Early Performance and Energy Prediction of Neural Networks Deployed on Multi-Core Platforms

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

[1]German Aerospace Center - Institute System Engineering for future mobility (DLR SE), Oldenburg, Germany

[2]Nantes Université - Institut d'Electronique et des Technologies du numéRique (IETR) UMR CNRS 6164, France

Abstract – Early evaluation of Neural Networks (NN) deployments on edge multi-core platforms with memory limitations is necessary to find deployments that optimize resource usage, performance and energy. In this paper, we propose a timing and power modeling methodology which combines simulation, analytical models, and measurements to offer fast yet accurate performance and energy prediction of NNs on multi-core platforms. The proposed approach is validated against measurements obtained from a real implementation of 27 mappings of four NNs with high accuracy and a fast evaluation time of approximately 20 s per mapping.

1 Introduction

With the important growth of the Internet-of-Things field comes the need for the deployment of Artificial Intelligence (AI) algorithms such as Neural Networks (NNs) at the edge, where their execution close to the sensors allows improving response times and saving bandwidth. However deploying NNs on embedded platforms available at the edge is difficult as NNs are computation-intensive and require important amount of resources whereas embedded platforms have limited processing and memory resources and strict energy constraints. To find implementations that optimize performance, energy and resource usage, several evaluation flows for edge NN deployment have been proposed. Many focus on evaluation through systematic implementation and characterization of NNs on a real platform [10, 1, 2]. The effort to obtain evaluation results is however important as numerous mappings must be deployed on the implementation platform and tested which represents a time-consuming effort. The implementation technology is also fixed due to the need of having the real platform in the loop which restrict the possibilities in regards to architectural exploration. Other approaches [4, 8, 7] are based on pure analytical models that demonstrate efficient exploration of hardware accelerators for NNs, but they have limited scalability for multi-core platforms due to the possible influence of shared resources (bus, memory resources). On multi-core platforms one difficulty comes from the accurate prediction of shared resource contention, which has non-negligible impact on timing and power.

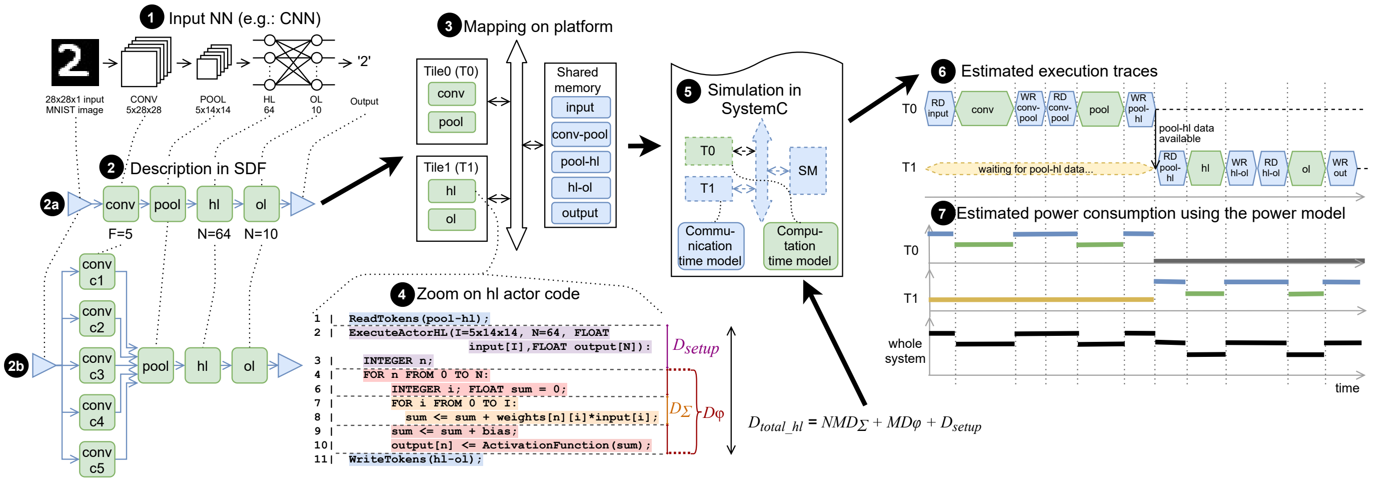
To tackle this challenge, we propose in this paper a timing and power modeling methodology used to efficiently predict performance and energy of NNs deployed on multi-core embedded platforms. This approach relies on a hybrid modeling flow which combines simulation, analytical models and measurements. Simulable models are used to describe shared resources, analytical models are used to offer fast prediction and measurement on real prototypes are used to appropriately calibrate our models. The efficiency of our approach is validated against timing and power measurements from a real implementation of 27 mappings of four NNs, including one Convolutional NN (CNN) and three different Multi-Layer Perceptrons (MLPs), with more than 97 % accuracy on timing and 95 % accuracy on power. In Section 2 we explain our work hypothesis and how our timing and power models are obtained. Section 3 presents and discusses the experimental setup used to perform the calibration of our models and their validation. Conclusions and perspectives are drawn in Section 4.

2 Proposed modeling methodology

An overview of the proposed timing and power modeling methodology is provided in Figure 1. In the scope of this work we only consider MLPs and CNNs ❶. The description of the NN in a dataflow-oriented Model of Computation (MoC) ❷ and its mapping on the platform ❸ are discussed in Section 2.1. When executed on the platform, base delays can be identified and measured to calibrate analytical timing models used to describe NN computation part ❹ as explained in Section 2.2. The computation time analytical models are then integrated in a SystemC executable model ❺ to simulate the influence of shared resources on the overall system execution as discussed in Section 2.3. The execution traces generated from the simulation ❻ are then used to estimate the power consumption ❼. The power model is presented in Section 2.4.

2.1 Work hypothesis

The first step in the proposed methodology aims at describing the NN in a dataflow-oriented Model of Computation (MoC) in order to have a separation of computation and communication, which ease the analysis process. In this work we rely on the Synchronous DataFlow (SDF) [6] MoC ❷. The operations executed in NN layers (e.g. the neurons from dense layers) are represented as actors depicted in green in Figure 1. Communication channels, which are depicted in blue, corresponds to bounded FIFO buffers to store the data exchanged between actors. SDF offers the possibility to represent the application with several levels of granularity. In the coarsest level of granularity, each layer of the NN is described as one actor. Finer levels of granularity are possible, in which layers are split into several actors with equitable workload. They



```
ReadTokens(pool-hl);
ExecuteActorHL(I=5x14x14, N=64, FLOAT
                     input[I],FLOAT output[N]):
  INTEGER n;
  FOR n FROM 0 TO N:
      INTEGER i; FLOAT sum = 0;
      FOR i FROM 0 TO I:
        sum <= sum + weights[n][i]*input[i];
      sum <= sum + bias;
      output[n] <= ActivationFunction(sum);
WriteTokens(hl-ol);
```

Figure 1 – Overview of the proposed modeling flow for timing and power prediction of NNs on multi-core platform

allow exploiting more computation parallelism from the NN at the cost of a higher number of communication channels. An example of a CNN described in two SDF graphs of different levels of granularity is provided in Figure 1. The SDF graph 2a corresponds to the highest level of granularity possible, while in the graph 2b the convolution layer is split into 5 actors of equitable workload. Once the NN described in SDF, it must be mapped onto the platform 3. In our work the platform is composed of a set of tiles, each composed of a single core and a private memory for instructions and data. A shared memory is accessible through a communication infrastructure. Actors are mapped on tiles and communication channels are mapped on the shared memory. The separation of communication and computation offered by SDF is respected by the implementation platform. This allows building models with separate communication and computation time.

2.2 Timing model

MLPs are composed entirely of dense layers, whereas CNNs are composed of three types of layers: convolution, pooling and dense layers. These three layer types rely on the computation of the same operations a multitude of times over different inputs. It is therefore possible to propose an analytical computation time model for each of these layer types. For example on Figure 1 4 a zoom on the code performed in the actor 'hl' is provided which corresponds to a dense layer. Three delays can be identified: D_Σ for the multiply accumulate operations inside neurons, D_φ for the activation function of the neuron and D_{setup} for calling the function and setting up variables. D_{setup} occurs only once when calling the function corresponding to the actor computation. D_φ is repeated a total of N times, with N being the number of neurons inside the actor. D_Σ is repeated a total of $N \cdot I$ with I being the number of inputs of the dense layer. The resulting analytical model is presented in Equation 1. All the data (neuron's weights, inputs) and instructions are available in the local memory of tiles. The model is thus scalable to any actor issued from the partitioning of a dense layer regardless of the number of neurons it contains.

$$D_{dense}(N, I) = N \cdot I \cdot D_\Sigma + N \cdot D_\varphi + D_{setup} \quad (1)$$

Using the same approach, we propose an analytical computation time model for convolution and pooling layers. The delay needed to compute an actor issued from a convolution layer as shown in Equation 2 depends on the number of convolution filters F, the number of inputs I and the filters' size S, as well as the base delays D_{setup}, D_φ and D_*, with D_* being the delay needed to perform the convolution operation. The delay needed to compute the pooling layer as shown in Equation 3 depends on the number of filters F and the number of inputs I. In our approach base delays identified in the proposed models (e.g. D_Σ) are calibrated through measurement. Another possibility to alleviate the calibration effort is to provide an estimation for a given processor, using information from the chip provider for example. Once calibrated, the models can be used to predict the computation time of NN mappings without further re-calibration.

$$D_{conv}(F, I, S) = F \cdot I \cdot S \cdot D_* + F \cdot I \cdot D_\varphi + D_{setup} \quad (2)$$

$$D_{pool}(F, I) = F \cdot I \cdot D_{max} + D_{setup} \quad (3)$$

2.3 Simulation in SystemC

In order to predict the performance of NNs on multi-core platforms, both the computation time model presented in Section 2.2 and the message level communication time model presented in [11] are used inside a simulation described in SystemC as shown on Figure 1 5. 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 especially in the case of contentions at shared resources. The simulation allows knowing the state of cores and shared resources at any time of the estimated execution, and obtaining the estimated execution traces of the NN mapped on the platform. Due to the separation of computation and communication offered by SDF and respected by the platform and the use of separate models for computation and communication delays, the SystemC model is composable: the number of tiles does not modify the nature of the model.

2.4 Power model

The power model is used with the simulated execution traces from SystemC to estimate the total power consumption as shown in Figure 1 ❻ and ❼. During the execution of NNs, tiles execute computation or communication activities. The dynamic power consumption of the platform executing NNs can thus be described using two terms: $P^{\Sigma}_{comp}(t)$, the total power consumption of tiles in computation mode at time t and $P^{\Sigma}_{comm}(t)$, the total power consumption in communication mode at time t. To obtain the total power consumption of the system $P^{\Sigma}(t)$, the static power consumption of the circuit P_{static} must also be added. When tiles are in computation mode, they are executing the computations inside the NN (e.g. the neurons from dense layer) and are therefore independent from one another as all necessary data is contained in the private memory of the tile. When tiles are in the communication activity, they either: 1. perform a read or write in the shared memory, which is modeled by $P^{\Sigma}_{RW}(t)$, 2. wait for the availability of data, which is modeled by $P^{\Sigma}_{Wait}(t)$.

The calibration of the power model is performed by characterizing through measurements the static power consumption of the platform when no activity is executed, and the power consumption of each individual tile in the identified activities (computation, read and write access on the shared memory and waiting). A multi-linear regression is then performed on the measurements to extract the power model. Once the characterization phase done, the power model can be used for any NN mapping without further re-calibration.

3 Experimental results and discussions

3.1 Platform prototype and model calibration

In order to evaluate the prediction accuracy of the proposed modeling flow, we implemented a prototype of multi-core platform on the programmable logic section of a Xilinx UltraScale MPSoC+ FPGA (ZCU102 board). The processing core of the tiles is a MicroBlaze. The private memory of tiles and the shared memory are implemented as BRAM, which are internal to the FPGA SoC. The communication medium to access the shared memory is implemented as a shared AXI interconnect. The implementation platform is composed of 7 tiles. The timing measurement infrastructure is presented in [9]. The power measurements are obtained by probing the supply voltage of the programmable logic part of the FPGA using the R&S HMC8012 Digital Multimeter.

The computation and communication time models are calibrated by measuring directly the base delays (e.g. D_{Σ} as introduced in Section 2.2). When performing the calibration, we observed that the variability of execution time based on input data was marginal. This allows fixing as constant the base delays identified in the analytical models. The power model is characterized through measurement as presented in Section 2.4. Once they are calibrated, all models can be used to evaluate any NN deployment on multi-core platforms without further re-calibration.

3.2 Results and discussions

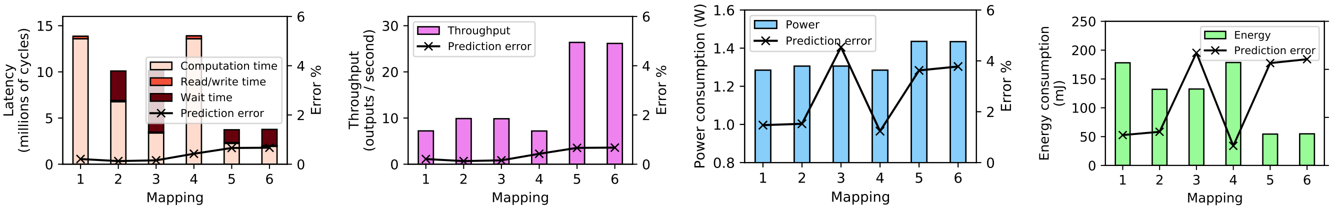
To validate our timing and power modeling flow, we trained and tested one CNN and three MLPs. These different use-cases offer a variety of complexity as well as different communication workloads which leads to a comprehensive validation of our models. The *CNN*, *MLP1* and *MLP2* were trained on the MNIST [5] dataset whereas the *MLP3* was trained on the GTSRB dataset [3]. The *CNN* is composed of one convolution layer with 5 filters, one pooling layer and two dense layers. *MLP1* is composed of three dense layers while *MLP2* and *MLP1* are composed of four. All NNs were trained using lightweight open source C libraries available online [1] and use the ReLU activation function.

Table 1 – Observed average and maximum error on tested mappings regarding four different metrics: the latency in processor cycles (T) , the throughput in outputs/s (Φ), the power consumption in W (P) and the energy consumption in J (E). The column titled "# tested mappings" provides the total number of different mappings tested for each application.

Application	**# tested mappings**	**Metric**	**Error %**	
			Average	**Maximum**
MLP1 MNIST	**7**	T	0.71	2.85
		Φ	0.72	2.94
		P	1.14	2.22
		E	2.29	3.43
MLP2 MNIST	**7**	T	0.23	0.60
		Φ	0.23	0.61
		P	1.72	2.28
		E	1.69	2.44
MLP3 GTSRB	**7**	T	0.54	0.99
		Φ	0.54	0.98
		P	1.50	2.43
		E	1.07	1.47
CNN MNIST	**6**	T	0.37	0.67
		Φ	0.38	0.67
		P	2.70	4.54
		E	2.81	4.69
All applications	**27**	T	0.47	2.85
		Φ	0.47	2.94
		P	1.73	4.54
		E	1.93	4.69

Table 1 provides a summary of the observed average and maximum errors on all tested mappings of the four NNs. The latency T and the power P are predicted using our modeling flow. To obtain the throughput Φ we compute how many outputs per second the mapping offer based on the predicted latency. To obtain the energy consumption E, we integrate the power consumption over the latency. The highest error $2.85\,\%$ on latency and $2.94\,\%$ on throughput is observed for *MLP1* on a 7-core mapping with a communication workload of more than $70\,\%$. This NN has the simplest topology compared to the others and thus the lowest amount of computations. The low computation rate and high communication rate of this mapping leads to this error, which remain acceptable. The highest error $4.54\,\%$ on power and $4.69\,\%$ on energy is observed for one mapping of the CNN. The CNN is composed of convolution, pooling and dense layers, which leads to a slightly higher prediction errors on power and energy due to the variety of layers. This error is acceptable for a high level modeling flow. It can be noted that the error on timing is overall smaller

[1] MLPs: `https://github.com/libfann/fann`
CNN: `https://github.com/tranleanh/CNN-cpp`



(a) Latency (b) Throughput (c) Power consumption (d) Energy

Figure 2 – Predicted latency in millions of processor cycles, throughput in outputs per second, power in mW and energy in mJ - and error in % for every considered mappings of the CNN application. Mapping IDs in X-axis are defined in Table 2.

Table 2 – Tested mappings of the CNN application.

ID	Mapping of actor on tile $T_\#$							
	Conv					Pool	Dense1	Dense2
1	T_0					T_0	T_0	T_0
2	T_0					T_1	T_0	T_0
3	T_0					T_1	T_2	T_3
4	T_0	T_0	T_0	T_0	T_0	T_0	T_0	T_0
5	T_1	T_2	T_3	T_4	T_5	T_0	T_0	T_0
6	T_1	T_2	T_3	T_4	T_5	T_6	T_0	T_0

than the error on power and energy. This is due to the timing measurement infrastructure being cycle accurate which leads to more accurate calibration of the timing model as well as a more reliable use for the validation of the models. The power measurements are performed with a multimeter with a higher measurement error, which leads to a less accurate calibration and validation of the model. The error on the energy is also higher due to the propagation of both the error on timing and power when computing it. One of the main advantage of the proposed modeling flow is the evaluation speed. The evaluation of a given mapping takes only 20 s, which is two times faster than our fully automatized timing and power measurement infrastructure. A more important effort is however required if the models need to be re-calibrated (e.g. change in the platform, or consideration of other NN layer types).

We also provide a focus of the validation of the models regarding the CNN application, as presented in Figure 2. The tested mappings shown in the X-axis are given in Table 2. Regarding the latency and throughput, the model has very high accuracy as the error lies below 0.67 %. Regarding the power consumption, it can be noted that the error lies on average around 2.70 %, with important variations which are due to the accuracy limitations of the power measurement infrastructure. When comparing the plots of the power consumption and energy consumption, it can be noted that the error on power is directly propagated on the energy, as the plots of error follows the same shapes. The overall observed increase in error of the energy consumption comes from the propagation also of the error on latency, used to compute it.

4 Conclusion

This paper presents a early performance and energy prediction methodology for NNs on multi-core platforms which uses simulation, analytical models and measurements. This methodology offers highly accurate predictions with a fast evaluation time and modularity in regards to the number of cores used in the platform and the communication workload. In future work we will extend this modeling flow to architectures including external memory and caches. More information on the project can be found on our GitLab repository [2].

Acknowledgements

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

References

[1] I. Galanis, I. Anagnostopoulos, C. Nguyen, G. Bares, and D. Burkard. Inference and Energy Efficient Design of Deep Neural Networks for Embedded Devices. *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020.

[2] L. Heim, A. Biri, Z. Qu, and L. Thiele. Measuring what Really Matters: Optimizing Neural Networks for TinyML. *arXiv preprint, arXiv:2104.10645*, 2021.

[3] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel. Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, 2013.

[4] L. Ke, X. He, and X. Zhang. NNest: Early-Stage Design Space Exploration Tool for Neural Network Inference Accelerators. In *International Symposium on Low Power Electronics and Design*. Association for Computing Machinery, 2018.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 1998.

[6] E.A. Lee and D.G. Messerschmitt. Synchronous Data Flow. *Proceedings of the IEEE*, 1987.

[7] 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)*, 2016.

[8] A. Parashar, P. Raina, Y. S. Shao, Y. H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019.

[9] R. Stemmer, H.-D. Vu, S. Le Nours, K. Grüttner, S. Pillement, and W. Nebel. A Measurement-Based Message-Level Timing Prediction Approach for Data-Dependent SDFGs on Tile-Based Heterogeneous MPSoCs. *Applied Sciences*, 2021.

[10] F. Tsimpourlas, L. Papadopoulos, A. Bartsokas, and D. 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*, 2018.

[11] H.-D. Vu, S. Le Nours, S. Pillement, R. Stemmer, and K. Grüttner. A Fast Yet Accurate Message-level Communication Bus Model for Timing Prediction of SDFGs on MPSoC. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021.

[2] https://gitlab.univ-nantes.fr/lenours-s/pssim4ai