GreenRT: A Framework for the Design of Power-Aware Soft Real-Time Applications

Bo Chen, William Pak Tun Ma, Yan Tan, Alexandra Fedorova, Greg Mori
School of Computing Science, Simon Fraser University
Burnaby, BC, Canada
{bca11, will, yta19, fedorova, mori}@cs.sfu.ca

Abstract

While soft real-time applications must run quickly enough to meet the deadline, there is usually no extra benefit from running more quickly than that. This property provides the opportunity for energy savings using Dynamic Voltage and Frequency Scaling (DVFS). In this paper, we propose the GreenRT framework that allows an application to monitor its own progress, and subsequently adjust the processor frequency dynamically to meet the deadline. Our approach assumes the application can be subdivided into well-defined sub-tasks, each with quantifiable input and output sizes. Using information from the application's past history, the algorithm estimates the amount of input that will be passed into and generated by each sub-task. We demonstrate the effectiveness of the GreenRT framework by using it to implement a pedestrian detection algorithm and show GreenRT could provide an accurate estimation of workload and precisely adapt the execution time of an application under different deadlines.

1. Introduction

Soft real-time applications perform a series of deadline-sensitive tasks, where each such task must be completed within a fixed time frame. Each of these time frames is usually assigned manually or determined by features of the application. Examples of such applications include audio and video decoding, image recognition and certain robot controllers. While missing a task deadline in these applications presents no danger to human well-being (in contrast with hard real-time applications), meeting the deadline is important for good interactive user experience. Another property of these applications is that while each task must run quickly enough to meet the deadline, it does not benefit from running more quickly than that. For example, it is of little use to finish the computation of the robot command much earlier than the robot is ready to accept it. In summary, soft real-time applications must run quickly enough to meet their deadline, but not more quickly than that.

This property presents interesting opportunities for power savings on modern processors capable of Dynamic Voltage and Frequency Scaling (DVFS). DVFS allows dynamically increasing and reducing the processor voltage and frequency. If the voltage and frequency are reduced the applications deliver lower instruction throughput, but the system saves energy. As such, DVFS allows trading off performance and power consumption. DVFS is present on many modern processors, such as PowerTune in IBMs PowerPC 970-based systems, AMDs PowerNow, and Intels Enhanced SpeedStep.

With DVFS, soft real-time applications lend themselves for energy optimizations without the loss to performance as perceived by the user. We envision the following scenario. A soft real-time application monitors its own performance and estimates whether its observed instruction throughput is sufficient to make the deadline. If it is, the application can scale down the frequency on the processor and save energy. If it is not, the application can increase the frequency to make sure that it delivers satisfactory performance to the user.

To evaluate this idea, we have built GreenRT, a framework for incorporating power-awareness into soft real-time applications. The framework consists of three parts: (1) monitoring self-progress by the application; (2) estimating the instruction rate needed to meet the deadline; (3) dynamically adjusting the processor frequency to trade-off performance and energy consumption. We experiment with GreenRT in the context of a pedestrian detection application. We show that using the GreenRT framework, applications use less energy while meeting deadlines. Energy savings are thus achieved without a noticeable impact on user experience. GreenRT framework could become very important for mobile devices, where soft real-time applications are common and power conservation is essential.

2. Related Work

2.1. Power Saving

Power saving is currently a hot topic especially in the field of embedded system. Canturk Isci *et al*. [7] have proposed a method that maximizes the system throughput given a power budget. By monitoring the workload, they estimate the performance degradation which is proportional to the degradation in frequency. Power savings are achieved by scaling down the frequency when power savings are much more important than performance loss.

Rajamani *et al*. [12] propose an application-aware power management algorithm. They continuously monitor the critical workload and achieve power management in two different directions. The first is Performance Maximizer that exploits DVFS levels to maximize performance while ensuring power consumption is within a set limit. The other direction is Power Save that provides energy savings while maintaining specific performance. The performance metric used in that work is IPC.

2.2. Running Time Estimation

Estimating the running time of the application is critical for the GreenRT framework. Corti [2] attempts to find the worst-case execution time of soft real-time applications by computing the duration of each instruction under a specific hardware platform and the maximum amount of possible iterations. By finding the upper-bound execution time of each crucial step, the worst-case execution time could be estimated. Poplavko *et al*. [11] proposed a generic loop execution time estimate giving a tight upper bound and taking parallelism into account. Using the worst-case execution time, however, can lead to overly pessimistic frequency adjustments and will not account for variations due to changes in input. Therefore, we estimate the execution time bases on runtime measurements.

Some previous works have proposed precise prediction for a specific application. For example, Foley *et al*. [5] propose a preliminary predictive model for a key biomedical imaging application which is highly input-dependent. It could give a precise running time estimation by analyzing the algorithm in detail.

Xu *et al*. [16] propose a model that uses Dynamic Voltage Scaling (DVS) for completing tasks under deadline constraint while minimizing energy consumption. Their model estimates the workload of each task bases on the probability density function of the workload of previous tasks. Their assumption is that all tasks are independent and can be run in any order.

More general approaches for running time estimation usually rely on the principle that applications having similar characteristics have similar running times [3, 6, 13]. Shonali Krishnaswamy *et al*. [9] propose a model that uses data mining to estimate application runtime. Execution time of the current application could be categorized by exploring the similarity of the current application to the ones in the database.

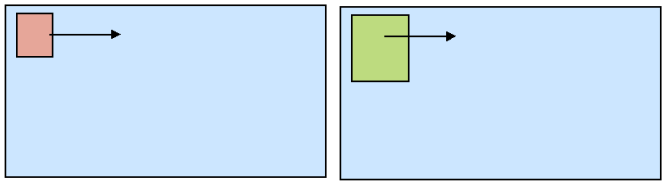

Figure 1. The pedestrian detector uses a windows scanning approach where windows of different size are used to scan the input image.

While all these approaches could be used in our framework, they are more complex than an approach based on runtime measurements employed in our framework.

3. Application Model

We describe properties of a soft real-time application that make it suitable for GreenRT. There are two key requirements: first, the application must consist of well defined tasks. For example, in our pedestrian detection application, a task is the processing of a single image.

Second, it must be possible to set a deadline for each particular task, in a fashion reasonable for the application. For example, for the pedestrian detection application, it should be possible to determine the reasonable time frame within which the image should be processed. (This time frame will depend on the particular scenario where the pedestrian detection is used.)

Third, it must be possible to estimate (with some acceptable amount of error) the amount of time needed to complete a task. This can be done in one of the following ways. If same tasks are periodically repeated, past completion time of a task could be used to estimate its future completion time. Otherwise, it must be possible to represent a task as a series of smaller operations and to estimate performance of each operation based on past performance. With that, the entire tasks completion time can be trivially estimated.

Our application of choice, the pedestrian detection program, answers these three requirements and thus fits within the GreenRT framework. We describe the application in the following section. Many soft real-time applications, such as audio/video encoder/decoder, have a similar structure as our application. This allows our approach to be implemented similarly into such software.

3.1. Pedestrian Detection

For this work, we consider pedestrian detectors in the context of human-computer interaction. Most common pedestrian detectors use a window scan approach, where they scan a variable sized window over the input frame from a video sequence and compute the likelihood of the window-covered area containing a pedestrian (see figure 1). We have chosen an algorithm based on Shapelets [15]. This work approaches the pedestrian detection problem by incorporating a set of classifiers named Shapelets. During the window scanning phase, each window is passed to the Shapelets to compute the likelihood.

The original detector passes each window to all available classifiers for the likelihood computation. However, to make the algorithm suitable for the GreenRT framework we had to break it down into smaller sub-tasks. We modified the Shapelet detector to a cascading structure similar to the one developed by Viola and Jones [14]. With cascading structure, the classifiers are separated into five groups. During the windows scanning phase, we pass all windows to the first group. Only windows that pass the first group will be used as input to the second group of classifiers, and so on. This is sensible because most of the windows do not contain pedestrians and can be rejected by checking a few strong classifiers. Essentially, we have separated the workload into unique stages (as our sub-tasks), where output of one stage is passed down (as a set of windows) to the next stage. Therefore, if we can estimate the amount of work that will be passed down to the next stage, we can estimate the total workload and therefore the total required time and the optimal frequency. This will be explained with more detail in section 4.1.

In summary, we have structured our application into tasks and sub-tasks, where each task is the work of processing one frame from the video, and each of these tasks is separated into interdependent sub-tasks with windows as their input and output.

4. Framework

The key components of the framework are extraction of stages from the application, and estimation of the running time of each stage. Because of the application-specific nature of program stages, the stage extraction in GreenRT is a manual process. Automating this process is an interesting subject of the future work, but it is out of scope of the current paper.

The workload C_i of stage i $(i = 1 \dots n)$ is defined as the total number of cycles this stage will take.

Based on our understanding of the application, we assume that C_i depends linearly on $\vec{W_i}$, the number of windows passed down to stage i. Notice that since the windows are of different sizes, $\vec{W_i}$ is a N_w-dimensional vector, where N_w is the number of scales.

From preliminary experiments we observed that the IPC is static within each stage and approximately constant across different frequencies, for input images as large as 1920×1440. This means our application is CPU bounded. This is expected given the long running time required to process one image, and the fact that the application runs by itself on a single thread.

Finally, we assume that the total instructions I_i to process stage i depends only on the number of windows. According to the following relationships:

$$C_i = \frac{I_i}{IPC_i} \quad (1)$$

$$f_i = \frac{C_i}{T_i} \quad (2)$$

$$T_i = \frac{C_i}{f_i} = \frac{I_i}{IPC_i f_i} \quad (3)$$

where f_i is the frequency at which stage i runs, and T_i is the time used for stage i, we know that C_i depends only on $\vec{W_i}$. We use $C_i = \vec{k_i}\vec{W_i} + b_i$ to model the total cycles. $\vec{k_i}$ is the number of cycles required to compute each window (again, $\vec{k_i}$ should be a N_w dimensional vector). The constant b_i is added to account for non-major operations. $\vec{k_i}$ and b_i are learned online.

In addition, equation 3 suggests an inverse relationship between the frequency and the running time.

We can use our model to assign frequency at each stage such that the application runs at minimum energy while meeting its deadline. Here we use the linear DVFS assumption as used in [7], and write the energy consumption as:

$$E(f,t) = f^{\alpha}t, \quad \alpha \geq 2 \quad (4)$$

where α describes how drastic the energy grows with respect to the frequency. The problem specification is given as:

$$min \sum_{i}^{n} E(f_i, t_i)$$

$$s.t. \sum_{i}^{n} t_i \leq D$$

where D is the deadline, and n is the total number of stages. In order to know the optimal CPU frequency for each stage, we have to address the uncertainty of the number of windows passed down into the current stage.

4.1. Dynamic Assignments

In order to dynamically adjust the CPU frequency to the thread, we need to predict the number of windows in each stage. Let $\vec{r_i}$ be the acceptance ratio between the number of windows in adjacent stages, $\vec{r_i} = \frac{\vec{W_i}}{\vec{W_{i-1}}}$.

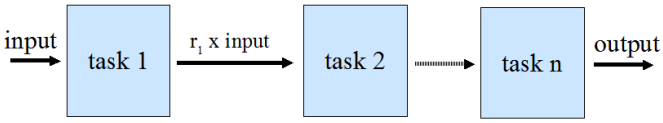

Figure 2. The acceptance ratio is used to calculate how much input will be pass down to the next stage, based on the current stage's input.

We adaptively update the ratios and use them to compute the number of windows to be processed. The ratios are updated by taking a weighted average of their empirical values and the observed value. Specifically, at stage i, when $\vec{W_j}$ are known for all $j \leq i$, $\vec{r_i}$ is updated using:

$$\vec{r_i} \leftarrow \lambda \frac{\vec{W_i}}{\vec{W_{i-1}}} + (1-\lambda)\vec{r_i}, \quad 0 < \lambda < 1 \tag{5}$$

The updated value cannot be used for the current image since the number of input windows at stage i is used for updating the ratio for stage $i-1$. However, the new value will be used in stage $i-1$ during the processing of the next frame in the video sequence and will have a decaying influence for the following frames, depending on the magnitude of λ (see figure 2).

At stage i, the number of windows in the following stages are estimated by:

$$\vec{W_j} = W_i \prod_{k=i+1}^{j} \vec{r_k}, \quad \forall i < j < n \tag{6}$$

Then the workload is computed using the above-mentioned linear equation $C_i = \vec{k_i}\vec{W_i} + b_i$, and we calculate the frequency f_i such that the entire application, if runs at f_i, can finish on time:

$$f_i = \frac{\sum_{j=i}^{n} C_j}{D-t}$$

where t is the elapsed time, and $D-t$ gives the remaining time. According to Ishihara and Yasutjra [8], this frequency f_i will in fact consume the least power under the deadline $D-t$. Unlike the general conception, running the application at the maximum frequency and then set the core to sleep will not be the optimal saving strategy, because it can be observed from equation 3 and 4 that the energy grows at least linearly with the frequency. On the other hand, although it gives the minimum energy consumption to always run at the minimum frequency, the running time often exceed the enforced deadline.

Unfortunately, DVFS does not provide fine granularity in frequency steps. Most of the time we will end up working with a few frequency options (three on our experimental system). One solution for this is to always round up the frequency in order to safely meet the deadline. Another solution is to linearly combine frequencies to achieve the effect no worse than that achieved by the calculated frequency. In the single-threaded case, it has been shown in [8] that the optimal combination can be achieved by running at two adjacent frequencies f_A and f_B for time periods t_A and t_B respectively:

$$f_i t_i = f_A t_A + f_B t_B \tag{7}$$

$$t_A + t_B \leq t_i \tag{8}$$

$$E(f_A, t_A) + E(f_B, t_B) \leq E(f_i, t_i) \tag{9}$$

4.2. Static Assignment

In the static assignment, we are interested in knowing the optimal energy consumption without linearly combining the frequencies. First, we run the detector on all possible frequencies in advance and collect the oracular execution time of each stage under each frequency. Then we permute all frequency assignments and select the one that consumes the minimum energy under each deadline. Finally, the performance under static assignment will be used to evaluate the quality of the dynamic assignments.

5. Implementation of GreenRT

The pedestrian detection algorithm has been implemented in the GreenRT framework using Matlab. When the detector first starts up and begins detecting an input video sequence, the starting frames of the video are used for measuring how many cycles each window of different size will take for each stage. This measurement allows us to determine the values for the vector $\vec{k_i}$ in the equation $C_i = \vec{k_i}\vec{W_i} + b_i$, since the number of work done for each window of a specific size is constant for each stage.

The starting frames are also used to initialize the acceptance ratios needed for estimating the number of input windows that will be passed to the consecutive stages. At each new frame after the starting frames, these ratios are used to estimate the total number of windows (and therefore cycles) needed for completing the task. These ratio is also updated as described in the above section.

At the beginning of each stage, the estimated number of remaining cycles is used to determine the optimal frequency assignment for the remaining stages. However, solving for the exact best frequency for each stage is a knapsack problem and requires dynamic programming, which would add significant overhead to the algorithm. Instead, we solve for one lowest fixed frequency that can be used for all remaining stages while meeting the deadline, using the equation $T = \frac{C}{f}$. Since the number of possible frequencies are fixed, we try both the Round-up strategy and the Linear combination strategy to approximate the final frequency choice.

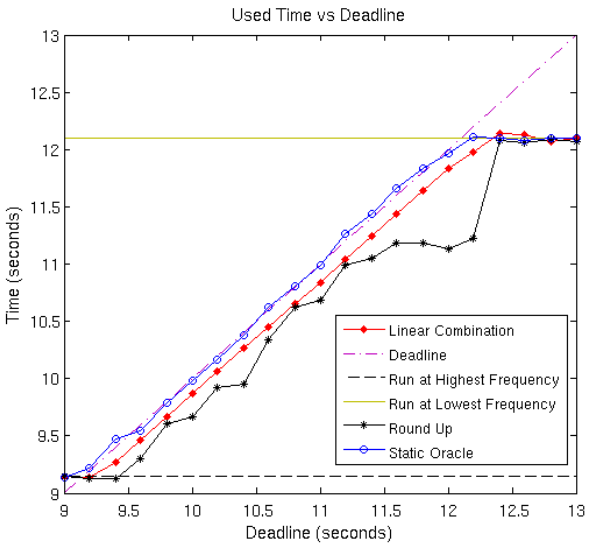

Figure 3. This figure shows the actual execution time of the application under different deadlines.

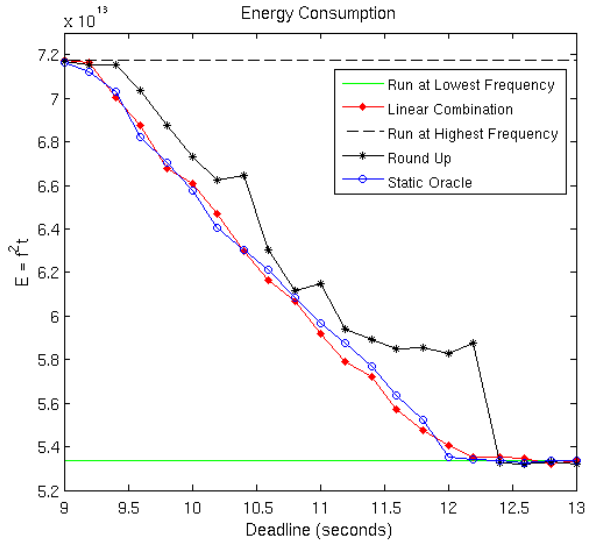

Figure 4. This figure shows the energy consumption under different deadlines. Each line represents a specific frequency adjustment strategy.

6. Results

We ran our experiments on a Dual-CPU hyperthreaded Pentium 4 (Northwood). We use only a single CPU and disable hyperthreading, since our test application is single-threaded. The processor is capable of running at three frequency settings (2.10GHz, 2.45GHz and 2.80GHz) via DVFS. These settings are controlled using system calls from within Matlab. The execution time is measured using `tic` and `toc` functions provided by MATLAB. We compare the Round-up and the Linear-combination strategies with the cases where the application is always running at the highest/lowest frequency. We do not include the Linux default energy manager in the comparison because frequency assignments performed by it are often affected by workloads of other cores and do not account for application deadlines. For the energy consumption function in equation 4 we use $\alpha = 2$.

For our first experiment, we tested our application behavior on one specific frame from the video sequence. The required time for processing our selected frame at the highest frequency is slightly over 9 seconds. We varied the deadline from 9-13 seconds as shown in figure 3. The graph shows the actual used time for all different strategies under the different deadline. The two horizontal lines show the execution time of the application running at the highest and lowest frequencies respectively. The static assignment can almost perfectly meet the deadline, except for the time when the small system noises are affecting the result due to the preciseness of our schedule. The reason for the good performance is that for the static assignment, we use the actual information of the workload instead of estimating it. Thus the execution time could be precisely controlled by assigning appropriate frequencies for each stage. However, this method is not yet the best possible ideal schedule, due to the fact that we are not using linear combination of frequencies. For dynamic methods, both Round-up and Linear-combination strategies have met the deadlines. The execution time under the Round-up strategy is always less than that under the Linear-combination strategy given the same deadline.

In terms of energy savings, we show in figure 4 that energy consumption of both static and dynamic methods is less than that of running at the highest frequency. Among these three methods, the Round-up strategy has the highest energy consumption. The Linear-combination strategy has almost the same energy consumption as the static assignment. Note that the flat line showing the scenario when the processor runs at the highest frequency is always consuming more energy than the others, indicating the correctness of [8].

There are two reasons for the less-than-ideal performance of the Round-up strategy. First, the estimated workload always has some discrepancies compared to the real workload. Second, the Round-up strategy cannot accurately approximate the required frequency. This results in the waste of energy caused by always running at higher frequency than actually required.

The Linear-combination strategy, on the other hand, was able to fit to the deadline by its capability to approximate the required frequency.

Figure 5 shows the performance of our workload estimation algorithm over a video sequence with 92 frames (indexed 109 to 200), with a fixed deadline of 10.6 seconds.

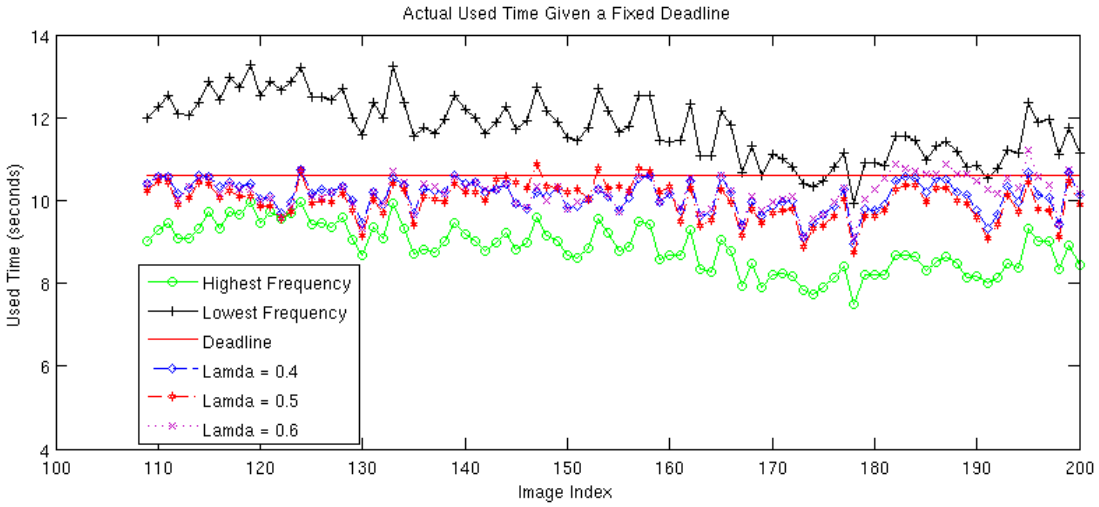

Figure 5. The effect of changing the weight for the rejection ratio estimation. This graph compares the used time given a fixed deadline, with the time needed running at the highest and lowest frequency.

The actual required time for processing each frame running at the highest frequency and the lowest frequency, without using any frequency adjustment or windows prediction, are plotted as green and black lines. While there are rapid changes on the amount of computation between consecutive images, there is a general trend where images early in our sequence require more computation, while images later in our sequence require less. This trend exists because there are less pedestrians in the second half of the video. Our prediction algorithm is able to adapt to this trend and create a more stable completion time. The graph also shows the effect of changing the weight used in the updating formula of the acceptance ratios (equation 5). Choosing a weight of 0.4, where old history is weighted slightly more than the newest information, proves to provide the best result.

7. Conclusions

In this paper, we have described a method that allows soft real-time application to meet a specific deadline while saving energy. We proposed a GreenRT framework. GreenRT allows an application to monitor its own progress, and subsequently adjust the processor frequency dynamically to meet the deadline. This framework is built upon the knowledge that soft real-time applications must run quickly enough to meet the deadline, and there is usually no extra benefit from running more quickly than that. Therefore, if we can predict the future workload of our application, we can adjust the frequency using Dynamic Voltage and Frequency Scaling (DVFS) to run the application as slowly as possible.

Our approach assumed that the application could be subdivided into well-defined sub-tasks, each with quantifiable input and output sizes. We introduced the concept of acceptance ratio, which was used to estimate the amount of input that is passed into and generated by each sub-task. Acceptance ratios were calculated using a weighted average of the past history and the observed value. Combining the estimated workload with the remaining deadline permits us to compute the optimum frequency. Since DVFS only allows fixed frequency steps, we used a linear combination approach that used two different frequencies over the estimated execution time to approximate the actual calculated optimal frequency.

We demonstrated the steps necessary for incorporating GreenRT, and the effectiveness of the framework, by implementing a pedestrian detection algorithm within this framework. We compared the performance of the acceptance ratios under different weights, and found that using a weight of 0.4, which weights past history more than the newest measurement, provides the best prediction. We showed that our prediction algorithm is able to predict the general behavior of the application, allowing for accurate frequency changes. We also compared our linear combination method against a Round-up method. The Round-up method always rounds the optimum frequency to the next possible frequency from DVFS. While rounding up guarantees to meet the deadline, it uses more energy. We showed that the linear combination method allows for a tighter fit to the deadline, and also consumed less energy.

In terms of comparison against the static frequency assignment given oracle knowledge of the application's behavior, we were able to show that our linear combination method gives similar performance to the static assignment.

In our work, we limited the application to run alone on one CPU core. This was based on the fact that most soft real-time applications generally are the only applications running in a system, such as the audio decoder on a portable

media player. However, we believe that the workload estimation and the linear combination scheme used in GreenRT are effective enough to be extended to more complex scenarios. In particular, under a multithreaded environment, workload estimation will be more accurate if the actual IPC is measured and incorporated into the framework, in which case the progress of each thread can be monitored separately using the GreenRT scheme. On the other hand, if there are multiple applications competing for the cores, we need to estimate performance using techniques suitable for multi-core processors [1, 4, 10]. Then we could continue using the GreenRT by offsetting the frequency options.

References

[1] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 340–351, Washington, DC, USA, 2005. IEEE Computer Society.

[2] M. Corti. *Approximating the Worst-Case Execution of Soft Real-Time Applications*. PhD thesis, Swiss Federal Institute of Technology, 2005.

[3] A. Downey. Predicting queue times on space-sharing parallel computers. *ipps*, 00:209, 1997.

[4] A. Fedorova, M. Seltzer, and M. D. Smith. Improving performance isolation on chip multiprocessors via an operating system scheduler. In *PACT '07: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, pages 25–38, Washington, DC, USA, 2007. IEEE Computer Society.

[5] B. Foley, D.P.Spooner, P.J.Isitt, S.A.Jarvis, and G.R.Nudd. Performance prediction for a code with data-dependant runtimes. *UK e-Science Programme All Hands Meeting*, 2005.

[6] R. Gibbons. A historical application profiler for use by parallel schedulers. In *IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 58–77, London, UK, 1997. Springer-Verlag.

[7] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 347–358, Washington, DC, USA, 2006. IEEE Computer Society.

[8] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of 1998 International Symposium on Low Power Electronics and Design*, pages 197–202, 1998.

[9] S. Krishnaswamy, S. W. Loke, and A. Zaslavsky. Application run time estimation: a quality of service metric for web-based data mining services. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 1153–1159, New York, NY, USA, 2002. ACM.

[10] T. Moseley, D. Grunwald, J. L. Kihm, and D. A. Connors. Methods for modeling resource contention on simultaneous multithreading processors. In *ICCD '05: Proceedings of the 2005 International Conference on Computer Design*, pages 373–380, Washington, DC, USA, 2005. IEEE Computer Society.

[11] P. Poplavko, M. Pastrnak, A. Basten, J. v. Meerbergen, M. v. Bekooij, and P. d. With. Run-time prediction of execution times of stream-oriented applications in multiprocessors on chip. *ESR-2005-06*, pages 1–17, 2005.

[12] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson. Application-aware power management. *Workload Characterization, 2006 IEEE International Symposium on*, pages 39–48, Oct. 2006.

[13] W. Smith, I. T. Foster, and V. E. Taylor. Predicting application run times using historical information. In *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 122–142, London, UK, 1998. Springer-Verlag.

[14] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[15] J. Wawerla, S. Marshall, G. Mori, K. Rothley, and P. Sabzmeydani. Bearcam: Automated wildlife monitoring at the arctic circle. *Journal of Machine Vision Applications*, 2008.

[16] R. Xu, D. Mossé, and R. Melhem. Minimizing expected energy in real-time embedded systems. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 251–254, New York, NY, USA, 2006. ACM.