

Stream-processing pipelines: processing of streams on multiprocessor architecture

Nikolay Kavaldjiev, Gerard J. M. Smit, Pierre G. Jansen

Department of EEMCS, University of Twente, the Netherlands
 {nikolay, smit, jansen}@cs.utwente.nl

Abstract—In this paper we study the timing aspects of the operation of stream-processing applications that run on a multiprocessor architecture. Dependencies are derived for the processing and communication times of the processors in such a system. Three cases of real-time constrained operation and four cases of communication organization are considered and compared. Examples of application are given for the derived results.

Index Terms—multiprocessor architectures, stream-processing

I. INTRODUCTION

For decades multiprocessor architectures have been studied and used for building high-performance computing platforms. In the recent years the advances in technology made it possible to build single-chip multiprocessor systems also known as MultiProcessor System-on-Chip (MPSoC). These systems can be found in number of consumer products nowadays.

Together with their advantages multiprocessor systems bring some difficulties that designers have to manage with. One of them is the application representation. Since their architecture is parallel, the traditional sequential program description does not suit that purpose.

The most often used parallel representation for applications is a Kahn based process network [1], which is a directed graph with nodes representing sequential processes and edges representing communications between the processes. After the application has been represented as a process graph the next step towards a running application is the application mapping. Mapping is a procedure that assigns every node in the process graph to a processor in the architecture and every edge in the process graph to a communication channel in the architecture. When this step is done the application is ready to run.

Many processing-intensive applications typical for the new MPSoCs deal with processing of streaming data. These are, for example: the base-band processing for wireless digital communications, where a stream of data coming from an antenna's analogue front-end is processed in order to extract the pay-load; the same process is carried on in reverse direction in the sending part; audio/video applications where a

stream of data is processed in order to be decoded and played or to be coded and stored etc.

We speak about a data stream when a data source delivers portions of data continuously and regularly (in a fixed or variable period of time). The portions of data can be of different size and we usually refer to them as stream items.

From our experience with applications for base-band processing in digital wireless communications, which includes HyperLAN2, UMTS, Bluetooth, [2], [3], [4], we observe that this application can be represented through a process graph of general form given in *Figure 1*. A similar observation for media processing applications is made in [5].

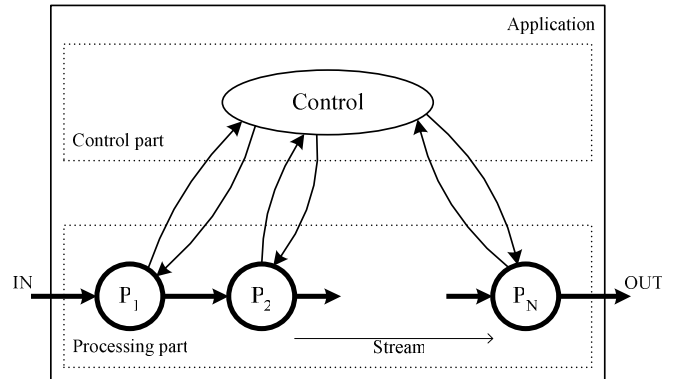


Figure 1: A general process graph of a stream-processing application

Two parts can be distinguished in process graph shown in *Figure 1*: a *stream-processing part* and a *control part*. The *stream-processing part*, given in bold in the figure, comprises the most computationally intensive processes and the most communication intensive edges of the graph and usually works under real-time constraints. It does the actual processing on the data stream. The processes there (P_1, P_2, \dots, P_N in the figure) are arranged in a chain graph. The first process in the chain receives the input stream units and the last process sends out the output stream units. After has entered the chain, a stream unit follows it through to its output as it is being processed at each node. The processes in the chain graph represent computational kernels that perform specific algorithms, like FFT, DCT, FIR, that are applied to the stream items.

We think, this typical chain form of the process graph can

probably be explained by the application design process and the sequential nature of the human thinking.

The *control part* of the application process graph comprises all the processes dealing with application organization and control, run-time adaptation and reconfiguration; in the figure they are represented by a single node named “Control”. Because of the reactive nature of these processes we expect them to run not so often, to require typically light computation and to connect in a process graph of no typical topology. We also expect the communications in this part of the application to be much less intensive than in the processing part and without or with weak real-time constraints.

The control part is connected by communication edges to all nodes in the processing part. These communications are optional and depend on the application; some of them might be missing or there might be applications without a control part. Typical functions of these communications are monitoring and control of the processing part in order the application to react and adapt to events or conditions. For example, in a base-band processing application the processing part generates an event “a bit error rate threshold is passed” to which the control part reacts with “change of the data coding scheme”.

We envision the data traffic between the control part and the processing part of the application to consists mostly of event and state exchange and of light control streams, and so to be of weak intensity, not so regular in time and with real-time constrains that are easy to fulfil.

Thus we can conclude that in many stream-processing applications we can separate the heavy computation and communications and represent them through a process graph of simple chain form. This part of the application, which later we refer to as a stream-processing part, usually works under real-time constraints and forms the main part of the system computation and communication load.

In this paper we focus on the stream-processing part of the applications as we are interested in the timing aspects of its operation when run on a multiprocessor platform.

The rest of the paper is organized as follows. *Section II* gives details about the multiprocessor architecture and its operation. *Section III* introduces the timing characteristics we are interested in. *Section IV* presents an example of a stream processing application running on a multiprocessor platform. *Section V* discusses the real-time constraints on the application’s operation we consider. *Section VI* presents the four different cases of communications organisation we consider. *Section VII* introduces the notation used during the later derivations. In *Section VIII* we derive the time dependencies for each of the considered cases. In *Section IX* we discuss the derived results.

II. STREAM PROCESSING PIPELINES

When the stream-processing part of an application runs on a multiprocessor platform its operation reminds very much a pipeline where each stage of the pipeline is a processor.

If the stream-processing part of the application from *Figure*

1 is mapped and run on a general multiprocessor architecture we have a situation like that shown in *Figure 2*. We assume that each process (kernel) from the application process graph is mapped on a separate processing unit (PU) and all inter-process communications are mapped on separate communication channels provided by the communication infrastructure of the platform.

A PU could be a processor, an application specific IP core, reconfigurable processing tile, etc. We assume each PU has its own local data and code memory and is capable of working independently from the others. A PU behaves as follows: (i) the processing starts after the data item has been received and data item is sent after it has been processed; (ii) when a PU is ready to receive, but the next data item does not arrive the PU blocks (waits) until the receive is done; (iii) when the PU is ready to send, but the data cannot be accepted by the communication channel the PU blocks (waits) until the send is done.

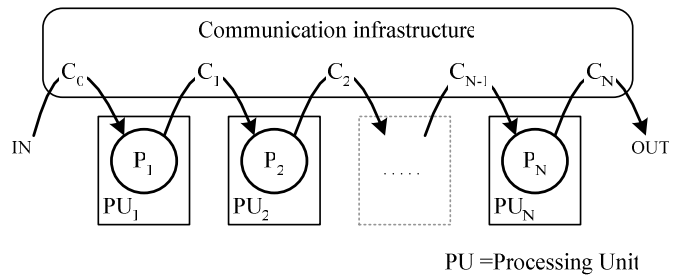


Figure 2: Stream-processing part of an application running on a multiprocessor architecture

For communication infrastructure we assume any type of message passing communication architecture that is able to provide throughput guarantees for its communication channels.

Since the processed data are stream items that arrive regularly at the pipeline input, the pipeline’s PUs run periodically, passing cyclically through four phases of operation: *receive*, *process*, *send* and *wait*. Each of these phases is characterised by its start time and duration and together they form the PU’s operational cycle.

As we shall see, the duration of the phases is a subject of choice during the mapping procedure. In this paper we study what are the dependencies and limits that drive and constrain that choice.

III. CONTROL ON THE COMMUNICATION AND PROCESSING TIMES IN THE PIPELINE

In general, the time for processing a data item in a pipeline stage and the time for transmitting a data item between two stages are not predetermined but are, to a certain extend, subject of choice during the application mapping.

Since a message-passing communication mechanism is assumed here, the time for transmission of data between two consecutive stages is in fact the send time for the one and the receive time for the other stage. Thus, in order to simplify the notation, we do not make a distinction between send and

receive times, but refer to them in general as communication times.

The time for communication, C , between two consecutive stages depends on the amount of transmitted data (data item size), L , and the throughput, S , guaranteed by the communication channel for this data:

$$C = \frac{L}{S} \quad (1)$$

where:

- L – amount of data [bit]
- S – channel throughput [bit/s]
- C – time for communication [s]

(The delay of the communication channel is neglected here. In a typical case it is a constant additive component to the communication time, much smaller than the latency component. Including the delay will lead to a subtle shift of the pipeline's phases, but will not change the time dependencies between them.)

The amount of data, L , is determined by the application and cannot be changed. The throughput, S , is selected during the application mapping by the mapping procedure. Depending on the communication needs of the application and on the currently available system communication resources the mapping procedure assigns to each process graph edge a communication channel with certain Quality-of-Service parameters (including throughput guarantees).

Thus the communication time, C , is decided during the application mapping. Since the maximal throughput of a channel, S_{max} , is limited by the channel capacity, the minimal communication time is also limited.

We assume to have a communication infrastructure that provides throughput guarantees [6] and thus upper bound on the communication time C .

The processing time, P , depends on the number of cycles it takes the PU to process a data item of size, L , and on the frequency, F , at which the PU works:

$$P = \frac{f(L)}{F} \quad (2)$$

where:

- L – amount of data [bit]
- $f(L)$ – number of computational cycles
- F – operating frequency [Hz]
- P – processing time in [s]

The function $f()$ gives the number of cycles needed for a PU to process a certain amount of data. Generally, it is an algorithm complexity function that depends on the algorithm itself and on the PU type. The PU on which a process will run is decided by the mapping procedure during the application mapping and so this choice influences the processing time.

“Programmable operating frequencies” is a feature often attributed to future MPSoC. It allows programming the clock frequency for each PU in the system thus adapting the system processing power to the application needs and saving energy. If this feature is available, the selection of the PU operating

frequency, F , is done most probably also by the mapping procedure together with the PU selection. Hence, the processing time, P , is also decided during the application mapping.

Since a choice has to be made about all processing and communication times in the pipeline, we must clear what are the constraints on that choice. Beyond the physical limits, like maximal channel throughput, restricted set of PU types, minimal and maximal operating frequency, there are also time-limits on the pipeline operation that must be taken into consideration.

IV. HYPERLAN2 EXAMPLE

Here we use a HyperLAN2 receiver as an example of a stream-processing application. HyperLAN2 [7] is a WLAN standard, based on Orthogonal Frequency Division Multiplexing (OFDM), similar to the IEEE802.11a WLAN standard.

Our intention is to run the HyperLAN2 receiver on a multiprocessor architecture that consists of a homogeneous array of domain-specific processing tiles [8].

The HyperLAN2 receiver operates as follows. Every 4 μ s a data packet of fixed size (called OFDM symbol) arrives from the antenna's front-end to the receiver's input and these are the stream items that the application process. Each stream item is processed separately. It passes through several stages of processing as the purpose is to extract the useful information carried by the wireless channel – the pay-load. This pay-load is produced by the receiver's output. Since missing an OFDM symbols is unacceptable, the pipeline must be ready to accept new data item every 4 μ s.

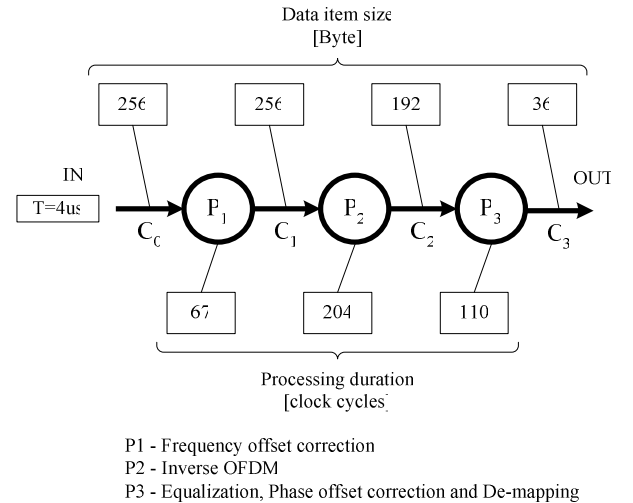


Figure 3: Process graph of a HyperLAN2 receiver

First step towards our goal is to represent the HyperLAN2 receiver through a process graph [2], as shown in Figure 3. The graph is a chain of three processes P_1 , P_2 and P_3 . The details about the algorithms implemented by these processes are not of importance for the example and are not given here. What is important is the number of cycles it takes to process a

data item. These numbers are given in the figure as graph nodes' labels (the boxes below the graph). They are what in *Section III* was denoted by $f(L)$, so these numbers are specific for the processing tiles in our architecture.

The labels on the process graph edges (the boxes above the graph) represent the size of the transported data items. We see the size changes as the item progress in the pipeline and that is because some information is stripped out during the processing.

After the process graph has been derived and labelled the next step is to map it on the architecture. Since the architecture is homogeneous all the processes will be mapped on a same type of PU – our processing tile. What the mapping procedure has to do is: first, to find a free tile for each process and eventually to choose its operating frequency (thus choosing a processing time); second, to find for each edge a communication channel with certain throughput guarantees (thus choosing a communication time). Doing this it has to consider all the labels in the graph and to guarantee that the resulting processing and communication times are such that the pipeline will meet the $4 \mu\text{s}$ deadline on its input.

The choice of processing and communication times is not trivial and it is what we address in this article.

V. TIME CONSTRAINTS ON THE PIPELINE OPERATION

When the pipeline works under real-time constrains, which is the case for most of the applications, the timing aspects of its operation has to be taken into account. Even when there is no real-time constrains, during the application mapping it is important to know how the choice of processing times and the communication times influence the pipeline behaviour in order to guarantee proper application operation.

Regarding the time-constrains, in our study we distinguish three different cases of pipeline operation: *time-constrained input*, *time-constrained output* and *constrained throughput* operation.

In the case of *time-constrained input* operation the pipeline input data arrive at a fixed period of time and the pipeline must always be ready to accept it; if it is not ready, the data is lost. The pipeline output data are consumed immediately. This is the case of operation for the receiving part of base-band processing applications where a data to be processed arrive periodically from the antenna's analogue front-end.

In the case of *time-constrained output* operation the pipeline output data are read at a fixed period of time and the pipeline output must always be ready to send at that time; if it is not ready, the data is lost. The pipeline input is fed with new data immediately when it is ready to accept. This is the case of operation for the transmitting part of base-band processing applications where the processed data is sent to the antenna at exact time, periodically.

In the case of *constrained throughput* operation there are no time constraints on the pipeline's input and output: its input is fed immediately and its output is immediately consumed. But still there is a constraint on the pipeline throughput – it is

expected the pipeline to process data at a certain rate. This is the case of operation for audio/video processing application. The two previous cases, time-constrained input and time-constrained output, can also be reduced to this one if the pipeline input or output, respectively, is decoupled from the data stream by a data-buffer of reasonable capacity.

In our work we study how each of these three cases of real-time constraints restricts the choice of processing and communication times in the pipeline. But to be complete one more aspect of the pipeline operation has to be taken into account. This is the level of freedom the PU has to perform receive, process and send in parallel.

VI. PARALLELISMS IN A PU OPERATION

The aspect of the pipeline operation we are interested here is the ability of a PU to perform the basic operations - receive, process and send - in parallel. Considering this aspect we distinguish the following four basic models of PU operation:

- *sequential communications and sequential processing (SCSP)* – the three operations are performed strictly sequentially
- *parallel communication and sequential processing (PCSP)* – the send and receive operations can be performed at the same time, but the processing can be performed only if the PU does not communicate.
- *sequential communications and parallel processing (SCPP)* – the tile can process while communicating, but still sending and receiving cannot be performed in parallel.
- *parallel communications and parallel processing (PCPP)* – all the operations (send, receive and process) can be performed simultaneously.

This paper considers only pipelines consisting of PUs that use a same model of operation – homogeneous pipelines. We elaborate on each of the four models applying each of the three cases of real-time constraints.

VII. NOTATION

For clarity, in this section we explain the notation used further in the paper.

N – number of stages in the pipeline

$C_{i,n}$ – period of time needed to transmit the n -th data item out of stage i ,

$P_{i,n}$ – period of time needed to process the n -th data item in the i -th stage

$W_{i,n}$ – period of time during which the stage i has to wait after has processed the n -th data item

$n=1,2,\dots,\infty; i=1,2,\dots,N$

$C_{0,n}$ – period of time needed to receive the n -th pipeline's input data item

$C_{N,n}$ – period of time needed to transmit the n -th data item out of the pipeline

$\uparrow T$ – event “start of a period of time T ”

$\downarrow T$ – event “end of a period of time T ”

@(e) – time of occurrence of an event e

Thus:

$@(\uparrow C_{i,n})$ – time at which starts the transmission of the n -th data item out of stage i

$@(\downarrow C_{i,n})$ – time at which ends the transmission of the n -th data item out of stage i

Since the output communication channel of stage $i-1$ is the input channel for stage i , the n -th data item enters the i -th stage during $C_{i-1,n}$, it is being processed during $P_{i,n}$ and is sent to the next stage during $C_{i,n}$.

In the following developments we use the assumption that for all data items the processing and communication times for a pipeline stage are constant and the pipeline is in the steady operation:

$$(A1): C_{i,j} = C_i, P_{i,j} = P_i, W_{i,j} = W_i, \forall j \in \{0,1,2,\dots,\infty\}$$

$$0 < C_i, 0 < P_i, 0 \leq W_i, \forall i \in \{1,2,\dots,N\}$$

That is the case for the base-band processing applications we have studied. For applications with varying times the upper bounds for these times should be used.

VIII. PIPELINE BEHAVIOUR

In this section we elaborate on each of the four models of operation in order to extract basic time dependencies of the stream pipeline operation.

The *assumptions* we make are:

- the pipeline is homogeneous – all PUs use the same model of operation
- a message passing mechanism is used for communication between the PUs
- the communication channels can give throughput guarantees
- the processing times are predictable and do not depend on the input data (manifest processes). If this is not the case, the upper bounds of the processing times should be used
- the time constraints on the pipeline operation are constant and do not vary in time (fixed rates of arrival, departure or processing of data items). If this is not the case, the lower bound of the constraint variation should be used

A. Sequential communication and sequential processing (SCSP)

When this model is valid the pipeline stage performs the three operations in a sequence following the strict order: receive $C_{i-1,n}$, process $P_{i,n}$ and send $C_{i,n}$, as shown on the time diagram in *Figure 5*.

This model of operation corresponds to a PU organization where the PU uses a single memory to store the received, processed and sent data (see *Figure 4*). First, the new data is received and stored in the empty memory. After that, during processing, the PU uses the memory and at the end it stores the result again in the same memory. Finally the result from the memory is sent to the next stage and the memory is emptied again ready for the next cycle. Since the memory is shared between the three steps, these steps are mutually exclusive.

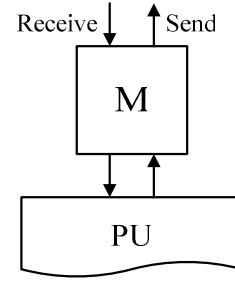


Figure 4 : PU organization for SCSP pipeline

Figure 5 presents a time diagram of a stage operation. When data has been received the processing can start immediately. After the processing is done the data can be sent to the next stage. If in that moment the next stage is not ready to accept the data, a waiting time $W'_{i,n}$ is introduced. When the data is sent the stage finishes its cycle and is ready to receive again. If the previous stage is not ready with the data, a waiting time $W''_{i,n}$ is introduced after $C_{i,n}$.

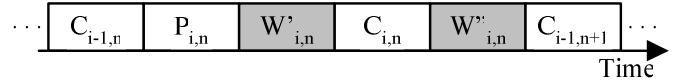


Figure 5 : Time diagram of a stage operation for a SCSP pipeline

From the described behavior we see that the period between two successive input data items for stage $i+1$ is:

$$\begin{aligned} & @(\uparrow C_{i,n+1}) - @(\uparrow C_{i,n}) = \\ & = C_{i,n} + P_{i+1,n} + W'_{i+1,n} + C_{i+1,n} + W''_{i+1,n} \end{aligned} \quad (3)$$

For stage i the period between two successive output data is:

$$\begin{aligned} & @(\uparrow C_{i,n+1}) - @(\uparrow C_{i,n}) = \\ & = C_{i,n} + W''_{i,n} + C_{i-1,n+1} + P_{i,n+1} + W'_{i,n+1} \end{aligned} \quad (4)$$

From the equivalence between (3) and (4) follows:

$$\begin{aligned} & C_{i,n} + W''_{i,n} + C_{i-1,n+1} + P_{i,n+1} + W'_{i,n+1} = \\ & = C_{i,n} + P_{i+1,n} + W'_{i+1,n} + C_{i+1,n} + W''_{i+1,n} \end{aligned} \quad (5)$$

Using the assumption (A1) from *Section VII* and substituting:

$$W'_i + W''_i = W_i$$

from (5) we derive the following recurrent equation:

$$\begin{aligned} & C_{i-1} + P_i + W_i + C_i = C_i + P_{i+1} + W_{i+1} + C_{i+1}, \\ & \forall i \in \{1,2,\dots,N-1\} \end{aligned} \quad (6)$$

When the pipeline operates with *time-constrained input* a new data arrives every period of time T , which from (3) is:

$$C_0 + P_1 + W_1 + C_1 = T \quad (7)$$

When the pipeline operates with *time-constrained output* its result is consumed every period of time T , which from (4) is:

$$C_{N-1} + P_N + W_N + C_N = T \quad (8)$$

When the pipeline operates with constrained throughput its slowest stage, say stage b , becomes a bottleneck for the

pipeline and determines the pipeline throughput. Stage b permanently cycles over the different phases, say with period T , but never waits, $W_b=0$.

$$C_{b-1} + P_b + C_b = T \quad (9)$$

If any of the equations (7), (8) or (9) is substituted in (6), the same result is derived:

$$C_{i-1} + P_i + W_i + C_i = T, \quad (10)$$

$$\forall i \in \{1, 2, \dots, N\}$$

This equation represents the basic time dependency between the stages of a SCSP pipeline. Since $0 \leq W_i$, in order real-time constraints to be met for each pipeline stage it must hold:

$$C_{i-1} + P_i + C_i \leq T \quad (11)$$

This is the general constraint for communication and processing times in the SCSP pipeline. It holds for the three cases of time-constrained pipeline.

B. Parallel communication and sequential processing (PCSP)

This model allows sending and receiving data in parallel but the processing still has to be done when the PU does not communicate (see *Figure 7*).

This communication model corresponds to a PU organisation where two separate memories are used for sent and received data, but both of them are used by the PU during the processing, *Figure 6*.

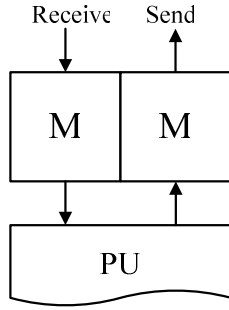


Figure 6 : PU organization for PCSP pipeline

Similar behaviour can be derived from the PU organization for SCSP in *Figure 4* if a dual port memory is used and the received data is stored in the memory freed by the sent data. But such an organisation requires synchronization between the send and receive operation, which will complicate the time dependencies in the pipeline. In this work such an organization is not considered.

A time diagram for a SCSP pipeline stage operation is presented in *Figure 7*. When the input data has been received and the previously processed data has been sent the processing can start. If one of these two conditions does not hold, either waiting time $W^{a''}_{i,n}$ or $W^{b''}_{i,n}$ is introduced (but not both together). After the processing has finished, communication can start. If the previous stage is not ready to transmit or the next stage is not ready to receive, waiting times $W^{a''}_{i,n}$ and $W^{b''}_{i,n}$ are introduced here.

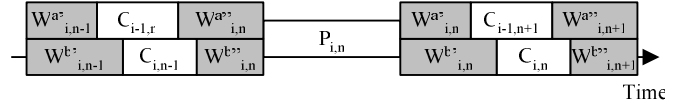


Figure 7 : Time diagram of a stage operation for a PCSP pipeline

From *Figure 7* we see that:

$$W^{a''}_{i,n-1} + C_{i-1,n} + W^{a''}_{i,n} = W^{b''}_{i,n-1} + C_{i,n-1} + W^{b''}_{i,n} \quad (12)$$

From the described behavior we find that for stage $i+1$ the period between two successive input data is:

$$\begin{aligned} @(\uparrow C_{i,n+1}) - @(\uparrow C_{i,n}) &= \\ &= C_{i,n} + W^{a''}_{i+1,n} + P_{i+1,n} + W^{a''}_{i+1,n} \end{aligned} \quad (13)$$

For stage i the period between two successive output data is:

$$\begin{aligned} @(\uparrow C_{i,n+1}) - @(\uparrow C_{i,n}) &= \\ &= C_{i,n} + W^{b''}_{i,n+1} + P_{i,n+1} + W^{b''}_{i,n+1} \end{aligned} \quad (14)$$

From the equivalence between (13) and (14):

$$\begin{aligned} C_{i,n} + W^{b''}_{i,n+1} + P_{i,n+1} + W^{b''}_{i,n+1} &= \\ &= C_{i,n} + W^{a''}_{i+1,n} + P_{i+1,n} + W^{a''}_{i+1,n} \end{aligned} \quad (15)$$

Using the assumption (A1) from *Section VII* and substituting:

$$W^{a''}_i + W^{a''}_i = W_i^a \quad \text{and} \quad W^{b''}_i + W^{b''}_i = W_i^b$$

From (12) and (15) we derive:

$$W_i^a + C_{i-1} = W_i^b + C_i \quad (16)$$

$$\begin{aligned} C_i + P_i + W_i^b &= C_i + P_{i+1} + W_{i+1}^a, \\ C_{i-1} + P_i + W_i^a &= C_i + P_{i+1} + W_{i+1}^a, \end{aligned} \quad (17)$$

$$C_i + P_i + W_i^b = C_{i+1} + P_{i+1} + W_{i+1}^b,$$

$$\forall i \in \{1, 2, \dots, N-1\}$$

With the same reasoning as for the SCSP model, for the three cases of time-constrained operation we can write, respectively:

$$C_0 + P_1 + W_1^a = T$$

$$C_N + P_N + W_N^b = T$$

$$C_{b-1} + P_b + W_b^a = T$$

Substituting each of them in (17) we derive:

$$C_{i-1} + P_i + W_i^a = T, \quad (18)$$

$$C_i + P_i + W_i^b = T$$

Since $0 \leq W_i^a$ and $0 \leq W_i^b$, for the general constraint on the communication and processing times in a PCSP pipeline we derive:

$$C_{i-1} + P_i \leq T,$$

$$C_i + P_i \leq T, \quad (19)$$

$$\forall i \in \{1, 2, \dots, N\}$$

C. Sequential communication and parallel processing (SCPP)

This model allows communication and processing to be performed simultaneously, but still sending and receiving must be done sequentially, see *Figure 9*.

This communication model is valid for a PU organisation where two separate memories are used for communication and processing as they are swapped in the beginning of each operation cycle (see *Figure 8*). Since same memory is used for sent and received data these operation are mutually exclusive.

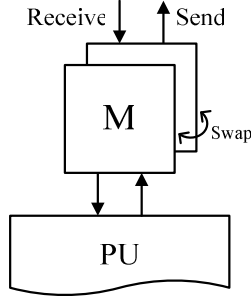


Figure 8 : PU organization for SCPP pipeline

Figure 9 presents a time diagram of a SCPP stage operation. At the beginning of the cycle, immediately after swapping, the PU memory contains the last received data and the communication memory contains the last processed data. The processing of the new data starts immediately. At the same time, sending of the last processed data can start too. If the next stage is not ready to receive, waiting time $W'_{i,n-1}$ is introduced before $C_{i,n-1}$. After the data has been sent the communication memory is free and receiving of the next data can start. If the previous stage is not ready to send, waiting time $W''_{i,n+1}$ is introduced before $C_{i-1,n+1}$. Note that new data cannot be received until the old data is sent. The result of the processing is stored in the PU memory. After the PU has finished processing it waits until the new data is received. At that moment the memories are swapped again and a new cycle starts.

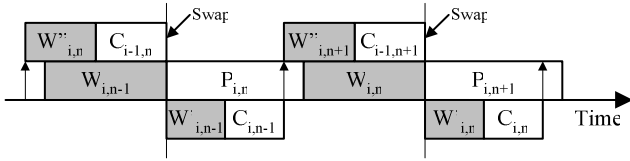


Figure 9 Time diagram of a stage operation for a SCPP pipeline

From *Figure 9* we see that:

$$P_{i,n} + W_{i,n} = W'_{i,n-1} + C_{i,n-1} + W''_{i,n+1} + C_{i-1,n+1} \quad (20)$$

From the described behavior for stage $i+1$ the period between two successive input data is:

$$@(\downarrow C_{i,n+1}) - @(\downarrow C_{i,n}) = P_{i+1,n} + W_{i+1,n} \quad (21)$$

For stage i the period between two successive output data is:

$$\begin{aligned} @(\downarrow C_{i,n+1}) - @(\downarrow C_{i,n}) &= \\ &= W''_{i,n+2} + C_{i-1,n+2} + W'_{i,n+1} + C_{i,n+1} \end{aligned} \quad (22)$$

From the equivalence between (21) and (22):

$$P_{i+1,n} + W_{i+1,n} = W''_{i,n+2} + C_{i-1,n+2} + W'_{i,n+1} + C_{i,n+1} \quad (23)$$

Using the assumption (A1) from *Section VII* and substituting:

$$W'_i + W''_i = W_i^a$$

from (20) and (23) we have:

$$P_i + W_i = C_{i-1} + W_i^a + C_i \quad (24)$$

$$C_{i-1} + W_i^a + C_i = P_{i+1} + W_{i+1},$$

$$P_i + W_i = P_{i+1} + W_{i+1}, \quad (25)$$

$$C_{i-1} + W_i^a + C_i = C_i + W_{i+1}^a + C_{i+1}$$

With the same reasoning as for the *SCSP* model for the three cases of time-constrained operation we can write, respectively:

$$P_1 + W_1 = T$$

$$C_{N-1} + W_N^a + C_N = T$$

$$C_{b-1} + W_b^a + C_b = T$$

Substituting any of the above equations in (25) the same result is derived:

$$P_i + W_i = T,$$

$$C_{i-1} + W_i^a + C_i = T, \quad (26)$$

$$\forall i \in \{1, 2, \dots, N\}$$

Since $0 \leq W_i^a$ and $0 \leq W_i^b$:

$$P_i \leq T,$$

$$C_{i-1} + C_i \leq T \quad (27)$$

D. Parallel communication and parallel processing (PCPP)

This model allows the three operations - receive, process and send - to be performed simultaneously, see *Figure 11*.

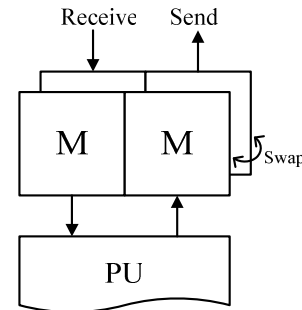


Figure 10 : PU organization for PCPP pipeline

The model is valid for a PU organisation where two memory banks each containing two separate memory blocks are used for processing and communication (see *Figure 10*). A swap between the memory banks is performed at the beginning of each stage cycle. The presence of two separate memory blocks

in the communication bank allows simultaneous sending and receiving of data.

A time diagram of a stage operation is shown in *Figure 11*. At the beginning of the cycle the memory banks are swapped. After the swapping one of the memory block in the PU bank contains the new data to be processed and the other block is empty. In the communication bank one block contains the data to be sent and the other block is empty. Immediately after the swap processing of the new data can start. Both, sending and receiving can start too. If either the next stage is not ready to receive or the previous stage is not ready to send, waiting times $W_{i,n+1}^a$ and $W_{i,n-1}^b$ are introduced before $C_{i-1,n+1}$ and $C_{i,n-1}$ respectively. The cycle finishes when all three operations are finished. If an operation finishes before the end of the cycle, waiting time is introduced after it: $W_{i,n}$, $W_{i,n+1}^a$ or $W_{i,n-1}^b$ respectively.

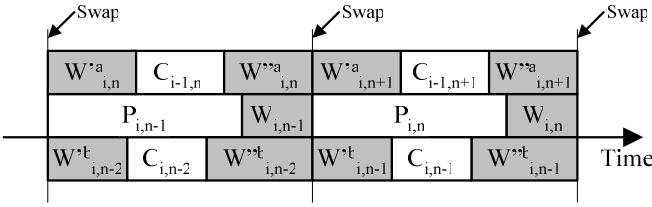


Figure 11 Time diagram of a stage operation for a PCPP pipeline

From the time diagram presented in *Figure 11* we see that:

$$\begin{aligned} W_{i,n+1}^a + C_{i-1,n+1} + W_{i,n+1}^a &= P_{i,n} + W_{i,n} = \\ &= W_{i,n-1}^b + C_{i,n-1} + W_{i,n-1}^b \end{aligned} \quad (28)$$

From the described behavior, for stage $i+1$ the period between two successive input data is:

$$@(\uparrow C_{i,n+1}) - @(\uparrow C_{i,n}) = C_{i,n} + W_{i+1,n}^a + W_{i+1,n+1}^a \quad (29)$$

For stage i the period between two successive output data is:

$$@(\uparrow C_{i,n+1}) - @(\uparrow C_{i,n}) = C_{i,n} + W_{i,n}^b + W_{i,n+1}^b \quad (30)$$

From the equivalence between (29) and (30):

$$C_{i,n} + W_{i+1,n}^a + W_{i+1,n+1}^a = C_{i,n} + W_{i,n}^b + W_{i,n+1}^b \quad (31)$$

Using the assumption (A1) from *Section VII* and substituting:

$$W_i^a + W_i^a = W_i^a \quad \text{and} \quad W_i^b + W_i^b = W_i^b$$

From (28) and (31) we derive:

$$\begin{aligned} C_{i-1} + W_i^a &= P_i + W_i = C_i + W_i^b \\ \forall i \in \{1, 2, \dots, N-1\} \end{aligned} \quad (32)$$

With the same reasoning as for the *SCSP* model, for the three cases of time-constrained operation we can write, respectively:

$$C_0 + W_1^a = T$$

$$C_N + W_N^b = T$$

$$P_b + W_b = T$$

Substituting any of the above equations in (32) the same result is derived:

$$\begin{aligned} C_{i-1} + W_i^a &= T, \\ P_i + W_i &= T, \\ C_i + W_i^b &= T, \\ \forall i \in \{1, 2, \dots, N\} \end{aligned} \quad (33)$$

Since $0 \leq W_i^a$, $0 \leq W_i$ and $0 \leq W_i^b$:

$$\begin{aligned} C_{i-1} &\leq T, \\ P_i &\leq T, \\ C_i &\leq T \end{aligned} \quad (34)$$

IX. DISCUSSION OF THE RESULTS

The main result of the previous section is the derivation of the dependencies between the processing and communication times in a stream-processing pipeline. They are given for the four models of operation respectively by the equations (10), (18), (26) and (33). From these equations and having in mind that the waiting times are greater or equal to zero immediately follow the constraints on the processing and communication times expressed by the inequalities (11), (19), (27) and (34).

A. Memory requirements

Table I summarizes the memory requirements and the constraints for the four models of operation. The memory requirements are presented normalized to the *SCSP* model.

TABLE I

Model	SCSP	PCSP	SCPP	PCPP
Mem*	1	2	2	4
Constraints	$C_{i-1} + P_i + C_i \leq T$	$C_{i-1} + P_i \leq T$ $P_i + C_i \leq T$	$P_i \leq T$ $C_{i-1} + C_i \leq T$	$P_i \leq T$ $C_i \leq T$

* normalized to *SCSP*

We see the four models differ in required memory and dependencies introduced between communication and computation. The general relation is: the less the dependencies are, the more memory is required. For example, the *SCSP* model requires the least amount of memory but the processing time of a stage depends on the times for communication with the previous and the next stage, while for the *PCPP* model the processing and communication times are fully independent, but this model requires 4 times more memory.

We have to notice that the extra memory requirement applies only to the memory a PU uses for external communication, not to the all PU's memory. The size of the memory needed for external communications is application dependent and is determined by the size of the data items. For the three stages of the *HyperLAN2* example from *Section IV* these sizes are respectively: 256, 256 and 192 bytes.

B. PUs utilisation

The results from *Table I* can be used to compare the level of PUs utilisation that the different models of operation allow. We define the utilisation of the PU in the i -th pipeline stage as:

$$U_i = \frac{P_i}{T} \quad (35)$$

So, to achieve high PU utilisation we have to aim at stage processing times approaching the period of operation T . This is easy to achieve for the SCPP and PCPP models since their processing times are independent of the communication times. But for the SCSP and PCSP models in order to get a higher PU utilisation we have to minimise the communication times. However, the communication times are lower-bounded by the communication channels capacity. Thus these models of operation potentially result in a poor PUs utilisation.

For the HyperLAN2 example (see *Figure 3*) given in *Section IV*, assuming SCSP pipeline and 16-bit communication channels working at 100 MHz, we can calculate the maximal tile utilisation as follows.

Calculate minimal communication times $C_{i_{min}}$ using (1):

$$C_{0_{min}} = 1,28\mu s, C_{1_{min}} = 1,28\mu s,$$

$$C_{2_{min}} = 0,96\mu s, C_{3_{min}} = 0,18\mu s$$

Using (11) calculate the maximal possible processing times

$$P_{i_{max}}:$$

$$P_{i_{max}} = T - C_i - C_{i-1}$$

$$P_{1_{max}} = 1.44\mu s, P_{2_{max}} = 1.76\mu s, P_{3_{max}} = 2.86\mu s$$

Using (2) we can calculate the tiles operating frequency:

$$F_1 = 47\text{MHz}, F_2 = 116\text{MHz}, F_3 = 39\text{MHz}$$

For the maximal tile utilisation that can be achieved for that application using SCSP model we calculate:

$$U_{1_{max}} = 0.36, U_{2_{max}} = 0.44, U_{3_{max}} = 0.715$$

As expected, these values are not high.

Thus, if we aim at high PU utilisation, the use of SCSP model is only justified when long complex processing is done on not very big portions of data in which case the processing times dominate and the PUs utilisation is higher.

The same calculations for the HyperLAN2 example were done for the other three models as well and the results are summarised in *Table II*.

TABLE II

	SCSP	PCSP	SCPP	PCPP
$U_{1_{max}}$	0.36	0.68	1	1
$U_{2_{max}}$	0.44	0.68	1	1
$U_{3_{max}}$	0.715	0.76	1	1
F_1 [MHz]	47	25	17	17
F_2 [MHz]	116	75	51	51
F_3 [MHz]	39	36	29	29

These figures show that the models with higher memory requirements allow higher PU utilisation and lower operating frequencies.

If assume that the maximum clock frequency of a PU is 100 MHz, we see that the SCSP mode is not feasible ($F_2 = 116$ MHz). In such a case the process that does not fit would have to be split and mapped on two PUs

C. Communication requirements

The results from *Table I* can also be used to compare what requirements the different models put on the communication

channels. Having the same P_i for all models we see that the PCPP will have the most relaxed communication requirements. The communication time there can stretch over the whole period T , thus avoiding traffic burstiness and minimizing the throughput requirements for the communication channels.

The other extreme is the SCSP model where the communication requirements are most tight. Since time of the period T is shared between C_{i-1} , P_i and C_i the communication times are short, traffic is bursty and the throughput requirements are high.

Using the HyperLAN2 example, assuming a SCSP pipeline with all tiles operating at a fixed frequency 100 MHz we can calculate the throughput requirements for the communication channels as follows.

Calculate processing times P_i using (2):

$$P_1 = 0.67\mu s, P_2 = 2.04\mu s, P_3 = 1.1\mu s$$

From (11) we derive the following system of constraints on the maximal communication times:

$$C_0 + C_1 \leq T - P_1 = 3.33\mu s$$

$$C_1 + C_2 \leq T - P_2 = 1.96\mu s$$

$$C_2 + C_3 \leq T - P_3 = 2.9\mu s$$

In a real case we would look for a solution for C_i that, for example, minimizes the required channel throughput or fits in the physical limitations of the channels. Here, in order to simplify the example, we use an easier criteria $C_1=C_2$ (with no practical meaning). Then for C_i we have:

$$C_0 = 2.35\mu s, C_1 = 0.98\mu s, C_2 = 0.98\mu s, C_3 = 1.92\mu s$$

Finally, from (1) we derive the required moment throughput S_i [MByte/s]:

$$S_0 = 109\text{MB/s}, S_1 = 261\text{MB/s},$$

$$S_2 = 196\text{MB/s}, S_3 = 19\text{MB/s}$$

High numbers, as it was expected.

The same calculations were done for the other three operation models and the results are summarised in *Table III*:

TABLE III

	SCSP	PCSP	SCPP	PCPP
S_0 [MByte/s]	109	77	128	64
S_1 [MByte/s]	216	131	128	64
S_2 [MByte/s]	196	98	96	48
S_3 [MByte/s]	19	12	18	9

D. Applying system level power saving techniques

Let us consider two power saving strategies that can be applied to each PU in the system separately: *clock/voltage scaling* and *idling*.

The *clock/voltage scaling* technique aims at lower PU operating frequency and power supply voltage and thus reduces the PU power consumption. Lowering the operating frequency increases the pipeline processing times P_i proportionally. Therefore this power saving technique aims at longest processing times ($0 << P_i < T$).

The *idling* technique force PU to run at the highest possible

frequency in order to finish the processing of the current data item as fast as possible and during the rest of the time, until the next stream unit arrives, it is set to a power saving mode (e.g. its clock is stopped or its power supply is shut down). Therefore this power saving technique aims at longest waiting times ($0 < W_i < T$) and shortest processing times ($P_i \ll W_i$).

Referring to the constraints summarised in *Table I* we can conclude that applying each of the above described techniques to a PCPP pipeline will not lead to communication restrictions, but applying it to a SCSP pipeline will require minimising of the communication times and increasing the traffic burstiness and the throughput requirements as a result.

E. Mapping of an application

The derivations of *Section VIII* can be used as input for a procedure for mapping of applications onto a multiprocessor architecture. We show how it works for the HyperLAN2 example, a SCSP pipeline and optimising for high PU utilisation. We assume that the communications is performed by a network with a mesh topology.

For the SCSP pipeline we have the following constraints:

$$C_0 + P_1 + W_1 + C_1 = T$$

$$C_1 + P_2 + W_2 + C_2 = T$$

$$C_2 + P_3 + W_3 + C_3 = T$$

$$C_3 + P_4 + W_4 + C_4 = T$$

In order to map the application process graph on a tiled architecture the following procedure can be followed:

1. Map each process on a tile considering only the tile occupancy and optimising for communication locality
2. For so mapped processes choose minimal networks paths between the communicating tiles
3. For each path determine what is the maximal throughput that can be guaranteed
4. From the maximal throughputs using (1) find the corresponding minimal C_i
5. Substitute C_i in the set of equations, substitute $W_i = 0$ and determine the maximum possible values for P_i .
6. From the processing times using (2) determine the tiles operating frequencies
7. If the result of the mapping is not satisfactory, identify the bottleneck communication channels and try to change their paths with other with a higher throughput.
8. If iterating on 7 does not lead to success, consider remapping of the processes which communication channels are the bottleneck

X. MORE COMPLICATED PROCESS GRAPHS

In this section we consider three cases of process graphs that are not straight chain but are slightly complicated: a chain with a parallel split stage, a chain with feed-forward stage and a chain with a feed-back stage. We think these are process graph topologies also typical for stream-processing applications.

Here the three graphs are briefly presented and calculations are made only for the SCSP case.

A. Parallel split

This process graph is derived from the chain graph by replacing one of its processes with several parallel processes. *Figure 12* presents a part of a chain where the stage i is split in K parallel stages denoted with P_i^j . The input communication channels of these stages are denoted with C_{i-1}^j and the output communication channels are denoted with C_i^j for $j=1,2,\dots,K$.

The stage before the split one, stage $i-1$, scatters its output data items over the K processes in the i -th stage:

$$C_{i-1,n}^j = C_{i-1,K(n-1)+j} \quad (36)$$

The stage after the split one, stage $i+1$, gathers its input data items from the K processes in the i -th stage:

$$C_{i,n}^j = C_{i,K(n-1)+j} \quad (37)$$

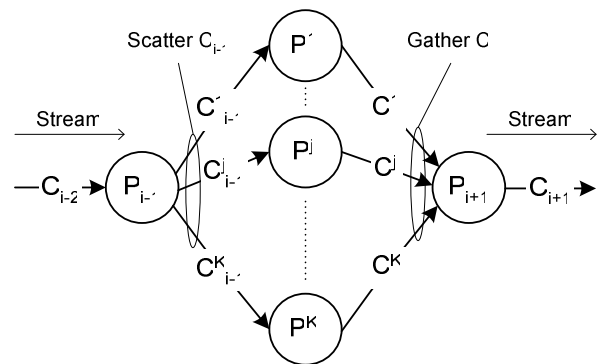


Figure 12 A pipeline with split i -th stage

In a real application a reason for such a splitting would be the need for higher performance of the i -th stage.

For stage $i-1$, using (5) and (36) we write:

$$\begin{aligned} C_{i-2,n} + W''_{i-2,n} + C_{i-3,n+1} + P_{i-2,n+1} + W'_{i-2,n+1} &= \\ = C_{i-2,n} + P_{i-1,n} + W'_{i-1,n} + C_{i-1,b}^a + W''_{i-1,n} \end{aligned} \quad (38)$$

Where:

$$a = (n \bmod K) + 1 \text{ and } b = (n \operatorname{div} K) + 1$$

(*div* denotes integer division)

Using the assumption (A1) from *Section VII* and the usual substitution of the waiting times, from (38) for the operation the before the split one, stage $i-1$:

$$C_{i-3} + P_{i-2} + W_{i-2} + C_{i-2} = C_{i-2} + P_{i-1} + W_{i-1}^j + C_{i-1}^j, \quad j \in \{1,2,\dots,K\} \quad (39)$$

For stage $i+1$, using (5) and (37) we write:

$$\begin{aligned} C_{i+1,n} + W''_{i+1,n} + C_{i,b}^a + C_{i,n+1} + P_{i+1,n+1} + W'_{i+1,n+1} &= \\ = C_{i+1,n} + P_{i+2,n} + W'_{i+2,n} + C_{i+2,n} + W''_{i+2,n} \end{aligned} \quad (40)$$

Where a and b are the same as in (38).

Using the assumption (A1) from *Section VII* and the usual substitution of the waiting times, from (40) for the operation of the stage after the split one, stage $i+1$:

$$C_i^j + P_{i+1} + \overset{\bullet}{W}_{i+1}^j + C_{i+1} = C_{i+1} + P_{i+2} + W_{i+2} + C_{i+2}, \quad (41)$$

$$j \in \{1, 2, \dots, K\}$$

The period between two successive input data items for the j -th process in stage i is:

$$\begin{aligned} & @(\uparrow C_{i-1, n+1}^j) - @(\uparrow C_{i-1, n}^j) = \\ & = C_{i-1, n}^j + P_{i, n}^j + W_{i, n}^j + C_{i, n}^j + W_{i, n}^j \end{aligned} \quad (42)$$

But:

$$\begin{aligned} & @(\uparrow C_{i-1, n+1}^j) - @(\uparrow C_{i-1, n}^j) = \\ & = @(\uparrow C_{i-1, Kn+j}^j) - @(\uparrow C_{i-1, K(n-1)+j}^j) \end{aligned} \quad (43)$$

And:

$$\begin{aligned} & @(\uparrow C_{i-1, Kn+j}^j) - @(\uparrow C_{i-1, K(n-1)+j}^j) = \\ & = \sum_{p=K(n-1)+j}^{Kn+j} (C_{i-2, p} + P_{i-1, p} + W_{i-1, p} + C_{i-1, p}) \end{aligned} \quad (44)$$

Therefore:

$$\begin{aligned} & \sum_{p=K(n-1)+j}^{Kn+j} (C_{i-2, p} + P_{i-1, p} + W_{i-1, p} + C_{i-1, p}) = \\ & = C_{i-1, n}^j + P_{i, n}^j + W_{i, n}^j + C_{i, n}^j + W_{i, n}^j \end{aligned} \quad (45)$$

Using the assumption (A1) from Section VII and the usual substitution of the waiting times, from (45) and (39):

$$\begin{aligned} & \sum_{q=1}^K \left(C_{i-2} + P_{i-1} + \overset{\bullet}{W}_{i-1}^q + C_{i-1}^q \right) = \\ & K(C_{i-3} + P_{i-2} + W_{i-2} + C_{i-2}) = \\ & = C_{i-1}^j + P_i^j + W_i^j + C_i^j \end{aligned} \quad (46)$$

The period between two successive output data items for the j -th process in stage i is:

$$\begin{aligned} & @(\uparrow C_{i, n+1}^j) - @(\uparrow C_{i, n}^j) = \\ & = C_{i, n}^j + W_{i, n}^j + C_{i-1, n+1}^j + P_{i, n+1}^j \end{aligned} \quad (47)$$

But:

$$\begin{aligned} & @(\uparrow C_{i, n+1}^j) - @(\uparrow C_{i, n}^j) = \\ & = @(\uparrow C_{i, Kn+j}^j) - @(\uparrow C_{i, K(n-1)+j}^j) \end{aligned} \quad (48)$$

And:

$$\begin{aligned} & @(\uparrow C_{i, Kn+j}^j) - @(\uparrow C_{i, K(n-1)+j}^j) = \\ & = \sum_{p=K(n-1)+j}^{Kn+j} (C_{i, p} + P_{i+1, p} + W_{i+1, p} + C_{i+1, p}) \end{aligned} \quad (49)$$

Therefore:

$$\begin{aligned} & C_{i, n}^j + W_{i, n}^j + C_{i-1, n+1}^j + P_{i, n+1}^j = \\ & = \sum_{p=K(n-1)+j}^{Kn+j} (C_{i, p} + P_{i+1, p} + W_{i+1, p} + C_{i+1, p}) \end{aligned} \quad (50)$$

Using the assumption (A1) from Section VII and the usual substitution of the waiting times, from (50) and (41):

$$\begin{aligned} & C_i^j + W_i^j + C_{i-1}^j + P_i^j = \\ & = K(C_{i+1} + P_{i+2} + W_{i+2} + C_{i+2}) = \\ & = \sum_{q=1}^K \left(C_i^q + P_{i+1} + \overset{\bullet}{W}_{i+1}^q + C_{i+1} \right) \end{aligned} \quad (51)$$

With the same reasoning for the three cases of time-constrained operation as in Section VIII we can write, respectively:

$$\begin{aligned} & C_0 + P_1 + W_1 + C_1 = T, \\ & C_{N-1} + P_N + W_N + C_N = T, \\ & C_{b-1} + P_b + C_b = T \end{aligned}$$

Then substituting in (6) and using (39), (41), (46) and (51) for the split stage i and the stages before and after it:

$$\begin{aligned} & C_{i-2} + P_{i-1} + \overset{\bullet}{W}_{i-1}^j + C_{i-1}^j = T, \\ & C_i^j + W_i^j + C_{i-1}^j + P_i^j = KT, \end{aligned} \quad (52)$$

$$\begin{aligned} & C_i^j + P_{i+1} + \overset{\bullet}{W}_{i+1}^j + C_{i+1} = T, \\ & j \in \{1, 2, \dots, K\} \end{aligned}$$

Since the waiting times are greater or equal than zero:

$$\begin{aligned} & C_{i-2} + P_{i-1} + C_{i-1}^j \leq T, \\ & C_{i-1}^j + P_i^j + C_i^j \leq KT, \\ & C_i^j + P_{i+1} + C_{i+1} \leq T, \\ & j \in \{1, 2, \dots, K\} \end{aligned} \quad (53)$$

B. Feed forward process

This graph is derived from the chain by adding in parallel a new process that forms a feed-forward connection between two chain stages. Figure 13 presents a part of a chain process graph where a feed-forward process is added between stages i and $i+K$. The feed-forward process is denoted with P_{FF} and its input and output communication channels are denoted respectively with C_{FF_in} and C_{FF_out} . P_{FF} receives its input data from the P_i 's feed-forward output channel C_i^{FF} and sends its output data to the P_{i+K} 's feed-forward input channel C_{i+K}^{FF} :

$$\begin{aligned} & C_{FF_in, n} = C_{i, n}^{FF} \\ & C_{FF_out, n} = C_{i+K-1, n}^{FF} \end{aligned}$$

In a particular case it might be that:

$$C_{i, n}^{FF} = C_{i, n}$$

The feed-forward process forms chain which is parallel to the main one. For this chain we can again derive the recurrent equation (6) using the same arguments as in Section VIII and can derive the time dependencies for the feed-forward stage for the three cases of real-time constraints:

$$C_{FF_in} + P_{FF} + W_{FF} + C_{FF_out} = T \quad (54)$$

And because $0 \leq W_{FF}$:

$$C_{FF_in} + P_{FF} + C_{FF_out} \leq T \quad (55)$$

For stages P_i and P_{i+K} we have respectively:

$$C_{i-1} + P_i + C_i + C_{FF_in} \leq T \quad (56)$$

$$C_{i+K-1} + C_{FF_out} + P_{i+K} + C_{i+K} \leq T \quad (57)$$

(assuming that communication with the feed-forward stage is performed sequentially)

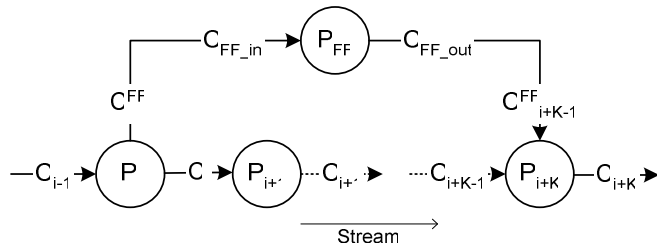


Figure 13 A pipeline with a feed-forward connection

Since the processing in stage $i+K$ is fired only when both data items, from the main chain and from the feed-forward chain, are available, the n -th data item coming from the feed-forward chain is processed together with the $n-K$ -th data item delayed by the main chain.

C. Feed back process

This graph is derived from the chain by adding in parallel a new process that forms a feed-back connection between two chain stages. Figure 14 presents a part of a chain process graph where a feed-back process is added between stages $i+K$ and i . The feed-back process is denoted with P_{FB} and its input and output communication channels are denoted respectively with C_{FB_in} and C_{FB_out} . P_{FB} receives its input data from the P_{i+K} 's feed-back output channel $C_{FB_{i+K}}^{FB}$ and sends its output data to the P_i 's feed-back input channel $C_{FB_{i-1}}^{FB}$:

$$C_{FB_in,n} = C_{i+K,n}^{FB}$$

$$C_{FB_out,n} = C_{i-1,n}^{FB}$$

In a particular case it might be that:

$$C_{i+K,n}^{FB} = C_{i+K,n}$$

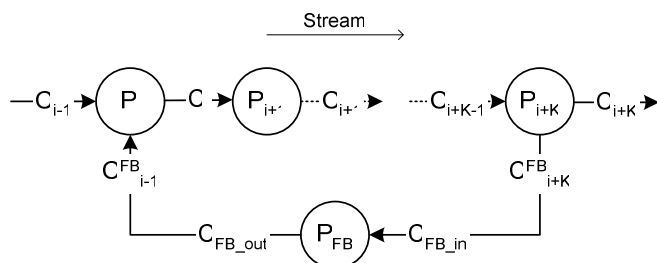


Figure 14 A pipeline with a feed-back connection

Using the same arguments as in Section VIII for the feed-forward stage it can be shown that:

$$C_{FB_in} + P_{FB} + W_{FB} + C_{FB_out} = T$$

Since $0 \leq W_{FB}$:

$$C_{FB_in} + P_{FB} + C_{FB_out} \leq T$$

For stages P_i and P_{i+K} we have respectively:

$$C_{i-1} + C_{BF_out} + P_i + C_i \leq T \quad (58)$$

$$C_{i+K-1} + P_{i+K} + C_{i+K} + C_{FB_in} \leq T \quad (59)$$

Since the processing in stage i is fired only when both data items from the feed-back stage and $i-1$ -th stage are available, the n -th data item will be processed together with the $n-K-2$ -th data item delayed through the feed-back.

XI. CONCLUSIONS

In this work we studied the timing aspects of the operation of streaming applications on multiprocessor architecture. We derived dependencies for the processing and the communication times of the processors and found the constraints on them.

We considered three cases of real-time constraints on the application and also four cases of organization of the processors communications. While all cases of real-time constraints influenced the application performance in a same way, the performance of the application differs strongly depending on the communications organization. Increasing the processors' communication memory allows achieving higher processor utilisation and applying more effectively system level power-saving techniques. The increase of the communication memory also relaxes the inter-processor communications by reducing the traffic burstiness and lowering the channels throughput requirements. Since the traffic in a system with minimized processors' communication memory is quite bursty, introducing priorities would be an effective approach for providing throughput guarantees in such a system.

We also showed how the results derived in this work could be used by an application mapping procedure.

REFERENCES

- [1] G. Kahn, "The semantics of a simple language for parallel programming." In *Information Processing*, pp. 471-475, Stockholm, August 1974.
- [2] Gerard K. Rauwerda, Paul M. Heysters, Gerard J.M. Smit, "Mapping Wireless Communication Algorithms onto a Reconfigurable Architecture", *Journal of Supercomputing*, volume 30, issue 3, pp 263-282, Kluwer Academic Publishers, December 2004.
- [3] Gerard K. Rauwerda, Gerard J.M. Smit, "Implementation of a Flexible RAKE Receiver in Heterogeneous Reconfigurable Hardware", *Proceedings of the IEEE International Conference on Field-Programmable Technology*, pp 437-440, Brisbane, Australia, December 2004.
- [4] 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.
- [5] William Dally et al. "Stream Processors: Programmability with Efficiency" *ACM Queue*, pp. 52-62, March 2004.

- [6] 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.
- [7] ETSI, Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer, ETSI TS 101 475 V1.2.2 (2001-02), 2001.
- [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.

APPENDIX

This appendix contains example time diagrams of a 4-stage pipeline for the four models of operation.

A. Sequential communication and sequential processing (SCSP)

Figure 15, Figure 16 and Figure 17 present example time-diagrams of a 4-stage SCSP pipeline respectively for the three cases of time-constrained operation: time-constrained input, time-constrained output and constrained throughput.

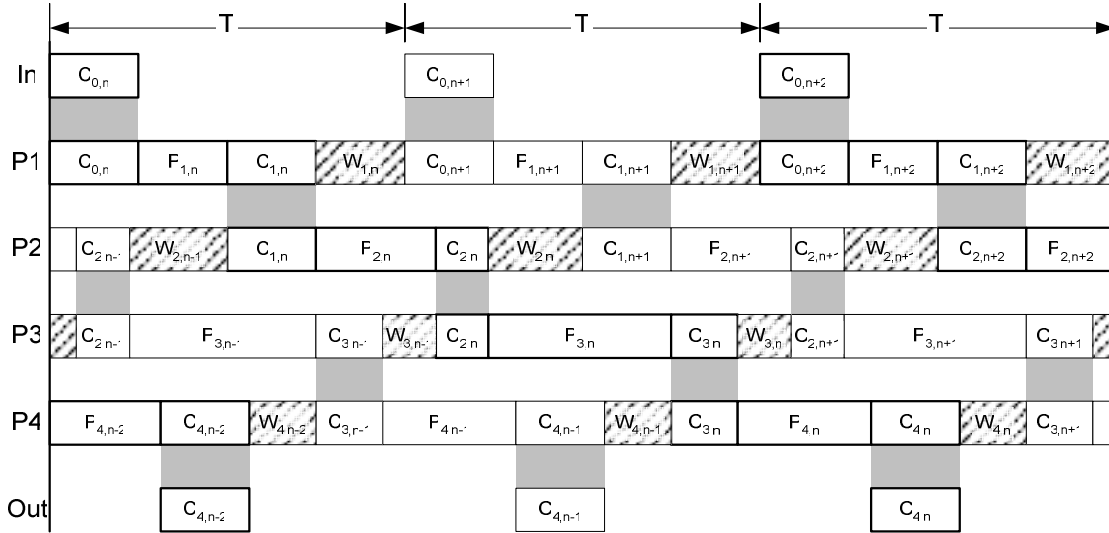


Figure 15 Time-diagram of 4-stage SCSP pipeline with time-constrained input

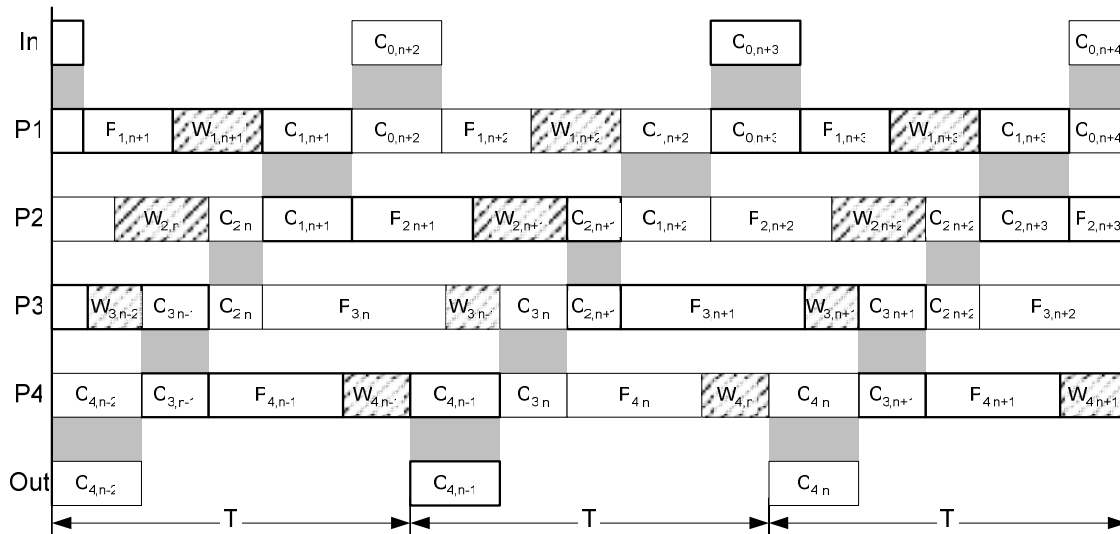


Figure 16 Time-diagram of 4-stage SCSP pipeline with time-constrained output

For the case of throughput constrained operation, shown in Figure 17, stage 2 is the one with longest period of operation, T . It becomes the bottleneck stage that determines the throughput of the whole pipeline. The stages before the bottleneck stage operate like a pipeline with time-constrained output with constraint period T . The stages after the bottleneck stage operate like a pipeline with time-constrained input with constraint period T .

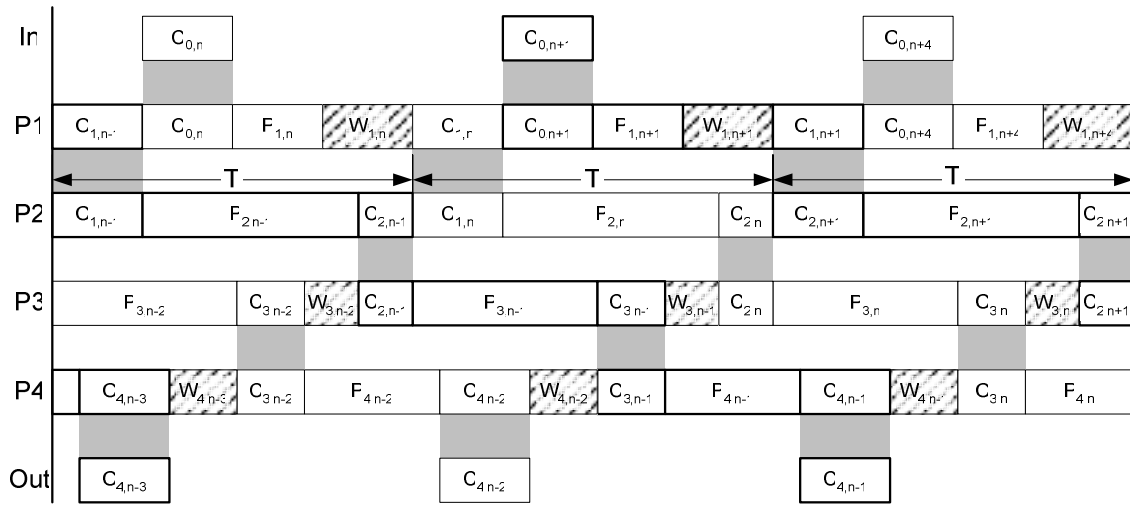


Figure 17 Time-diagram of 4-stage SCSP pipeline with constrained throughput

B. Parallel communication and sequential processing (PCSP)

Figure 18 and Figure 19 present example time-diagrams of a 4-stage PCSP pipeline respectively for time-constrained input operation and time-constrained output operation. A time-diagram for the third case of throughput constrained operation is not presented here since as we saw it is a combination of the first two cases.

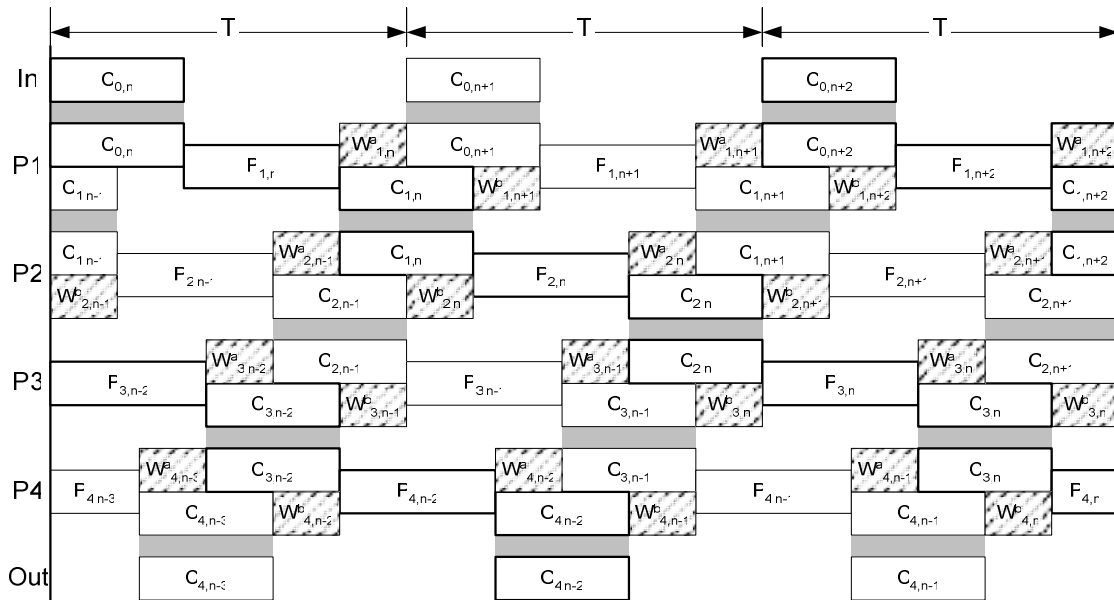


Figure 18 Time-diagram of 4-stage PCSP pipeline with time-constrained input

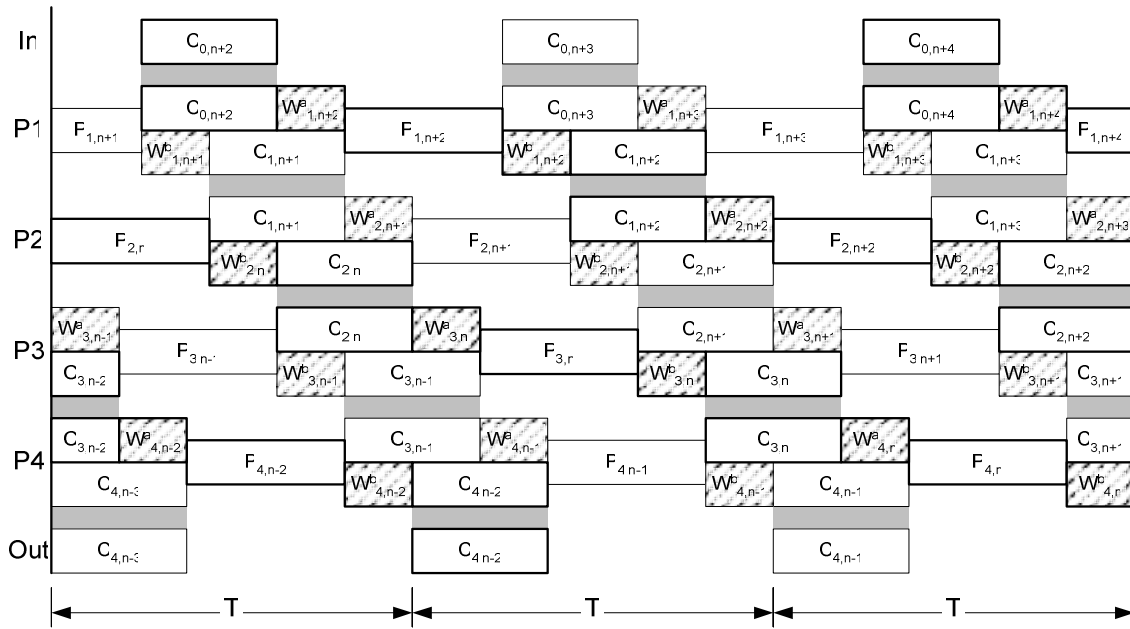


Figure 19 Time-diagram of 4-stage PCSP pipeline with time-constrained output

C. Sequential communication and parallel processing (SCPP)

Figure 20 and Figure 21 present example time-diagrams of a 4-stage SCPP pipeline respectively for time-constrained input operation and time-constrained output operation. A time-diagram for the third case of throughput constrained operation is not presented here since as we saw it is a combination of the first two cases.

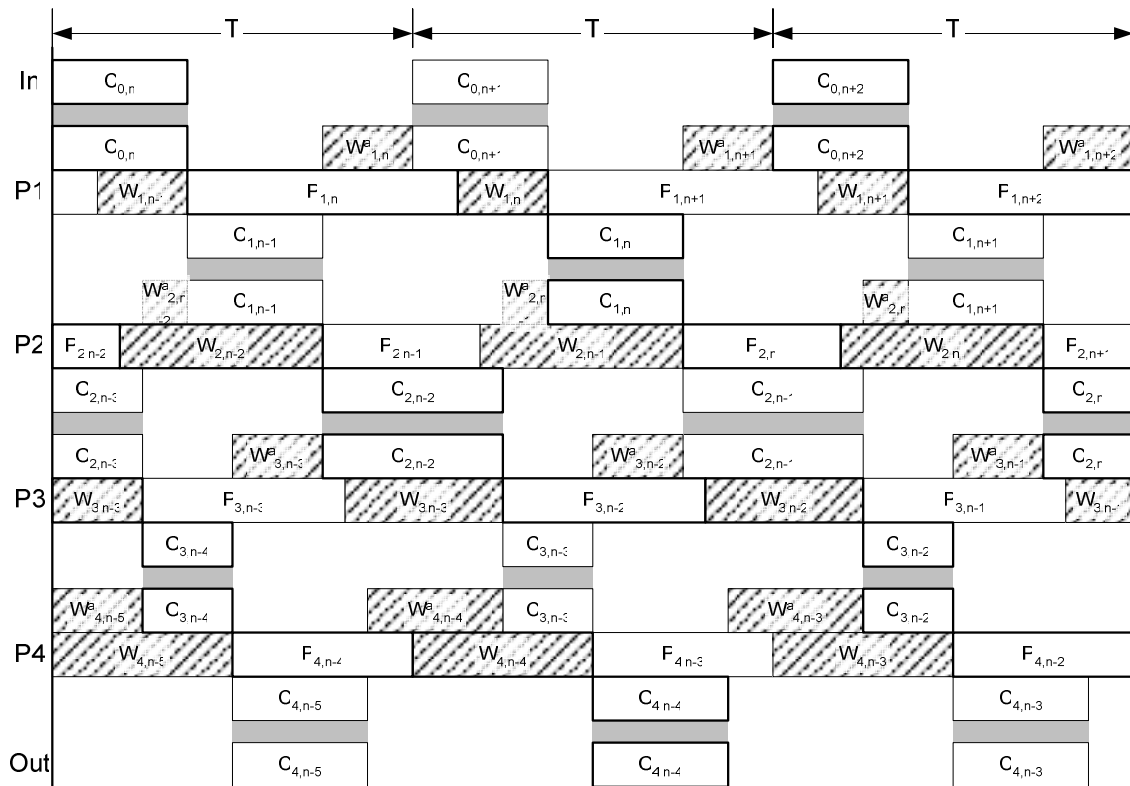


Figure 20 Time-diagram of 4-stage SCPP pipeline with time-constrained input

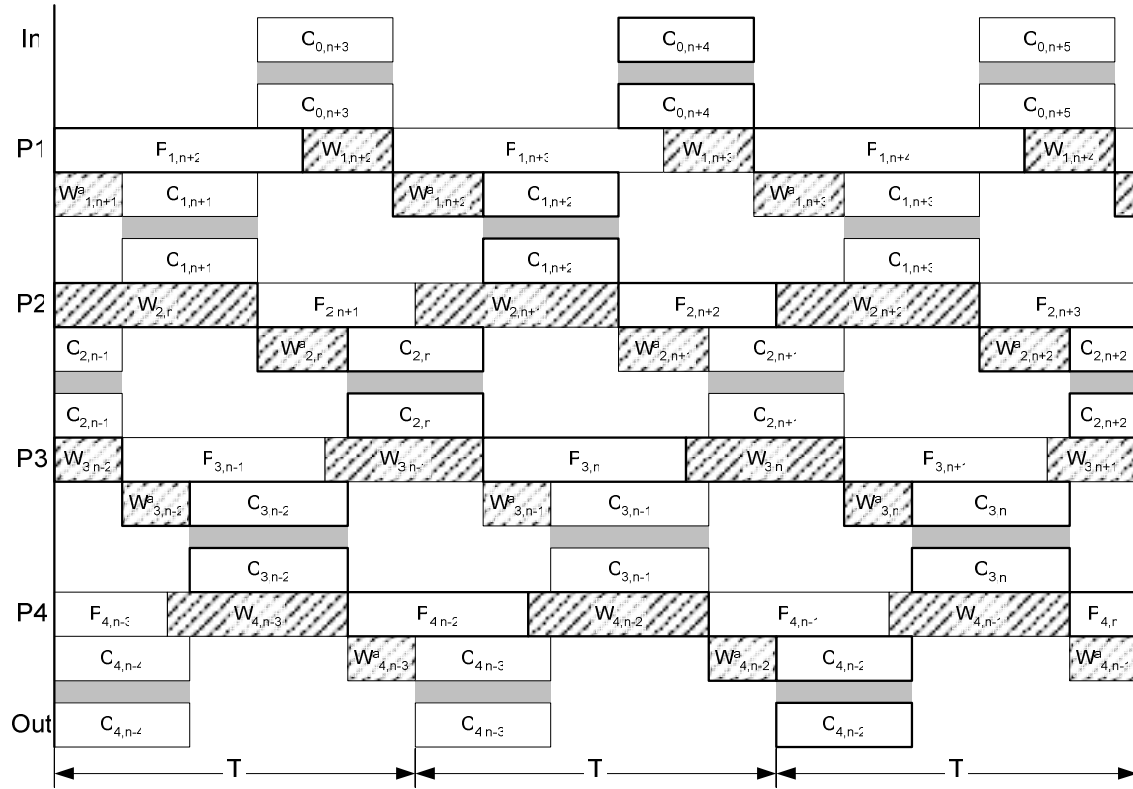


Figure 21 Time-diagram of 4-stage SCPP pipeline with time-constrained output

D. Parallel communication and parallel processing (PCPP)

Figure 22 and Figure 23 present example time-diagrams of a 4-stage PCPP pipeline respectively for time-constrained input operation and time-constrained output operation. A time-diagram for the third case of throughput constrained operation is not presented here since as we saw it is a combination of the first two cases.

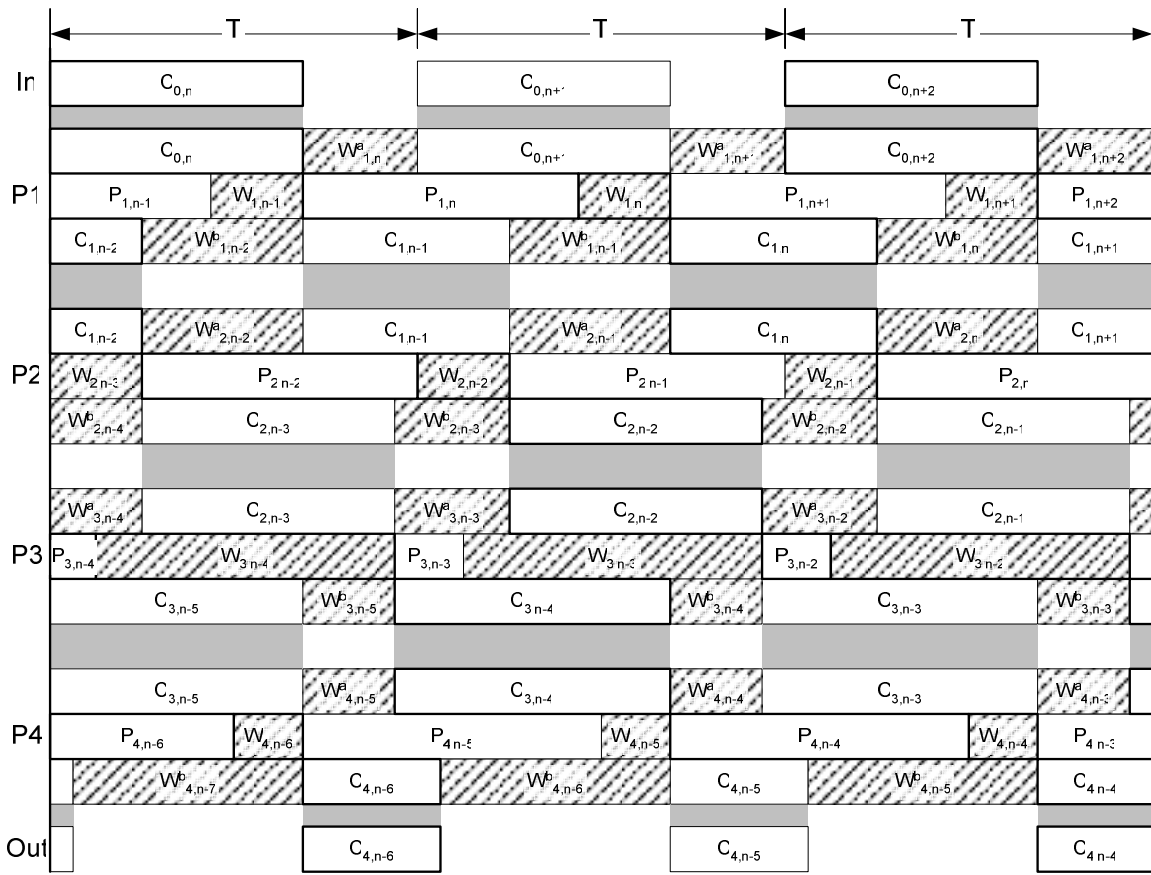


Figure 22 Time-diagram of 4-stage PCPP pipeline with time-constrained input

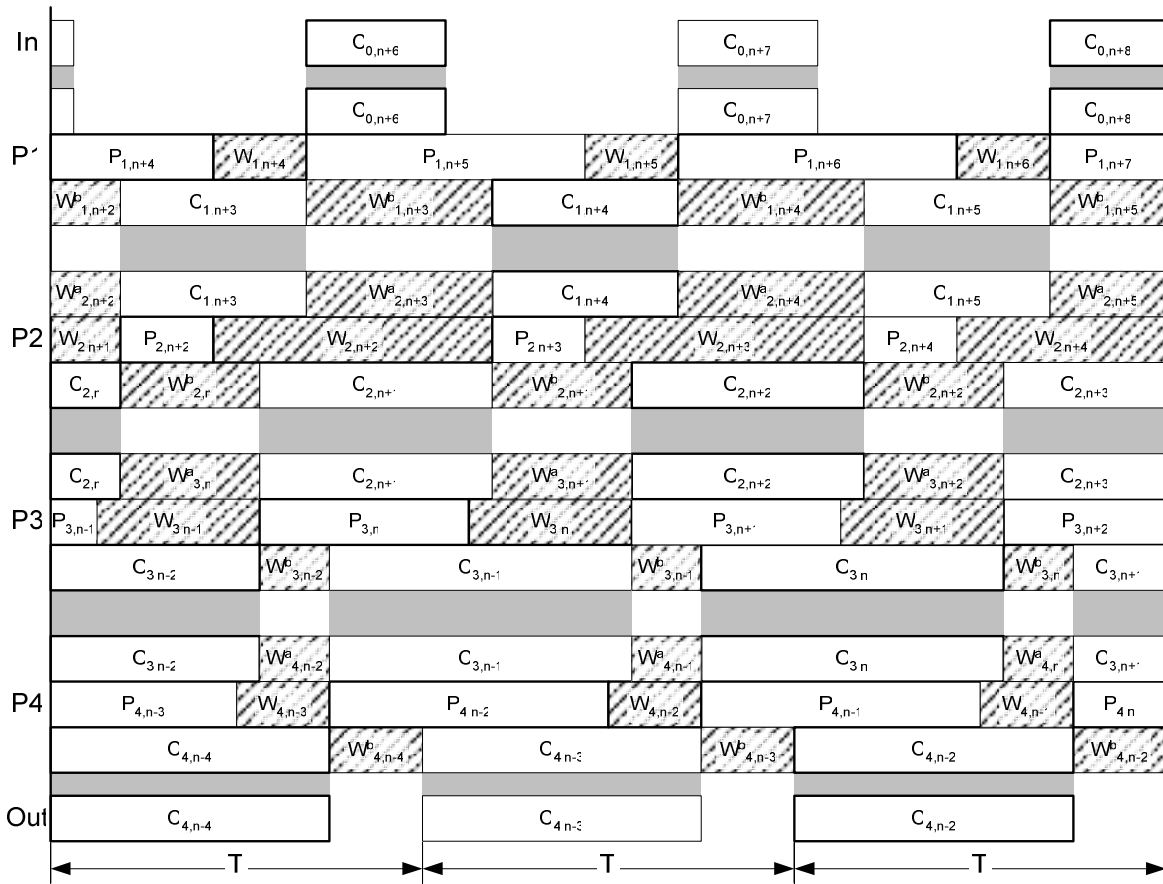


Figure 23 Time-diagram of 4-stage PCPP pipeline with time-constrained output