DOI: https://doi.org/10.1109/ICCWorkshops49005.2020.9145197

Towards Very Low-Power Mobile Terminals through Optimized Computational Offloading

Hergys Rexha*, Sébastien Lafond*, Giovanni Rigazzi†, Jani-Pekka Kainulainen‡
*Åbo Akademi University, Faculty of Science and Engineering, Turku, Finland
‡InterDigital Europe, London, UK
†InterDigital Germany GmbH, Berlin, Germany

***Abstract*—Energy consumption is a major issue for modern embedded mobile computing platforms, and with new technological developments, such as IoT and Edge/Fog computing, the number of connected embedded mobile computing systems is rapidly increasing. Heterogeneous multi-core CPUs seek to improve the performance of these platforms, with a particular focus on energy efficiency. By using different techniques like DVFS, core mapping, and multi-threading, a substantial improvement in the achievable CPU energy efficiency level for Multi-processor system-on-chip (MPSoC) can be observed. However, controlling only the CPU power dissipation has a limited effect on the overall platform energy consumption. Other components of the platform, including memory, disk, and other peripherals, play an important role in the energy efficiency of the platform and need to be taken into account. The availability of different sleep strategies at various levels of the platform makes the energy efficiency issue even more complex. In this paper, we set the view of energy efficiency at the entire platform level and discuss computation offloading as a mechanism to help in reaching the optimal platform energy-efficient state. As an application, we consider object detection performed on several types of images to define when offloading is beneficial to the platform energy efficiency. We survey the energy efficiency of different neural network algorithms in an embedded environment, with the possibility to perform computation offloading, and discuss the obtained results concerning the level of object recognition accuracy provided by different neural networks.**

***Index Terms*—Energy efficiency, computation offloading, object recognition, embedded computing platforms.**

I. Introduction

In recent years, the number of connected devices has been dramatically increasing, with predictions for 2025 suggesting the number to reach over 8 Billion mobile broadband connections and over 5 Billion IoT connections [1]. Furthermore, new classes of services are emerging requiring support in future networks, such as rich 4K/8K video services for Mixed Reality (MR) applications with tactile feedback [1]. These, together with industrial automation control, autonomous ground, and air vehicles will provide further challenging requirements for future networks, noticeably from bandwidth, latency, reliability, and energy efficiency perspectives. Edge computing can theoretically provide high bandwidth, low latency, and the computing agility required by today's new digital services. On the other hand, Information and Communication Technology (ICT) drives an unstoppable process of ever-growing energy consumption, with new devices, services, and data produced at a rapid pace. The 5G and beyond network is set to support huge volumes (in the order of TBs of data per day [2]) of multi-dimensional data coming from different heterogeneous nodes, devices, and applications. This raises major challenges with respect to data transmission and processing. Ultimately, such a vast amount of data together with low latency application requirements, e.g., cloud gaming [3] calls for new approaches to perform processing in an energy-efficient manner. One key aspect of edge is that many of the devices are battery-powered or deployments are power limited. The more energy-efficient the edge processing is, the more computation can be done with the same power budget. Application complexity and power consumption are increased when distributed AI is deployed to edge devices. Furthermore, due to the ever-increasing number of such devices, the technology is expected to consume a significant amount of energy [4], thus playing a major concern in future strategies of curbing down energy consumption.

Multi-core heterogeneous CPUs promise to give a substantial contribution to the increase of the energy efficiency in edge platforms. Different strategies inside the MPSoC are explored to achieve a reduction in power dissipation and relative energy consumption. Nevertheless, it is not enough to consider only one component, e.g., at the CPU level, in the strategy for achieving energy efficiency, rather one must consider energy consumption at the platform level [5]. Depending on the platform type, I/O connected, and the type of applications executed on it, different execution strategies might be the solution to energy conservation. For instance, the Race to Halt strategy is proved to be a solution if the CPU is not the major power consumer in the platform [6]. By contrast, if the static power dissipation of the platform is relatively low, the execution with a lower clock frequency of the application might save energy by going slowly to the application results [6].

In this paper, we aim at locating the working configuration in which the overall platform energy consumption is minimized. Computational offloading is the mechanism of moving heavy tasks to more powerful computing units. This mechanism can be a winning strategy for achieving good levels of energy efficiency in the case of highly demanding applications. The goal of this paper is to analyze the impact of computational offloading on modern embedded mobile computing platforms, having multi-core heterogeneous architectures, to achieve platform-level energy efficiency in the case of highly computational-demanding object detection and recognition applications.

The main contributions of this paper are the following:

- We present experimental results including energy ef-

ficiency analysis of embedded environment from the platform energy consumption perspective.
- We demonstrate computational offloading as a solution to help to achieve the goals of platform energy efficiency for embedded computing platforms.
- We analyse the energy cost of different neural networks performing object detection and recognition on a mix of input images.

The rest of the paper is organized as follows. In Section II, we review the related work. In Section III, we develop the discussion of energy efficiency extended to the platform level. In Section IV, we present an offloading strategy aiming at achieving energy efficiency. Section V is explained AI approach for object detection and recognition. Finally, we conclude with experimental testing and numerical result discussion in Sections VI and VII.

II. Related Work

Several works are addressing the topic of energy management and computation offloading in mobile systems. More specifically, offloading compute-intensive tasks towards more powerful systems is a well-known research area, which dates back to the '70s. In [7], the authors present a framework for computation offloading that is based on a comparison of local and remote cost of computations. The decision making part of the framework predicts the communication bandwidth for assessing the costs. In this work, we experimentally validate different communication bandwidths and propose the bandwidth as a metric for the offloading decision. In [8]–[12] authors present offloading techniques that target battery-operated devices with the characteristic of possible offloading of single methods inside the task that needs to be executed. By contrast, we focus on offloading the full task without previously running it on the local or remote side. On the energy management side of the multi-core architectures, [5], [13] show the possibility to maintain low-power dissipation and at the same time to accommodate computation-intensive applications. In [5], authors propose an algorithm that selects the best execution option in terms of energy conservation, depending on the type of application and during runtime the mapping decision might change depending on the phases of the program. On the other hand, we adopt a joint approach of finding the optimal execution configuration inside the heterogeneous architecture and explore the usage of offloading as a mechanism for achieving energy efficiency.

III. Energy efficiency at the platform level

Multi-core heterogeneous CPUs have been introduced in an attempt to increase the energy efficiency of mobile devices, by using the appropriate computing element for the considered task. Complex out-of-order cores are used to handle compute-intensive tasks where performance is central to the outcome, meanwhile, simple lightweight in-order cores provide support for background repetitive tasks that run continuously, yet do not have time constraints. Modern platforms offer a range of different core capabilities: simple cores have simple data-path and low power profile, medium-complexity cores introduce more complex pipeline, and high-complexity cores work with high levels of parallelism and high voltages. The use of high voltages in complex cores comes with highly dynamic power dissipation which will increase the temperature of the chip and also the relative static power dissipation which comes as main side effect [14]. The adoption of actuators in today's MPSoC can certainly provide energy efficiency at the chip level as demonstrated in [15], [16] but cannot have a major effect in the overall platform energy consumption. Other parts of the system like memory and I/O result in high energy expenditure. Lowering the core voltage (V) can make transistors switch slowly which, in turn, forces the frequency (f) of the core to a lower level. This can have a great impact on reducing the dynamic power which is proportional to $f * V^2$. However, it comes with the cost of increasing the running time of a task which may increase the energy consumption of other parts of the platform [17] and, therefore decrease the overall energy efficiency of the platform. In Figure 1 we run a neural network application on a Single-board Computer (SBC) and we measured the energy consumption of the CPU, platform and overall board for different CPU frequencies. As it can be noticed from the results, the best strategy from an energy point of view would be neither to use the low nor the high clock frequencies, but setting the clock frequency in between. The dots represent experimental data while the lines show the polynomial curve fitting.

In our previous work [15] we explored the energy efficiency of the MPSoC in multi-core heterogeneous architectures by using the concept of configuration points, that is a tuple of parameters, such as the type of cores used in computations, number of cores, performance level of cores (DVFS), and utilization level of the task using a core. By selecting for execution a specific class of these points, we can achieve better energy efficiency on the level of the CPU.

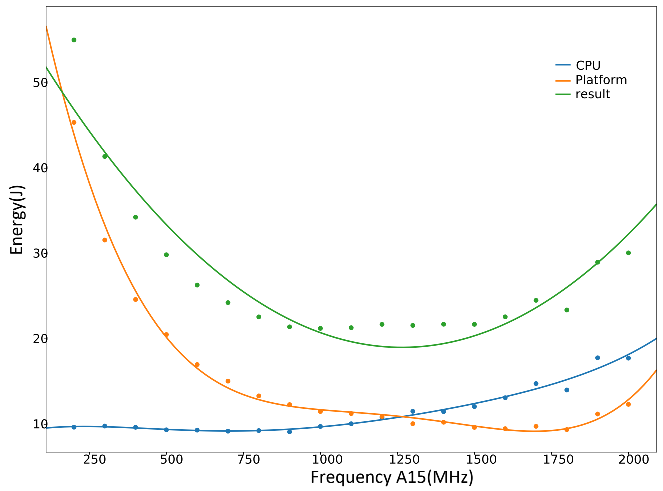

Fig. 1: Measured CPU, base, and total energy consumption of the platform.

IV. Offloading strategy

Despite continuous technological and performance improvements, embedded mobile computing devices come also with physical constraints, such as processing power and, more importantly, battery life. Many applications including natural language processing, image recognition or online gaming, are challenging in terms of computational requirements. Computational offloading is a mechanism where a task can be sent for computation to a more capable remote unit. As soon as the remote machine has completed the task execution, the results are sent back to the device which initiated the offloading. In turn, computational offloading is expected to be an effective solution to address the data explosion produced by the massive increase of digital services most often run by resource-constraint devices.

A. *Application type*

The range of applications that are deployed on embedded mobile computing devices, for example in autonomous robots or drones, potentially benefiting from offloading, is very broad. 3-D mapping, speech recognition, object detection and recognition are among the usual candidates for employing computational offloading. Applications change between each other with a diversity of factors e.g., concurrency, which might be at the task level or at the data level as in streaming applications. Emerging use cases are those requiring intelligent image processing in drones for example, where first processing of the captured images could be done on the drones, even though computationally expensive pattern recognition to detect certain special conditions, as fire or floods, could be done in the edge. Another interesting application from Industry 4.0 seeks for achieving zero-defect manufacturing (ZDM), where cameras provide 4k/HD video stream of the goods moving in conveyor belts, and AI services deployed in the edge/fog can provide the intelligence for identifying damaged parts and controlling the robotic arm to sort the material in accordance. In this work, we consider an object-detection application run through a deep neural network. We run different neural networks for object detection on images of different sizes and background complexity. We then define what are the conditions to consider to achieve an optimal solution when offloading tasks.

B. *Offloading criteria*

Offloading is a natural solution when the application presents real-time constraints that could not be met by the mobile platform. Applications are divided into two parts: one part is executed on the mobile platform and the second part, which requires the majority of computations, is handled by the remote server. The mobile platform could be considered as an interface that handles requests or takes images from a camera, and the remote side might be responsible for the heavy computation. If we denote mobile platform speed as s_l and the application workload as w, the time needed to compute locally would be $\frac{w}{s_l}$. Next, to offload to a remote server, we need to send some input data denoted as d_i over network communication, which has a bandwidth B, to a server, whose speed is denoted as s_r, and receive the results as d_o.

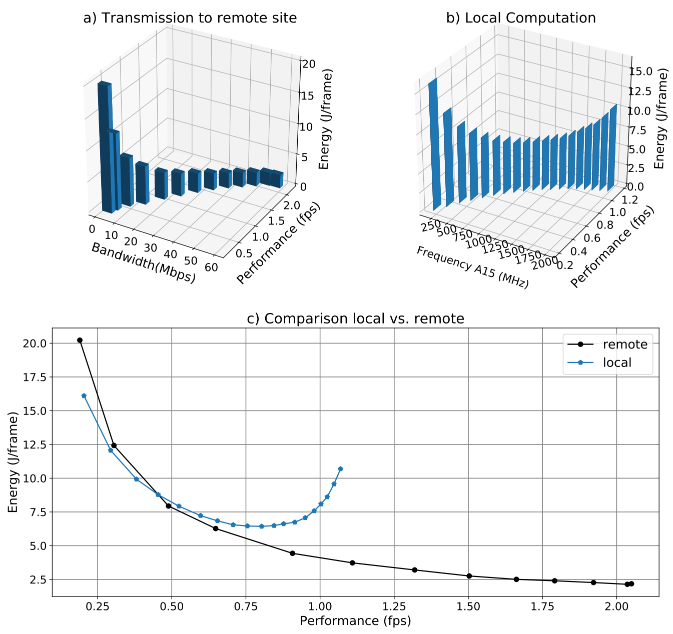

Fig. 2: Energy efficiency of local and remote computation.

The synchronization with the remote side is achieved over TCP protocol and the synchronization time is denoted as t_{tcp}. Overall the time spent to execute the remote computation is:

$$2 * t_{tcp} + \frac{d_i}{B} + \frac{w}{s_r} + \frac{d_o}{B}$$

Offloading increases the performance if the inequality holds:

$$\begin{aligned} &\frac{w}{s_l} > 2 * t_{tcp} + \frac{d_i}{B} + \frac{w}{s_r} + \frac{d_o}{B} \Longrightarrow \\ &w * \left(\frac{s_r - s_l}{s_r * s_l}\right) > \frac{d_i + d_o}{B} + 2 * t_{tcp} \end{aligned} \tag{1}$$

and this can happen in the following cases:

- large w: the amount of computation is considerable;
- $s_r \gg s_l$: the remote side is way faster than local side;
- $d_i + d_o$, the amount of transmitted and received data, is small;
- B is large.

We can also notice that if $\frac{w}{s_l} < \frac{d_i + d_o}{B}$ then offloading will not increase performance even if the remote side is infinitely fast ($s_l \longrightarrow \infty$). In Figure 2 we show the result of performing object detection, using an Single Shot Detector (SSD) neural network as described below, locally by a mobile platform or offloaded to a remote server for computation. In graph c), we present a comparison of the performance vs. energy efficiency reached by the local platform and remote side. We can notice that the mobile platform can achieve performance up to 1.2fps, and if we need an additional frame rate, the computation should be offloaded to the remote side. Again from Figure 2, we notice in graph a) that offloading is energy efficient if the bandwidth of network transmission is as high as possible, on the other hand from graph b) we see that local computation is energy efficient if we use middle core frequencies instead of high or low frequencies.

Another factor that concerns mostly battery-operated devices is energy. Despite technological improvements, battery technology is still far behind the improvements made in other areas of computing systems such as memory, CPU, and screen. Therefore, offloading to extend battery life and conserve energy is of paramount importance, and this criterion can be named as offloading for energy efficiency. If the power to compute task w by the mobile platform is p_l, then the energy consumed by executing the task locally is $p_l * \frac{w}{s_l}$. Otherwise with p_i the power of the system while idling and p_t the power of platform while transmitting, the energy to offload the task can be defined as:

$$p_t * \left(\frac{d_i + d_o}{B} + 2 * t_{tcp} \right) + p_i * \frac{w}{s_r}$$

In this case, the offloading mechanism saves energy if:

$$p_l * \frac{w}{s_l} > p_t * \left(\frac{d_i + d_o}{B} + 2 * t_{tcp} \right) + p_i * \frac{w}{s_r} \tag{2}$$

The same observation applies as before, so: to save energy we should look at the amount of data to be transferred, the type of workload and the available bandwidth of the transmission network.

V. Object detection in images

All object recognition pipelines are usually composed of 3 stages: the detection phase, where objects of interest are detected and located in bounding boxes, feature extraction where the object inside the bounding box is analyzed to extract features that represent it, and the recognition phase, where labels are assigned to each object in the bounding box. In the deep learning approach to object recognition, two components are distinguished: the meta-architecture (or object detection framework) and the base network.

A. Type of neural network meta-architectures

In this paper, we analyzed two different types of commonly used neural network meta-architectures: SSD and You Look Only Once (YOLO).

1) SSD: This general term is used to refer to architectures that use a single feed-forward convolutional network to directly predict classes and anchor offsets. This algorithm eliminates proposal generation and subsequent pixel or feature resampling and encapsulates computations into a single network. This makes the system easy to train and integrates into components that require detection [18].

2) YOLO: This introduces a new approach for the detection phase. Instead of using an image classifier at the end, a single neural network predicts bounding boxes and class probabilities directly from the image. This gives the possibility to optimize end-to-end performance. However, because of the detection approach used in YOLO, small objects or groups of small objects located next to each other might become difficult to detect [19].

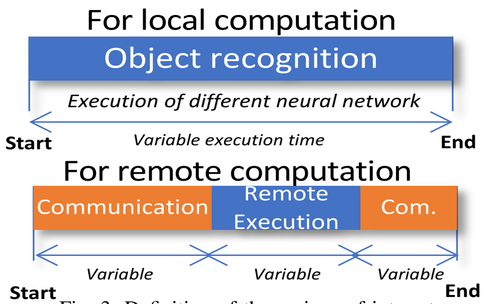

Fig. 3: Definition of the regions of interest

B. Base networks for feature extraction

Several networks, such as Resnet, VGG-16, Inception, Xception, MobileNet, have been developed as base networks performing feature extraction. Some of them, e.g., VGG-16, VGG-19, Resnet-101, have a relatively large memory footprint which makes them impossible to execute on embedded mobile computing platforms, having limited memory capabilities. Other solutions, like Resnet-50 and especially MobileNet, are optimized for running on embedded computing platforms and have therefore smaller memory footprints and computational requirements. In this paper, we evaluate the following four base networks: Resnet, MobileNet, Inception, and Xception, as they are the ones designed to be executed on embedded computing platforms.

VI. Experimental platform and testing

As a central mobile platform, we use an SBC which will run our main application. We run our application on an Odroid XU4 development platform provided by HARDKERNEL. The board is equipped with an Exynos 5 MPSoC, which is an octa-core composed of 4 ARM Cortex A7 and 4 ARM Cortex A15 organized in a big.LITTLE architecture with Global Task Scheduling (GTS). The A7 cores (little cores) can scale up to a frequency of 1.4 GHz, while A15 cores (big cores) can reach a frequency of 2 GHz. The development board runs a Linux OS with kernel 4.2. The board network connectivity is provided by a USB Wi-Fi 802.11n dongle W522U, which is a dual-band wireless USB adapter that provides maximum wireless speed up to 300Mbps over two bands, with a maximum transmit power of 18dBm. The drivers offer two work modalities, one for full power transmission and the other for low power operation while listening for connections. We use both modalities in our experiments. We define our Region Of Interest (ROI) as illustrated in Figure 3. When the object recognition task is executed locally, the ROI is defined as the execution time of the used neural network to recognize an object in the provided input image. When the object recognition task is executed remotely, the ROI is defined from the moment the TCP connection is negotiated with the remote server until the moment the embedded mobile computing platform receives all coordinates, size, and tags associated to all recognized objects.

For measuring the power dissipation inside the ROI, we need an approach that records at a high sampling rate the current consumed during variable execution and communication periods. For example, for the input pictures used in this

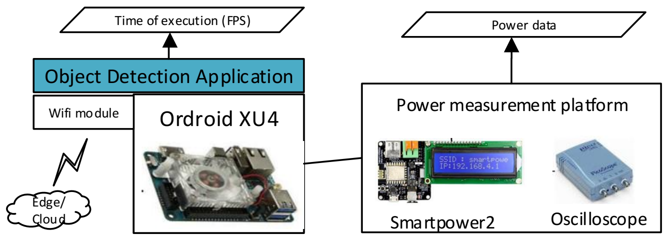

Fig. 4: Schematic of the components of the experiment

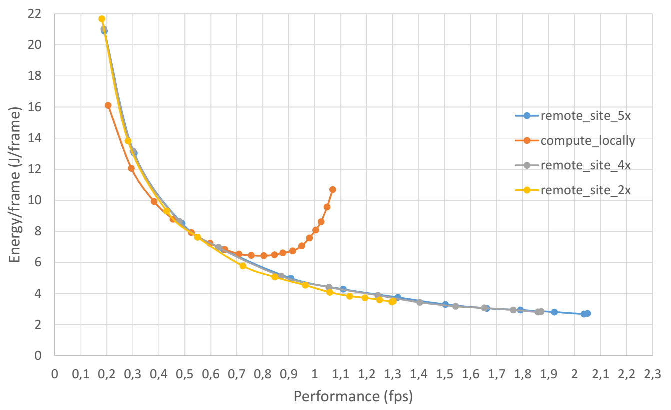

Fig. 5: Comparison of energy spent per frame of local computation versus remote computation when using SSD-MobileNet

paper, the smallest remote execution time is around 200 ms while the longest execution time is around 1100 ms. For the same set of used input pictures, the execution time of the object recognition task on the embedded mobile computing platforms range from 900 ms to 4 s. For setting the boundaries of our ROI, we use one GPIO pin of the board as a flag to measure the start and end the time of the ROI. For measuring the power dissipation, while choosing to offload the task, we use an oscilloscope with one probe recording the current drawn from the board and the other sensing the voltage on the GPIO pin, which is used as a flag for defining our ROI. The testbed is composed of several components as shown in Figure 4.

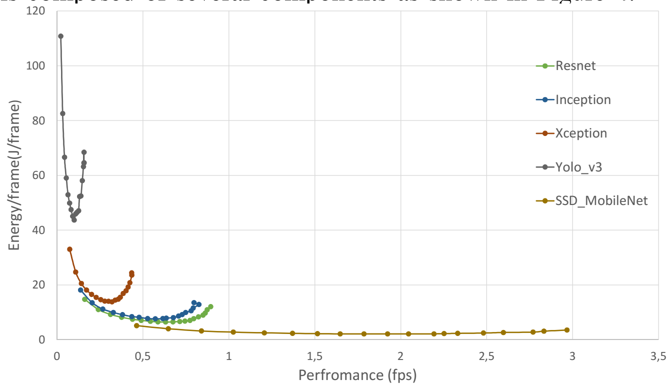

Fig. 6: Comparing the Energy Efficiency of different object detection algorithms.

As a remote side, we use a server that is on the same LAN as the Odroid board and listens for connections by the client. We set the server for different performance levels, such as 5x (five times faster than local side), 4x, and 2x. To measure the power dissipated by the entire board during the ROI we used an oscilloscope and a power meter, depending on the required sampling rate, with measurements logged at a resolution of up to 10 kHz.

The neural network used in the object detection and recognition application is a CAFFE implementation of an SSD meta-architecture with MobileNet [20] as a base network. We tested the change in the neural network used for our object detection. By using neural networks for object detection recognition in embedded environments, the most stringent limitation is the memory available on the device. With only 2 GB of memory, a few networks might be executed on the board. We analyzed three pre-trained base networks: Resnet-50, Inception and Xception implemented in Keras and trained with ImageNet dataset. We used YOLO v3 neural network which is implemented in the darknet framework and trained with COCO dataset.

In the Odroid board, we have different options for running computations related to a certain application. Having 8 computing cores divided into two cluster types, with each type of core being able to work in different performance levels (DVFS), opens possibilities for selecting the optimal energy-efficient way for running the application. We run the object detection application on a single image with all the possible configurations present in our experimental board and we ranked the energy efficiency of each configuration. By reviewing the results of the energy efficiency achieved for each configuration we discover that only the configurations using all four A15 cores, which provide high performance, are ”relevant” for this use case. The best scores in energy efficiency are only achieved by the configurations using all four A15 cores at the middle range frequency. We consider this configuration point as the one we will use in local computations, providing the highest energy efficiency at the platform level.

We executed the neural networks on a set of pictures composed of mix images with high and low resolution, large and small complexity in terms of the number of objects inside, and with different backgrounds. When offloading the detection and recognition task to the remote side, the input images were transmitted with bandwidths from 1 to 60 Mbps.

VII. Numerical Results

Figure 5 shows the average energy consumed to process, using SSD, an image with different execution speeds provided by the remote and local side. We can notice that when using offloading we reach similar energy efficiency for an achieved performance level, regardless of the performance of the remote side. The crossing with local computation is done from 0.5fps to 0.7fps, and for higher performance, the energy cost to perform the detection and recognition task locally is higher than using computational offloading.

We also measured the energy efficiency for processing an image by different types of neural networks. We used a mix of images for testing different neural networks and the results of the average energy consumed to process a frame are shown in Figure 6. As shown, there is a large difference in the results from each neural network: while we can notice that SSD-MobileNet is the most efficient one in terms of energy, it also provides the highest performance possible on the SBC. Next, we have three other models that provide nearly similar results of energy efficiency: Resnet, Inception and Xception with the first one performing better in terms of energy and achievable performance. Yolo is the one that is energy demanding and has the highest requirements in terms of computations. In [21] authors experiment with different base networks and report the results of the accuracy that the networks provide in terms of the mean Average Precision (mAP) score. They test different meta-architectures with different base models and the results show that SSD with MobileNet is overall the fastest one but provides the lowest accuracy. According to [22], Xception outperforms Inception and Resnet in terms of accuracy. On the other hand, Resnet is less accurate than Inception. Regarding YOLO, according to [23], this network is 20% to 30% more accurate than SSD-MobileNet, but from Figure 6 we notice that it is 170% to 180% less energy efficient. In comparison with Resnet, Yolo is more accurate as described in [24].

VIII. Conclusions and future work

In this paper, we discussed computation offloading as a possible mechanism to reach the optimal platform energy-efficient state considering the energy consumption of all components in the mobile system. As an application test case we considered object detection and recognition performed on several types of images to define when offloading is beneficial to the platform energy efficiency. We identified the configuration points, where the platform provides the maximum energy conservation approach. We concluded that if the bandwidth of the network connection is large enough, then the offloading strategy turns out to be more energy-efficient than the local computing. We surveyed the energy efficiency of different neural network algorithms in an embedded environment and concluded that not many neural networks for object detection can be handled by average embedded platforms. In some cases, to improve the accuracy between 20% to 30%, the cost in degrading energy efficiency is 170% to 180%.

As future work, we consider to extend investigations with more recent MPSoC like Kirin 960 and extend the discussion of platform efficiency by including the GPU for the neural network computations. We also plan to run experiments with GPUs ranging from the ones present in smartphone chips, like ARM Mali-G71 in Kirin 960, to NVIDIA GPUs present in Jetson-TX2 with 265 CUDA Pascal cores and NVIDIA Xavier with 512 CUDA Volta cores.

Acknowledgment

This work has been partially funded by the H2020 EU/TW joint action 5G-DIVE (Grant no. 859881).

References

[1] ericsson.com, "Ericsson mobility report," 2019. https://www.ericsson.com/4acd7e/assets/local/mobility-report/documents/2019/emr-november-2019.pdf.

[2] W. W. R. Forum, "Wireless of big data of smart 5g." https://www.wwrf.ch/files/wwrf/content/files/publications/outlook/White%20Paper%202-%20Wireless%20Big%20Data%20of%20Smart%205G.pdf.

[3] R. Schmoll, S. Pandi, P. J. Braun, and F. H. P. Fitzek, "Demonstration of vr / ar offloading to mobile edge cloud for low latency 5g gaming application," in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–3, Jan 2018.

[4] Y. Li, A.-C. Orgerie, I. Rodero, B. L. Amersho, M. Parashar, and J.-M. Menaud, "End-to-end energy models for Edge Cloud-based IoT platforms: Application to data stream analysis in IoT," *Future Generation Computer Systems*, vol. 87, pp. 667–678, Oct. 2018.

[5] E. Rotem, U. C. Weiser, A. Mendelson, R. Ginosar, E. Weissmann, and Y. Aizik, "H-EARtH: Heterogeneous Multicore Platform Energy Management," *Computer*, vol. 49, pp. 47–55, Oct. 2016.

[6] H. Hoffmann, "Racing and pacing to idle: An evaluation of heuristics for energy-aware resource allocation," in *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '13, (New York, NY, USA), Association for Computing Machinery, 2013.

[7] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi, "Using bandwidth data to make computation offloading decisions," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–8, April 2008.

[8] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *2007 International Conference on Parallel and Distributed Systems*, pp. 1–8, Dec 2007.

[9] P. A. Rego, P. B. Costa, E. F. Coutinho, L. S. Rocha, F. A. Trinta, and J. N. d. Souza, "Performing Computation Offloading on Multiple Platforms," *Comput. Commun.*, vol. 105, pp. 1–13, June 2017.

[10] K. Kumar, Y. Nimmagadda, and Y.-H. Lu, "Energy Conservation for Image Retrieval on Mobile Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 11, pp. 66:1–66:22, Sept. 2012.

[11] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-Efficient Offloading for Mobile Edge Computing in 5g Heterogeneous Networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[12] A. Bhattacharya and P. De, "A survey of adaptation techniques in computation offloading," *Journal of Network and Computer Applications*, vol. 78, pp. 97–115, Jan. 2017.

[13] M. Malik and H. Homayoun, "Big data on low power cores: Are low power embedded processors a good fit for the big data workloads?," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, (New York City, NY, USA), pp. 379–382, IEEE, Oct. 2015.

[14] S. Variable, "A multi-core cpu architecture for low power and high performance," *Whitepaper-http://www. nvidia. com*, 2011.

[15] H. Rexha, S. Holmbacka, and S. Lafond, "Core level utilization for achieving energy efficiency in heterogeneous systems," in *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pp. 401–407, March 2017.

[16] U. Gupta, C. A. Patil, G. Bhat, P. Mishra, and U. Ogras, "DyPO: Dynamic Pareto-optimal configuration selection for heterogeneous MpSoCs," *ACM Transactions on Embedded Computing Systems*, vol. 16, p. 123, Sept. 2017.

[17] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: Eliminating Server Idle Power," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, (New York, NY, USA), pp. 205–216, ACM, 2009.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016.

[19] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[21] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," 2016.

[22] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2016.

[23] D. Holz, K. Genter, M. Saad, and O. von Stryk, *RoboCup 2018: Robot World Cup XXII*. Lecture Notes in Computer Science, Springer International Publishing, 2019.

[24] J. Redmon, "Darknet-53 accuracy comparisons." https://pjreddie.com/darknet/imagenet/#darknet53_448. Accessed: 2020-01-17.