A Hybrid Performance Prediction Approach for Fully-Connected Artificial Neural Networks on Multi-Core Platforms

Quentin DARIOL[1,2], Sebastien LE NOURS[1], Sebastien PILLEMENT[1], Ralf STEMMER[2], Domenik HELMS[2], and Kim GRÜTTNER[2]

1. Nantes Université, IETR UMR CNRS 6164
2. German Aerospace Center (DLR-SE)
quentin.dariol@etu.univ-nantes.fr

Abstract. Predicting the performance of Artificial Neural Networks (ANNs) on embedded multi-core platforms is tedious. Concurrent accesses to shared resources are hard to model due to congestion effects on the shared communication medium, which affect the performance of the application. Most approaches focus therefore on evaluation through systematic implementation and testing or through the building of analytical models, which tend to lack of accuracy when targeting a wide range of architectures of varying complexity. In this paper we present a hybrid modeling environment to enable fast yet accurate timing prediction for fully-connected ANNs deployed on multi-core platforms. The modeling flow is based on the integration of an analytical computation time model with a communication time model which are both calibrated through measurement inside a system level simulation using SystemC. The ANN is described using the Synchronous DataFlow (SDF) Model of Computation (MoC), which offers a strict separation of communications and computations and thus enables the building of separated computation and communication time models. The proposed flow enables the prediction of the end-to-end latency for different mappings of several fully-connected ANNs with an average of 99.5 % accuracy between the created models and real implementation.

Keywords: Performance prediction · Multi-processor systems · SystemC simulation models · Artificial neural networks

1 Introduction

The Internet-of-Things (IoT) market is continuing to grow, as the number of connected devices is expected to reach 27 billion by 2025 (an increase of more than 200 % compared to 2020) [5]. Along with this increase, the need for smart embedded devices is emerging and efficient execution of AI algorithms such as

This work has been funded by the WISE consortium, France (pSSim4AI project) and by the Federal Ministry of Education and Research (BMBF, Germany) in the project Scale4Edge (16ME0465).

Artificial Neural Networks (ANNs) has become thus a key challenge. To avoid the loss in throughput and energy caused by data transmissions when executing AI algorithms on distant servers, the focus is nowadays on their deployment on edge devices. Among edge devices, multi-core platforms are widely used due to the versatility they offer. However, ANNs are computation-intensive applications that require important amount of resources while embedded platforms are limited in memory and computing capacity and bear strict energy constraints. In this context, an intensive evaluation of ANN implementations on multi-core platforms is needed early in the design process to optimize solutions that meet performance constraints.

Several approaches have already been proposed to allow fast evaluation of ANN deployment on embedded platforms. Some approaches focus on the implementation and performance measurement on real target. The disadvantage of this process is that it represents a time consuming and error prone process as the ANN must be trained and then deployed on the real platform to evaluate the achieved performance. Because of long development cycle, the coverage of the design space is limited while using this evaluation technique. Other approaches focus on building analytical models to predict the achieved performance and possible influence of captured design parameters. These models tend however to be inaccurate when targeting architectures with complex effects due to shared resources usage. In the context of our work, execution of highly parallel applications such as ANNs on multi-core platforms may cause concurrent accesses from processing elements to shared memories which lead to performance loss. Modeling and predicting the impact of communications on the execution time of the application is a key element to enable accurate performance prediction for a large scale of multi-core architectures.

In this paper we propose a hybrid performance prediction workflow for fully-connected ANN deployment on multi-core platforms. This modeling approach enables the estimation of the end-to-end latency of the application while predicting the importance and impact of communications. The proposed models can be used at several levels of granularity to describe the ANN in combination with different mappings on the target platform. This offers the ability to compare configurations and thus explore the design space to find solutions that optimize timing. In our experiments, we observed that the prediction accuracy of the created models when compared with measurements performed on a real platform is more than 99.5 % on average for 21 different scenarios. The proposed flow is tested with three different fully-connected ANNs, for which several partitionings and mappings are considered on architectures containing up to 7 cores. The platform used to measure real execution times is implemented on two different FPGAs.

The paper is organized as follows: Section 2 presents related work and how the work presented in this paper extends our previous workflow for performance prediction of ANNs. Section 3 presents the features of the proposed workflow. Section 4 presents the experimental setup and results. Section 5 summarizes what is presented in the paper and presents our future work directions.

2 Related work

Several approaches have been carried out to tackle the challenge of performance prediction for ANNs on embedded platforms. The approach in [3] focuses on performing Neural Architecture Search on embedded GPU platforms under accuracy, timing and energy constraints. Another notable approach is [18], which proposes a latency and energy evaluation flow to find optimized mappings of neural networks on edge devices. These approaches focus on the evaluation of the neural network through implementation and testing on real prototype. Such evaluation approaches require to systematically train and execute considered ANNs on the platform. To alleviate the development effort, models of performance are needed.

[6] [1] [4] focus on proposing performance models for the exploration of parameters of ANN topology such as the number of layers and the number of neurons. Their models allow predicting the impact of reduction techniques on ANN's latency. These approaches aim however at optimizing the inference of ANNs on single processor systems and do not enable prediction of timing for parallel execution of ANNs. Other approaches propose analytical modeling techniques to explore and optimize performance for highly parallel architectures [2] [8] [13] [16]. In these approaches, the emphasis is put on the exploration of architectural alternatives used to implement an hardware accelerator for ANN inference. Because these architectures are equipped with dedicated high-speed communication channels between functions, the part of communications is irrelevant in overall inference time, and is not considered in the proposed models of performance. Multi-core platform's architectures are conceived to bear a high versatility in the regard that they can execute several different types of applications. They rely for this reason on more general purpose communication bus which bear a higher part in the application's latency and energy, and must be modeled. In our work we aim at enabling performance prediction for multi-core processing platforms by proposing an innovative modeling approach that combines simulation, analytical models and partial characterization through measurement.

The workflow presented in this paper is based on previous work presented in [17]. The previous workflow was based on a probabilistic simulation model and demonstrated to deliver fast yet accurate analysis on video processing applications. This paper presents the following contributions: (1) an hybrid analytical computation time model for fully-connected ANNs on processing cores. This model only need to be calibrated once and can then predict accurately the execution time of any partitioning of fully-connected ANNs, (2) the integration of this analytical computation time model in a simulation model with a communication time model to predict the execution of the application executed on several processing cores, (3) the validation of the modeling approach by comparing the predicted execution time with real duration measured on an execution platform.

3 Proposed modeling approach

A schematic of the proposed workflow is given in Fig. 1. This figure highlights the steps to build performance models used to predict execution time of fully-

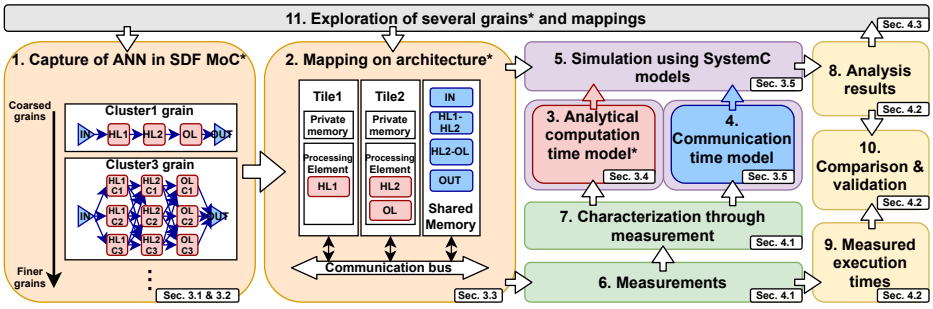


Fig. 1: Schematic of the modeling flow. This flow aims at enabling performance prediction for fully-connected ANNs on multi-core platforms. The processes marked with the * symbol are novel features to our flow.

connected ANNs deployed on multi-core platforms. The first activity in the proposed flow is captured in orange on the figure. It focuses on the capture of the ANN in a model of computation and its mapping on the target platform (discussed in Sections 3.1, 3.2 and 3.3). The next activity (highlighted in purple) focuses on the simulation and prediction of performance using the developed SystemC models (Sections 3.4 and 3.5). The third activity (highlighted in green) focuses on the measurement of execution time in order to characterize the timing models proposed in this flow (Section 4.1). The fourth activity (highlighted in yellow) and final activity (highlighted in grey) focus respectively on the validation of the proposed models by comparing the predictions to the measured execution times and the exploration of several grains and mappings to optimize latency using the proposed models (Sections 4.2 and 4.3).

3.1 Target application: fully-connected ANNs

ANNs are applications often used to address data classification problems. Among neural networks, several algorithms are available. The most classical neural network algorithm is the fully-connected network, also called Multi-Layer Perceptron (MLP) [15]. The MLP consists of a set of neurons organized in layers (input layer, hidden layers and output layer). In such a fully-connected network each neuron is connected to every neurons of the previous layer. The operations required to compute the output of a neuron are presented in Equation (1) from [14]. In this equation, φ is the activation function of the neuron. n is the number of inputs of the neuron and thus the number of neurons from the previous layer due to the MLP definition. The x_i and w_i are respectively the inputs and the weights of the neuron. B is the bias of the neuron. Because neurons of a same fully-connected layer are independent from one another, it is possible to parallelize their execution.

$$output = \varphi\left(\sum_{i=1}^{n} w_i x_i + B\right) \quad (1)$$

Our approach focuses solely on the optimization of the execution of the ANN on a embedded platform. The training of the ANN is not considered in the scope of this work.

3.2 Modeling ANNs with SDF

To ease the analysis and exploration of studied ANN implementations on the target platform, Synchronous Data Flow (SDF) Model of Computation (MoC) [10] is used. SDF is used to describe the data flow between actors via communication channels. It offers a strict separation of computation and communication (read, write) phases of actors. Computation and communication separation eases the performance prediction process by allowing building separated computation time model and communication time model. To apply the SDF MoC to fully-connected ANNs, we define as actors a set of neurons, called clusters. In this work we call cluster a given set of neurons from the same layer, which execution is modeled as an actor. The number of actors (i.e. the number of clusters) by layer is established based on the desired granularity. *ClusterN* denotes an organization where each layer is partitioned into N clusters of neurons. The communications channels of the SDF graph correspond to the set of data exchanged between clusters. The source (IN channel) of the SDF graph is the input data that needs to be classified by the ANN. The sink (OUT channel) of the SDF graph is the output result of the ANN. An example of how an ANN is described in a SDF graph is given in Fig. 2. In this case the actors of the SDF graph are the layers of the ANN and the channels between actors are the outputs of the layers.

Exploring the partitioning of ANNs using various grains is necessary. In this work, the level of granularity of a SDF graph is defined by the number of clusters. Coarser grains inhibit the acceleration that can be exploited from the parallelism of ANNs, whereas finer grains invoke numerous communication channels, which overload the communication bus of the considered platform. [11] presented a way of capturing ANN in SDF using several levels of granularity. In the example presented in Fig. 2 the ANN is described using the *Cluster1* grain, which is the more coarse level of granularity considered. Using this level of granularity, the actors considered in the SDF graph are the layers of the ANN. The finest granularity considered is the neuron grain, where the actors considered in the SDF graph are the neurons of the ANN. Intermediate grains are considered, which separates each layer into sets of actors. The number of actors (i.e. the number of clusters) by layer is established based on the desired granularity. The partitioning of layers into clusters is done in such a way that each actor contains nearly the same number of neurons. This allows for an equitable distribution of workload between actors. In Fig. 1, *Cluster1* and *Cluster3* graphs obtained from the same ANN are presented. Each layer of the ANN forms 1 actor in the *Cluster1* graph and 3 actors in the *Cluster3*. The *Cluster1* graph includes 3 actors and 4

communications channels, whereas the *Cluster3* graph includes 9 actors and 24 communication channels.

3.3 Mapping on the targeted architecture

The considered model of architecture is composed of a set of tiles where a tile is one single-core processor with private data and instruction memories. 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 the considered platform is given in Fig. 2. In this work, tiles are assumed to be identical except for private memory sizes. The target platform is thereby assumed to be homogeneous.

In the mapping step, the actors of the identified SDF graph are mapped on the tiles available on the platform. The communication channels between actors are mapped on the shared memory. The tiles will read (*ReadTokens()* statement) and write (*WriteTokens()* statement) the data necessary for the execution of the actors using the communication channels. During the execution of actors, the processing elements cannot be interrupted. The application is self-scheduled: the scheduling is established based on the dependency between actors. For a given SDF graph, several mappings of the application are possible. An example of mapping for a given ANN captured in SDF is shown in Fig. 2. In this example, the *HiddenLayer1* of the SDF graph is mapped on one tile while the *HiddenLayer2* and *OutputLayer* actors are mapped on a second tile. Comparing levels of granularity and mappings of the application allows finding solutions that jointly optimize timing properties and number of tiles.

3.4 Computation time model

We propose an analytical computation time model to predict the execution time of clusters of neurons from the same layers of the ANN. The analytical model is established based on the computations performed in order to set the output of a fully connected layer of an ANN (Equation (1)). According to this equation, the execution time of a neuron from a fully-connected layer depends linearly on the number of inputs it has. From this information we can deduce the theoretical delay D_{neuron} required to compute the output of an artificial neuron. This delay is given in Equation (2). In this equation n is the number of inputs of the neuron, D_{Σ} is the delay needed to compute the multiplication $w_i x_i$ and perform the associated sum, and D_{φ} corresponds to the delay needed to add the bias and compute the activation function of the neuron.

$$D_{neuron}(n) = nD_{\Sigma} + D_{\varphi} \tag{2}$$

Because neurons from a same layer are independent, the execution time of a cluster of neurons depends linearly on the number of neurons it contains. As presented in the code of actor *HiddenLayer2* in Fig. 2, the operations to compute

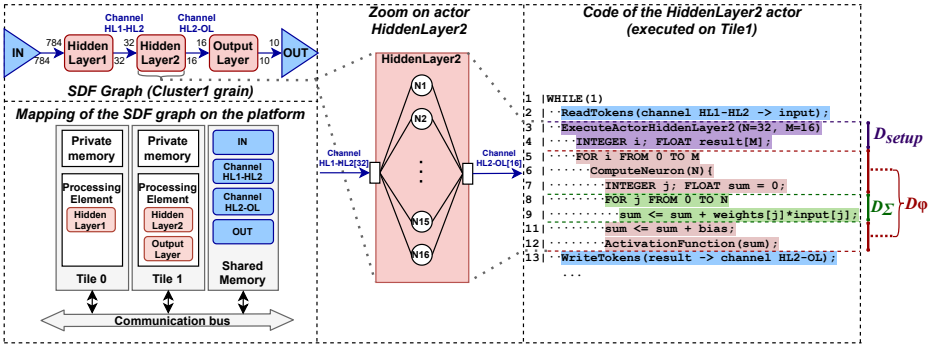


Fig. 2: Illustration of a coarse grain SDF graph (*Cluster1* as introduced in Section 3.2) mapped on the platform (introduced in Section 3.3). The content of the HiddenLayer2 actor and the computations performed inside it are detailed. The delays needed to perform the operations inside the actor are predicted using the highlighted analytical formula which is presented in details in Section 3.4.

neurons are repeated m times, where m is the number of neurons contained in the considered actor. The delay to compute these operations is therefore $m.D_{neuron}$. In addition to this delay, the initialization of the variables and the call and return procedures of functions provoke a delay named D_{setup}. Equation (3) presents the formula of the delay needed to compute a cluster of neurons.

$$D_{cluster}(n,m) = mD_{neuron}(n) + D_{setup} = nmD_{\Sigma} + mD_{\varphi} + D_{setup} \quad (3)$$

The D_{Σ}, D_{φ} and D_{setup} unitary delays can be characterized based on measured execution time of neurons. These delays only need to be characterized once for any given processing element, as the sum and multiplication operations and the activation functions considered in this work present insignificant execution time variability on the target platform. Once the unitary delays characterized, the proposed computation time model can therefore predict accurately performance for any considered level of granularity used to describe ANN applications.

3.5 Simulation using SystemC models

Our performance models were created with the SystemC language with the same organization as in [17]. In order to predict the performance of a given mapped SDF graph (SDFG) on the considered platform, both the computation time model presented in Section 3.4 and the message level communication time model presented in [19] are used. These two models are integrated in a behavioral description of each tile which describes the sequence of the mapped computation and communication statements. When an actor is being executed in simulation, the analytical computation time model is called to compute the corresponding delay. During communications through channels, the communication time model

is called to compute the delays of communications on the platform even in the case of contentions at shared resources. The proposed model of performance is by construction composable: the number of tiles do not modify the nature of the model. It is therefore scalable regarding the mappings and the level of granularity used to describe the ANN. Different mappings are simulated and the obtained results are compared with measurements.

4 Experiments

4.1 Testing configuration

To characterize our models of performance and to validate their prediction we implemented a hardware platform that followed the hypothesis of the proposed model of architecture, as presented in Section 3.3. The platform is implemented on both Xilinx Zynq-7000 and UltraScale MPSoC+ FPGA boards to test the portability of the modeling flow to several FPGAs. The observed variation in execution time was lesser than 0.05 % when comparing the durations measured on both platforms.

Each MicroBlaze is equipped with a Floating Point Unit (FPU) and a hardware multiplier. The platform is composed of 7 tiles connected via an AXI shared interconnect. The tiles communicate through a shared memory via the shared interconnect. This platform integrates a timing measurement infrastructure, used to measure the computation time and communication time for SDF graphs executed on the platform. The measurement infrastructure was introduced in previous work. It is composed of two parts: the communication time measurement part presented in [19], and the computation time measurement part. The computation time measurement part measures the computation time of the SDF actor with code instrumentation. When an SDF graph iteration is started, the system issues a start signal. When it ends, the system issues a stop signal. Based on the elapsed time between the start and stop signals, the execution time of the SDF graph is measured.

In order to calibrate the proposed analytical computation time model, the base delays identified in Equation (3) have been measured on the experimental platforms by varying the number of inputs of neurons and the number of neurons in clusters. The results showed that for fully-connected ANNs executed on the platform tile, the execution time varies linearly based on the number of inputs of neurons, and based on the number of neurons executed inside the cluster. We set $D_{\Sigma} = 47$, $D_{\varphi} = 146$ (ReLU activation function), and $D_{setup} = 39$ processor cycles. In this work, the ANNs are implemented using the lightweight open source library LibFANN [12] which enables the training and execution of MLPs in C programming language. All the tested ANNs used the ReLU activation function and the *float32* precision. To verify and validate our modeling approach, we considered three fully-connected ANNs with different topologies. Two ANNs were developed and trained to perform digit recognition using the MNIST data set introduced in [9]. The first algorithm is a 3 layers fully connected ANN with a input layer of size 784 (28×28 grey scale pictures), a hidden layer containing

10 neurons and a output layer containing 10 neurons. This ANN reached 85 % prediction accuracy. This algorithm is reffered as *MNIST 784-10-10* in the rest of the paper. The second algorithm is a 4 layers fully connected ANN with a input layer of size 784, a first hidden layer containing 32 neurons, a second hidden layer containing 16 neurons and an output layer containing 10 neurons. It is referred as *MNIST 784-32-16-10* in the rest of the paper. This ANN was trained and reached 89 % prediction accuracy. We also considered a third ANN trained using the German Traffic Sign Recognition Benchmark (GTSRB) [7]. Different filters are applied to the images from the GTSRB data-set to render the building, training and deployment of the ANN on the considered platform possible. These filters include a max pooling and a grayscale filters to reduce the size of the input images to 576 pixels ($24 \times 24 \times 1$) and thus reduce the memory usage. This ANN is composed of a input layer of size 576, two hidden layers both containing 30 neurons and a output layer of 43 outputs. It is referred as *GTSRB 576-30-30-43*. Do to the simplification/optimization to allow the implementation of this application, this ANN reached less than 20 % prediction accuracy. In order to test the accuracy of the proposed models for several grains, a set of SDF graphs which express different levels of granularity were used. Several mappings were also considered for each of these levels of granularity.

4.2 Experiment results

In Table 1 the predicted end-to-end latencies by the proposed simulable model are compared with the end-to-end latencies measured on the real platform for several scenarios. The considered scenarios allow stressing the performance models to test and highlight three levels of scalability: (1) ANN topologies and level of granularity: several clusterings (from *Cluster1* to *Cluster7*) for the considered ANN topologies are tested to verify the accuracy of the proposed computation time model for coarse and fine grains, (2) amount of communications: the communication time model is tested and stressed when running multiple tiles simulations, as concurrent accesses to shared memories occur, (3) mapping and number of processing elements: several mappings are tested for each considered grains, with various number of processing elements. Mappings using up to 7 tiles and relying both on parallel and pipeline execution are tested.

For each considered partitionings, a 1-tile scenario is tested, in which all actors are implemented on only 1 tile. For the *Cluster2*, *Cluster3*, *Cluster4*, *Cluster6* and *Cluster7* mapped respectively on 2, 3, 4, 6 and 7 tiles, the actors issued from the partitioning of layers are mapped in order to enable a parallel execution of the application. For the *Cluster1* on 2/3 tiles scenario, each actor of the graph is mapped on a separate tile, to enable a streaming execution of the application. The other scenarios enable both a parallel and streaming execution of the application. E.g. in the *Cluster3* on 7 tiles scenario for the *MNIST 784-10-10* topology, each actor is mapped on a separate tile, to enable a parallel and streaming execution of the application. In the same scenario for the *MNIST 784-32-16-10*, the *HiddenLayer2* clusters actors and *OutputLayer* clusters actors were mapped to a same tile due to the target platform only having 7 tiles available. The column

Table 1: Comparison of the measured and predicted end-to-end latency for different partitionings and mappings of fully-connected ANNs.

Experiment				End-to-end latency in thousands of processor cycles					
ANN topology	**Grain**	**Nb. of actors / comm. channels**	**Tiles used**	**Measured**	**Comp. time model only**		**Simulation model**		**Comm. %**
MNIST 784 -10 -10	**Cluster1**	**2 / 3**	**1**	393	376	−4.35 %	394	+0.22 %	5 %
			2	387	370	−4.32 %	387	+0.21 %	52 %
	Cluster3	**7 / 16**	**1**	430	376	−12.4 %	431	+0.31 %	13 %
			3	170	151	−11.6 %	170	−0.01 %	26 %
			7	191	148	−22.5 %	187	−2.28 %	89 %
	Cluster7	**15 / 64**	**1**	502	377	−25.0 %	506	+0.63 %	26 %
			7	120	75	−37.4 %	121	+0.75 %	64 %
MNIST 784 -32 -16 -10	**Cluster1**	**3 / 4**	**1**	1238	1219	−1.51 %	1239	+0.07 %	2 %
			3	1201	1184	−1.51 %	1203	+0.18 %	67 %
	Cluster3	**9 / 25**	**1**	1281	1220	−4.83 %	1279	−0.15 %	5 %
			3	443	421	−5.14 %	442	−0.24 %	8 %
			7	443	407	−8.03 %	440	−0.57 %	70 %
	Cluster7	**21 / 113**	**1**	1368	1220	−10.8 %	1363	−0.33 %	11 %
			7	241	192	−20.4 %	241	+0.01 %	28 %
GTSRB 576 -30 -30 -43	**Cluster2**	**6 / 13**	**1**	967	930	−3.79 %	964	+0.32 %	3 %
			2	486	466	−4.04 %	483	+0.49 %	4 %
			7	433	408	−5.80 %	437	−0.92 %	57 %
	Cluster4	**12 / 41**	**1**	1002	931	−7.12 %	996	+0.55 %	7 %
			4	301	279	−7.12 %	302	−0.42 %	16 %
	Cluster6	**18 / 85**	**1**	1039	931	−10.4 %	1030	+0.80 %	10 %
			6	193	156	−18.8 %	193	−0.39 %	20 %

Comm. % shows the average predicted amount of time spent in polling and executing memory accesses on the shared memory for every tile according to the simulable model.

The predicted end-to-end latencies with associated prediction error are presented for the analytical computation time model used alone (*Comp. time model only* column). The predictions of the computation time model only are too optimistic and bear 10.8 % error on average for the presented scenarios. The prediction error of the computation time model rises with the amount of communications in the considered scenario. When considering multiple tiles execution and SDF graphs of fine grains with numerous communication channels such as *Cluster7*, the computation time model error goes up to 37.4 %. In order to predict the execution time of ANNs on multi-core platforms, a computation time model alone is not sufficient and the modeling of communications is necessary.

The simulable model bears an average accuracy of 99.5 % for all the considered scenarios. The highest prediction error is reached for the *MNIST 784-10-10*, with granularity level *Cluster3* on 7 tiles scenarios: 2.89 %. In this scenarios the average time spent in communication per tile over the overall execution time of the application is also the highest: 89 %. For all the other scenarios, the error of the simulable model is less than 1 %.

The high accuracy of our hybrid modeling flow is rendered possible by the following work hypothesis: (1) The strict separation of the computation and communication using SDF and the model of architecture, which enables the building of separated computation and communication time models. (2) The negligible effect of data dependant paths on latency when executing MLPs on the tar-

geted platform. We have validated this hypothesis by performing measurements on the implementation platform while providing different input images to the ANNs, and observed that the impact of data dependency on the execution time is marginal. This is rendered possible by the type of ANN considered and by the use of FPUs and hardware multipliers on tiles, which normalize the execution of multiplication.

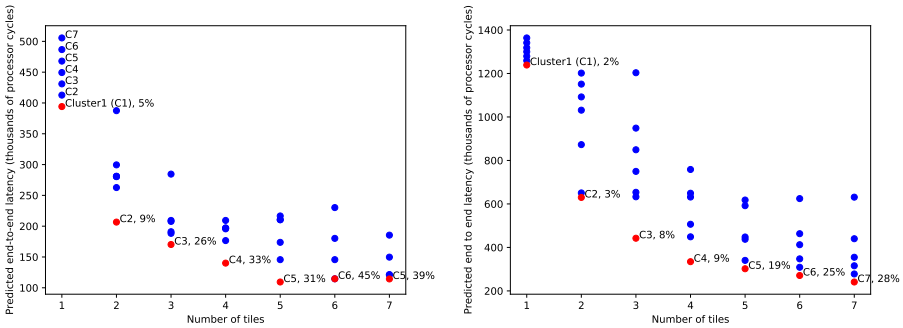
In our experiments, we only tested our modeling flow for ANNs using the ReLU activation function and *float32* precision. Extending the proposed modeling flow to other precisions (e.g. *float16* and fixed point precisions) and other activation functions would simply require a re-calibration of the proposed models. The current time measurement infrastructure used to calibrate and validate our models is built for architectures based on the AXI communication protocol. The effort to port our modeling flow to architectures featuring this protocol is thus minimal, while the effort to port it to other architectures is more significant.

4.3 Exploration of partitionings and mappings

The validated model of performance can be used to explore partitionings and mappings of fully-connected ANNs on multi-core platforms. This exploration allows evaluating deployments of ANNs on the considered platforms with no further prototyping effort on real platform. The results of the exploration of several partitionings and mappings for the two ANNs trained on MNIST are presented in Fig. 3. The displayed graphs give the predicted end-to-end latency of the application based on the number of tiles used for several levels of granularity and mappings. For each number of tiles the solution that optimize timing is highlighted in red on the graph, and the associated level of granularity (Cluster) as well as the average communication rate per tile are displayed. The other scenarios are depicted in blue.

The first graph (Fig. 3a) displays the exploration of end-to-end latency for the *MNIST 784-10-10* application. The end-to-end latency decreases when the number of tiles used increases until 5 tiles, as the parallel execution of the actors of the application allows to accelerate its execution time. The scenario that optimize the end-to-end latency for this topology is the *Cluster5* (C5) SDF graph executed on 5 tiles with a latency of 109 thousands of processor cycles. For configurations relying on more than 5 tiles, an increase of the overall end-to-end latency with the number of tiles used is observed, due to the raise of the communication time. For the 7 tiles execution, the scenario that optimize the timing is the *Cluster5* SDF graph with a latency of 114 thousands of processor cycles. The *Cluster7* execution on 7 tiles is 6 % longer with a latency of 121 thousands of cycles. The average time spent in communications per tile is 89 % for the *Cluster7* scenario (as displayed in Table 1) and 39 % for the *Cluster5* scenario.

The second graph (Fig. 3b) displays the exploration of end-to-end latency for the *MNIST 784-32-16-10* application. In this case the overall end-to-end latency decreases with the number of tiles used up to 7 tiles. The computations account for a bigger part of the overall execution time. Therefore unlike the *MNIST 784-10-10*, the time spent in communications do not produce a significant overhead



(a) MNIST 784-10-10 (b) MNIST 784-32-16-10

Fig. 3: Evolution of the end-to-end latency in thousands of processor cycles based on the number of tiles used for several partitionings and mappings of the considered applications.

when using a platform containing up to 7 tiles. The optimal scenario is the *Cluster7* (C7) executed on 7 tiles with a predicted latency of 241 thousands of processor cycles and average communication rate per tile of 28 %.

5 Conclusion

In this paper, we propose a hybrid performance prediction approach for fully-connected ANNs on multi-core platforms. This approach is based on SystemC simulation, which integrates an analytical computation time model and a communication time model calibrated through measurement to predict execution time. The proposed workflow achieves overall 99.5 % accuracy for estimating the end-to-end latency of three fully-connected ANNs, with the highest prediction error on tested scenarios of 2.28 %. This high accuracy is made possible by the separation of computation and communication using the SDF MoC and by the negligible effect of data dependant paths on the execution time. In future work we will expand our modeling flow to other ANN types such as convolutional neural networks, and we will consider adding power prediction to our modeling flow to enable the exploration of candidate solutions under both timing and power constraints.

References

1. Banbury, C., Zhou, C., Fedorov, I., Matas, R., Thakker, U., Gope, D., Janapa Reddi, V., Mattina, M., Whatmough, P.: Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. In: Proceedings of Machine Learning and Systems (2021)

2. Chen, Y.H., Emer, J., Sze, V.: Using dataflow to optimize energy efficiency of deep neural network accelerators. IEEE Micro (2017)
3. Galanis, I., Anagnostopoulos, I., Nguyen, C., Bares, G., Burkard, D.: Inference and energy efficient design of deep neural networks for embedded devices. IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (2020)
4. Garbay, T., Dobiás, P., Dron, W., Lusich, P., Khalis, I., Pinna, A., Hachicha, K., Granado, B.: Cnn inference costs estimation on microcontrollers: the est primitive-based model. 2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS) (2021)
5. Hasan, M.: State of iot 2022: Number of connected iot devices growing 18% to 14.4 billion globally (May 2022), https://iot-analytics.com/number-connected-iot-devices/, accessed: 07.06.2022
6. Heim, L., Biri, A., Qu, Z., Thiele, L.: Measuring what really matters: Optimizing neural networks for tinyml (2021), https://arxiv.org/abs/2104.10645
7. Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., Igel, C.: Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In: International Joint Conference on Neural Networks (2013)
8. Ke, L., He, X., Zhang, X.: Nnest: Early-stage design space exploration tool for neural network inference accelerators. In: Proceedings of the International Symposium on Low Power Electronics and Design (2018)
9. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998)
10. Lee, E., Messerschmitt, D.: Synchronous data flow. Proceedings of the IEEE (1987)
11. Luenemann, D., Fakih, M., Gruettner, K.: 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 (2020)
12. Nissen, S.: Implementation of a fast artificial neural network library (fann) (2003), https://github.com/libfann/fann
13. Parashar, A., Raina, P., Shao, Y.S., Chen, Y.H., Ying, V.A., Mukkara, A., Venkatesan, R., Khailany, B., Keckler, S.W., Emer, J.: Timeloop: A systematic approach to dnn accelerator evaluation. In: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) (2019)
14. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review (1958)
15. Rumelhart, D., Hinton, G., Williams, R.: Learning representations by back-propagating errors. Nature 323 (1986)
16. Sombatsiri, S., Yu, J., Hashimoto, M., Takeuchi, Y.: A design space exploration method of soc architecture for cnn-based ai platform. Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI) (2019)
17. Stemmer, R., Vu, H.D., Le Nours, S., Grüttner, K., Pillement, S., Nebel, W.: A measurement-based message-level timing prediction approach for data-dependent sdfgs on tile-based heterogeneous mpsocs. Applied Sciences (2021)
18. Tsimpourlas, F., Papadopoulos, L., Bartsokas, A., Soudris, D.: A design space exploration framework for convolutional neural networks implemented on edge devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2018)
19. Vu, H.D., Nours, S.L., Pillement, S., Stemmer, R., Grüttner, K.: A fast yet accurate message-level communication bus model for timing prediction of sdfgs on mpsoc. In: 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC) (2021)