XLIII
CILAMCE
21-25, November, 2022
FOZ DO IGUAÇU - PR - BRAZIL

Convolutional Neural Networks Implementation on a Network-on-Chip Platform

Alexandre N. Cardoso[1], Luiza de Macedo Mourelle [1,2], Nadia Nedjah[1,3]

[1]*Electronics Engineering Postgraduate Programme*
[2]*Department of Systems Engineering and Computation*
[3]*Department of Electronics Engineering and Telecommunications*
State University of Rio de Janeiro
524 São Francisco Xavier St., 20550-013, Rio de Janeiro, RJ, Brazil
cardoso.alexandre@posgraduacao.uerj.br, ldmm@eng.uerj.br, nadia@eng.uerj.br

Abstract. Delivering more throughput to a given computational system or device may be crucial when using computational intelligence-based approaches as a design preference for a growing set of applications. On the other hand, these approaches are frequently under rigorous constraints regarding processing time, power consumption and required memory. One of the main topics of interest in computational intelligence is machine learning. It deals with computational methods and models based on observational data. In machine learning, the machine develops the ability to continually learn with data, in an attempt to predict and recognize patterns as humans do. Deep neural networks use several hidden layers to achieve pattern recognition. The main difference between traditional neural networks and deep neural networks is the amount of network layers. A convolutional neural network is a deep learning model, usually used to classify and recognize patterns in image and video-based applications. One of the most known designs for convolutional neural network is LeNet-5. It allows manuscript characters recognition. This kind of neural network consists of an input layer, that receives the image, a series of layers, that implement image operations for characteristics mapping, and a last layer, that consists of a classification neural network, using the characteristics map and provides the classification result as output. The network structure consists of a series of paired convolutional layers followed by pooling layers. The output is classified by a fully connected layer. A convolutional layer is used to allow image characteristics mapping. A pooling layer is responsible for reducing the matrixes dimensionality and data complexity. Our work aims at investigating the use of parallel processing for the implementation of a convolutional neural network on a multiprocessor system-on-chip. It exploits a network-on-chip platform for communication between the processing elements. Mainly, our work consists of grouping the network operations into conceptual units called tasks. These tasks are the workload to be distributed between the processing units, which will operate in a parallel manner. As a case study, we implement LeNet-5 on the multiprocessor system-on-chip MEMPHIS platform. We demonstrate that the distribution of the convolutional neural network workload over a set of processing elements leads to significant performance gain over the serial implementation.

Keywords: Network-on-Chip, Convolutional Neural Networks, Parallel Processing

1 Introduction

Delivering more throughput to a given computational system or device may be crucial in a moment when the use of Artificial Intelligence (AI) approaches becomes a design preference for a growing set of applications. On the other hand, these cases are frequently under rigorous constraints on processing time, power consumption and available memory. Nevertheless, physical barriers may impose themselves since Moore's Law met its frontiers because of material limitations. In this context, hardware accelerators are an important field of investigation, due to its relevance as enablers for real, effective AI applications.

However, processing an AI application based on techniques like Convolutional Neural Networks (CNN) may lead to dealing with a massive set of linear operations to extract features from some input data (an image, for instance) to infer its content, based on a previously trained machine learning model. On the other hand, these operations may be independent to a certain extent and, as a result, it is feasible grouping them in oder to explore the use of parallel processing approaches. Traditionally, CNNs have been executed in CPUs and GPUs.

However, the low throughput and low energy efficiency that these platforms offer present a bottleneck to is use. While Application Specific Integrated Circuits (ASICs) can offer customized implementation, it implies a complex design, with high initial investment and lack of reconfigurability, which is crucial, concerning the rapid advances in CNNs architecture.

Due to various attractive characteristics, Field Programmable Gata Arrays (FPGAs) offer a promissing platform for accelerating CNNs in hardware. Usually, FPGAs are more energy efficient than CPUs and GPUs, and offer better performance than CPUs. Besides this and differently from ASICs, the capability for reconfiguration offered by FPGAs allows for rapid exploitation of the vast space of design configuration parameters of CNNs.

Another important aspect to consider about hardware as a powerful enabler for machine learning tasks is that the state-of-art in Solid-State Physics and Computer Architecture fields allows building fully featured Multi-Processing System-on-Chip (MPSoC). An MPSoC may include several, independent Processing Elements (PEs), each one a complete computer itself, since they will have a Central Processing Unit (CPU), local memory, control logic and communication interfaces. Actually, communication between logic blocks within the system is critical. The classic bus-based approach may represent an important constraint source, due to its geometry and electrical characteristics. So, it may turn the interconnection of numerous elements within an MPSoC a very difficult task, compromising scalability.

Therefore, active approaches, like Network-on-Chip (NoC), may represent an interesting way to interconnect MPSoC inner components and provide scalability. Identifying the logical blocks that constitute the components of a given application, understanding how the data flows through these components and how the hardware layer can be used in an optimized way to execute the software will produce the necessary insights on how to allocate tasks to these components and to map them within the PEs on the MPSoC in order to obtain speed-up through parallelism.

In most scenarios, NoC can provide flexibility and efficiency to the project, since it may simplify the system inner blocks interconnection. Bringing a packet relaying approach to the system may turn problems as communication overhead, media dispute and energy spending more manageable.

Our work consists of implementing a CNN in an MPSoC simulation platform, using NoC as communication topology between PEs. We aim to investigate solutions for increasing performance through parallelism.

In Section 2, we present some related work, mainly based on the idea of implementing CNN in hardware. In Section 3, we introduce some aspects of the LeNET-5 architecture, which is the case study of our work. In Section 4, we present some basic characteristics of the MEMPHIS platform and its NoC topology, used as the framework of our experiment. In Section 5, we describe the approach used to implement LeNET-5 on MEMPHIS, in order to exploit parallelism. In Section 6, we drawn some conclusions and future works.

2 Related Work

In [1], we find a discussion about an architecture aimed at the acceleration of some processing elements of a CNN. It is based on spacial array of processing elements, composed of registers, arithmetic unit and some control logic. The PEs are interconnected by a NoC, in order to reduce the amount of transactions between PEs and a common buffer or even to an external memory. The eventual increase in performance would be from the reduction of data search operations related to the memory hierarchy.

In [2] there is a similar discussion, where the use of reconfigurable hardware technologies, such as Field-Programmable Gate Arrays (FPGA), would be preferable if compared to Application-Specific Integrated Circuits (ASIC) since we could recycle the hardware in production. However, even on homogeneous MPSoCs, there are chances that we could reach good performance levels through careful project decisions in order to obtain advantage from parallelism. Nevertheless, specialized hardware may be an interesting solution when the computational resources are scarce, as in embedded devices, for instance.

According to [3], the use of specialized hardware on machine learning applications is a key subject. It eventually enables strategies aiming at optimizing computational resources, avoiding the heavy use of the system's memory hierarchy and other costly techniques. In fact, in [4], we can observe, since early 2000, the growing interest in the implementation, through specialized hardware, of dedicated processing strutctures, in special the convolutional layer of a CNN.

3 The LeNET-5 Architecture

The LeNET-5 CNN architecture was proposed in [5] as a method aiming at handwritten digit's recognition, primarily. Its structure is shown in Figure 1 . The input is a 32x32 greyscale pixels. It consists of seven layers, two of them implementing sliding kernels, each one followed by a subsampling layer.

C1 is a convolutional layer that produces 6 features maps. There will be six 5x5 kernels that will produce

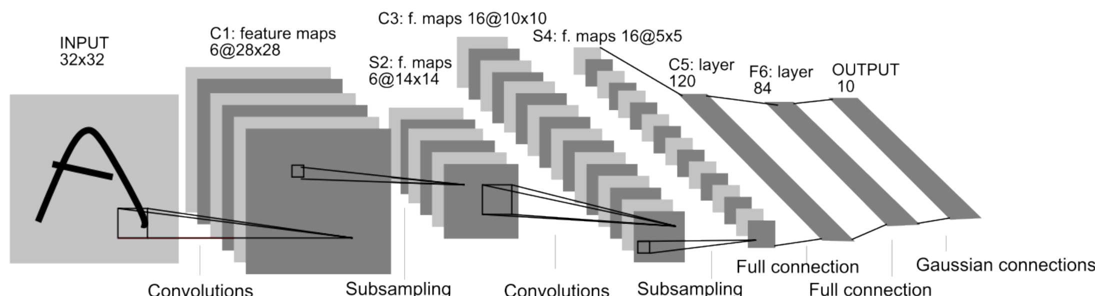

Figure 1. The LeNET-5 reference architecture [5]

28x28 units feature maps.

S2 is a sub-sampling layer with 6 feature maps which elements will be connected to a 2x2 units, non-overlapping, neighbourhood on C1 feature maps, producing 14x14 units feature maps as a result.

C3 is a convolutional layer with 16 feature maps. The kernel's face in C3 would be 5x5 units. The originally described architecture in [5] suggests that these feature maps shall be connected to a subset of the S2's ones with the purpose of reducing the number of connections and forcing some asymmetry that could induce C3 feature maps to capture different, maybe complementary features. The proposed interconnection is shown in Table 1.

Table 1. The connections map between S2 and C3, as proposed in [5]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	X				X	X	X			X	X	X	X		X	X	X
1	X	X				X	X	X			X	X	X	X		X	X
2	X	X	X				X	X	X			X		X	X	X	X
3		X	X	X				X	X	X			X		X	X	X
4			X	X	X			X	X	X	X		X	X		X	X
5				X	X	X			X	X	X	X		X	X	X	X

S4 is a subsampling layer with 16 feature maps of size 5x5 units. Each pixel is connected to a neighbourhood of 2x2 units. It would be similar to the connection between C1 and S2.

C5 is a convolutional layer with 120 feature maps that are connected to a neighbourhood of 5x5 units size to each of the 16 features map of S4. Since the features maps of S4 and C5 have the same size, C5 features maps would be 1x1 units size. Some would say that this configuration implies to understand C5 as a fully-connected layer. It is explained in [5] that it wouldn't be true if the input size of the CNN was made bigger, resulting in a feature map with size greater than 1x1 unit.

F6 is fully-connected to the feature maps of C5 and has 84 units. This number is related to the way the output is calculated as described in [5], where each output would be determined by an Euclidean Radial Basis Function (RBF) measuring the fit of distinct class models and the data provided by F6.

Hyperbolic tangent functions would be the one applied on activations through all the layers. The exception is in the output layer, where RBF-Softmax would be the commonly applied function.

4 The MEMPHIS Platform

According to [6], a NoC is a packet-switching communication network built to interconnect the components of an MPSoC. NoC is an alternative to classic interconnection system, like buses. NoC uses an active approache, since they are based on routing elements positioned as nodes on an interconnection physical structure, under a geometry such as a mesh. These routing elements are associated to a PE on an MPSoC and their function is to route the packages, related to a message produced by a task running on a sender PE, to a consuming task, running on a receiver PE. As a router on a traditional packet-switching computer network, these routing elements are able to buffer and forward packets to their destination following a routing protocol, eventually avoiding defective or jammed paths. Figure 2 represents a hypothetical MPSoC structure, where several PEs are interconnected by a NoC. The different colours, applied to the PEs representation, means that these PEs may be different or

specialized. The orange dots on mesh nodes represent the routing elements, with up to four ports connecting them to their neighbours.

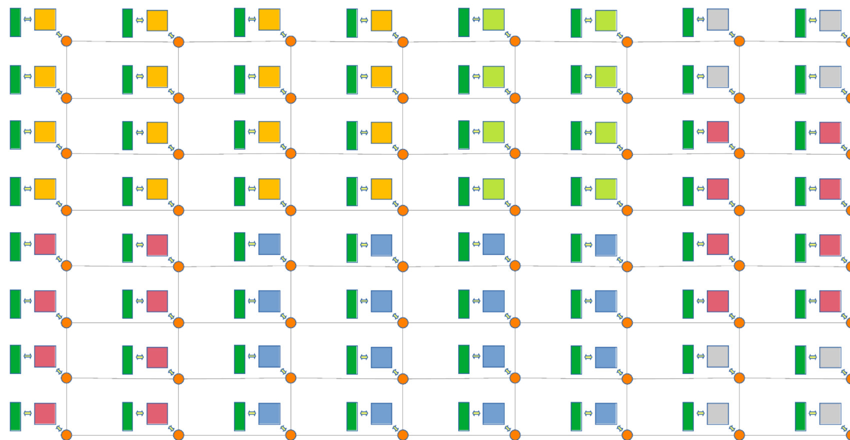

Figure 2. Representation of multiple PEs on a MPSoC, interconnected by a 2D-mesh NoC

MEMPHIS is an acronym for Many-Core Modelling Platform for Heterogeneous MPSoCs. It is an MPSoC simulator that comprises a 2D- mesh architecture NoC. As explained in [7], the platform allows simulating a many-core system, disposing a set of PEs that would include a CPU, some local memory, some control logic for managing Direct Memory Access (DMA) and communications operations and, finally, a routing element. The NoC itself is called HERMES, described in [8], where the routing element, associated to each PE, consists of the active part of the NoC. Figure 3 represents a PE structure as defined in MEMPHIS.

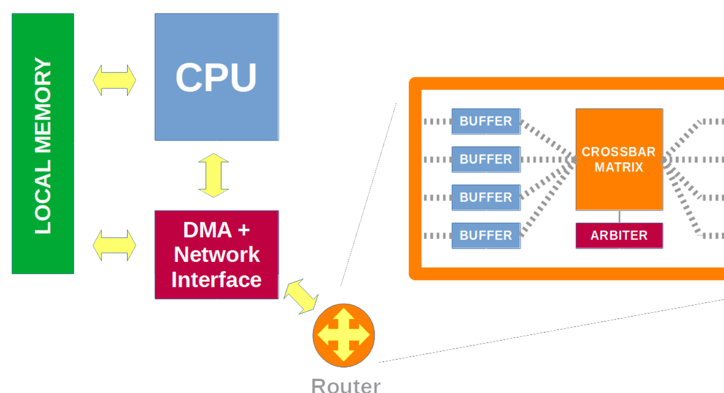

Figure 3. A representation of a PE as defined in MEMPHIS and its router element

Concerning the user's software, the programs that will be mapped to a certain PE, constituting its workload, is called task. A task will be written in C and compiled with a cross-platform compiler, having as objective the specific hardware where the task will be mapped. The default CPU architecture in MEMPHIS is Plasma, described in [9] as an open source, 32-bits CPU that is compliant with MIPS I instruction set. The platform has an API that can be applied to implement communication operations, enabling message exchange between tasks in a simplified way. The tasks will be part of an application, which consists, itself, a closed communication domain: a task will be able to exchange messages only with other tasks that are within the same application, through NoC. Communication clusters can be forcibly implemented too, if restricting the routed traffic in NoC is a need.

Figure 4 shows MEMPHIS GUI debugger. The PE named 0x0 is a Master PE, that manages the MPSoC and cannot be used for running user tasks, itself. The arrows pairs pointing to opposite sides and in orthogonal or diagonal position shows the messages flow within NoC: orthogonal denotes a message routed through the routing element of the PEs and diagonal, the departure or arrival of a message to that particular PE. The PE 0x1 has a single arrow pointing to it; it means that there is a peripheral connected to it for inputting data. In that case, it concerns the App Injector, a mandatory, default peripheral that inserts in the MPSoC the user-defined PE workloads.

Figure 4. The MEMPHIS GUI debugger main window

5 Implementing a LeNET-5 CNN

With the LeNET-5 reference architecture in mind, we divided it into functional blocks in order to group them in such a way that we could build tasks that could reproduce its work with parallelism in mind. Some functionalities have no codependence, so they are obvious targets for parallelization efforts.

Within a convolutional layer, sliding kernels have no data codependence. Thus, since the only data they share is the input from a previous layer or the input image itself, we noticed we could process them in separate tasks. The outputs would be dispatched to a next layer asynchronously and their tasks would reassemble this output locally.

Here, the advantage of using a NoC is obvious, since it simplifies the forwarding data between layers. Producer tasks do their work, group the output data for populating a message payload, prepare the message and dispatch it. In MEMPHIS API, receiving a message from another task is a blocking operation, but sending is not; so even if the receiver is not ready to consume a message from a given producer task, chances are that we can simply start the transaction, buffer the message and keep processing data, producing a latency hiding effect. We noticed that this asynchronous behaviour contributes with the application global speedup.

Subsampling, on the other hand, depends on the data generated by one of the kernels associated to a certain convolutional layer. However, PEs are rare resources and mapping single subsampling tasks on their own PEs increases the cost and the complexity of the system. Another issue would be the need for adding communication transactions to transfer data from a convolutional layer to its subsampling neighbour. The reason this is an issue is the communication overhead: message exchange may have a time cost, so it can only be justified if this time cost is smaller than the time cost of increasing the workload over a PE. So, we decided to associate a kernel with a subsampling operation when possible, making them a channel as a whole. Figure 5 illustrates the conceptual organization of tasks in order to implement individual processing channels.

The fully-connected layers receive a similar approach: since the matrix multiplications can be processed while the input data stream arrives, we can promptly start its computations as soon the first data chunks, from the previous layer, arrives. Again, the NoC buffering and forward mechanism can act as a latency hiding resource. On the other hand, the single Perceptron operations may spend a fraction of the necessary time to process a convolutional kernel plus a subsampling operation. This allows us to consider groups of Perceptrons being processed within a single task. Based on these concepts, we defined that our experiment would be organized in five kinds of tasks, summarized in Table 2.

For reasons concerning our simulation environment, we had to write our tasks using strictly integer data types, creating precision problems that we are trying to surpass in order to achieve better results. Today, we are tied to integer computing methods that must be tuned to preserve the necessary precision on processing a CNN.

Our actual approach on dealing with the precision problems consists of manipulating CNN weights, within the experiments, noted as fractions as much as we can. This idea is inspired by [10] and consists of operating floating point numbers as a ratio between two integer values in order to avoid rounding operations as much as possible. Actually, multiplying fractions is not expensive, computationally. Sums, however, impose the calculation of the

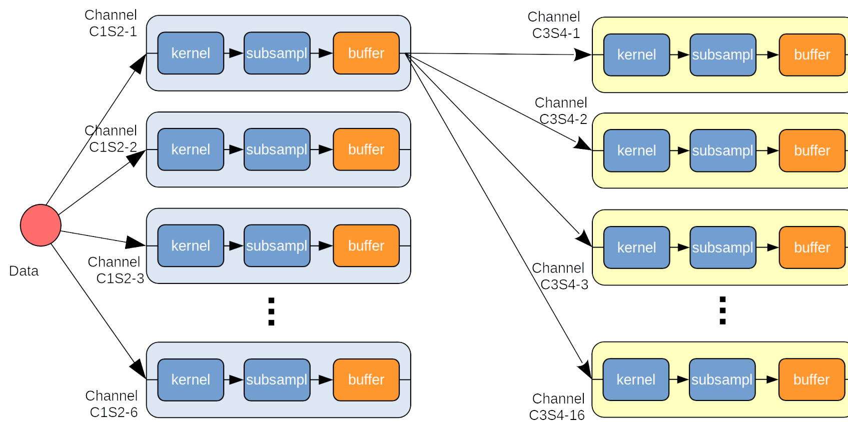

Figure 5. Two convolutional plus subsampling layers, organized as channels

Table 2. Task species, grouping CNN functionalities

Task	Layers scope	Description	Num.Instances
T0	C1 and S2	Runs a C1 kernel and do subsampling	6
T1	C3 and S4	Runs a C3 kernel and do subsampling	16
T2	C5	Executes a C5 kernel	120
T3	F6	Executes 14 F6 Perceptrons	6
T4	OUTPUT	Executes 10 Perceptrons	1

Least Common Multiple (LCM) between denominators and its cost depends on the LCM algorithm efficiency. This approach is useful since it allows preserving accuracy on float-point numbers arithmetic when the support of non-integers values representation is not available. On the actual state of our implementation, we are using weights noted as fractions and a multiplier, a power of 10, in order to partially preserve the mantissa when operating the fraction is needed. The representation limits are a concern too: our simulation environment deals with 32 bits, long integer representations, so all numeric values within the environment is limited to $\pm 2.147 \times 10^9$ if signed, natural numbers, or 4.294×10^9 for unsigned values. These representation limits shall be considered during the weights datasets preparation, since numerators and denominators must be restricted to them. So, a single weight is noted as a tuple (numerator, denominator), such as the following example:

$$0.0028846217 = \frac{5427551}{1881546894} \Longrightarrow (5427551, 1881546894)$$

We built two versions of our LeNET-5 CNN: a "serial" one, planned to execute on a single PE, and a "parallel" one, designed to run several tasks on separate PEs. No special optimization was made on them. The serial version returns a class associated to a given input image in 314.88ms and the parallel one in 141.66ms. So, the parallel version is 2.22 times faster. Since we are still working on the accuracy issues, while implementing a complete set of functionalities to support the arithmetic demanded by linear transformations within a CNN using the fraction notation approach, like efficient LCM calculations, the classification performance keeps poor at the moment.

6 Conclusions

The speed-up of 2.22 achieved by exploring the parallelism provided by MEMPHIS, based on a 2D-mesh NoC, is quite significant. On preliminary experiments, we perceived that a NoC can be an effective tool not only on design simplification, but on latency hiding too. However, the communication overhead shall be considered on design phase, imposing a coherent use of messages exchange and its payloads use.

It is part of our roadmap, aside improving the numeric processes within our experiment, reimplement our "parallel" version of LeNET-5 under different functionalities and task mappings on SoC, in order to investigate its impact on execution time and speed-up against a serial version of it.

Acknowledgements. Bazyl, A. kindly supplied us some advice and pre-trained weight sets we could apply directly

on our experiment. His curriculum can be seen in http://lattes.cnpq.br/7751502543309320 .

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang. Hardware for machine learning: Challenges and opportunities. In *2018 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–8, 2018.
[2] S. Mittal. A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications*, vol. 32, n. 4, pp. 1109–1139, 2020.
[3] W. J. Li, S. J. Ruan, and D. S. Yang. Implementation of energy-efficient fast convolution algorithm for deep convolutional neural networks based on FPGA. *Electronics Letters*, vol. 56, n. 10, pp. 485–488, 2020.
[4] A. Shawahna, S. M. Sait, and A. El-Maleh. Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, vol. 7, pp. 7823–7859, 2019.
[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Ha. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, vol. 86, n. 11, pp. 1–46, 1998.
[6] S. Pasricha and N. Dutt. *On-chip communication architectures: system on chip interconnect*. Morgan Kaufmann, 2010.
[7] M. Ruaro, L. L. Caimi, V. Fochi, and F. G. Moraes. Memphis: a framework for heterogeneous many-core SoCs generation and validation. *Design Automation for Embedded Systems*, vol. 23, n. 3-4, pp. 103–122, 2019.
[8] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost. HERMES: An infrastructure for low area overhead packet-switching networks on chip. *Integration, the VLSI Journal*, vol. 38, n. 1, pp. 69–93, 2004.
[9] S. Rhoads. Plasma - MIPS I compatible Processor, 2001.
[10] D. E. Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998.