LMS-based Low-Complexity Game Workload Prediction for DVFS

Benedikt Dietrich[1], Swaroop Nunna[1], Dip Goswami[1], Samarjit Chakraborty[1], Matthias Gries[2]
[1]*Institute for Real-Time Computer Systems, TU Munich, Germany*
[2]*Intel Labs, Braunschweig, Germany*
Email: {benedikt.dietrich, swaroop.nunna, dip, samarjit}@rcs.ei.tum.de, matthias.gries@intel.com

***Abstract*—While dynamic voltage and frequency scaling (DVFS) based power management has been widely studied for video processing, there is very little work on game power management. Recent work on proportional-integral-derivative (PID) controllers for predicting game workload used hand-tuned PID controller gains on relatively short game plays. This left open questions on the robustness of the PID controller and how sensitive the prediction quality is on the choice of the gain values, especially for long game plays involving different scenarios and scene changes. In this paper we propose a Least Mean Squares (LMS) Linear Predictor, which is a regression model commonly used for system parameter identification. Our results show that game workload variation can be estimated using a *linear-in-parameters* (LIP) model. This observation dramatically reduces the complexity of parameter estimation as the LMS Linear Predictor learns the relevant parameters of the model iteratively as the game progresses. The only parameter to be tuned by the system designer is the *learning rate*, which is relatively straightforward. Our experimental results using the LMS Linear Predictor show comparable power savings and game quality with those obtained from a highly-tuned PID controller.**

I. Introduction

Graphics-intensive game applications gained significant popularity in recent years. Although most of them are available on high-end desktops, the advent of these applications on battery-powered mobile devices (e.g., laptops, PDAs, cell phones and portable game consoles) is steadily increasing. This recent development is resulting in a constantly widening gap between the demand for computational resources on portable devices and the corresponding energy resources available through batteries [2]. In this context, power management techniques play a significant role in reducing this gap by increasing the energy efficiency of these devices. Most of these devices are equipped with dynamic voltage/frequency-scalable processors in which the power dissipation per clock cycle is directly proportional to its frequency and the square of the supply voltage. Therefore, one can reduce energy consumption through dynamic voltage/frequency scaling techniques, where the processor's clock frequency is dynamically adjusted in response to a varying workload.

Over the last few years, DVFS based power management schemes have been widely explored and successfully applied to audio [3], digital signal processing, and video frame decoding/encoding applications [1], [5], [12], [17]. However, the development of these schemes in the domain of interactive game applications is still in its infancy. This is mainly due to the fact that game applications are highly interactive in nature, where the content is dynamically generated, making it impossible to buffer frames, as it is done in the case of audio or video processing applications.

In this context, recent work [10] has shown that game frames exhibit sufficient workload variability, making graphics-intensive game applications amenable to DVFS schemes. Mallik *et al.* [13], [14] introduced a user-driven DVFS approach wherein the user can manually switch to a higher clock frequency whenever a drop in game quality is observed during the game play. This approach requires user intervention that may not always be desirable and restricts the possibility of applying DVFS schemes at smaller time intervals.

A. Problem Statement and Contributions

In particular, the DFVS techniques proposed by Gu and Chakraborty [7]–[9] are based on proportional-integral-derivative (PID) controllers, where the PID *gain values* had to be hand-tuned. In other words, the *proportional*, *integral* and *derivative* gain values had to be carefully chosen in order to maximize both power savings and the quality of the game play (measured by the number of frame deadline misses). Questions on the robustness of the controller and the sensitivity of frame-workload prediction quality on the choice of the gain values were however left open.

Space of PID gain values: In this paper we first study the influence of the choice of the PID gain values on the quality of the game play (i.e., the number of frame deadline misses) and the achieved power savings. Note that an incorrect estimation of frame workload will result in a wrong setting of the processor's clock frequency. This can either lead to frame deadline misses (if the workload was underestimated) or higher power consumption (because of a higher clock frequency setting). Our experiments – based on the well-known Quake II game engine – show that for most combinations of the proportional, integral and derivative gain values, the PID controller becomes unstable. This leads to large oscillations in the processor's clock frequency setting, which results in poor power savings and low game quality. Further, even for the small choice of gain values – that result in a stable controller – the variation in the number of frame deadline misses is large. Hence, when using a PID

controller to predict frame workload, the gain values need to be carefully chosen. The optimal choice of these values heavily depends on the particular game play and can change from one game scene to the next.

Gu and Chakraborty [7] assume that workload variations across game frames depend only on the game engine. As mentioned above, this assumption does not hold in practice. Hence, hand-tuning the PID gain values only once, at the start of the game, leads to suboptimal performance. Further, our experiments indicate a variation in workload values not only across game plays (for the same game engine), but also across different runs of the same game play (because of the variations induced by the operating system).

Using a LMS Linear Predictor: Hence, optimally hand-tuning the PID gain values is tedious, error-prone and not practical for relatively long game plays with multiple scene changes. To address this problem, we propose using a LMS Linear Predictor, which is a regression model commonly used for system parameter identification [4]. Our choice of a LMS Linear Predictor is motivated by our experimental results, which show that the game frame workload can be accurately estimated using a *linear-in-parameters* (LIP) model. LIP models are those for which output can be represented as a linear combination of their *inputs* and a number of *system parameter* values. In our case, these inputs are previous frame workloads and the output is the predicted workload of the next frame to be processed. The values of the system parameters are iteratively learnt by the LMS Linear Predictor. Hence, the LMS Linear Predictor auto-tunes itself to model the game play at hand accurately. The only parameter that needs to be tuned by the system designer is the *learning rate*, which is relatively straightforward.

Using our proposed LMS Linear Predictor, we obtain power savings and game quality that are comparable to (and sometimes better than) those obtained using a PID controller with carefully hand-tuned gain values. Once again, it may be noted that the LMS Linear Predictor is significantly easier to use and is practically more feasible since it does not require hand-tuning its parameters for different game plays/scenes.

Outlook: At this point it is natural to question the choice of a PID controller in Gu and Chakraborty [7]. PID controllers are used for systems that are more general than those that may be modeled using LIP models, e.g., they may also be used in the case of systems modeled as a collection of differential and difference equations. Our main contribution in this paper is the observation that the full generality of PID controllers is not required for game workload prediction, i.e., here LMS Linear Predictors would suffice. This observation in turn simplifies the problem of suitably tuning the parameter/gain values of the controller/LMS Linear Predictor.

Organization of the paper: In the next section we briefly outline the main features of DVFS schemes for games. This is followed by a description of the PID controller in Section III. Next, the simulation setup for tuning the PID gain values is presented in Section IV. Results obtained from using the PID controller are presented in Section V. We describe our proposed LMS Linear Predictor in the next section (Section VI), along with our methodology for evaluating it in Section VII. Finally, Sections VIII and IX describe our setup for measuring power consumption and frame deadline misses, followed by some concluding remarks in Section X.

Algorithm 1 Structure of DVFS scheme

```
1: Initialize Game and DVFS
2: for i = 0; i < Number of Frames; i++; do
3:     p[i] = predictor(e[i − 1]);
4:     f[i] = quantize(p(i));
5:     Scale frequency to f[i];
6:     Process Frame;
7:     Determine Required Processing Cycles c[i];
8:     e[i] = c[i] − p[i];
9: end for
```

II. DVFS for Game Applications

DVFS-based power management schemes for games primarily depend on estimating game frame workload values. The clock frequency (and hence the supply voltage) of the underlying processor is scaled to match the varying workload of game frames. In this section, we briefly describe the high-level structure of this scheme.

All the DVFS schemes discussed here have the following structure: (i) the workload of a frame is predicted from the workloads of previous frames using a control-theoretic approach, (ii) based on prediction results and the desired frame rate, the required clock frequency of the processor is computed, (iii) the processor's frequency is scaled accordingly, and (iv) the frame is eventually processed (involving game AI, physics-related computations, etc.) and rendered, and finally (v) the estimation error is computed based on cycle-accurate measurements and is fed back to the controller (see Algorithm 1).

An important question that arises here is the choice of frame rate that leads to good gaming experience. User perception studies reported by Claypool *et al.* [6] show that game frame rate has a high impact on perceived game quality and a frame rate of 30 appears to be optimal. In this paper we therefore target a frame rate of 30 fps. This implies that each frame has a deadline of $1/30$-th of a second. Note that unlike in video processing applications, a frame that misses its deadline is not dropped; it only leads to a possibly poor gaming experience.

III. PID Controller Based Workload Prediction

A controller is usually designed for achieving a desired behavior of a dynamical system. In other words, a controller

computes appropriate input signal to the dynamical system which results in desired system behavior. While computing the input signal, the controller utilizes the current status of the system (i.e., feedback signal) and its difference from the desired behavior (i.e., error signal). Various controllers are designed based on how they utilize the error signal. A Proportional-Integral-Derivative (PID) controller is one of the most commonly used controllers. The input signal computed by a PID controller consists of three components,

1) *Proportional:* $P_{comp}(t) = K_p \cdot e(t)$
2) *Integral:* $I_{comp}(t) = \frac{1}{K_I} \cdot \sum_{T_I} e(t)$
3) *Derivative:* $D_{comp}(t) = (K_D) \cdot \frac{e(t) - e(t-T_D)}{T_D}$

where $e(t)$ is the error signal, K_p, K_I, K_D are proportional, integral, derivative gains respectively and T_I, T_D are intervals for integral and derivative components respectively. The output of the PID controller is given by,

$$PID_{output}(t) = P_{comp}(t) + I_{comp}(t) + D_{comp}(t) \quad (1)$$

Let $c[i]$ and $\bar{c}[i]$ be the respective actual and estimated workload values for the i^{th} frame in terms of clock cycles. Here, the goal is to predict the workload $\bar{c}[i+1]$ of the $(i+1)^{th}$ frame by utilizing the actual workload $c[i]$ of the i^{th} frame and the PID control signal computed using Equation (1), i.e.,

$$\bar{c}[i+1] = c[i] + PID_{output}[i]. \quad (2)$$

Towards this, we compute $PID_{output}[i]$ with $e(t) = c[i] - \bar{c}[i]$ being the error signal and

$$\begin{aligned} PID_{output}[i] &= K_p \cdot e(t) \\ &+ \tfrac{1}{K_I} \cdot \textstyle\sum_{T_I} e(t) \\ &+ K_D \cdot \tfrac{e(t) - e(t-T_D)}{T_D} \end{aligned} \quad (3)$$

In order to achieve stable and optimal prediction performance, the values of K_p, K_I and K_D need to be chosen appropriately.

IV. Simulation Setup

The highly dynamic nature of Quake II workload characteristics and additional variations introduced by the underlying OS necessitate an exhaustive exploration of the space of PID gain values. Towards this, we developed a simulation environment for a systematic evaluation of the performance of both the PID controller and the LMS-based workload predictor.

The DVFS algorithm shown in Algorithm 1 is replicated in the simulation, where steps 6 and 7 of the algorithm, i.e., the processing of frames and the workload measurement is replaced by a workload *model*. This workload model is based on recorded workload profiles of Quake II game plays.

However, unlike video applications, in game applications the content of every frame and hence its corresponding workload depends on the processor frequency with which the past frames have been rendered. Now, let the i^{th} frame at time $t[i]$ require $c[i]$ clock cycles and $f[i]$ be the corresponding processor frequency. The time $\Delta t[i]$ taken for rendering the i^{th} frame is then given by $c[i]/f[i]$. Further, the next frame will be rendered at time $t[i+1] = t[i] + \Delta t[i]$. After the i^{th} frame has been rendered, the physics engine calculates the player's new position based on the player's speed and $\Delta t[i]$ (which is the real passage of time). The position of the player and the next frame's content therefore depends on $\Delta t[i]$ and hence on the frequency $f[i]$ (when the frequency is higher, more frames are used to "fill" a certain time interval).

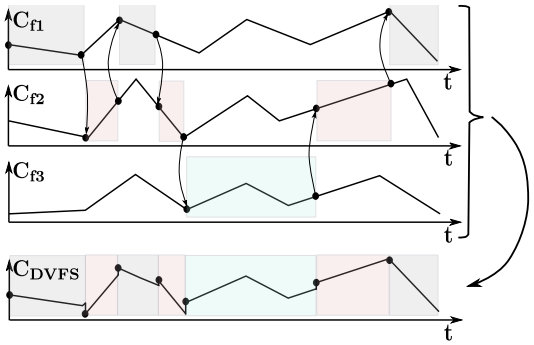

Figure 1. Workload modeling based on fixed frequency recordings

Table I
Run-Times for varying # of Simulation Runs

# of Simulation Runs	t_{sim} [s]	t_{game} [s]	Ratio
400	116.4	78.7e+03	676
1600	206.3	31.2e+04	1515
442401	15650	86.3e+06	5512

This dependency prohibits the direct usage of recorded workload profiles. To get around this, we assume that workload profiles have a "linear" behavior over small time scales. Thus, for each available processor frequency f_i, the corresponding workload profile C_{f_i} is recorded and transformed from the *frame* to the *time* domain by interpolating the missing values. For each frame *pseudo*-processed in simulation with frequency f_i, the corresponding workload profile C_{f_i} is evaluated as shown in Figure 1. The resulting workload profile is denoted as C_{DVFS}.

This workload model together with the replicated steps of the DVFS scheme now allows accurate approximation of the system behavior and an evaluation of different controller settings for DVFS. The runtimes of the simulation compared to concrete runtimes of the game are given for different runs in Table I. A speedup of $5512\times$ is achieved with a Matlab implementation, which clearly shows the advantage of using a simulation-based approach for tuning the controller parameters (gain values).

V. Evaluating the PID Controller

As described in Section III, the choice of PID controller gains is crucial for the performance of the predictor.

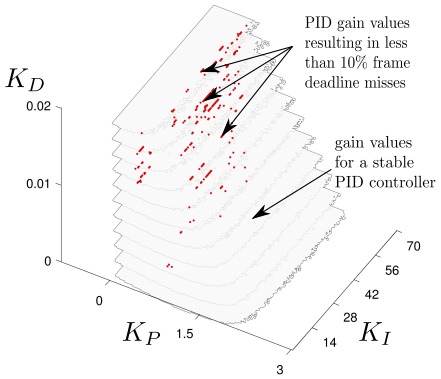

Figure 2. We vary the PID controller gains in the range of interest for Shooting_2. Each plane indicates a stable choice of PID gains (with $K_D = constant$ and K_I, K_P varied)

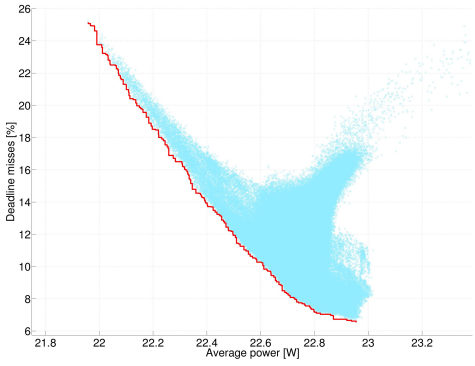

Figure 3. Performance space for different choices of PID controller gains for Level_2

Figure 2 shows the distribution of stable controller gains for a particular game play (i.e., Shooting_2). It may be noticed that only a small portion of the entire space of the controller gains ensures prediction stability. The controller gains with reasonable prediction performance (i.e., smaller than 10% frame deadline miss) are indicated by the red points. Note that the number of such points is limited and distributed over the entire stable space of controller gains. Hence, identifying gain values that lead to a controller with acceptable performance is a nontrivial task. In the following subsection we show how suitable gain values may be chosen.

A. PID Performance Space

We introduce two metrics to judge the prediction quality and the performance of the resulting DVFS. The first metric is the percentage of frames missing their deadlines, i.e., d. As described in Section II, we aim to achieve a frame rate of 30 fps. Therefore, each frame has a deadline of $\frac{1}{30}$ sec. The second metric is the average power consumption. The average power $\overline{P}_{fix}(f_i)$, $\forall f_i \in \mathcal{F}$, is measured as described in Section VIII. We compute $p(f = f_i)$ utilizing the simulation setup. Subsequently, values of $\overline{P}_{fix}(f_i)$ are used together with the frequency probabilities $p(f = f_i)$ to compute the average power for one simulation run using the following equation.

$$\overline{P}_{DVFS} = \sum_{f_i \in \mathcal{F}} p(f = f_i) \cdot \overline{P}_{fix}(f_i)$$

The performance of the PID controller is quantified using two metrics. We obtain a performance space by plotting the values of the metrics corresponding to various sets of PID controller gains. For example, Figure 3 shows the resulting performance space of the PID controller for a particular game play (e.g., Level_2). In Figure 3, every point resembles the values of the performance metrics for a specific set of PID controller gains. The optimal choice of gains is marked by the Pareto-front in Figure 3. It may be noted that the average power consumption is in the range of 22 to 23 Watts resulting in maximum possible savings of 35% (compared to the maximum power consumption of the laptop, which is 34 Watts). However, reducing the power consumption to 22 Watts comes at the cost of an unreasonably high number of frames missing their deadline (i.e., over 25%). Moreover, the variation in power consumption is small compared to the maximum average power consumption in a laptop. It may also be observed that the percentage of frames missing their deadlines is highly influenced by the choice of the gain values. Hence, we choose the set of gain values with the lowest percentage of frame deadline misses (to maximize game quality). Clearly, a systematic identification of suitable controller gains is necessary for each game play/scene.

Table II
WORKLOAD STATISTICS FOR THE UTILIZED GAME PLAYS

Game Play	$\overline{\mathbf{C}}$ $\left[\frac{\mathbf{cycles}}{\mathbf{frame}}\right]$	σ $\left[\frac{\mathbf{cycles}}{\mathbf{frame}}\right]$	$\mathbf{d_{max}}$ [%]	$\mathbf{d_{min}}$ [%]
Explore_1	3.7e+07	3.7e+06	0.7	0.0
Explore_3	3.8e+07	3.2e+06	3.3	0.0
Shooting_1	4.1e+07	4.9e+06	71.8	0.0
Shooting_2	4.1e+07	4.1e+06	67.5	0.0
Level_2	4.0e+07	6.4e+06	66.6	0.2
Massive_1	4.5e+07	7.7e+06	86.5	1.8

B. Selection of Game Plays

We used a number of game plays to evaluate the predictor's performance for diverse workload characteristics. We recorded our own game plays to have realistic workload profiles which commonly occur in Quake II. We recorded four short game plays among which two (i.e., Shooting_1 and Shooting_2) include highly dynamic scenarios involving events like enemy contact. The other two short game plays resemble an exploration phase of the game with comparatively low workload (i.e., Explore_1 and Explore_3). Additionally, we recorded a long game play (i.e., Level_2) with average workload and dynamics. The dynamic behavior of the predictor was also examined using Massive_1 which is a well-known Quake II benchmarking demo with high CPU demand and workload variation. Several runs have

been recorded for each game play to take the variations caused by the underlying OS into account. The resulting statistics of all game plays are shown in Table II where the workload $\overline{C}$ and its standard deviation σ are given in terms of processor cycles per frame. It is obvious that Massive_1 has the highest average workload and standard deviation caused by the highly dynamic nature of this game play. On the other hand, the exploration phases show the lowest $\overline{C}$ and σ. The minimum (d_{min}) and the maximum (d_{max}) percentage of frames missing their deadlines were obtained by running the processor at the highest and the lowest frequency respectively. It may be observed that a certain percentage of frames missing their deadlines are present even when the processor runs at the highest frequency. We used these game plays for the evaluation of the prediction quality of the PID controller and the LMS Linear Predictor.

C. Influence of Game Plays on PID Prediction

To investigate the influence of workload variations, we ran exhaustive simulations for the above-mentioned game plays. For each game play, we used our simulation setup to explore the effect of the controller gain values. Such exhaustive search results in performance spaces similar to the one shown in Figure 3. We selected the set of gains resulting in the lowest rate of frame deadline misses for every game play (i.e., set_{E1}, set_{E3} etc). Table III shows that the percentage of deadline misses is the least when the controller gains are optimized (indicated in bold). For example, the game play Level_2 encounters minimum deadline misses of 6.71% when the gain values set_{L2} is used. However, we get inferior performance with gain values that are optimized for other game plays, e.g., set_{E1} (14%) or set_{M1} (8.19%). It is apparent from Table III that the percentage of frame deadline misses may increase when the PID predictor uses non-optimized controller gains.

However, the PID gains can also be optimized for all the game plays taken together, i.e., set_{ALL} in Table III. This set has been computed by merging the performance spaces of all game plays. Nevertheless, the controller gains again need re-optimization when a new game play is considered. For example, we optimized the PID gains by considering all the game plays listed in Table II except for the game play Level_2. These PID gains were now used for Level_2, which resulted in an unstable predictor for Level_2. Hence, we can conclude that the PID predictor's performance is dependent on the nature of the game play and the controller gains should be optimized for individual game plays to obtain good prediction performance. Since all game plays (or individual player profiles) cannot be known in advance, it is desirable to have a predictor that adapts itself to the current workload characteristics.

VI. LMS Linear Predictor

The LMS Linear Predictor [11] is a statistical approach mainly used for parameter identification of various dynamical systems. Such approaches are suitable for systems that are *linear-in-parameters* (LIP), i.e., the output of the system can be modeled as a linear combination of system inputs and (unknown) system parameters. The LMS Linear Predictor learns the system parameters by recursively updating its values over several iterations.

We use the LMS Linear Predictor for estimating the workload $\bar{c}[i+1]$ of the $(i+1)^{th}$ frame by utilizing the actual workloads of the previous frames. Towards this, we represent the predictor output as a linear combination of known workloads of previous frames and unknown predictor coefficients. If $c[i]$ represents the workload value of the i^{th} frame, then according to the general structure of a one-step LMS Linear Predictor, the predicted workload of the $(i+1)^{th}$ frame is given by,

$$\bar{c}[i+1] = \sum_{k=0}^{n-1} w[k]c[i-k] = W[i]^T C(k) \tag{4}$$

where $w[k]$, for $k = 0, \ldots, n-1$ are unknown predictor coefficients and

$$W[i] = [w[0], w[1], \ldots, w[n-1]^T \tag{5}$$

and

$$C(k) = [c[i], c[i-1], \ldots, c[i-n+1]]^T \tag{6}$$

The goal is to learn $W[i]$ adaptively such that $\bar{c}[i+1]$ in Equation 4 results in the minimum error $e[i]$ in Equation 7.

$$e[i] = c[i+1] \ - \ \bar{c}[i+1] \tag{7}$$

Therefore, from Equations 5 and 6 we have,

$$e[i] = c[i+1] \ - \ W[i]^T C(k) \tag{8}$$

The unknown predictor coefficients are initialized to 0 and after each prediction step, the coefficients are updated according to Equation 9.

$$W[i+1] = W[i] \ + \ \mu \cdot e[i] \cdot C(k) \tag{9}$$

where μ is the *learning rate*. In order to reduce the sensitivity of the learning process to the choice of the learning rate μ, we use a normalized version of Equation 9 that is given by Equation 10.

$$W[i+1] = W[i] \ + \ \frac{\mu \cdot e[i] \cdot C(k)}{||C(k)||^2} \tag{10}$$

The normalized version of LMS is less sensitive to the learning rate μ (it being between 0 and 2). The values of the prediction coefficients, being updated recursively as per Equation 10, converge to the statistical mean [11] after a sufficient number of iterations. The LMS Linear Predictor significantly reduces the design parameter space compared to the unbounded space in the case of PID controller gains.

Table III
PERCENTAGE OF FRAME DEADLINE MISSES OF INDIVIDUALLY TUNED PID GAIN SETS AND OBTAINED RESULTS FOR LMS LINEAR PREDICTOR

	PID							LMS
Game Play	$\mathbf{set_{E1}}$	$\mathbf{set_{E3}}$	$\mathbf{set_{S1}}$	$\mathbf{set_{S2}}$	$\mathbf{set_{M1}}$	$\mathbf{set_{L2}}$	$\mathbf{set_{ALL}}$	$\mathbf{d_{LMS}}$
Explore_1	**0.00**	0.33	0.66	0.82	0.49	0.49	0.49	0.25
Explore_3	0.70	**0.19**	0.44	0.44	0.50	0.56	0.50	0.76
Shooting_1	8.99	6.15	**5.13**	5.85	5.28	5.87	5.72	6.69
Shooting_2	6.65	5.47	6.41	**4.77**	6.78	7.89	6.99	6.61
Massive_1	20.76	16.08	14.65	16.20	**13.43**	14.30	13.82	15.17
Level_2	14.00	8.90	7.65	7.89	8.19	**6.71**	7.79	8.38

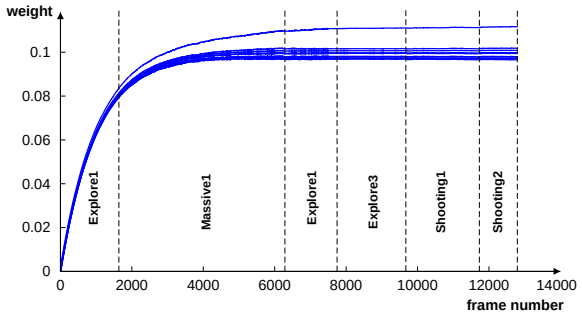

Figure 4. Convergence of LMS-Weights with varying game dynamics

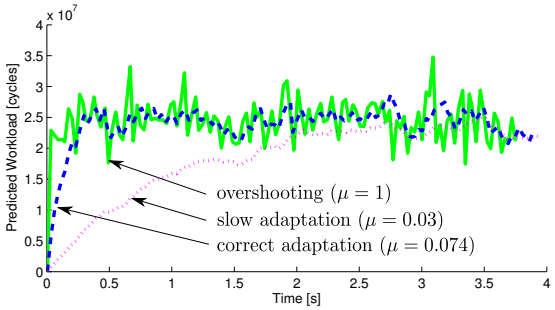

Figure 5. Comparison of different start values of μ

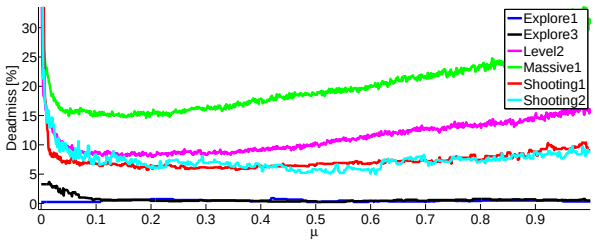

Figure 6. % of missed deadlines for different values of μ for LMS

VII. EVALUATING THE LMS LINEAR PREDICTOR

The Performance of LMS Linear Prediction is determined by the *order* of the predictor and the *learning rate* μ. The order of the predictor indicates the number of workload values of the previous frames being utilized to model the output of the predictor, i.e., n in Equation 4. If the order of the predictor is too low then predictor coefficients will not be able to accurately model the predictor output and will not converge. However, an unnecessarily high order makes the learning process computationally expensive. Hence, the order of the predictor should be chosen considering a trade off between the convergence of the coefficients and the computational cost. The convergence of the predictor coefficients (Figure 4) indicates that the dynamics of the predictor can be modeled as a LIP system as per Equation 4. Moreover, convergence in turn implies accurate approximation of the system using an LMS Linear Predictor.

We experimentally found that an order of 10 is sufficient to model Quake II game plays. Figure 4 depicts the variations of the weights over a sequence of frames. It is clear that the weights converge after 6000 frames. As indicated in the figure, we initiated switches between game plays during the simulation to verify that convergence is preserved under changing system dynamics. Therefore, it is observed that Quake II system dynamics can be accurately approximated by the LMS Linear Predictor.

The performance of prediction is affected by the choice of learning rate μ. As can be seen in Figure 5, a very small learning rate μ results in a significantly high percentage of deadline misses as the learning process is too slow for appropriate adaptation of the weights. On the other hand, a high learning rate results in overdrive effects, i.e., the weights also learn noise. This especially affects dynamic scenes for which the resulting processor frequency varies abruptly. For LMS it is possible to plot the resulting percentage of deadline misses for different values of μ. Figure 6 shows the influence of μ on different game plays. From Figure 6 one can also conclude that the optimal choice of μ is consistent across game plays. Therefore, choosing the optimal μ is more intuitive than choosing PID controller gains. In this work, we use $\mu = 0.074$ for all evaluated game plays.

With LMS Linear Prediction it was possible to obtain the same results as with the optimized PID controller (see Table III). However, the process of finding an optimal μ for LMS is relatively straightforward when compared to the PID controller, since only a one dimensional parameter space with a fixed range needs to be explored in case of LMS.

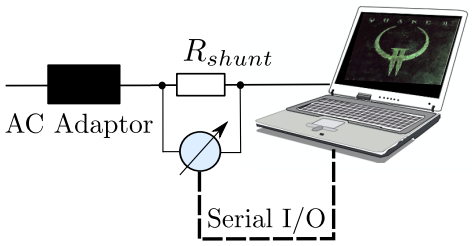

Figure 7. Measurement Setup

VIII. Measurement Setup

In this section we briefly describe our experimental setup. Our experiments employ the Quake II game engine which forms the core of some of the most popular first person shooter games like UFO:Alien Invasion, Anachronox and Heretic II. The source code of Quake II is available under GNU public license and has been modified to incorporate the proposed DVFS power management algorithms. As many of the existing portable devices do not support graphics processing hardware, the graphic processing of the game has been compiled to run completely in software in order to ensure an appropriate evaluation of power measurements. The software video mode of the game has been set to mode 5 which corresponds to a frame resolution of 960 x 720 pixels. The modified source code has been compiled in release mode with processor specific optimizations and the game plays are run on Ubuntu 9.10 operating system with Linux kernel version 2.6.31-20-generic.

The experiments have been performed on a laptop equipped with a 1.86GHz Intel® Pentium® M Processor and 1.5 GB RAM. The processor supports Enhanced Intel SpeedStep® Technology and offers frequency scaling between five different frequency levels that correspond to 800MHz, 1066MHz, 1333MHz, 1600MHz and 1866MHz. In order to obtain a precise processor cycle count, the cycle measurements have been performed with the help of the RDTSC (read-time stamp counter) instruction. The RDTSC instruction was incorporated into the source code of Quake II along with DVFS algorithms and cycle counts corresponding to the processing of each frame were logged during the game play.

The power measurements were performed at the output of the laptop's AC Adaptor as shown in Figure 7. A Texas Instruments microcontroller MSP430 was employed to measure both, the DC voltage $v(t)$ and the current $i(t)$ with the help of a shunt resistor and an amplifier. The average power consumption $\overline{P}$ was then calculated according to

$$\overline{P} = \frac{1}{l} \sum_{t=0}^{t=l} v(t) i(t) T \ ,$$

where l is the duration of the game and $\frac{1}{T}$ corresponds to the sampling rate, which was set to 1kHz. The microcontroller was operated via a serial connection from the laptop and the power measurements were logged for every game play. The control interface for operating the microcontroller was also integrated into the Quake II source code in a way that the measurements through the controller could be started and stopped at the beginning and the end of a game play. In this manner, we ensured synchronization between the start and the stop of the game play and its corresponding power measurements. Moreover, we were also able to reproduce the measurements for different runs of the same game play under identical settings.

Table IV
Average power consumption for available frequencies

Frequency [MHz]	800	1066	1333	1600	1866
Power [Watt]	21.1	23.3	25.8	29.1	33.0

The given setup provided measurements corresponding to the power consumption of the entire laptop. During the measurements the battery was removed from the laptop so as to avoid measuring the power consumed for re-charging the battery as well. Additionally, we ensured that the settings of the laptop remain constant for all the measurements (i.e., we maintained the same settings for display brightness, switched off the wireless LAN and removed all the devices connected to the laptop except the microcontroller used for measuring power consumption).

Table IV shows the laptop's average power consumption for all available frequencies of the processor. In our simulation, these recordings were used to approximate real power consumption. As we measured the system's total power consumption, our measurements include the power consumption of, for example, memory or front-side bus, which highly depends on the load of the system [16]. Therefore, we acquired power measurements for each utilized Quake II demo and all available frequencies together with the corresponding workload profiles. A maximum variation of 2.4% in average power consumption was observed. We conclude from this data that 36% of the power (compared to maximum power consumption of the laptop) may be saved at the maximum by running the system in the lowest frequency all times. This, however, will result in an unreasonably high percentage of deadline misses, significantly reducing game's quality. In the following section we show how we minimized both the deadline misses and the power consumption.

IX. Measurement Results

A. Power Savings

We incorporated the LMS Linear Predictor into the Quake II source code. Table V gives the measured average power consumption ($\overline{P}_{LMS}$) for each game play using the LMS Linear Predictor with a μ of 0.074. Results ($\overline{P}_{PID}$) obtained using the PID controller based workload predictor are comparable to those obtained using LMS $\overline{P}_{LMS}$. The last column shows the achieved power savings of LMS compared to the power consumed if the laptop clocked at the highest

Table V
POWER CONSUMPTION OF PID-BASED WORKLOAD PREDICTOR AND LMS LINEAR PREDICTOR FOR DIFFERENT GAME PLAYS

Game Play	$\overline{\mathbf{P}}_{\mathbf{PID}}$	$\overline{\mathbf{P}}_{\mathbf{LMS}}$	Savings [%]
Explore_1	21.42	21.6	34.5
Explore_3	21.71	21.9	33.6
Shooting_1	23.1	23.6	28.5
Shooting_2	25.2	24.8	24.8
Massive_1	25.1	23.9	27.6
Level_2	23.7	23.2	29.7

frequency. It may be noted that between 24.8% and 34.5% of power is saved depending on the characteristics of the game play.

B. Linux Power Management

Linux is equipped with a widely-used Ondemand Governor [15] for power management. We ran the game plays with the Ondemand Governor (with default settings) enabled and logged the current frequencies, workload profiles and average power consumption. We observed that with the Ondemand Governor, it is possible to obtain approximately 7% power savings (for all game plays), whereas the LMS Linear Predictor achieves power savings of up to 34.5%. As Quake II is programmed as endless loop, the Ondemand Governor will always detect high system utilization. Consequently, voltage/frequency scaling cannot be enabled during most of the time.

X. CONCLUDING REMARKS

In this paper we have proposed a LMS Linear Predictor-based *practical* DVFS scheme for game applications. Our experimental results show that the LMS Linear Predictor does *not* always perform better than a hand-tuned PID controller. There exist game plays for which the PID controller performs better, when such a game is among the set of games for which the PID values were optimized. At the same time, there also exist game plays for which the LMS Predictor performs better. The main reason for proposing the LMS Predictor is that it does not require hand-tuning of the controller parameters, which is tedious and can be error prone. It gives *similar* performance as the PID controller, but with significantly less effort, and is more amenable for implementation in real-life settings (where all game plays and player profiles are not known in advance). As a part of future work, we plan to combine this with power management schemes for other subsystems of a mobile device such as its wireless interface and display.

Acknowledgments: The work reported in this paper was partially funded through a joint project with Intel Labs Braunschweig. Thanks are also due to our shepherd Bronis R. de Supinski for the many helpful comments which helped in improving the paper.

REFERENCES

[1] A. Acquaviva, L. Benini, and B. Ricco. An adaptive algorithm for low-power streaming multimedia processing. In *Design, Automation and Test in Europe (DATE)*, March 2001.

[2] B. Anand, A. L. Ananda, M. C. Chan, L. T. Le, and R. K. Balan. Game action based power management for multiplayer online game. In *ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds (MobiHeld)*, 2009.

[3] T. D. Burd, T. Pering, A. Stratakos, and R. W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuit*, 35(11):1571–1580, November 2000.

[4] E. F. Camacho and C. Bordons. *Model Predictive Control*. Springer-Verlag, 2004.

[5] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *International Conference on Computer-Aided Design (ICCAD)*, November 2002.

[6] M. Claypool, K. Claypool, and F. Dama. The effects of frame rate and resolution on users playing first person shooter games. In *ACM/SPIE Multimedia Computing and Networking (MMCN)*, January 2006.

[7] Y. Gu and S. Chakraborty. Control theory-based DVS for interactive 3D games. In *Design Automation Conference (DAC)*, June 2008.

[8] Y. Gu and S. Chakraborty. A hybrid DVS scheme for interactive 3D games. In *IEEE Real-Time Technology and Applications Symposium (RTAS)*, April 2008.

[9] Y. Gu and S. Chakraborty. Power management of interactive 3D games using frame structures. In *International Conference on VLSI Design (VLSID)*, January 2008.

[10] Y. Gu, S. Chakraborty, and W. T. Ooi. Games are up for DVFS. In *Design Automation Conference (DAC)*, July 2006.

[11] S. Haykin. *Adaptive Filter Theory*. Prentice Hall, Englewood Cliffs, NJ, USA, 1991.

[12] C. J. Hughes and S. V. Adve. A formal approach to frequent energy adaptations for multimedia applications. In *Intl. Symp. on Computer Architecture (ISCA)*, June 2004.

[13] B. Lin, A. Mallik, P. A. Dinda, G. Memik, and R. P. Dick. User- and process-driven dynamic voltage and frequency scaling. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.

[14] A. Mallik, B. Lin, G. Memik, P. A. Dinda, and R. P. Dick. User-driven frequency scaling. *Computer Architecture Letters*, 5(2), 2006.

[15] V. Pallipadi and A. Starikovskiy. The ondemand governor - past, present, future. In *Linux Symposium*, Aug 2006.

[16] D. C. Snowdon, S. Ruocco, and G. Heiser. Power management and dynamic voltage scaling: Myths and facts. In *Workshop on Power Aware Real-time Computing*, September 2005.

[17] W. Yuan and K. Nahrstedt. Practical voltage scaling for mobile multimedia devices. In *ACM Multimedia (MM)*, October 2004.