Asynchronous Hybrid Deep Learning (AHDL): A Deep Learning Based Resource Mapping in DVFS Enabled Mobile MPSoCs

Somdip Dey
Engineering & Research
Nosh Technologies
U.K.
dey@nosh.tech; somdip.dey@essex.ac.uk

Suman Saha
Engineering & Research
Nosh Technologies
U.K.
suman.saha@nosh.tech

Amit Singh
CSEE
University of Essex
U.K.
a.k.singh@essex.ac.uk

Klaus McDonald-Maier
CSEE
University of Essex
U.K.
kdm@essex.ac.uk

Abstract—**Mapping resources to tasks accurately in order to gain performance, energy efficiency, reduction in peak temperature, etc. on an embedded/Edge device is a big challenge. Machine learning has proven to be effective in learning heuristics based resource mapping approaches, but its success is bound by the quality of feature extraction. Additionally, feature extraction in such approaches not just requires expert domain knowledge and human effort, but at the same time requires the application (tasks) to be profiled for such processes. Therefore, the efficacy of such resource mapping methodologies depends on expertise, skills, profiling time and architecture of the system. In this paper, we propose a novel methodology, Asynchronous Hybrid Deep Learning (AHDL), which sets a new paradigm of using Deep Learning approaches to map resources to application (tasks). In our approach, we leverage task profiling methodologies to achieve accurate mapping in order to achieve greater reward from the system, but at the same time is able to allocate resources to unprofiled application (tasks) at the same time without the need of manual feature extraction by domain experts. Our proposed methodology is able to achieve competitive results in comparison with the state-of- the-art without the usual associated challenges such as manual feature extraction.**

Index Terms—**DVFS, MPSoC, asynchronous hybrid deep learning, resource mapping**

I. Introduction and Motivation

Recently we could see an increase in embedded/Edge computing systems [1], which employs heterogeneous multiprocessor systems-on-chips (MPSoCs) consisting of different types of processing cores such as CPU and GPU. These architectures provide opportunities to exploit distinct features of CPU and GPU cores to meet performance, power consumption, thermal behaviour and security requirements. Furthermore, the cores in these MPSoCs support dynamic voltage and frequency scaling (DVFS) [2]–[5], which can be used to reduce dynamic power consumption ($P \propto V^2 f$) [6], [7]. This helps to reduce the power consumption by executing the workload over extra time at a lower voltage and frequency, which could be accounted for reduced power consumption.

On the other hand, several studies, which are focused on extracting features from the source code of an application and then utilizing several machine learning models [8]–[13] such as Support Vector Machines (SVMs), Nearest Neighbor, etc. to classify different set of applications and then deciding the resources that need to be allocated to such applications. Using such methodologies also have their own disadvantages. Depending on feature extraction such as number of code blocks, branches, divergent instructions, and then utilizing machine learning on them usually requires accurate identification of features from the training data and then feeding them to the model. Extracting features from a source code of a program and then feeding to the machine learning model so that further inference could be made is difficult in many ways and often requires human intervention (domain experts). One of the key challenges in this area is to automate such feature extraction process from the source code such that appropriate computing resources could be allocated to the associated application to reduce power consumption and peak temperature of the device.

In this paper, we propose a novel method to allocate resources to application (tasks) without the need of manual feature extraction by domain experts. We leverage Deep Learning [14] approach to classify different types of applications and then allocate computing resources for the particular application to improve performance, power consumption, thermal behaviour and security of the device, as required. To this extent, the concrete contribution of this paper are as follows.

1) Propose a method for resource allocation to application (tasks) without the need of manual feature extraction by domain experts.
2) Experimental results on real hardware device (Odroid XU4 [15]).
3) Comparative study of the proposed approach with the state-of-the-art methods.

II. Related Work

In several earlier studies [16]–[20], many researchers have focused on designing methodologies and frameworks to optimize power and operating temperature of MPSoCs based on different approaches to computing resource mapping. One such noteworthy study is performed by Ghasemazar et al. [16] where the researchers proposed a hierarchical framework leveraging DVFS capabilities of the processing cores to find

This work was supported by Nosh Technologies under Grant nosh/agritech-000001 and by the U.K. Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/P017487/1, Grant EP/R02572X/1, Grant EP/P016006/1 and Grant EP/V000462/1.

the optimal voltage-frequency to cater for power consumption and temperature. This methodology was successful in achieving 20% performance boost without impacting the overall operating temperature but their experiments focused mainly on CISC architectures and all results were based on a MATLAB-based Chip Multiprocessor simulator. In another paper [17], Kamal et al. proposed a heuristic based thermal stress-aware mechanism for management of power and temperature in MPSoCs formulated in a convex optimization problem. This approach was implemented on Sniper multicore simulator [18] and was able to reduce spatial and temporal thermal gradients by 7% and 18% respectively when compared to the work in [16].

In [19] Sigla et al. present a predictor using power sensors to predict the next power consumption based on the following frequency setting is developed. Their technique uses a leakage power model of the ARM's big.LITTLE architecture on the Odroid-XU3 to validate its predictor and Dynamic Power and Frequency Management technique. An Extension of this work has also been published in [20]. Both [19], [20] methods involve predicting the future core temperatures to adjust the workloads or frequencies before exceeding a set threshold. In [21], Reddy et al. proposed a Memory Reads Per Instruction (MRPI) metric based computing resource mapping, whereas, in [2], Dey et al. proposed a methodology to optimize temperature by performing DVFS on CPUs based on the relationship between operating frequency of the processing cores and temperature of the MPSoC.

Moreover, none of the aforementioned approaches [16]–[20] optimize the power and thermal behaviour of the MPSoC by optimizing the computing resource as required by the type of executing applications. Executing applications could potentially fall into any of the following three categories [22]: Compute intensive, Memory intensive & Mixed workload, where the application is both Compute and Memory intensive. Additionally, executing application could also require different processing cores such as CPU and GPU at different times of the execution. The aforementioned approaches does not account for such changing demand in processing elements based on the type of the executing application.

The primary focus of this work is to design an approach to allocate computing resources to application/tasks by extracting features of executing applications automatically such that performance, power and temperature of the MPSoC could be optimized.

III. Proposed Methodology: Asynchronous Hybrid Deep Learning (AHDL)

To overcome the challenge of mapping computing resources (M) to new tasks or applications, which might or might not have been profiled or executed before, we propose an **Asynchronous Hybrid Deep Learning** (**AHDL**) resource mapping approach, which determines (classify) the type of executing application (tasks) without requiring manual feature extraction and then allocates the appropriate computing resources based on that. AHDL has two phases: *Learning Phase* and *Decision Phase*. To get more accurate resource mapping in order to achieve the most from the system we use offline profiling of the application instance (App_i). We refer our desired output requirement of the App_i on the system as *Reward* (R). Our goal is to maximize the *Reward* of App_i while minimizing the resources being allocated to App_i. *Reward* will vary depending on the application and the user's need. For example, if we are trying to map resources for a face detection application on the MPSoC where performance of the application is crucial in terms of *frames per second* (fps) for the recorded video then fps becomes the *Reward* in this case. Whereas in another example where we are trying to map resources to an instance of audio decoding application then execution time ET_{App_i} of the application becomes the *Reward*, which we need to prioritize to maximize. Therefore, if M is the resources that need to be allocated to the application App_i, PEs is the set of all available processing elements in the system, F_{PEs} is the set of all associated frequency scaling levels leveraging DVFS capabilities for all processing elements in PEs, the main objective of the AHDL resource mapping approach is as follows:

$$\forall M \in \{PEs, F_{PEs}\} : maximize(R) \propto minimize(M) \tag{1}$$

Note: For some cases, the main objective of AHDL resource mapping could be to minimize the value of reward (R) instead while minimizing the resources to be allocated. For example, if an application is utilizing data encryption algorithm then instead of focusing on performance the main objective should be to reduce the operating temperature of the processing elements in order to protect against temperature side-channel attacks. Hence, in this case the *Reward* will be temperature and the goal is to minimize R while minimizing the resources being allocated to the application App_i. Therefore, the main objective of the AHDL could be modified to represent as follows:

$$\forall M \in \{PEs, F_{PEs}\} : minimize(R) \propto minimize(M) \tag{2}$$

A. Overview of AHDL

For every instance of an application (App_i) from a set of applications ($Apps$), App_i is profiled in the *Regression Module* in *Training Phase*, where frequency of the processing elements (PEs) are modified along with the number of processing elements (PE) mapped to the task(s) to generate a set of *Rewards* ($S(R)$). From this set of *Rewards* ($S(R)$), the maximum of the set is chosen to reflect the maximized *Reward* and the associated mapped process elements, associated frequencies and other relationship variables are saved on the memory. These are the *Reward Profiled Data*, which will be used during the *Decision Phase* to map proper resources along with associated frequencies of the processing elements.

Back in the *Training Phase*, the source code of App_i is fed to *Imager Module*, which is a software agent that converts the source code into visual images such that the image (I_{App_i}) representing the source code of the application App_i could be used to train a pre-Trained convolutional neural network (CNN) to understand the difference between different set of applications. After the training of the CNN is complete, the trained CNN (*Coder CNN*) could be used to classify an application if that application is not profiled before. Using this approach we could classify new applications during runtime and map appropriate resources to this application based on our heuristics in the *Decision Phase*. More details on each phases (*Training & Decision*) and modules (*Regression & Visual Convolutional Neural Network*) are provided in the following subsections.

B. Training Phase

The *Training Phase* is performed completely offline due to the time required for profiling the applications (*Apps*). In this Phase we take an application instance App_i from a set of applications $Apps$ and pass it through a *Regression Module*. We call this as *Regression Module* because we use simple linear regression to evaluate and set the frequency levels of the processing elements in order to achieve the maximum *Reward*. But before utilizing this approach, the type of *Reward* is required to be finalized by the user based on the application. For example, if the application is a computer vision one then frames per second is chosen as the *Reward*, whereas for execution critical application the execution time is chosen as a *Reward*. Once the *Reward* is finalized, we profile App_i in the *Regression Module*.

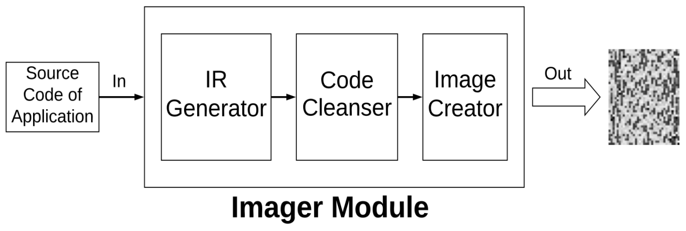


Fig. 1. Block digram of Imager Module

1) Regression Module: If we assume that a processing element PE_i has an operating frequency of f_{PE_i} and utilizing the concept of DVFS we could derive the governing equation (see Eq. 3) for our regression model as follows:

$$R = \alpha f_{PE_i} + \beta \tag{3}$$

In Eq. 3 α is the relationship variable whereas β is the intercept. Now, for MPSoCs which allow cluster wide DVFS capabilities with a cluster of processing elements having the same operating frequency ($f_{PE_{cluster}}$) instead of having individual operating frequencies (f_{PE_i}) for each PE, the equation could be written as follows.

$$R = \alpha f_{PE_{cluster}} + \beta \tag{4}$$

If we consider that F_{PEs} is the set of all the possible frequencies at which the processing elements could operate and $S(R)$ be the set of all associated *Reward*s for each operating frequency in F_{PEs} and $R_{desired}$ be the desired maximum *Reward* we want to achieve then we get the governing equation as follows:

$$\forall f_{PE_i}, f_{PE_{cluster}} \in F_{PEs} : R_{desired} = max\ S(R) \tag{5}$$

Therefore, using the Eq. 3, 4 and 5 we profile App_i and save the number of PEs, associated operating frequencies, α and β on the memory. We then repeat this approach to profile the same App_i for M number of times to achieve a *Reward* ($Reward_{desired_i}$) for each instance and use ensemble average approach [23] to get the desired *Reward* ($R^{App_i}_{desired}$) for the application. The mathematical representation of this could be fulfilled by Eq. 6.

$$R^{App_i}_{desired} = \frac{1}{M} \times \sum_{i=1}^{M} R_{desired_i} \tag{6}$$

The output of Eq. 6 (see Fig. 2) are saved as the *Reward Profiled Data*, which would be later used in the *Decision Phase*. Now, the program source code of App_i is fed to our *Imager Module* of *Visual Convolutional Neural Network Module*, which is the main software agent aiding in representing the program source code into an image so that it could be used by a visual CNN for automatic feature extraction method without requiring skilled manual feature extraction.

2) Visual Convolutional Neural Network Module: This module is responsible for converting program source code into images and train a chosen pre-Trained CNN using Transfer Learning such that the CNN is capable of extracting features automatically from images. This approach was first proposed in [22] by Dey et al.

Imager Module: Fig. 1 shows the inner working of the *Imager Module* and the algorithm is provided in Algo. 1. The *Imager Module* is a software agent where the program source code is fed and the agent first converts the source code to an optimized LLVM intermediate representation (IR) [24]. In the *Imager Module*, a sub module named *Code Cleanser* then acts on the LLVM IR to strip off all the unwanted codes such as metadata and comments and later processed by another sub module named *Image Creator* to be represented as an image (I_{App_i}). In the *Image Creator* sub module, each byte of cleansed LLVM IR is read as an integer value, which ranges from 0 to 255 (ASCII value). Each of these bytes are then represented as a pixel value in a $w \times h$ image where w is the width and h is the height of the image. If there are n number of bytes of cleansed LLVM IR code then n, w & h form the relationship of $n = w \times h$.

After the creation of image representation by *Imager Module*, I_{App_i} of several different applications (*Apps*) is then categorized into different classes so that the visual CNN could be trained for future classification and further decision making based on our heuristics. If we consider I_{Apps} as the dataset of images representing all the source codes of $Apps$ then we run the dataset I_{Apps} through the pre-Trained CNN and training a new, randomly initialized classifier on top of the semantic image output vector using the concept of *Transfer Learning in CNN* [25], [26]. After training the CNN with our own custom classifier we call the CNN as *Coder CNN*. **Note**: Categorizing each I_{App_i} would depend on the choice of the user and the object of maximizing the *Reward*. For example, based on the types of applications being profiled we segregated the image representation into 3 categories: compute intensive, memory intensive and mixed load (both compute

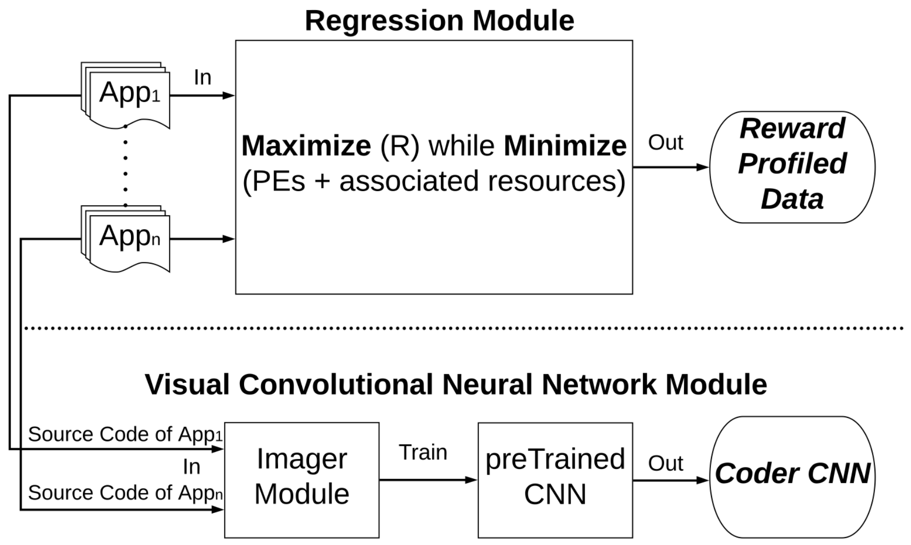


Fig. 2. Block digram of Training Phase of Asynchronous Hybrid Deep Learning Resource Mapping

Algorithm 1: Imager Module Execution

```
Input: $SC_{App_i}$: source code of $App_i$
Output: $I_{App_i}$: image representation of the source code of $App_i$
IR Generator:
$IR_{App_i}$ = GenerateLLVMIR($SC_{App_i}$);
  ▷ GenerateLLVMIR($SC_{App_i}$) is a function to generate the
  optimized LLVM IR code from the source code $SC_{App_i}$
  and returns IR code as $IR_{App_i}$
Code Cleanser:
$CIR_{App_i}$ = CleanIRCode($IR_{App_i}$);
  ▷ CleanIRCode($IR_{App_i}$) is a function to strip off unwanted
  parts from the IR code such as comments and meta-data
  and returns the cleansed IR code as $CIR_{App_i}$
Image Creator:
$n$ = TotalBytes($CIR_{App_i}$);
    ▷ TotalBytes($CIR_{App_i}$) is a function to return the total
  number of bytes in thecleansed IR code
$< w, h >$ = CalculateWidthHeight($n$);
    ▷ CalculateWidthHeight($n$) is a function to calculate the
  width ($w$) and height ($h$) of the image representation such
  that $n = w \times h$ is true and returns $w$ & $h$ as a tuple
$I$ = CreateAMatrix($w, h$);
▷ CreateAMatrix($w, h$) is a function to create a blank matrix
  with $w \times h \times 3$ dimensions to represent a RGB image
foreach Byte $b_i$ in $n$ bytes of $CIR_{App_i}$ do
    $i_i$ = Int($b_i$);
    ▷ Int($b_i$) is a function to return the integer value of byte
    $b_i$
    PaintImage($i_i$, $I$);
    ▷ PaintImage($i_i$) is a function to copy the value of $i_i$ at
    the position of $b_i$ in $I$ matrix for all 3 channels(R,G,B)
$I_{App_i}$ = SaveMatrixToImage($I$);
▷ SaveMatrixToImage($I$) is a function to save the matrix into
  an image format on the memory and returns the image
  $I_{App_i}$
return $I_{App_i}$;
```

and memory intensive) [22]. We chose these 3 categories because we allocate resources to new applications based on their computational capacities in the *Decision Phase*.

C. Decision Phase

The block diagram of *Decision Phase* is provided in Fig. 3. The knowledge gathered through profiling (*Reward Profiled Data & Coder CNN*) from the *Training Phase* is transfered on the MPSoC. *Decision Phase* is implemented as a software agent. When an application instance App_i is executed on the MPSoC, the agent first checks whether App_i is profiled in advance in the *Training Phase*. If no associated *Reward Profiled Data* on App_i is found then the source code of the application App_i is fed to the *Imager Module* to create the image representation I_{App_i} so that it could be classified by the *Coder CNN* and then allocation of resources take place based on the classification. But if the associated *Reward Profiled Data* on App_i is found then resources are allocated based on the *Reward Profiled Data*. We have to keep in mind that resource allocation through *Regression Module* of *Decision Phase* is more accurate because the application has been profiled in advance but the resource allocation achieved from the *Visual Convolutional Neural Network Module* in the *Decision Phase* is less accurate but has been able to produce competitive results to achieve maximum *Reward* when compared to the state-of-the-art (see Sec. IV for comparison) without requiring any manual feature selection of the application. **Note**: For *Visual Convolutional Neural Network Module* we are setting different resources (operating frequency and number of processing elements such as big and LITTLE CPUs) for different types of applications. The following resources are chosen for different types of applications for our experimental platform (Odroid XU4 [15]): compute intensive (all the big CPUs, all the LITTLE CPUs, maximum operating frequency of the big CPUs + LITTLE CPUs), memory intensive (half of the total number of LITTLE CPUs, maximum operating frequency of the LITTLE CPUs) and mixed load (half of the total number

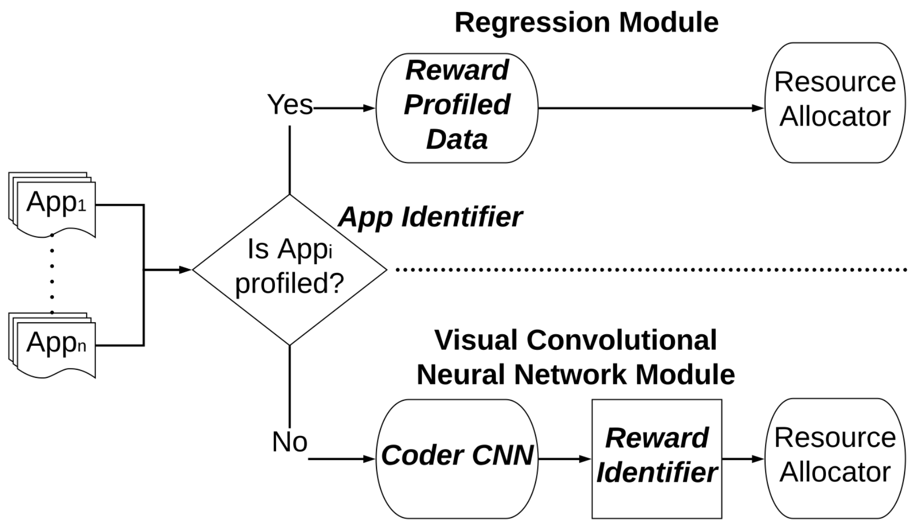
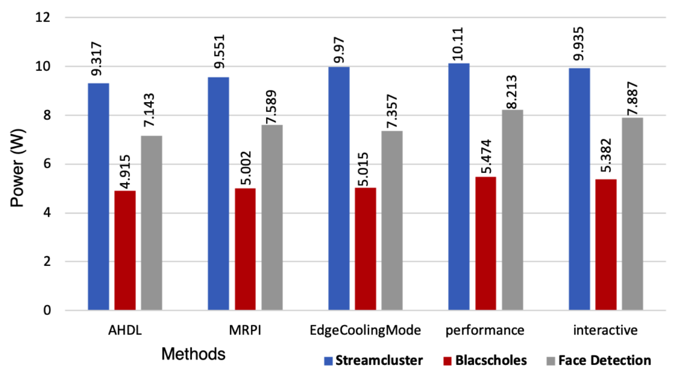
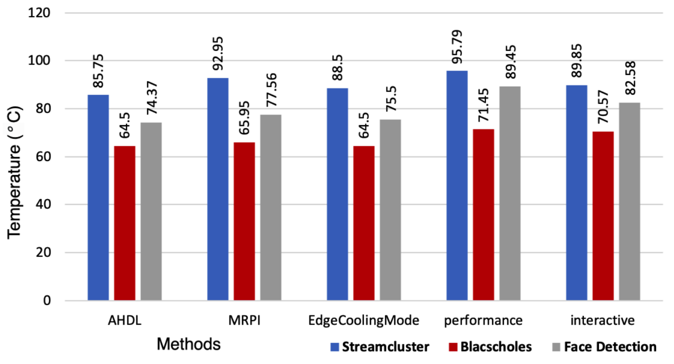


Fig. 3. Block digram of Decision Phase of Asynchronous Hybrid Deep Learning Resource Mapping

of the big CPUs, all the LITTLE CPUs, middle operating frequency of the big CPUs + LITTLE CPUs).



(a) Power consumption



(b) Peak temperature

Fig. 4. Comparative study of power consumption and peak temperature for different methods on different benchmark applications

IV. Experimental Results

A. Hardware & Software Infrastructure

1) Hardware Infrastructure: Nowadays heterogeneous MPSoCs consist of different types of cores, either having the same or different instruction set architecture (ISA). Moreover, the number of cores of each type of ISA can vary based on MPSoCs and are usually clustered if the types of cores are similar. For this research, we have chosen an Asymmetric Multicore Processors (AMPs) system-on-chip (AMPSoC), which is a special case of heterogeneous MPSoC and has clustered cores on the system. Our study was pursued on the Odroid XU4 board [15], which employs the Samsung Exynos 5422 [27] MPSoC. Exynos 5422 is based on ARM's big.LITTLE technology [28] and contains cluster of 4 ARM Cortex-A15 (big) CPU cores and another of 4 ARM Cortex-A7 (LITTLE) CPU cores, where each core implements the ARM v7A ISA. This MPSoC provides dynamic voltage frequency scaling feature per cluster, where the big core cluster has 19 frequency scaling levels, ranging from 200 MHz to 2000 MHz with each step of 100 MHz and the LITTLE cluster has 13 frequency scaling levels, ranging from 200 MHz to 1400 MHz, with each step of 100 MHz. Additionally, each core on the cluster has a private L1 instruction and data cache, and a L2 cache, which is shared across all the cores within a cluster.

Since Odroid XU4 board does not have an internal power sensor onboard, hence an external power monitor [29] with networking capabilities over WIFI is used to take power consumption readings. On the Odroid XU4 the temperature sensors are located on the big CPUs.

2) Software Infrastructure: For multi-core systems, multi-threaded applications are heavily used in recent times to represent workloads as they could leverage concurrency and parallel processing. Examples of such applications are available in several benchmarks such as PARSEC [30]. For our experiments we chose Streamcluster with $native$ option and Blackscholes with $native$ option from the PARSEC benchmarks' suit because these applications closely represented real-world mixed load applications. We also validated our approach for face detection using Haar-cascade [31] application [32]. For our experiments each benchmark applications were executed five times and the average power consumption and peak temperature of the big CPUs are reported in the results. For Streamcluster and Blackscholes the Reward (desired output) was chosen to be the least execution time, whereas, for the face detection application 30 FPS was chosen to be the Reward (desired output) of AHDL method. We have run all our experiments on UbuntuMate version 14.04 (Linux Odroid Kernel: 3.10.105).

B. Results and comparative study

We evaluated our AHDL approach with state-of-the-art methods such as Memory Reads Per Instruction (MRPI) by Reddy et al. [21] and EdgeCoolingMode by Dey et al. [2] along with default Linux governors such as performance and interactive. Fig. IV.(a) shows the comparative study for the average power consumption between the different methods including AHDL for the benchmark applications, whereas, Fig. IV.(b) shows the comparative study for the average peak temperature of the big CPUs for the different methods. From Fig. IV we can notice that AHDL outperforms the state-of-the-art methods and achieves reduced power consumption and peak temperature.

V. Conclusion

In this paper, we proposed Asynchronous Hybrid Deep Learning (AHDL) resource mapping approach, which classifies the type of executing application (tasks) without requiring manual feature extraction and then allocates the appropriate computing resources based on that to achieve reduced power consumption and peak temperature in embedded mobile devices. Experimental results on real hardware platform, Odroid XU4, proves the efficacy of the proposed approach while being competitively better than the state-of-the-art methods.

References

[1] S. Dey *et al.*, “Energy efficiency and reliability of computer vision applications on heterogeneous multi-processor systems-on-chips (mpsocs).”

[2] ——, “Edgecoolingmode: An agent based thermal management mechanism for dvfs enabled heterogeneous mpsocs,” in *VLSI Design and 18th International Conference on Embedded Systems, 2019. 32nd International Conference on*. IEEE, 2019.

[3] S. Dey, A. K. Singh, and K. D. McDonald-Maier, “P-edgecoolingmode: an agent-based performance aware thermal management unit for dvfs enabled heterogeneous mpsocs,” *IET Computers & Digital Techniques*, vol. 13, no. 6, pp. 514–523, 2019.

[4] S. Dey, A. K. Singh, X. Wang, and K. D. McDonald-Maier, “Deadpool: Performance deadline based frequency pooling and thermal management agent in dvfs enabled mpsocs,” in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE, 2019, pp. 190–195.

[5] A. K. Singh, S. Dey, K. R. Basireddy, K. McDonald-Maier, G. V. Merrett, and B. M. Al-Hashimi, “Dynamic energy and thermal management of multi-core mobile platforms: A survey,” *IEEE Design & Test*, 2020.

[6] S. Isuwa, S. Dey, A. K. Singh, and K. McDonald-Maier, “Teem: Online thermal-and energy-efficiency management on cpu-gpu mpsocs,” in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 438–443.

[7] S. Dey, A. K. Singh, S. Saha, X. Wang, and K. D. McDonald-Maier, “Rewardprofiler: A reward based design space profiler on dvfs enabled mpsocs,” in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE, 2019, pp. 210–220.

[8] B. Taylor, V. S. Marco, and Z. Wang, “Adaptive optimization for opencl programs on embedded heterogeneous systems,” *ACM SIGPLAN Notices*, vol. 52, no. 5, pp. 11–20, 2017.

[9] C. Cummins, P. Petoumenos, Z. Wang, and H. Leather, “End-to-end deep learning of optimization heuristics,” in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2017, pp. 219–232.

[10] S. Dey, S. Saha, A. Singh, and K. McDonald-Maier, “Fruitvegcnn: Power-and memory-efficient classification of fruits & vegetables using cnn in mobile mpsoc,” in *2020 IEEE 17th India Council International Conference (INDICON)*. IEEE, 2020, pp. 1–7.

[11] S. Dey, A. K. Singh, D. K. Prasad, and K. McDonald-Maier, “Temporal motionless analysis of video using cnn in mpsoc,” in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2020, pp. 73–76.

[12] S. Dey, A. K. Singh, D. K. Prasad, and K. D. Mcdonald-Maier, “Ironman: An approach to perform temporal motionless analysis of video using cnn in mpsoc,” *IEEE Access*, vol. 8, pp. 137 101–137 115, 2020.

[13] S. Dey *et al.*, “User interaction aware reinforcement learning for power and thermal efficiency of cpu-gpu mobile mpsocs,” in *2020 DATE*. IEEE, 2020.

[14] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.

[15] “Odroid-xu4,” https://goo.gl/KmHZRG, accessed: 2018-07-23.

[16] M. Ghasemazar, H. Goudarzi, and M. Pedram, “Robust imization of a chip multiprocessor’s performance under power and thermal constraints,” in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*. IEEE, 2012, pp. 108–114.

[17] M. Kamal, A. Iranfar, A. Afzali-Kusha, and M. Pedram, “A thermal stress-aware algorithm for power and temperature management of mpsocs,” in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 954–959.

[18] T. E. Carlson, W. Heirman, and L. Eeckhout, “Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations,” in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2011, pp. 52:1–52:12.

[19] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, “Predictive dynamic thermal and power management for heterogeneous mobile platforms,” in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 960–965.

[20] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras, “Algorithmic imization of thermal and power management for heterogeneous mobile platforms,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 544–557, 2018.

[21] B. K. Reddy *et al.*, “Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores,” *IEEE Transactions on Multiscale Computing Systems*, pp. 1–14, 2017.

[22] S. Dey, A. K. Singh, D. K. Prasad, and K. D. Mcdonald-Maier, “Socodecnn: Program source code for visual cnn classification using computer vision methodology,” *IEEE Access*, vol. 7, pp. 157 158–157 172, 2019.

[23] D. Opitz and R. Maclin, “Popular ensemble methods: An empirical study,” *Journal of artificial intelligence research*, vol. 11, pp. 169–198, 1999.

[24] K. Anand *et al.*, “A compiler-level intermediate representation based binary analysis and rewriting system,” in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013.

[25] S. J. Pan *et al.*, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[26] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[27] “Exynos 5 octa (5422),” https://www.samsung.com/exynos, accessed: 2018-07-23.

[28] “Arm big.little technology,” http://www.arm.com/, accessed: 2018-07-23.

[29] “Odroid smartpower2,” https://www.hardkernel.com/shop/smartpower2-with-15v-4a/, accessed: 2020-07-07.

[30] C. Bienia, “Benchmarking modern multiprocessors,” Ph.D. dissertation, Princeton University, January 2011.

[31] S. Soo, “Object detection using haar-cascade classifier,” *Institute of Computer Science, University of Tartu*, pp. 1–12, 2014.

[32] H. J. Curnow and B. A. Wichmann, “A synthetic benchmark,” *The Computer Journal*, vol. 19, no. 1, pp. 43–49, 1976.