Knowledge Driven Dynamic Frequency Scaling

Anirban Lahiri, Nagaraju Bussa, Pawan Saraswat

Abstract

This paper proposes a novel, knowledge driven dynamic frequency scaling approach for dynamic power management. It uses a neural network model for learning the behavior of various applications and then uses this knowledge for predicting the optimal processor frequency at any instance of time. The experimental results show the elegance of the approach. The approach may be suitably modified for application in multicore processors and SoCs(System on Chips).

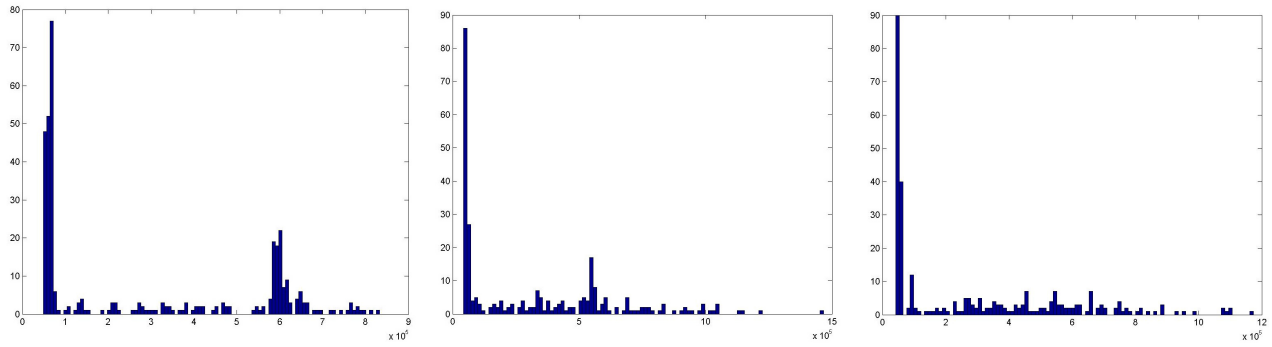
1. Introduction

Software application profiling can be utilized for a variety of purposes like checking the reliability of software, finding performance limitations, code usage, etc. Data obtained from such profiling may in turn be used for predicting the behavior of the concerned software in future. Common attributes of software like reliability can be easily predicted from past data with the help of various techniques for example [4]. Dynamic prediction of application behavior, based on previous profile data, has been used effectively in the past for performance optimization, aid thermal management, reducing resource contention etc. It has also been used for predicting the energy consumption and battery life for mobile embedded devices [6]. However, the issue of how such predictions can be used efficiently for dynamic power / energy optimizations still remains an open area for research. It may be observed that such predictions can be obtained at different levels of granularity. Also the results obtained from these predictions may be utilized using various techniques for energy optimization, again at various levels.

A large number of techniques for power / energy optimization using Dynamic Voltage / Frequency Scaling (DVFS) can be found in literature [11]. In prior work researchers have attempted to guess the behavior of applications [2]. Some of these work attempt to identify the characteristics of system based on history like [8]. Added to this is the fact that various applications or application classes behave in different ways as identified by [2]. The primary insight this work is that the power / energy manager could learn the behavior of applications by itself and devise the DVFS policy rather than the designer or architect dictating them. The current work explores such a learning based strategy using neural networks. The paper describes how it is possible to capture the various aspect of program execution such that the system learns the computational requirements of individual applications. Neural networks score over other knowledge based systems as they require fixed computation time, limited memory space and able to generalize their knowledge, thus characteristics pertaining to one application may be extended to a class of applications similar to it. Furthermore, application behavior is often dependent to a large extent on user profiles and may be subjected to change from time to time. For example, the application of software patches may lead to major changes in software profile. The neural network model used in this work may be retrained in such a case to accommodate the modifications. The retraining can be carried out during periods of time when the device is not in use, like while recharging the device batteries.

The approach discussed in this paper has numerous other advantages over previous ones. The approach renders itself easily applicable to multi-core processors and complex SoCs (System-on-Chips). In such scenarios there may be more than one processor core which support DVFS. It is essentially a hard problem to determine the optimal operating frequency of each processor at a particular instance of time. Any solution to the above problem must necessarily consider the mutual interdependence between the various tasks running on the different cores. This alludes to the fact that the tasks need to be executed in a way such that the real-time deadlines of any application running on the system are met. Though the current work does not provide a comprehensive treatment of the above mentioned scenario, it has been found that a self stabilizing system as the one being described performs relatively well in such a scenario. The mentioned problem would be taken up as a future work.

Another important advantage of the mentioned approach is its ability to work in a non-intrusive way. That is, it does not require any additional code to be embedded in the application unlike many existing techniques [1, 7]. Parameters related to application execution are extracted by



(a) (b) (c)

Figure 1: Distribution of the execution times of AAC4 at frequency (a) 250 MHz (b) 162Mhz (c) 133MHz

instrumenting the operating system and hardware abstraction layers. The method takes advantage of performance counters present on most modern embedded processors to log the necessary statistics.

The next section of the paper discusses some prior work. This is followed by a short analysis on the behavior of embedded applications and the parameters which closely reflect it, especially for the current purpose. Subsequent sections describe how these parameters can be learned, used for predicting the application requirements and implementing the DVFS scheme. The paper concludes with a summary of the results and a section on future work.

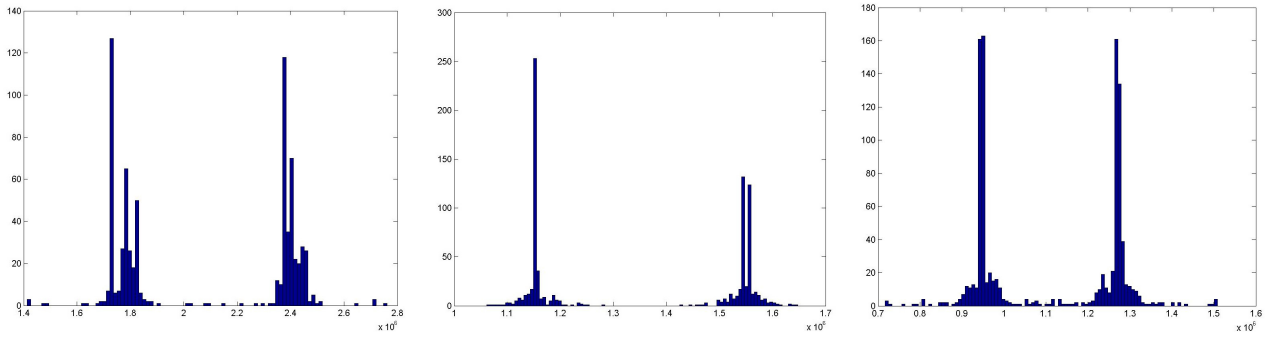
2. Background

The problem of efficiently changing the processor frequency and voltage for saving energy has been a widely researched one. The reader may be referred to [11] for a discussion on the various existing techniques for DVFS since a detailed description of these methods is beyond the scope of the current document. However, these work do not follow a thorough learning based strategy. It may also be mentioned that the importance of knowledge based approaches is expected to rise steadily as embedded processors encompass more than one processor core and each capable of working at multiple frequencies and voltages. This paper addresses the frequency scaling problem since the platform used for experimentation in the current also once the frequency is found determining the operating voltage is trivial. The subsequent sections of the paper describe how the profile of an application can be learned by a neural network and be used for predicting the processor frequency at runtime. Such a learning based system can learn from a variety of heuristic techniques apart from those used in this paper. Hence, the work actually provides a method for improving the performance of previous heuristic techniques for DVFS by predicting future behavior. The paper uses an MPEG4 player as the sample application for the purpose.

3. Learning Application Profiles

The status of an application during runtime can be determined from a number of parameters related to the application profile. In case of an applications involving streaming data, like the MPEG4 player in this case, the execution time of the tasks and the pattern in which they repeat have been found to be important.

Patterns in task execution have been found to be manifested by the distribution of the task execution times and the time periods after which tasks repeat. These time periods may be referred to as the periodicity of the tasks. The distribution curves corresponding to the execution times and the periodicity values vary with change in the processor frequency and indicate the status of the application to some extent. A change in the distribution curve refers to either a change in the shape of the curve or a change in the statistical parameters (like mean, mode, standard deviation etc.) related to the distribution or both. Figure 1 shows the distribution of the execution times of the task ‘AAC4’ which corresponds to an audio decoder , for different frequencies of operation. The shape of the distribution curve undergoes a noticeable change when the CPU frequency is changed. It may be observed that at lower frequencies the execution time of some tasks become short. The reason for this is that these tasks are swapped out frequently by other higher priority tasks. This underlines another aspect of running applications at a reduced frequency. The system buffering used and the context switch interval may also be optimized at different levels for better performance. However, this is



(a) (b) (c)

Figure 2: Distribution of the periodicities of VRV0 at frequency (a) 250 MHz (b) 162Mhz (c) 133MHz

outside the scope of the current work.

The statistical properties of the distribution also change with change in frequency as mentioned above. This is illustrated by Figure 2 which shows the distribution of the task periodicities for the video rendering task 'VRVO'. In this case the shape of certain distribution curves may show only minor changes with change in frequency, but it shows a distinct shift on the X-axis for different processor frequencies.

Apart from CPU speed the execution pattern is also influenced by parameters like number of other applications running on the system, the relative priority of each task, resource requirements of each task, etc. Out of these certain patterns correspond to the "healthy execution" . This means that no application deadline is missed and there is no performance degradation when that particular execution pattern is observed. On the other hand, other patterns may indicate failure in meeting deadlines and quality degradation. Theoretically, such patterns could lead to immediate failure of deadlines, but in the actual case the system is able to sustain itself for sometime due to buffers. Hence, it would be possible to tell whether the system output would be compromised in the near future. Hence, by learning which patterns indicate healthy execution and which probably indicate system failure it is possible to predict the frequency to which the system can be scaled down without affecting the system performance. The next section explains how this can be achieved using Neural Networks.

4. Neural Network Structure and Inputs

The neural network used for learning the application profile has a fully connected feed-forward structure [3]. The ability of the neural network to learn different patterns in

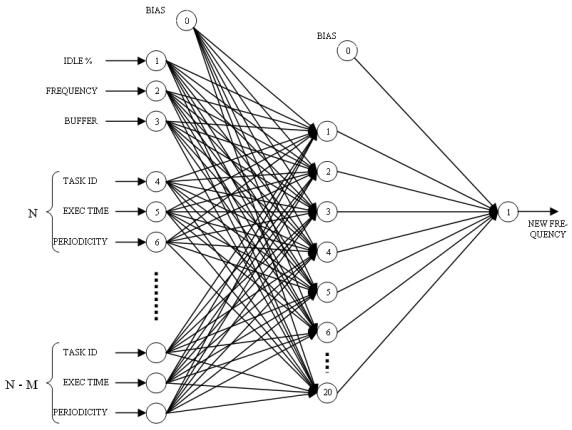


Figure 3: Generic structure of the Neural Network

task execution is dependent on the number of hidden nodes and the type and structure of the inputs provided. It may be mentioned that the number of hidden nodes required is proportional to the number of inputs to the neural network in this case. However, the learning ability of the network does not change significantly when the number of hidden layers is increased beyond unity [9]. Hence, for the current work the number of hidden layers is restricted to one. This also reduces the computational requirement for the prediction model.

Figure 3 shows the generic structure of the neural network. The neural network takes as input a sequence of tasks which have been executed in the recent past. The Nth task indicates the most recently executed task while the $(N-1)$th task refers the one before it and so on. A sequence of $(M+1)$ tasks are examined at a time by the neural network, this indicates the size of the observation window. Each of these tasks is represented by a unique

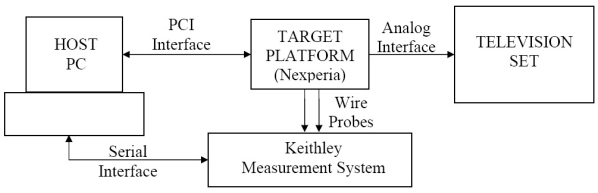


Figure 4: Block diagram of the experimental setup

task identification number referred to as 'TASK ID', the execution time taken by the task and the period of time after which it has recurred (Periodicity) since its last execution. The unique task IDs also allow the neural network to identify whether one or more applications are running The other inputs to the neural network include the current operational frequency, the status of important buffers in the system and the approximate idle percentage. The idle percentage mentioned is calculated over non-overlapping windows of fixed size.

The status of critical system buffers play an important role in determining the future frequency. For example, if the system buffers are close to being empty then there is no rationale behind reducing the frequency further even if the indicated percentage of CPU idle time is considerable. A more detailed discussion on buffers their significance is postponed to a later section. The mentioned inputs are fed to neural network during each task switch and the neural network in turn predicts the new processor frequency. Experiments have been carried out with 7 to 20 hidden nodes corresponding to 6 to 15 inputs for values of M ranging from 0 to 3. A greater number of inputs and hidden nodes allows a better scope for the neural network to learn patterns in task execution. However, it also increases the computational load of the neural network model. It has been found that a value of M=2 suffices in most cases.

5. Training and Functional Operation

The neural network was first trained offline before it was actually ported on to the target embedded platform. For the purpose of training the neural network, a couple of simple heuristic DFS techniques were initially implemented on the target platform. The profile data that was generated by various runs of these techniques were logged on a host PC (Personal Computer) from the target platform using a PCI interface as shown in Figure 4. This data was then used to train the neural network offline until stable neural network weights were obtained. The neural network was ported on to the target platform with the obtained weights. The weights are then fine tuned in an iterative manner.

Among the heuristic techniques mentioned above the first is a idle percentage based DFS approach. This technique computes the percentage of processor idle time within fixed size windows thereafter the new frequency is calculated with the help of the the following expression.

$$F_N = F_C - (Idle\% - Threshold) * F_C \qquad (1)$$

where, F_C and F_N denote the current and new frequencies respectively. Threshold refers to the percentage of idle time that may be allowed for the CPU since 100% utilization is not practically feasible. Furthermore, it leaves some scope for the system to recover in case of a sudden increase in the computational demand of the system. The threshold value also determines the amount of power saving, a higher threshold value usually indicates lower energy savings. The threshold value is initially put at about 20%. It is iteratively reduced to fine tune the neural network weights. However, the idle based approach by itself often leads to poor output quality including jittery audio and video as it is highly dependent on the size of the observation window.

The primary insight used in the second approach is that system performance is not compromised as long as there is sufficient data in the output interface buffers. If any of these buffers become empty at some point of time a discontinuity will be observed in the output. Thus, the previous idle based approach is combined with a mechanism for monitoring the buffer conditions periodically. In case the percentage of the buffer that is empty rises above a certain threshold then the processor frequency is increased by a factor proportional to the empty percentage. Though the buffer monitoring approach leads to significant improvement in the output quality (no audio or video discontinuities are observed), it sometimes leads to excessive fluctuations in the frequency value. This is especially true when the empty percentage of the buffers are close to the above mentioned threshold. This leads to behavior similar to "thrashing". The neural network is taught to identify a good buffer condition with a reward, on the other hand a poorly filled buffer (filled below the threshold) is treated with a punishment.

In either of the above approaches the future values of the frequency are used to train the network. The obtained neural network weights are then fine tuned by running the system on the target platform and logging exceptions (i.e. high buffer empty or idle percentages) if any. The exceptions are then used to retrain the neural network. In a practical system this may be done when the system is not in operation, like while recharging. Hence, the system battery is not taxed for the learning process and computational power is not sapped while the system is in operation.

The functional operation of the system may be explained with the help of figure 5. The data logged by hardware counters in the processor are accessed by the run-time profiler using some specific interface. The relevant statistics are then extracted from this data and forwarded to the neural network. The neural network then makes a prediction

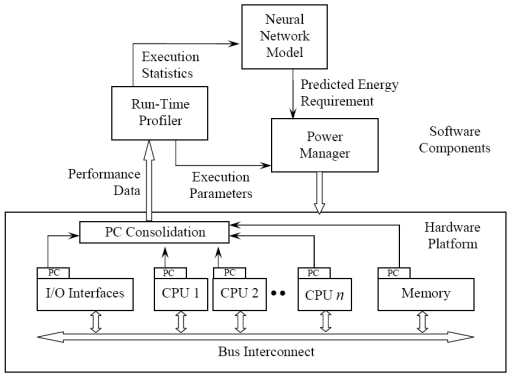


Figure 5: System functional diagram

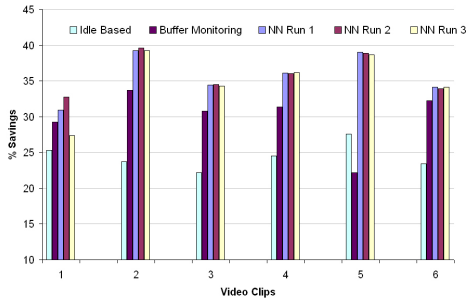


Figure 6: Energy savings obtained for different video streams

Video Clip	Type	Bit Rate (Kbps)	Resolution (pixels)	Frame Rate (frames per sec.)	Audio Sampling Rate (Hz)
1	Drama	2078	704 X 384	29.97	2400 Stereo
2	Moderate Action / Drama	1225	352 X 480	23.98	1600 Mono
3	Moderate Action / Drama	1280	720 X 304	23.98	44100 Stereo
4	High Action	1543	320 X 180	23.98	3200 Stereo
5	High Action	1278	704 X 384	23.98	44100 Stereo
6	Moderate Action / Drama	1281	720 X 336	23.98	44100 Stereo

Table 1: Video clip details

regarding frequency switching based on the given statistics. The frequency and voltage modifications are then effected through an interface to the power manager hardware.

6. Results

The system was coded in C, compiled using NDK (Nexperia Development Kit) [10] and ported on to target platform along with the MPEG4 player application such that they can run parallely. The target platform consisted of a LCP 1500 board [10] which is connected to a standard television for displaying the output video stream. The board is also connected a standard PC for all other communication purposes as shown previously in figure 4. A Keithley 2700 system [5] was used for estimating the power by measuring the voltage across a shunt resistance present in series with the power lead of the processor(see figure 4).

The system was tested using more than 50 MPEG4 video streams having varying profiles (in terms of bit-rate, resolution, frames per second etc.) and content ranging from high action (involving rapid scene changes) to drama and news-reading (having low inter frame differences). The video streams have been broadly classified into three classes - drama, high action, moderate drama/action. The neural network DFS approach yielded a power savings in the range of 26% to 41% with an average of around 32% when compared to power consumption of the processor when running at maximum frequency.The neural network outperforms the idle based and buffer based approaches from which it derived its knowledge by 20% and 6% respectively. It may be asserted again that the neural network may be trained to learn from any previous heuristic approach and then perform better than the same due to its ability to predict the future. Figure 6 shows the power savings obtained in different runs for a small sample set of the video streams having different parameters, representative of the different classes (see Table 1) so as to provide a good comparison. It may be observed that the power savings are different for various runs of the same video stream since a predictive approach is involved rather than a deterministic one and there might be minor variations in task execution for different runs of the same video stream. It may be observed from figure 6 that the buffer based approach sometimes performs worse than the idle based approach due to "thrashing" (Clip 5) as mentioned before. The energy savings were achieved only through frequency scaling since the board did not provide support for voltage scaling. Combined with voltage scaling the approach is expected to achieve far greater savings as power consumption is proportional to the square of the voltage and only linearly related to the frequency. A major advantage of this neural network approach is that it is applicable to processors with on-chip hardware accelerators (like the PNX1500) and multi-core processors with each core capable of DVFS. In such cases a deterministic approach is difficult to realize, hence a predictive approach often proves beneficial.

Figure 7 shows a typical steady state frequency transition graph of the various mentioned techniques. Observations windows of the order of 10^8 and 10^9cycles have been used for the buffer and idle based approaches respectively. A large window size for the idle based approach obtains an average value for the CPU utilization but it fails to capture the dynamic variations in computational load, therefore it may sometimes prevent tasks from meeting their deadlines. It may be observed that though the neural network learns from the idle based and the buffer monitoring approach it does not inherit the drawbacks of either. These drawbacks

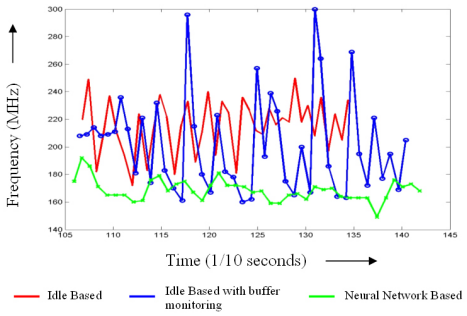


Figure 7: Frequency transition graph

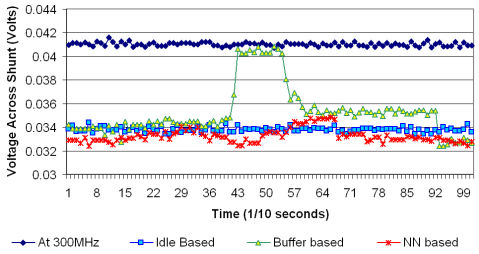


Figure 8: Processor power profiles

as mentioned previously refers to poor output quality and rapid, large changes in the processor frequency (often detrimental to batteries) pertaining to the idle and buffer based approaches respectively.This indicates that the neural network is able to learn successfully from the positive and negative examples provided by other approaches. The power profile, measured by the voltage across the shunt resistance, also reflects a similar trend as shown in figure 8 .

7. Conclusions and Future Work

This paper proposes a novel approach for learning the behavior application on embedded platforms and subsequently use the knowledge for power optimization through dynamic frequency scaling(DFS). The results have been found to be encouraging and the approach is suitable for implementation in multi-core processors and SoCs(Systems on Chips). The technique may also be implemented in hardware for faster operation. It is envisioned that such knowledge based approaches would play a major role in future embedded processors having multiple cores.

References

[1] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau. Profile-based dynamic voltage scheduling using program checkpoints. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, page 168, Washington, DC, USA, 2002. IEEE Computer Society.

[2] Krisztian Flautner, Steve Reinhardt, and Trevor Mudge. Automatic performance setting for dynamic voltage scaling. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 260–271, New York, NY, USA, 2001. ACM Press.

[3] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[4] Nachimuthu Karunanithi, Darrell Whitley, and Yashwant K. Malaiya. Prediction of software reliability using connectionist models. *IEEE Trans. Softw. Eng.*, 18(7):563–574, 1992.

[5] Keithley. http://keithley.com/products/dmm/?mn=2700.

[6] Chandra Krintz, Ya Wen, and Rich Wolksi. Application-level prediction of battery dissipation. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics*, pages 224–229, 2004.

[7] Grigorios Magklis, Michael L. Scott, Greg Semeraro, David H. Albonesi, and Steven Dropsho. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 14–27, New York, NY, USA, 2003. ACM Press.

[8] Ali Manzak and Chaitali Chakrabarti. Variable voltage task scheduling algorithms for minimizing energy. In *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*, pages 279–282, New York, NY, USA, 2001. ACM Press.

[9] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.

[10] Momentum Data Systems. http://www.mds.com.

[11] Vasanth Venkatachalam and Michael Franz. Power reduction techniques for microprocessor systems. *ACM Comput. Surv.*, 37(3):195–237, 2005.