I. Introduction

This technical report presents an abstract model for the performance estimation of the multiprocessor CoreVA-MPSOC. The CoreVA-MPSOC targets streaming applications in embedded and energy-limited systems. The abstract model is used by our CoreVA-MPSOC compiler [1] to estimate the performance of a certain streaming application.

Our CoreVA-MPSOC compiler reads applications that are described in the programming language StreamIt [2]. A StreamIt program is represented by a structured data flow graph of its tasks (filter). The CoreVA-MPSOC compiler partitions all filter of a program onto particular cores of the MPSOC. An abstract model for such a partitioning is presented in Section II. Section III shows the abstract model of the hardware architecture of the CoreVA-MPSOC.

The configurable VLIW CPU CoreVA [3] is used as the basic building block for our MPSOC. The CPU features L1 scratchpad memories for instruction and data. Several CPU cores are tightly coupled within a cluster [4]. Several of those clusters are connected via a network on chip (NoC) [5] (cf. Fig. 1).

Within a cluster each CPU can access the L1 data memories of other CPUs via a bus based interconnect (shared, partial or full crossbar). The NoC interconnect is composed of three components: The (i) routers transport the data through the NoC in a packet-based manner. Routers are connected via (ii) network links. A (iii) network interface (NI) implements the interface between the routers and the CPUs.

Section IV shows the abstract model for the total throughput of a certain partition of a StreamIt application. A goal for the CoreVA-MPSOC compiler is to maximize this throughput to achieve the best performance for an application.

II. Model of the StreamIt Program

A StreamIt program can be represented as a structured graph \( G = (F, E) \), where \( F \) is a set of filters and \( E \) is a set of edges. Each \( e \in E \) is of form \((a, b)\) which represents a communication channel between filter \( a \in F \) and \( b \in F \) in which a message with size \( |e| \) in work function of \( a \) is send from filter \( a \) to filter \( b \). Each filter \( f \in F \) has a work function with the estimated execution time \( W(f) \) in cycles. The execution time \( W(f) \) includes repeated executions of a work function, which may be required to consume or produce enough data for the filter at it's edges.

There exists a unique filter \( L \) without outgoing edges, which is the last filter of the application:

\[ \exists L \in F \text{ s.t. } (L, b) \in E \]

There exists a unique filter \( S \) without ingoing edges, which is the first filter of the application:

\[ \exists S \in F \text{ s.t. } (a, S) \in E \]

\( M \): Multiplicity how often the work functions of all filters are called during one steady state iteration (steady state iteration).

III. Model of the CoreVA-MPSOC

The CoreVA-MPSOC consist of a set of processors \( P \).

The StreamIt compiler maps each filter to a processor:

\[ M : F \rightarrow P \]

The MPSOC has a set of clusters \( C \) and each processor belongs to a cluster: \( C : P \rightarrow C \)

Additionally the MPSOC consist of a set of network links \( N \). Each \( n \in N \) has a maximum bandwidth \( B(n) \) bytes/cycle that it can handle. A network link could be a bus-link within a cluster, a network interface (NI) or a NoC-link.

\( N(p_a, p_b) \) is a list of all network links involved when sending a message from processor \( p_a \in P \) to processor \( p_b \in P \) (depending on the routing algorithm):

\[ N : (p_a, p_b) \rightarrow [N] \]
IV. MODEL OF THE THROUGHPUT

This section shows an abstract model for throughput estimation of a certain streamIt program given by II and mapped to a configuration of CoreVA-MPSoC given by III.

A. Throughput of a Processor

A filter \( f \in F \) has input edges: \( I(f) = \{(a, f) | (a, f) \in E\} \)

A filter \( f \in F \) has output edges: \( O(f) = \{(f, b) | (f, b) \in E\} \)

For each filter \( f \in F \) we generate code of the form:

```plaintext
foreach (Channel i in I(f))
  i.WaitInputReady

foreach (Channel o in O(f))
  o.WaitOutputReady

Work_f()

foreach (Channel i in I(f))
  i.DoneWithInput

foreach (Channel o in O(f))
  o.DoneWithOutput
```

Before executing the work function \( \text{Work}_f \) of filter \( f \in F \) it is necessary to wait until all communication channels (input \( I(f) \) and output \( O(f) \)) edges are ready to use. After \( \text{Work}_f \), all communication channels \( (I(f) \) and \( O(f) \)) can be set to done. The execution time of these wait and done functions is given by the channel type of edge \( (a, b) \in E \), which depends on the location of the filter \( a \) and \( b \) (same processor, different processor but same cluster, or different cluster): \( M(a) = M(b) \rightarrow \text{memory channel} \)

\( M(a) \neq M(b) \land C(M(a)) = C(M(b)) \rightarrow \text{cluster channel} \)

\( C(M(a)) \neq C(M(b)) \rightarrow \text{NoC channel} \)

The execution time of the wait for input channels of edge \( e \in E \) is represented by \( I_w(e) \) and \( O_w(e) \) for the input channels. The execution time of the done for output channels is represented by \( I_d(e) \) and \( O_d(e) \) for the output channels.

The execution time \( E(f) \) (in cycles per steady state iteration) of filter \( f \in F \) is the sum of the execution time of all filters work function \( \text{Work}(f) \) multiplied by the Multiplicity \( \mathcal{M} \) and a sum of all software overheads for the different communication channels of all input and output edges.

\[
E(f) = \mathcal{M} \cdot W(f) + \sum_{e \in I(f)} (I_w(e) + I_d(e)) + \sum_{e \in O(f)} (O_w(e) + O_d(e)) \tag{1}
\]

The maximum throughput \( T(p) \) (in steady state iteration per cycle) of processor \( p \in P \) is the inverse of the sum of the execution time of all filters \( f \in F \) mapped to processor \( p \).

\[
T(p) = \frac{1}{\sum_{f \in M(p)} E(f)} \tag{2}
\]

Where \( M'(p) \) are all filters mapped to processor \( p \in P \):

\( M'(p) = \{ f \in F | M(f) = p \} \)

B. Throughput of a Network Link

An amount of data \( D(n) \) (in bytes per steady state iteration) is crossing each network link \( n \in \mathbb{N} \). This amount of data is based on the Multiplicity \( \mathcal{M} \) and the message sizes of all edges going through this network link \( n \)

\[
D(n) = \mathcal{M} \sum_{e \in \{(a, b) | E \in \mathbb{N}(M(a), M(b))\}} |e| \text{ bytes per steady state iteration} \tag{3}
\]

The maximum throughput \( T(n) \) of network link \( n \in \mathbb{N} \) is the maximum bandwidth (in bytes per cycle) a network link \( n \) can handle divided by the time the network link needs to transmit all the data (in bytes) of one steady state iteration.

\[
T(n) = \frac{B(n)}{D(n)} \text{ bytes per steady state iteration} \tag{4}
\]

C. Total throughput of the system

The throughput of all processors is given by set \( T_{\text{comp}} \).

\[
T_{\text{comp}} = \{ T(p) | p \in P \} \text{ bytes per steady state iteration} \tag{5}
\]

The throughput of all network links is given by set \( T_{\text{network}} \).

\[
T_{\text{network}} = \{ T(n) | n \in \mathbb{N} \} \text{ bytes per steady state iteration} \tag{6}
\]

The amount of data \( D(f) \) (in bytes per steady state iteration) consumed by a filter \( f \in F \) depends on Multiplicity \( \mathcal{M} \) and the message sizes of all its input edges.

\[
D(f) = \mathcal{M} \sum_{e \in I(f)} |e| \text{ bytes per steady state iteration} \tag{7}
\]

The total throughput of the StreamIt application \( T_{\text{system}} \) is given by the bottleneck of the system. The bottleneck of the system is the component (processor or network link) with the lowest throughput.

\[
T_{\text{system}} = \min(T_{\text{comp}} \cup T_{\text{network}}) \text{ bytes per steady state iteration} \tag{8}
\]

Or if we also consider the amount of the produced data within one steady state:

\[
T_{\text{system}} = D(A) \min(T_{\text{comp}} \cup T_{\text{network}}) \text{ bytes per steady state iteration} \tag{9}
\]

V. CONCLUSION

In this report, an abstract model for the performance estimation of the CoreVA-MPSoC has been presented. The abstract model is able to estimate the maximum throughput of a certain streaming application mapped to a particular configuration of the CoreVA-MPSoC.
REFERENCES


