and the NoC. The performance parameters (latency, throughput and energy) can be introduced in the model at run-time, so the overall system performance of a specific mapping can be determined at run-time.

References

[1] William Dally et al. "Stream Processors: Programmability with Efficiency" ACM Queue, March 2004, pp. 52-62

[2] P. M. Heysters, "Coarse-grained reconfigurable processors", CTIT Ph.D.-thesis series No. 04-66, 2004.

[3] D. Patterson, Arvind, K. Asanovic, D. Chiou, J. Hoe, C. Kozyrakis, S. Lu, M. Oskin, J. Rabaey, J. Wawrzynek, "RAMP: Research Accelerator for Multiple Processors," Technical Record of the 18th Hot Chips Symposium, Palo Alto, CA, August 2006

[4] http://www.smart-chips.net

[5] L.T. Smit, J.L. Hurink, G.J.M. Smit: "Run-time Mapping of Applications to a Heterogeneous SoC.", In: Proceedings 2005 International Symposium on System-on-Chip, Tampere, Finland. pp. 78-81, November 2005

[6] N. Kavaldjiev, G. J. M. Smit, and P. G. Jansen. "A virtual channel router for on-chip networks" In Proceedings of IEEE International SOC Conference, pages 289–293, IEEE Computer Society Press, September 2004.

[7] E. A. Lee et al. "Synchronous dataflow.", Proceedings of the IEEE, 75(9):1235{1245, September 1987.

[8] J. L. Pino et al. "Hierarchical static scheduling of dataflow graphs onto multiple processors". In Proceedings of ICASSP, May 1995.

[9] M.D. van de Burgwal, G.J.M. Smit, G.K. Rauwerda, P.M. Heysters, "Hydra: an Energy-efficient and Reconfigurable Network Interface" In: Proceedings of ERSA2006, 26-29 June 2006

[10] Vangal, Sriram et al: "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS", proceedings ISCC 2007, pp 98-99