Work-In-Progress: Could Tensorflow applications benefit from a mixed-criticality approach?

Alan Le boudec+*, Frank Singhoff*, Hai Nam Tran*, and Stephane Rubini*, Sebastien Levieux*, Alexandre Skrzyniarz+

+ Thales DMS, Brest, France
*Univ. Brest, Lab-STICC, CNRS, UMR 6285, F-29200 Brest, France
alan.le-boudec@thalesgroup.com, singhoff@univ-brest.fr, rubini@univ-brest.fr, hai-nam.tran@univ-brest.fr, alexandre.skrzyniarz@fr.thalesgroup.com

Abstract. In this article, we investigate the interest in applying a mixed-criticality approach to schedule convolutional neural network (CNN) applications on multicore architectures. We deal with software composed of real-time interactive applications and CNNs that have different criticality levels. A classical means to schedule software with various criticality levels is to apply partitioning methods to enforce spatial and temporal isolation, which may be inefficient if application execution times have a high level of variability. In that case, applying a mixed-criticality approach may improve resource usage. We conducted a measurement campaign to assess the variability of CNN execution time and investigate whether this kind of application could benefit from a mixed-criticality approach. The results show that the execution times of the chosen CNN application vary with an average execution time of 109 ms and a worst case of 252 ms. Furthermore, they indicate a potential save of computing resources up to 73% when applying a mixed-criticality approach instead of partitioning methods.

Keywords: Real-time Scheduling, Mixed-criticality, CNN.

1 Introduction

[Context]

This article focuses on real-time applications embedding AI algorithms. Let's assume a video surveillance system where AI algorithms are run to detect objects in images acquired by cameras. Once the objects are identified, users may focus on one of them and perform authentication. Such a system involves applications that analyze sensor data with AI algorithms and interactive applications that handle user actions. We assume those applications have different criticality levels: analysis applications may have a higher level of criticality than interactive ones. They have both to meet timing constraints such as deadlines, and we must schedule them in order to enforce that the higher their criticality level is, the higher the probability of meeting the deadline is.

To schedule these applications, a classic way [5] is to apply a partitioning approach leading to a spatial and temporal isolation of the applications according to their criticality. Unfortunately, this approach leads to over-reservation and wastes computing resources [15]. An alternative method is to use a mixed-criticality approach [3] where the tasks are adapted at runtime according to the available resources. In a mixed-criticality approach, each task may have several execution time budgets, each reflecting a safety level that will be used during scheduling and schedulability analysis to maximize the number of met deadlines of the tasks. Such an approach contributes also to minimizing the waste of computing resources.

[Problem Statement]

Mixed-criticality scheduling theory aims to design efficient resource management of systems that exhibit non-determinism, variability, and uncertainty during runtime [2]. While variability and uncertainty due to multiprocessor architectures are widely accepted, less work has investigated the level of variability and non-determinism due to CNN applications. Evaluating the variability of CNN application execution times may motivate the use of mixed-criticality approaches instead of partitioning for such kinds of applications.

[Contribution]

In this article, we investigate the timing behavior of a set of tasks running CNN applications on a multiprocessor architecture. CNN applications are implemented with $Tensorflow$. By a measurement campaign, we evaluate the variability and uncertainty of their execution times. We quantify how such applications could benefit from a mixed-criticality approach.

The article is organized as follows. Section 2 presents the background of our work. Section 3 discusses the measurement campaign and analyses the results. Section 4 provides a brief overview of related works. Finally, section 5 concludes the article.

2 Background

In this section, we introduce the mixed-criticality approach and define the CNN models used in this article.

2.1 Mixed-Criticality Approach

The concept of mixed-criticality has been introduced when software and hardware became more complex and less predictable from a temporal point of view [4].

In a mixed-criticality system, high and low-criticality tasks share the same computing units without resource reservation. In contrary of a spatial and temporal isolation solution, mixed-criticality systems promote the adaptation of low-criticality tasks when resource starvation is detected for high-criticality tasks.

A key aspect of mixed-criticality described by [4] is that task parameters, such as the worst-case execution time (WCET), that become dependent on the criticality level of the task and may reflect the level of safety the task belongs to. The Vestal's [15] model was one of the first mixed-criticality approaches. In his work, the author shows that applying a mixed-criticality approach allowed to increase of 20% the resource usage of the system.

2.2 Convolutional Neural Networks

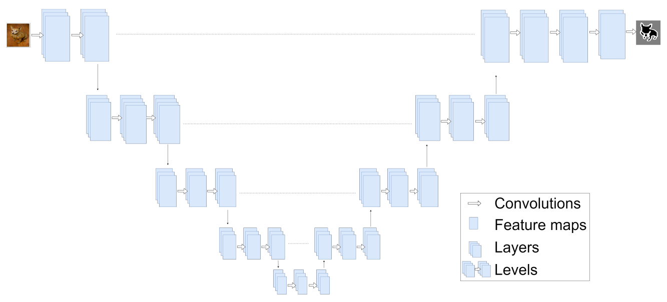

Fig. 1: U-NET architecture

The architecture of CNNs is inspired by visual perception [8]. CNN are composed of full-connected neurons organized in layers able to learn hierarchical representations [9]. CNNs apply deep learning to analyze images, classify visuals, or perform computer vision tasks.

Figure 1 shows a specific CNN architecture called a U-NET [14]. A U-NET is a CNN model designed for image processing. It is made of a set of levels. Each level may have one or several layers. When a layer is used to apply a filter on an image, it is called a feature map. Figure 1 shows a 9-level U-NET with 64 feature maps. The first level in the left corner of the figure has 2 layers. This work is being carried out in an industrial context in which the CNNs are U-NET. Then, in this article, we restrict the analysis to U-NET only.

3 Approach and Experiments

In this section, we present the results of our experiments. The objective of them is to estimate the CNN execution time variability and its level of non-determinism.

3.1 Platform and Experiment Conditions

We run our experiments on a multicore platform with 2 processors Intel Xeon Gold with 18 computing cores, giving us 72 virtual cores and 256 GO RAM. This platform hosts a UNIX system Debian 11. We run a U-NET written in python [12]. This implementation uses the *Tensorflow* library [1]. Each *Tensorflow* application was run in isolation (i.e. no other application was run during the experiment). We arbitrarily run this Python program 100 times for each U-NET model. We use 32 U-NET models, each had a given number of levels and feature maps. The number of levels is ranging from 1 to 4, and the number of feature maps is ranging from 8 to 64 in increments of 8.

3.2 Results

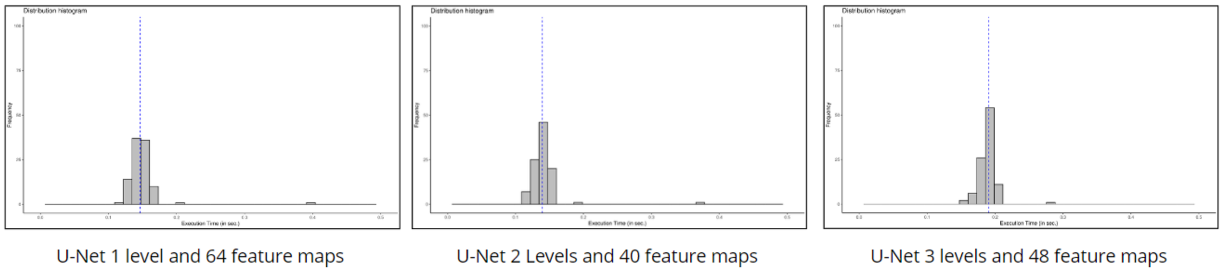

Fig. 2: Distribution of the execution times. x: execution times; y: data distribution

The programs are run in the `SCHED_OTHERS` scheduling policy with a priority level of 0. First, we let *Tensorflow* automatically creates threads to run the program and maps them on the cores of the platform. In that case, we notice that all cores were used and 65 threads were launched.

Figure 2 shows the execution time of the U-NET models for this first experiment. Each histogram presents the density distribution of the execution times for one U-NET model, i.e. each bar in the figures is the percentage of similar execution time values. Note that only few samples are represented in figure 2 and 3. All measures can be found in the URL given in the artefact section.

In this figure, we can see that most of values are around the median and that very few of them are far from it. In the case of a U-NET with 2 levels and 40 feature maps, the standard deviation is 26,5 with an average execution time of 141 ms and the worst-case execution time is 373 ms. It confirms the variability in the execution times. Furthermore, if the resource reservation is made according to this worst-case execution time, it leads to a waste of 62% of the computing resource.

The variability of the execution time comes either from 1) the program itself. It may come from the computations in the nodes of the U-NET; 2) either from the execution model of *Tensorflow*. It may come from how *Tensorflow* maps and schedules the threads making the U-NET computations; 3) from the shared resources of the multicore platform. In this last case, it may come from task migration or the unpredictability related to the memory cache units.

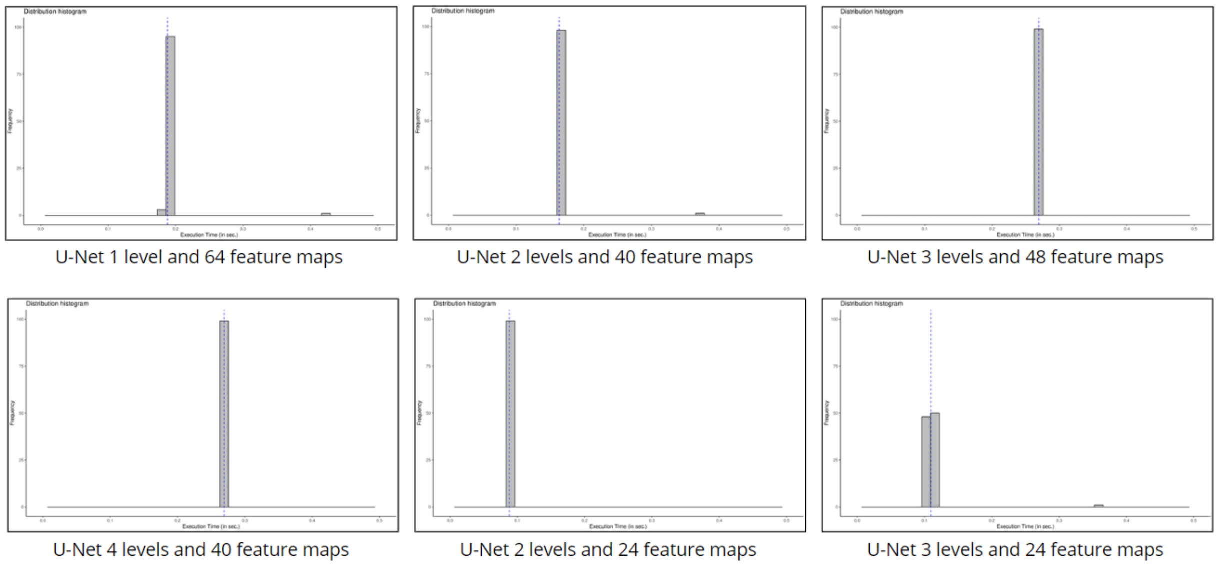

Fig. 3: Distribution of the execution times without thread migration. x: execution times; y: data distribution

In a second experiment, we aim to reduce the execution time variability due to *Tensorflow* parallelization and due to the multicore platform. For such purposes, we now forbid thread migration between the cores by the use of a thread affinity mechanism. Again, we run the 32 U-Net models (each 100 times) with those new constraints and we get the results shown in figure 3.

If we take our example of U-NET with 2 levels and 40 feature maps of the first experiment, the standard deviation is now of 20 ms with an average execution time of 166 ms and a worst-case execution time of 369 ms. We obtain a waste of computing resources of 55%. Without migration, we notice for this example a reduction of the waste of computing resources from 62% to 55%. To conclude, forbidding migration reduces execution time variability but does not make *Tensorflow* programs fully predictable from a timing point of view.

Now, we evaluate the gap between the worst-case execution time and the average execution time for all U-NET models. This gap represents the amount of resources that a mixed-criticality approach may save if applied on U-NET applications in the best case. This gap is shown in the figure 4.

The gap is calculated by subtracting the worst-case execution time and the average execution time itself computed without the worst-case execution time. This gives us the potential saving in computing resources when applying a mixed-criticality approach instead of a resource reservation made on the worst-case execution times.

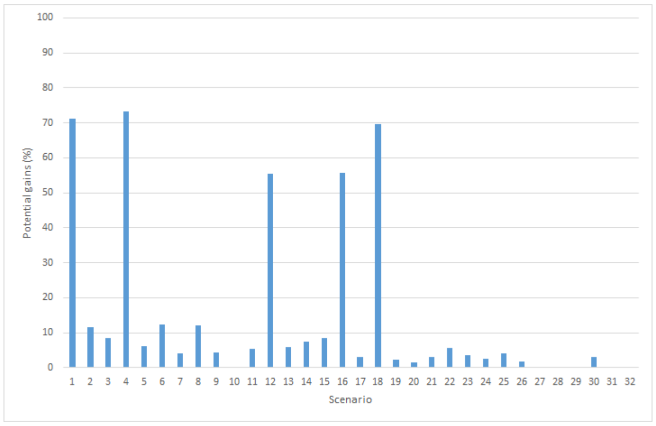

Fig. 4: Potential computing resource saving depending on U-NET scenario

Figure 4 shows the gap for a set of U-NET sorted by their number of neurons. This gap ranges from 0 to 73%. In the best case, a gap of 73% means that a mixed-criticality approach would save upto 73% of the resource to execute the program. In the gap, we do not consider the overhead due to the mixed-criticality services implementation. Note that the real gaps may be higher than the one shown in figure 4. Indeed, we are computing a gap with a worst-case execution time measured during U-NET execution while the real worst-case execution time could be higher.

The U-NET with the largest gap has an average execution time of 109 ms, and a WCET of 252 ms, i.e. 2.5 times more. In this U-NET, only one program execution reaches the worst-case value, implying that reserving the resources with this WCET represents only 1% of the execution number of this U-NET. About the 32 U-NETs, 9,65% of the 3200 program executions cases are above the upper fence. The upper and lower fences represent the cut-off values for upper and lower outliers in a dataset. It is calculated from the interquartile range (IQR), representing the difference between the first and 3rd quartile.

Furthermore, we notice that the larger the U-NET in terms of the number of neurons is, the lower the gap (and consequently the variability) is.

To conclude, these experiments confirm that the U-NET parameters such as the number of neurons, the number of levels or the number of feature maps may have an impact on execution time variability. Furthermore, applying a mixed-criticality approach when U-NETs are run on a multicore platform could save resources compared to resource reservation.

4 Related work

Several works already investigated the temporal behavior of CNN programs.

Inference phase of a CNN program is composed of basic operations in which the same parameters have a similar behavior between two executions. [6,13,7]

have designed mathematical models and performed measures of those basic operations. They have shown that the timing behavior of basic operations are predictable. Contrary to us, they have not investigated the timing behavior of the overall network.

In [10], CNN programs are run with various inputs. These authors show that the execution time of CNN varies when the scenario of the inputs varies. In our experiments, we did not make any assumption on the inputs: CNN are run with the same set of inputs.

Execution time variation may come from hardware hosting the CNN programs [11]. The authors of [11] have investigated execution time variations of co-located jobs to reduce access memory or execution time variation due to communications on a ROS middleware. Furthermore, a limited amount of memory may also affect the latency of inference [16]. In our experiment, we have run CNN programs on a multicore platform where shared resource accesses (e.g. L1 or L2 caches) are known to reduce predictability of execution times.

Finally, measures have been made to test different ways to decrease the execution time latencies of a CNN inference, generally to meet system requirements such as a minimal frequency to run a periodic CNN. Examples of those means are the lowering of the quality or by adapting the CNN to the hardware platform [16]. In this article, we do not look for any execution time reduction.

To summarize, we presented several articles investigating CNN execution times, but none have quantified the variability of the execution time. In this article, we want to quantify how and when CNN program execution times may vary on a multicore architecture and if applying a mixed-criticality approach could save computing resources.

5 Conclusion

Mixed-criticality theory aims to provide an efficient management of computing resources to master the unpredictability and the variability of task execution times in real-time systems. In this article, we investigated the execution times of CNN tasks implemented with *Tensorflow* and running on a multicore platform running a Debian Linux distribution. This work is being carried out in an industrial context in which CNNs are U-NET. Then, in this article, we restrict the analysis to U-NET only.

We evaluated the variability of U-NET execution times. We have shown that in the worst scenario, the average execution time is about 109 ms with a worst-case execution time of 252 ms. The experiments showed that applying a mixed-criticality approach instead of a partitioning/resource reservation with CNN applications could save upto 73% of the computing resources. The experiments also showed that the number of neurons, feature maps, and levels contributes to this variability.

The overall results underline the relevance of using a mixed-criticality approach for the scheduling of CNN tasks instead of a classic partitioning method.

In these experiments, we run tasks on a multicore platform and a part of the execution time variability/unpredictability may come from the hardware platform. In future works, we plan to run similar experiments with GPU to identify the variably/unpredictability which is strictly due to *Tensorflow* CNN programs. Until now, we also focus on CNN implementations with *TensorFlow*, but the next step would be to explore other artificial intelligence engines.

6 Artefacts

Elements of artefacts can be reached at http://beru.univ-brest.fr/svn/CHEDDAR/artefacts/RTSS23

References

1. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
2. Sanjoy K Baruah. Mixed-criticality scheduling theory: Scope, promise, and limitations. *IEEE Des. Test*, 35(2):31–37, 2018.
3. Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, pages 1–69, 2013.
4. Alan Burns and Robert I Davis. A survey of research into mixed criticality systems. *ACM Computing Surveys (CSUR)*, 50(6):1–37, 2017.
5. Nuno Diniz and Jose Rufino. Arinc 653 in space. In *DASIA 2005-Data Systems in Aerospace*, volume 602, 2005.
6. Thomas Garbay, Khalil Hachicha, Petr Dobias, Wilfried Dron, Pedro Lusich, Imane Khalis, Andrea Pinna, and Bertrand Granado. Accurate estimation of the cnn inference cost for tinyml devices. In *2022 IEEE 35th International System-on-Chip Conference (SOCC)*, pages 1–6. IEEE, 2022.
7. Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving dnns like clockwork: Performance predictability from the bottom up. *arXiv preprint arXiv:2006.02464*, 2020.
8. David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.
9. Teja Kattenborn, Jens Leitloff, Felix Schiefer, and Stefan Hinz. Review on convolutional neural networks (cnn) in vegetation remote sensing. *ISPRS journal of photogrammetry and remote sensing*, 173:24–49, 2021.
10. Liangkai Liu, Zheng Dong, Yanzhi Wang, and Weisong Shi. Prophet: Realizing a predictable real-time perception pipeline for autonomous vehicles. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 305–317. IEEE, 2022.

11. Liangkai Liu, Yanzhi Wang, and Weisong Shi. Understanding time variations of dnn inference in autonomous driving. *arXiv preprint arXiv:2209.05487*, 2022.
12. Margaret Maynard-Reid. U-Net image segmentation in keras. *PyImageSearch*, 2022. https://pyimg.co/6m5br.
13. Francesco Restuccia and Alessandro Biondi. Time-predictable acceleration of deep neural networks on fpga soc platforms. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 441–454. IEEE, 2021.
14. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.
15. Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE international real-time systems symposium (RTSS 2007)*, pages 239–243. IEEE, 2007.
16. Zhao Yang and Qingshuang Sun. Efficient resource-aware neural architecture search with dynamic adaptive network sampling. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.