UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CIÊNCIAS DA COMPUTAÇÃO

José Luiz de Souza

Evaluation of a machine learning model to predict the performance of energy-aware, real-time, multicore embedded systems across multiple architectures

Florianópolis
2023

José Luiz de Souza

Evaluation of a machine learning model to predict the performance of energy-aware, real-time, multicore embedded systems across multiple architectures

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em CIÊNCIAS DA COMPUTAÇÃO do CENTRO TECNOLÓGICO da Universidade Federal de Santa Catarina como requisito para obtenção do título de Bacharel em Ciência da Computação.
Supervisor: Prof. Antonio Augusto Medeiros Frohlich, Dr.

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Souza, José Luiz de
   Evaluation of a machine learning model to predict the performance of energy-aware, real-time, multicore embedded systems across multiple architectures / José Luiz de Souza ; orientador, Antonio Augusto Medeiros Frohlich, 2023.
   77 p.

   Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2023.

   Inclui referências.

   1. Ciências da Computação. 2. Sistemas Embarcado. 3. Otimização Energética. 4. Arquite-tura de Computadores. 5. Contadores de Desempenho. I. Frohlich, Antonio Augusto Medeiros. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

José Luiz de Souza

Evaluation of a machine learning model to predict the performance of energy-aware, real-time, multicore embedded systems across multiple architectures

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo curso de Graduação em CIÊNCIAS DA COMPUTAÇÃO.

Florianópolis, 2023.

ACKNOWLEDGEMENTS

The completion of this study could not have been possible without the expertise and help of Dr. Frohlich, Hoffmann and Horstmann. I also like to thank Dr. Giovani for taking time to read this study.

A debt of gratitude is also owed to my debt on the University Library that restrain me for locking the course.

Lastly, but most importantly, I want to thanks to my parents, Pedro and Rosinete, for prevented me from dropping out of the course by threatening me, my brother who kept asking me to play terraria while I was doing this study and also want to thank my friends, especially Samuel, for accompanying me during this journey.

Many are kept busy either striving after other people's wealth or complaining about their own. Many who have no consistent goal in life are thrown from one new design to another by a fickleness that is shifting, never settled and ever dissatisfied with itself. Some have no goal at all toward which to steer their course, but death takes them by surprise as they gape and yawn. I cannot therefore doubt the truth of that seemingly oracular utterance of the greatest of poets: "Scant is the part of life in which we live." All the rest of existence is not living but merely time.

Lucius Annaeus Seneca

RESUMO

Arquiteturas de computadores modernas estão cada vez mais heterogêneas. Essa heterogeneidade dá a arquitetura a capacidade de aumentar seu desempenho geral ou controlar o consumo energético. Esse controle energético no que lhe concerne se dá por técnicas como Dynamic Voltage and Frequency Scaling (DVSF), que reduzem a voltagem e frequência do processador baseada em certos requisitos. Com isso, algumas técnicas de Machine Learning estão sendo utilizadas para otimização energética, em que utilizam informações providas pela arquitetura para dar suporte a otimização. Este trabalho utiliza como base uma rede neural que utiliza informações obtidas em tempo de execução, como contadores de desempenho, sensores e variáveis do sistema operacional para realização de otimizações energéticas no sistema. A rede neural citada é testada em apenas uma plataforma, não possuindo resultados sobre o comportamento, desempenho da rede neural e sobre as escolhas de contadores em outras plataformas, visto que uma série de contadores de desempenho são dependentes da arquitetura. Neste contexto, este trabalho se propõe a testar esta solução nas arquiteturas ARMv7 e ARMv8, avaliando o modelo de rede neural e as escolha dos contadores de desempenho.

Palavras-chave: Sistemas Embarcados. Machine Learning. Otimização Energética. Arquitetura de Computadores. Contadores de Desempenho.

ABSTRACT

Modern computer architectures are becoming more heterogeneous. This heterogeneity gives the architecture the capacity to increase the overall performance or control the energetic consumption. This energetic control uses techniques like Dynamic Voltage and Frequency Scaling (DVSF) that reduce the voltage and frequency of the processor based on some requisites. With this, some Machine Learning techniques are being used for energy optimization, using information provided by the architecture to give support to the optimization. This work uses as a base an Artificial Neural Network (ANN) that uses information that is built at runtime out of hardware performance counters, sensors, and OS variables to perform the energy optimization on the system. However, the aforementioned ANN was only tested in one platform, not having results on the behavior and performance of the Machine learning model and on the choices of counters on other platforms, since some performance counters are architecture dependent. In this context, the objective of this work is to test the ANN solution on the ARMv7 and ARMv8 architectures, evaluating the model and the chose of the hardware performance counters.

Keywords: Embedded Systems. Machine Learning. Energy-Aware. Computer Architecture. Hardware Performance Counters.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Modern computer architectures are becoming more heterogeneous (ZAHRAM 2016) [14]. This heterogeneity takes place through a wide variety of different processors inside the same hardware, where they can differ from each other through the use, technology or energy management techniques. This gives the architecture the capacity to increase the overall performance, through the use of features like SIMD units and application-specific accelerators or technologies like hyper-threading. All of this is usually connected by some network-on-chip (ALI 2018) [4].

Along with the growth of the complexity of these systems, the power consumption grows as well, making energy management an important topic to deal with. The architecture can use some energy management techniques like Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management(DPM) to decrease the energy consumption of chips, scaling down the voltage and frequency based on the targeted performance requirements of the running application [2].

These techniques can be used to reduce the energy consumption in real-time based systems, where many of them are battery-operated embedded devices. Real-time based systems are designed to execute applications and respond to events within a predictable time, and therefore, using energy management techniques in these systems mean to reduce the energy consumption as much as possible without interfere on the real-time system applications constraints [13].

To provide runtime energy optimization, some Machine Learning (ML) techniques are being used. The ML use traces provided by the architecture and process them into useful information to support the optimization. However, ML-based optimization in runtime can have problems dealing with real-time base systems because the ML-techniques could end up violating the performance and temporal constraints of the real-time applications.

(HOFFMANN 2021) [5] presents a non-intrusive, ANN-based runtime energy optimizer that was designed and implemented considering the application's constraints. The model is able to predict the performance of tasks in lower frequency levels and safely optimize real-time embedded systems' power saving operations. This is done by taking execution traces that are built at runtime out of hardware performance counters, sensors, and OS variables of individual tasks as input to predict their performance in case the core where they are running is subjected to DVFS.

While the aforementioned work predicts and safely optimizes power saving operations, there is an assumption about the chosen hardware performance counters used as input by the ANN. In [5], the performance counters were analyzed and selected through three different techniques to filter and acquire only the more relevant counters that exist on the target platform, that was a Cortex-A53 processor, running in an ARMv7 mode where there are in total of 54 available performance counters.

However, only the above-mentioned platform was used for testing and there is no guarantee that the chosen performance counters remain the most relevant option on other platforms nor that the feature selection methods manage to capture the best performance counters due to the possible unpredictability that exists in the counters. In fact, not all hardware performance counters remain consistent across multiple architectures. Some performance counters are architecture dependent and vary across different processor architectures and may also change with processor enhancement (e.g. cache and TLB accesses) (SANJEEV 2019) [10].

Therefore, the objective of this work is to evaluate the performance of the (HOFFMANN 2021) [5] machine learning model running on other platforms and verify the reliability of feature selection methods when tested on other architectures. This will be done on the architectures ARMv7 and ARMv8, mediated by the Embedded Parallel Operating System (EPOS - https://epos.lisha.ufsc.br/).

1.1 OBJECTIVE

1.1.1 General Objective

Evaluate a machine learning model to predict the performance of energy-aware, real-time, multicore embedded systems and verify the reliability of feature selection methods across multiple architectures.

1.1.2 Specific Objectives

- Replicate the machine learning model on the original architecture.
- Identify the best performance counters on the additional target architectures through the feature selection.
- Analyze the performance of the machine learning model on the additional target machines.
- Draw conclusions or locate interesting results from the data.

1.2 METHODOLOGY

With the objective of replicating and simulating the ML-model across multiple architectures, the execution of this work was as follows:

1. Study the ARMv8 architecture, as well the ARMv7 architecture used in the (HOFFMANN 2021)[5] paper.
2. Replicate the machine learning model on the ARMv7 architecture with the (HOFFMANN 2021)[5] selected performance counters.

3. Implement the necessary code to run the machine learning model on the target architectures.
4. Execute a feature selection on the target architectures to obtain the best performance counters.
5. Execute the machine learning model on the target architectures using the performance counters obtained on the feature selection.
6. Evaluate the performance of the machine learning model on the target architectures and compare it with the ARMv7 replication.

1.3 ORGANIZATION OF THE WORK

The remainder of this work is organized as follows. In chapter 2 we present a background on real-time systems, performance monitor unit, computer architectures, machine learning and feature selection methods. In chapter 3 we discuss the principal related work. In chapter 4 we detail the experiment environment, the data collecting and the data analysis. In chapter 5 we detail the results of the experiments on each target architecture. Finally, in chapter 6 we draw our conclusions and future works.

2 BACKGROUND

This chapter introduces fundamentals concepts and terms that are used throughout this work.

2.1 EPOS

Embedded Parallel Operating System [8], or simply EPOS, is an operating system project developed at the Software/Hardware Integration Lab (LISHA) [9] and aims to automate the development of embedded systems.

EPOS relies on the Application-Driven Embedded System Design (ADESD) [7] method that use a strategy to systematically construct application-oriented operating systems as arrangements of adaptable software components. This feature enables the development of software/hardware components that can be automatically adapted to fulfill the requirements of particular applications.

Furthermore, EPOS has support for many architectures and platforms, being able to run in both of the target architectures used through this work, ARMv7 and ARMv8. For each architecture supported, some architecture dependent components are available too, like the Memory Management Unit and Performance Monitor Unit.

2.2 PERFORMANCE MONITOR UNIT

Performance Monitor Unit (PMU) is an architectural component available in the most of the modern processors that allow the capture and record of events that occur in the processor. The monitoring of these events can be used to evaluate how the code interacts with the hardware, what resources they used and how many clock cycles the code took to execute, helping to identify, for instance, performance bottlenecks and opportunities to speed up the application [3].

Because of this detailed analysis of program execution, these events are used in many researches areas and for a lot of purposes, such as Analysis of Profiling, Power analysis [3], Security Analysis and due to the significant level of unpredictability during the execution of a program and the nondeterminism introduced by the concurrent execution in modern processors, the events counters can also be used for Random Number Generation [11].

Although the PMU is present in most of the modern processors, the number and the type of events that can be monitored vary across different processor architectures and can also vary within the same family of processors that have the same architecture set. Besides this, the events can be classified into two major groups: Architectural and Micro-Architectural events.

- Architectural events behave consistently within a processor's family that has the same architectural set architecture. Some examples of architectural events are the number of executed instructions and the number of exceptions that were taken.

- Micro-Architectural events are events that don't behave consistently within a processor's family. Events captured by two different processors can differ even though they have the same architecture set because this set of events captures information about, for instance, cache hit and cache miss, which are influenced by the cache size and the cache size is a micro-architectural decision.

Collecting accurate event data can be complex due to several reasons that can contaminate the PMU counters, like hardware interrupts that insert a source of non-determinism or over-counting, leading to discrepancies in the counters. In addition, the collection of the event data needs to take into consideration operational aspects, like context-switch monitoring, saving the counters during context switches avoiding contamination from other processes [10].

These events are useful, but there is usually a hardware limitation in how many events can be tracked at the same time. These events are monitored through counters, also called channels, and the available number of these channels varies between architectures, but they are limited and only a few events can be tracked at the same time. Due to the limit in the number of channels, the available information at run-time is also limited and therefore is necessary to select the most crucial events to be monitored.

2.3 SENSORS AND OS VARIABLES

Sensors and OS Variables are another sources of information available in modern hardware and operating systems. The sensors collect information about the hardware, and therefore collect information like Temperature and Energy Consumption.

SO Variables, on the other hand, collect information about the system, and therefore, are dependent on the operating system that is used. The EPOS have several events of the system that can be collect, some examples are CPU Execution Time, Deadline Misses and Running Thread. These events can be used to understand, track back and to optimize the performance of the system.

2.4 ARM ARCHITECTURE

ARM is a family of RISC architectures for computer processors. The ARM is designed to be used in various environments, from portable, battery-powered devices to smartphones, desktops and servers. This is possible due to their low costs, power consumption and heat generation, when compared with their competitors. The two most major version of the ARM family are ARMv7 and ARMv8.

- ARMv7 is a 32-bit architecture, with 32-bit wide registers.
- ARMv8 is a 64-bit architecture, with 64-bit wide registers.

Although the ARMv8 is a 64-bit architecture, it has a backward compatibility with the ARMv7 architecture. This is done by having two execution states: AArch64 and AArch32. The AArch64 is a normal ARMv8 state that uses 64-bit registers. The AArch32 state uses 32-bit registers, having compatibility with the ARMv7 architecture, and in addition supports some features included in the AArch64 state.

Beside the version, the ARM is divided into three different profiles, each one having its application niche. These profiles are M-type, directed for embedded applications, R-type for real-time applications and A-type, a profile that targets high performance markets, such as PC and mobile and is used for general applications.

2.5 FEATURE SELECTION

Feature selection is a process used to reduce the number of input variables in a model, removing non-informative or redundant variables. This is done to reduce the computation cost, improve the performance of the model and avoid the curse of Dimensionality.

The curse of dimensionality is a scenario where the quality of the data required for processing by Machine Learning algorithms decreases due to the exponential increase of collected data. The higher the dimension of data, the higher the prevalence of irrelevant and redundant data [12].

In other cases the feature selection is used, in addition to reducing the redundant variables, to reduce the total number of features to a certain number due to some limitation, like a feature selection of performance counters that are limited by the hardware.

The relevance of a feature is measured not based on its value but on the characteristics of the data and the relationship with other features. This measure is done by feature selection methods that are classified into three types, Filter, Wrapper and Embedded Methods[12].

- Filter method selects features based on statistical measures. Pearson's Correlation and Information Gain are examples of statistical measurements used to measure the importance of the features.

- Wrapper method selects features based on learning algorithms, and the performance of the feature selection depends on the classifier. It's more expensive than Filter methods but more accurate.

- Embedded Methods have the benefits of both the wrapper and filter methods, using ensemble learning and hybrid learning methods. Lasso CV is an example of an embedded method.

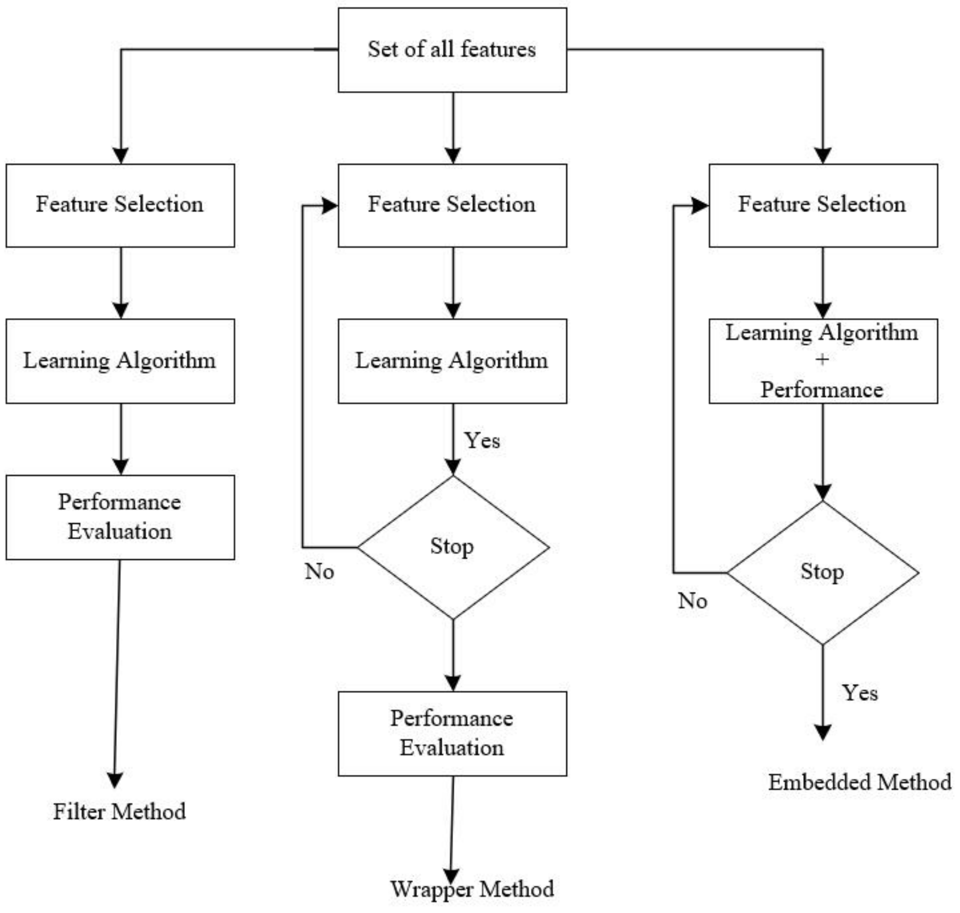

Figure 1 – Overview of feature selection methods

Source: [12]

2.5.1 Pearson’s Correlation Coefficient

Pearson’s Correlation Coefficient (PCC) is a measure to detect a linear relation between two continuous variables. In other words, PCC indicates the effect of change in one variable when the other changes. The equation 2.1 calculate the PCC between an independent variable x and a dependent variable y [12]:

$$\rho(x,y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 (y_i - \bar{y})^2}} \tag{2.1}$$

The coefficient value can range from -1 to +1 and gives information about the direction and the magnitude of the relation.

The direction of the association is defined by the signal of the coefficient, being positive or negative. A positive coefficient indicates that the variables are positively correlated and the values increase or decrease together. A negative coefficient indicated the opposite, when a variable increase, the other decrease.

The magnitude of the association is defined by the value of the coefficient and indicates how strong is the relationship between the variables. The closest from -1 or +1 more strong is the relationship. The value 0 in the coefficient indicates that there is no relationship between the variables.

2.5.2 Information Gain Ratio

Information Gain Ratio is a statistical measure based on the entropy of the data, which indicates the uncertainty in the data. A low entropy indicates the data is uniform, and a high entropy indicates the opposite.

The information gain ratio is calculated as the entropy of the data minus the conditional data entropy, divided by the intrinsic value of the feature, to reduce the biases of features with many distinct values.

2.5.3 Lasso CV

The Least Absolute Shrinkage and Selection Operator with Cross Validation is a linear regression method that performs L1 penalty. If two features are linearly correlated, their presence will increase the value of the cost function and lasso regression will try to shrink the coefficient of the less important feature to zero. With this, lasso regression will automatically select the more useful features.

3 RELATED WORK

In this chapter, we discuss previous research related to this work and compare them to this work.

3.1 ONLINE MACHINE LEARNING FOR ENERGY-AWARE MULTICORE REAL-TIME EMBEDDED SYSTEMS

HOFFMANN's work [5] presents a non-intrusive, Artificial Neural Network (ANN) based runtime energy optimizer that was designed and implemented considering the application's constraints. The model is able to predict the performance of tasks in lower frequency levels and safely optimize real-time embedded systems' power saving operations. This is done by taking execution traces that are built at runtime out of hardware performance counters, sensors, and OS variables of individual tasks as input to predict their performance in case the core where they are running is subjected to DVFS. If it's not, an algorithm is performed to determine if it is possible to realize a migration of the tasks between cores. The proposed solution achieved energy-savings of 24.97% on average when compared to the run-to-end approach. The performance counters used as input by the ANN are selected through 3 feature selection methods after the collect of all the available performance counters on the Cortex-A53 running on ARMv7 mode. The work only takes in consideration the proposed solution running on one architecture. The main difference with our work is the demonstration of the proposed solution when running on other architecture.

4 EXPERIMENTS

This chapter gives information about the execution and environment of the experiments. The experiments are executed running task-sets with distinct behaviors on platforms with ARMv7 and ARMv8 architectures under the Embedded Parallel Operating System.

The experiments are divided into five major sections. First, Section 4.1 describes the overall environment that the experiment was conducted, the operating system configuration, what architectures and platforms are used and what task-sets are executed. Section 4.2 describes how to execute the data capture, what events are collected and its implications. Section 4.3 discusses what methods are used to select the best features. Section 4.4 shows the configuration and how the IA training was conducted. Lastly, Section 4.5 describes how the energy data was collected and how to interpret the results of the collected data.

4.1 ENVIRONMENT

The execution of the experiment involves to collect performance data at run-time in a real-time system. However, some characteristics in this collection need to be fulfilled by the operating system. First, a non-intrusive data acquisition method is necessary in order to not interfere with real-time applications constraints. Second, the data collection needs to be done in a strict way to avoid inaccurate measurements and contamination due to events from other processes. Lastly, besides the performance data that is provided by the Performance Monitor Unit (PMU), is necessary a way to collect information about the use of the system (e.g. CPU execution time, Thread execution time and Deadline misses) to distinguish tasks and be able to make more accurate assumptions while having a guarantee that the system is working as expected.

Therefore, the Embedded Parallel Operating System (EPOS) was chosen due to the fact that it fulfill all the above-mentioned characteristics.

4.1.1 Operating System

Embedded Parallel Operating System (EPOS) is an operating system project developed at the Software/Hardware Integration Lab (LISHA) [9] and aims to automate the development of embedded systems.

EPOS implements a framework that provides non-intrusive mechanisms to collect run-time data from sensors, counters, and SO variables while the system runs real task sets with real workloads (HORSTMANN, HOFFMANN) [6]. To be able to collect performance counters finer gradually without contaminating other threads counters, the data collection is done when a context-switch is performed, saving the collected data and restoring when the task will be running again.

Due to the limitation of the number of channels of PMU imposed by hardware platforms, only a few performance counters can be monitored at the same time, leading to a necessity to execute the same application several times to collect data from the full set of performance counters.

4.1.2 Architectures and Platforms

The experiments are executed in two architectures, ARMv7 and ARMv8. The experiment in ARMv7 was conducted to compare the results described in Hoffmann's [5] work and draw conclusions.

The architectures and platforms used are described below:

- **ARMv7** architecture running on a Raspiberry Pi 3b+ with a ARMv8 Cortex-A53 processor, executing in ARMv7 compatibility mode.
- **ARMv8** architecture running on a Raspiberry Pi 3b+ with a ARMv8 Cortex-A53 processor.

4.1.2.1 Raspberry Pi 3b+

The Raspberry Pi 3b+ uses an ARMv8 Cortex-A53 processor that has 54 performance counters, and a full list of the counters can be founded in the processor manual [1]. The ARMv8 processor has an ARMv7 compatibility mode that allows it to execute ARMv7 assembly code as if it were a ARMv7 processor.

The Cortex-A53 has 6 channels, allowing it to collect 6 performance counters at the same time.

4.1.3 Task-Sets

To collect representative and meaningful data over the performance counters in a multicore real-time embedded system, we need tasks with distinct performance behaviors running in parallel to simulate a real use of the system. To achieve this behavior, 3 types of tasks are selected: a memory bound task that focus on memory access; a cpu bound task that focus on mathematical operation and uses the ALU; and a disparity task that simulates a real workload of embedded systems.

- **Bandwidth** is a memory-bound, performing constantly read and write operations on a data structure with at least the size of the L2 Cache defined by the running platform. This task can be also divided into four types, low use of memory, high use of memory, an alternate between low and high use of memory and random use of memory.

- **CPU Hungry** is a cpu-bound task that implements an Iterative Fibonacci how to execute a loop with mathematical operations constantly using the ALU.
- **Disparity map** is a task that represents a real workload task of embedded systems.

To simulate resources sharing contention between the tasks, the tasks are combined to run in parallel. Two combinations of parallel execution are defined in [5] and replicated here: One composed of a Memory Bound, CPU Bound and Disparity and another composed of a CPU Bound and 2 Disparity. Overall, the configuration of the task-set is presented in table 1.

This task-set is well-defined to have a heavy sharing contention and therefore is used as a main task-set to collect performance data for feature selection.

Task-Set 1			
Task	**Period/WCET (in ms)**	**CPU**	**ID**
Bandwidth	500/100	1	1
CPU Hungry	500/100	2, 3	4, 5
Disparity	500/100	1, 2, 3	2, 3, 6

Table 1 – Task-Set 1 configuration

Besides the task-set 1, two task-sets are defined, and the configuration are described in table 2 and table 3.

Task-Set 2			
Task	**Period/WCET (in ms)**	**CPU**	**ID**
Bandwidth	250/50	1	1
CPU Hungry	250/35, 125/20	2, 3	4, 5
Disparity	500/100, 1000/200, 250,100	1, 2, 3	2, 3, 6

Table 2 – Task-Set 2 configuration

Task-Set 3			
Task	**Period/WCET (in ms)**	**CPU**	**ID**
Bandwidth	100/10, 100/5, 1000/100	1, 2, 3	1, 2, 7
CPU Hungry	100/30, 250/60	3, 3	4, 6
Disparity	1000/400, 500/100	2, 3	3, 5

Table 3 – Task-Set 3 configuration

4.1.4 Equipament

To collect information about the energy consumption when running the task-sets, the KCX-017 USB Power Meter was used.

4.2 DATA CAPTURE

The first step of the experiment is to capture meaningful data that represents a heavy use of the system and resource sharing contention between the tasks. To fulfill this objective, task-set 1, described in table 1 was used.

In this step, all the performance counters and some SO events are collected. Due to the hardware limitation on the number of channels of the PMU, the number of performance counters that can be collected at the same time is limited. This implies running the application several times to collect all necessary data. For example, on Raspiberry PI 3b+, it's necessary to collect data 9 times, because this platform has 54 performance counters but only 6 can be collected at the same time.

Besides the performance counters, the SO events also need to be collected. In this experiment, the SO events collected are CPU Usage, Thread Usage and Running Thread. They are necessary to collect information about the running task at the moment that the collection was done. This gives information about the cpu usage, thread usage and task id, used to know what the data collected refer to. The capture of the data is done with the system on the maximum frequency and for 180 seconds in each collection.

After the collection, all the logs of the system are filtered and cleared, removing all unnecessary information and merging all the information collected in chronological order. Due to the fact that the collected performance counters are done in a cumulative fashion way (i.e. each collection use the previous one as a base), it's necessary to subtract a current collection from its previous corresponding collection to have the correct values.

4.3 FEATURE SELECTION

After the data capture, the next step is to select the most important features within all the performance counters that the platform offers. The number of features that will be selected depends on the limitation of the PMU channels of the processor.

To achieve this objective, 3 methods for feature selection are used: Pearson Correlation Coefficient (PCC), Lasso CV and Info Gain Ratio. The mentioned feature selection methods give distinct results over different features. With this, we used a combination of the methods results to select the features.

The PCC is a measure to detect a linear relation between two continuous variables, being the main methods used in the decision of the feature selection. The other 2 algorithms will be used to give extra information and to decide in case there are some features with close relation and if it's necessary to select one of them.

As mentioned above, the PCC detects a relation between two variables, therefore, the relation between all the performance counters will be calculated, but the relation between the performance counters and the Thread usage will be our main concern. A feature that has a great relationship with the thread usage indicated that the feature is a good indication and therefore, a good feature to be selected.

For instance, two features can be on the top of the best features list when compared with the thread usage but when compared between each other, the two features can be highly correlated, meaning that if we selected the two features, only one will give a meaningful value. With this, we need to verify the correlation between the top features to eliminate the redundant variables. The Lasso CV and Info Gain Ratio are used to decide what feature will be selected.

4.4 IA TRAINING

After the feature selection phase, we already know what are the best features to be used to train the IA. The online artificial neural network model in this work is used to predict the utilization of the task if the frequency is reduced. With this, we can predict the total utilization of the cores and decide if it's possible to scale the frequency of the CPU using Dynamic Voltage and Frequency Scaling (DVFS), reducing the overall energy consumption of the chip. The configuration of the ANN is the same used on (HOFFMANN 2021) [5]. The model uses a learning rate of 0.35 and an average accuracy of 97% considering the validation data-set.

Even though the ANN used to run the experiments is trained online, offline training is necessary beforehand to avoid cold start issues. This training is done using two datasets of data, one for training and the other for validation. The training sample is got by running the task-set 1, described in the table 1 and collecting data of the selected performance counters under all levels of frequency scaling. With this, we collect the behavior and performance counter of the system under each frequency level. The training data is collected using task-set 2, described in the table 2 under the same circumstances.

4.5 ENERGY ANALYSIS

The last step of the experiment is to measure the energy consumption of the system when under the energy optimizer, which scale the frequency based on the use of the system. Not only the DVFS is used, but also a balanced workload using migration of tasks, improving, even more, the energy more the energy consumption.

To capture the energy consumption of the system, the task-set 1, task-set 2 and task-set 3 are executed, with and without the ANN prediction and DVFS actuation. The data is collected by the equipment KCX-017 USB Power Meter. Besides this, each task-set has a Bandwidth task, that can be divided into four types, low use of memory, high use of memory, an alternate between low and high use of memory and random use of memory. The task-sets are executed for each type of bandwidth task as well.

The first step is to capture how much energy the idle system consumes, running the application without tasks, in the maximum frequency. Next, the consumption of all the task-sets is collected, running on the maximum frequency. After that, the energy consumption of the task-sets with the ANN prediction is measured. With this, we subtract the energy consumption of the idle system and compare the energy consumption.

5 RESULTS

In this chapter, we present and discuss the results of the experiments.

Section 5.1 presents the results in the ARMv7 architecture and compares it with the original work. Section 5.2 presents the results in the ARMv8 architecture and compares it with the ARMv7 results.

5.1 ARMV7 ARCHITECTURE

5.1.1 Feature Selection

In the feature selection phase, the results of the most relevant features are described in Figure 2. They are ordered by the results of the PCC method.

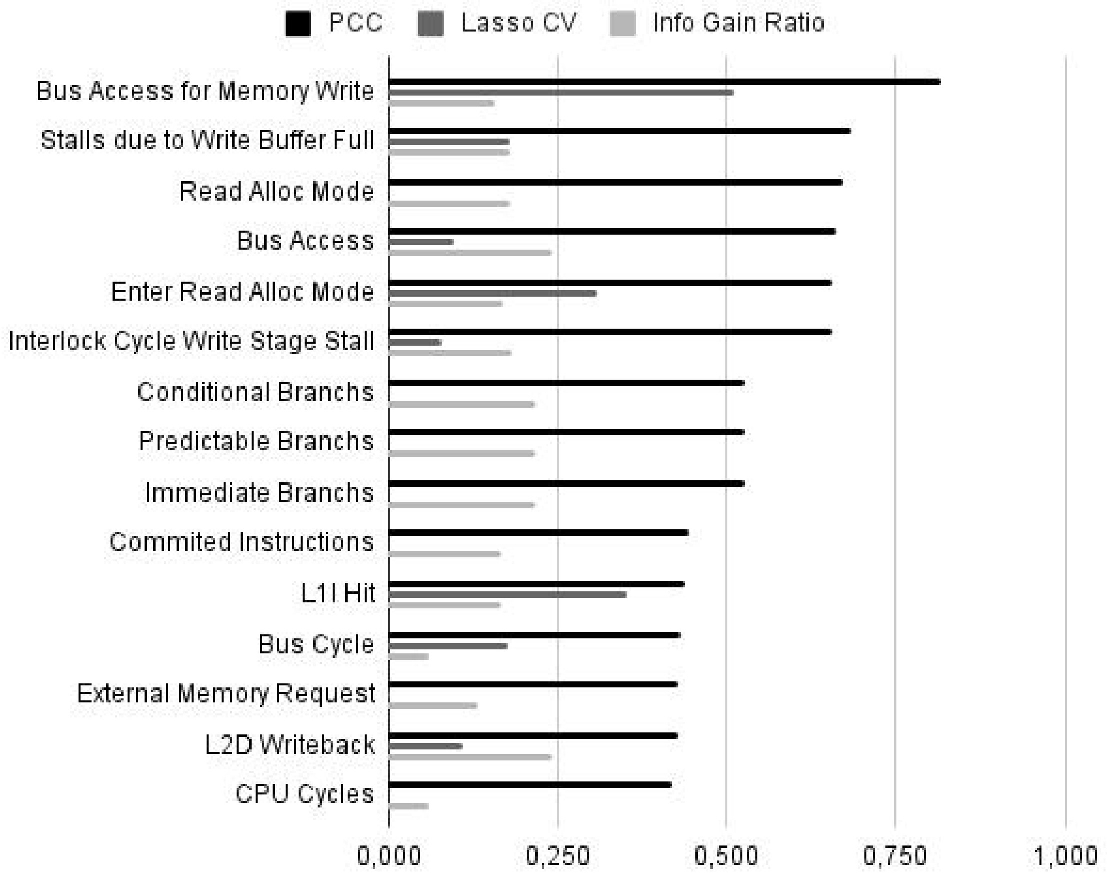

Figure 2 – Ranking results for the most relevant features on ARMv7.

The first feature, Bus Access for Memory Write will be selected for having the best results in both PCC and Lasso CV methods. This feature also has a relation of 92% with the feature Bus Access, which has a high result in PCC as well. Due to this, only the first one will be selected. The performance trace of these features are shown in Figure 3 and Figure 4.

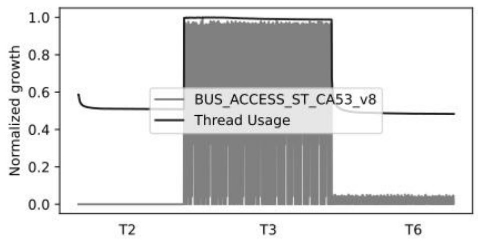

Figure 3 – ARMv7 - Bus Access for Memory Write

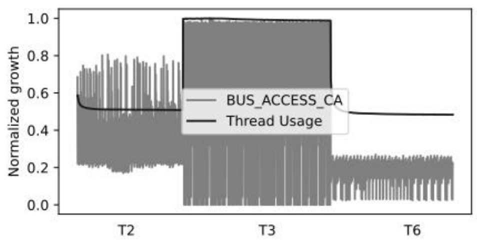

Figure 4 – ARMv7 - Bus Access

Following the list, we have the feature Stalls due to Write Buffer Full. This feature has a high relation with other three features, which are also in the list: Read Alloc Mode (94%), Enter Read Alloc Mode (96%) and Interlock Cycle Write Stage Stall(96%). The performance trace of these features are shown in figures 5, 6, 7 and 8. Between these features, only Stalls due to Write Buffer Full will be selected for having a higher value on PCC and having significant values on Lasso CV and Info Gain.

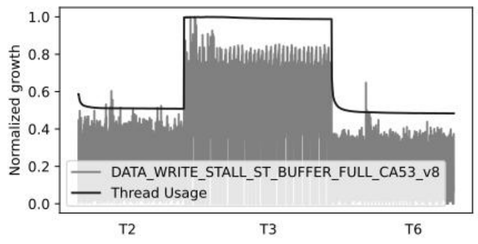

Figure 5 – ARMv7 - Stalls due to Write Buffer Full

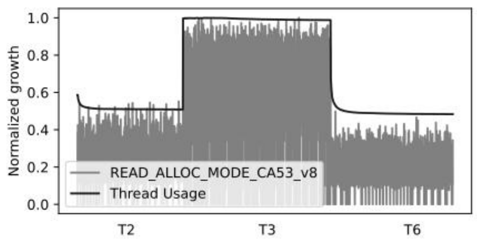

Figure 6 – ARMv7 - Read Alloc Mode

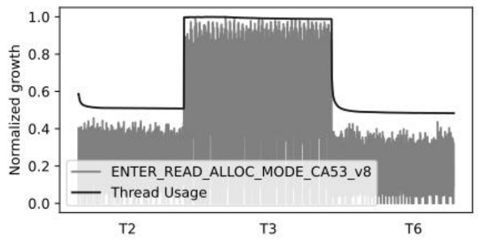

Figure 7 – ARMv7 - Enter Read Alloc Mode

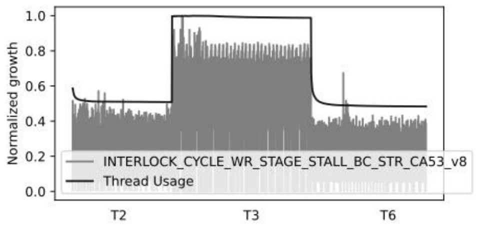

Figure 8 – ARMv7 - Interlock Cycle Write Stage Stall

The next feature is Conditional Branchs. This feature has a high relation with the features Predictable Branchs (99%), Immediate Branchs (99%), Commited Instructions (91%) and L1I Cache Hit (91%). The performance trace of these features are shown in figures 9, 10, 11 and 13. With this, only the Conditional Branchs is selected for having the highest PCC among the cited features.

Bus Cycle is the next feature, having significant values on PCC and Lasso CV, and therefore was selected. This feature has a high PCC relation with the feature CPU Cycles (98%). The performance trace of these features are shown in figures 14 and 15. With this, only the Bus Cycle is selected for having the highest PCC among the cited features.

External Memory Request also was selected for having a high value in PCC and Info Gain Ratio. This feature doesn't have a high PCC relation with other feature of the list and the performance trace of this feature is shown in Figure 16.

The last feature is L2D Writeback, having good results in all three methods. This feature also doesn't have a high PCC relation with other feature of the list, and the performance trace of this feature is shown in Figure 17.

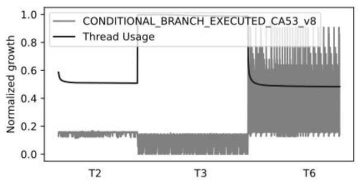

Figure 9 – ARMv7 - Conditional Branchs

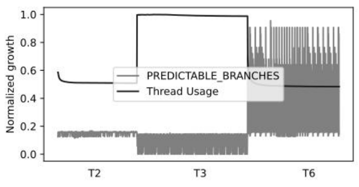

Figure 10 – ARMv7 - Predictable Branchs

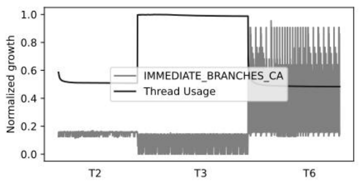

Figure 11 – ARMv7 - Immediate Branchs

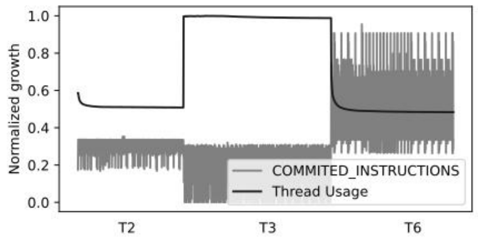

Figure 12 – ARMv7 - Committed Instructions

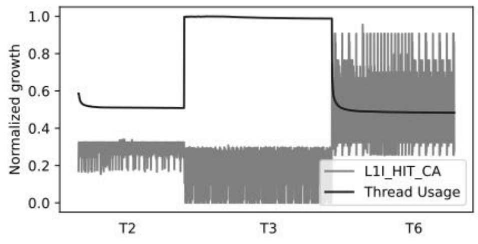

Figure 13 – ARMv7 - L1I Cache Hits

With this, the selected features are: Bus Access for Memory Write, Stalls due to Write Buffer Full, Conditional Branchs, Bus Cycle, External Memory Request and L2D Writeback.

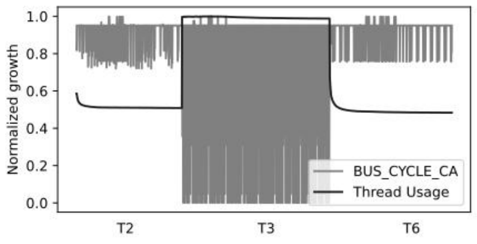

Figure 14 – ARMv7 - Bus Cycles

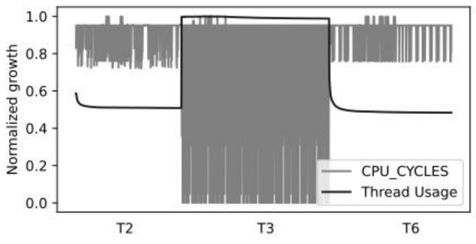

Figure 15 – ARMv7 - CPU Cycles

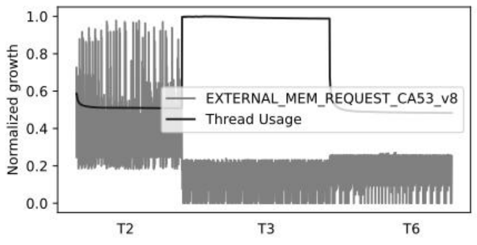

Figure 16 – ARMv7 - External Memory Request

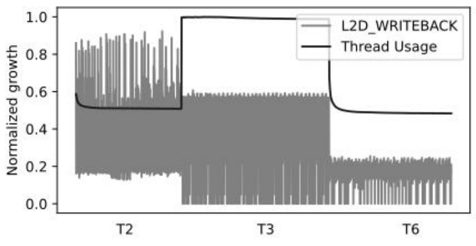

Figure 17 – ARMv7 - L2D Writeback

5.1.2 Energy Optimizer

After the feature selection phase, the execution of the task-sets 1, 2 and 3 described in section 4.1 are performed with and without the ANN optimizer, collecting data about the energy consumption of the tasks. Under the ANN, each performance counter has a weight associated. After the offline training, each task-set is executed several times to perform online training to find the best weights that the IA can use to predict more precisely the migrations, swaps and frequency scaling of a specific task-set. After this, we use this weight to execute the task-sets to collect information about energy consumption.

All the task-sets have Bandwidth tasks, which are divided into four types: heavy, light, mixed and random. Each one has a different memory consumption, for a maximum to a minimum use, making the simulation of a real use of the system more accurate. In this section, all graphs are under heavy use of memory. In addition, all data collected of energy consumption are done in executions of 180 seconds.

5.1.2.1 Task-set 1

The first task-set to be executed is described in table 1. The traces of the execution of the task-set with the energy optimizer are described in graph 18. The lines on the graph described the current utilization of the CPUs and the dots represent the prediction of the performance if the CPU decreased the frequency. The bars described the current frequency of the CPU. In this task-set, the frequency is decreased from 1,2GHz to 0,6GHz, the minimum supported on the platform, without the necessity of migrations.

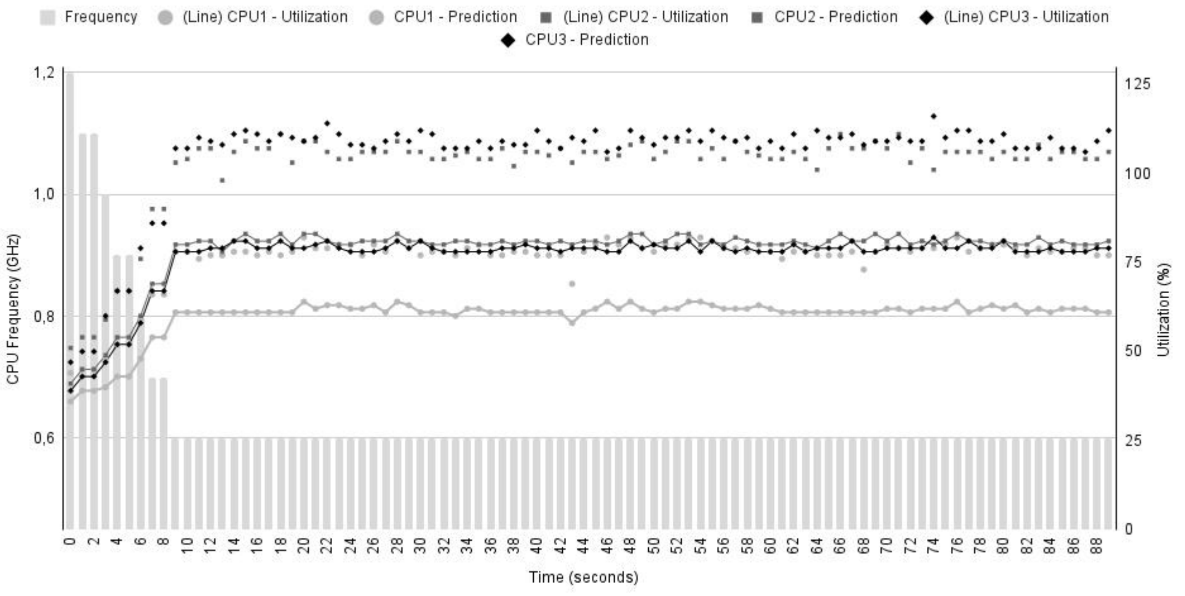

Figure 18 – ARMv7 - Task Set 1

The actuation of energy optimizer in this task-set resulted in a reduction on energy consumption of 23%. The graph of the consumption is described in graph 19. The graph shows that the reduction in energy consumption was 37% when executed with a heavy use of memory. On the other hand, with random use of the memory, the energy consumption didn't change.

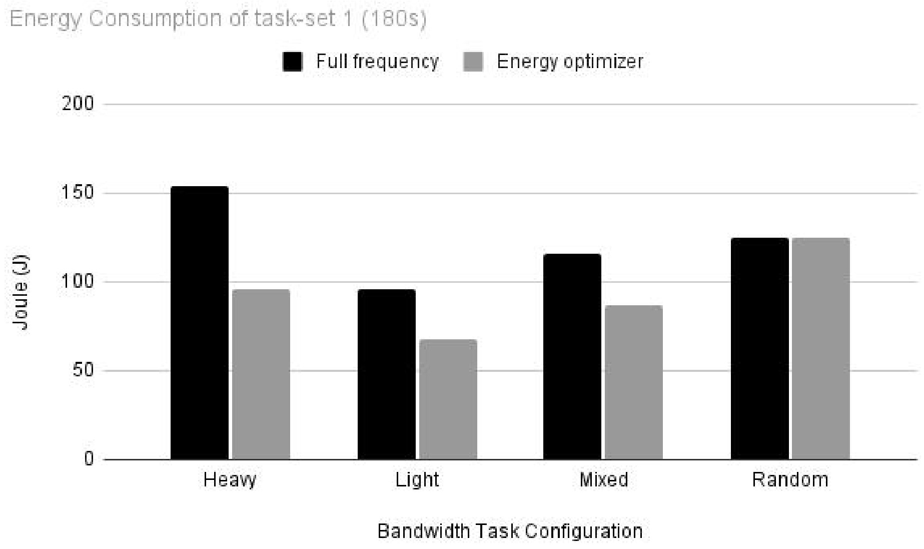

Figure 19 – ARMv7 - Task Set 1 Energy Consumption

5.1.2.2 Task-set 2

The second task-set to be executed is described in table 2. The traces of the execution of the task-set with the energy optimizer is described in graph 43. The lines on the graph described the current utilization of the CPUs and the points represent the prediction of the performance if the CPU decreased the frequency. The bars described the current frequency of the CPU.

Initially, this task-set has an imbalance in the use of the cores, decreasing the frequency to a maximum of 0,7GHz where the CPU3 has a use of almost 100% while the CPU1 has a use of nearly 60%. With this, the energy optimizer performs a migration of tasks from CPU3 to CPU1, described on the graph, leading to a more balanced use of the platform, decreasing the frequency to a minimum of 0,6GHz.

The energy consumption of this task-set results in a reduction of 10% on average. The graph of the energy consumption is described in the graph 21. The graph shows that the energy consumption was reduced by 15% and 18% with heavy and light bandwidth configuration, respectively.

5.1.2.3 Task-set 3

The third task-set to be executed is described in table 3. The traces of the execution of the task-set with the energy optimizer is described in graph 23. The lines on the graph described

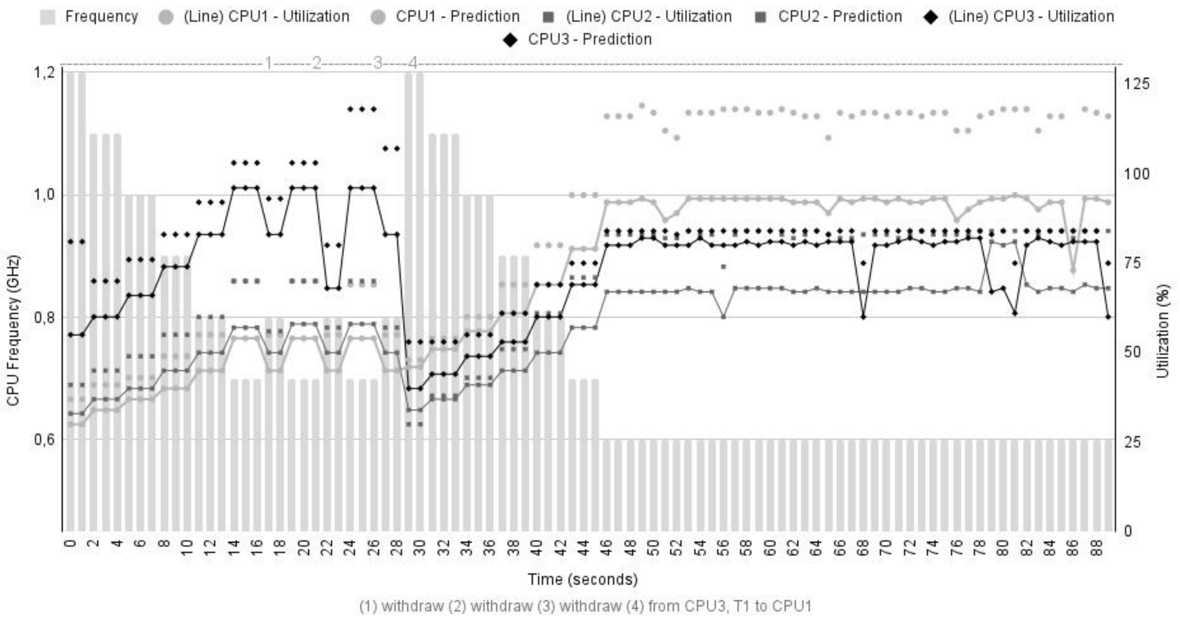

Figure 20 – ARMv7 - Task Set 2

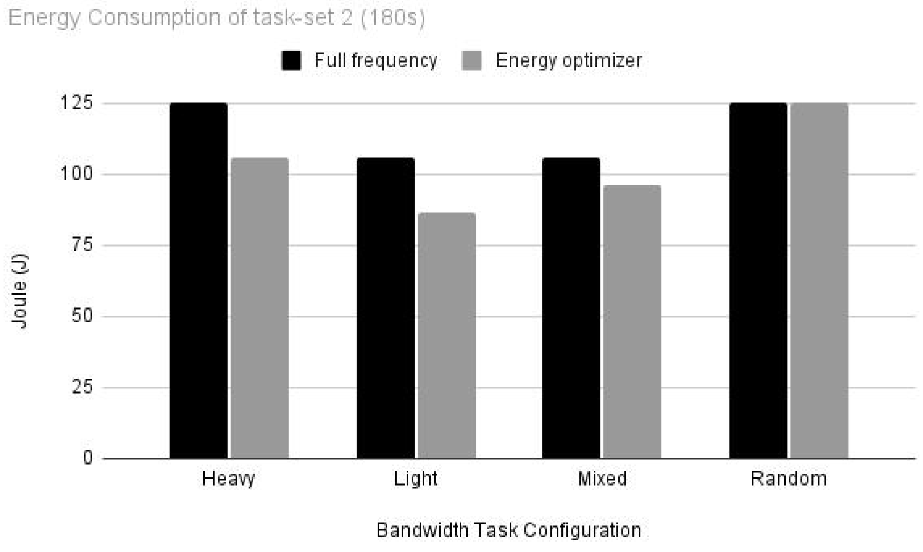

Figure 21 – ARMv7 - Task Set 2 Energy Consumption

the current utilization of the CPUs and the points represent the prediction of the performance if the CPU decreased the frequency. The bars described the current frequency of the CPU.

This task-set is the most complex among the task-sets described in section 4.1 and takes several executions to find weights to configure the ANN to converge to a frequency. The graph 22 are shows the execution trace of the task-set 3 using default values for the weights of the ANN. In this execution, the tasks didn't accomplish a stability on the frequency, leading to several migrations and swaps.

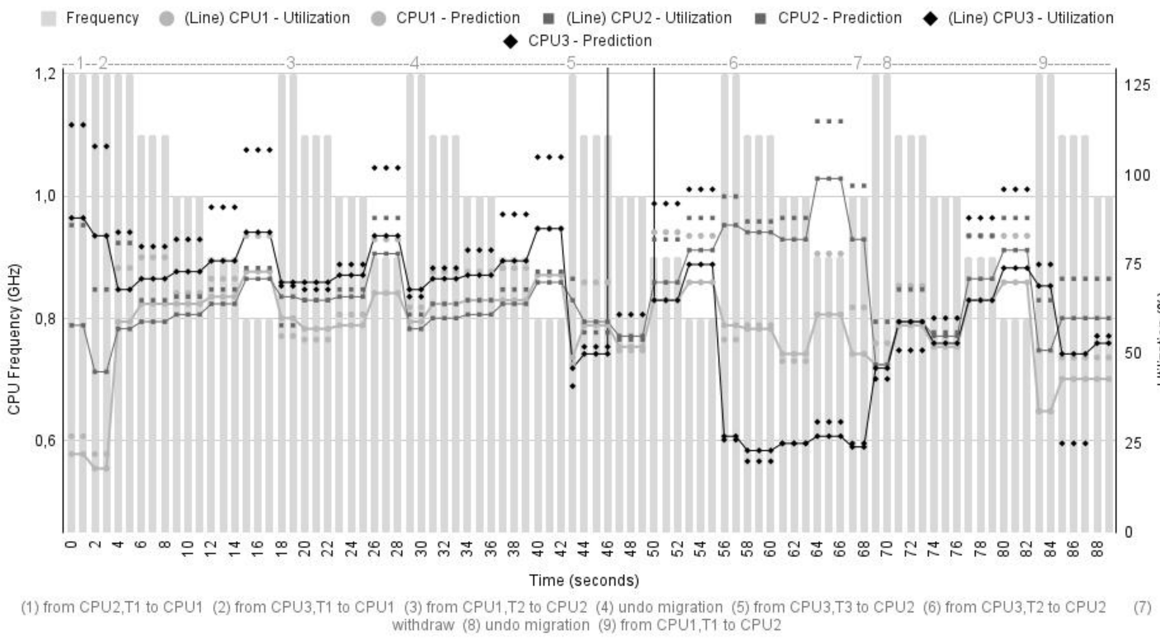

Figure 22 – ARMv7 - Task Set 3 - Default Weights

After several executions to find the migrations weights that fit the task-set, the execution trace of the task-set 3 with the weights defined is shown in graph 23. The graph shows that the frequency was decreased and found stability on 0,9GHz after 5 migrations.

The energy consumption of this task-set results in a reduction of 10% on average. The graph of the consumption is described in graph 24. The graph shows that the energy consumption was reduced by 26% with heavy bandwidth configuration.

5.1.3 Energy Consumption

After considering all task-sets, the overall energy consumption reduction is shown in graph 25. The graph shows the energy reduction of all task-sets when considering all bandwidth task configurations.

The heavy configuration on the bandwidth task was the one that shows the best results, with an average reduction of 26%. The next is the light configuration, which has a low use of the memory, achieve an average reduction in the consumption of 18%. The next, mixed configuration, has an average reduction of 13%. The last configuration, the random, which makes random use of memory, doesn't achieve a reduction in energy consumption on the tests.

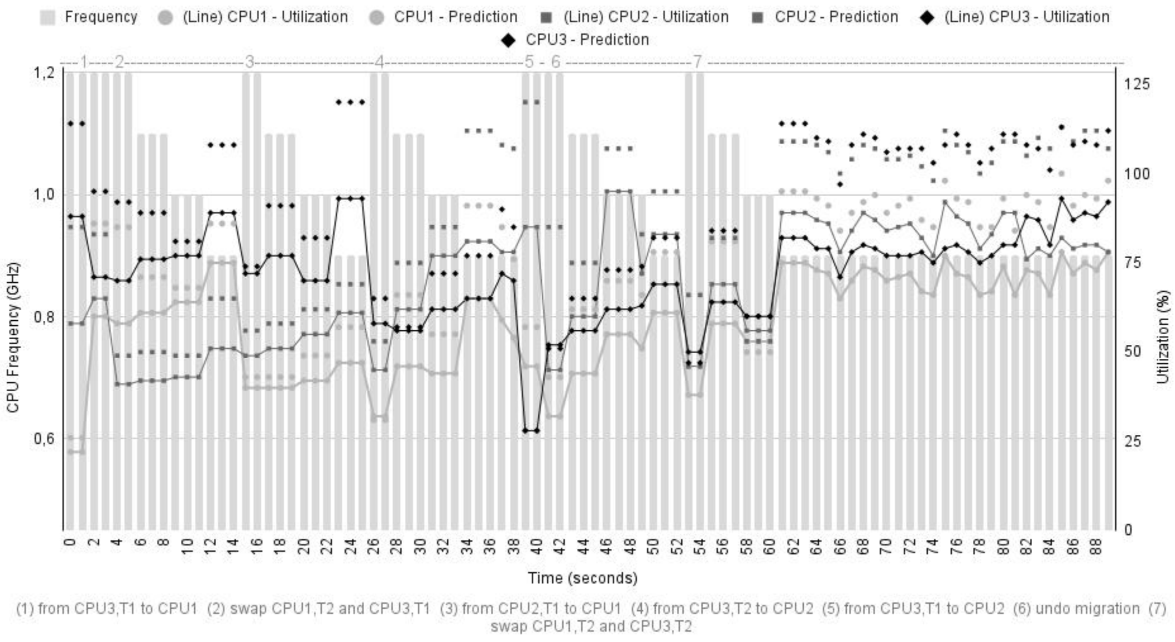

Figure 23 – ARMv7 - Task Set 3

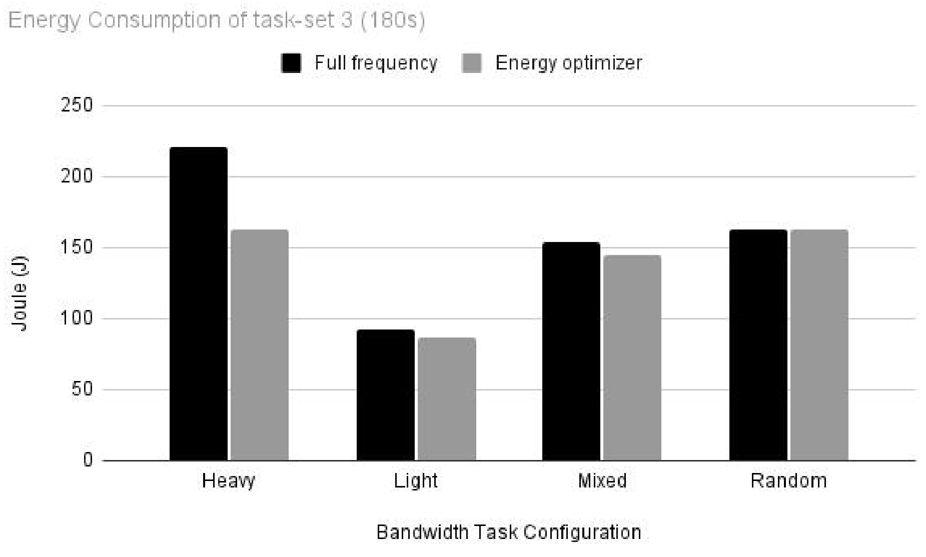

Figure 24 – ARMv7 - Task Set 3 Energy Consumption

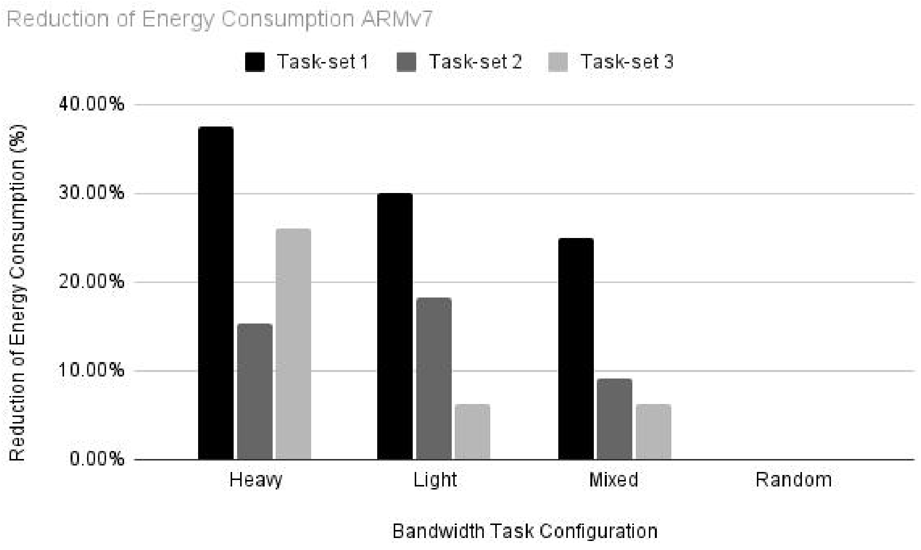

Figure 25 – ARMv7 - Energy Consumption

On average, the reduction of energy consumption considering all the task-sets and all the bandwidth task configurations was 19%.

5.1.4 Comparison with the original work

After executing the tests under the ARMv7 architecture, the results can be compared with the results described in HOFFMANN's [5] work.

The first step is to compare the results of the feature selection phase. The features selected by HOFFMANN are Bus Access for Memory write operations, Stalls due to Write Buffer Full, L2D Writeback, Immediate Branches, CPU Cycle Count, and L1 Cache Hits. Between the features, Bus Access for Memory write operations, Stalls due to Write Buffer Full and L2D Writeback are selected in the feature selection phase of this work as well. The Immediate Branch was changed to Conditional Branches, which has a correlation of 99% with Immediate Branches in PCC. The CPU Cycle was changed to Bus Cycles, which has a correlation of 98% in PCC. The last feature is different. This work selected the feature called External Memory Request while HOFFMANN selected a feature L1 Cache Hits. While the features are different, there are some relation between them. The first, External Memory Request, has a high relation (92%) with another similar feature, L2 Cache Hits. Therefore, even if the feature is different, there are semantically relations.

The next step is the execution of the tasks with the energy optimizer and collect data about the energy consumption. The execution trace of the task-sets has a similar behavior to the HOFFMANN's work. The only difference was on task-set 3, where the energy optimizer reduced the frequency to 0,8GHz, while in this work was only 0,9GHz. This made a difference in the average energy consumption of the task-set 3 by 5%.

Considering the bandwidth task configuration, this work shows the best results in the heavy and low configurations, but a worse result in the mixed configuration. Comparing the results, this work achieves a decrease of 2% more on the heavy configuration and 1% more on the low configuration. On the other side, the difference in mixed configuration was 7%. The overall decrease of energy consumption of HOFFMANN was 21.38% while this work achieve an overall reduction of 19.30%.

5.2 ARMV8 ARCHITECTURE

5.2.1 Feature Selection

In the feature selection phase, the results of the most relevant features are described in Figure 26. They are ordered by the results of the PCC method.

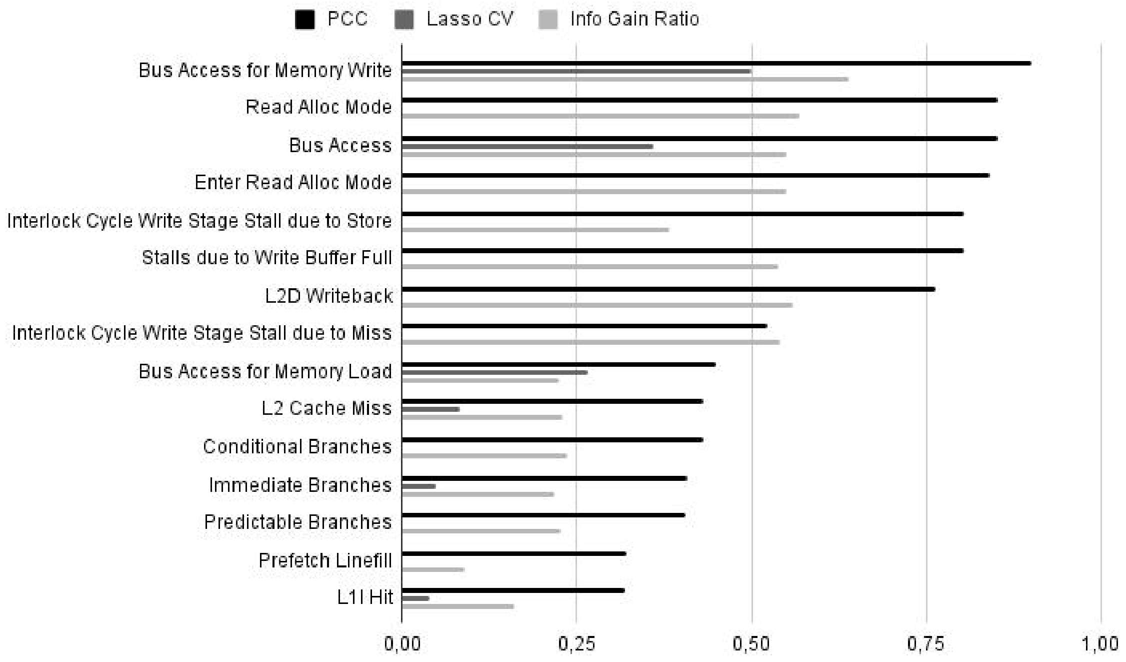

Figure 26 – Ranking results for the most relevant features on ARMv8.

The first feature, Bus Access for Memory Write will be selected for having the best results in the three methods used to select the best features: PCC, Lasso CV and Info Gain methods. This feature has a high relation with others features, being the most important the Read Alloc Mode (96%), Bus Access (95%), Enter Read Alloc Mode (95%) and L2 Writeback (91%). The performance trace of theses features are shown in figures 27, 28, 29, 30 and 31. Between these features, only Bus Access for Memory Write will be selected for having the best values over all the others features.

Following the list, we have the feature Interlock Cycle Write Stage Stall due to Store. This feature has a high value in both PCC and Info Gain methods and has a high relation with the feature Stalls due to Write Buffer Full (97%). The performance trace of these two features are shown in figures 32 and 33. Between these two features, only the Interlock Cycle Write

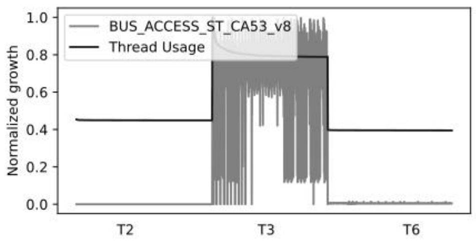

Figure 27 – ARMv8 - Bus Access for Memory Write

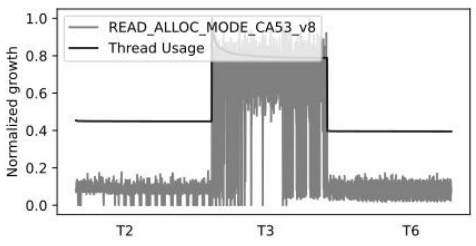

Figure 28 – ARMv8 - Read Alloc Mode

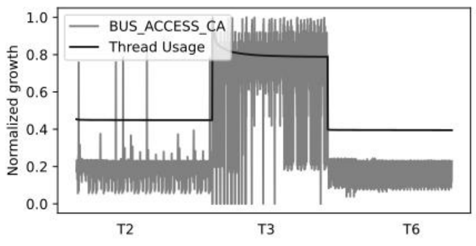

Figure 29 – ARMv8 - Bus Access

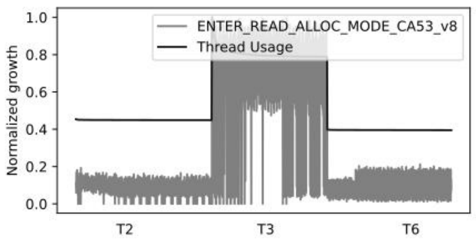

Figure 30 – ARMv8 - Enter Read Alloc Mode

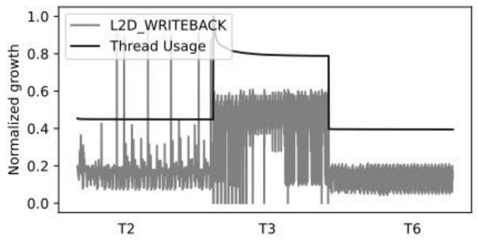

Figure 31 – ARMv8 - L2D Writeback

Stage Stall due to Store will be selected for having a high value on PCC and Info Gain. This feature also has a high relation with the others features discussed before, but it's still selected for having a high value on Info Gain and for being a complementary feature, that is, stalls when writing.

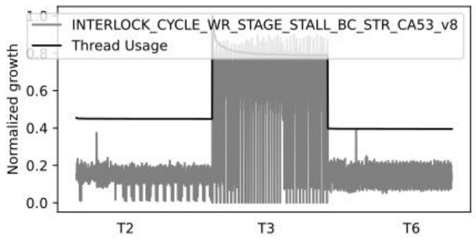

Figure 32 – ARMv8 - Interlock Cycle Write Stage Stall due to Store

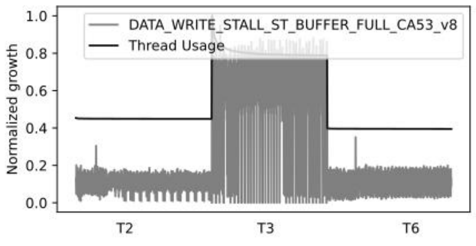

Figure 33 – ARMv8 - Stalls due to Write Buffer Full

The next feature is Interlock Cycle Write Stage Stall due to Miss, having a high value on PCC and Info Gain, being therefore, selected. This feature doesn't have a high PCC relation with otehrs features of the list. The performance trace of this feature is shown in Figure 34.

Bus Access for Memory Load is the next feature and also was selected. This feature has a high value in both PCC and Lasso CV. This feature doesn't have a high PCC relation with others features of the list, and the performance trace of this feature is shown in Figure 35.

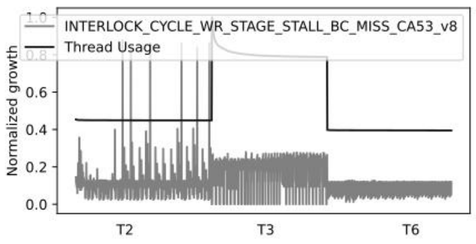

Figure 34 – ARMv8 - Interlock Cycle Write Stage Stall due to Miss

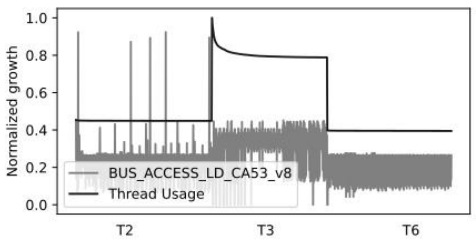

Figure 35 – ARMv8 - Bus Access for Memory Load

Following the list, we have the feature L2 Cache Misses, who also has a high value in PCC and Lasso CV. This feature doesn't have a high PCC relation with others features of the list, and the performance trace of this feature is shown in Figure 36.

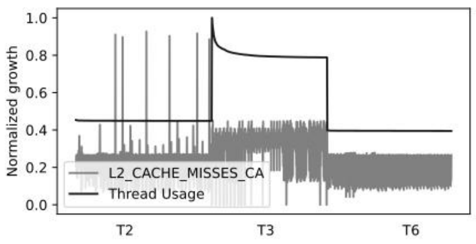

Figure 36 – ARMv8 - L2 Cache Miss

The last feature is Conditional Branchs. This feature has a high relation with the features Immediate Branchs (98%), Predictable Branchs (98%) and L1I Cache Hit (93%). The performance trace of these features are shown in figures 37, 38, 39 and 40. With this, only the Conditional Branchs is selected for having the highest PCC among the cited features.

With this, the selected features are: Bus Access for Memory Write, Interlock Cycle Write Stage Stall due to Store, Interlock Cycle Write Stage Stall due to Miss, Bus Access for Memory Load, L2 Cache Misses and Conditional Branchs.

5.2.2 Energy Optimizer

After the feature selection phase, the execution of the task-sets 1, 2 and 3 described in section 4.1 are performed with and without the ANN optimizer, collecting data about the energy consumption of the tasks. After the offline training, each task-set is executed several times to

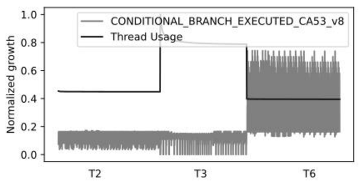

Figure 37 – ARMv8 - Conditional Branchs

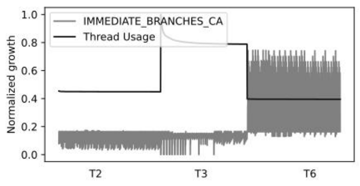

Figure 38 – ARMv8 - Immediate Branchs

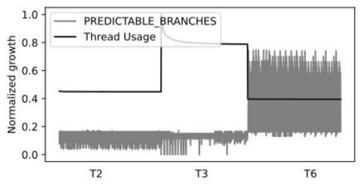

Figure 39 – ARMv8 - Predictable Branchs

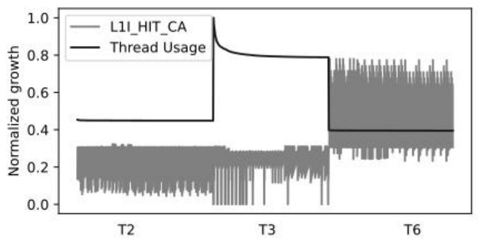

Figure 40 – ARMv8 - L1I Cache Hits

find the best migrations weights to predict more precisely the migrations and swaps. After this, we use this weight to execute the task-sets to collect information about energy consumption.

All the task-sets have Bandwidth tasks, with are divided into four types: heavy, light, mixed and random. Each one has a different memory consumption, for a maximum to a minimum use, making the simulation of a real use of the system more accurate. In this section, all graphs are under heavy use of memory. In addition, all data collected of energy consumption are done in executions of 180 seconds.

5.2.2.1 Task-set 1

The first task-set to be executed is described in table 1. The traces of the execution of the task-set with the energy optimizer is described in graph 41. The lines on the graph described the current utilization of the CPUs and the points represent the prediction of the performance if the CPU decreased the frequency. The bars described the current frequency of the CPU. In this task-set, the frequency is decreased from 1,2GHz to 0,6GHz, the minimum supported on the platform, without the necessity of migrations.

The energy consumption of this task-set results in a reduction of 23% on average. The graph of the consumption is described in graph 42. The graph shows that the reduction in energy consumption was 33% when executed with a heavy use of memory. On the other hand, with random use of the memory, the energy consumption didn't change.

5.2.2.2 Task-set 2

The second task-set to be executed is described in table 2. The traces of the execution of the task-set with the energy optimizer is described in graph 43. The lines on the graph described the current utilization of the CPUs and the points represent the prediction of the

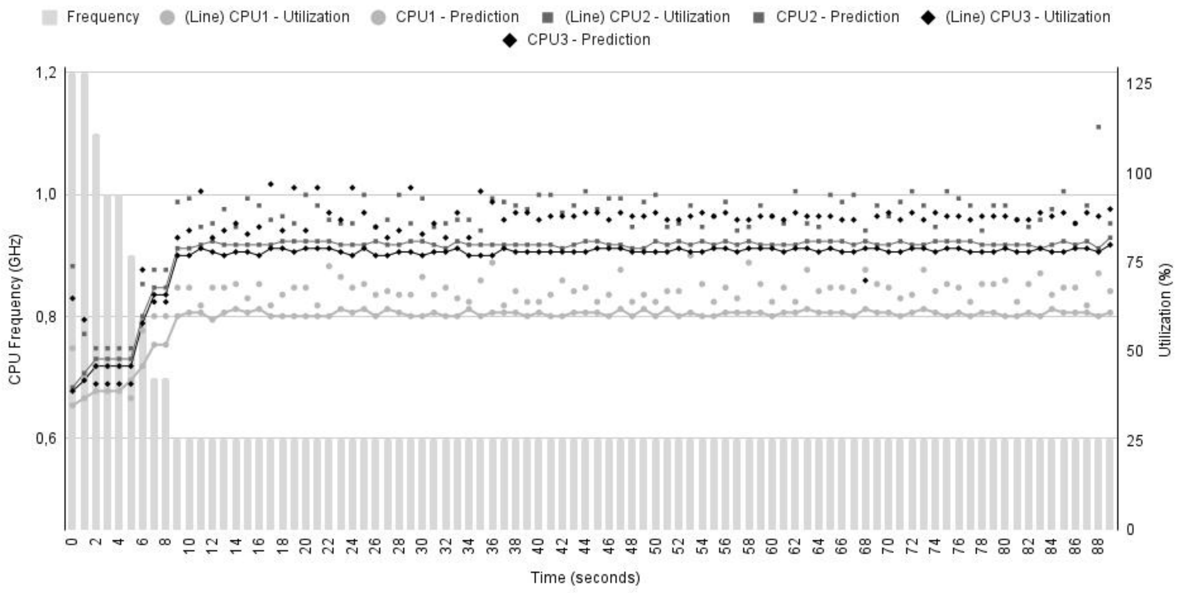

Figure 41 – ARMv8 - Task Set 1

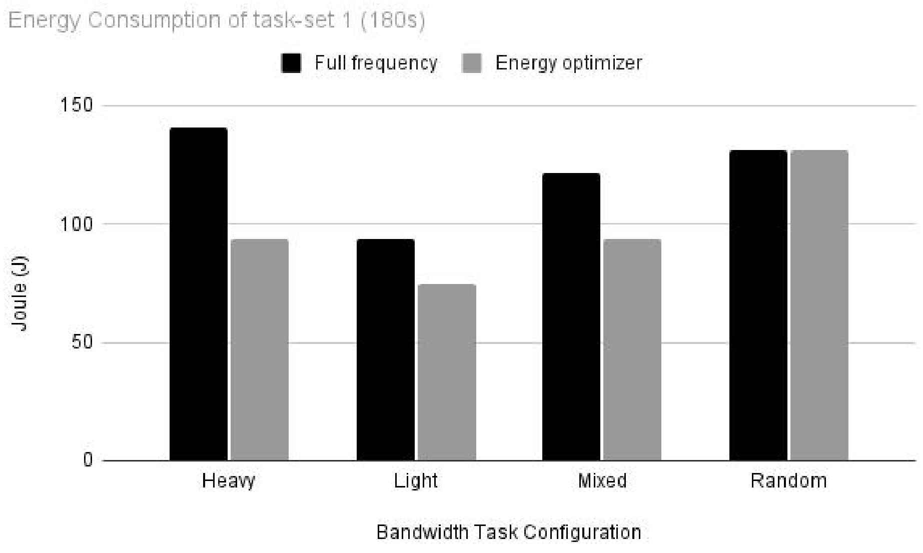

Figure 42 – ARMv8 - Task Set 1 Energy Consumption

performance if the CPU decreased the frequency. The bars described the current frequency of the CPU.

Initially, this task-set has an imbalance in the use of the cores, decreasing the frequency to a maximum of 0,7GHz where the CPU3 has a use of almost 100% while the CPU1 has a use of nearly 60%. With this, the energy optimizer performs a migration of tasks from CPU3 to CPU1, described on the graph, leading to a more balanced use of the platform, decreasing the frequency to a minimum of 0,6GHz.

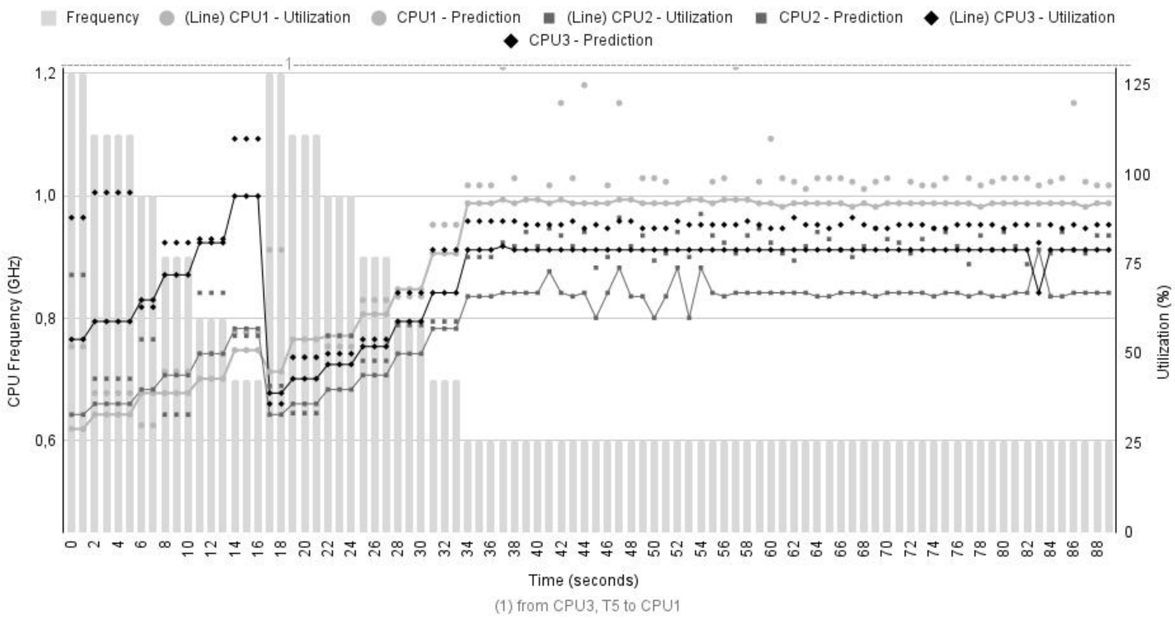

Figure 43 – ARMv8 - Task Set 2

The energy consumption of this task-set results in a reduction of 8% on average. The graph of the consumption is described in the graph 44. The graph shows that the energy consumption was reduced by 11% in both heavy and light bandwidth configuration.

5.2.2.3 *Task-set 3*

The third task-set to be executed is described in table 3. The traces of the execution of the task-set 3 is described in graph 45. The lines on the graph described the current utilization of the CPUs and the points represent the prediction of the performance if the CPU decreased the frequency. The bars described the current frequency of the CPU.

This task-set is the most complex among the task-sets described in section 4.1. The graph shown that this task set starts with a high difference on the current use and prediction of the use of the CPU3. On the first try, there is a discrepancy on the use of the CPU 3 compared to the others CPUs, therefore, a migration is performed, sending one task from CPU3 to CPU1. Even with this migration, the prediction of the CPU3 is still very high, therefore, the frequency is not decreased. This problem occurs on the way the offline and online learning is performed.

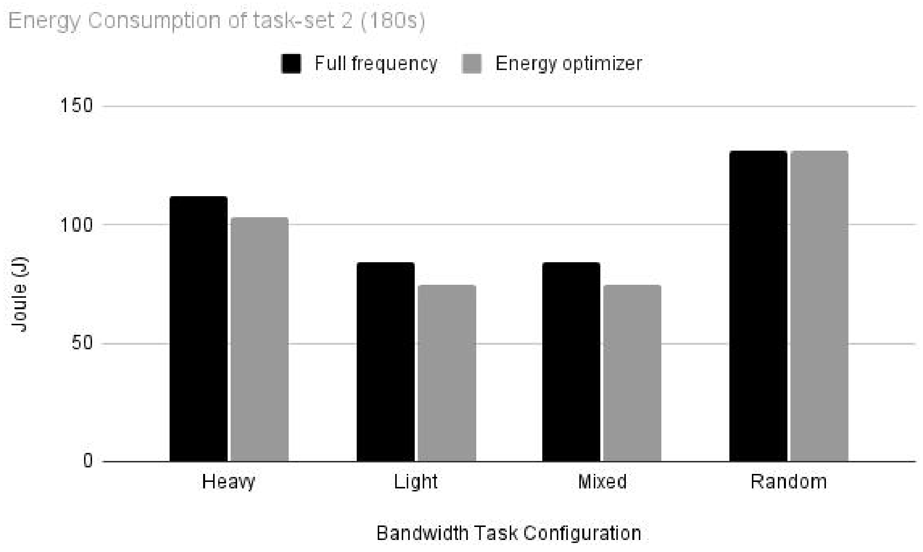

Figure 44 – ARMv8 - Task Set 2 Energy Consumption

The offline training is performed to have a warm-up start, allowing bypass an initial stage of online training. This training is done with information of the Task Set 1 and not the current task set. This happens because the Task Set 1 have a configuration for different scenarios, where we have different tasks running in parallel. Even though the task set 3 run the same tasks, there are some differences on the number of tasks and what tasks are running in parallel, therefore, not every scenario can be covered.

To complement this offline training, an online training is also performed, to make sure the current task set configuration is being predicted correctly, and this training is executed when we have a variation on the frequency.

This lead the problem on the task set 3. In this scenario, the offline training is not enough to predict correctly the configuration of the task set 3, leading to a bad warm-up. Even though migration was performed, this doesn't solve the problem. The online training also was never performed because we never decreased the frequency, making the frequency stuck on the 1,2GHz.

5.2.3 Energy Consumption

After considering all task-sets, the overall energy reduction is shown in graph 46. The graph shows the energy reduction of the task-sets 1 and 2 when considering all bandwidth task configurations. The Task Set 3 was not included due to the problem where the frequency was not decreased.

The heavy configuration on the bandwidth task was the one that shows the best results, with an average reduction of 21%. The next is the light configuration, which has a low use of the memory, achieve an average reduction in the consumption of 16%. The next, mixed configuration, has an average reduction of 17%. The last configuration, the random, which

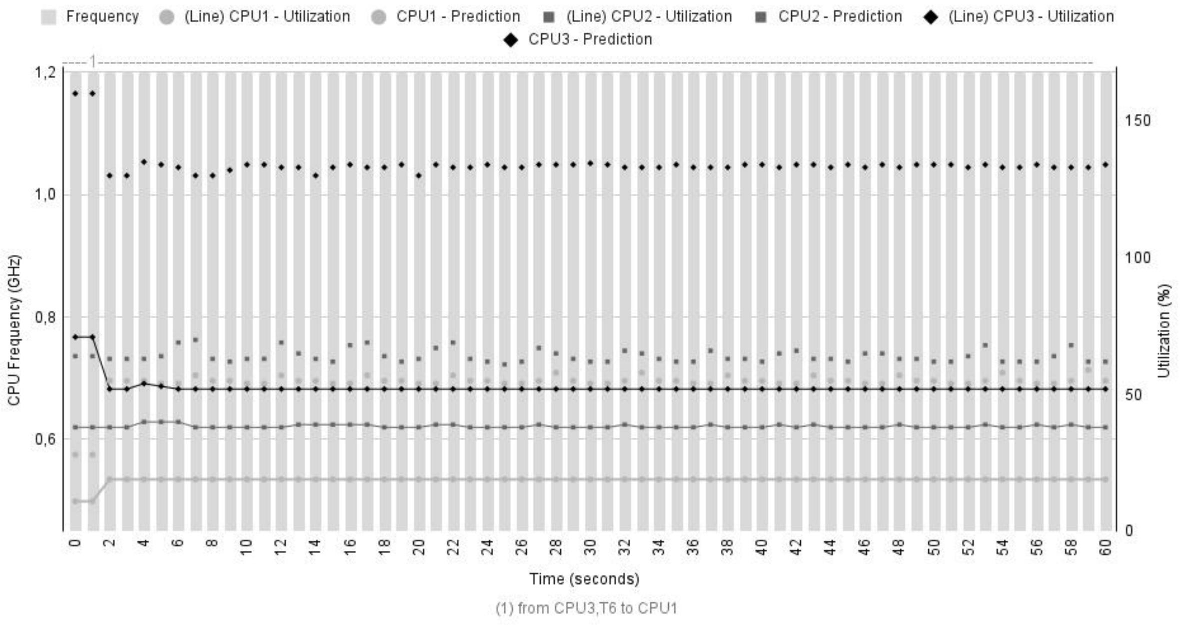

Figure 45 – ARMv8 - Task Set 3 - Default Weights

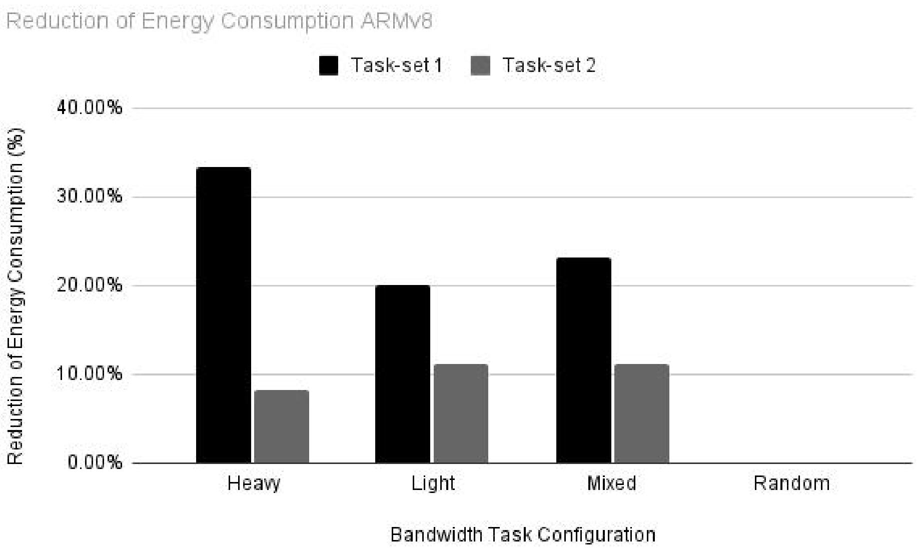

Figure 46 – ARMv8 - Energy Consumption

makes random use of memory, doesn't achieve a reduction in energy consumption on the tests.

On average, the reduction of energy consumption considering all the task-sets and all the bandwidth task configurations was 13%.

5.2.4 Comparison with the ARMv7

After executing the tests under the ARMv8 architecture, the results can be compared with the ARMv7 results.

The first step is to compare the results of the feature selection phase. The best features on ARMv7 are Bus Access for Memory Write, Stalls due to Write Buffer Full, Conditional Branchs, Bus Cycle, External Memory Request and L2D Writeback.

Two features are also selected on ARMv8, Bus Access for Memory Write and Conditional Branchs. Between these features, the first to reach the first place of importance in both architectures. Looking to the others features, it can be noted that some of the others features also appear as the best features of ARMv8, but are not selected because they have a huge correlation with others features. Stalls due to Write Buffer Full was not selected because Interlock Cycle Write Stage Stall due to Miss was selected instead, with a 90% of relation. L2D Writeback was not selected for having a relation of 90% with Bus Access for Memory Write. The Bus Cycle and External Memory Request doesn't appear on the ARMv8 list.

On the other hand, comparing the ARMv8 features with ARMv7. The best feature on ARMv8 are Bus Access for Memory Write, Interlock Cycle Write Stage Stall due to Store, Interlock Cycle Write Stage Stall due to Miss, Bus Access for Memory Load, L2 Cache Misses and Conditional Branchs. It's important to note that the difference in feature on ARMv8 is mainly because of memory access for load. This lead to more misses, stalls, L2D Cache Misses and Prefetch Linefill on cache, as showning on the feature selection phase.

There are some things that could cause this features to appear. First one is the difference on the size of the variable, and therefore, is necessary to load more times. Looking at the tasks source code, this is not true to this experiment.

Another problem that could cause this is a difference in the cache configuration. Both of the experiments, armv7 and armv8, are done in the same board, therefore the cache size is the same. Looking at the EPOS code for armv7 and armv8, the way the cache is configured is the same to both architectures, therefore a cache configuration is not the cause.

With this, the problem that cause this phenomenon was not explicit found, and we can infer that architectures are complex, and some problems couldn't be explained in a simple and easy way.

The next step is to compare the execution of the tasks. The execution trace of the task-sets 1 and 2 have a similar behavior to the ARMv7 results. The only difference was on task-set 3, where the ARMv8 couldn't decrease the frequency due to a problem with a bad warm-up, who lead the prediction to be overestimated.

The last step is to compare the data about the energy consumption. The ARMv7 show the best results compared to ARMv8. Considering the bandwidth task configuration, ARMv8 achieved an average reduction of 21% on the heavy configuration, while the ARMv7 achieved a reduction of 26%. In the low configuration, ARMv8 reduced by 16% and ARMv7 by 18%. In the mixed configuration, on the other hand, the ARMv8 shows the best results than ARMv8, with an average reduction of 17% compared to the 13% of ARMv7. Lastly, the random configuration, both architectures don't achieved any reduction in energy consumption on the tests.

6 CONCLUSION AND FUTURE WORK

Modern computer architectures are becoming more heterogeneous. This heterogeneity gives the architecture the capacity to increase the overall performance or control the energetic consumption. This energetic control uses techniques like Dynamic Voltage and Frequency Scaling (DVSF) that reduce the voltage and frequency of the processor based on some requisites. With this, some Machine Learning techniques are being used for energy optimization, using information provided by the architecture to give support to the optimization. This work uses as a base a non-intrusive Artificial Neural Network (ANN) that uses information that is built at runtime out of hardware performance counters, sensors, and OS variables to perform the energy optimization on the system. However, the aforementioned ANN was only tested in one platform, not having results on the behavior and performance of the Machine learning model and on the choices of counters on other platforms, since some performance counters are architecture dependent. This work provided an execution on ARMv7 and ARMv8 architectures, selecting features, executing task-sets, collecting data about the energy consumption and comparing the results.

The results on ARMv7 shows that the feature selection phase can lead to some different features compared to the HOFFMANN [5] work without interfering in the overall energy consumption. The execution trace of the tasks-sets described, as expected, the migrations and swaps of the tasks, converging to a scenario where the frequency is decreased. The tests show an overall reduction of energy consumption of 19.30%, with a maximum reduction of 26.32%.

The results on ARMv8 shows a different set of features on the feature selection phase when comparing to the ARMv7. The difference is mainly on features related to access for load, leading to cache misses and stalls. Even with these features, some others remain the same. The best feature in ARMv7 is also the best feature on ARMv8, and some others features also appear in both feature selection.

The execution trace of the tasks-sets 1 and 2 on ARMv8 shows the same scenario compared to ARMv7, performing migrations and swaps, and therefore, decreasing the frequency. The task-set 3, on the other hand, shows a scenario where the frequency couldn't be decreased due to a bad warm-up start of the ANN. This occurs when the offline training is not enough, leading to the prediction of the ANN be overestimated, and with this, the frequency is not decreased. The online training don't solve this problem because this training is only performed when the frequency is decreased.

Lastly, the energy consumption on ARMv8 shows an overall reduction of 13,37%, with a maximum reduction of 20,88%.

REFERENCES

[1] ARM. “ARM Cortex-A53 MPCore Processor”. In: (2016).

[2] Semiconductor Engineering. *Dynamic Voltage and Frequency Scaling (DVFS)*. https://semiengineering.com/knowledge_centers/low-power/techniques/dynamic-voltage-and-frequency-scaling/, Last accessed on 2022-07-11. 2022.

[3] Bhavishya Goel. “Per-core Power Estimation and Power Aware Scheduling Strategies for CMPs”. In: 2011.

[4] HAIDER, A.; TARIQ, U. U.; YONGJUN, Z.; XIAOJUN, Z.; LU, L. “Contention Energy-Aware Real-Time Task Mapping on NoC Based Heterogeneous MPSoCs”. In: *IEEE Access* 6 (2018), pp. 75110–75123. DOI: 10.1109/ACCESS.2018.2882941.

[5] HOFFMANN, J. L. C; FROHLICH, A. A. “Online Machine Learning for Energy-Aware Multicore Real-Time Embedded Systems”. In: (2021).

[6] HORSTMANN, L. P.; HOFFMANN, J. L. C; FROHLICH, A. A. “A Framework to Design and Implement Real-time Multicore Schedulers using Machine Learning”. In: (2019).

[7] Software/Hardware Integration Lab. *Application-Driven Embedded System Design Method*. https://lisha.ufsc.br/Application-Driven+Embedded+System+Design+Method, Last accessed on 2022. 2022.

[8] Software/Hardware Integration Lab. *Embedded Parallel Operating System*. https://epos.lisha.ufsc.br, Last accessed on 2022. 2022.

[9] Software/Hardware Integration Lab. *Software/Hardware Integration Lab*. https://lisha.ufsc.br, Last accessed on 2022. 2022.

[10] SANJEEV, D.; JAN, W.; MANOS, A.; MICHALIS, P.; FABIAN, M. “SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 20–38.

[11] Alin Suciu, Sebastian Banescu, and Kinga Marton. “Unpredictable Random Number Generator Based on Hardware Performance Counters”. In: *Digital Information Processing and Communications*. Ed. by Vaclav Snasel, Jan Platos, and Eyas El-Qawasmeh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 123–137. ISBN: 978-3-642-22410-2.

[12] B. Venkatesh and J. Anuradha. “A Review of Feature Selection and Its Methods”. In: *Cybernetics and Information Technologies* 19.1 (2019), pp. 3–26. DOI: doi:10.2478/cait-2019-0001. URL: https://doi.org/10.2478/cait-2019-0001.

[13] Ruibin Xu, Daniel Mossé, and Rami Melhem. “Minimizing expected energy consumption in real-time systems through dynamic voltage scaling”. In: *ACM Trans. Comput. Syst.* 25 (Dec. 2007). DOI: 10.1145/1314299.1314300.

[14] M. ZAHRAN. “Heterogeneous computing: Here to stay”. In: *Queue* 14.no. 6 (2016), pp. 40:31–40:42.

APPENDIX A – SBC ARTICLE

Evaluation of a machine learning model to predict the performance of energy-aware, real-time, multicore embedded systems across multiple architectures

José Luiz de Souza

[1]Curso de Bacharelado em Ciência da Computação - Departamento de Informática e Estatística – Universidade Feraderal de Santa Catarina (UFSC) - 88040-900 Florianópolis - SC - Brasil

joseloolo26@gmail.com

Abstract. *Modern computer architectures are becoming more heterogeneous and increasing the energy consumption. This work uses as a base an Artificial Neural Network (ANN) that uses information that is built at runtime out of hardware performance counters, sensors, and OS variables to perform the energy optimization on the system. However, the aforementioned ANN was only tested in one platform, not having results on the behavior and performance of the Machine learning model and on the choices of counters on other platforms. In this context, the objective of this work is to test the ANN solution on the ARMv7 and ARMv8 architectures, evaluating the model and the chose of the hardware performance counters.*

1. Introduction

Modern computer architectures are becoming more heterogeneous [ZAHRAN 2016]. This heterogeneity takes place through a wide variety of different processors inside the same hardware, where they can differ from each other through the use, technology or energy management techniques. This gives the architecture the capacity to increase the overall performance, through the use of features like SIMD units and application-specific accelerators or technologies like hyper-threading. All of this is usually connected by some network-on-chip [HAIDER 2018].

Along with the growth of the complexity of these systems, the power consumption grows as well, making energy management an important topic to deal with. The architecture can use some energy management techniques like Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management(DPM) to decrease the energy consumption of chips, scaling down the voltage and frequency based on the targeted performance requirements of the running application [Engineering 2022].

These techniques can be used to reduce the energy consumption in real-time based systems, where many of them are battery-operated embedded devices. Real-time based systems are designed to execute applications and respond to events within a predictable time, and therefore, using energy management techniques in these systems mean to reduce the energy consumption as much as possible without interfere on the real-time system applications constraints [Xu and Mossé 2007].

To provide runtime energy optimization, some Machine Learning (ML) techniques are being used. The ML use traces provided by the architecture and process them into useful information to support the optimization. However, ML-based optimization in runtime can have problems dealing with real-time base systems because the ML-techniques could end up violating the performance and temporal constraints of the real-time applications.

[HOFFMANN 2021] presents a non-intrusive, ANN-based runtime energy optimizer that was designed and implemented considering the application's constraints. The model is able to predict the performance of tasks in lower frequency levels and safely optimize real-time embedded systems' power saving operations. This is done by taking execution traces that are built at runtime out of hardware performance counters, sensors, and OS variables of individual tasks as input to predict their performance in case the core where they are running is subjected to DVFS.

While the aforementioned work predicts and safely optimizes power saving operations, there is an assumption about the chosen hardware performance counters used as input by the ANN. In [HOFFMANN 2021], the performance counters were analyzed and selected through three different techniques to filter and acquire only the more relevant counters that exist on the target platform, that was a Cortex-A53 processor, running in an ARMv7 mode where there are in total of 54 available performance counters.

However, only the above-mentioned platform was used for testing and there is no guarantee that the chosen performance counters remain the most relevant option on other platforms nor that the feature selection methods manage to capture the best performance counters due to the possible unpredictability that exists in the counters. In fact, not all hardware performance counters remain consistent across multiple architectures. Some performance counters are architecture dependent and vary across different processor architectures and may also change with processor enhancement (e.g. cache and TLB accesses) [SANJEEV 2019].

Therefore, the objective of this work is to evaluate the performance of the [HOFFMANN 2021] machine learning model running on other platforms and verify the reliability of feature selection methods when tested on other architectures. This will be done on the architectures ARMv7 and ARMv8, mediated by the Embedded Parallel Operating System (EPOS - https://epos.lisha.ufsc.br/).

2. EPOS

Embedded Parallel Operating System [Lab 2022b], or simply EPOS, is an operating system project developed at the Software/Hardware Integration Lab (LISHA) [Lab 2022c] and aims to automate the development of embedded systems.

EPOS relies on the Application-Driven Embedded System Design (ADESD) [Lab 2022a] method that use a strategy to systematically construct application-oriented operating systems as arrangements of adaptable software components. This feature enables the development of software/hardware components that can be automatically adapted to fulfill the requirements of particular applications.

Furthermore, EPOS has support for many architectures and platforms, being able to run in both of the target architectures used through this work, ARMv7 and ARMv8. For each architecture supported, some architecture dependent components are available too, like the Memory Management Unit and Performance Monitor Unit.

3. Performance Monitor Unit

Performance Monitor Unit (PMU) is an architectural component available in the most of the modern processors that allow the capture and record of events that occur in the processor. The monitoring of these events can be used to evaluate how the code interacts with the hardware, what resources they used and how many clock cycles the code took to execute, helping to identify, for instance, performance bottlenecks and opportunities to speed up the application [Goel 2011].

These events are useful, but there is usually a hardware limitation in how many events can be tracked at the same time. These events are monitored through counters, also called channels, and the available number of these channels varies between architectures, but they are limited and only a few events can be tracked at the same time. Due to the limit in the number of channels, the available information at run-time is also limited and therefore is necessary to select the most crucial events to be monitored.

4. Sensors and OS Variables

Sensors and OS Variables are another sources of information available in modern hardware and operating systems. The sensors collect information about the hardware, and therefore collect information like Temperature and Energy Consumption.

SO Variables, on the other hand, collect information about the system, and therefore, are dependent on the operating system that is used. The EPOS have several events of the system that can be collect, some examples are CPU Execution Time, Deadline Misses and Running Thread. These events can be used to understand, track back and to optimize the performance of the system.

5. ARM Architecture

ARM is a family of RISC architectures for computer processors. The ARM is designed to be used in various environments, from portable, battery-powered devices to smartphones, desktops and servers. This is possible due to their low costs, power consumption and heat generation, when compared with their competitors. The two most major version of the ARM family are ARMv7 and ARMv8.

- ARMv7 is a 32-bit architecture, with 32-bit wide registers.
- ARMv8 is a 64-bit architecture, with 64-bit wide registers.

Although the ARMv8 is a 64-bit architecture, it has a backward compatibility with the ARMv7 architecture. This is done by having two execution states: AArch64 and AArch32. The AArch64 is a normal ARMv8 state that uses 64-bit registers. The AArch32 state uses 32-bit registers, having compatibility with the ARMv7 architecture, and in addition supports some features included in the AArch64 state.

6. Feature Selection

Feature selection is a process used to reduce the number of input variables in a model, removing non-informative or redundant variables. This is done to reduce the computation cost, improve the performance of the model, reduce the redundant variables and avoid the curse of Dimensionality.

The relevance of a feature is measured not based on its value but on the characteristics of the data and the relationship with other features. This measure is done by feature selection methods that are classified into three types, Filter, Wrapper and Embedded Methods[Venkatesh and Anuradha 2019].

6.1. Pearson's Correlation Coefficient

Pearson's Correlation Coefficient (PCC) is a measure to detect a linear relation between two continuous variables. In other words, PCC indicates the effect of change in one variable when the other changes. The equation **??** calculate the PCC between an independent variable x and a dependent variable y [Venkatesh and Anuradha 2019]:

6.2. Information Gain Ratio

Information Gain Ratio is a statistical measure based on the entropy of the data, which indicates the uncertainty in the data. A low entropy indicates the data is uniform, and a high entropy indicates the opposite.

The information gain ratio is calculated as the entropy of the data minus the conditional data entropy, divided by the intrinsic value of the feature, to reduce the biases of features with many distinct values.

6.3. Lasso CV

The Least Absolute Shrinkage and Selection Operator with Cross Validation is a linear regression method that performs L1 penalty. If two features are linearly correlated, their presence will increase the value of the cost function and lasso regression will try to shrink the coefficient of the less important feature to zero. With this, lasso regression will automatically select the more useful features.

7. Environment

The execution of the experiment involves to collect performance data at run-time in a real-time system. However, some characteristics in this collection need to be fulfilled by the operating system. First, a non-intrusive data acquisition method is necessary in order to not interfere with real-time applications constraints. Second, the data collection needs to be done in a strict way to avoid inaccurate measurements and contamination due to events from other processes. Lastly, besides the performance data that is provided by the Performance Monitor Unit (PMU), is necessary a way to collect information about the use of the system (e.g. CPU execution time, Thread execution time and Deadline misses) to distinguish tasks and be able to make more accurate assumptions while having a guarantee that the system is working as expected.

Therefore, the Embedded Parallel Operating System (EPOS) was chosen due to the fact that it fulfill all the above-mentioned characteristics.

7.1. Architectures and Platforms

The experiments are executed in two architectures, ARMv7 and ARMv8. The experiment in ARMv7 was conducted to compare the results described in [HOFFMANN 2021] work and draw conclusions.

The architectures and platforms used are described below:

- **ARMv7** architecture running on a Raspiberry Pi 3b+ with a ARMv8 Cortex-A53 processor, executing in ARMv7 compatibility mode.
- **ARMv8** architecture running on a Raspiberry Pi 3b+ with a ARMv8 Cortex-A53 processor.

The Cortex-A53 has 6 channels, allowing it to collect 6 performance counters at the same time.

7.2. Task-Sets

To collect representative and meaningful data over the performance counters in a multicore real-time embedded system, we need tasks with distinct performance behaviors running in parallel to simulate a real use of the system. To achieve this behavior, 3 types of tasks are selected: a memory bound task that focus on memory access; a cpu bound task that focus on mathematical operation and uses the ALU; and a disparity task that simulates a real workload of embedded systems.

- **Bandwidth** is a memory-bound, performing constantly read and write operations on a data structure with at least the size of the L2 Cache defined by the running platform. This task can be also divided into four types, low use of memory, high use of memory, an alternate between low and high use of memory and random use of memory.
- **CPU Hungry** is a cpu-bound task that implements an Iterative Fibonacci how to execute a loop with mathematical operations constantly using the ALU.
- **Disparity map** is a task that represents a real workload task of embedded systems.

To simulate resources sharing contention between the tasks, the tasks are combined to run in parallel. Two combinations of parallel execution are defined in [HOFFMANN 2021] and replicated here: One composed of a Memory Bound, CPU Bound and Disparity and another composed of a CPU Bound and 2 Disparity. Overall, the configuration of the task-set is presented in table 1.

This task-set is well-defined to have a heavy sharing contention and therefore is used as a main task-set to collect performance data for feature selection.

Table 1. Task-Set 1 configuration

Task-Set 1			
Task	**Period/WCET (in ms)**	**CPU**	**ID**
Bandwidth	500/100	1	1
CPU Hungry	500/100	2, 3	4, 5
Disparity	500/100	1, 2, 3	2, 3, 6

Besides the task-set 1, two task-sets are defined, and the configuration are described in table 2 and table 3.

7.3. Equipament

To collect information about the energy consumption when running the task-sets, the KCX-017 USB Power Meter was used.

Table 2. Task-Set 2 configuration

Task-Set 2			
Task	**Period/WCET (in ms)**	**CPU**	**ID**
Bandwidth	250/50	1	1
CPU Hungry	250/35, 125/20	2, 3	4, 5
Disparity	500/100, 1000/200, 250,100	1, 2, 3	2, 3, 6

Table 3. Task-Set 3 configuration

Task-Set 3			
Task	**Period/WCET (in ms)**	**CPU**	**ID**
Bandwidth	100/10, 100/5, 1000/100	1, 2, 3	1, 2, 7
CPU Hungry	100/30, 250/60	3, 3	4, 6
Disparity	1000/400, 500/100	2, 3	3, 5

8. Data Capture

The first step of the experiment is to capture meaningful data that represents a heavy use of the system and resource sharing contention between the tasks. To fulfill this objective, task-set 1, described in table 1 was used.

In this step, all the performance counters and some SO events are collected. Due to the hardware limitation on the number of channels of the PMU, the number of performance counters that can be collected at the same time is limited. This implies running the application several times to collect all necessary data. For example, on Raspiberry PI 3b+, it's necessary to collect data 9 times, because this platform has 54 performance counters but only 6 can be collected at the same time.

Besides the performance counters, the SO events also need to be collected. In this experiment, the SO events collected are CPU Usage, Thread Usage and Running Thread. They are necessary to collect information about the running task at the moment that the collection was done. This gives information about the cpu usage, thread usage and task id, used to know what the data collected refer to. The capture of the data is done with the system on the maximum frequency and for 180 seconds in each collection.

After the collection, all the logs of the system are filtered and cleared, removing all unnecessary information and merging all the information collected in chronological order.

9. Feature Selection

After the data capture, the next step is to select the most important features within all the performance counters that the platform offers. The number of features that will be selected depends on the limitation of the PMU channels of the processor.

To achieve this objective, 3 methods for feature selection are used: Pearson Correlation Coefficient (PCC), Lasso CV and Info Gain Ratio. The mentioned feature selection methods give distinct results over different features. With this, we used a combination of the methods results to select the features.

The PCC is a measure to detect a linear relation between two continuous variables, being the main methods used in the decision of the feature selection. The other 2 algorithms will be used to give extra information and to decide in case there are some features with close relation and if it's necessary to select one of them.

As mentioned above, the PCC detects a relation between two variables, therefore, the relation between all the performance counters will be calculated, but the relation between the performance counters and the Thread usage will be our main concern. A feature that has a great relationship with the thread usage indicated that the feature is a good indication and therefore, a good feature to be selected.

For instance, two features can be on the top of the best features list when compared with the thread usage but when compared between each other, the two features can be highly correlated, meaning that if we selected the two features, only one will give a meaningful value. With this, we need to verify the correlation between the top features to eliminate the redundant variables. The Lasso CV and Info Gain Ratio are used to decide what feature will be selected.

10. IA Training

After the feature selection phase, we already know what are the best features to be used to train the IA. The online artificial neural network model in this work is used to predict the utilization of the task if the frequency is reduced. With this, we can predict the total utilization of the cores and decide if it's possible to scale the frequency of the CPU using Dynamic Voltage and Frequency Scaling (DVFS), reducing the overall energy consumption of the chip. The configuration of the ANN is the same used on [HOFFMANN 2021]. The model uses a learning rate of 0.35 and an average accuracy of 97% considering the validation data-set.

Even though the ANN used to run the experiments is trained online, offline training is necessary beforehand to avoid cold start issues. This training is done using two datasets of data, one for training and the other for validation. The training sample is got by running the task-set 1, described in the table 1 and collecting data of the selected performance counters under all levels of frequency scaling. With this, we collect the behavior and performance counter of the system under each frequency level. The training data is collected using task-set 2, described in the table 2 under the same circumstances.

11. Energy Analysis

The last step of the experiment is to measure the energy consumption of the system when under the energy optimizer, which scale the frequency based on the use of the system. Not only the DVFS is used, but also a balanced workload using migration of tasks, improving, even more, the energy more the energy consumption.

To capture the energy consumption of the system, the task-set 1, task-set 2 and task-set 3 are executed, with and without the ANN prediction and DVFS actuation. The data is collected by the equipment KCX-017 USB Power Meter. Besides this, each task-set has a Bandwidth task, that can be divided into four types, low use of memory, high use of memory, an alternate between low and high use of memory and random use of memory. The task-sets are executed for each type of bandwidth task as well.

The first step is to capture how much energy the idle system consumes, running the application without tasks, in the maximum frequency. Next, the consumption of all the task-sets is collected, running on the maximum frequency. After that, the energy consumption of the task-sets with the ANN prediction is measured. With this, we subtract the energy consumption of the idle system and compare the energy consumption.

12. ARMv7 Results

12.1. Feature Selection

In the feature selection phase, the results of the most relevant features are described in Figure 1. They are ordered by the results of the PCC method.

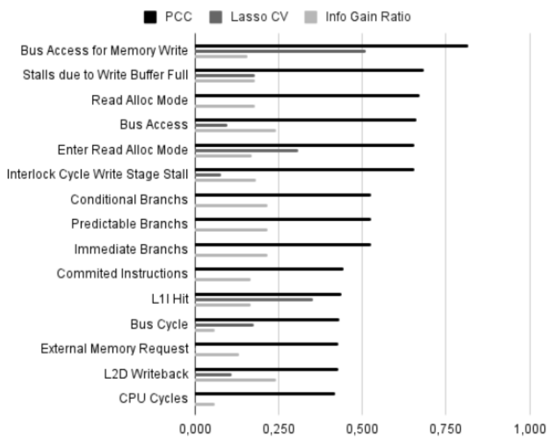

Figure 1. Ranking results for the most relevant features on ARMv7

The first feature, Bus Access for Memory Write will be selected for having the best results in both PCC and Lasso CV methods. This feature also has a relation of 92% with the feature Bus Access, which has a high result in PCC as well. Due to this, only the first one will be selected.

Following the list, we have the feature Stalls due to Write Buffer Full. This feature has a high relation with other three features, which are also in the list: Read Alloc Mode (94%), Enter Read Alloc Mode (96%) and Interlock Cycle Write Stage Stall(96%). Between these features, only Stalls due to Write Buffer Full will be selected for having a higher value on PCC and having significant values on Lasso CV and Info Gain.

The next feature is Conditional Branchs. This feature has a high relation with the features Predictable Branchs (99%), Immediate Branchs (99%), Commited Instructions (91%) and L1I Cache Hit (91%). With this, only the Conditional Branchs is selected for having the highest PCC among the cited features.

Bus Cycle is the next feature, having significant values on PCC and Lasso CV, and therefore was selected. This feature has a high PCC relation with the feature CPU Cycles (98%). With this, only the Bus Cycle is selected for having the highest PCC among the cited features.

External Memory Request also was selected for having a high value in PCC and Info Gain Ratio. This feature doesn't have a high PCC relation with other feature of the list.

The last feature is L2D Writeback, having good results in all three methods. This feature also doesn't have a high PCC relation with other feature of the list.

With this, the selected features are: Bus Access for Memory Write, Stalls due to Write Buffer Full, Conditional Branchs, Bus Cycle, External Memory Request and L2D Writeback.

12.2. Energy Optimizer

After the feature selection phase, the execution of the task-sets 1, 2 and 3 described in section 7 are performed with and without the ANN optimizer, collecting data about the energy consumption of the tasks. Under the ANN, each performance counter has a weight associated. After the offline training, each task-set is executed several times to perform online training to find the best weights that the IA can use to predict more precisely the migrations, swaps and frequency scaling of a specific task-set. After this, we use this weight to execute the task-sets to collect information about energy consumption.

All the task-sets have Bandwidth tasks, which are divided into four types: heavy, light, mixed and random. Each one has a different memory consumption, for a maximum to a minimum use, making the simulation of a real use of the system more accurate. In this section, all graphs are under heavy use of memory. In addition, all data collected of energy consumption are done in executions of 180 seconds.

12.2.1. Task-set 1

The first task-set to be executed is described in table 1. In this task-set, the frequency is decreased from 1,2GHz to 0,6GHz, the minimum supported on the platform, without the necessity of migrations.

The actuation of energy optimizer in this task-set resulted in a reduction on energy consumption of 23%. The reduction in energy consumption was 37% when executed with a heavy use of memory. On the other hand, with random use of the memory, the energy consumption didn't change.

12.2.2. Task-set 2

The second task-set to be executed is described in table 2. Initially, this task-set has an imbalance in the use of the cores, decreasing the frequency to a maximum of 0,7GHz where the CPU3 has a use of almost 100% while the CPU1 has a use of nearly 60%. With this, the energy optimizer performs a migration of tasks from CPU3 to CPU1, described on the graph, leading to a more balanced use of the platform, decreasing the frequency to a minimum of 0,6GHz.

The energy consumption of this task-set results in a reduction of 10% on average. The energy consumption was reduced by 15% and 18% with heavy and light bandwidth configuration, respectively.

12.2.3. Task-set 3

The third task-set to be executed is described in table 3. This task-set is the most complex among the task-sets described in section 7 and takes several executions to find weights to configure the ANN to converge to a frequency. The execution of the task-set 3 first is using default values for the migration weights. In this execution, the tasks didn't accomplish a stability on the frequency, leading to several migrations and swaps.

After several executions to find the migrations weights that fit the task-set, the frequency was decreased and found stability on 0,9GHz after 5 migrations.

The energy consumption of this task-set results in a reduction of 10% on average. The energy consumption was reduced by 26% with heavy bandwidth configuration.

12.3. Energy Consumption

After considering all task-sets, the overall energy consumption reduction is shown in Figure 2. The figure shows the energy reduction of all task-sets when considering all bandwidth task configurations.

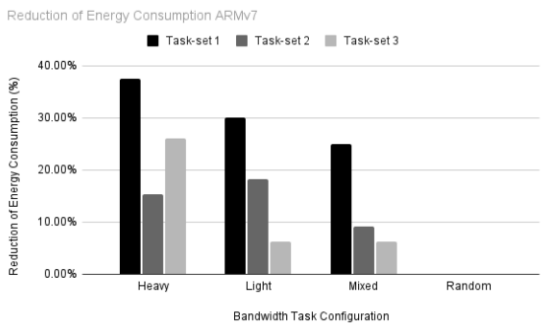

Figure 2. ARMv7 - Energy Consumption

The heavy configuration on the bandwidth task was the one that shows the best results, with an average reduction of 26%. The next is the light configuration, which has a low use of the memory, achieve an average reduction in the consumption of 18%. The next, mixed configuration, has an average reduction of 13%. The last configuration, the random, which makes random use of memory, doesn't achieve a reduction in energy consumption on the tests.

On average, the reduction of energy consumption considering all the task-sets and all the bandwidth task configurations was 19%.

12.4. Comparison with the original work

After executing the tests under the ARMv7 architecture, the results can be compared with the results described in HOFFMANN's [HOFFMANN 2021] work.

The first step is to compare the results of the feature selection phase. The features selected by HOFFMANN are Bus Access for Memory write operations, Stalls due to Write Buffer Full, L2D Writeback, Immediate Branches, CPU Cycle Count, and L1 Cache Hits. Between the features, Bus Access for Memory write operations, Stalls due to Write Buffer Full and L2D Writeback are selected in the feature selection phase of this work as well. The Immediate Branch was changed to Conditional Branches, which has a correlation of 99% with Immediate Branches in PCC. The CPU Cycle was changed to Bus Cycles, which has a correlation of 98% in PCC. The last feature is different. This work selected the feature called External Memory Request while HOFFMANN selected a feature L1 Cache Hits. While the features are different, there are some relation between them. The first, External Memory Request, has a high relation (92%) with another similar feature, L2 Cache Hits. Therefore, even if the feature is different, there are semantically relations.

The next step is the execution of the tasks with the energy optimizer and collect data about the energy consumption. The execution trace of the task-sets has a similar behavior to the HOFFMANN's work. The only difference was on task-set 3, where the energy optimizer reduced the frequency to 0,8GHz, while in this work was only 0,9GHz. This made a difference in the average energy consumption of the task-set 3 by 5%.

Considering the bandwidth task configuration, this work shows the best results in the heavy and low configurations, but a worse result in the mixed configuration. Comparing the results, this work achieves a decrease of 2% more on the heavy configuration and 1% more on the low configuration. On the other side, the difference in mixed configuration was 7%. The overall decrease of energy consumption of HOFFMANN was 21.38% while this work achieve an overall reduction of 19.30%.

13. ARMv8 Results

13.1. Feature Selection

In the feature selection phase, the results of the most relevant features are described in Figure 3. They are ordered by the results of the PCC method.

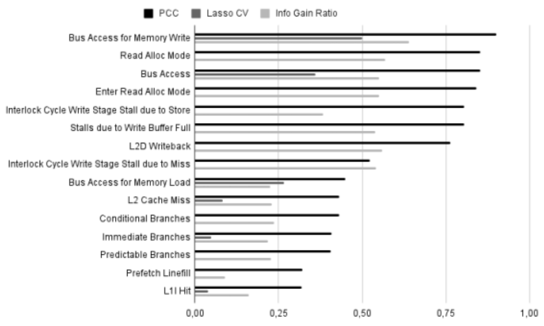

Figure 3. Ranking results for the most relevant features on ARMv8

The first feature, Bus Access for Memory Write due to Store will be selected for having the best results in the three methods used to select the best features: PCC, Lasso CV and Info Gain methods. This feature has a high relation with others features, being the most important the Read Alloc Mode (96%), Bus Access (95%), Enter Read Alloc Mode (95%) and L2 Writeback (91%). Between these features, only Bus Access for Memory Write due to Store will be selected for having the best values over all the others features.

Following the list, we have the feature Interlock Cycle Write Stage Stall. This feature has a high value in both PCC and Info Gain methods and has a high relation with the feature Stalls due to Write Buffer Full (97%). Between these two features, only the Interlock Cycle Write Stage Stall will be selected for having a high value on PCC and Info Gain. This feature also has a high relation with the others features discussed before, but it's still selected for having a high value on Info Gain and for being a complementary feature, that is, stalls when writing.

The next feature is Interlock Cycle Write Stage Stall due to Miss, having a high value on PCC and Info Gain, being therefore, selected. This feature doesn't have a high PCC relation with otehrs features of the list.

Bus Access for Memory Load is the next feature and also was selected. This feature has a high value in both PCC and Lasso CV. This feature doesn't have a high PCC relation with others features of the list.

Following the list, we have the feature L2 Cache Misses, who also has a high value in PCC and Lasso CV. This feature doesn't have a high PCC relation with others features of the list.

The last feature is Conditional Branchs. This feature has a high relation with the features Immediate Branchs (98%), Predictable Branchs (98%) and L1I Cache Hit (93%). With this, only the Conditional Branchs is selected for having the highest PCC among the cited features.

With this, the selected features are: Bus Access for Memory Write due to Store, Interlock Cycle Write Stage Stall, Interlock Cycle Write Stage Stall due to Miss, Bus Access for Memory Load, L2 Cache Misses and Conditional Branchs.

13.2. Energy Optimizer

After the feature selection phase, the execution of the task-sets 1, 2 and 3 described in section 7 are performed with and without the ANN optimizer, collecting data about the energy consumption of the tasks. After the offline training, each task-set is executed several times to find the best migrations weights to predict more precisely the migrations and swaps. After this, we use this weight to execute the task-sets to collect information about energy consumption.

All the task-sets have Bandwidth tasks, with are divided into four types: heavy, light, mixed and random. Each one has a different memory consumption, for a maximum to a minimum use, making the simulation of a real use of the system more accurate. In this section, all graphs are under heavy use of memory. In addition, all data collected of energy consumption are done in executions of 180 seconds.

13.2.1. Task-set 1

The first task-set to be executed is described in table 1. In this task-set, the frequency is decreased from 1,2GHz to 0,6GHz, the minimum supported on the platform, without the necessity of migrations.

The energy consumption of this task-set results in a reduction of 23% on average. The reduction in energy consumption was 33% when executed with a heavy use of memory. On the other hand, with random use of the memory, the energy consumption didn't change.

13.2.2. Task-set 2

The second task-set to be executed is described in table 2. Initially, this task-set has an imbalance in the use of the cores, decreasing the frequency to a maximum of 0,7GHz where the CPU3 has a use of almost 100% while the CPU1 has a use of nearly 60%. With this, the energy optimizer performs a migration of tasks from CPU3 to CPU1, described on the graph, leading to a more balanced use of the platform, decreasing the frequency to a minimum of 0,6GHz.

The energy consumption of this task-set results in a reduction of 8% on average. The energy consumption was reduced by 11% in both heavy and light bandwidth configuration.

13.2.3. Task-set 3

The third task-set to be executed is described in table 3. This task-set is the most complex among the task-sets described in section 7. On the first try, there is a discrepancy on the use of the CPU 3 compared to the others CPUs, therefore, a migration is performed, sending one task from CPU3 to CPU1. Even with this migration, the prediction of the CPU3 is still very high, therefore, the frequency is not decreased. This problem occurs on the way the offline and online learning is performed.

The offline training is performed to have a warm-up start, allowing bypass an initial stage of online training. This training is done with information of the Task Set 1 and not the current task set. This happens because the Task Set 1 have a configuration for different scenarios, where we have different tasks running in parallel. Even though the task set 3 run the same tasks, there are some differences on the number of tasks and what tasks are running in parallel, therefore, not every scenario can be covered.

To complement this offline training, an online training is also performed, to make sure the current task set configuration is being predicted correctly, and this training is executed when we have a variation on the frequency.

This lead the problem on the task set 3. In this scenario, the offline training is not enough to predict correctly the configuration of the task set 3, leading to a bad warm-up. Even though migration was performed, this doesn't solve the problem. The online training also was never performed because we never decreased the frequency, making the frequency stuck on the 1,2GHz.

13.3. Energy Consumption

After considering all task-sets, the overall energy reduction is shown in Figure 4. The figure shows the energy reduction of the task-sets 1 and 2 when considering all bandwidth task configurations. The Task Set 3 was not included due to the problem where the frequency was not decreased.

The heavy configuration on the bandwidth task was the one that shows the best results, with an average reduction of 21%. The next is the light configuration, which

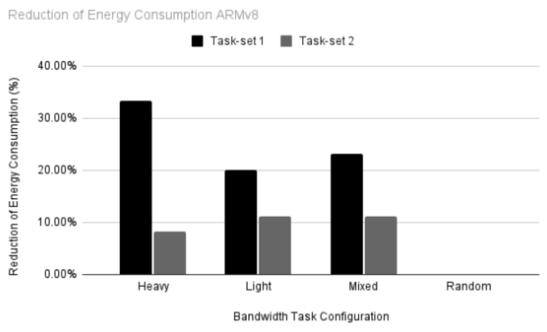

Figure 4. ARMv8 - Energy Consumption

has a low use of the memory, achieve an average reduction in the consumption of 16%. The next, mixed configuration, has an average reduction of 17%. The last configuration, the random, which makes random use of memory, doesn't achieve a reduction in energy consumption on the tests.

On average, the reduction of energy consumption considering all the task-sets and all the bandwidth task configurations was 13%.

13.4. Comparison with the ARMv7

After executing the tests under the ARMv8 architecture, the results can be compared with the ARMv7 results.

The first step is to compare the results of the feature selection phase. The best features on ARMv7 are Bus Access for Memory Write, Stalls due to Write Buffer Full, Conditional Branchs, Bus Cycle, External Memory Request and L2D Writeback.

Two features are also selected on ARMv8, Bus Access for Memory Write due to Store and Conditional Branchs. Between these features, the first to reach the first place of importance in both architectures. Looking to the others features, it can be noted that some of the others features also appear as the best features of ARMv8, but are not selected because they have a huge correlation with others features. Stalls due to Write Buffer Full was not selected because Interlock Cycle Write Stage Stall due to Miss was selected instead, with a 90% of relation. L2D Writeback was not selected for having a relation of 90% with Bus Access for Memory Write due to Store. The Bus Cycle and External Memory Request doesn't appear on the ARMv8 list.

On the other hand, comparing the ARMv8 features with ARMv7. The best feature on ARMv8 are Bus Access for Memory Write due to Store, Interlock Cycle Write Stage Stall, Interlock Cycle Write Stage Stall due to Miss, Bus Access for Memory Load, L2 Cache Misses and Conditional Branchs. It's important to note that the difference in feature on ARMv8 is mainly because of memory access for load. This lead to more misses, stalls, L2D Cache Misses and Prefetch Linefill on cache, as showning on the feature selection phase.

There are some things that could cause this features to appear. First one is the difference on the size of the variable, and therefore, is necessary to load more times. Looking at the tasks source code, this is not true to this experiment.

Another problem that could cause this is a difference in the cache configuration. Both of the experiments, armv7 and armv8, are done in the same board, therefore the cache size is the same. Looking at the EPOS code for armv7 and armv8, the way the cache is configured is the same to both architectures, therefore a cache configuration is not the cause.

With this, the problem that cause this phenomenon was not explicit found, and we can infer that architectures are complex, and some problems couldn't be explained in a simple and easy way.

The next step is to compare the execution of the tasks. The execution trace of the task-sets 1 and 2 have a similar behavior to the ARMv7 results. The only difference was on task-set 3, where the ARMv8 couldn't decrease the frequency due to a problem with a bad warm-up, who lead the prediction to be overestimated.

The last step is to compare the data about the energy consumption. The ARMv7 show the best results compared to ARMv8. Considering the bandwidth task configuration, ARMv8 achieved an average reduction of 21% on the heavy configuration, while the ARMv7 achieved a reduction of 26%. In the low configuration, ARMv8 reduced by 16% and ARMv7 by 18%. In the mixed configuration, on the other hand, the ARMv8 shows the best results than ARMv8, with an average reduction of 17% compared to the 13% of ARMv7. Lastly, the random configuration, both architectures don't achieved any reduction in energy consumption on the tests.

14. Conclusion

Modern computer architectures are becoming more heterogeneous and increasing the energy consumption. This work uses as a base an Artificial Neural Network (ANN) that uses information that is built at runtime out of hardware performance counters, sensors, and OS variables to perform the energy optimization on the system. However, the aforementioned ANN was only tested in one platform, not having results on the behavior and performance of the Machine learning model and on the choices of counters on other platforms. In this context, the objective of this work is to test the ANN solution on the ARMv7 and ARMv8 architectures, evaluating the model and the chose of the hardware performance counters.

The results on ARMv7 shows that the feature selection phase can lead to some different features compared to the HOFFMANN [HOFFMANN 2021] work without interfering in the overall energy consumption. The execution trace of the tasks-sets described, as expected, the migrations and swaps of the tasks, converging to a scenario where the frequency is decreased. The tests show an overall reduction of energy consumption of 19.30%, with a maximum reduction of 26.32%.

The results on ARMv8 shows a different set of features on the feature selection phase when comparing to the ARMv7. The difference is mainly on features related to access for load, leading to cache misses and stalls. Even with these features, some others remain the same. The best feature in ARMv7 is also the best feature on ARMv8, and some others features also appear in both feature selection.

The execution trace of the tasks-sets 1 and 2 on ARMv8 shows the same scenario compared to ARMv7, performing migrations and swaps, and therefore, decreasing the frequency. The task-set 3, on the other hand, shows a scenario where the frequency couldn't be decreased due to a bad warm-up start of the ANN. This occurs when the offline training is not enough, leading to the prediction of the ANN be overestimated, and with this, the frequency is not decreased. The online training don't solve this problem because this training is only performed when the frequency is decreased.

Lastly, the energy consumption on ARMv8 shows an overall reduction of 13,37%, with a maximum reduction of 20,88%.

References

Engineering, S. (2022). Dynamic voltage and frequency scaling (dvfs). `https://semiengineering.com/knowledge_centers/low-power/techniques/dynamic-voltage-and-frequency-scaling/`, Last accessed on 2022-07-11.

Goel, B. (2011). Per-core power estimation and power aware scheduling strategies for cmps.

HAIDER, A. (2018). Contention energy-aware real-time task mapping on noc based heterogeneous mpsocs. *IEEE Access*, 6:75110–75123.

HOFFMANN, J. L. (2021). Online machine learning for energy-aware multicore real-time embedded systems.

Lab, S. I. (2022a). Application-driven embedded system design method. `https://lisha.ufsc.br/Application-Driven+Embedded+System+Design+Method`, Last accessed on 2022.

Lab, S. I. (2022b). Embedded parallel operating system. `https://epos.lisha.ufsc.br`, Last accessed on 2022.

Lab, S. I. (2022c). Software/hardware integration lab. `https://lisha.ufsc.br`, Last accessed on 2022.

SANJEEV, D. (2019). Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. pages 20–38.

Venkatesh, B. and Anuradha, J. (2019). A review of feature selection and its methods. *Cybernetics and Information Technologies*, 19(1):3–26.

Xu, R. and Mossé (2007). Minimizing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Trans. Comput. Syst.*, 25.

ZAHRAN, M. (2016). Heterogeneous computing: Here to stay. *Queue*, 14(no. 6):pp. 40:31–40:42.