TIME SERIES MODELING BASED POWER AND PERFORMANCE SCALING FRAMEWORK

Moinul H. Khan, Yu Bai, Bin Xiao, Priya N. Vaidya *members IEEE*

ABSTRACT

This paper enhances a software based framework to dynamically scale power and performance with high accuracy in a resource limited embedded system, like cellular phones and PDAs. Key challenges for such a framework are accurate forecasting of dynamic resource demands inherent in the workloads. In this paper we describe three innovative methods: (1) smart forecast method based on linear and non-linear filtering models; (2) policy decision based on high fidelity memory and computation characterization; and (3) adaptive sampling period to adapt to dynamic changes in the workloads. Power and performance framework driven by the proposed algorithms reduces power consumption thus improving battery lifetime for the end-user.

Index/ Key Words – System Modeling, Embedded Multimedia, Low Power Multi-media

1. INTRODUCTION

Next generation cellular phones and PDAs face stringent power and performance requirements. In order to take advantage of dynamic voltage and frequency management, software driven adaptive power management methods are emerging as the key to performance and power scaling [1, 2]. An adaptive power management technique was proposed for Intel® XScale™ Microarchitecture based platforms, which dynamically characterizes executing workloads based on system level events and adapts frequency and voltage so as to save power dissipation [3]. Figure 1 shows the overall framework [3] for embedded software based power management framework. The framework is composed of two basic components: (1) Profiler; (2) Policy Manager. The profiler is responsible for probing the system, collecting the statistics from performance monitoring unit and operating system and making them available to the policy manager. Then the policy manager uses these inputs and optimally chooses suitable system operating point (including core, bus, memory frequencies and processor state) and even different power modes to save power while still satisfying the application's dynamic needs. We define the operating point as:
OP = {PM (State/Mode), f (Freq), V (Voltage)}.

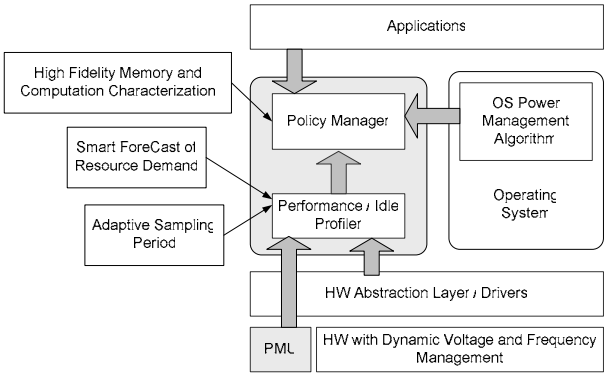

Figure 1: Architecture of the Framework [3]

The effectiveness of the framework relies on the following factors, the observability of different events in the system, being able to track the changes in the resource demand and utilization, being able to accurately predict the future resource demand, and being able to decide the operating point of the device. Using the framework, we seek to improve the solution provided in [3] by improving all areas. In this paper, we present three new schemes to improve accuracy and efficiency in performance and power scaling to significantly extend the whole framework's effectiveness in terms of power savings and performance satisfaction. The Intel® PXA27X processor [4], as our test target, is a highly integrated System-on-a-Chip (SoC) targeting wireless and handheld platforms, featuring Intel® XScale® microarchitecture. To meet the power and performance scaling challenges, system on-chip solutions for handheld devices, such as PXA27X, Intel® XScale™ Microarchitecture based wireless application processor, deploys two key technologies: (1) dynamic voltage/frequency control of the core, interconnect, and memories, and (2) multiple power modes, covering Run, Idle, Deep Idle, Standby, Sleep, and Deep Sleep.

2. PROPOSED IMPROVEMENT SCHEMES

The behavior of an application is dynamic and based on the content being processed by the application, user inputs, and etc. Assuming that the application is slow varying compared to the sampling rate of the performance profiler, the paper [3] proposed to use the application's resource usage in the current sampling period to be the same as that in the previous sampling period. We believe we can extend the prediction capabilities using many known statistical signal processing techniques such as moving average as well as auto regressive modeling. Workload behaviors can be usually represented by (a) core instructions consumption, (b) memory transactions, and (c) other resource usages. From an optimal operating point adjustment perspective, we need to consider the core and memory in greater detail as it consumes the bigger share of the overall power budget. The prediction and control is an intrusive method. In order to reduce the intrusion, we also adopted an adaptive sampling period selection. Based on the prediction, the adjustment of the frequency and voltages are performed. We believe a better model of performance is required to ensure that the system's future operating point is appropriate for the predicted resource forecast. We adopted a finer modeling approach for the same. The following sections describe the proposed improvements in all three areas, specially focusing on the prediction methods.

2.1 Prediction Algorithms

In this paper, we elaborate two methods of dynamically predicting the workloads based on linear and non-linear filtering and also auto-regressive models. Certain simplifications were made in-order to ensure that prediction cost is minimal and does not compromise the real-timeliness of the control mechanism.

2.1.1 Prediction Using Moving Average (MA) Model

In this case, we model system usage demands being a Moving-Average process, where the demand for resource utilization at n^{th} probing window is a weighted sum of the same over last N windows. Let us

define $\hat{W}_R[n]$ as the prediction of demand or usage of a resource *R* at the n^{th} probing window, $\hat{W}_R[n] = \sum_{i=1}^{N} a_R[i]E_R[n-i]$; where $E_R[i]$ is the prediction error at the window *i*. For a simplified model we can have $E_R[i] = W_R[i] - W_R[i-1]$. $a_R[i]$ is the coefficient of prediction. At the end of each probing window, the frequency is adjusted based on the demand forecast of the respective resource. The frequency forecast is considered directly proportional to the demand forecast. Thus, at each probing window, the adjustment to the frequency is done as follows:

$$\Delta f_c[n] = \alpha \sum_{i=1}^{N} c_c[i]E_c[n-i] - f_c[n-1]$$

$$\Delta f_b[n] = \beta \sum_{j=}^{M}\sum_{i=1}^{N} c_b[i,j]E_b[n-i,j] - f_b[n-1]$$

$$\Delta f_m[n] = \gamma \sum_{j=1}^{M}\sum_{i=1}^{N} c_m[i,j]E_m[n-i,j] - f_m[n-1]$$

Here α, β, γ are the proportional constants. For f_b and f_m adaptation, demands for the other resources are also considered and hence the second summation over *M* number of resources. In this paper, we study different models to determine {C}, α, β, γ and different order of history *N*.

2.1.2 Prediction Using Auto-Regressive (AR) Model

The operating frequency is a three number triplet: *f = {fc, fb, fm}*, representing core, bus, and memory frequencies, respectively. We model system usage demands being an auto-regressive process, where the demand for resource utilization at n^{th} probing window is a weighted sum of the same over last *N* windows. Let us define $\hat{W}_R[n]$ as the prediction of demand or usage of a resource *R* at the n^{th} probing window, $\hat{W}_R[n] = \sum_{i=1}^{N} c_R[i]W_R[n-i]$; where $W_R[i]$ is the observed demand and $c_R[i]$ is the coefficient of prediction. At the end of each probing window, the frequency is adjusted based on the demand forecast of the respective resource. The frequency forecast is considered directly proportional to the demand forecast. Thus, at each probing window, the adjustment to the frequency is done as follows:

$$\Delta f_c[n] = \alpha \sum_{i=1}^{N} c_c[i]W_c[n-i] - f_c[n-1]$$

$$\Delta f_b[n] = \beta \sum_{j=}^{M}\sum_{i=1}^{N} c_b[i,j]W_b[n-i,j] - f_b[n-1]$$

$$\Delta f_m[n] = \gamma \sum_{j=1}^{M}\sum_{i=1}^{N} c_m[i,j]W_m[n-i,j] - f_m[n-1]$$

Here as stated in the previous MA model description α, β, γ are the proportional constants. For f_b and f_m adaptation, demands for the other resources are also considered and hence the second summation over *M* number of resources. In this paper, we study different models to determine {C}, α, β, γ and different order of history *N*. For ARMA modeling the frequency adjustment is as follows:

$$\Delta f_c[n] = \alpha \sum_{i=1}^{N} c_c[i]W_c[n-i] + \alpha' \sum_{i=1}^{N} c'_c[i]E_c[n-i] - f_c[n-1]$$

$$\Delta f_b[n] = \beta \sum_{j=}^{M}\sum_{i=1}^{N} c_b[i,j]W_b[n-i,j] + \beta' \sum_{j=}^{M}\sum_{i=1}^{N} c'_b[i,j]E_b[n-i,j] - f_b[n-1]$$

$$\Delta f_m[n] = \gamma \sum_{j=1}^{M}\sum_{i=1}^{N} c_m[i,j]W_m[n-i,j] + \gamma' \sum_{j=1}^{M}\sum_{i=1}^{N} c'_m[i,j]E_m[n-i,j] - f_m[n-1]$$

The appropriate {c}, {c'}, $\{\alpha, \beta, \gamma\}$ and $\{\alpha', \beta', \gamma'\}$ are to be derived dynamically or through some predefined methods.

2.1.3 Coefficient Derivation and Selection

The accuracy and hence the effectiveness of the control relies on the right selection of the prediction coefficients. Determining these coefficients requires complex computation, which will make the dynamic prediction intrusive and will pose an extra overhead. So we simplified AR, ARMA and MA methods to have predefined coefficients. Also, for lower order of prediction model even certain non-linear filtering methods are viable.

For MA models, we adopted three variants of the prediction methods: (a) equal weight moving average, (b) linear prediction, and (c) median filtering. For two different workloads (MP3 playback and MPEG4 decoding), Figure 2 shows improvement of accuracy based on different prediction methods. It covers three prediction orders, i.e., *N* = 1, 2, and 3. Interestingly, moving average and median schemes perform much better than linear estimate scheme. Furthermore, median scheme error is extremely low even with the lowest prediction order, 1. If considering both its simplicity and effectiveness, the median scheme is excellent to forecast audio and video workloads. Please note that in this experiment, for audio workload, 128 kbps bit stream of 44.1 KHz stereo MP3 playback is considered. For video workload, 384 kbps MPEG4 video with akiyo and coastguard clips are considered.

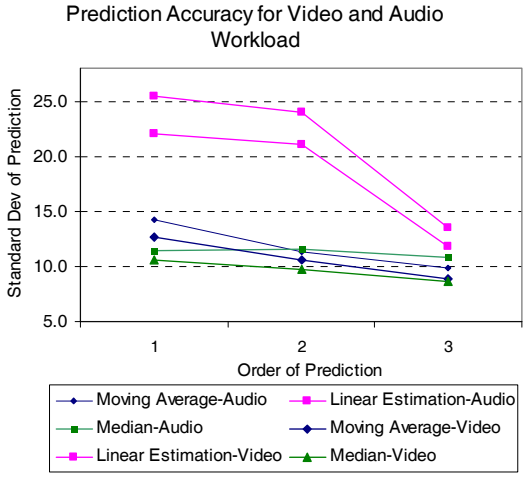

Figure 2: Prediction Error of Three Models for Audio and Video Playbacks

More accurate prediction of the workload demands can easily translate to running the core, bus and memory sub-system at the correct frequencies and not burn any extra power or pay performance penalty. Instead of picking a single prediction model, a hybrid approach can also be adopted.

For AR and ARMA models different methods can be adopted. We implemented 4 different configurations. The configurations are as follows: "Configuration a" represents equal weights for MA samples and median operation of the AR samples. "Configuration b" represents equal weights for MA samples and AR samples. "Configuration c" represents median filtering on both MA and AR samples. "Configuration d" represents reverse of configuration b. Figure 3 shows

how the prediction error changes between different configurations for an audio workload. It can be seen that for different configuration the prediction errors varies.

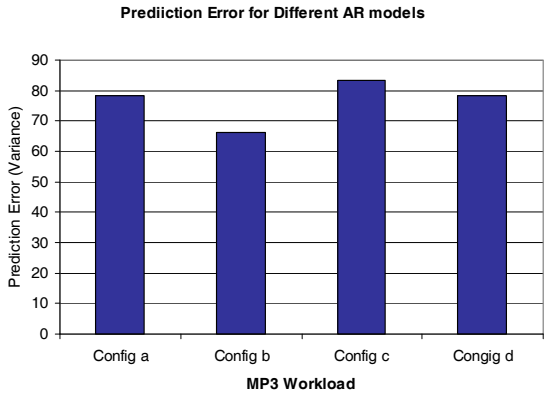

Figure 3: Prediction Error of Different AR Model Configurations for an Audio Workload

2.2 Adaptive Sampling Period

Naturally, some applications change resource requirements at a fast pace, but others do not. Therefore, using a fixed-size probing and decision window cannot meet different applications' needs. The framework adopts three adaptation methods for sampling periods: (1) linear adaptation (LILD-Linear Increase and Linear Decrease) and (2) linear increase-exponential decrease (LIED) of the sampling period (3) linear increase and direct decrease (LIDD). If the prediction error goes up, the sampling period is reduced (linearly or exponentially or reset back to the initial conservative value) and when the prediction error reduces or stabilizes, the sampling period is increased. This way, the scaling technique is more flexible and faster to detect the application's dynamic needs so as to seek more chances to reduce more power dissipation and on the other hand avoid possible performance hazards happening momentarily.

2.3 Memory/Computation Characterization

The characterization algorithm uses the predicted resource demands by the application. For communication fabric demands and memory demands, the similar prediction would hold true. However, the demand forecast can be better understood by classifying if the application is memory bound or computation bound. In terms of deciding the operating point for the next time window, the current nature of the application (memory bound or compute bound) can be taken into account. If an application is identified as a compute bound, the core frequency change is more important and effective than the memory frequency change and opposite for a memory bound case. The frame work is adopting characterization mechanism described in [5]. Based on the memory operations, number of instructions retired etc. the characterization model decomposes the number of cycles consumed in a window into computation and memory cycles. The model also identifies if the memory cycles are throughput related or latency related. Based on the analysis, operation point selection can be actuated.

Figure 4 gives an example how the memory model breaks down execution cycles into memory and compute components for an MPEG4 decoding application. It also shows how the memory and computational composition changes with memory settings and the core frequency. More analysis and conclusion can be found in the paper [5].

Currently we are measuring the effectiveness of the model in terms of actuating the operating point predicted by the prediction model.

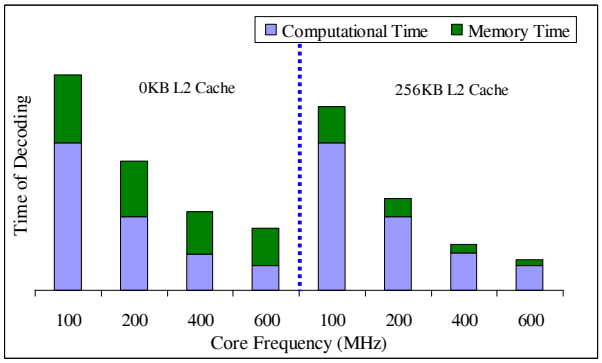

Figure 4: MPEG4 Simple Profile Decoder Composition with Various CPU Frequencies and L2 Cache On/Off

3. RESULTS

We propose a goodness metric based on energy-delay product to compare different approaches. The goodness metric is defined as follows:

$$\sum_{All\ Windows} Power\_penalty(f_p, f_a) \cdot (Performance\_penalty(f_p, f_a))^2$$

Here, f_p = predicted frequency requirement and f_a = actual frequency requirement at an instant. Table 1 provides the definition for performance penalty and power penalty.

Table 1: Definition of Goodness Metric for Different Cases

	Power Penalty	Performance Penalty
$f_p > f_a$	$power(f_p) / power(f_a)$	$-f_p / f_a$
$f_p = f_a$	0	0
$f_p < f_a$	0	f_a / f_p

3.1 Goodness Comparison for Prediction Model

Based on the penalty computation proposed above, we measured the performance, power and prediction errors so that a goodness metric can be compared between different prediction approaches. Table 2 compares different MA models for the audio application. It can be seen that the equal weight moving average prediction method performs the best for the Energy Delay based goodness factor as proposed earlier.

Similarly, we compared the moving average prediction method with the ARMA model configurations (Config "a" to "d", definitions of the configuration can be found Section 2.1.3) in Table 3. It appears that configuration "c" performs the best in terms of the ED goodness metric.

The improvement of the ARMA model can be attributed to the reduced prediction error for the ARMA/AR models. However, we also found that for the audio applications, some of the AR/ARMA models perform

equally or worst compared to the MA model. We are currently investigating the non-intuitive aspects. For the chosen configurations, we are currently measuring the power of the system.

Table 2: Performance and Power Penalty for Different Prediction Methods (Goodness Metric) — MA Adaptive Approach[1]

Model	Prediction Order				
	Metric	1	2	3	4
Moving Average	Prediction Error	1226	718	500	570
	Power Penalty	12	8	6	9
	Performance Penalty	855	376	142	153
Linear Estimate	Prediction Error	N/A	4060	3737	1205
	Power Penalty	N/A	24	14	12
	Performance Penalty	N/A	80000	617	569869
Median Filtering	Prediction Error	N/A	722	743	752
	Power Penalty	N/A	8	8	9
	Performance Penalty	N/A	378	426	273

Table 3: Comparison of a MA Model with Different Configurations of AR Model

Stream	Metric	MA	Config a	Config b	Config c	Config d
akiyo @ 104MHz	Pred Error	123.36	189.69	162.18	237.72	178.64
	Power	1.82	2.14	2.01	2.53	2.12
	Performance	8.88	13.00	11.14	36.71	11.57
akiyo @ 312MHz	Pred Error	17.11	17.96	19.90	15.72	16.59
	Power	0.95	1.25	1.41	0.26	1.29
	Performance	5.54	5.84	5.04	6.33	5.90
coastguard @ 104 MHz	Pred Error	74.78	103.71	95.27	146.44	92.03
	Power	0.85	1.07	1.19	0.65	1.14
	Performance	0.92	1.11	1.21	0.67	1.19
coastguard @ 312 MHz	Pred Error	111.71	137.29	128.33	133.81	130.06
	Power	1.32	1.47	1.42	1.20	1.46
	Performance	1.36	1.42	1.43	1.19	1.46
coastguard @416 MHz	Pred Error	5.72	5.71	7.31	4.65	5.57
	Power	0.58	1.01	1.29	0.12	1.20
	Performance	0.38	0.95	1.10	0.10	1.07

3.2 Sample Rate Adaptation

Figure 5 shows how four different sampling window size adaptive schemes work for the audio workload. The "origin" scheme is to have a fixed-size sampling window. The "LILD" scheme is to linearly increase and decrease the sampling window size as needed. The "LIED" scheme is to linearly increase while exponentially decrease the sampling window size. The "LIDD" scheme is to predefine a minimal window size[2] and then linearly increase the window size, but shrink to the minimal window size once a down-side adaptation is determined. Clearly, "LILD", "LIED", and "LIDD" are detecting the same trends in the workload. "LIDD" does the fastest response and resets the window size. "LILD" is the lowest one, which might miss the opportunities to save more power or catch up performance degradation. To be conservative and keep the fast response speed, the "LIED" and "LIDD" schemes are recommended.

4. CONCLUSION

This paper is intended to enhance a software framework based on Intel® XScale™ Microarchitecture technology, to adapt system resources and frequencies on the fly to meet different applications' dynamic requirements and as a result to reduce power dissipation. We propose three novel techniques that will help predict adapt to system behaviors with high accuracy and therefore save as much as possible power meanwhile maintaining performance. Experiments currently show that higher order MA and Median prediction perform the best in terms of the goodness. AR and ARMA modeling show even higher improvements for video workloads. Memory modeling adaptive windowing shows promise for further optimization. Current results recommend adoption of LIED method of sampling window adjustment.

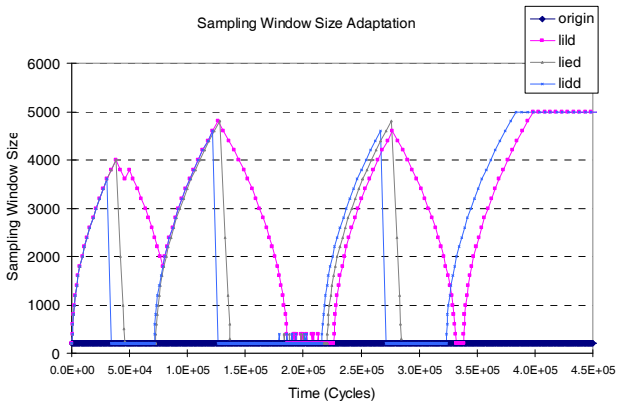

Figure 5: Sampling Window Variation with the Time for Four Different Window Size Adaptations

REFERENCES

[1] Y. Shin, K. Choi, and T. Sakurai, "Power Optimization of Real-Time Embedded Systems on Variable Speed Processors", *Proceedings of the International Conference on Computer-Aided Design*, 2000

[2] Oman S. Unsal and Israel Koren, "System-Level Power Aware Design Techniques in Real-time Systems", *Proceedings of the IEEE*, Vol. 91, July 2003

[3] Priya N. Vaidya, Moinul H. Khan, Bryan Morgan, and Premanand Sakarda, "System Level Adaptive Framework for Power and Performance Scaling on Intel® PXA27X Processor", *International Conference on Acoustics, Speech, and Signal Processing*, Philadelphia, PA, March 2005

[4] Intel® PXA27X Application Processor Architecture Reference Manual

[5] J. Bao, B. Xiao, P. Vaidya, Y. Bai, "A Memory Characteristic Study of Video Applications in Embedded Systems", *Picture Coding Symposium*, April 2006

[1] Lower the number better. The numbers represent ratios and are unitless.

[2] In this experiment, the minimal window size is set to be the window size in the "origin" scheme.