Architecture Design Space Exploration for Streaming Applications Through Timing Analysis

Maarten H. Wiggers, Nikolay Kavaldjiev, Gerard J. M. Smit, Pierre G. Jansen
Department of EEMCS,
University of Twente, the Netherlands
{wiggers, nikolay, smit, jansen}@cs.utwente.nl

Abstract. In this paper we compare the maximum achievable throughput of different memory organisations of the processing elements that constitute a multiprocessor system on chip. This is done by modelling the mapping of a task with input and output channels on a processing element as a homogeneous synchronous dataflow graph, and use maximum cycle mean analysis to derive the throughput. In a HiperLAN\2 case study we show how these techniques can be used to derive the required clock frequency and communication latencies in order to meet the application's throughput requirement on a multiprocessor system on chip that has one of the investigated memory organisations.

Introduction

Advances in silicon technology enable multi-processor system-on-chip (MPSoC) devices to be built. MPSoCs provide high computing power in an energy-efficient way, making them ideal for multimedia consumer applications. Multimedia applications often operate on one or more streams of input data, for example: base-band processing, audio/video (de)coding, and image processing. An MPSoC consists of Processing Elements (PE). For scalability reasons we envision that in the near future MPSoCs will include a Network-on-Chip (NoC) for communication between PEs, as i.e. [1].

Multimedia applications can be modelled conveniently using a task graph, where the vertices represent functions and the edges data dependencies. The data streams through the graph from function to function.

A subclass of multimedia applications operates under hard real-time constraints: throughput and latency requirements are put on the inputs and outputs of the task graph. To satisfy these requirements, methods are needed that allow reasoning, predicting and guaranteeing the application performance for a given mapping on a multi-processor architecture. Using such an analysis method different architectures can be compared, so that for given timing requirements the architecture that runs at the lowest clock frequency can be found.

This paper analyses the temporal behaviour of multimedia applications mapped on a multiprocessor architecture by modelling the mapping with Homogeneous Synchronous DataFlow (HSDF) graphs and applying the associated analysis techniques. The contribution of this paper is that it shows how these analysis techniques can be used for design space exploration, to find an architecture instance given the timing constraints and given an optimisation criterion (in our case clock frequency) which has its influence on the energy efficiency. We explore different memory organisations for the PEs and their consequences for the clock frequency of the processor and the requirements imposed on the NoC.

The approach is based on the following assumptions: i) an upper bound on the task's execution time can be given; ii) upper bounds on the data communication latencies can be given. Finding a tight upper bound on the execution time of a piece of code is a hard problem, but using techniques as presented by Li this can be done [2]. When multiple tasks are mapped on the same processor, then a scheduling policy needs to be applied on this processor that provides an upper bound on the waiting time of the task. An upper bound on the communication latencies can be given by a communication infrastructure that provides guaranteed latency such as [1][3].

Poplavko [4] uses SDF inter-processor communication (IPC) graphs [5] to find minimal buffer sizes by accurately modelling the Æthereal NoC [3] and analysing the temporal behaviour of a JPEG decoder mapped on an MPSoC consisting of ARM processors and the Æthereal NoC. We do not aim for buffer minimization but aim for an architecture that meets the applications timing constraints at low energy consumption.

An untimed HSDF graphs is similar to a Marked Graph Petri Net [6]. The time semantics applied here for HSDF graphs is similar to time Petri Nets [7].

The organisation of this paper is as follows. In Section 1, the organisation of the MPSoC template is given. The HSDF model of computation and its associated analysis technique is presented in Section 2. In Section 3, the different memory organisations for the PEs are presented and their throughput is analysed, after which in Section 4 the consequences are described when an application is mapped over multiple PEs. Section 5 describes a case study in which the data processing part of a HiperLAN\2 receiver is mapped on a MPSoC consisting of a number of MONTIUM processing tiles [8], after which we conclude in Section 6.

1. System Organization

An abstract representation of the multiprocessor system considered in this paper is given in Figure 1. It consists of multiple *Processing Elements* (PEs) that are connected to a *Network-on-Chip* (NoC) through *Network Interfaces* (NI). A PE includes a processor, instruction memory, and data memory; the processor is for instance a domain-specific or general purpose processor. One or several tasks (τ_i) can execute on a PE. When communicating tasks are mapped on the same PE then the communication channel between them is mapped on the local memory. When communicating tasks are mapped on different PEs then the channel is mapped over the local memories of both PEs and the NoC is used to transport data from one PE to the other. Tasks only access the PE's local memory.

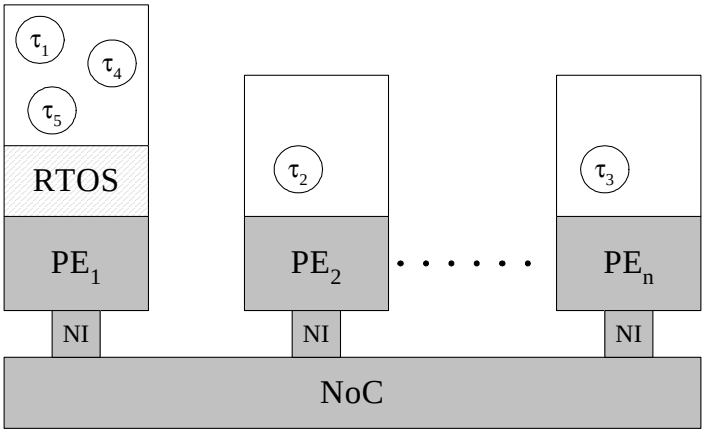

Figure 1. An abstract representation of a multiprocessor system

The NoC provides reliable, in-order, and guaranteed latency services on *connections*. A connection is a channel between NIs, and can go over routers in the NoC. The size of the data items on the connection is known. Guaranteed latency provides an upper bound on the time between the moment that the first word of the data item is written on the connection and the moment that the last word is available for reading. Communication over the NoC is event-triggered: data can be transferred as soon as both NIs (sending and receiving) are ready for communication on the same connection.

The NI hides the NoC details from the PEs. It also has DMA (direct memory access) functionality and can transmit data from the PE's memory on the network and write data received from the network in the memory.

The organisation of a PE together with its NI is presented in Figure 2. It consists of a processor, instruction memory, data memory and a NI. The NI can operate in parallel to the processor and accesses the memory for inter-PE communication. Furthermore, the NI has separate sending and receiving parts that operate independently. In this case three parties can request memory access at a particular time – PE, sending and receiving part of the NI. An extension to more than one input or output connection can be further considered, but for clarity reasons it will not be discussed in this paper.

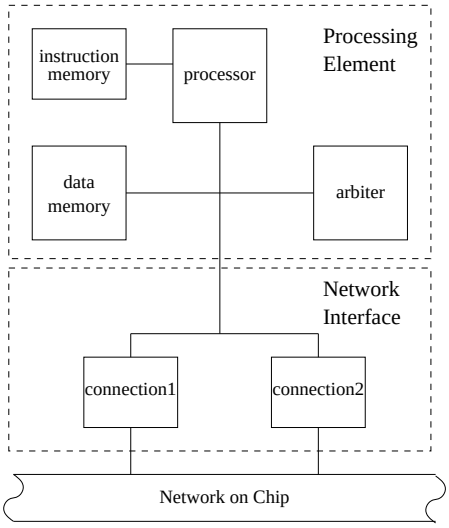

Figure 2. PE organization

Conflicts between the three parties requesting memory access can be solved through scheduling of memory accesses or through multiple memory ports. Several options for solving the conflicts are discussed in this paper. Each of the options is studied as an HSDF model of a single task running on a PE. Throughput is derived for the models and compared.

2. Homogeneous Synchronous DataFlow

HSDF [9] is a model of computation in which multimedia applications can be conveniently modelled and with which analysis techniques are well suited to derive the throughput and latency of hard real-time applications.

The vertices of an HSDF graph are called actors. Actors communicate by exchanging tokens over channels which are represented by the edges of the graph. The channels are unbounded first-in first-out (FIFO) buffers. In the HSDF graph, tokens are represented as black dots on the edges.

The actors in the HSDF graph represent some activity. An HSDF actor has a firing rule that specifies the number of tokens that needs to be present on the input channels. When the

firing rule is met the actor is enabled after which it can fire. The difference between the firing time and the finish time is the execution time. At the finish time the actor atomically removes a predefined number of tokens from its input channels and places a predefined number of tokens on its output channels. By definition the actors in a homogeneous SDF graph always consume and produce a single token on a channel; SDF graphs allow the modelling of so-called multi-rate applications. For clarity reasons we restrict the present discussion to HSDF graphs, a similar approach can be taken with SDF graphs. In all the HSDF graphs the token consumption and production rates are omitted for clarity reasons. Self-timed execution of an HSDF graph means that the actor fires as soon as it is enabled.

Figure 3 shows an example HSDF graph that models a bounded FIFO buffer of capacity two data items. The actors A1 and A2 are the producer and consumer on this FIFO. The number of tokens on the cycle between the actors corresponds to the capacity of the FIFO. A self edge with one initial token enforces that the previous firing of the actor must have finished before the next firing can start. A self-edge is required to model state over different firings of the same actor.

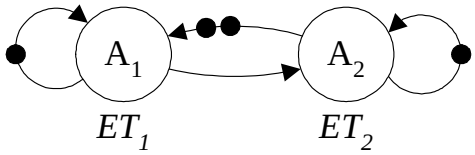

Figure 3 HSDF model of a FIFO

HSDF graphs have two important properties: (1) monotonicity, and (2) periodicity. Self-timed execution of an HSDF graph is monotonic [10]. This means that decreasing actor execution times will only lead to non-increasing actor firing times, and thus will only lead to increasing or unchanged throughput.

After a transient phase in the beginning, the self-timed execution of a strongly connected HSDF graph will exhibit periodic behaviour. The throughput of the HSDF graph after the transient phase can be derived using Maximum Cycle Mean (MCM) analysis of a strongly connected HSDF graph [11]. The mean of a simple cycle *c* in an HSDF graph is defined as the sum of the execution times (ET) of the actors, *a*, on the cycle divided by the number of tokens on the cycle. The MCM of an HSDF graph *G*, λ_G, is found by calculating the cycle mean of every simple cycle *c*:

$$\lambda_G = \max_{c \in G} \left[\frac{\sum_{a \in c} ET(a)}{tokens(c)} \right] \quad (1)$$

The throughput T of the graph G is:

$$T_G = \frac{1}{\lambda_G} \quad (2)$$

For example, the HSDF graph in Figure 3 contains three cycles and its λ_G is max[ET1/1, ET2/1, (ET1+ET2)/2], while the throughput is the inverse of the λ_G.

3. Modelling of a Single Task on a PE

This section discusses a single task running on a PE. The task receives and sends its data from/to other PEs. It is shown how the task including the communication can be modelled as an HSDF graph, taking into account the PE architecture.

The processor and the sending and receiving part of the NI access the data memory in parallel and contention may occur on the memory port. In order to resolve the contention, arbitration on the memory port is used. The arbitration can be done at two levels: token level and word level. At token level the arbitration is done on a coarse granularity. Access is granted to either the processor or the NI until it finishes its operation: processing, sending or receiving of a data item respectively. At word level the arbitration is done on a finer granularity. Access to the memory is granted on a word-by-word basis.

Intuition says that arbitration on the word level is advantageous if either the processor or the NI does not access the memory every clock cycle. This will for instance occur for control-oriented tasks, and for processors with a large register set or multi-cycle operations. In this paper we only consider token level arbitration, because our focus is on the data processing part of the application that frequently accesses the memory. For a discussion on word level arbitration see [12].

Figure 4 shows how a dataflow graph of an application is mapped on our MPSoC. The application is partitioned into three tasks: τ_1, τ_2 and τ_2. We call the dataflow graph in Figure 4 a mapping-unaware graph. Information about the mapping is included in the graph by extending the mapping-unaware graph with actors that model the communication latency.

Figure 5 shows how the mapping-unaware graph of a single task, τ_i, is extended with the knowledge that the tasks are mapped on different PEs and that communication between the tasks has a certain (guaranteed) latency. The annotated times (ET_{Ci-1}, $ET\tau_i$, and ET_{Ci}) represent either the upper-bound on the execution time in the case of the tasks or the upper-bound on the latency of moving a data item from one memory to another memory.

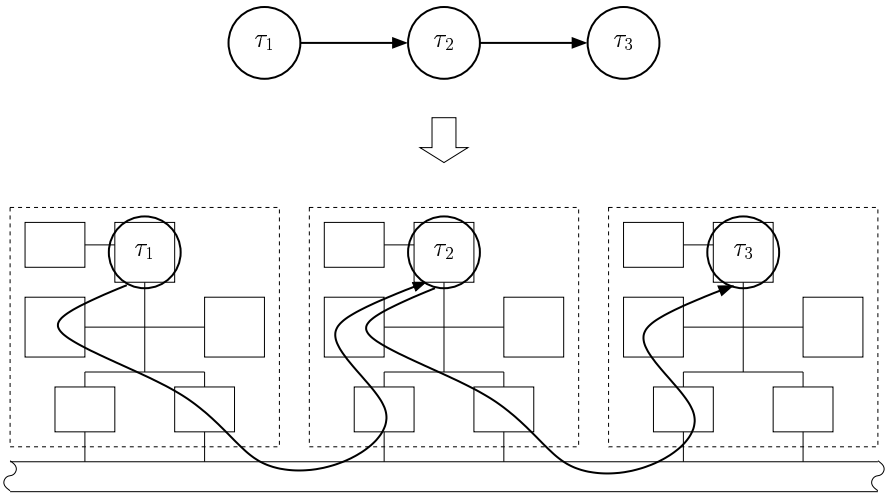

Figure 4. Mapping of an application graph on a MPSoC.

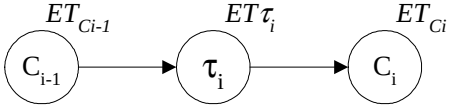

Figure 5. The dataflow between receiving part of the NI, processor, and sending part of the NI.

The graph from Figure 5 still does not contain all the information about the PE architecture. It has to be further extended with information about the memory organisation and the arbitration on the data memory port.

We consider three data memory organisations in the following subsections: (1) a single-port, (2) dual-port or (3) three-port data memory organisation. For each organisation an HSDF model is constructed and achievable throughput is compared. In a later section it is shown how a model of a complete application running on multiple PEs can be derived using the results for a single PE.

3.1. Arbitration on 1 Memory Port

Assume a PE has one single-port data memory. To resolve the conflicts between the three entities (task, input connection and output connection) that access the memory a static schedule S_0 can be applied. Figure 6 presents this schedule as an HSDF graph. Because of the 1-to-1 mapping one can view the actors modelling either the logical entities as mentioned or the processor, receiving part of the NI, and sending part of the NI. The token can be interpreted as a grant for memory usage: the actor that currently possesses the token owns the memory. The edges model the data dependencies between the entities: memory access should be first granted to the input connection Ci-1, then to the task on the processor τ_i and then to the output connection Ci. The execution time of an actor equals the maximal time that the corresponding entity will keep the memory.

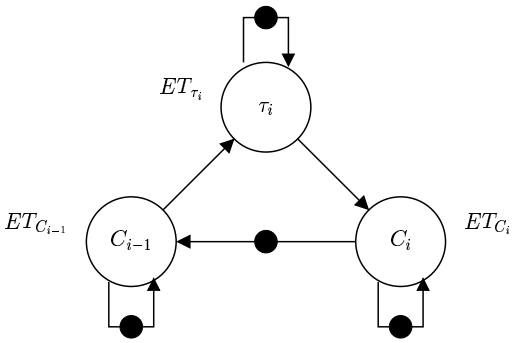

Figure 6. HSDF graph corresponds to schedule S_0

Excluding the self edges the graph contains one cycle with one token. Applying Eq. (1) and (2) the throughput of the graph is derived:

$$\lambda_{S_0} = ET_{Ci-1} + ET\tau_i + ET_{Ci} \rightarrow T_{S_0} = \frac{1}{ET_{Ci-1} + ET\tau_i + ET_{Ci}}$$

If a lower bound T on the throughput has to be guaranteed, then from the above equation we see that the following must hold:

$$ET_{Ci-1} + ET\tau_i + ET_{Ci} \leq \frac{1}{T}$$

3.2. Arbitration on 2 Memory Ports

When the PE's data memory is implemented as a dual-port memory or two separate single-port memories, then two entities can access it simultaneously. Note that in the case of multiple single-port memories combined with a task that carries state from one firing to the next firing special care needs to be taken for storing and retrieving the state. We assume here that the task is a function that does not have state (the self-edge only enforces sequential firings). Figure 7 and Figure 8 present HSDF graphs of two contention free schedules, S_1 and S_2, for that memory organization. There are two tokens circulating in the

graph that correspond to the two memory ports. The actor τ_i corresponds to task *i*, and actors C_{i-1} and C_i correspond to the task's input and output connection respectively.

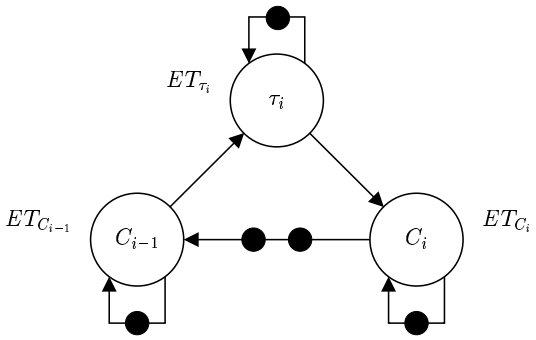

Figure 7 The HSDF graph corresponding to schedule S_1.

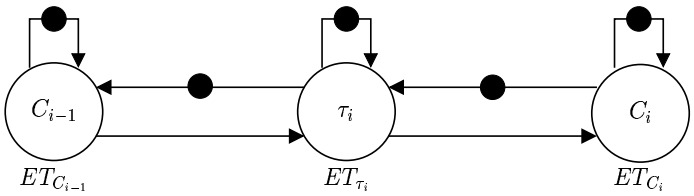

Figure 8 The HSDF graph corresponding to schedule S_2

Applying Eq. (1) and (2) the throughput of the schedules is:

$$\lambda_{S_1} = \frac{ET_{Ci-1} + ET\tau_i + ET_{Ci}}{2} \rightarrow T_{S_1} = \frac{2}{ET_{Ci-1} + ET\tau_i + ET_{Ci}},$$

$$\lambda_{S_2} = \max\left(ET_{Ci-1} + ET\tau_i, ET\tau_i + ET_{Ci}\right) \rightarrow T_{S_2} = \frac{1}{\max\left(ET_{Ci-1} + ET\tau_i, ET\tau_i + ET_{Ci}\right)}$$

The throughput of S_1 is greater than or equal to the throughput of S_2. This is because in S_2 the task is granted access to both memory ports.

If a lower bound T on the throughput has to be guaranteed, then from the above equation it is seen that the following must hold:

$$ET_{Ci-1} + ET\tau_i + ET_{Ci} \le \frac{2}{T}, \text{ for } S_1; \quad \begin{vmatrix} ET_{Ci-1} + ET\tau_i \le \frac{1}{T} \\ ET\tau_i + ET_{Ci} \le \frac{1}{T} \end{vmatrix}, \text{ for } S_2$$

3.3. Arbitration on 3 Memory Ports

When the PE data memory is implemented as a three port memory or three separate single-port memories, then all three actors can access a memory simultaneously. Arbitration on the memory ports is not needed. It is only necessary to keep the data dependencies. Two HSDF graphs, S_3 and S_4, for that memory organisation are shown in Figure 9 and Figure 10.

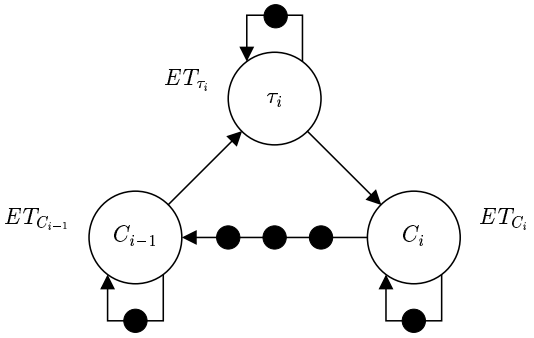

Figure 9 This HSDF graph corresponds to schedule S_3

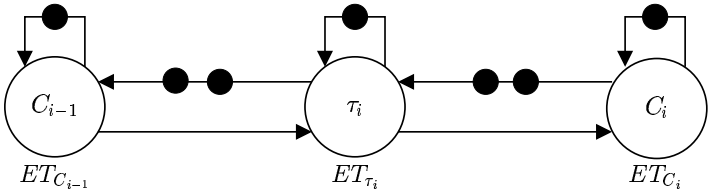

Figure 10 This HSDF graph corresponds to schedule S_4.

Applying Eq. (1) and (2) we derive the throughput of the schedules:

$$\lambda_{S_3} = \frac{ET_{Ci-1} + ET\tau_i + ET_{Ci}}{3} \rightarrow T_{S_3} = \frac{3}{ET_{Ci-1} + ET\tau_i + ET_{Ci}},$$

$$\lambda_{S_4} = \max\left(\frac{ET_{Ci-1} + ET\tau_i}{2}, \frac{ET\tau_i + ET_{Ci}}{2}\right) \rightarrow T_{S_4} = \frac{1}{\max\left(\frac{ET_{Ci-1} + ET\tau_i}{2}, \frac{ET\tau_i + ET_{Ci}}{2}\right)}$$

The throughput of schedule S_3 is greater than or equal to the throughput of schedule S_4.

If a lower bound T on the throughput has to be guaranteed, then from the above equations it is seen that the following must hold:

$$ET_{Ci-1} + ET\tau_i + ET_{Ci} \le \frac{3}{T}, \text{ for } S_3; \quad \begin{vmatrix} ET_{Ci-1} + ET\tau_i \le \frac{2}{T} \\ ET\tau_i + ET_{Ci} \le \frac{2}{T} \end{vmatrix}, \text{ for } S_4$$

Extending this discussion to multiple tasks mapped on the processor and thus multiple connections can either be done by extending the static order schedule with these tasks and connections or applying i.e. Time Division Multiple Access (TDMA) arbitration, as presented by Bekooij [12], on the processor and NIs.

3.4. Comparison

Table 1 summarises the result for the memory organisations discussed above. For each of them the table gives the throughput and the constraints on the actors' execution times implied by an application throughput bound T.

Table 1 Summary of the results

Mem.		Throughput	Constraints
Single-port	S0	$T_{S0} = \frac{1}{ET_{Ci-1} + ET\tau_i + ET_{Ci}}$	$ET_{Ci-1} + ET\tau_i + ET_{Ci} \le \frac{1}{T}$
Dual-port	S1	$T_{S1} = \frac{2}{ET_{Ci-1} + ET\tau_i + ET_{Ci}}$	$ET_{Ci-1} + ET\tau_i + ET_{Ci} \le \frac{2}{T}$
	S2	$T_{S2} = \frac{1}{\max(ET_{Ci-1} + ET\tau_i, ET\tau_i + ET_{Ci})}$	$ET_{Ci-1} + ET\tau_i \le \frac{1}{T}$ $ET\tau_i + ET_{Ci} \le \frac{1}{T}$
Three-port	S3	$T_{S3} = \frac{3}{ET_{Ci-1} + ET\tau_i + ET_{Ci}}$	$ET_{Ci-1} + ET\tau_i + ET_{Ci} \le \frac{3}{T}$
	S4	$T_{S4} = \frac{1}{\max\left(\frac{ET_{Ci-1} + ET\tau_i}{2}, \frac{ET\tau_i + ET_{Ci}}{2}\right)}$	$ET_{Ci-1} + ET\tau_i \le \frac{2}{T}$ $ET\tau_i + ET_{Ci} \le \frac{2}{T}$

To compare the throughput results we assume the same actors' execution times (ET_{Ci-1}, $ET_{\tau i}$ and ET_{Ci}) in the five cases. This results in a lattice:

$$\left.\begin{array}{l} T_{S_0} < T_{S_1} < T_{S_3} \\ T_{S_2} < T_{S_4} \\ T_{S_2} \le T_{S_1} \\ T_{S_4} \le T_{S_3} \end{array}\right\}$$

S_0 has lowest throughput and S_3 has highest throughput. As can be expected an increase in memory ports (or the number of separate memories used) leads to an increase of the PE throughput.

Given an application throughput bound T, the maximal achievable processor utilisation can be derived from the constraints in Table 1. Higher processor utilization leads to lower clock frequencies and therefore to lower power consumption. Processor utilisation ρ is defined as the ratio between the time a processor is busy and the period at which the data arrives. For each data item a processor is busy for time $ET_{\tau i}$. The data arrival period is 1/T. Thus $\rho = T * ET_{\tau i}$. Taking into account that the throughput bound requires that the execution times for all the actors are smaller than or equal 1/T, from the constraints we derive the maximal achievable $ET_{\tau i}$ and thus the maximal achievable processor utilisation. The results are given in Table 2. S_0 has worst utilisation while S_1, S_3 and S_4 allow for 100% utilisation of the processor.

Table 2 Maximal achievable processor utilization

Mem.		Maximal processor utilisation
Single-port	S0	$1 - T(ET_{Ci-1} + ET_{Ci})$
Dual-port	S1	1
	S2	$\min[(1 - T * ET_{Ci-1}), (1 - T * ET_C)]$
Three-port	S3	1
	S4	1

In the same way the latency requirements can be compared. Consider the constraints inequalities in Table 1 and assume that the processing time $ET_{\tau i}$ is fixed. Then it can be seen that the latency requirements (ET_{Ci-1} and ET_{Ci}) are most difficult for S_0 and most relaxed for S_3 and S_4.

4. Application Model

The previous section discussed how a single task of an application can be modelled such that information about the PE architecture where the task runs is included in the HSDF graph. Here the model is extended to the entire application.

Consider the application shown in Figure 4 and assume that all its tasks (τ_1, τ_2 and τ_3) are mapped on PEs with a single-port memory. The HSDF graph of the mapping is shown in Figure 11. It is constructed by extending the original application graph with the communication latencies and the constraints between the different actors due to the scheduling on the memory port. The communication latency ET_{Ci} is the time that it takes to move a token (data item) from the data memory in PE_i to the data memory in PE_{i+1}.

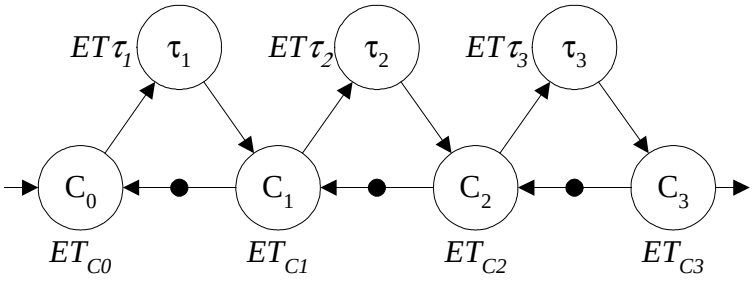

Figure 11 An HSDF graph of the application from Figure 4 assuming PEs with a single-port memory and direct communication between the tasks

This graph contains three simple cycles each with a single token. Applying Eq. (1) and (2) for this HSDF graph we find that the throughput of the application is:

$$T_G = \frac{1}{\max\limits_{i\in\{1,2,3\}}\left(ET_{Ci-1} + ET\tau_i + ET_{Ci}\right)}$$

The last can be restated in the following way: the necessary and sufficient condition for the application having throughput equal to or higher than T is:

$$ET_{Ci-1} + ET\tau_i + ET_{Ci} \le \frac{1}{T}, \text{ for } i \in \{1,2,3\}$$

This system of inequalities gives the relation between the global application throughput requirement T and the constraints for a particular mapping of the tasks.

When the communication between PEs is not direct and data is buffered in between then the application HSDF graph is changed as shown in Figure 12 for a buffer capacity of *n* data items. For example, data is written through the network to a logical FIFO properly implemented on a memory that is larger than the local memories and later read again through the network. The execution times of the send (S) and receive (R) actors equal the latency guarantees given by the NoC for transmission of the data to and from this secondary memory plus the time required to update the FIFO administration.

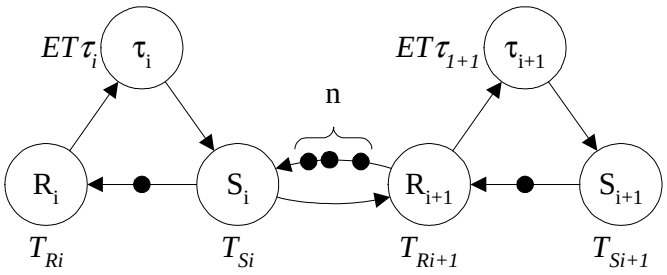

Figure 12 Buffered communication between the PEs. It is assumed storage with FIFO organization and capacity of n data items

Figure 13 presents an HSDF model of the application from Figure 4 assuming PEs with a dual-port memory using schedule S_2. It is derived by extending the original application graph with details about the PEs architecture as in Figure 8. The communication between the PEs is direct.

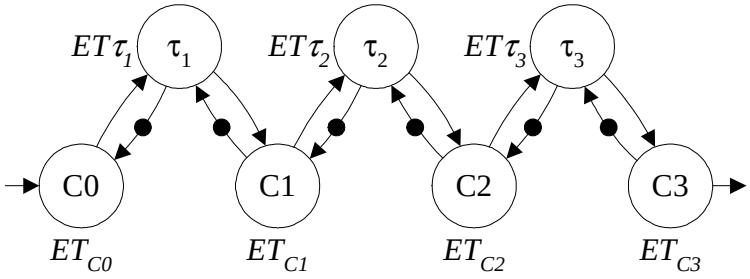

Figure 13 HSDF graph of the application from Figure 4 assuming PEs with dual-port memory and direct communication between the tasks.

The graph contains six simple cycles each with one token. Applying the Eq. (1) and (2) the throughput of the application is derived:

$$T_G = \frac{1}{\max\limits_{i\in\{1,2,3\}}\left[\left(ET_{Ci-1} + ET\tau_i\right), \left(ET\tau_i + ET_{Ci}\right)\right]}$$

If a lower bound T of the application throughput has to be guaranteed then the following should hold:

$$\begin{vmatrix} ET_{Ci-1} + ET\tau_i \le \dfrac{1}{T} \\ ET\tau_i + ET_{Ci} \le \dfrac{1}{T} \end{vmatrix}, \text{ for } i \in \{1,2,3\}$$

In the same way HSDF models for the other PE organizations can be constructed. It is not necessary for all PEs to have the same organization, the architecture can be heterogeneous as for each PE a corresponding HSDF graph is substituted. Figure 14 shows an example HSDF graph of the same application assuming that the first PEs has a dual-port memory with schedule S_1, the second PE has a three-port memory with schedule S_4, and the PE where task τ_3 is mapped on has a single port memory.

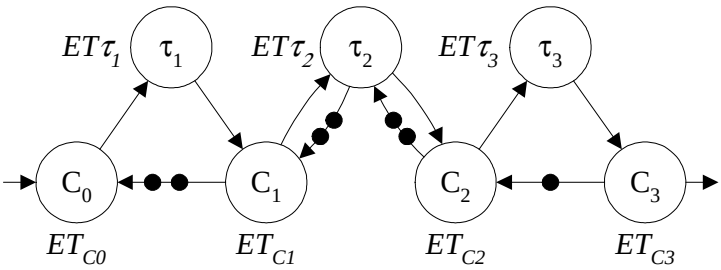

Figure 14 HSDF graph of the application from Figure 4 assuming PEs with dual port memory and direct communication between the tasks

The graph contains 4 simple cycles – three with two tokens on them and one with a single token. According Eq. (1) and (2) the throughput of the application is:

$$T_G = \frac{1}{\max\left[\left(\frac{ET_{C0} + ET\tau_i + ET_{C1}}{2}\right), \left(\frac{ET_{C1} + ET\tau_i}{2}\right), \left(\frac{ET\tau_i + ET_{C2}}{2}\right), \left(\frac{ET_{C2} + ET\tau_i + ET_{C3}}{1}\right)\right]}$$

Each of the four terms in the max function corresponds to one of the cycles in the graph. If lower bound T of the application throughput has to be guaranteed then it should be provided:

$$\begin{cases} ET_{C0} + ET\tau_i + ET_{C1} \le \frac{2}{T} \\ ET_{C1} + ET\tau_i \le \frac{2}{T} \\ ET\tau_i + ET_{C2} \le \frac{2}{T} \\ ET_{C2} + ET\tau_i + ET_{C3} \le \frac{1}{T} \end{cases}$$

5. HiperLAN/2 Example

In this section a HiperLAN/2 receiver is used as an example to demonstrate how HSDF throughput analysis is applied for real streaming applications. HiperLAN/2 [13] is a wireless local area network (WLAN) standard, based on Orthogonal Frequency Division Multiplexing (OFDM), which is defined by the European Telecommunications Standards Institute (ETSI).

The HiperLAN/2 receiver will run on three PEs. The PEs are MONTIUM processing tiles [8] – domain-specific processors for the domain of mobile communications. The tiles communicate through a NoC as presented in [1].

The application is partitioned in three tasks [14] each of which will run on a separate PE. The dataflow graph is given in Figure 15. The tasks τ_1, τ_2 and τ_3 implement the base band processing of the HiperLAN/2 receiver. The graph is annotated with the sizes of the data items on the communication channels and the number of cycles required for processing the data item on a Montium. In order to request a guaranteed latency connection the data item size is required. The number of cycles enables calculation of the task execution times. Further the graph is a homogeneous SDF graph: all consumption and production rates are 1.

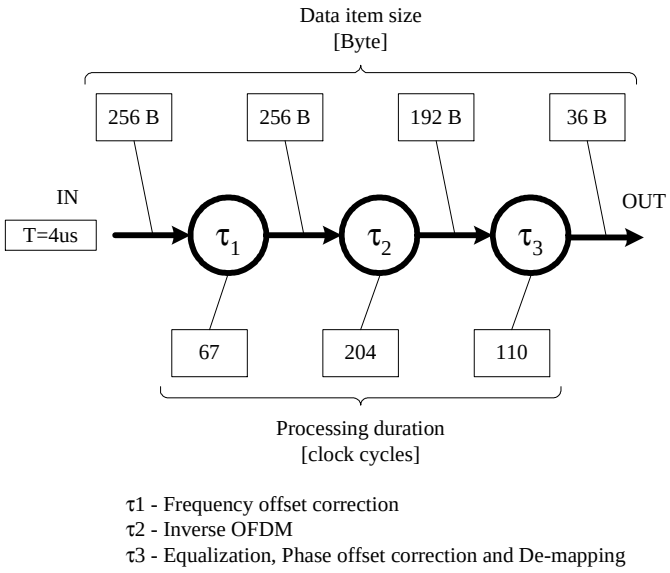

Figure 15 Process graph of a HiperLAN/2 receiver

A HiperLAN/2 receiver has to handle a new OFDM symbol (data item) every 4μs. This is the throughput requirement of this application. It is required that the application has a throughput greater than or equal to 1/(4 us) = 250 OFDM symbols per ms.

The MONTIUM tile has a single-port memory and the NoC provides direct communication without buffering. Therefore, the HSDF graph from Figure 11 can be directly used for modelling the application. Here the arriving OFDM symbols correspond to tokens arriving to the application. The lower bound on the application throughput is T=250 [token/ms].

Assuming that the three tiles run on a clock frequency of 100 MHz and considering the number of cycles per firing given in Figure 15 we can calculate the execution times for the processing actors in the HSDF graph: $ET_{\tau 1}$=0.67us, $ET_{\tau 2}$=2.04us, $ET_{\tau 3}$=1.1us. Taking into account the throughput requirement T and system of inequalities given for the graph in Figure 11,

$$ET_{Ci-1} + ET\tau_i + ET_{Ci} \le \frac{1}{T}, \text{ for } i \in \{1,2,3\},$$

we derive the constraints for the communication latencies:

$$\begin{cases} ET_{C0} + ET_{C1} \le 3.33\mu s \\ ET_{C1} + ET_{C2} \le 1.96\mu s \\ ET_{C2} + ET_{C3} \le 2.9\mu s \end{cases}$$

One possible solution of this system of inequalities is: ET_{C0}=2.35us, ET_{C1}=0.98us, ET_{C2}=0.98us, ET_{C3}=1.92us. These are the upper bounds on the latency guarantees to be requested from the network. The utilisation of the MONTIUM tiles will be: ρ_1=0.17, ρ_2=0.51, ρ_3=0.28.

In the case that the network cannot provide the requested latency guarantees we can take the lowest possible latency that can be provided. Now starting with these fixed latencies the system of inequalities will give the minimum task execution times $ET_{\tau 1}$, $ET_{\tau 2}$ and $ET_{\tau 3}$ and consequently the minimum processor clock frequencies.

If the MONTIUM tiles had dual-port memory, then according Table 2 it would be possible to achieve 100% processor utilisation (applying S_1). Assume that this is the case. In order to keep the tiles busy all the time, the tasks execution times are set equal to the arrival period of the data items: $ET_{\tau 1}=ET_{\tau 2}=ET_{\tau 3}=ET=4us$. Taking into account the number of cycles given in Figure 15 the tiles clock frequencies are calculated: $f_1=16.75MHz$, $f_2=51MHz$, $f_3=27.5MHz$. Considering schedule S1, the graph in Figure 7 is used for constructing the HSDF graph, given in Figure 16, of the application running the three tiles.

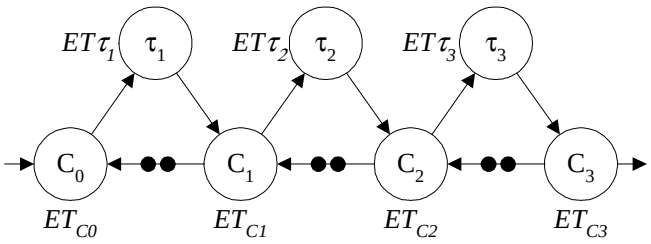

Figure 16 HSDF graph of a HiperLAN/2 receiver running on three Montium tiles assuming the tiles had dual-port memories organized according schedule S_1

The throughput equations for the graph in Figure 7 are already derived. They give the necessary and sufficient conditions for guaranteeing a lower bound on the application throughput T:

$$ET_{Ci-1} + ET\tau_i + ET_{Ci} \le \frac{2}{T}, \text{ for } i \in \{1,2,3\},$$

Since the tasks execution times are already fixed, for the communication latencies it must hold that:

$$\begin{cases} ET_{C0} + ET_{C1} \le 4\mu s \\ ET_{C1} + ET_{C2} \le 4\mu s \\ ET_{C2} + ET_{C3} \le 4\mu s \end{cases}$$

One possible solution of this system of inequalities is: $ET_{C0}=2us$, $ET_{C1}=2us$, $ET_{C2}=2us$, $ET_{C3}=2us$. Compared with the results for the real MONTIUM architecture we see that with dual-port memory and S_1 the highest possible tile utilisation is achieved while the latency requirements are the same or relaxed.

6. Conclusion

We have shown how different memory organisations of the processing elements that constitute an MPSoC can be compared based on their throughput. Further we have shown how the throughput of a mapping can be evaluated by first modelling the application as an HSDF graph and then extending this graph with actors that model the effects of the mapping, e.g. the latency of the communication channels.

Even though we have only presented an application that is organised as a pipe, we believe that this approach can be extended in a straightforward way to include arbitrary application graph topologies.

One of the strengths of this approach is that we can model the application as well as the mapping on possibly heterogeneous PEs in a single graph in an intuitive way. Throughput can be derived from this graph by analytical means, allowing for tool support, which will be necessary for larger or multi-rate graphs.

HSDF graphs can only model static behaviour, in the sense that it cannot model dynamic token consumption or production rates or dynamic (data dependent) execution times. How we can accurately model and analyse the interaction between the control and data parts of the application is therefore future work.

References

[1] Nikolay Kavaldjiev, Gerard J. M. Smit, Pierre G. Jansen, "A Virtual Channel Router for On-chip Networks", *Proceedings of IEEE International SOC Conference*, pp. 289-293, September 2004.

[2] Y-T. S. Li and S. Malik, Performance analysis of real-time embedded software, ISBN 0792383826, Kluwer academic publishers, 1999.

[3] E. Rijpkema, K.G.W. Goossens, and A. Radulescu, *Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip*. In Proceedings of DATE'03, 350-355, ACM, 2003.

[4] P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman. "Task-Level Timing Models for Guaranteed Performance in Multiprocessor Networks-on-Chip", CASES'03, October 2003.

[5] S. Sriram, and S.S. Bhattacharyya, Embedded Multiprocessors: Scheduling and Synchronization, Marcel Dekker, Inc., 2002.

[6] T. Murata, *Petri Nets: Properties, Analysis, and Applications*. In Proceedings of the IEEE, vol. 77, no. 4, pp. 541-580, April 1989.

[7] A. Cerone and A. Maggiolo-Schettini, Time-based expressivity of time Petri nets for system specification. Theoretical Computer Science 216, pp. 1-53, 1999

[8] Heysters P.M., Smit G.J.M. & Molenkamp E. "A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems", *The Journal of Supercomputing*, volume 26, issue 3, Kluwer Academic Publishers, November 2003.

[9] E.A. Lee, and D.G. Messerschmitt, *Synchronous Data Flow*. In Proceedings of the IEEE, vol. 75, pp. 1235-1245, 1987.

[10] M. Bekooij, O. Moreira, P. Poplavko, B. Mesman, M. Pastrnak, and J. van Meerbergen. *Predictable embedded multi-processor system design*. In Scopes 2004, 8th International workshop on software and compilers for embedded systems. Amsterdam, The Netherlands, 2-3 September 2004.

[11] F. Baccelli, G. Cohen, G.J. Olsder, and J-P. Quadrat, Synchronization and Linearity. New York: Wiley, 1992.

[12] M. Bekooij, S. Parnar, and J. van Meerbergen, *Performance guarantees by simulation of process networks*. To appear in Scopes 2005.

[13] ETSI, Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer, ETSI TS 101 475 V1.2.2 (2001-02), 2001.

[14] Gerard K. Rauwerda, Paul M. Heysters, Gerard J.M. Smit, "Mapping Wireless Communication Algorithms onto a Reconfigurable Architecture", Journal of Supercomputing, Kluwer Academic Publishers, December 2004.

[15] Pascal T. Wolkotte, Gerard J.M. Smit, L.T. Smit, "Partitioning of a DRM receiver", Proceedings of the 9th International OFDM-Workshop, pp. 299-304, Dresden, September 2004.