Transformations for polyhedral process networks

Meijer, S.

Chapter 4

Process Merging Transformations

Recall from Chapter 3 that the partitioning strategy of the `pn` compiler may not necessarily result in PPNs that meet the performance/resource requirements. To meet the performance requirements, a designer can apply the process splitting transformation as discussed in Chapter 3. In this chapter, we introduce the process merging transformation that reduces the number of processes in a PPN. The process merging transformation is not only useful to meet the performance constraints, but also allows a designer to achieve the same performance using fewer processes in some cases. We show that many solutions exist to merge different processes in a PPN with great differences in performance results. Thus, it is not trivial to select the best merging solution. We address this issue in this chapter by presenting a compile-time solution to evaluate different merging alternatives.

4.1 Process Merging: Definitions

The process merging transformation reduces the number of processes in a PPN by sequentializing n processes in a single compound process.

Definition 16 *The **process merging** transformations takes n processes $P_1, .., P_n$ and sequentializes them into one compound process $P_{1..n}$.*

Definition 17 *A **compound process** is formed by merging n processes and executes in a sequential way the functions of the processes that are merged.*

A compound process has, therefore, the following properties:

- Per iteration of the compound process, process functions of $P_1, .., P_n$ are executed sequentially.

- The process iteration domain sizes of $P_1, .., P_n$ can be different. Then, the different process functions are executed sequentially per compound process iteration for a number of overlapping process iterations. In the remaining compound process iterations, where the process iterations do not overlap, only the process function(s) is executed of the process that has the largest number of process iterations.

- If there exists a dependency between the processes, then the `pn` compiler calculates a safe offset between the process functions in the compound process.

As a result of using the process merging transformation, less processes need to be mapped on the platform's processing elements, at the price of possibly having less processes running in parallel. A designer needs to apply the process merging transformation in case *i)* the number of processes is larger than the number of processing elements, or *ii)* the network is not well balanced and therefore the same overall performance can be achieved using less resources. For both cases, the problem is that many different options exist to merge two or more processes. The total number of options to merge different processes for a PPN with *n* processes is $\sum_{i=2}^{n} \left(\begin{array}{c} n \\ i \end{array} \right)$. To give an example for a PPN with 5 processes there are $\left(\begin{array}{c} 5 \\ 2 \end{array} \right) + \left(\begin{array}{c} 5 \\ 3 \end{array} \right) + \left(\begin{array}{c} 5 \\ 4 \end{array} \right) + \left(\begin{array}{c} 5 \\ 5 \end{array} \right) = 26$ different options to merge 2, 3, 4, or 5 processes. The challenge is how to find the best solution from all these options. To solve this problem, an analytical throughput modeling framework for Polyhedral Process Networks (PPNs) is defined in this chapter. The throughput model is used to evaluate the throughput of different process mergings in order to select the best option which gives a system throughput as close as possible to the initial PPN.

4.2 Challenges of Applying the Process Merging Transformation

With 3 motivating examples we show that selecting the best merging option is not a straightforward task as it depends on the inter-play of many factors which may not be evident at first sight. The first factor to be considered is the *workload* of a process. Recall from Chapter 2, that the workload W_{P_i} of a process P_i denotes the number of time units that are required to execute a function, i.e., the pure computational workload, excluding the communication. Figure 4.1 shows a PPN consisting of 6 processes. It is annotated with the process workload and shows the number of readings/writings from/to each FIFO channel. Process $P2$, for example, has a workload of 10 time units and a single token is read/written from/to a FIFO channel per process iteration, which is denoted by "$[1]$" and can be repeated (possibly) in-

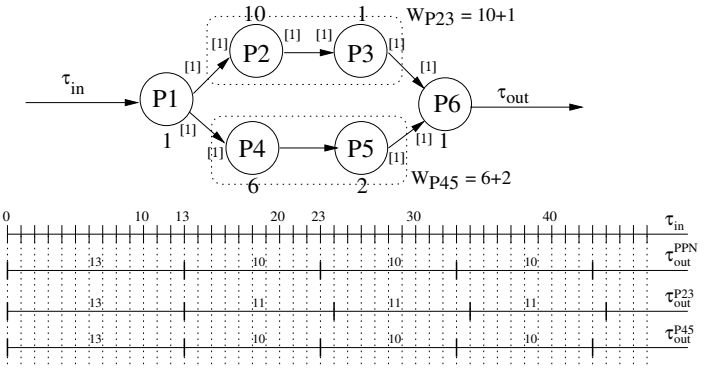

Figure 4.1: Process Workload Influencing the System Throughput

finitely many times. The network has two datapaths $DP1 = (P1, P2, P3, P6)$ and $DP2 = (P1, P4, P5, P6)$ that transfer an equal amount of tokens. We observe that process $P2$ determines the system throughput, which is illustrated with the time lines at the bottom of Figure 4.1. The first time line shows the rate τ_{in} at which tokens arrive at the network, i.e., each time unit. The second time line shows the system throughput of the initial PPN, denoted by τ_{out}^{PPN}.

Definition 18 *The **system throughput**, denoted by τ_{out}, is defined as the number of data tokens produced by the network per time unit.*

Process $P6$ needs 13 time units (1+10+1+1) to produce its first token. Then, it produces a new token each 10 cycles which is dictated by the slowest process $P2$. If we apply the process merging transformation to processes $P2$ and $P3$, then compound process $P23$ becomes the most computationally intensive process of the network. Processes $P2$ (10 time units) and $P3$ (1 time unit) are sequentialized and thus it will take 10+1=11 time units instead of 10 time units for process $P6$ to produce a new token, as shown in the time line denoted by τ_{out}^{P23}. We observe that the throughput of this network is lower than the throughput of the initial PPN. The fourth time line, denoted by τ_{out}^{P45}, shows the system throughput after merging processes $P4$ and $P5$. In this case, however, we see that the system throughput is not affected, i.e., it is the same as the throughput of the initial PPN, because the two merged and sequentialized processes do not dictate the system throughput. Thus, a designer can safely merge these processes and achieve the same system throughput as the initial PPN.

With the following example, we show that considering the process workload W_{P_i} only is not enough; a second factor that needs to be taken into account is the *rate of producing tokens*. Consider the PPN in Figure 4.2 which is topologically the same as in the previous example. The only difference is that both datapaths transfer a different

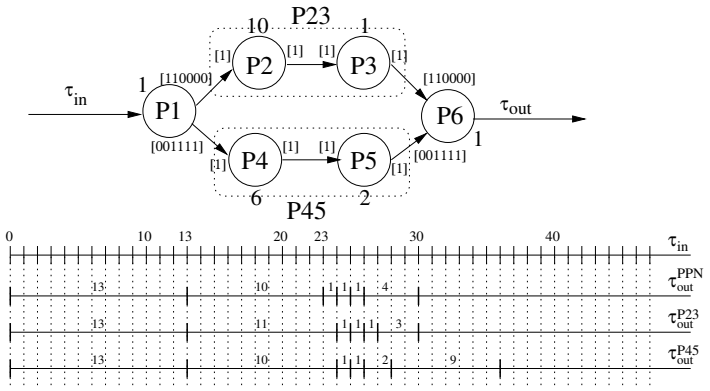

Figure 4.2: Production Rate Influencing the System Throughput

number of tokens. This is indicated with the patterns `[110000]` and `[001111]` at which process $P1$ writes to its outgoing FIFO channels. A "`1`" in these patterns indicates that data is read/written and a "`0`" that no data is read/written. So, the FIFO channel connecting $P1$ and $P2$, for example, is written the first two iterations of $P1$, but not in the remaining 4. As a consequence of these patterns, more tokens are communicated through the second datapath $DP2 = (P1, P4, P5, P6)$. Therefore, we observe that, despite process $P2$ largest workload of 10 time units, process $P4$ with a workload of 6 is more dominant. Therefore, merging processes $P4$ and $P5$ leads to a lower network throughput compared to merging $P2$ and $P3$, as can be seen in the time lines τ_{out}^{P45} and τ_{out}^{P23} in Figure 4.2. We observe a trend which is completely different from the previous example. According to Figure 4.2, a designer can safely merge processes $P2$ and $P3$ as opposed to $P4$ and $P5$ to achieve a system throughput that is equal to the throughput of the initial PPN.

In the last motivating example, we consider the PPN shown in Figure 4.3. The processes always read and/or write a single token when they are executed. Therefore, one could expect that this example is different from the example in Figure 4.2, but similar to the example in Figure 4.1. We show, however, that neither case applies and that a third factor needs to be taken into account. In this example, process $P1$ is the computationally most intensive process with a workload of 53 time units. If a designer wants to merge processes, a logical choice would be to merge $P2$ and $P3$ and not to consider the heavy process $P1$.

Processes $P2$ and $P3$ both have a workload of 25 time units and thus the compound process $P23$ has a summed workload of 50 time units, which is smaller than process $P1$ (53 time units). For this reason, we expect performance results that are equally good as the initial PPN. However, when we measure the performance results of both the initial PPN and the transformed PPN on the ESPAM platform [61], there is a

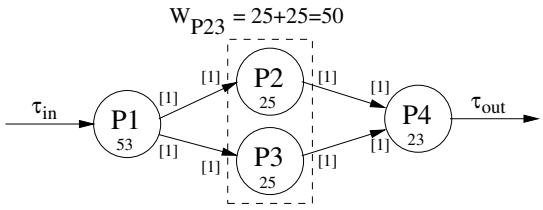

Figure 4.3: Sequentialized FIFO Accesses Influencing the System Throughput

20% degradation in the performance results. Although the workload of compound process $P23$ is lower than $P1$, the compound process reads sequentially from two input channels and writes sequentially to two output channels. This makes it the heaviest process in the network. So, besides sequential execution of the process workloads, we observe that *sequential FIFO reading/writing* is another aspect that should be taken into account.

The 3 examples above show that it is not trivial to merge processes and to achieve performance results as close as possible to the initial PPN. Therefore, we want to have a compile-time framework to evaluate the system throughput such that the best possible merging can be selected. Our compile-time framework is based on the throughput modeling techniques presented in Section 4.4.

4.3 Restrictions on the Throughput Modeling

A number of restrictions apply on the throughput model as presented in Section 4.4. First of all, we consider acyclic PPN graphs. Cycles in a PPN are responsible for sequential execution of some of the processes involved in the cycle. The sequential execution can vary from a single initial delay, to a delay at each iteration of some of the processes. For accurate throughput modeling, these cycles must be taken into account which we do not study in this work. The reason is that throughput modeling for acyclic networks is already a very difficult task, which is even more challenging for cyclic networks. There are recent works that started to investigate the performance analysis of cyclic dataflow graphs [86], but more research is required in that area in the future.

Secondly, it is important to state that our goal is not to compare different PPNs, but to compare transformed PPNs derived from a single PPN. Therefore, in the throughput modeling, we do not take into account the latency of a token, i.e., the time that elapses between injecting a token in the PPN and the time when that token leaves the PPN. Thus, we do not calculate the total execution time of PPNs, but only want to capture the throughput trend instead. The reason is that the framework should be fast,

and only as accurate as needed to correctly capture the throughput trend for different process mergings.

Thirdly, the process workload W_{P_i} and the costs for FIFO communication are parameters in our system throughput modeling. These are constant values that should be provided by the designer who can obtain them, for example, by executing the function and FIFO read/write primitives once on the target platform. The reader is referred to Section 3.6 for a discussion on the modeling of the process workload and FIFO read/write primitives with constant values. Although our approach is extensible to heterogeneous MPSoCs, we restrict ourself to MPSoCs with programmable homogeneous cores. The reason is that a process function implemented as software cannot be merged with a process function that is implemented as a hardware IP core. Similarly, one cannot merge two processes both implemented as IP cores. This means that once the process workload of a given process is determined, that this process workload value is the same for all programmable homogeneous cores in the target platform.

Finally, we do not study the effect of different buffer sizes. Although buffer sizes play an important role in the performance results, there are studies [17] showing that saturation points can be found where performance does not increase for larger buffer sizes. The `pn` compiler can find such points and we use buffer sizes that correspond to these points, i.e., the buffer sizes that give maximum performance.

4.4 Throughput Modeling

We introduce first the solution approach to model the throughput of polyhedral process networks with an example. Then, we define all concepts and steps of the throughput model in detail. Finally, we present the overall algorithm for the throughput modeling.

4.4.1 Process Throughput and Throughput Propagation

The solution approach for the overall Polyhedral Process Network (PPN) throughput modeling relies on calculating the throughput τ_{P_i} of a process P_i for all processes and propagation of the lowest process throughput to the sink processes. For a process P_i, the propagation consists of selecting either the aggregated incoming FIFO throughput $\tau_{F_{aggr}}$ or the isolated process throughput $\tau_{P_i}^{iso}$:

$$\tau_{P_i} = min(\tau_{F_{aggr}}, \tau_{P_i}^{iso}), \tag{4.1}$$

Before defining formally $\tau_{F_{aggr}}$ and $\tau_{P_i}^{iso}$ (in Sections 4.4.2 - 4.4.4), we first give an intuitive example of the solution approach applied on the PPN shown in Figure 4.3

and explain the meaning of Equation 4.1. First, the workload of each process is taken into account and let us assume that it takes 10, 20, 10, 10 time units for processes $P1, P2, P3, P4$, respectively, for executing its function. This means that, for example, $P1$ can read and produce a new token every 10 time units if there is input data. Thus, we define the isolated process throughput to be $\tau_{P1}^{iso} = \frac{1}{10}$ tokens per time units (excluding communication costs for the sake of simplicity). Similarly for the other processes, we define $\tau_{P2}^{iso} = \frac{1}{20}, \tau_{P3}^{iso} = \frac{1}{10}, \tau_{P4}^{iso} = \frac{1}{10}$. However, the required input data for a process can be delivered with a different throughput, i.e., the aggregated incoming FIFO throughput $\tau_{F_{aggr}}$. Consequently, the lowest throughput ($\tau_{F_{aggr}}$ or $\tau_{P_i}^{iso}$) determines the actual process throughput τ_{P_i}. Therefore, the minimum throughput value is selected as shown in Equation 4.1. This is repeated for all processes by iteratively applying Equation 4.1 on each process to select the lowest throughput and to propagate it to the sink processes. First, the PPN graph is topologically sorted to obtain a linear ordering of processes, e.g., $P1, P2, P3, P4$. In step I)

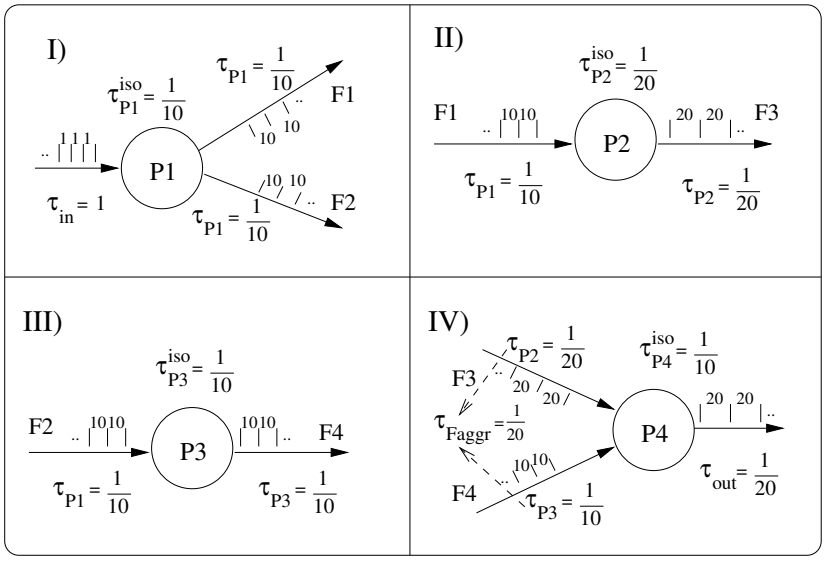

Figure 4.4: Throughput Propagation Example

of Figure 4.4, process $P1$ is the first process to be considered. While it receives tokens at each time unit ($\tau_{in} = 1$), it is ready to execute again after 10 time units due to the process workload ($\tau_{P1}^{iso} = \frac{1}{10}$). We see that the actual process throughput is determined by the process itself (it is the slowest) and Equation 4.1 is used to find this: $\tau_{P1} = min(1, \frac{1}{10}) = \frac{1}{10}$ with which it writes to both its outgoing FIFO channels $F1$ and $F2$.

If we continue with the second process in step II), we see that $P2$ receives tokens

from $P1$ with a throughput of $\tau_{P1} = \frac{1}{10}$. However, $P2$ is twice slower than $P1$ which is delivering the data: $\tau_{P2} = min(\frac{1}{10}, \frac{1}{20}) = \frac{1}{20}$. Thus, we know that $P2$ writes its results to FIFO channel $F3$ with a throughput of $\frac{1}{20}$.

In step III), we calculate the throughput for process $P3$. It receives data from $P1$ with a throughput of $\tau_{P1} = \frac{1}{10}$, and it can process data with a throughput of $\tau_{P3}^{iso} = \frac{1}{10}$. We compare what is slower by calculating $\tau_{P3} = min(\frac{1}{10}, \frac{1}{10}) = \frac{1}{10}$ and set this as the throughput at which $P3$ writes to FIFO channel $F4$.

Finally, we consider process $P4$ in step IV). Process $P4$ reads from two FIFO channels $F3$ and $F4$, which are written by $P2$ and $P3$ with different throughputs. Therefore, the FIFO throughput must be aggregated in order to have a single throughput value at which data arrives. If we assume that both channels are read per process iteration of $P4$, then the slowest FIFO throughput determines the aggregated FIFO throughput. For this example, $\frac{1}{20}$ is the slowest component and we set $\tau_{F_{aggr}} = \frac{1}{20}$. Applying Equation 4.1 shows that the data is delivered with a lower throughput than $P4$ can actually process: $\tau_{P4} = min(\frac{1}{20}, \frac{1}{10}) = \frac{1}{20}$ and set this to be the process throughput. In this way, we have propagated the slowest throughput from $P2$ to the sink process $P4$, which in the end determines the overall system throughput. In the next sections we exactly define how the (isolated) process throughput and (aggregated) FIFO throughput can be calculated.

4.4.2 Isolated Throughput of a (Compound) Process

Definition 19 *The **isolated process throughput** of a process P_i, denoted by $\tau_{P_i}^{iso}$, is the number of tokens produced by P_i per time unit when the input rate of its input data is ∞.*

We illustrate the isolated process throughput with the example shown in Figure 4.5.

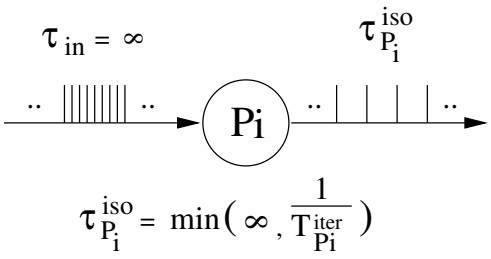

Figure 4.5: Isolated Process Throughput

We model the input data to arrive infinitely fast, i.e., $\tau_{in} = \infty$, such that the time $T_{P_i}^{iter}$ that is required for one process iteration, determines the throughput at which

tokens are produced by P_i. This means that the isolated process throughput is determined only by the workload W_{P_i} of a process and the number of FIFO reads/writes per process iteration provided that no blocking occurs:

$$\tau_{P_i}^{iso} = \frac{1}{T_{P_i}^{iter}}, \tag{4.2}$$

where $T_{P_i}^{iter}$ is the time to execute one process iteration as we have defined in Formula 3.9. It is important to note that two factors as identified in the motivating examples are taken into account in modeling the isolated process throughput: the time $T_{P_i}^{iter}$ for one process iteration includes the *process workload* W_{P_i} and also the number of *sequential FIFO accesses* (i.e., the data transfers).

In a similar way, we must also model the isolated throughput $\tau_{P_m}^{iso}$ of a compound process P_m in order to evaluate the system throughput for a PPN with merged processes. Assume that P_m is formed by merging processes P_i and P_j with iteration domains D_{P_i} and D_{P_j}, respectively. We define the isolated compound process throughput as $\tau_{P_m}^{iso} = \frac{1}{T_{P_m}^{iter}}$, where

$$T_{P_m}^{iter} = \frac{|D_{P_i}|}{|D_{P_j}|} \cdot (T_{P_i}^{iter} + T_{P_j}^{iter}) + \frac{|D_{P_j}| - |D_{P_i}|}{|D_{P_j}|} \cdot (T_{P_j}^{iter}) \tag{4.3}$$

with $|D_{P_i}| \leq |D_{P_j}|$. To model the time $T_{P_m}^{iter}$ for executing the compound process, we take into account the generated schedule of the compound process as produced by the `pn` and ESPAM tools [61, 95]. The execution of the process functions are interleaved as much as possible. This means that per iteration of the compound process, all functions are sequentially executed if this is allowed by the inter-process dependencies. In case of inter-process dependencies, an offset is calculated for the producer-consumer pair to ensure correct program behavior, and then the function execution is interleaved again. Therefore, we calculate fractions where the execution of the functions overlap and multiply it with the process iteration costs of these functions, i.e., the first term in Equation 4.3. And then we consider for the remaining iterations the cost of the process with the largest domain size only, i.e., the second term in Equation 4.3. Note that the coefficients in Equation 4.3 represent these fractions which should sum up to 1. Formula 4.4 below shows how $T_{P_m}^{iter}$ is calculated when n process are merged into a compound process P_m.

$$T_{P_m}^{iter} = \frac{|D_1|}{|D_n|} \cdot (\sum_{i=1}^{n} T_i^{iter}) + \sum_{j=2}^{n} \left(\frac{|D_j| - |D_{j-1}|}{|D_n|} \cdot (\sum_{k=j}^{n} T_k^{iter}) \right) \tag{4.4}$$

where the different process iteration domains have been sorted and renumbered according to their domain sizes, i.e., $D_1 \leq .. \leq D_{i-1} \leq D_i \leq D_{i+1} \leq .. \leq D_n$.

4.4.3 FIFO Channel Throughput

The throughput of a FIFO-channel is determined by the throughput of the processes accessing it. Let us consider the example shown in Figure 4.6. Assume that $P1$ executes 500 times, i.e., $|D_{P1}| = 500$, and each time it writes to $F1$ and $F2$.

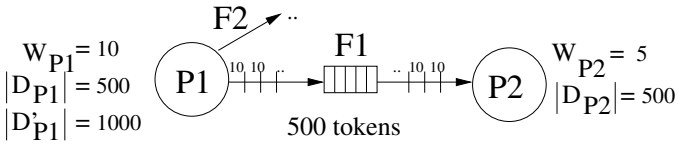

Figure 4.6: FIFO Channel Throughput

Process $P1$ needs 10 time units to produce a token. Consumer process $P2$ is twice as fast and needs only 5 time units to consume a token, but still it receives tokens only each 10 time units due to the slower producer. As a result, $P2$ blocks on reading and waits for data, which follows the operational semantics of the PPN model of computation: a process stalls if it tries to read from an empty FIFO channel and proceeds only if data is available again. This example shows that, to calculate the FIFO throughput τ_{f_i} of a FIFO channel f_i, the minimum is taken of the FIFO write throughput $\tau_{f_i}^{Wr}$ and the FIFO read throughput $\tau_{f_i}^{Rd}$:

$$\tau_{f_i} = min(\tau_{f_i}^{Wr}, \tau_{f_i}^{Rd}), \tag{4.5}$$

where $\tau_{f_i}^{Wr} = \tau_{P1}$ (see Equation 4.1) and $\tau_{f_i}^{Rd} = \tau_{P2}^{iso}$ (see Equation 4.2). Let us consider another example where $P1$ executes 1000 times, i.e., $|D'_{P1}| = 1000$ as also shown in Figure 4.6. Assume that in one iteration of $P1$ data is written to FIFO channel $F1$, and in the next iteration to $F2$. This is repeated such that in total 500 tokens are written to both FIFOs $F1$ and $F2$. To compensate for a producer that does not write data to a FIFO channel at each iteration, we define a coefficient that divides the total number of tokens transfered over a channel by the iteration domain size of a producer process P_i. This coefficient denotes an average production rate, expressed in a number of producer iteration points. Note that this takes into account the different *production rates* of processes as also identified in the motivating example in Figure 4.2. By multiplying this coefficient with the process throughput, we define FIFO write/read throughput $\tau_{f_i}^{Wr}$ and $\tau_{f_i}^{Rd}$ of a FIFO channel f_i as shown

in Equations 4.6 and 4.7. In this way, we model a lower throughput if necessary.

$$\tau_{f_i}^{Wr} = \frac{|OP_{P_i}^j|}{|D_{P_i}|} \cdot \tau_{P_i} \tag{4.6}$$

$$\tau_{f_i}^{Rd} = \frac{|IP_{P_i}^j|}{|D_{P_i}|} \cdot \tau_{P_i}^{iso}, \tag{4.7}$$

For the example, we see that $\tau_{f1}^{Wr} = \frac{500}{1000} \cdot \frac{1}{10} = \frac{1}{20}$ and the FIFO read throughput is $\tau_{f1}^{Rd} = \frac{500}{500} \cdot \frac{1}{5} = \frac{1}{5}$. Consequently, the FIFO throughput is $\tau_{f1} = min(\frac{1}{20}, \frac{1}{5}) = \frac{1}{20}$ tokens per time unit.

4.4.4 Aggregated FIFO Throughput

The throughput of a process τ_{P_i} is either determined by the FIFO throughput from which it receives its data, i.e., $\tau_{F_{aggr}}$, or by the computational workload of the process itself, i.e., $\tau_{P_i}^{iso}$, as shown in Equation 4.1. $\tau_{P_i}^{iso}$ is computed with Equation 4.2. Here we show how to compute $\tau_{F_{aggr}}$, which deals with the problem how to model the throughput of data in case there are multiple incoming FIFO channels. This is illustrated with the example in Figure 4.7.

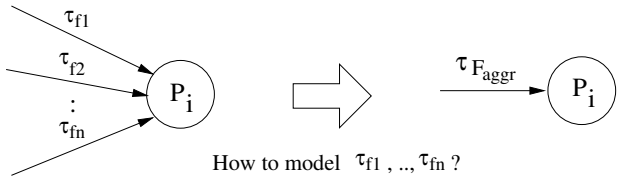

Figure 4.7: Modeling Multiple Incoming FIFO Channels

Process P_i has n incoming FIFO channels each with its own throughput. We need to model these different incoming FIFO channel throughputs as one throughput value, i.e., $\tau_{F_{aggr}}$, because we must determine what is slower: the arrival of the input data or the process itself. The throughput of the incoming FIFO channels are aggregated according to the way the process function input arguments are read.

To illustrate the calculation of the aggregated FIFO throughput, let us first consider Process P in Figure 4.8, which has **one** input argument value a that is read from two different input ports $IP1$ and $IP2$. Thus, two tokens are delivered, but only one is read for each iteration of the consumer process. The other token will be read in another iteration. To model the throughput at which data arrives, the sum is taken of the FIFO throughput $F1$ and $F2$, i.e., $\tau_{F_{aggr}} = \tau_{f_1} + \tau_{f_2}$. Effectively, this means that

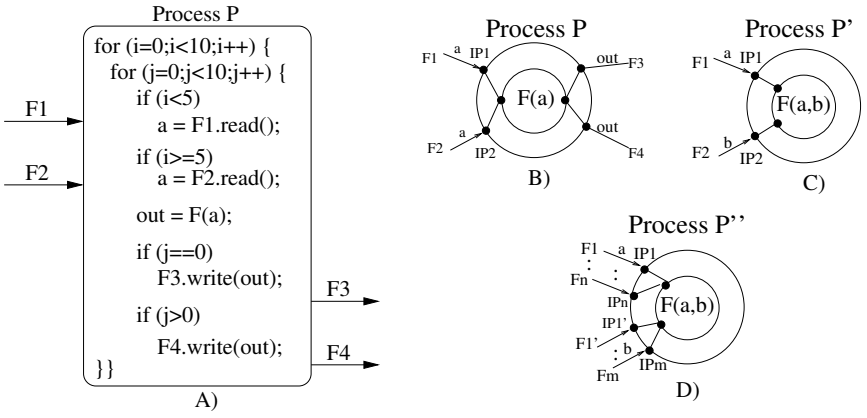

Figure 4.8: Process Structure (left) and FIFO Throughput Aggregation (right)

the aggregated incoming FIFO throughput becomes higher, which corresponds to the behavior that one token is needed but two are delivered. Note that any imbalance in the number of tokens transfered over each FIFO channel has already been taken into account in the FIFO read/write throughput as defined in Equation 4.6 and 4.7.

Process P' in Figure 4.8 is the second example, which reads its **two** input arguments values a and b from FIFOs $F1$ and $F2$. Thus, both FIFOs are read per process iteration of P'. If one FIFO throughput is fast and the other one is slower, then the slowest FIFO throughput determines the aggregated FIFO throughput. Therefore, we select the minimum throughput in this case, i.e., $\tau_{F_{aggr}} = min(\tau_{f_1}, \tau_{f_2})$.

Finally, the general case is illustrated with process P'' in Figure 4.8, i.e., it combines the previous two examples. Process P'' has multiple function input arguments and multiple incoming FIFO channels per input argument. To calculate the aggregated FIFO throughput, the throughput is summed of all the FIFO channels that are connected to one function input argument (the first example). Next, the minimum throughput, i.e., the slowest throughput, is taken of all the throughputs for the different function input arguments (the second example). Thus, the aggregated FIFO throughput $\tau_{F_{aggr}}$ for P'' is calculated as follows:
$\tau_{F_{aggr}} = min(\tau_{f_1} + .. + \tau_{f_n},\ \tau'_{f_1} + .. + \tau'_{f_m})$.

The general formula to calculate the aggregated FIFO throughput $\tau_{F_{aggr}}$ is given below:

$$\tau_{F_{aggr}} = min(\sum_{i=1}^{n} \tau_{f_i}, ..., \sum_{j=1}^{m} \tau_{f_j}) \tag{4.8}$$

where each sum corresponds to the sum of the throughputs of a number of FIFO channels connected to *one* process function input argument. Thus, the first term sums the throughput τ_{f_i} of n different FIFO channels connected to one process function input argument, and the last term sums the throughput τ_{f_j} of m different FIFO channels connected to another process function input argument. Finally, the minimum is taken to determine the slowest FIFO throughput.

4.4.5 System Throughput Calculation Algorithm

Up to now, we have formally defined all the components that allow the throughput calculation and propagation to be done in a systematic and automated way. The pseudo code of the throughput calculation and propagation algorithm is shown in Algorithm 1.

Algorithm 1 : PPN Throughput Estimation Pseudo-code

Require: PPN: a Polyhedral Process Network
Require: W_{P_i}: the computational workload of all processes.
Require: $C^{Rd,Wr}_{intra,inter}$: the costs for the FIFO read/write primitives.
$list \leftarrow$ Create topological ordering for PPN
for all process $P_i \in list$ **do**
1) Calculate $\tau^{iso}_{P_i} = set\_isolated\_throughput(P_i, W_{P_i}, C^{Rd,Wr}_{intra,inter})$
2) Set $\tau^{Rd}_{f_j}$ for all incoming FIFOs f_j of P_i.
3) Set τ_{f_j} for all incoming FIFOs f_j of P_i.
4) Calculate $\tau_{F_{aggr}} = calc\_fifo\_aggr(\tau_{f_j}, .., \tau_{f_n})$
5) Set $\tau_{P_i} = min(\tau^{iso}_{P_i}, \tau_{F_{aggr}})$
6) Set $\tau^{Wr}_{f_j}$ for all outgoing FIFO f_j of P_i.
end for
return $\tau^{PPN}_{out} = \tau_{P_{|list|}}$

This algorithm was introduced informally with the example in Section 4.4.1. Here we give the formal solution by applying Algorithm 1 on this example. All steps of Algorithm 1 are shown in Figure 4.9. The example PPN in Figure 4.3 consists of 4 processes and thus we obtain first a topologically ordered list of 4 processes, i.e., $list = \{P1, P2, P3, P4\}$. For each of these processes, we calculate the throughput at which the incoming data arrives, how fast a process can actually process this data, and the slowest value is propagated to the outgoing FIFO channels. The most interesting steps are $4.2.1 - 4.4$ in Figure 4.9, because the throughput of FIFO channels $F3$ and $F4$ are aggregated. Process $P4$ needs input tokens from both channels for each of its process iterations. Since the slowest FIFO throughput determines the aggregated FIFO throughput, the minimum FIFO throughput is selected in step 4.4.

$W_{P1} = W_{P3} = W_{P4} = 10,\ W_{P2} = 20$
$C^{Rd} = C^{Wr} = 0$

0 $list = \{P1, P2, P3, P4\}$
1.1 $\tau_{P1}^{iso} = \frac{1}{10}$
1.2 $\tau_{f_{in}}^{Rd} = \infty$
1.3 $\tau_{f_{in}} = \infty$
1.4 $\tau_{F_{aggr}} = \infty$
1.5 $\tau_{P1} = min(\frac{1}{10}, \infty) = \frac{1}{10}$
1.6.1 $\tau_{F1}^{Wr} = \frac{1}{10}$
1.6.2 $\tau_{F2}^{Wr} = \frac{1}{10}$

2.1 $\tau_{P2}^{iso} = \frac{1}{20}$
2.2 $\tau_{F1}^{Rd} = \frac{1}{20}$
2.3 $\tau_{F1} = min(\tau_{F1}^{Wr}, \tau_{F1}^{Rd}) = \frac{1}{20}$
2.4 $\tau_{F_{aggr}} = min(\frac{1}{20}) = \frac{1}{20}$
2.5 $\tau_{P2} = min(\frac{1}{20}, \frac{1}{20}) = \frac{1}{20}$
2.6 $\tau_{F3}^{Wr} = \frac{1}{20}$

3.1 $\tau_{P3}^{iso} = \frac{1}{10}$
3.2 $\tau_{F2}^{Rd} = \frac{1}{10}$
3.3 $\tau_{F2} = min(\tau_{F2}^{Wr}, \tau_{F2}^{Rd}) = \frac{1}{10}$
3.4 $\tau_{F_{aggr}} = min(\frac{1}{10}) = \frac{1}{10}$
3.5 $\tau_{P3} = min(\frac{1}{10}, \frac{1}{10}) = \frac{1}{10}$
3.6 $\tau_{F4}^{Wr} = \frac{1}{10}$

4.1 $\tau_{P4}^{iso} = \frac{1}{10}$
4.2.1 $\tau_{F3}^{Rd} = \frac{1}{10}$
4.2.2 $\tau_{F4}^{Rd} = \frac{1}{10}$
4.3.1 $\tau_{F3} = min(\tau_{F3}^{Wr}, \tau_{F3}^{Rd}) = \frac{1}{20}$
4.3.2 $\tau_{F4} = min(\tau_{F4}^{Wr}, \tau_{F4}^{Rd}) = \frac{1}{10}$
4.4 $\tau_{F_{aggr}} = min(\frac{1}{10}, \frac{1}{20}) = \frac{1}{20}$
4.5 $\tau_{P4} = min(\frac{1}{20}, \frac{1}{10}) = \frac{1}{20}$
4.6 $\tau_{out}^{PPN} = \tau_{P4} = \frac{1}{20}$

Figure 4.9: Throughput Calculation

In this way, we have propagated the slowest throughput of process $P2$ to the sink process, which determines in the end the overall system throughput.

4.5 Case-Studies

In this section we map two different nested loop kernels on the ESPAM platform prototyped on a Xilinx Virtex 2 Pro FPGA. Each process is mapped one-to-one on a MicroBlaze softcore processor and the processors are point-to-point connected. FIFO communication is implemented with FSL links and a FIFO access costs 10 clock cycles. We investigate if our throughput modeling captures the differences in performance results for different process merging configurations and process workloads.

4.5.1 Merging Light-Weight Producers

In the first experiment, we merge two light-weight producers (workload of 54 time units) into a single process, and we should observe that the new compound process does not become the process that determines the system throughput, i.e., the through-

put of the PPNs before and after the process merging are the same. Then, we increase the workload of the producers to 59 time units such that we intentionally introduce a new bottleneck in the PPN. The throughput of the PPN after the process merging should be less than the initial PPN, and we test whether this is captured by our throughput model.

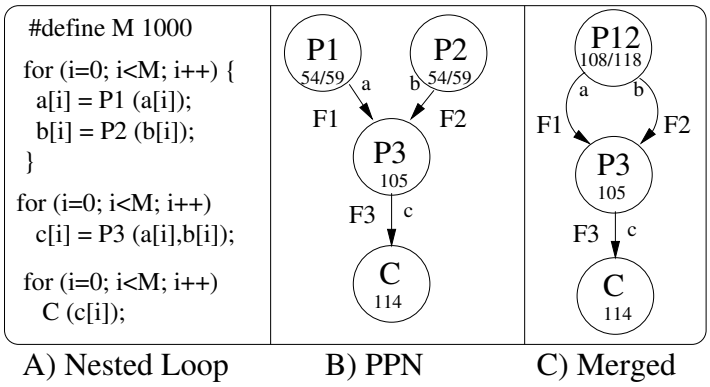

Figure 4.10: Example PPN

Figure 4.10 shows the nested loop program in A), the derived PPN in B), and the PPN with producers $P1$ and $P2$ merged in C). We calculate the throughput of the PPN before and after merging by applying Algorithm 1.

Figure 4.11 shows the analysis for process $P1, P2, P3$ and C. In process $P3$, two FIFO throughput values are aggregated as shown in step 3.4 of the throughput calculation in Figure 4.11. We find a process throughput of $\tau_{P3} = \frac{1}{135}$ for process $P3$, which is propagated to C such that the system throughput is $\tau_{out}^{PPN} = \tau_C = \frac{1}{135}$ as well.

Next, we calculate the system throughput for the PPN with processes $P1$ and $P2$ merged into one compound process. The throughput calculation is shown in Figure 4.12, and thus we find a system throughput of $\tau_{out}^{PPN'} = \frac{1}{135}$. Since we find a throughput of $\tau_{out} = \frac{1}{135}$ for both PPNs before and after merging, we predict that the initial PPN and transformed PPN$'$ perform equally well. This is confirmed by the actual measured performance results shown in Figure 4.13. That is, the first and second bar in Figure 4.13 denote the cycle numbers for the initial PPN and transformed PPN$'$, which are the same.

Then we increase the workload of the producer processes and intentionally create a compound process that is the most compute intensive process. We check if this is captured by our throughput model by analyzing the throughput of the PPNs before and after the merging. The throughput model gives a throughput of $\frac{1}{135}$ and $\frac{1}{138}$

0 $list = \{P1, P2, P3, C\}$

1.1 $\tau_{P1}^{iso} = \frac{1}{54+0+10} = \frac{1}{64}$
1.2 $\tau_{f_{in}}^{Rd} = \infty$
1.3 $\tau_{f_{in}} = \infty$
1.4 $\tau_{F_{aggr}} = \infty$
1.5 $\tau_{P1} = min(\frac{1}{64}, \infty) = \frac{1}{64}$
1.6 $\tau_{F1}^{Wr} = \frac{1000}{1000} \cdot \frac{1}{64} = \frac{1}{64}$

2.1 $\tau_{P2}^{iso} = \frac{1}{54+0+10} = \frac{1}{64}$
2.2 $\tau_{f_{in}}^{Rd} = \infty$
2.3 $\tau_{f_{in}} = \infty$
2.4 $\tau_{F_{aggr}} = \infty$
2.5 $\tau_{P2} = min(\frac{1}{64}, \infty) = \frac{1}{64}$
2.6 $\tau_{F2}^{Wr} = \frac{1000}{1000} \cdot \frac{1}{64} = \frac{1}{64}$

3.1 $\tau_{P3}^{iso} = \frac{1}{105+(2\cdot10)+10} = \frac{1}{135}$
3.2.1 $\tau_{F1}^{Rd} = \frac{1000}{1000} \cdot \frac{1}{135}$
3.2.2 $\tau_{F2}^{Rd} = \frac{1000}{1000} \cdot \frac{1}{135}$
3.3.1 $\tau_{F1} = min(\frac{1}{64}, \frac{1}{135}) = \frac{1}{135}$
3.3.2 $\tau_{F2} = min(\frac{1}{64}, \frac{1}{135}) = \frac{1}{135}$
3.4 $\tau_{F_{aggr}} = min(\frac{1}{135}, \frac{1}{135}) = \frac{1}{135}$
3.5 $\tau_{P3} = min(\frac{1}{135}, \frac{1}{135}) = \frac{1}{135}$
3.6 $\tau_{F3}^{Wr} = \frac{1000}{1000} \cdot \frac{1}{135} = \frac{1}{135}$

4.1 $\tau_{C}^{iso} = \frac{1}{114+10+0} = \frac{1}{124}$
4.2 $\tau_{F3}^{Rd} = \frac{1000}{1000} \cdot \frac{1}{124} = \frac{1}{124}$
4.3 $\tau_{F3} = min(\frac{1}{135}, \frac{1}{124}) = \frac{1}{135}$
4.4 $\tau_{F_{aggr}} = \frac{1}{135}$
4.5 $\tau_{C} = min(\frac{1}{135}, \frac{1}{124}) = \frac{1}{135}$
4.6 $\tau_{out}^{PPN} = \tau_{C} = \frac{1}{135}$

Figure 4.11: Throughput Estimation of Processes $P1, P2, P3, C$ in Figure 4.10 B)

for the initial and transformed PPN, respectively. Thus, the throughput calculation indicates that the throughput of the merged PPN is lower, which is confirmed by the third and fourth bar in the measured performance results in Figure 4.13.

0 $list = \{P12, P3, C\}$
1.1 $\tau^{iso}_{P12} = \frac{1}{54+54+0+2\cdot10} = \frac{1}{128}$
1.2 $\tau^{Rd}_{f_{in}} = \infty$
1.3 $\tau_{f_{in}} = \infty$
1.4 $\tau_{F_{aggr}} = \infty$
1.5 $\tau_{P12} = min(\frac{1}{128}, \infty) = \frac{1}{128}$
1.6.1 $\tau^{Wr}_{F1} = \frac{1000}{1000} \cdot \frac{1}{128} = \frac{1}{128}$
1.6.2 $\tau^{Wr}_{F2} = \frac{1000}{1000} \cdot \frac{1}{128} = \frac{1}{128}$
2.1 $\tau^{iso}_{P3} = \frac{1}{105+2\cdot10+1\cdot10} = \frac{1}{135}$
2.2.1 $\tau^{Rd}_{F1} = \frac{1000}{1000} \cdot \frac{1}{135} = \frac{1}{135}$
2.2.2 $\tau^{Rd}_{F2} = \frac{1000}{1000} \cdot \frac{1}{135} = \frac{1}{135}$
2.3.1 $\tau_{F1} = min(\frac{1}{128}, \frac{1}{135}) = \frac{1}{135}$
2.3.2 $\tau_{F2} = min(\frac{1}{128}, \frac{1}{135}) = \frac{1}{135}$
2.4 $\tau_{F_{aggr}} = min(\frac{1}{135}, \frac{1}{135}) = \frac{1}{135}$
2.5 $\tau_{P3} = min(\frac{1}{135}, \frac{1}{135}) = \frac{1}{135}$
2.6 $\tau^{Wr}_{F3} = \frac{1000}{1000} \cdot \frac{1}{135} = \frac{1}{135}$

3.1 $\tau^{iso}_{C} = \frac{1}{114+10+0} = \frac{1}{124}$
3.2 $\tau^{Rd}_{F3} = \frac{1000}{1000} \cdot \frac{1}{124} = \frac{1}{124}$
3.3 $\tau_{F3} = min(\frac{1}{135}, \frac{1}{124}) = \frac{1}{135}$
3.4 $\tau_{F_{aggr}} = \frac{1}{135}$
3.5 $\tau_{C} = min(\frac{1}{135}, \frac{1}{124}) = \frac{1}{135}$
3.6 $\tau^{PPN}_{out} = \tau_{C} = \frac{1}{135}$

Figure 4.12: Throughput Estimation after merging $P1$ and $P2$

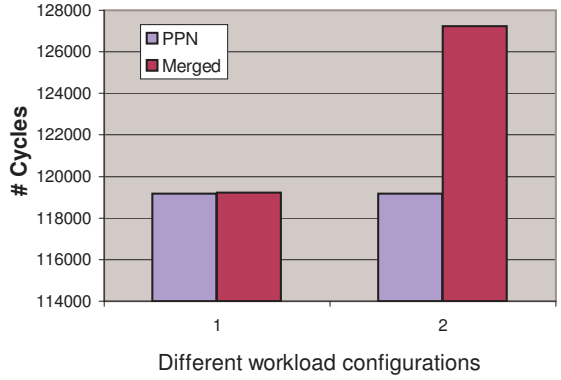

Figure 4.13: Measured Performance Results Before/After Merging $P1$ and $P2$

4.5.2 Merging Processes in Networks with Different Data Paths

In this experiment we consider the more complicated network shown in Figure 4.14 that combines different properties. First of all, it has processes with different domain sizes. Processes $P1$ and $P2$ execute 500 times, while the other processes execute 1000 times. As a result, coefficients will scale down the $F1$ and $F2$ FIFO read throughput. Second, two data paths come together in process $P3$ where one token

is needed per iteration of $P3$ similar to the example in Figure 4.8 B). Third, in process $P6$ two datapaths are joined as well where both tokens are needed for each iteration, similar to the example in Figure 4.8 C). We estimate the system through-

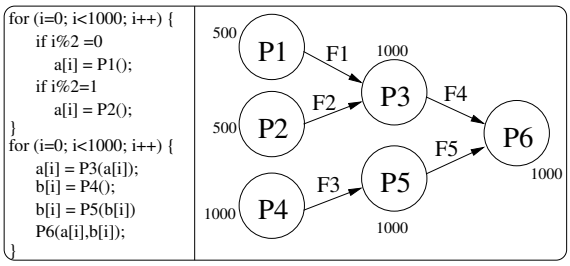

Figure 4.14: Nested-loop Program and its Derived PPN

put by applying Algorithm 1 again and test the throughput modeling with 3 different process workload configurations. Each configuration is a tuple where the first value corresponds to the workload of process P1, the 2nd value to workload of P2, etc. Figure 4.15 shows the measured performance results and for each configuration the initial PPN in Figure 4.14 is used as a reference (the first bar) and different mergings are shown in the 2nd, 3rd and 4th bars. For example, the second bar denotes the performance results after merging processes P1, P2 and P3. If we take the 2nd workload configuration as an example, our model finds the following throughputs: $\frac{1}{65}, \frac{1}{100}, \frac{1}{65}, \frac{1}{80}, \frac{1}{75}$. Thus, the estimation indicates that the first merging (i.e., $\frac{1}{100}$), leads to a lower throughput than the initial PPN (i.e., $\frac{1}{65}$). The second merging ($\frac{1}{65}$) gives the same performance results, and the third ($\frac{1}{80}$) and fourth ($\frac{1}{75}$) are worse than the initial PPN. From these estimations, we conclude that processes $P2$ and $P4$ can be merged and achieve the same system throughput. This estimation is correct as confirmed by the actual measured performance results shown in Figure 4.15.

4.6 Discussion and Summary

We have presented a solution approach for throuhgput modeling of Polyhedral Process Networks (PPNs) to evaluate process merging transformations. Our approach takes into account all major factors that influence the throughput. Therefore, we can accurately capture the throughput trend and select the best possible merging as illustrated with the experiments.

The throughput model defined in this chapter, requires the cost estimations of the process workloads and the FIFO communication primitives, similar to the process splitting transformation. Therefore, the same remark with respect to the modeling of the workload and FIFO communication with a constant value should be taken into

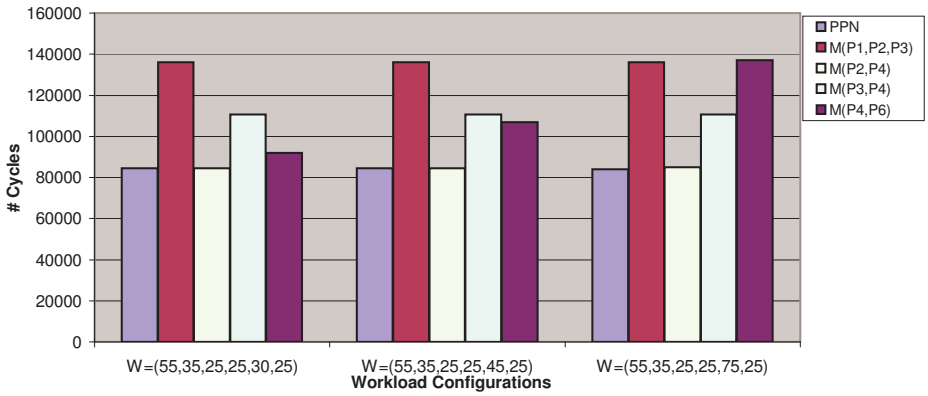

Figure 4.15: Measured Results on the ESPAM Platform

account. For an in-depth discussion, the reader is referred to Section 3.6.

Our throughput model calculates an average throughput for a given PPN, i.e., we do not take into account the dynamic behavior how output tokens are produced. This is best illustrated with the coefficient used in Formula 4.6 to determine the FIFO write throughput: the number of tokens written to a FIFO channel is divided by the total number of process iterations. However, the calculation of average throughput values allows efficient evaluation of the process merging transformations on the ESPAM platform, for two reasons. First, recall from Section 4.3 that the process workload is the same for all programmable cores in the target platform, i.e., we use a homogeneous MPSoC and assign the processes one-to-one to the cores. Second, also recall that we use buffer sizes that give maximum performance, which are calculated by the `pn` compiler. This is different in the work of [86], where the workload of a processor can vary as multiple processes can be assigned to that processor. To estimate buffer sizes and/or the system performance in this case, the dynamic behavior of the platform and application are important. In Section 1.3, we have indicated that this dynamic behavior is captured with maximum and minimum values of arrival/service curves. This throughput calculation is more complex than our approach, which we do not need for evaluating the process merging transformation on the ESPAM platform, because we assign the processes one-to-one and use buffer sizes that give maximum performance.