A Machine Learning-Based Strategy for Efficient Resource Management of Video Encoding on Heterogeneous MPSoCs

Arman Iranfar, William Andrew Simon, Marina Zapater, David Atienza
Embedded Systems Laboratory (ESL), Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland
{arman.iranfar, william.simon, marina.zapater, david.atienza}@epfl.ch

***Abstract*—The design of new streaming systems is becoming a major area of research to deploy services targeted in the Internet-of-Things (IoT) era. In this context, the new High Efficiency Video Coding (HEVC) standard provides high efficiency and scalability of quality at the cost of increased computational complexity for edge nodes, which is a new challenge for the design of IoT systems. The usage of hardware acceleration in conjunction with general-purpose cores in Multiprocessor Systems-on-Chip (MPSoCs) is a promising solution to create heterogeneous computing systems to manage the complexity of real-time streaming for high-end IoT systems, achieving higher throughput and power efficiency when compared to conventional processors alone. Furthermore, Machine Learning (ML) provides a promising solution to efficiently use this next-generation of heterogeneous MPSoC designs that the EDA industry is developing by dynamically optimizing system performance under diverse requirements such as frame resolution, search area, operating frequency and stream allocation. In this work, we propose an ML-based approach for stream allocation and Dynamic Voltage and Frequency Scaling (DVFS) management on a heterogeneous MPSoC composed of ARM cores and FPGA fabric containing hardware accelerators for the motion estimation of HEVC encoding. Our experiments on a Zynq7000 SoC outline 20% higher throughput when compared to the state-of-the-art streaming systems for next-generation IoT devices.**

***Index Terms*—HEVC, resource management, machine learning, heterogeneous MPSoC.**

I. Introduction

Real-time video streaming and processing is one of the major services targeted by the Internet of Things (IoT) paradigm. Video streaming is expected to reach 80% of global traffic by 2019 [1], with services such as Netflix and YouTube accounting for over 50% of downstream traffic [2]. High Efficiency Video Encoding (HEVC) is a next-generation video coding standard that provides twice the compression of its predecessors for the same video quality. If properly managed, its unprecedented configurability enables matching the throughput/quality needs of the myriad of available IoT edge nodes. However, due to its increased complexity, to achieve real-time encoding, the use of heterogeneous Multiprocessor Systems-on-Chip (MPSoCs) brought by the EDA industry and, in particular, of hardware accelerators, is required.

In this context, a few works on the EDA community have considered hardware acceleration for HEVC encoding and, in particular, for Motion Estimation (ME), due to its high computational burden [3]. Among them, a high-level synthesis design flow that maps the intra-prediction block into a SoC-FPGA is presented in [4], while different design aspects of a heterogeneous multicore model are assessed in [5]. Nonetheless, none of these works take into account inter-picture prediction as one of the most demanding, yet essential, stages of the encoder. Although Paul *et al* [6] investigate the advantages of heterogeneous MPSoCs for accelerating different stages within an image processing algorithm, they do not address the HEVC computational complexity.

However, variations in video features (e.g., resolution) and contents (e.g., motion) pose a challenge to efficient resource management (RM) of heterogeneous MPSoCs, making hardware acceleration by itself not sufficient to provide the efficiency (in terms of throughput and power) required to enable real-time video streaming. Therefore, static hardware acceleration needs to be combined with heterogeneity-aware runtime RM, with the goal of serving as many users (i.e., encoding as many videos) as possible, with a given quality and under a certain power cap. In such scenarios, the large variety of devices requiring different video configurations, together with the high workload variation in terms of number of requests, makes Machine Learning (ML), and in particular reinforcement learning, a promising approach to deal with such large environment-dependent problems [7].

In this work, we propose an ML-based RM strategy for heterogeneous MPSoCs that allocates different video streams (in terms of resolution and motion) to general-purpose cores and hardware accelerators (IPs), while setting the frequency of both. In particular, we consider a heterogeneous MPSoC composed of ARM cores and an FPGA fabric on which we implement several IPs of the ME algorithm. Our goal is to maximize the number of streams being processed at the same time and their total throughput while satisfying a minimum throughput per-stream and a user-defined power constraint. Therefore, the ML learns to properly allocate each input stream to an ARM core based on the frame resolution, the motion (which is mainly driven by search area), and the cores' available capacity while adjusting the operating frequency of the hardware accelerators (i.e, IPs) and ARM cores. Fig. 1 shows an overview of our approach.

The main contributions of this paper are as follows:

- We propose a ML-based approach for multistream HEVC encoding on heterogeneous MPSoCs that learns from per-stream throughput and total power consumption based on resolution and motion. The ML agent selects the best frequency per IP and core, and the best allocation to

This work has been partially supported by the EC H2020 MANGO project (GA No. 671668), and the ERC Consolidator Grant COMPUSAPIEN (GA No. 725657).

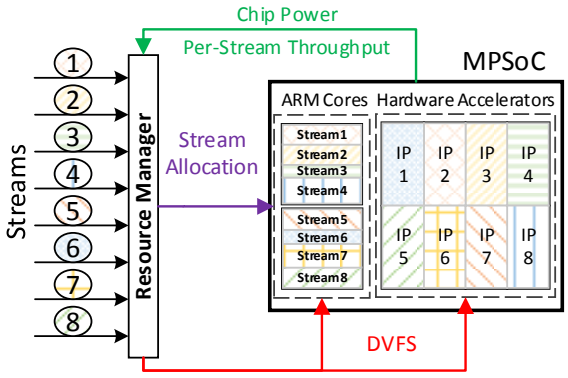


Fig. 1. Overall view of the proposed ML-based resource management approach.

available cores, maximizing the per-stream throughput and the total number of streams that can be processed simultaneously under a user-defined power constraint.

- We show how our ML-based approach is able to achieve 20% higher throughput without any power violation when compared to other state-of-the-art techniques.
- Upon any change in the number of streams or their resolution and motion, our approach reaches its maximized throughput 1.5x faster than the state-of-the-art techniques.

II. Proposed Method

Efficient heterogeneity-aware RM for HEVC encoding in MPSoCs requires tackling application configuration, stream allocation and DVFS for both general-purpose cores and IPs. This requires exploring a very large and dynamic design space. On one hand, streams have different inherent features, such as frame resolution, and need specific encoding configurations [8], such as search area (SA), which drives motion estimation, that make the workload and resource demand vary from one stream to the other. On the other hand, among the different combinations of streams that could potentially be processed at the same time, there could be many sub-optimalities in regards to core allocation and core and IP frequencies. An exhaustive search in the design space is required to avoid such sub-optimalities. However, conventional offline static approaches cannot guarantee handling the dynamic changes in the environment (e.g., when a new configuration is needed).

Therefore, in this work, we propose machine learning (ML) and, in particular, reinforcement learning as a promising solution to deal with such a dynamism in the design space at runtime [7]. The goal is to learn the best allocation of streams to cores, as well as the operating frequency of each core and IP, from the total power consumption and the output throughput, for each SA and resolution combination.

In particular, we leverage the Q-learning algorithm which is composed of a finite action set, **A**, a finite state space, **S**, and an agent. The agent acts according to a learned policy,π, which is a mapping from the state space to the action set while taking into account the reward value granted to each state-action pair. This value implies whether, given a state, an action is worth applying. The Q-Learning agent maximizes this reward by storing a $Q^{\pi}(s,a)$ value to represent the quality of each state-action pair in a Q-table. This value demonstrates the most probable long-term reward, considering starting from state, s, applying action a, and following the policy π. Note that the Q-table does not need to keep values of all theoretically defined state-action pairs, but only those that are observed at runtime. The Q-values and the Q-table are updated as follows [7]:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha_t(s_t,a_t) \times [R_{t+1} + \gamma \max Q_{t+1}(s_t,a) - Q_t(s_t,a_t)] \quad (1)$$

where $Q_t(s_t,a_t)$ and $Q_{t+1}(s_t,a_t)$ are, respectively, the current and updated Q-values corresponding to a_t action and s_t state, R_{t+1} is the reward observed after a_t is applied for state s_t, $\alpha_t(s_t,a_t)$ determines the learning rate, and γ is the discount factor and controls the significance of the history of the Q-values against the recently obtained reward.

Our proposed ML-based approach consists of two phases. In the *exploration* phase, once the learning process starts, at each observed state, the ML agent takes a random action from an action pool and calculates the reward, updating the Q-table by Eq. 1. Since we aim at learning per-stream SA and resolution, we need to create and keep one Q-table for each resolution/SA pair. The exploration phase for each state-action pair (s_t,a_t) continues until the corresponding learning rate, which is defined as:

$$\alpha_t(s_t,a_t) = \lambda / Num(s_t,a_t), \quad (2)$$

drops below a threshold. In this formulation, $Num(s_t,a_t)$ is the number of observations of the state-action pair, and λ is a constant [9]. Afterwards, the *exploitation* phase begins, where the ML agent stops updating the Q-tables and selects the most appropriate action for each observed state.

In what follows, we describe the state space, the action set, and the proposed reward functions.

A. States

Since the goal of this work is power- and throughput-aware RM of heterogeneous MPSoCs, the state space is defined as:

$$\mathbf{S} = \{P_{total}, \boldsymbol{Th}\} \quad (3)$$

where P_{total} is the total power consumption, and $\boldsymbol{Th}$ is a vector of throughputs for each running stream.

B. Actions

The proposed action set includes adding a stream, removing a stream, increasing/decreasing the frequency of an IP, and increasing/decreasing the frequency of an ARM core:

$$\boldsymbol{A} = \{Str_+, Str_-, f_{IP,inc}, f_{IP,dec}, f_{ARM,inc}, f_{ARM,dec}\} \quad (4)$$

While adding a new stream to the existing running streams may lead to a higher total throughput, removing a stream is not desirable. In other words, once an encoding request is accepted, the encoding process must be guaranteed to complete within a certain time. However, we introduce this action since it might be required to reduce power consumption.

At each decision step, only one action must be taken. Therefore, we can only change the frequency of one (and only one) IP or core. When the action is $f_{IP,dec}(f_{IP,inc})$ or $f_{ARM,dec}(f_{ARM,inc})$, both in exploration or exploitation, we let the agent apply the frequency change only for the stream with the highest(lowest) throughput.

C. Reward Function

The reward function must provide useful feedback about the selected action for a previous state. Since we are minimizing power consumption and maximizing performance, we propose a reward function composed of two sub-functions, as follows:

$$r_{tot} = c_1 r_{perf} + c_2 r_{power} \tag{5}$$

where r_{perf} and r_{pow} are the reward functions for performance and power, respectively. In this work, we consider equal significance coefficients (c_i) for both rewards.

We define r_{perf} to encourage the ML agent to choose actions leading to higher performance while avoiding those resulting in any performance loss:

$$r_{perf} = \begin{cases} N_{str}^{\beta} \sum_{i=1}^{N_{str}} Th_i / Th_{ref,i} & \forall i \quad Th_{ref,i} < Th_i \\ -1 & \exists i \quad Th_i < Th_{ref,i} \end{cases} \tag{6}$$

where N_{str} is the total number of streams being processed, and $Th_{ref,i}$ is the reference throughput (frame per second, FPS) for the i^{th} stream. This reference throughput is obtained based on the maximum throughput achievable on an Intel server at its maximum frequency, as explained in Section III. We experimentally prove that the following inequality holds for all SAs and resolutions given the maximum frequencies for IPs and ARM cores, as specified in Sec. III:

$$1 \leq Th_i / Th_{ref,i} < 15 \tag{7}$$

This inequality indicates that on our specific heterogeneous MPSoC, the achieved throughput for an individual stream is always greater than the reference one, but not larger than 15 times. However, when considering multiple streams at the same time, the cumulative throughput does not increase beyond 40 FPS. Therefore, since the goal in (7) is to first maximize the number of served streams while meeting their minimum reference throughput, we choose β equal to 5.3 which satisfies the worst case scenario. This value leads the ML agent to seek for increasing the number of concurrent streams, while guaranteeing the minimum required throughput of each individual stream. After fully occupying all available resources, the ML agent takes actions to increase the throughput of each individual stream.

In order to keep power consumption under a user-defined constraint (P_{const}) we propose the following reward function:

$$r_{power} = \begin{cases} 0 & P < P_{const} \\ -2.5 \times 10^6 & P > P_{const} \end{cases} \tag{8}$$

TABLE I
THROUGHPUT (ENCODED FRAMES PER SECOND) FOR DIFFERENT RESOLUTIONS AND SEARCH AREAS, INTEL PROCESSOR

Resolution	Search Area				
	4	6	8	10	12
704x576	0.62	0.31	0.19	0.13	0.09
1280x720	0.26	0.13	0.08	0.05	0.04
1920x1080	0.11	0.066	0.03	0.02	0.02

On one hand, reducing the power consumption below the constraint should not give a positive reward, since it ultimately results in lower throughput. On the other hand, any violation of the power constraint must add a large enough negative value (here, -2.5×10^6 because the maximum value of the performance reward is always less than $8^{\beta} \times 40$) to the total reward function, so that the corresponding action is avoided.

III. EXPERIMENTAL SETUP

In this work, experiments are performed on the Xilinx ZC-706 equipped with a Zynq7000 SoC, type Z-7045 [10]. The chip comprises a dual core Cortex-A9 ARM processor with a maximum frequency of 1 GHz. The chip also contains FPGA fabric consisting of 350K logic blocks and 19.2 Mb of BRAM. The board comes with a 1GB DDR3 RAM chip clocked at 533 MHz, and a 8GB SD card as primary storage.

Our reference throughputs are calculated on an Intel E5-2690 v4 server [11]. These values represent the highest throughputs achievable on a high-performance homogeneous platform, and ultimately show the gains of our ML-based approach running on a low-power heterogeneous SoC. This server contains 14 cores with a maximum clock speed of 3.5 GHz, 28MB LLC, and 250GB of memory. When calculating application throughput we tie the application to one core to prevent OS interference. TABLE I contains our reference throughput in frames per second, sorted by frame resolution and SA.

Our system utilizes FPGA-mapped accelerators (IPs) to speed up motion estimation (ME), which allows HEVC encoding to be performed on the much smaller and energy efficient ARM core, while maintaining comparable or improved throughput compared to the Intel processor [12] at a much lower power consumption. We implement 8 such IPs on the FPGA fabric, allowing up to 8 encoding applications to run simultaneously on the FPGA. These applications will share the 2 ARM cores. The IPs can be individually clocked to 50, 100, 150, or 200 MHz, and the two ARM cores can be individually clocked to 333, 666, 800, or 1000 MHz.

Some simplifications are made in our setup to ease experimentation. We assume the 3 resolutions listed in TABLE I and we limit SA to even values between 4 and 12. This already gives us a range of over 1.4 trillion combinations. We also assume that there are always streams queued to be added to the system, meaning the ML approach can add a new stream as an action whenever it is necessary.

We have profiled each combination of SA, resolution, and ARM and IP frequency. This profiling data is used to estimate

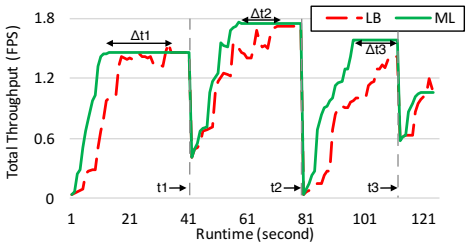


Fig. 2. Total throughput of the proposed ML vs. LB over time.

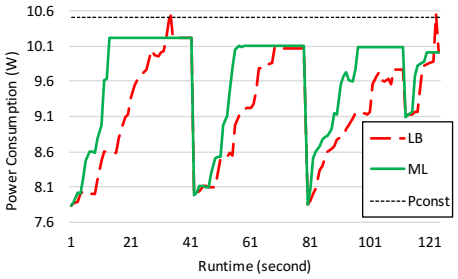


Fig. 3. Power consumption of the proposed ML vs. LB over time.

the power and throughput of any number of streams running on any combination of IP or ARM frequencies. However, the manner of this estimation is beyond the scope of this paper.

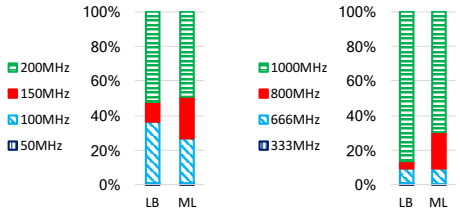
We stress the system by randomly changing the number of streams running, which represents the effect of users that enter/leave the system. We apply the stress at random intervals ranging from 30 to 40 seconds based on the statistics reported by Delmondo[1] (a social video analytics company). Each new stream has its own resolution and SA. In order to provide reliable results, we apply these stress points 10^5 times.

For comparison, we implement a workload balancing strategy for stream allocation on the ARM cores. The frequency of all IPs is set using a heuristic. In particular, at each decision step we increase by one step the frequency of the IPs, starting from the minimum, if $P < P_{const}$, to achieve higher throughput.

IV. Results and Discussion

In a typical video streaming IoT deployment, servers need to meet the demands of edge nodes by initiating/terminating encoding requests, and responding to dynamic changes is of paramount importance. Our ML-based approach improves average throughput by 20% over the load balancing (LB) algorithm when considering the whole runtime (including stress points), as shown in Fig. 2. Our ML approach demonstrates greater robustness against system dynamism as seen at the 40, 78, and 112 second marks (shown by t_1, t_2, and t_3), where new streams with new requirements replace older streams. On the contrary, the LB algorithm requires more time to maximize throughput when faced to system changes (shown by Δt_1, Δt_2, Δt_3). The reason lies in the fact that the ML agent has learned the optimal actions that maximize throughput for different combinations of SAs and resolutions, while LB first optimizes

[1]http://delmondo.co



(a) IPs (b) ARM cores

Fig. 4. Percentage of time that each frequency is used in ML and LB.

the minimum throughput of each stream by scaling up the frequency from the initial value, and then adds more streams to increase total throughput.

In the absence of such stress, both systems will eventually reach a steady state in which throughput cannot be further increased. At this point, the total throughput obtained by ML and LB are similar, with ML achieving 7% higher throughput on average. This is because ML is aware of corner cases, which cannot be resolved through heuristics.

Then, Fig. 3 shows the power consumption of the system for LB and ML, corresponding to the total throughput in Fig. 2. Our approach is able to optimally use the available power budget and increase the total throughput. In addition, since the ML has learned optimal actions for a power state, the power consumption never violates P_{const}. On the contrary, as the LB algorithm always optimizes for higher throughput and cannot predict power consumption, it may perform an action resulting in power constraint violation. This occurs at time intervals of 34, and 123 seconds shown in Fig. 3. On average, the power violation occurs once every 100 seconds for the LB algorithm.

Finally, Fig. 4 shows the percentage of the time that each frequency is used for the ARM cores and IPs for both the ML and LB approaches. Although the maximum frequency is selected most of the time in both cases, it is not always the optimal choice for all combinations of SA, resolution, number of running streams, power consumption, and per-thread and total throughput. These cases can be distinguished by ML, resulting in serving more streams and increasing the total throughput at lower frequencies, not only after each stress point, but also when no stream is coming into or leaving the system.

V. Conclusion

Real-time video streaming services in the IoT era require efficient resource management of heterogeneous MPSoCs. Given the large number of combinations possible during HEVC encoding, machine learning (ML) is a good candidate to address this challenge. In this work, we have proposed a ML approach that maximizes throughput while meeting a user-defined power constraint by learning optimum stream allocation as well as optimum core and IP frequency from the total throughput and power consumption of the system when encoding videos with different resolutions and motions. Our methodology achieves 20% higher throughput, and converges 1.5x faster to the optimal solution than traditional strategies.

REFERENCES

[1] Cisco Systems, Inc., "Cisco visual networking index: Forecast and methodology 2015-2020. cisco whitepaper." 2016.

[2] Sandvine, Inc., "Global internet phenomena report," 2013.

[3] Y. Lu, W. Cheng, L. Huang, X. Zeng, and Y. Fan, "A flexible HEVC intra mode decision hardware for 8kx4k real time encoder," in *ASIC (ASICON), 2015 IEEE 11th Int. Conf. on*. IEEE, 2015, pp. 1–4.

[4] P. Sjövall, J. Virtanen, J. Vanne, and T. D. Hämäläinen, "High-level synthesis design flow for HEVC intra encoder on SoC-FPGA," in *DSD, 2015 Euromicro Conf. on*. IEEE, 2015, pp. 49–56.

[5] J. Brandenburg and B. Stabernack, "Simulation-based hw/sw co-exploration of the concurrent execution of HEVC intra encoding algorithms for heterogeneous multi-core architectures," *Journal of Systems Architecture*, vol. 77, pp. 26–42, 2017.

[6] J. Paul, W. Stechele, B. Oechslein, C. Erhardt, J. Schedel, D. Lohmann, W. Schröder-Preikschat, M. Kröhnert, T. Asfour, É. Sousa *et al.*, "Resource-awareness on heterogeneous MPSoCs for image processing," *Journal of Systems Architecture*, vol. 61, no. 10, pp. 668–680, 2015.

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[8] A. Iranfar, M. Zapater, and D. Atienza, "Work-in-progress: a machine learning-based approach for power and thermal management of next-generation video coding on MPSoCs," in *CODES+ISSS*, 2017, pp. 1–2.

[9] A. Iranfar, S. N. Shahsavani, M. Kamal, and A. Afzali-Kusha, "A heuristic machine learning-based algorithm for power and thermal management of heterogeneous MPSoCs," in *ISLPED, 2015 IEEE/ACM Int. Symp. on*. IEEE, 2015, pp. 291–296.

[10] [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html

[11] [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-e5-v4-datasheet-vol-1.pdf

[12] A. Iranfar, F. Terraneo, W. A. Simon, L. Dragić, and I. P. et al., "Thermal characterization of next-generation workloads on heterogeneous MPSoC," in *SAMOS XVII.*, 2017.