Making FreeRTOS Pervasive Systems Learn to Select Energy Saving Technique for Mixed Taskset

Deepak Ramegowda, Man Lin

Department of Computer Science, St. Francis Xavier University

Abstract

Many cyber-physical systems or pervasive systems are built with small devices which are powered by batteries. Energy-saving problem is essential for these systems. Software-based dynamic voltage and frequency scaling (DVFS) has been a powerful technique for energy saving. There has been little work using machine learning to derive a software controller for small devices to save energy. This paper presents a methodology for making FreeRTOS-based embedded systems learn to select energy-saving techniques for mixed taskset. Simulation has shown that DVFS techniques, along with reinforcement learning, save energy while handling periodic real-time tasksets. However, algorithm evaluations on a real platform for pervasive systems are seldom done. We extend the hybrid DVFS approach using reinforcement learning to handle a mixed taskset on embedded platforms and evaluate the algorithm on real devices. We choose FreeRTOS as the real-time operating system for the embedded platform as it is one of the most popular RTOS, and it is freely available. To the best of our knowledge, we are the first to consider experimenting with DVFS approach in a FreeRTOS framework with reinforcement learning on real-time devices. The implementation and evaluation of the proposed method EHYMT-FreeRTOS are carried out on an ARM Cortex-M7 (STM32H7B3I-DK). Results show that we can successfully make the system select suitable deadline-guaranteed DVFS technique while focusing on energy saving. Our implementation can potentially lead to a number of innovative AI-based DVFS algorithms targeted on FreeRTOS on real embedded systems.

Keywords

Pervasive Systems, Machine Learning, Energy-Saving, Mixed Taskset, FreeRTOS.

1. Introduction

Many cyber-physical systems or pervasive systems are built with small devices which are powered by batteries. Energy-saving problem is essential for these systems. The latest research shows that the processor accounts for 18% to 30% of total power consumption. On newer fabrication technologies, it can even transcend 50% for CPU-intensive workloads [1]. A recent experiment shows that sleep modes do not restore the energy lost at operating at the higher voltage, and only voltage scaling can save energy when the device is not shut down. That is, when not shutdown, to save energy (and therefore battery life), the operating voltage (and corresponding operating frequency) must be lowered [2]. Software-based dynamic voltage and frequency scaling (DVFS) has been used as a powerful technique for energy saving [3] [4] [5] [6], [7], [8].

Previous research shows that there is no single DVFS algorithm that is optimal for all applications on a system, and a machine learning method was proposed to select the best DVFS technique suitable for the current condition [9].

Most of the implementation has considered experimenting with DVFS techniques on simulators. There has been little work using machine learning to derive a software controller for small devices to save energy.

In this paper, we implement a learnable, thus adaptable software controller to scale the frequency on real embedded systems with the support of market-leading real-time operating systems. We choose FreeRTOS as it is one of the most popular RTOS, and it is built with an emphasis on reliability, accessibility, and ease of use [10] and is supported by many hardware platforms.

With the practical application of embedded systems, it is essential we consider a Mixed task model as real-time systems are required to respond to aperiodic requests as well.

The main objective of this paper is to use a machine learning method to select DVFS technique for applications with periodic and aperiodic requests running in a FreeRTOS environment. To the best of our knowledge, we are the first to consider experimenting with the dynamic frequency scaling approach considering a FreeRTOS framework with reinforcement learning on a real-time device.

We have taken online parameters like actual execution time, real battery power consumption, and frequency scaling into consideration for the learning algorithm. To summarize, we work on real hardware considering the above factors on FreeRTOS for DVFS task scheduling while focusing on energy saving.

Proceedings of the 1st International Workshop on Computational Intelligence for Process Mining (CI4PM) and the 1st International Workshop on Pervasive Artificial Intelligence (PAI), co-located with the IEEE World Congress on Computational Intelligence (WCCI), Padua, Italy, 18–23 July 2022

mlin@stfx.ca (M. Lin)

2. Related Works

Our approach aims to select a suitable DVFS-based energy-saving technique from a set of existing DVFS techniques to serve periodic and aperiodic real-time requests in small devices that do not have too much support for DVFS. We will next discuss some works related to the various components in the system implementation that make this possible.

2.1. Learning Based DVFS Approaches

Early researchers [11] [12] proposed DVFS learning methods. However, they are based on supervised learning and cannot learn on the fly.

Huang et al. [13] proposed a double Q-learning energy-efficient scheduling with a frequency scaling approach for mobile computing devices. This mechanism reduces the overestimation in Q-values, consequently enhancing the accuracy of frequency predictions. The Double-Q governor is implemented in the Linux kernel. However, the governor does not take into account the timing requirement of individual tasks.

The hybrid DVFS technique proposed in [9] uses multiple DVFS techniques and switches to the most effective one during execution using reinforcement learning. It is clear that their hybrid DVFS technique outperforms other single DVFS methods to conserve energy. However, the evaluation is only conducted in a simulation environment. We find no consideration of mixed taskset handling aperiodic requests along with the periodic request, and no real-platform was considered in the paper.

2.2. DVFS Governors in Linux Kernel

Linux provides good support for DVFS. It has a *FreqScale* driver, and Linux Kernel makes both static and dynamic DVFS governors available. Dynamic governors are able to change the frequency of the processor dynamically by monitoring and responding to the workload changes. The *onDemand* and *Conservative* governors behave similarly, except that *Conservative* slowly changes the frequency to respond to the workload changes, and *onDemand* jumps to the desired level quickly [14].

However, the governors in Linux change the frequency only based on the performance counters. They are not designed for real-time tasks and do not have learning capabilities.

To implement a governor in Linux to support DVFS for real-time tasks with run-time information is very complex as the governor modules need to access the scheduler module. And most importantly, extra real-time scheduling support is needed for Linux. RTLinux is a real-time OS. But integrating additional modules with the RTLinux kernel is very complex, without sufficient support for small devices.

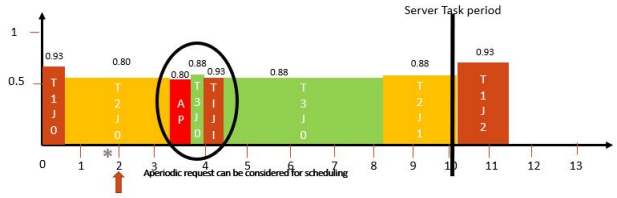

Figure 1: Mixed Taskset Execution with Various Frequency Levels

Our goal is to study if small devices can learn to choose a good DVFS strategy. Instead of working with a complex OS like Linux, we decide to choose an OS that supports real-time tasks. We choose FreeRTOS in this work as it is one of the most commonly used real-time operating systems for small devices and is freely available.

2.3. Individual DVFS Algorithms

Due to the big implementation effort, we only choose to extend two individual DVFS algorithms to include in the set of candidate DVFS techniques for the learning algorithms. One is cycle conservative (CC), and one is look-ahead (LA) [15]. Both CC and LA work for real-time tasks, and they can perform DVFS for a set of periodic tasks to reduce energy while satisfying the tasks' deadline. More DVFS techniques can be added to the candidate list to achieve better energy savings.

The candidate DVFS techniques that we considered extending to handle mixed task sets include CCMT (extended from Cycle Conserving CC) and LAMT (extended from Look Ahead algorithm LA) to handle mixed task sets.

The extended cycle conserving algorithm (CCMT) for a mixed taskset considers the slack time of the server along with the periodic task for calculating the utilization.

The look-ahead algorithm (LAMT) tries to do minimum work before the earliest deadline by pushing as much work as possible beyond that deadline, but at the same time, it makes sure that the future deadlines are met, even if it has to run at a higher frequency [15]. Note that handling a mixed taskset request on both CCMT, and LAMT models is the same.

Fig. 1 shows an execution sequence that handles an aperiodic request using the sever budget with various frequency settings. Note that the frequency levels are normalized between 0 and 1, with 1 being the highest frequency (which consumes the highest power consumption).

As it is unclear when individual DVFS will work bet-

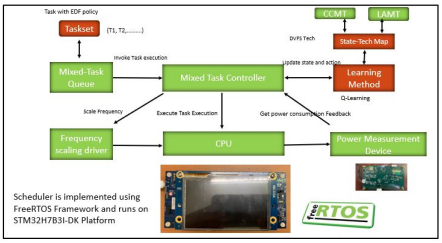

Figure 2: System Framework: Selecting Energy Saving Algorithm in FreeRTOS

ter for different workloads and devices that the workloads are executed, our proposed machine learning-based energy-saving method will learn from different execution of workloads on the underlying devices and select a suitable one based on the current running conditions. Note that the previous machine learning-based hybrid method cannot deal with mixed task sets, including periodic and aperiodic tasks. Our work applies the machine learning-based algorithm that makes decisions to select the extended DVFS algorithms for mixed task sets.

3. Machine-Learning based DVFS Selection on FreeRTOS

3.1. Framework

This section describes the system framework and implementation for the machine-learning-based DVFS Selection on FreeRTOS. The framework of the Extended Hybrid Mixed Taskset Model (EHYMT) in the FreeRTOS environment is shown in Fig. 2. The hardware platform STM32-H7B3I-DK has frequencies ranging from 64 MHz to 280 MHz. We use a smart battery device (by CED labs) to power up the device and measure the voltage and current levels. The power consumption is used as feedback for training the learning model.

The software framework includes the scheduling component related to scheduling and deriving OS and task state 3.3 and the learning component 3.2.

3.2. Learning Component

Q-learning can be used to learn effective policy from its history of interactions with the environment. Given a system consisting of a state space S, and a set of actions A, Q-learning selects an action $a \in A$ at state $s \in S$, leading the system to a new state, and results in a reward or penalty.

For each state-action pair (s, a), the learning system maintains a value function $Q^{\pi}(s, a)$ that represents the penalty or reward. Based on the value function, the agent decides which action should be taken to achieve long-term rewards in the current state.

In the problem of learning to select a good DVFS scheduling algorithm, the action is to choose a frequency scaling method which in turn decides the frequency level for the system. (Note that the system will consume different amounts of power and energy when running at different frequency levels.) The details can be found in 3.2.1. The rewards of the action are related to the power consumption of the system. Unlike the simulation-based method, our method measures the board power consumption and uses this as reward feedback for the Q-Learning method. The details can be found in 3.2.2.

Note that the machine learning component is embedded in the device and works together with Real-Time Operating System to control the device. Some run time features of the current operating system state are used as the state of the learning component to make a decision. The states are described in 3.2.3.

3.2.1. Action: Frequency Scaling Method Performing Frequency Control in FreeRTOS

The core of FreeRTOS is mainly based on three files: task.c, list.c and queue.c [10]. Unlike Linux, the original FreeRTOS does not have software support for frequency scaling. We used a hardware platform STM32-H7B31-DK with FreeRTOS as the OS for the experiment.

The frequency scaling is achieved through a customized clock driver on the STM32-H7B31-DK hardware. The clock control configuration is based on RCC (Reset and Clock Control) subsystem that supports the oscillator [16] to select the appropriate clock for the CPU core and peripherals. Based on the system hardware support and clock configuration divider values, the system supports up to 5 different frequencies from 64 MHz to 280 MHz.

Due to limitations of debugging tools and reading hardware registers on real devices, we have considered the impact of frequency variation on battery power consumption.

3.2.2. Reward: Power Measurement Approaches

In the application level, there are several types of research and tools available which estimate the amount of power consumption by each application or device in a system, e.g., Joulemeter [17], pTop [18], PowerTOP [19], RAPL [20] etc. As on portable devices, we would not be able to find battery drivers that are being used by the software utilities like PowerTop PTop, we have a Smart Battery pack with I2C Communication support from CEDLabs that has integrated analog peripherals to measure and maintain a record of available battery capacity, voltage, and current [21].

Figure 3: Battery Management Module Output without Load

Figure 4: Battery Management Module Output with Load

The CEDLabs Smart Battery Pack has integrated analog peripherals that act as a power source and also help us to measure and maintain a record of the available battery current, voltage, and power. The battery can also report the data to the system host controller over an I2C/UART communication [21].

Fig. 3 shows the battery output without connecting to any load: The output is in the format of current voltage and power. We see that Data= 10mA,7887mV, 1569513uW represents the discharge rate of battery current, voltage, and power.

Fig. 4 shows the battery output when connected to load: The output is in the format of current, voltage, and power. We see that Data= 199mA,7887mV,1569513uW represents the discharge rate of battery current, voltage, and power.

In our Implementation, we integrate the battery management module with the Embedded platform. The battery management module acts as a source of power for the embedded platform and also provides power consumption feedback via the UART Terminal.

3.2.3. State

The learning component considers the CCMT and LAMT DVFS models and switches between these models on the fly at every hyperperiod using reinforcement learning. The job gets scheduled based on the following events.

1. The time at which a periodic task gets released.
2. The time at which an aperiodic task gets released.
3. The time at which a task gets completed or preempted.

The state adopted is composed of a vector of 2 parameters [22]: System Utilization (SU and Dynamic Slack (DS), computed as in Equation 1 and 3. At the start of every hyperperiod, a component that we implement as part of the scheduler control (MixedTask handler) performs the action of calculating the system utilization and dynamic slack to locate the present state. By using the present state, it tries to find the appropriate extended DVFS technique among the CCMT and LAMT models from the state-action map, which has the lowest Q-value in the table. The selected DVFS technique will be used in the next hyperperiod to perform voltage-frequency scaling [9].

The System Utilization (SU) is calculated as follows [4], which is based on the rate of the worst-execution time and the period of tasks.

$$SU = \sum_{i=1}^{n} \frac{w_i}{p_i} + \frac{w_s}{p_s} \tag{1}$$

To determine the dynamic slack, we first calculate the sum of all the tasks' actual execution times in a hyperperiod.

$$et_{hyp} = \sum_{i=1}^{n} \sum_{j \in hyp} et_j^i + \sum_{A_k \in hyp} et(A^k) \tag{2}$$

where et_{hyp} is the hyperperiod, et_j^i is the actual execution time of j_{th} job and i_{th} task, and $et(A^k)$ is the the actual execution time of aperiodic job A^k.

In a hyperperiod, the dynamic slack (DS) of a mixed taskset T is calculated as [4], which is the percentage of slack because some tasks finish early.

$$DS = 1 - \frac{et_{hyp}}{\sum_{i=1}^{n} \frac{hyp}{p_i} \times (w_i) + \frac{hyp}{p_s} \times (w_s)} \tag{3}$$

Where $\frac{et_{hyp}}{p_i}$ is the number of jobs of task t_i in a hyperperiod, p_i is the period for periodic job t_i, p_s is server period, w_i is worst case execution time of periodic task, and w_s is the server budget.

These parameters are updated at every hyperperiod in a scheduling point. Both SU and DS have values between [0,1] and are discretized such that the state space is finite. The system learns the mapping between DVFS techniques, and environment states by interacting with the environment. The action set of selecting the minimum Q-value in the state-tech table is the set of DVFS techniques to be considered. The penalty is calculated

using the average energy consumption in a hyperperiod [9] to update the Q-value.

The Q-table structure looks like the following. E(s_t, a_t)/(Real Battery Power Consumption) is the total power consumption measured by using the battery device in the h_{th} hyperperiod. The state, and penalty evaluation of the learning approach system are essential components of the learning algorithm. The choice of the state and the penalty function typically has to be adjusted or systematically decided based on the experiments and evaluation results [9].

(D_s S_u)	Actions	
	CCMT	LA MT
...		

Table 1
Q-table/State-TechMap

3.3. Scheduling Component

The individual candidate DVFS algorithms CCMT and LAMT are task-based and update the schedule and frequencies at each scheduling point (when a task is released, completed, or preempted). They both use Deadline First scheduling (EDF) as the scheduling policy. However, EDF is not available in FreeRTOS. Carraro [23] implemented (EDF) algorithm in FreeRTOS. However, their work modified the original kernel files.

We chose the ESFree library, the open-source Efficient Scheduling Library proposed by Kase [24], to support our DVFS schedulers that run with EDF policy because it does not change the kernel files. The ESFree library is implemented in the user-space that is user-friendly and runs with low overhead. The implementation of EDF scheduling using ESFree results in a context switch to the scheduler task handler (user-defined) at the beginning and end of a periodic task. The scheduler task handler will then calculate and assign priorities for every periodic task according to their absolute deadlines when it is switched in. This procedure is necessary for ensuring that the priority policy based on EDF is maintained on FreeRTOS[24].

We have implemented a polling server for handling the aperiodic task. Response time is based on the server period. Fig. 5 shows examples of the polling server period, server budget, arrival time representing the time at which an aperiodic request arrives, and the response time (the difference between the arrival time and completion time). Here the aperiodic request is computed offline, and it is known prior and merged with the Mixed Task queue.

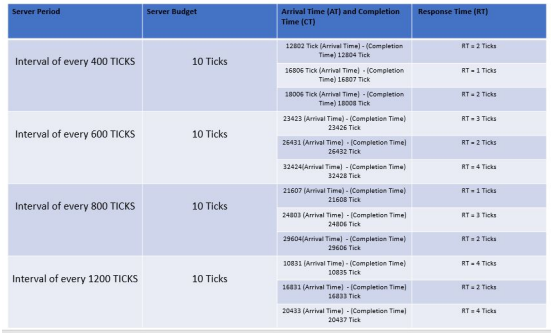

Figure 5: Handling of aperiodic request in FreeRTOS

We keep track of the period, deadline, actual execution time, energy value computed in the hyperperiod, and the DVFS technique to be considered. The data structure is added using FreeRTOS Vlist (ready and running list) and sorted based on the priority. The decision of the chosen frequency is based on the current running state, considering the slack of the tasks using different strategies depending on which candidate DVFS algorithm is chosen. The data structures maintained in the scheduling component in the mixed task queue is used by the mixed-task controller for scheduling decision. The schedule (whether it is CCMT or LAMT) is chosen at the end of each hyperperiod of all the tasks by the learning component.

The Mixed-Task-Handler manages the running of periodic and aperiodic task events, responding to the preemption, and invokes the scheduler before and after the completion of a task to update the priority and the actual execution time. Scheduler-Task-Handler is implemented to perform the operation for selecting the DVFS technique, updating the reinforcement learning method, scaling the frequency, handling timing error detection, and handling WCET excess. The context switch happens from a Mixed-Task-Handler to the Scheduler-Task-Handler at the beginning and the end of a mixed job.

The algorithm is shown in Algorithm 1.

A snapshot of the running queue and frequency chosen is shown in Fig. 7.

3.4. EHYMT-FreeRTOS Hardware Integration

Fig. 8 shows the complete hardware integration of the Hybrid DVFS Model on the STM32 target. The battery management module acts as a primary source of power for the hardware platform STM32H7B3I-DK, and provides feedback on the battery discharge rate for the Learning Algorithm. The Real-Time tasks run on the STM32H7B3I-

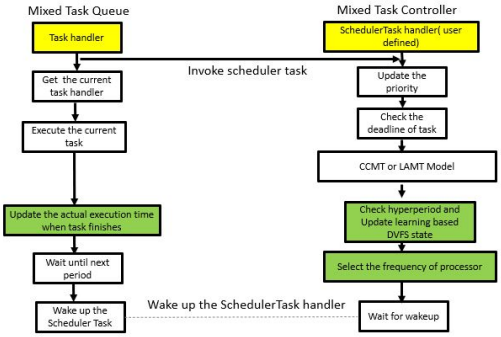

Figure 6: Mixed Task controller Handling on FreeRTOS

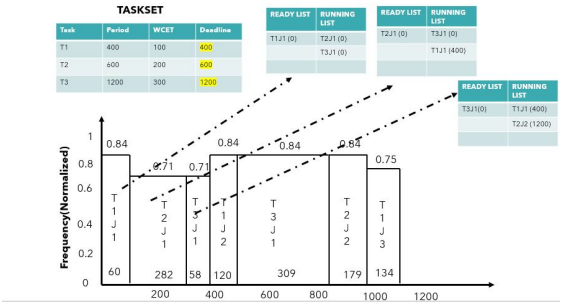

Figure 7: A snapshot of Taskset Execution

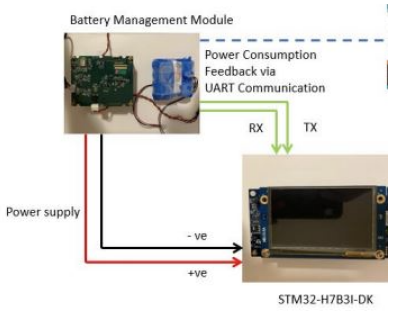

Figure 8: FreeRTOS Hardware Integration Overview

DK Hardware Target. The execution of tasks can be analyzed on the serial terminal using Tera Term. We have also integrated an extra module using an Arduino controller that shows the battery discharge rate results for monitoring purposes.

Fig. 9 shows the STM32 IDE supported for programming 32-bit ARM Cortex-M controllers, The Onboard STLINK-V3E debugger/programmer present on STM32H7B3I-DK hardware is helpful for real-time de-

Algorithm 1: EHYMT-FreeRTOS for Mixed Taskset Handling in FreeRTOS

procedure Initialization
    HAL_Init(): Reset all peripherals.
    SystemClock_Config(): Configure the system clock.
    IRQHandler_Config(): Configure Hardware interrupts.
    SchedulerInit(): Initialize list structure
    MixedTaskCreate(): creates periodic aperiodic task.
    prvInitEDF(): Initializes priorities of all periodic tasks.
    SchedulerTask(): creates the Scheduler Task.
    Initilization Qtable
    vTaskStartScheduler(): Start the scheduler
    Start mixed tasks
procedure Initialization Qtable
    initialize Q-table $Q[S, A]$ by 0
    initialize energy en 0
    initialize action a 0
    initialize system utilization SU to 0 and dynamic slack DS to 0
procedure Learning-based Scheduling
    if hyperperiod **then**
        Apply Q-Learning:
        $ds \leftarrow$ calculate dynamic slack using eq. 3
        $su \leftarrow$ calculate system utilization using eq. 1
        $p \leftarrow$ determine reward p
        update Q-value at s: $(state(ds, su), a, p)$
        select action: DVFS method with min Q-value
    else
        select task t_i based on scheduling algorithm
        apply the frequency level based on the DVFS method selected
        if task-completed-or-preempted **then**
            update et_i of the task
        if aperiodic task At_i arrives **then**
            if budget w_s **then**
                allocate the budget w_s
            else
                allocate the budget w_s in next server period p_s
                wait( )

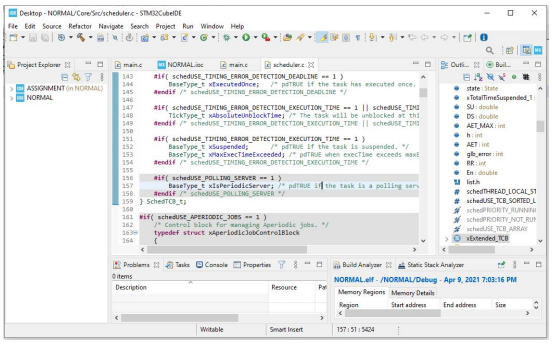

Figure 9: STM32 Integrated Development Environment

bugging and configuration of the device on the fly.

Average no. of runs	5
WCET	[0.5 ms, 10 s]
System utilization	10% to 80%
Actual execution time	10% to 80% of WCET
Power measurement type	Battery management system

Table 2
Task and Experiment Parameters

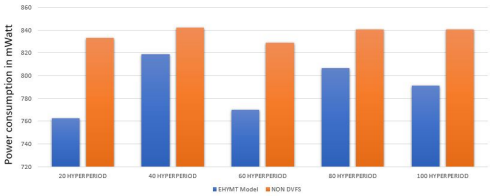

Figure 10: Variation between Learning based Mixed Taskset and NON-DVFS based Mixed Taskset on STM3H7B3I-DK

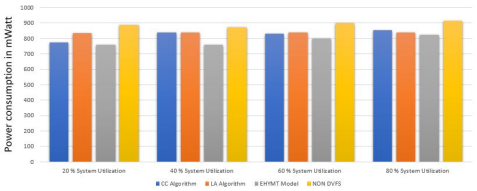

Figure 11: Variation in SU on STM3H7B3I-DK

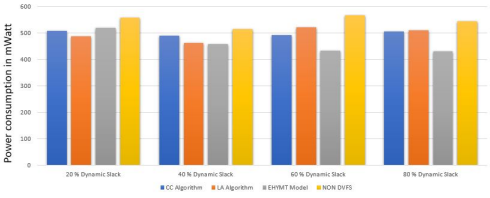

Figure 12: Variation in DS on STM3H7B3I-DK

4. Experiment Results on FreeRTOS

The EHYMT-FreeRTOS Model is implemented and evaluated in the real ARM Cortex-M7 device (a FreeRTOS platform).

The experimental results are based on the following task parameters.

We have considered measuring the power consumption of the battery device in this experiment using the BMS (Battery Management system) hardware. We can observe that the EHYMT-FreeRTOS model can learn and select the best algorithm to consider resulting in energy saving when experimented with variation in the number of tasks, system utilization, dynamic slack, and hyperperiod.

4.1. Variation between Learning based Mixed Taskset and NON-DVFS based Mixed Taskset

Fig. 10 shows the power consumption among DVFS techniques with variations in hyperperiods. We can observe that the EHYMT-FreeRTOS model is able to perform better when compared to Non-DVFS approach. Note: The taskset includes periodic and aperiodic requests.

The following results show the comparisons of EHYMT-FreeRTOS with CCMT and LAMT with variations of different parameters, including SU, DS, hyperperiod, and the number of tasks on STM3H7B3I-DK.

4.2. Variation in SU

Fig. 11 shows the energy consumption among DVFS techniques (EHYMT-FreeRTOS, CCMT, LAMT, and non-DVFS) with variations in system utilization (SU). We experimented with different system utilization of 20%, 40%, 60%, and 80%.

We observe that there is significant energy saving at 20% to 40% system utilization.

4.3. Variation in DS

Fig. 12 shows the energy consumption among DVFS techniques (EHYMT-FreeRTOS, CCMT, LAMT, and non-DVFS) with variation in dynamic slack (DS). We experimented with different dynamic slack 20%, 40%, 60%, and 80%.

The WCETs in each task set are randomly chosen in the range [0.5 ms, 10 s]. The AET is generated upon completion of the task and used for further computation.

We can observe that the EHYMT-FreeRTOS model is able to learn and select the best algorithm to be considered over the dynamic slack variation.

Note: The taskset includes periodic and aperiodic request.

We see that variation in dynamic slack is an essential factor for our learning algorithm, and we see EHYMT-FreeRTOS model is able to select the best DVFS technique that significantly reduces energy consumption when compared to individual DVFS techniques.

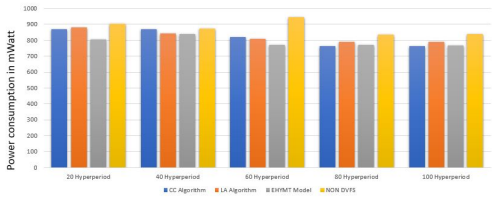

Figure 13: Variation in hyperperiod on STM3H7B3I-DK

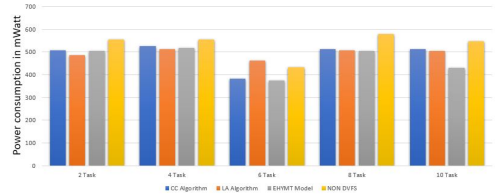

Figure 14: Variations in number of tasks on STM3H7B3I-DK

4.4. Variation in Number of Hyperperiod

Fig. 13 shows the energy consumption among DVFS techniques ((EHYMT-FreeRTOS, CCMT, LAMT, and non-DVFS) with variations in hyperperiod. We experimented with 20, 40, 60, 80, and 100 hyperperiods.

The task sets have a system utilization of 40%. WCETs in each task set are randomly chosen in the range [0.5 ms, 10 s]. The AET is generated upon completion of the task and is used for further computation. We can observe that the EHYMT-FreeRTOS model is able to have an energy-saving of up to 2% to 6% over individual extended DVFS methods on the FreeRTOS.

Note: On the FreeRTOS platform, time is measured in terms of Ticks. Aperiodic event is not offline or predetermined and can arise on the system at any point in time. And it confirms that our EHYMT-FreeRTOS module handles real-time aperiodic requests.

4.5. Variations in Number of Tasks

We have tried checking the impact of variation in the number of tasks. We see that the EHYMT-FreeRTOS model is able to handle multiple tasks and see that EHYMT-FreeRTOS outperforms each individual DVFS technique. Fig. 14 shows the power consumption among DVFS techniques with variations in the number of tasks. We experimented with 2, 4, 6, 8, and 10 tasks.

The task sets have a system utilization of 40%. WCETs in each task set are randomly chosen in the range [0.5 ms, 10 s]. The AET is generated upon completion of the task and is used for further computation.

To summarize, Over 5% to 10% energy savings can be achieved for a standard real-time scheduling mechanism without loss of application throughput. And the method generally reduces energy consumption more than the extended DVFS techniques (CCMT and LAMT) in the candidate list.

5. Time and Space Overhead

The learning method consists of a set of extended DVFS techniques and selects the best one based on the system state by looking up the Q-table. In order to determine the system state, it needs to calculate two parameters: system utilization (SU) and dynamic slack (DS). Among these, DS is updated at each scheduling event, and SU is updated only when new tasks are added, or existing tasks are removed from the task set.

The procedure to update the actual execution time needed to compute the dynamic slack has an $O(n)$ time complexity in the worst case, where n is the number of tasks in a hyperperiod.

The size of the Q-table is $O(SU \times DS \times N)$, where N is the number of extended DVFS candidate techniques that can deal with mixed task set, and SU and DS are the number of discrete levels chosen for the system utilization and dynamic slack [4].

In our learning-based implementation, the reinforcement learning system on the FreeRTOS platform has a small table size of 100. From the analysis, we can see that the learning-based DVFS does not add much computational overhead to the system but gradually becomes an expert in determining the best action if the number of tasks in a hyperperiod n is small.

6. Conclusions

This study shows a machine-learning-based method that works on a real-time embedded system supported by FreeRTOS to select suitable DVFS techniques to achieve energy reduction for mixed tasksset. To summarize, we have achieved the following.

1. We perform experiments with dynamic frequency scaling with a reinforcement-learning approach in the FreeRTOS framework for real devices that can be used for pervasive systems.
2. We have chosen online parameters like actual execution time, real battery power consumption, and frequency-scaling into consideration while scheduling the task.
3. We have approached a new way of measuring power consumption on embedded platforms using a Battery Management system and use the power consumption as feedback for the reinforcement learning method to update the Q-table.

4. Our implementation can be ported onto other embedded platforms powered by FreeRTOS with customization of frequency module. Our study can help the implementation of other individual DVFS techniques or learning-based DVFS methods on the FreeRTOS platform.
5. Experimental results on an ARM Cortex device show that the proposed method reduces more energy consumption than the individual DVFS technique in the candidate list for mixed task set and reduces substantial energy than the non-DVFS scheduling method.
6. Future works include experiments on different FreeRTOS platforms, extending with more candidate DVFS methods, and exploring other machine-learning methods.

References

[1] H. Aydin, R. Melhem, D. Mossé, P. Mejía-Alvarez, Power-aware scheduling for periodic real-time tasks, IEEE Transactions on Computers 53 (2004) 584–600.

[2] S. Mäkikyrö, S. Tuoriniemi, R. Anttila, L. Koskinen, Execution frequency and energy optimization for DVFS-enabled, near-threshold processors, in: 2020 10th International Conference on Advanced Computer Information Technologies (ACIT), IEEE, 2020, pp. 518–522.

[3] P. Pillai, K. G. Shin, Real-time dynamic voltage scaling for low-power embedded operating systems, in: Proceedings of the eighteenth ACM symposium on Operating systems principles, 2001, pp. 89–102.

[4] D. Ramegowda, M. Lin, Can Learning-Based Hybrid DVFS Technique Adapt to Different Linux Embedded Platforms?, in: 2021 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Internet of People and Smart City Innovation, 2021, pp. 170–177.

[5] T. Zhou, M. Lin, Deadline-aware deep-recurrent-q-network governor for smart energy saving, IEEE Transactions on Network Science and Engineering (2021). doi:10.1109/TNSE.2021.3123280.

[6] S. K. Panda, M. Lin, T. Zhou, Energy efficient computation offloading with dvfs using deep reinforcement learning for time-critical iot applications in edge computing, IEEE Internet of Things Journal (2022) 1–1. doi:10.1109/JIOT.2022.3153399.

[7] M. Lin, Y. Pan, L. T. Yang, M. Guo, N. Zheng, Scheduling co-design for reliability and energy in cyber-physical systems, IEEE Transactions on Emerging Topics in Computing 1 (2013) 353–365. doi:10.1109/TETC.2013.2274042.

[8] H. Liu, B. Liu, L. T. Yang, M. Lin, Y. Deng, K. Bilal, S. U. Khan, Thermal-aware and dvfs-enabled big data task scheduling for data centers, IEEE Transactions on Big Data 4 (2018) 177–190. doi:10.1109/TBDATA.2017.2763612.

[9] F. M. M. u. Islam, M. Lin, Hybrid DVFS scheduling for real-time systems based on reinforcement learning, IEEE Systems Journal 11 (2017) 931–940. doi:10.1109/JSYST.2015.2446205.

[10] The FreeRTOS, Kernel, 2022. https://www.lynx.com/embedded-systems-learning-center/most-popular-real-time-operating-systems-rtos.

[11] N. AbouGhazaleh, A. Ferreira, C. Rusu, R. Xu, F. Liberato, B. Childers, D. Mosse, R. Melhem, Integrated cpu and l2 cache voltage scaling using machine learning, in: Proceedings of the 2007 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, 2007, pp. 41–50.

[12] H. Jung, M. Pedram, Supervised learning based power management for multicore processors, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 29 (2010) 1395–1408.

[13] H. Huang, M. Lin, L. T. Yang, Q. Zhang, Autonomous power management with double-q reinforcement learning method, IEEE Transactions on Industrial Informatics 16 (2020) 1938–1946. doi:10.1109/TII.2019.2953932.

[14] R. J. W. Dominik Brodowski, Nico Golde, V. Kumar, Cpu frequency and voltage scaling code in the linux(tm) kernel, 2022-04-02. URL: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[15] S. Saha, B. Ravindran, An experimental evaluation of real-time DVFS scheduling algorithms, in: Proceedings of the 5th Annual International Systems and Storage Conference, 2012, pp. 1–12.

[16] S. M. Electronics, RM0455 Reference manual STM32H7A3/7B3 and STM32H7B0 Value line advanced Arm-based 32-bit MCUs, 2022-04-02. URL: https://www.st.com/resource/en/reference_manual/rm0455-stm32h7a37b3-and-stm32h7b0-value-line-_advanced-armbased-32bit-mcus-stmicroelectronics.pdf.

[17] R. Mittal, A. Kansal, R. Chandra, Empowering developers to estimate app energy consumption, in: Proceedings of the 18th Annual International Conference on Mobile Computing and Networking, Mobicom '12, ACM, New York, NY, USA, 2012, pp. 317–328. doi:10.1145/2348543.2348583.

[18] T. Do, S. Rawshdeh, W. Shi, ptop: A process-level power profiling tool, in: Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09), 2009.

[19] Powertop, 2022. https://01.org/powertop, last accessed on 2022-04-02.
[20] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, Z. Ou, Rapl in action: Experiences in using rapl for power measurements, ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) 3 (2018) 1–26.
[21] Cedlabs, Agnion lithium ion smart battery with i2c communication, 2021. http://www.cedlabs.in/Battery.php.
[22] F. M. M. ul Islam, M. Lin, L. T. Yang, K.-K. R. Choo, Task aware hybrid DVFS for multi-core real-time systems using machine learning, Information Sciences 433 (2018) 315–332.
[23] E. Carraro, Implementation and test of EDF and LLREF schedulers in freertos (2016).
[24] R. Kase, Efficient scheduling library for freertos, 2016.