Setup of an Experimental Framework for Performance Modeling and Prediction of Embedded Multicore AI Architectures

Quentin DARIOL (E-mail: quentin.dariol@etu.univ-nantes.fr)^{1,2}, Sebastien LE NOURS^{1}, Sebastien PILLEMENT^{1}, Kim GRÜTTNER^{2}, Domenik HELMS^{2}, and Ralf STEMMER^{2}

^{1}Nantes University, IETR UMR CNRS 6164, France
^{2}German Aerospace Center / Deutsches Zentrum für Luft und Raumfahrt (DLR), Germany

January 2022

Abstract

Evaluation of performance for complex applications such as Artificial Intelligence (AI) algorithms and more specifically neural networks on Multi-Processor Systems on a Chip (MPSoC) is tedious. Finding an optimized partitioning of the application while predicting accurately the latency induced by communication bus congestion, is hard using traditional analysis methods. This document presents a performance prediction workflow based on SystemC simulation models for timing prediction of neural networks on MPSoC.

Keywords — Model of Performance, Multi Processor, Real-Time Analysis, Embedded Artificial Intelligence, Neural Network

1 Introduction

The market of the Internet-of-Things (IoT) is still growing, as the number of connected devices is expected to reach more than 27 billion by 2025 (an increase of more than 200% compared to 2020) [15]. Along with its expansion, the need for smart devices used to infer Artificial Intelligence (AI) algorithms such has neural networks has become preponderant. Due to the complexity of neural network algorithms, they are often deployed on servers available at fog or cloud layer of the IoT application. The servers dispose of enough computation and memory resources for fast and accurate inference of AI algorithms. However, executing AI algorithms on cloud is not optimal due to the necessity to transmit data and computation results from and to the connected device. This bears high costs in terms of latency and energy consumption, and induces variability in execution time (limited bandwidth for data transfers, centralized processing for several applications in server).

To avoid loss in throughput and energy caused by data transmissions, the focus is nowadays on the deployment of AI algorithm on edge devices. Among edge devices, MultiProcessor Systems-On-Chip (MPSoC) are widely used due to the versatility they offer. However, neural networks are computation-intensive applications that require important amount of resources, while embedded platforms are limited in processing and memory resources, and bear strict energy constraints. For this reason, implementing neural networks on MPSoC is tough. In this context, an intensive evaluation of neural network implementations on MPSoC is needed early in the design process to identify solutions that optimize both performance and energy.

Several approaches have already been proposed to predict performance of embedded AI algorithms on embedded platforms. Most approaches focus on proposing frameworks to perform design space exploration for neural networks deployed on edge devices. Some of these approaches rely solely on the implementation and performance evaluation through measurement. Others propose analytical models used to explore and optimize architectures for neural network inference. These approaches fail to propose scalable system level models to allow fast yet accurate analysis of embedded AI algorithms on MPSoC. Building such models requires capturing the effects of accesses to shared resources through communication bus, while allowing to explore several mapping and scheduling of the AI application.

In the scope of this work, our aim is to deliver efficient performance models of neural networks mapped on MPSoCs. To do so, the following challenges have been considered during the 1st year of the PhD:

- **Modeling access to shared resources:** what is the impact of data transmissions on the platform on the application's performance? How to accurately capture this mechanism in performance models?
- **Partitioning of the neural network algorithm:**
  1. What level of abstraction should be considered to model the neural network application, to optimally exploit the parallelism of the application deployed on the considered platform?
  2. Once the grain set, how to partition optimally the application between a user defined number of processing elements on a given platform (mapping / scheduling)?

This document presents a modeling and analysis framework for performance prediction for neural networks deployed on MPSoC. The second section of this document positions our work compared to the state-of-the-art in this field. The third section presents our proposal: an overview of our workflow is given, then each part of the workflow is presented in details. The fourth section presents results obtained on examples. The fifth section presents the identified work directions for future work. The last section gives our conclusion on the work done so far.

2 State of the art

Several trends can be observed in the field of research on performance prediction for neural networks deployed in embedded platforms.

Some approaches focus on architectural exploration: the aim is to perform performance prediction for candidate SoC or FPGA-based accelerator for neural networks. These works are often based on analytical models that allow exploring the design space to find an architecture that optimize non-functional properties such as execution time, energy consumption and surface/memory usage. These approaches often rely on capturing the neural network in a model of computation such as Synchronous DataFlow (SDF) [7] to partition it in an efficient way to exploit its parallelism.

A notable work in this field is [19]. This work focuses on the exploration of Convolutional Neural Networks (CNN) deployment on FPGA using SDF under timing constraints. This approach relies on analytical models to find efficient accelerator candidate architectures in terms of execution time. The work in [16] presents a Design Space Exploration (DSE) method to allow finding the best SoC architectures to implement CNN algorithms. The work emphasizes on finding an efficient bus or interconnect IP for communications between actors. [2] proposes a tool for hardware architecture search for very low power neural network implementations based on their own accelerator architecture (UltraTrail).

Another notable approach is NNSim [8]. It proposes a set of SystemC models used to evaluate and search for optimized mapping parameters for CNN implementations using the Eyeriss accelerator architecture [3] [4]. The Eyeriss architecture is based on a set of processing elements connected through a Network on Chip, and managed by a controller IP. It is similar to GPU architectures [12]. NNSim [8] proposes a Transaction-Level Model (TLM) in SystemC to alleviate the long simulation time required by RTL models. However, this work focuses on the processing element timing models and does not consider the communications between processing elements. Mechanisms such as congestion on communication channels and the effect of arbitration of processing elements on execution time are overlooked.

All these approaches are focused on proposing analytical models or tools to explore candidate accelerator architectures for a given neural network. In our work, we focus on proposing models of performance to explore mappings of a partitioned neural network on a given MPSoC platform.

Other approaches aim at proposing a performance evaluation workflow to explore several mapping/schedulings of a given neural network on a given platform. A notable approach is [18], which proposes a framework to perform DSE for CNNs deployed on low power processor-based platforms designed for computer vision applications. Their work targets more specifically the Intel Movidius Myriad 2 Vision Processing Unit (VPU). Their work shows that the fine-tuned resource management offered by their workflow reduces the execution time up to 3,6% and the energy consumption up to 7,7% in comparison with straightforward implementations.

Another notable approach can be found in [5]. This paper proposes a method to explore topologies for a given Deep Neural Network (DNN) in order to explore trade-offs between accuracy (Quality of Service - QoS) and energy/inference time. The neural network is implemented on a NVIDIA Jetson TX2 Graphical Processing Unit (GPU). The candidate topologies are trained on non-embedded GPU boards, then implemented and tested on the target device.

Both these approaches rely on implementation and testing of candidate neural network algorithms on the considered platform to evaluate their performance. They do not propose analysis models to predict energy and execution time ahead of the design and implementation phases. Their work is also focused on VPU or GPU architectures, which contain resources to ease the execution of applications such as neural networks. In our work, we consider MPSoC platforms that do not present activated features for acceleration of neural networks.

The Table 1 summarizes the approaches led in the field of performance evaluation of neural networks deployed on embedded platforms.

Work	**Proposition**	**Target**	**Performance Evaluation Technique**	**Dataflow model**	**Effort on communications**
[19]	FPGA-based accelerator architecture that can explore and exploit all sources of parallelism	FPGA	Analytical model based on SDF	SDF	None
[10]	FPGA-based accelerator architecture that can explore and exploit all sources of parallelism	FPGA	Analytical models	None	None
[16]	SoC design exploration using analytical models for CNNs	SoC	Analytical model	Yes (not detailed)	Exploration of bus and interconnect IPs
[8]	SystemC TLM models to explore Eyeriss-type architectures Chen2016 Chen2018 for CNNs	FPGA	Various level of computation models	None	Not modeled
[2]	Exploration of very low power architectures for NN on FPGA, focused on memory optimization to reduce power	FPGA	Algorithm for architecture search with measurement	None	None
[18]	Performance evaluation (timing, energy) of CNNs on Intel Movidius Myriad2	VPU	Evaluation of measured performance	None	Optimization by reduction of data transfer number
[5]	Performance evaluation (timing, energy) of CNNs on GPU	GPU	Evaluation of measured performance	None	None
Our work	Performance prediction using SystemC models for NN deployed on MPSoC	MPSoC	System level communication and computation models	SDF	Probabilistic system level model of bus

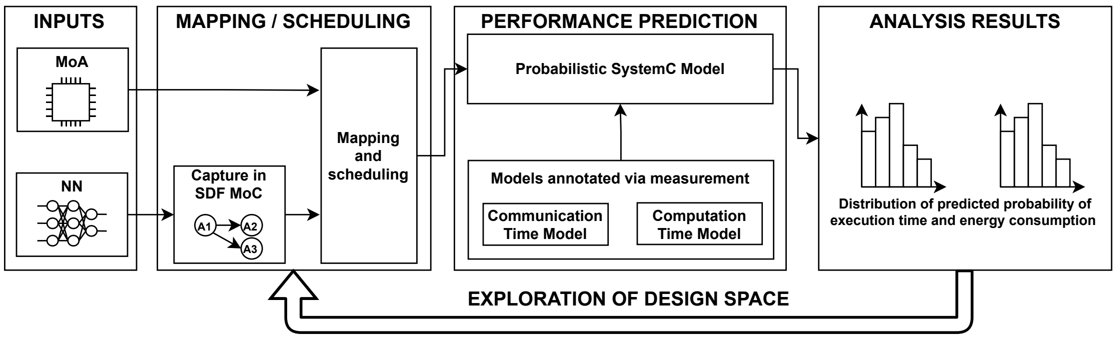
Table 1: Summary of notable state-of-the-art approaches for neural network evaluation on embedded platforms

The work led during the first year of this PhD focuses on the proposal of scalable system level models for timing analysis of neural networks deployed on MPSoC platforms. The models allow predicting accurately the effects of data exchanges of the communication bus on the platform. The workflow presented in this work is based on previous work presented in [17] [20]. The previous workflow was based on a probabilistic SystemC model and allows predicting execution times of video processing applications. This approach has demonstrated to deliver fast yet accurate analysis on video processing applications (1% error in end-to-end latency prediction achieved for 1.000.000 images, with analysis time around 5 seconds). During the first year of the PhD, this workflow has been updated to allow exploring several partitionings and mappings of neural network algorithms.

3 Proposal

We propose a workflow for performance prediction for neural networks deployed on MPSoC at system level. An overview of our workflow is given in the first section. Then every part of the workflow is presented in details: the considered input application, the considered model of architecture, the model of computation used, the model of performance and the results obtained so far.

3.1 Overview of the modeling and simulation workflow



INPUTS
MoA
NN
MAPPING / SCHEDULING
Capture in SDF MoC
A1
A2
A3
Mapping and scheduling
PERFORMANCE PREDICTION
Probabilistic SystemC Model
Models annotated via measurement
Communication Time Model
Computation Time Model
ANALYSIS RESULTS
Distribution of predicted probability of execution time and energy consumption
EXPLORATION OF DESIGN SPACE

Figure 1: Overview of the proposed modeling and analysis workflow

The proposed modeling and analysis workflow is presented in Figure 1. It is composed of the following elements, which are presented in details in the following sub-sections of this report:

- **The Neural Network (NN):** in our workflow, the considered input application is a neural network.
- **The Model of Architecture (MoA):** the considered platform is composed of a set of tiles, which contain one processing element with a separate connected private memory. It also contains shared resources such as memories which can be accessed by the tiles through a communication bus.
- **The Model of Computation (MoC):** Synchronous DataFlow (SDF) is used as a model of computation in our workflow. The neural network is captured in SDF.
- **The mapping and scheduling step:** in this step, the neural network captured in SDF is mapped and self scheduled on the targeted MPSoC platform.
- **The performance prediction step:** the performance prediction is based on both **a communication model and a computation model**, which are integrated in

a **probabilistic SystemC model**. The SystemC model is used for execution time analysis. In future work, the energy consumption will also be predicted.

- **The DSE step:** this DSE is performed by evaluating several mapping and scheduling for the considered neural network on the considered platform. Optimized candidate implementations under timing and energy constraints can be identified using this process.

3.2 Input Application: Neural Network

AI is a large field, covering numerous algorithms such as decision trees, data clustering and bayesian classifiers for the most commons ones. One of the most notorious AI algorithm used for image classification problems is the neural network [14]. This type of AI algorithm is based on the architecture of brain's neural networks. There are several architectures of neural networks. The classical, first architecture of neural networks is the Multi-Layer Perceptron (MLP) as introduced by Rosenblatt in 1952 [14]. The MLP is only composed of fully-connected layers. For now, this work is entirely focused on Multi-Layer Perceptrons type of neural networks.

To ease the implementation of neural networks, deep learning frameworks are widely used. These frameworks offer classes and functions that can be used to easily build, train, save, deploy and execute a neural network. Notorious examples of such frameworks are Tensorflow, released by Google in 2015 [1] and PyTorch [13]. In this work, we rely on the deep learning library LibFann (Fast Artificial Neural Network Library) [11]. This lightweight library allows capturing multi-layer perceptrons to deploy them on tiny embedded platforms. It offers simple to use and modify C code, which makes it a better suit for us than other deep learning frameworks that mainly use Python. The use of the workflow along with LibFann allows implementing neural network applications on the platform and characterizing the actors identified in the SDF graph.

In this paper, the work is focused on a multi-layer perceptron algorithm used to perform digit recognition. To train and test the neural network, we relied on the MNIST (Modified National Institute of Standards and Technology) database, which was introduced by Y. Lecun [6] to train neural networks for digit recognition tasks. The considered neural network algorithm is presented in Figure 2.

In order to test the LibFANN library, and manage memory issues on the targeted platform, an elementary application has been considered in addition to the work on the MNIST application. This application is used to predict the output of a two input XOR gate. This application is presented in Figure 3.

In the scope of this work, the neural network is considered to be already trained before its implementation on target. We do not consider the training of the neural network on the targeted platform.

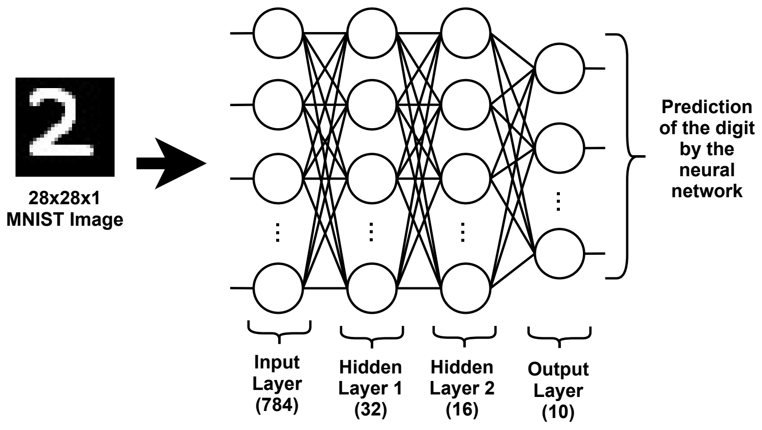


Figure 2: Studied MNIST application

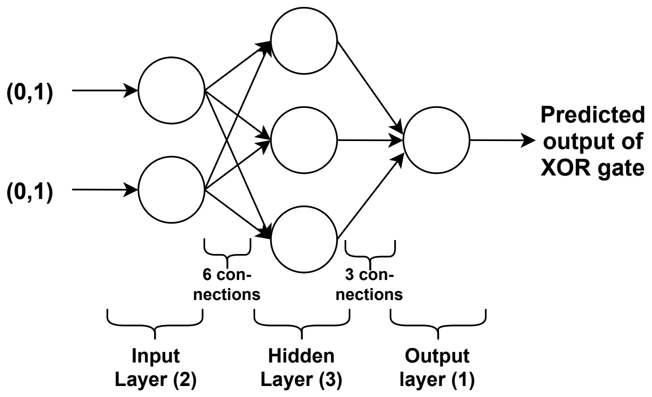


Figure 3: Studied XOR application

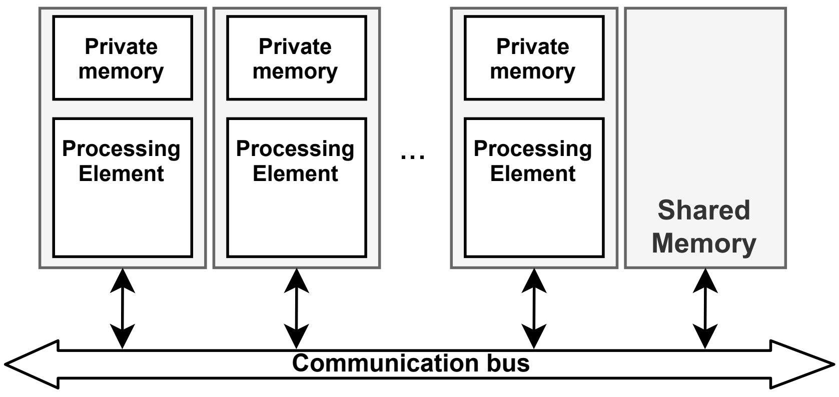


Figure 4: Schematic of our Model of Architecture (MoA)

3.3 Model of Architecture (MoA)

The considered platform is composed of a set of tiles (a *tile* is one processing element with a separate connected private memory). Executing instructions from this private memory causes no interference with other tiles. Data exchanges between different tiles are performed via a shared memory. The accesses to the shared memory are done using a communication bus. A schematic of our platform is given in Figure 4.

In this work, tiles are assumed to be identical, except for private memory sizes. The targeted platform is thereby assumed to be homogeneous.

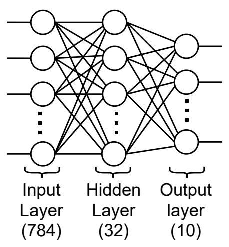
3.4 Model of Computation (MoC) - Synchronous DataFlow (SDF)

To ease the analysis and DSE of candidate neural network implementations on targeted platform, a model of computation is required. In our work, we consider SDF graphs, as introduced by Lee and Messerschmitt in 1987 [7]. This model is used to describe the data flow between actors via communication channels. SDF offers a strict separation of computation (compute) and communication phases (read, write) of actors. During the compute phase, no interference with any other actor can occur. Because of this, capturing neural networks in SDF model of computation enables a more flexible DSE process while removing restrictions on the targeted hardware platform.

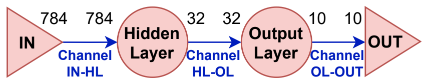
The work in [9] presents a way of capturing neural networks in SDF, featuring several levels of abstraction. The lowest granularity translation considered is the layer grain. For this granularity, the actors considered in the SDF graph are the layers of the neural network. A finer granularity is the neuron grain, where the actors considered in the SDF graph are the neurons of the neural network. However, the layer grain is too dense to exploit any parallelism from the neural network. The neuron grain invokes numerous communication channels, which overload the communication bus of the considered platform. Both the layer and neuron grains are, for this reason, not optimal choices of grain to capture the neural network application and deploy it on the considered MoA.

In this work, we also consider an intermediate grain, composed of a cluster of neurons. When using this level of abstraction, the actors of the SDF graph are sets of neurons obtained from the partitioning of layers. The number of clusters defines how many actors are generated for each layer in the SDF graph. This grain presents the advantage of allowing exploiting parallelism from the AI algorithm, while limiting the number of communications and thereby the communication time. A cluster of neurons only groups neurons of the same layer, therefore, by construction of neural networks, clusters formed from the same layer can be executed at the same time, which allows executing computations in parallel. The various granularity levels are depicted in Figure 5.

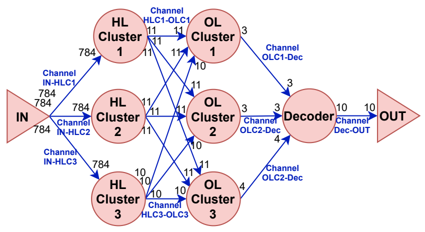
The parallelization degree of the SDF graph and the induced number of communication channels depend directly on the choice of number of clusters. Raising the number of clusters raises the degree of parallelism, but also raises the number of communication channels, which may reduce the computation time while raising the communication time of the application, and therefore impact the global execution time of the application. Exploring the clustering of the application (the number of clusters generated from every



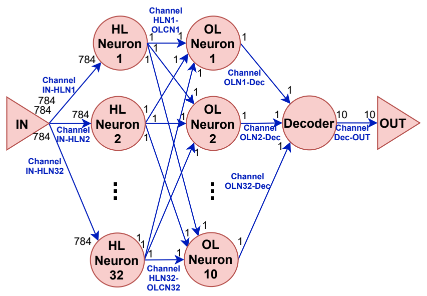
(a) Considered neural network



(b) SDFG using layer grain



(c) SDFG using 3-cluster grain, layers are divided in 3 clusters of neurons



(d) SDFG using neuron grain

Figure 5: Description of a neural network in SDF graphs using various levels of granularity

layer) is for this reason necessary.

Once the application captured in SDF, the mapping and scheduling of the SDF graph on the platform can be performed. The application is self-scheduled: the scheduling is established based on the dependency between actors. Therefore, no scheduling step is required once the mapping of the application on the platform is complete. In the mapping step, the actors of the identified SDF graph are mapped on the processing elements available on the platform. The communication channels between actors are mapped on the shared memory. The processing elements will read and write the tokens necessary for the execution of the actors on the shared memory. During the execution of actors, the processing elements cannot be interrupted. The instruction and data for the execution are available in the local memory of the tile. The local memory can only be accessed by the local processing element.

For a given SDF graph, several mappings of the application are possible. An example of mapping for a given neural network application captured in SDF is shown in Figure 6. Exploring the clustering of the application along with the mapping/scheduling allows finding optimized implementations based on timing constraints. The analysis of the mapping/scheduling is performed using our model of performance, presented in the following section.

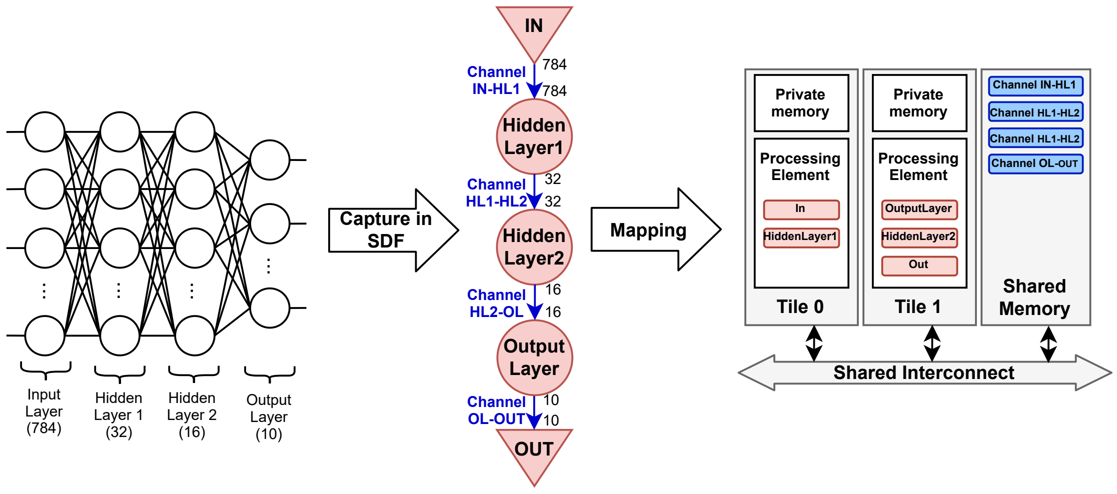


Figure 6: A neural network application captured using layer grain in the SDF model of computation, then mapped on the model of architecture

3.5 Model of Performance (MoP)

In order to predict the performance of a given mapped and self-scheduled SDF graph (SDFG) on the considered platform, models of performance are used. Our models of performance are system level models captured in SystemC. They integrate a computation time model and a communication time model. The computation time model allows predicting the execution time of actors based on their measured computation time. The communication time model allows predicting the time needed for data transactions on the platform. The Figure 7 presents an overview of our SystemC models, and the Figure 8 presents how the performance of a SDF graph mapped on the targeted platform can be predicted by our models of performance.

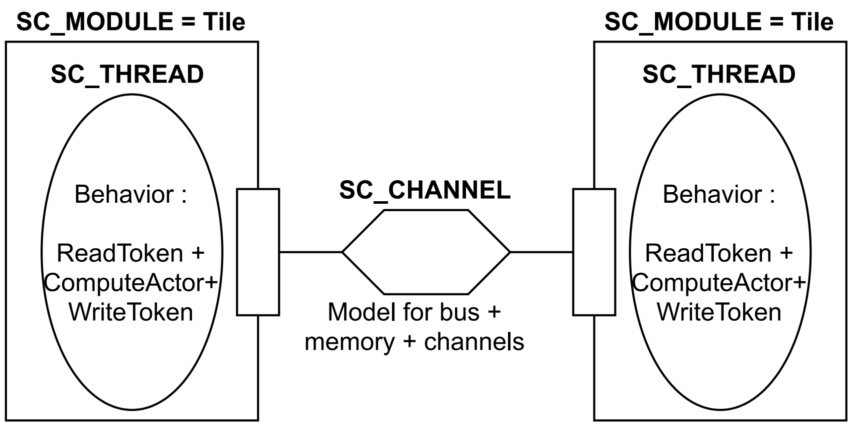


Figure 7: Overview of SystemC models architecture

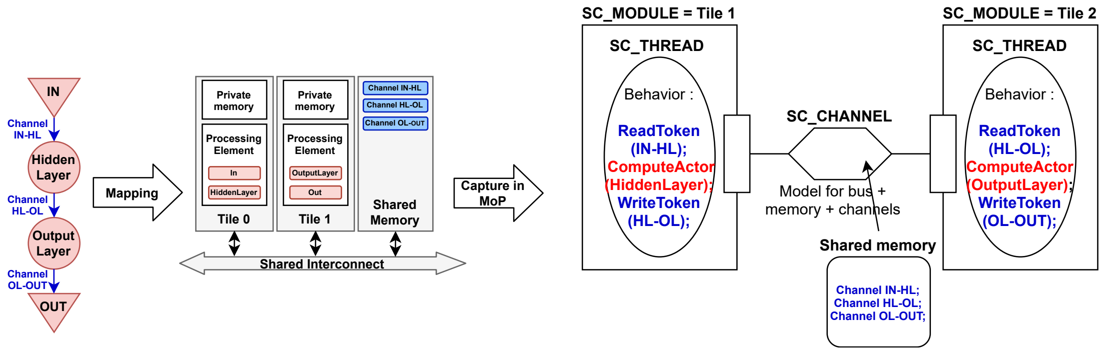


Figure 8: Schematic of how a mapped SDFG is captured in our SystemC model

The *ReadToken* and *WriteToken* operations delays are computed by the communication time model, based on channel size, concurrent accesses, and also bus and memory properties. The *ComputeActor* operation delay is computed using the computation time model. Two different computation time models can be used to predict the delay of *ComputeActor*:

- **Model A:** the model from our previous workflow, which predicts the delay based on a Gaussian law applied to measured actor execution time,
- **Model B:** the analytical model developed in this work, which allows predicting the execution time of any fully connected layer based actor, that is composed of a set of neurons.

The **model A** is characterized by measurement: the actors of a given SDF graph are first executed on the targeted platform, and their execution time is measured. Due to data dependant paths, the execution time of actor can vary from one iteration to another, based on the input data. Each measured execution times for a given actor is therefore associated with its probability to occur, computed based on the statistical appearance of the execution time during measurement. The execution times with their associated probability are then used by the model of computation to predict the execution time of actor while taking the execution time variability into account. Once the characterization of the model of computations performed using the execution time of actor, using it in association with the communications time model allows predicting the execution time of any mapping of the SDF graph on the targeted platform. Thanks to its fast prediction of execution time along with prediction of variability caused by data dependant paths, this model proved to be accurate and fast to execute for execution time prediction of video processing applications such as a Sobel filter application and a JPEG decoder application [17] [20]. However due to the need to characterize the model again when targeting a new SDF graph, this model for computation execution time lacks of versatility.

For this reason, and because neural network applications do not present data dependant paths, as shown in subsection 3.2, a new model for computation time has been built. The **model B** computes the delay based on the formula given for this class of applications in Equation 1, which allows computing the delay for a neuron. In this equation, D_{neuron} is the delay in clock cycles needed to compute a neuron. N_{n-1} is the number of neurons contained in the previous layer of the MLP. D_{rd_w} is the delay associated with the reading of the weights. D_{sum} is the delay to perform the sum. The weight reading operation and the sum operation must be performed for each input of the layer, that is for each neuron contained in the previous layer and one more time for the bias, so $N_{n-1} + 1$ times. $D_{act.func.}$ is the delay to compute the activation function. To compute the execution time of a layer, or any cluster of neuron, one simply needs to multiply the delay required to compute a neuron by the number of neurons contained in the layer, or in the cluster. This is highlighted in Equation 3. D_{layer} is the delay needed to compute a layer (or a cluster of neurons) of a MLP. NN_n is the number of neurons inside the considered layer (or cluster of neurons). D_{base} as set in Equation 2 is the base delay, composed of the sum of the read weight delay D_{rd_w} and sum delay D_{sum}.

$$D_{neuron} = (N_{n-1} + 1) \times (D_{rd\_input} + D_{rd\_w} + D_{sum}) + D_{act.func.} \tag{1}$$

We set

$$D_{base} = D_{rd\_input} + D_{rd\_w} + D_{sum} \tag{2}$$

The delay to compute a layer (or cluster of neuron) is:

$$D_{layer} = N_n \times D_{neuron} = N_n \times ((N_{n-1} + 1) \times D_{base} + D_{act.func.}) \tag{3}$$

The D_{base} and $D_{act.func.}$ delays can be characterized based on measured execution time of neurons.

Due to the strict separation of communication and computation offered by the SDF model of computation, and by construction of our model of architecture, the proposed model of performance is scalable: once the characterization performed, the model of performance is able to predict any configuration with any number of tiles with accurate results. Different possible mappings are simulated using the proposed model of performance. The actual execution times are then measured to compare with the results of the models.

The next section presents the measurements of neural networks execution time obtained using our measurement infrastructure, and the predictions of our models of performance.

4 Results

This section presents the preliminary results of our workflow. First the platform used to measure the execution time of neural networks captured in SDF is presented. Then the results of our prediction models are presented.

4.1 Measurement infrastructure and measured performance

To characterize our models of performance, and to validate their prediction, we rely on a measurement infrastructure, used to measure the execution of actors identified in SDF graphs. This measurement infrastructure is implemented in a platform that follows the hypothesis done on the model of architecture, as presented in Fig. 4. For the experiment, the platform is implemented on a ZC702 board, which features a Zynq7000 FPGA. In future work, the ZCU102 board which features a Ultrascale MPSoC+ component will also be considered to explore configurations with more tiles and more memory.

A tile is composed of a MicroBlaze processing element, and a local memory implemented as BRAM. The platform is composed of 7 tiles connected via an AXI shared interconnect. The tiles can access a shared memory, via the shared interconnect. The measurement infrastructure is composed of two parts: the computation time measurement part and the communication time measurement part. The computation time measurement part contains a set of functions to measure the computation delay of actors. When a SDF graph starts (when actor IN is executed), the system issues a start signal. When it ends (when actor OUT is executed), the system issues a stop signal. These signals are handled directly in the code deployed on the processing elements. Based on the elapsed time between the start and stop signals, the functions inside the measurement infrastructure computes the execution time of the SDF graph. The computation time measurement infrastructure then sends the measured execution time through an UART communication channel. The communication time measurement part relies on the Integrated Logic Analyzer (ILA) provided by Xilinx to measure the communication time on the shared interconnect. The platform with the measurement infrastructure is presented in figure 9.

The execution time of the XOR application (as presented in subsection 3.2, Figure 3) have been measured, as shown in Table 2. The execution time using iteration-grain (the neural network algorithm as whole is an actor) is compared with the execution time using the layer-grain. As depicted in the table, the execution time using layer grain on 1 and 2 MicroBlazes is 3% longer than the iteration grain. This longer execution time is due to the apparition of more communication channels when using a finer grain, which bear a cost in communication time on the platform. The parallelism from neural network applications cannot be exploited using the layer grain, as explained in subsection 3.4. For this reason, executing layer grain on 1 MicroBlaze and 2 MicroBlazes bears almost identical execution time (the 2 MicroBlaze implementation is slightly faster, with an acceleration of 0.27%).

Using the measurement infrastructure, it is also possible to measure the execution time of actors. The measured execution time of the layers identified in the XOR layer grain SDF graph are given in table 3. The execution time of the *HiddenLayer* actor is 2.80 times longer than the execution time of the *OutputLayer* actor. The *HiddenLayer* actor contains the computation of 3 neurons, with each 2 inputs (6 connections in total), whereas the *OutputLayer* actor contains the computation of 1 neuron, with 3 inputs. Less computations are performed in the *OutputLayer* actor, and for this reason the execution time of this actor is shorter. These measured execution times of actors are

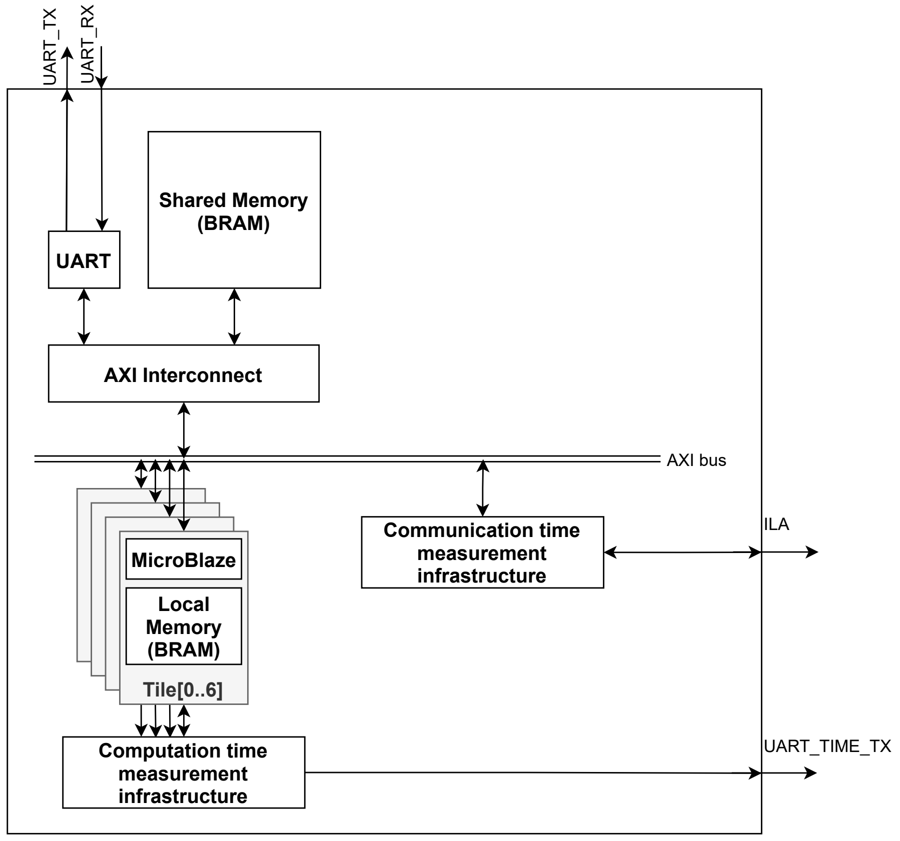


Figure 9: Platform implemented on Zynq7000 FPGA for measurements

used to characterize our models of performance.

The measurement of execution for the MNIST application is still undergoing. Implementing neural network applications such as the MNIST application on the targeted platform is tedious. These applications require important amount of memory resources, which are not available on the platform. The Table 4 shows the needed memory resources to store the parameters of the considered neural network applications, compared to the actual capacity of the targeted FPGA boards. Two MNIST algorithms are considered in this table, the MNIST 784-300-10 contains one hidden layer composed of 300 neurons, and an output layer composed of 10 neurons, and the MNIST 784-32-16-10 is composed of two hidden layers of respectively 32 and 16 neurons, and an output layer of 10 neurons.

The memory usage required for the MNIST 784-300-10 algorithm is too high to be implemented on both considered boards. For this reason, another algorithm is considered. The MNIST 784-32-16-10 requires 400kB for its parameters to be stored on the targeted platform. In addition to the parameters, the LibFANN libraries functions, the main execution code of the application and the input MNIST image must be provided, which approximately require 100kB of memory. In addition to this, some memory should

Input[0]	Input[1]	Accuracy	IT-grain 1MB	L-grain 1MB	L-grain 2MB*
-1	-1	97.17 %	103914 p.c.*	107509 p.c.*	107224 p.c.*
1	-1	98.02 %	103695 p.c.*	107287 p.c.*	107002 p.c.*
-1	1	97.67 %	103571 p.c.*	107111 p.c.*	106826 p.c.*
1	1	92.74 %	103726 p.c.*	107291 p.c.*	107006 p.c.*

Table 2: Measured execution time for the XOR multi-layer perceptron application in processor cycles (p.c.*) - each value depicted in the table has been measured constant over a large (+1000) number of iterations of the application. IT-grain means iteration grain, L-grain means layer-grain and MB means MicroBlaze

Input[0]	Input[1]	HiddenLayer actor*	OutputLayer actor*
-1	-1	79054 p.c.*	28311 p.c.*
1	-1	79087 p.c.*	28056 p.c.*
-1	1	78766 p.c.*	28201 p.c.*
1	1	78974 p.c.*	28173 p.c.*

Table 3: Measured execution time for the actors of the XOR multi-layer perceptron application in processor cycles (p.c.*) - each value shown in the table has been measured constant over a large (+1000) number of iterations of the application.

be left available for the execution of the code on the MicroBlaze (approximately 50 kB as measured on the considered applications). This new algorithm, along with the required functions and input for its execution, fits on the total amount of resources available on the targeted platforms. However, the whole BRAM memory surface cannot be allocated to one processing element on the targeted platforms. For instance, on board ZC702, the maximum BRAM that can be allocated to a MicroBlaze's local memory is 512kB. For this reason, the execution of the MNIST 784-32-16-10 is not possible on the current platform without reducing the size of the parameters, the size of the LibFANN library or the size of the memory needed by the MicroBlaze to execute its program.

	BRAM available in ZC702		BRAM available in ZCU102		Memory needed to run the considered applications		
	Total	Max. for 1 MicroBlaze	Total	Max. for 1 MicroBlaze	XOR	MNIST 784-300-10	MNIST 784-32-16-10
Size	1.12 MB	512 kB	7.3 MB	2 MB	∼155 kB	∼15MB	∼550 kB

Table 4: Memory available on targeted FPGAs (Block RAM), and memory needed to store parameters for considered neural network applications

4.2 Performance prediction results

Our probabilistic SystemC model produces a distribution of execution times (in processor cycles) with their associated probability. The predicted execution time for the XOR application are presented in the Table 5. The prediction are performed using both model A and model B as presented in Section 3.5. The models achieve high accuracy prediction, by having nearly identical prediction error smaller than 1% compared to measured execution time on platform.

Experiment	Exp 1MB	ModA 1MB	ModB 1MB	Exp 2MB	ModA 2MB	ModB 2MB
Average execution time (p.c.*)	107289	107405	107398	107004	107389	107384
Model error	/	+0.11%	+0,10%	/	+0.36%	+0,36%

Table 5: Measured and predicted execution time in processor cycles (p.c.*) for the XOR application on 1 MicroBlaze (MB) and 2 MicroBlazes, using layer grain (average over 500 iterations). The model A (ModA) and model B (ModB) are presented in Section 3.5

As the MNIST application was not characterized yet, it is impossible to predict its execution time using the previous computation time model. However, it is possible to predict the execution time using the new analytical computation model. The new model only need to be characterized once for a given neural network execution, and is then, by construction, able to predict the execution time of any neural network. The predicted execution times for the MNIST application using the new computation time model are given in table 6 and table 7. When using layer grain, using more MicroBlazes to execute the application does not accelerate its execution: the layer grain do not extract the parallelism from the neural network application. However, when using cluster3 grain (as introduced in subsection 3.4, figure 5c), the hidden layers of the neural network are divided in 3 actors, allowing to accelerate the execution of the application with a parallelization degree of 3. Therefore, as highlighted in the predicted execution times, the application can be accelerated by deploying it to up to 3 MicroBlazes. The cluster3 SDF graph deployed on 3 MicroBlazes is 2.8 times faster to execute than the layer SDF graph. The reason why it is not 3 times faster is because of the addition of more communication channels, which induce longer communication times on the platform.

The presented results for the prediction of the MNIST application execution time on the targeted platform require to be validated through experiment, by measuring the execution of the MNIST application using the measurement infrastructure.

ModB MNIST Layer 1MB	ModB MNIST Layer 2MB	ModB MNIST Layer 3MB
49252288 p.c.*	49252275 p.c.*	49252262 p.c.*

Table 6: Predicted execution time in processor cycles (p.c.*) of MNIST application using layer grain. Three mappings are considered: using 1, 2 and 3 MicroBlazes (MB)

ModB MNIST Cluster3 1MB	ModB MNIST Cluster3 2MB	ModB MNIST Cluster3 3MB	ModB MNIST Cluster3 6MB
49363231 p.c.*	33840723 p.c.*	17003282 p.c.*	17002754 p.c.*

Table 7: Predicted execution time of MNIST application in processor cycles (p.c.*) using Cluster3 grain as introduced in subsection 3.4, Figure 5c. Four mappings are considered: using 1, 2, 3 and 6 MicroBlazes (MB)

5 Future work

This section presents the work that has been identified for the second year of the PhD. It presents the work that needs to be done to complete our workflow's results, and the research directions that has been identified.

During the first year, our model of performance has been updated for neural network inference time prediction, and allowed predicting the performance of the MNIST application. However, the new model has not been validated by comparing the predictions to the measured execution time. For this reason, a key focus for the upcoming year is to complete the timing analysis study. To complete the timing analysis study, the following tasks must be performed:

- Obtain measurement for the MNIST application executed on ZC702,
- Validate the established computation model and our models of performance by comparing the prediction with the measured values.
- Finish porting the platform to ZCU102 board to enable more testing configurations, with more memory and number of tiles available.

In addition to this work, research topics that can be explored in the upcoming years of the PhD have been identified:

- Extension of the modeling approach to power & energy consumption prediction,
- Model of architecture update with caches and DDR (possible option to be added to extend the use-cases),
- Specification and evaluation of more complex partitioning techniques of neural networks (e.g. in-layer and multi-layer clustering techniques),
- Specification and evaluation of other neural network topologies such as Convolutional Neural Networks (CNN).

The main direction chosen is the extension of the modeling flow to prediction of power & energy consumption. The proposed flow should enable the evaluation of power management strategies available on multi-core platforms for ANN inference. The first step should therefore consists in performing a state-of-the-art and specification of the existing techniques used to manage power on MPSoC platforms. Then a power model of the platform (tiles, shared memory) should be specified and developed. The proposed model should then be integrated with the SystemC simulation flow with timing modeling in order to enable energy consumption prediction. To test, validate and evaluate the power & energy modeling approach, a power measurement infrastructure for ANNs on the targeted MoA should be developed and the power consumption of neural networks should be measured and confronted with the predictions.

References

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] Paul Palomero Bernardo, Christoph Gerum, Adrian Frischknecht, Konstantin LÃ¼beck, and Oliver Bringmann. Ultratrail: A configurable ultralow-power tc-resnet ai accelerator for efficient keyword spotting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39:4240–4251, 2020.

[3] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 367–379, 2016.

[4] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, July 2018.

[5] Ioannis Galanis, Iraklis Anagnostopoulos, Chinh Nguyen, Guillermo Bares, and Dona Burkard. Inference and energy efficient design of deep neural networks for embedded devices. *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 36–41, 2020.

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[7] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[8] Yi-Che Lee, Ting-Shuo Hsu, Chun-Tse Chen, Jing-Jia Liou, and Juin-Ming Lu. Nnsim: A fast and accurate systemc/tlm simulator for deep convolutional neural network accelerators. In *Electronics*, pages 1–4, Hsinchu, Taiwan, 2019. IEEE.

[9] Daniel Luenemann, Maher Fakih, and Kim Gruettner. Capturing neural-networks as synchronous dataflow graphs. In *MBMV 2020 - Methods and Description Languages for Modelling and Verification of Circuits and Systems; GMM/ITG/GI-Workshop*, pages 1–10, Stuttgart, Germany, 2020. VDE.

[10] Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi. Design space exploration of fpga-based deep convolutional neural networks. *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 575–580, 2016.

[11] Steffen Nissen. Implementation of a fast artificial neural network library (fann), `https://github.com/libfann/fann`, 12 2003.

[12] Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311 – 1314, 2004.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[14] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[15] Satyajit Sinha. State of iot 2021: Number of connected iot devices growing 9billion globally, cellular iot now surpassing 2 billion. `https://iot-analytics.com/number-connected-iot-devices/`, September 2021. Accessed: 2021-11-18.

[16] Salita Sombatsiri, Jaehoon Yu, Masanori Hashimoto, and Yoshinori Takeuchi. A design space exploration method of soc architecture for cnn-based ai platform. *SASIMI 2019 Proceedings*, 2019.

[17] Ralf Stemmer, Hai-Dang Vu, Kim Grüttner, Sébastien Le Nours, Wolfgang Nebel, and Sébastien Pillement. Towards Probabilistic Timing Analysis for SDFGs on Tile Based Heterogeneous MPSoCs. In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, number paper 59 in ERTS 2020, Toulouse, France, January 2020.

[18] Foivos Tsimpourlas, Lazaros Papadopoulos, Anastasios Bartsokas, and Dimitrios Soudris. A design space exploration framework for convolutional neural networks implemented on edge devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37:2212–2221, 2018.

[19] Stylianos I. Venieris and Christos-Savvas Bouganis. fpgaconvnet: A toolflow for mapping diverse convolutional neural networks on embedded fpgas, November 2017.

[20] Hai-Dang Vu. *Fast and Accurate Performance Models for Probabilistic Timing Analysis of SDFGs on MPSoCs*. Theses, Université de Nantes, March 2021.