Energy-Efficient Machine Learning on FPGA for Edge Devices: a Case Study

Guillaume Devic, Gilles Sassatelli, Abdoulaye Gamatié

Compas'2020 : Parallélisme / Architecture/ Système/ Temps Réel
MILC - Lyon, France, du 30 juin au 3 juillet 2020

Energy-Efficient Machine Learning on FPGA for Edge Devices : a Case Study *

Guillaume Devic, Gilles Sassatelli, Abdoulaye Gamatié

Université de Montpellier / CNRS,
LIRMM - 161 rue Ada, 34095 Montpellier, Cedex 5 - France
prénom.nom@lirmm.fr

Abstract
This paper presents a case study on the combination of a few static code optimization with an FPGA prototype of heterogeneous multicore architecture to address the energy-efficient execution of machine learning algorithms at the edge computing nodes. Two kinds of optimizations are applied : usual compiler optimizations and real number representations (fixed-point versus floating-point). This study is conducted while accounting for the trade-off between training precision, performance, and energy.

Mots-clés : Edge computing, Energy-efficiency, Machine learning, Heterogeneous architecture, FPGA, Embedded systems

1. Introduction

Edge computing [3, 13, 19] is a recent paradigm for Internet-of-Things (IoT) systems, where computations are distributed across a broad range of compact devices, so as to bring computing capability closer to data sources, e.g. environmental sensors and cameras. Given the limited hardware resources and energy constraints on edge computing nodes, one major design issue is the implementation of data analytics, in particular, machine learning (ML) techniques.
The major part of ML algorithms implemented on edge devices concerns *inference* (i.e. the process of straightforwardly using pre-trained models to solve an ML problem) instead of *training* (i.e. the process of minimizing the error as a function of the ML model parameters). Among the reasons [14], is the excessive bandwidth and latency costs required for exchanging network updates back and forth between different edge devices. In addition, energy and hardware cost represents another concern in the edge node design. Inference usually operates on structured labeled data. It is not the case of *deep learning* [16], which achieves training tasks necessitating high precision. Here, complex multi-layer artificial neural networks (ANNs) and huge amounts of raw data are processed. This requires high computing power and data storage capacity unavailable on edge devices.
Authors in [4] presented a survey to shed light on the expectations of future computing architectures for edge nodes. Such nodes will execute a diversity of workloads, which require various computing resources. Heterogeneous architectures [5,8] are good candidates to fill this

*. This work has been supported by the CONTINUUM ANR project under the grant number ANR-15-CE25-0007-01, and the R&D ARPE-CONTINUUM project funded by Région Occitanie (France).

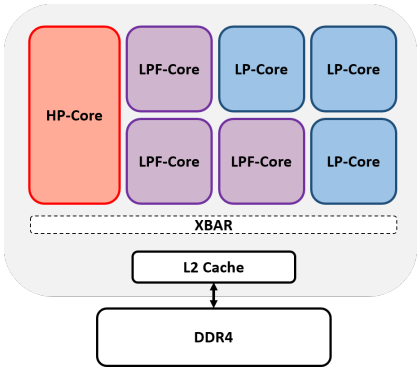
demand as they give access to different CPU types or memory technologies that can be selected for executing various kinds of workloads.
This paper describes a small case study to raise some opportunities in hardware-software co-design for the edge nodes. By considering a FPGA prototype of heterogeneous architecture [11] built from low-power cores, we evaluate the implementation of three typical ML algorithms. This architecture was designed with the ultra-low-power technology from Cortus [2], one of the world-leading semiconductor IP companies in the embedded domain. As a software-hardware codesign approach, we explore some static design optimizations w.r.t. training accuracy, execution time and energy consumption. Instead of using high-level modeling for design space exploration (e.g. cycle-approximate design of heterogeneous architectures in gem5 [6, 10]), we use a FPGA support for accurate ML workload evaluation [17].

2. Case study : FPGA design for ML at the edge

We first assess the impact of various compiler optimizations when implementing typical ML algorithms. Then, we show for these algorithms, how the underlying microarchitecture is better leveraged with floating-point versus fixed-point number representations to find a compromise between accuracy, latency and energy.

2.1. Experimental setup

Experimental architecture design for ML. We consider a heterogeneous multicore architecture [11], composed of seven CPU cores (referred to as *heptacore*) as shown in Fig. 1a. It includes six low power cores devoted to highly parallel workloads for higher throughput, and a single high-performance core for weakly parallel workloads. This architecture is developed with the cost-effective embedded cores provided by the Cortus Company. Here, the high-performance core referred to as HP-core is implemented with the Cortus APSX2 CPU. The three low power cores referred to as LP-core, are implemented with the Cortus APS25 CPU. The remaining three low power cores referred to as LPF-core, are implemented with the Cortus FPS26 CPU.



(a) Architecture organization

Components	Slices	Area (mm^2)
Heptacore	205135	860,93
HP-Core	122941	515,97
LPF-Core	7919	33,24
LP-Core	3981	16,71
XBAR	9047	37,97
L2 cache	270	1,13

(b) Components dimensioning on FPGA

FIGURE 1 – Considered asymmetric heptacore architecture.

While the APSX2 core is an application processor alike with advanced microarchitecture features, the APS25 and FPS26 are microcontrollers. The key difference between the two microcontrollers is the FPS26 includes a floating-point unit (FPU) while the APS25 does not. This provides a tradeoff on precision, latency and power consumption when executing programs on such CPU cores. The communication interconnect is implemented by a hierarchical crossbar. All CPU cores share both the program and data memory but possess local data and instruction

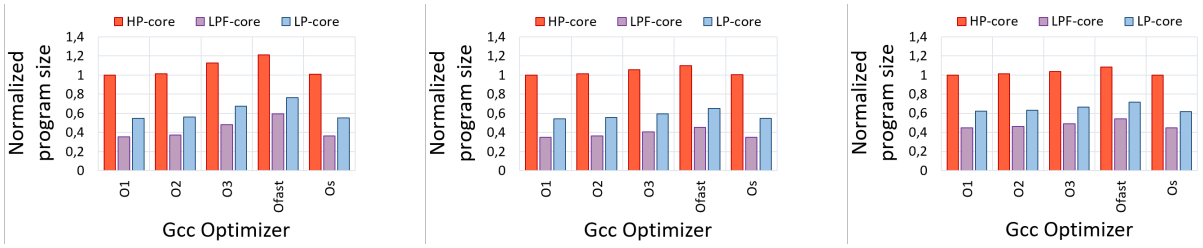
caches. A 128 kb L2-memory caches all the memory accesses to the external 2 Gb DDR4 memory, contributing to reducing the memory access latency. Fig. 1b summarizes FPGA synthesis characteristics of the main architectural components.
The architecture is implemented on the VCU108 FPGA board of Xilinx [1]. This board integrates an external and useful memory for storing the data to be analyzed during the execution of ML algorithms. It also embeds current and voltage sensors on most of its power rails, e.g. power rail of the FPGA chip. This enables us to obtain measures, by leveraging an API developed by Xilinx named SysMon. In our subsequent experiments, only the dynamic power consumption of the FPGA will be measured to evaluate the energy-efficiency of explored designs, by adopting a similar approach as in [9]. This is a commonly admitted basis for reasoning [15].

Selected ML algorithms. Three kinds of ML workloads are considered in our experiments : clustering, classification, and regression algorithms. The *K-means* clustering method is a popular algorithm for unsupervised cluster analysis in data mining. Given a data-set of size n, it produces a partitioning of n data into k clusters according to their proximity.
The *logistic regression* (LogReg) algorithm enables us to determine the probability for a given data to belong to a certain data class. The last ML model considered in the sequel is a *ANN with back-propagation* (Backprop), which exploits gradient descent to realize training and inference tasks. For the sake of simplicity, we consider a three-layer neural network : input, hidden and output layers. The network is composed of 30, 10 and 1 neurons in its input, hidden and output layers respectively. Its total number of weights is 310. The size of this network is deliberately kept small due to the limited hardware resources in the used FPGA.
To carry out our experiments with the above algorithms, we consider a partitioning of data items into two subsets (T, I), where the subset T is used for model training, and the subset I is used for inference based on the trained model. We use (260, 59) and (2000, 80) partitioning for Backprop and LogReg respectively. A set of 2000 items is handled to evaluate the K-means clustering. Note that for each algorithm, we considered as many data items as supported by the memory available in the FPGA prototype. The size of the program corresponding to Backprop is the biggest among the three, hence leaving less space for storing training and test data.

2.2. Software-hardware codesign considerations

In the sequel, we evaluate the impact of two kinds of static code optimizations w.r.t. the considered multicore architecture.

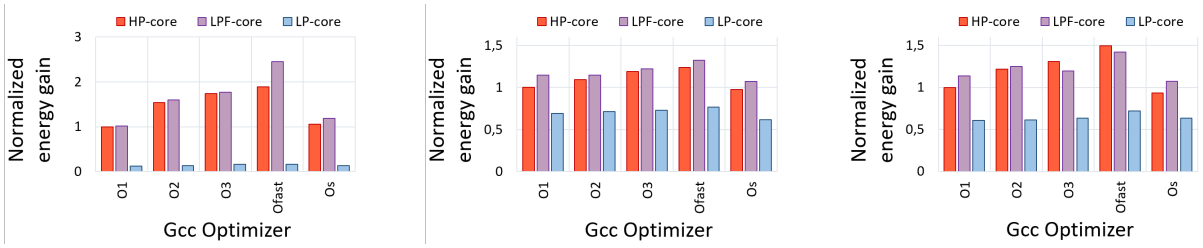
Impact of compiler optimizations. Given the implementation constraints of the target FPGA prototype system in terms of area and power consumption, some relevant optimizations of the GNU Compiler Collection (GCC) are applied to statically select the most efficient binary code versions to execute on the FPGA.
The following options are exercised : `-O1` which aims at reducing the execution time and the code size. It is the lightest optimization option ; `-O2` which is often considered as one of the safest options, it is commonly recommended ; `-O3` which applies more aggressive optimizations to the code beyond those applied by the previous one ; `-Ofast` that includes all `-O3` optimizations and aggressively reduces the execution time (however, it applies optimizations that are not valid for all standard-compliant programs, and can lead to biased program behavior) ; and `-Os` which mainly aims to minimize the code size (it is especially interesting for systems with limited memory resources like FPGAs).
As a preliminary analysis, we evaluate the impact of the above GCC optimizations on the programs corresponding to our selected ML algorithms (Figs. 2 and 3). We compare their respective impacts on the code size for each core type in the architecture, to save FPGA resources. In



(a) Backprop (b) LogReg (c) K-means

FIGURE 2 – Program size for different compiler optimizations, by core type.

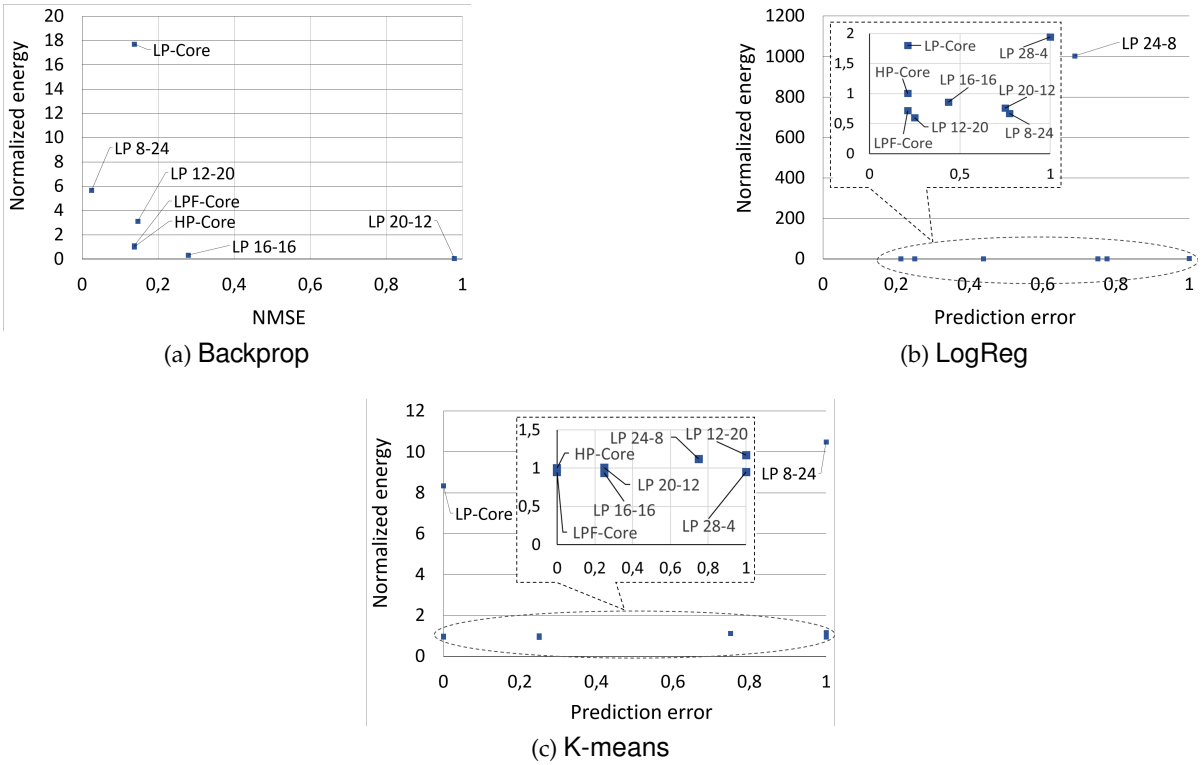
addition, we execute them on different core types to assess the related energy outcome.
More generally, moving towards more aggressive performance optimizations causes increased code size (see Fig. 2). This is a well-known impact of such optimizations, which also increase the code build time while degrading debug experience. Globally, the role of the optimization option in the code size is dependent of its composition. The `-Ofast` option increases the code size by more than 20% compared to `-O1` for Backprop because it contains the highest number of loop iterations. The `-O1` and `-Os` options result in a similar code size. The code size for the `-O2` option is 2% larger than for `-O1` and `-Os`. Regarding energy consumption when executing the different code versions, the `-O1` and `-Os` options do not bring any gain compared to the `-O2`, `-O3` and `-Ofast` options (see Fig. 3). The latter option, i.e. `-Ofast`, even shows a gain of about 135% compared to `-O1` in the case of the Backprop algorithm executed by the LPF-core. The reported energy consumption covers both training and inference phases.



(a) Backprop (b) LogReg (c) K-means

FIGURE 3 – Energy gain when executing optimized program versions, by core type.

From this preliminary observation, the `-O1` and `-Os` options are less attractive than `-O2`, `-O3`. On the other hand, the `-Ofast` option has the disadvantage of leading to biased program behaviors, which would make the comparison a bit difficult in our subsequent experiments. Therefore, for the rest of our experiments, we will only consider the `-O3` option.
Now, when focusing on the `-O3` scenarios in Fig. 3, the HP-core and LPF-core far outperform the LP-core in terms of energy consumption. This is due in major part to the lack of the FPU support in LP-core, which is heavily detrimental to the execution time of the considered algorithms. Floating-point computations are only software-emulated on LP-core, hence the slowdown. The energy gains provided by the LPF-core over the HP-core are respectively +5%, +8.5% and -2% for Backprop, LogReg and K-means respectively.
A preliminary interesting insight is the LPF-core microcontroller represents *a priori* a very relevant alternative to the HP-core application processor from the energy perspective. Moreover, further gains could be expected from the LPF-core regarding the static power consumption



(a) Backprop (b) LogReg

(c) K-means

FIGURE 4 – Energy and accuracy evaluation : fixed-point vs. floating-point LP-Cores.

since its area is 15 times smaller than that of HP-core (see Fig. 1b).

Evaluation of number representations : fixed-point versus floating-point precision. The lack of FPUs in LP-cores for addressing floating-point computations is a clear inconvenient compared to HP-core and LPF-core. To mitigate this limitation, we consider fixed-point number representation for program execution on LP-cores. Such a representation of real numbers defines a fixed number of digits after the radix point, thus inducing a reduction of the arithmetic precision compared to floating-point representation.

We devised code variants for all the three ML algorithms (Backprop, LogReg and K-means) with 32-bit fixed-point number representations. In our experiments, six fixed-point representation instances are evaluated on the LP-core. They are denoted as "LP X-Y", where X represents the number of digits before the radix point and Y represents the number of digits after the radix point (i.e. X + Y = 32). In other words, the integer part of the 32-bit fixed-point arithmetic units is composed of X digits, while the fractional part is composed of Y digits. For example, in the fixed-point instance denoted by LP 8-24, the 8 first digits are dedicated to the integer part of the real number, while the remaining 24 digits are dedicated to the decimal part.

Next, we evaluate the following arbitrary fixed-point instances : LP 8-24, LP 12-20, LP 16-16, LP 20-12, LP 24-8 and LP 28-4. Each of these instances is executed on a single LP-core, and compared with the floating-point variants of the ML algorithms on LP-core, LPF-core and HP-core. Fig. 4 summarizes this comparison in terms of energy and prediction error trade-off : the lower the energy (i.e. near 0), the better the evaluated instance; and the lower the prediction error (i.e. near 0), the more accurate the instance. Note that the reported energy consumption is normalized w.r.t. that of HP-core. In Figs. 4b and 4c, we added a zoom on some regions of the

plots in order to better highlight the difference between the concerned scenarios.
First of all, compared to floating-point algorithm variants executed on LP-core, the fixed-point variants globally reduce the energy consumption by 200% at least, at the cost of an acceptable training accuracy loss, i.e. less than 20% error difference. They even show energy improvements compared to floating-point algorithm variants executed on HP-core and LPF-core, which include an FPU in their microarchitecture. For Backprop and LogReg, the energy reduction grows up to 20%. The best fixed-point precision candidates vary from one algorithm to another. For instance, for LogReg, LP 12-20 appears as the most beneficial, while for the ANN LP 16-16 is the best fixed-point instance. Only the K-means algorithm does not show any energy improvement due to the fact that it manipulates less floating-point arithmetic.
The fixed-point precision therefore represents a relevant alternative to the lack of FPU support in LP-cores. In addition, FPGAs are known to be traditionally energy-efficient for fixed precision computations [20] (even though state-of-the-art FPGAs such as the Stratix 10 [2] from Altera/Intel now integrate competitive FPUs. We note that throughout all the above experiments, the power consumption estimation of the considered FPGA system only varies between 1.3 W to 1.5 W, where the static part represents about 80%.

TABLE 1 – Key performance numbers of considered cores w.r.t. ML model inference (BP and LR respectively denote Backprop and LogReg)

Hardware configurations	HP-core		LPF-core		LP-core		LP-core (fixed-point)		3x LP-core (fixed-point)	
Algorithms	BP	LR	BP	LR	BP	LR	BP	LR	BP	LR
Best perf. (#inferences/sec)	161.64	41.38	149.36	46.83	0.47	2.87	842.85	47.50	**2023.56**	**169.13**
Best energy-eff. (#inferences/J)	806.20	173.16	865.14	275.03	2.53	16.03	4617.13	254.86	**10749.33**	**891.77**

Table 1 summarizes some key performance scores obtained from these experiments. It exclusively reports the best inference scores achieved by each type of core on the Backprop and LogReg algorithms. The shown metrics indicate the number of inferences (or predictions) per second and joule. Interestingly, we observe the improved score provided by the fixed-point precision combined with the throughput optimization enabled by the parallel execution on three low-power LP-cores.

3. Summary and closing remarks

This paper presented a case study on static code optimization for the energy-efficient implementation of selected ML algorithms on low-power heterogeneous multicore system. Further optimization opportunities still remain for drastic improvements in the illustrated FPGA design. First, the major part of the data manipulated in our experiments is stored in the DDR4 memory, separated from the FPGA chip itself. As a result, expensive data accesses are induced both in terms of latency and energy. This issue goes beyond our case study and concerns more generally FPGA designs. Data access pattern analysis for better data locality and model parameters buffering (e.g. ANN weights) have been hitherto considered to improve deep learning networks execution on FPGAs [20]. Finally, designers should be assisted by user-friendly automation tools, capable of supporting all the suitable code optimizations. Existing high-level synthesis tools and approaches targeting FPGAs [7, 12, 18] are an interesting basis to start with.

2. https://www.intel.com/content/www/us/en/products/programmable/fpga/stratix-10.html#family-table

Bibliographie

1. Xilinx Virtex UltraScale FPGA VCU108 Evaluation Kit.
2. Cortus SAS – Advanced Processing Solutions, 2017.
3. Abbas (N.), Zhang (Y.), Taherkordi (A.) et Skeie (T.). – Mobile edge computing : A survey. *IEEE Internet of Things Journal*, vol. 5, n1, 2018, pp. 450–465.
4. Adegbija (T.), Rogacs (A.), Patel (C.) et Gordon-Ross (A.). – Microprocessor optimizations for the internet of things : A survey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, n1, Jan 2018, pp. 7–20.
5. Butko (A.), Bruguier (F.), Novo (D.), Gamatié (A.) et Sassatelli (G.). – Exploration of performance and energy trade-offs for heterogeneous multicore architectures, 2019, https ://arxiv.org/pdf/1902.02343.pdf.
6. Butko (A.), Gamatié (A.), Sassatelli (G.), Torres (L.) et Robert (M.). – Design exploration for next generation high-performance manycore on-chip systems : Application to big.little architectures. – In *IEEE Computer Society Annual Symposium on VLSI*, pp. 551–556, 2015.
7. Coussy (P.), Gajski (D. D.), Meredith (M.) et Takach (A.). – An introduction to high-level synthesis. *IEEE Design & Test of Computers*, vol. 26, n4, 2009, pp. 8–17.
8. Craeynest (K. V.) et Eeckhout (L.). – Understanding fundamental design choices in single-isa heterogeneous multicore architectures. *ACM Trans. Archit. Code Optim.*, vol. 9, n4, 2013, pp. 32 :1–32 :23.
9. da Silva (J. C. R.), Pereira (F. M. Q.), Frank (M.) et Gamatié (A.). – A compiler-centric infra-structure for whole-board energy measurement on heterogeneous android systems. – In Niar (S.) et Saghir (M. A. R.) (édité par), *13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC 2018, Lille, France, July 9-11, 2018*, pp. 1–8. IEEE, 2018.
10. Delobelle (T.), Péneau (P.-Y.), Gamatié (A.), Bruguier (F.), Senni (S.), Sassatelli (G.) et Torres (L.). – MAGPIE : System-level Evaluation of Manycore Systems with Emerging Memory Technologies. – In *EMS : Emerging Memory Solutions*, Lausanne, Switzerland, mars 2017.
11. Gamatié (A.), Devic (G.), Sassatelli (G.), Bernabovi (S.), Naudin (P.) et Chapman (M.). – Towards energy-efficient heterogeneous multicore architectures for edge computing. *IEEE Access*, vol. 7, 2019, pp. 49474–49491.
12. Gamatié (A.), Le Beux (S.), Piel (E.), Ben Atitallah (R.), Etien (A.), Marquet (P.) et Dekeyser (J.-L.). – A model-driven design framework for massively parallel embedded systems. *ACM Trans. Embed. Comput. Syst.*, vol. 10, n4, novembre 2011, pp. 39 :1–39 :36.
13. Khan (W. Z.), Ahmed (E.), Hakak (S.), Yaqoob (I.) et Ahmed (A.). – Edge computing : A survey. *Future Generation Computer Systems*, vol. 97, 2019, pp. 219 – 235.
14. Kukreja (N.), Shilova (A.), Beaumont (O.), Huckelheim (J.), Ferrier (N.), Hovland (P.) et Gorman (G.). – Training on the edge : The why and the how. – In *IEEE IPDPS Workshops*, pp. 899–903, 2019.
15. Kuon (I.) et Rose (J.). – Measuring the gap between fpgas and asics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, n2, 2007, pp. 203–215.
16. Le Cun (Y.), Bengio (Y.) et Hinton (G. E.). – Deep learning. *Nature*, vol. 521, n7553, 2015, pp. 436–444.
17. Nurvitadhi (E.), Venkatesh (G.), Sim (J.), Marr (D.), Huang (R.), Hock (J. O. G.), Liew (Y. T.), Srivatsan (K.), Moss (D. J. M.), Subhaschandra (S.) et Boudoukh (G.). – Can fpgas beat gpus in accelerating next-generation deep neural networks ? – In Greene (J. W.) et Anderson (J. H.) (édité par), *Proceedings of the ACM/SIGDA Int'l Symposium on Field-Programmable Gate Arrays, FPGA 2017, Monterey, CA, USA, February 22-24, 2017*, pp. 5–14. ACM, 2017.

18. Quadri (I. R.), Gamatié (A.), Boulet (P.) et Dekeyser (J.-L.). – Modeling of Configurations for Embedded System Implementations in MARTE. – In *1st workshop on Model Based Engineering for Embedded Systems Design - Design, Automation and Test in Europe (DATE 2010)*, Dresden, Germany, mars 2010.
19. Satyanarayanan (M.). – The emergence of edge computing. *Computer*, vol. 50, n1, janvier 2017.
20. Shawahna (A.), Sait (S. M.) et El-Maleh (A.). – Fpga-based accelerators of deep learning networks for learning and classification : A review. *IEEE Access*, vol. 7, 2019, pp. 7823–7859.