Time series characterization of gaming workload for runtime power management

Citation for published version (APA):

DOI:
10.1109/TC.2013.198

Document status and date:
Published: 01/01/2015

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Time Series Characterization of Gaming Workload for Runtime Power Management

Benedikt Dietrich, Dip Goswami, Samarjit Chakraborty, Apratim Guha, Matthias Gries

Abstract—Runtime power management using dynamic voltage and frequency scaling (DVFS) has been extensively studied for video processing applications. But there is little work on game power management although gaming applications are now widely run on battery-operated portable devices like mobile phones. Taking a cue from video power management, where PID controllers have been successfully used, they were recently applied to game workload prediction and DVFS. However, the use of hand-tuned PID controller gains on relatively short game plays left open questions on the robustness of the controller and the sensitivity of prediction quality on the choice of the gain values. In this paper we try to systematically answer these questions. We first show that from the space of PID controller gain values, only a small subset leads to good game quality and power savings. Further, the choice of this set highly depends on the scene and the game application. For most gain values the controller becomes unstable, which can lead to large oscillations in the processor’s frequency setting and thereby poor results. We then study a number of time series models, such as a Least Mean Squares (LMS) Linear Predictor and its generalizations in the form of Autoregressive Moving Average (ARMA) models. These models learn most of the relevant model parameters iteratively as the game progresses, thereby dramatically reducing the complexity of manual parameter estimation. This makes them deployable in real setups, where all game plays and even game applications are not a priori known. We have evaluated each of these models (PID, LMS and ARMA) for a variety of games — ranging from Quake II to more recent closed-source games such as Crysis, Need for Speed - Shift and World in Conflict — with very encouraging results. To the best of our knowledge, this is the first work that systematically explores (a) the feasibility of manually tuning PID controller parameters for power management, (b) time series models for workload prediction for gaming applications, and (c) power management for closed-source games.

Index Terms—Dynamic voltage scaling, Time series analysis, Prediction methods.

1 INTRODUCTION

Processor power management based on dynamic voltage and frequency scaling (DVFS) has been a topic of active research over the past one decade [8], [10], [32]. DVFS relies on predicting the future workload of an application to appropriately adapt the processor’s clock frequency and voltage at runtime. The aim is to reduce energy consumption without deteriorating the application’s quality (see [3], [11], [22], [42] for examples).

The increasing popularity of mobile computing in the form of smartphones and tablet PCs has also led to the growth of graphics-intensive gaming applications, which now constitute a significant fraction of the workload supported by these devices. However, power management targeting this class of applications has so far not received sufficient attention in the literature, although it has been shown that they are computationally expensive and game frames exhibit sufficient workload variability, making them amenable to DVFS schemes [20].

Many power management techniques for video applications do not naturally extend to games. This is primarily because of the interactive nature of games where the content is dynamically generated, making it impossible to buffer game frames. But online workload prediction based on control-theoretic techniques, that have been successful for video applications (see [37], [40]), have also been applied to games [18], [19]. They were based on proportional-integral-derivative (PID) controllers to predict the workload of a future game frame based on the workloads of previously processed frames. However, as in the case of video applications, 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). Some important questions were left open by this line of work.

First, if the PID gain values are tuned on the basis of one game application or a selection of game plays, then how robust or sensitive is the resulting controller to new game applications or to a different selection of game plays in the same application? Here, a game play refers to particular sequences of scenes in the game or inputs provided by the user. Second, the PID controller based prediction was evaluated on a relatively old game Quake II from id Software (because it is an open-source game), and in particular with it set to the software rendering mode. In other words, it was assumed that the mobile device did not have a graphics processing unit (GPU) and all the graphics processing had to be done in software on the CPU. Since the introduction of Quake II in 1997, the workload characteristics of games underwent substantial changes, GPUs are now available on mobile devices and the CPU increasingly processes complex Physics or AI related tasks. Hence, how does the PID-based prediction scheme work for more modern games with higher workload variation and less inter-frame correlation? Finally, is it possible to design workload prediction schemes that do not require...
game-specific manual tuning of parameters, so that they work on game plays or even games that are not a priori known.

Our contributions: In this paper we (i) first study the influence of PID gain values on the quality of game play (i.e., the number of frame deadline misses) and the achieved power savings for Quake II as well as for more recent games with hardware-based graphics rendering. Our results reveal that when PID gain values are not individually tuned for each game play then the controller might become unstable, resulting in significant loss in performance. This shows that such PID-based prediction mechanisms, while extensively studied in the power management literature, cannot be practically applied to games, especially when the game play or game is not a priori known. (ii) In order to avoid this hand-tuning of PID gain values, we propose a self-tuning Least Mean Squares (LMS) Linear predictor. It achieves power savings and frame deadline misses that are comparable to those from a carefully hand-tuned PID controller, while the parameters of the LMS Linear predictor do not necessarily need to be tuned for each play individually to provide an overall good performance. (iii) We then introduce a Microsoft DirectX-based DVFS framework to investigate different power management policies for more recent games for which, unlike Quake II, the source code is not available. Hence, they cannot be instrumented to record the processing times of individual frames. To the best of our knowledge this is the first time such lightweight Dynamic-link library (DLL) Injection [30] technique is being used for online workload prediction and power management for DirectX-based closed-source games. (iv) When this technique is used to evaluate our LMS Linear predictor on a set of recent games, we observed higher workload variations and reduced inter-frame correlation for games utilizing hardware-accelerated rendering. This diminished correlation affects the self-tuning process of the LMS Linear predictor, which now might become unstable and provide inaccurate prediction results. To solve this problem, we next study Autoregressive Moving Average (ARMA) models for workload prediction. Our results show that an ARMA model, that is tuned offline, performs well for a variety of games with both software-based and hardware-assisted rendering. Runtime power management techniques based on such prediction models show significant power savings, with very little perceived deterioration in the game quality.

This article extends [15] our earlier work, which only investigated the problems with PID controllers and discussed the potential of using an LMS Linear predictor. In particular, the extensions in this article include: (i) The DirectX-based DVFS framework that overcomes the restriction of using only open-source games that can be instrumented to record frame processing times. The current framework may be used for power management of any new DirectX 9-based game and may be easily extended to support DirectX 10/11. (ii) An exploration of both PID and LMS Linear predictors for current genre of games that use hardware rendering and whose source code is not available. (iii) Workload prediction using ARMA models, that work for a variety of games irrespective of whether they use purely software rendering or rely on GPUs or hardware-assisted rendering.

Related work: A considerable amount of work on runtime processor-energy management has viewed this problem from a scheduling perspective [5–7, 41, 43], i.e., how to select the appropriate clock frequency (which might include shutting off the processor) and voltage at each time step in order to maximize energy savings, taking into account issues like switching overhead and application quality. There has also been work on scheduling and energy management of multiple components like (multi-core) processors and communication links [13, 27]. This problem has also been studied as an online optimization or workload prediction problem – as we do in this paper – in particular using control-theoretic techniques [26, 36, 37, 40]. Since such techniques have been very successful for video processing applications, it was natural to investigate whether they extend to games, as both of them involve processing a sequence of frames. Examples of other work that focus on similar streaming multimedia applications include [16, 35, 44].

During the last 5-6 years, workload characterization and power management of graphics and gaming applications have started attracting attention. [31] exploited the richer content or structural information in graphics frames – in contrast to video frames that have only I, B or P types – to characterize their workload. A combination of such structural information within a frame along with online workload prediction was proposed in [19] for power management of games. On the other hand, the graphics processing pipeline and the architecture of graphics processors were also exploited in [38] for power management of games. In a different direction, techniques that rely on user-interactions [4] or require the user to manually switch to a higher clock frequency whenever a drop in game quality is observed [24, 28] have also been proposed.

Our work does not require user intervention at run time, although it offers the possibility to set a desired frame rate. We focus on frame workload prediction, and use the predicted workload along with the target frame rate to adapt the frequency of the processor. In addition to studying power management for games, from a methodological perspective our work also examines new issues related to control-theoretic techniques for runtime power management that have not been systematically studied before.

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 software- and the hardware-rendering schemes in games. Our simulation setup for tuning PID gain values is presented in Sec-
tion 3.3. Next, the PID controller-based power management scheme is described in Section 4. Results using the PID controller for both software- and hardware-based rendering are then discussed. This is followed by our proposed LMS Linear Predictor (Section 5), along with how we evaluated it. We show the improvements over the PID controller, together with its limitations when using hardware rendering. In Section 6 we discuss how time series analysis may be useful for modeling gaming workloads. Our results obtained from time series modeling, in particular from AR and ARMA models are then presented. Finally, in Section 7 power measurements are outlined before concluding with a summary and possible future extensions.

2 DVFS FOR GAME APPLICATIONS

DVFS schemes for games primarily depend on estimating future game frame workloads. We evaluate several workload prediction techniques in terms of their performance and suitability. All the discussed techniques consist of two parts, an offline and an online phase.

Each predictor has a set of parameters, e.g., the predictor gains and the prediction order, which need to be determined during the offline phase before the predictor can provide a good performance in the online phase. Towards this, we record a sample game play based on which we tune the predictor’s parameters. Such a sample game play can only resemble a small part of the game since not all future game plays of an interactive game application can be a priori known. Hence, the robustness of the predictor, i.e. its sensitivity to changes in workload characteristics, is of great importance. The ideal predictor should be tuned only once and then provide good performance for all future game plays. As the parameter tuning is done offline, it is not time critical.

The online phase of the DVFS scheme has the following structure: (i) The workload of a frame is predicted from the workloads $c$ and estimation errors $e$ of previous frames, (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 workload predictor. Since the online phase is time critical, the overhead of the different steps will be discussed in Section 7.1.

DVFS Performance metrics: We introduce two metrics to evaluate the prediction quality and the performance of the resulting DVFS scheme. The first metric is the average power $P$ consumed during a game play. The power consumption depends on the processing frequencies $f_i$ chosen from the set of available frequencies $F$. $P$ is measured as described in Section 7 or estimated as described in Section 3.3. The second metric is the percentage of frames missing their deadlines, $d$. A frame’s processing time $t_i$ is given by $c[i]/f_i$, where $c[i]$ is the frame’s workload in cycles and $f_i$ the clock frequency used to process the frame. If the processing time $t_i$ is greater than the deadline $d = \frac{fp^d_{desired}}{f}$ the frame is said to have missed its deadline, where $fp^d_{desired}$ is the desired frame rate.

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. Further, there is a clear dependency between the two metrics: always using the smallest available processing frequency $f_{min}$ will minimize $P$, but will lead to the maximum number of frames missing their deadline ($d_{max}$). Using the largest available processing frequency instead will lead to a high power consumption, but will result in the minimum number of frames missing their deadline ($d_{min}$). An optimal predictor (using an oracle) would allow optimizing $d$, $P$ or a combination of both.

Choosing a target frame rate: An important decision is the choice of the target frame rate $fp^d_{desired}$ as it influences both, the percentage of frames missing their deadline $d$ and the average power consumption $P$. Lowering $fp^d_{desired}$ will result in a lower workload, and therefore it is more likely that the frame can be processed in time with a lower frequency. This in turn will lead to higher power savings, but might lead to poor gaming experience. User perception studies reported by Claypool et al. [12] show that the game frame rate has a high impact on the perceived game quality. The perception varies from game to game, i.e., a strategy game’s frames per second (fps) demand is likely to be lower than the desired fps in case of a fast first person shooter game. For this work we have chosen the target frame rate for each game such that the perceived game quality appeared to be optimal for us, e.g. for Quake II this was a frame rate of 30. This implies that each frame has a deadline of $1/30$th of a second.

3 Architectural Setup

The correct choice of a prediction technique highly depends on the underlying architectural setup. Therefore, we will first describe the two hardware setups used, before we introduce the different prediction techniques.

The first setup consists of a software-based rendering setup similar to the one used in [20]. Low-cost mobile devices without hardware accelerated rendering will fall under this category. The second setup uses a graphics processing unit (GPU) to render the game’s graphics. This setup is now typically found in most high-end mobile devices like smart phones and tablets.

3.1 Software-based rendering setup

We employed the Quake II game engine which forms the core of some of the most popular first person shooter games like Raven Software’s Soldier of Fortune*, Anachronox* from EIDOS and Activision’s Heretic II*. Quake II was chosen for two reasons: i) We could compare our results to other related work that studied Quake II [4], [17], [20]. ii) Quake II uses software-based rendering and its engine forms the core of a variety of
TABLE 1
Workload statistics for the used Game Plays

<table>
<thead>
<tr>
<th>Quake II Game Play</th>
<th>Avg. workload $\bar{C}$ [cycles/frame]</th>
<th>Deviation $\sigma$ [cycles/frame]</th>
<th>Target FPS</th>
<th>Frames missing their deadline % $d_{max}$ $d_{min}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explore-1</td>
<td>3.7e+07</td>
<td>3.7e+06</td>
<td>30</td>
<td>0.7</td>
</tr>
<tr>
<td>Explore-2</td>
<td>3.8e+07</td>
<td>3.2e+06</td>
<td>30</td>
<td>3.3</td>
</tr>
<tr>
<td>Shooting-1</td>
<td>4.1e+07</td>
<td>4.9e+06</td>
<td>30</td>
<td>71.8</td>
</tr>
<tr>
<td>Shooting-2</td>
<td>4.1e+07</td>
<td>4.1e+06</td>
<td>30</td>
<td>67.5</td>
</tr>
<tr>
<td>Level-2</td>
<td>4.0e+07</td>
<td>6.4e+06</td>
<td>30</td>
<td>66.6</td>
</tr>
<tr>
<td>Massive-1</td>
<td>4.5e+07</td>
<td>7.7e+06</td>
<td>30</td>
<td>86.5</td>
</tr>
<tr>
<td>DirectX Game</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NFS-1</td>
<td>5.5e+07</td>
<td>1.0e+07</td>
<td>30</td>
<td>50.22</td>
</tr>
<tr>
<td>NFS-2</td>
<td>6.0e+07</td>
<td>1.1e+07</td>
<td>30</td>
<td>68.83</td>
</tr>
<tr>
<td>Crysis-1</td>
<td>3.8e+07</td>
<td>1.4e+07</td>
<td>40</td>
<td>33.92</td>
</tr>
<tr>
<td>Crysis-2</td>
<td>4.6e+07</td>
<td>3.7e+07</td>
<td>40</td>
<td>70.51</td>
</tr>
<tr>
<td>WIC-1</td>
<td>3.3e+07</td>
<td>8.0e+06</td>
<td>40</td>
<td>15.78</td>
</tr>
<tr>
<td>WIC-2</td>
<td>4.0e+07</td>
<td>8.4e+06</td>
<td>40</td>
<td>65.20</td>
</tr>
</tbody>
</table>

The experiments were performed on a laptop equipped with a 1.86 GHz Intel® Pentium® M Processor and 1.5 GB RAM. This processor supports Enhanced Intel SpeedStep® Technology and offers frequency scaling between five different frequency levels that correspond to 800 MHz, 1066 MHz, 1333 MHz, 1600 MHz and 1866 MHz. In order to obtain a precise processor cycle count, the cycle measurements were 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 the DVFS algorithms.

Selection of Game Plays: Quake II allows recording of game plays. These recordings include everything that is required (e.g., user input) to re-run exactly the same game play similar to a video, along with performing exactly the same computations. This allowed us to re-produce the measurements for different runs of the same game play under identical settings, but with different power management policies.

We recorded four short game plays among which two (i.e., Shooting-1 and Shooting-2) included highly dynamic scenarios involving events like enemy contact. The other two short game plays resembled an exploration phase of the game with comparatively low workload (i.e., Explore-1 and Explore-2). 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 relatively high CPU demand and workload variation. Several runs were 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 1, where the average workload $\bar{C}$ and its standard deviation $\sigma$ 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, Explore-1 and Explore-2 show the lowest $\bar{C}$ and $\sigma$. The minimum ($d_{min}$) and the maximum ($d_{max}$) percentage of frames missing their deadlines were obtained by running the processor at the highest (1600 MHz) and the lowest (800 MHz) frequency respectively. A certain percentage of frames miss their deadline even when the processor runs at the highest frequency, because the processor cannot support the maximum possible workload generated.

3.2 Hardware-based rendering setup

In the following, the hardware-based rendering setup is described, which is used to evaluate the prediction performance of games running on a HW-accelerated rendering system. Previously, rendering made up the major part of the total workload (in case of Quake II up to 90 percent). The advent of GPUs allowed game developers to offload most of the rendering workload from the CPU. Instead, the CPU is now increasingly used for complex artificial intelligence computations and realistic physics engines. Thereby, the workload characteristics of games have significantly changed.

The main advantage of the setup presented below over the previously presented approach is that (i) we can evaluate modern games that use hardware-accelerated rendering, and (ii) that we are no longer restricted to open source games. The current implementation allows...
applying our DVFS algorithm to any DirectX 9 based application and can be easily adapted to support DirectX 10/11, OpenGL* or OpenGL* ES. To incorporate our power management into recent closed-source games, we utilized the interface between the application and Microsoft’s DirectX rendering API. As shown in Figure 2, a game performs API calls to the Microsoft Direct3D run-time library to initiate the rendering of the scene. We intercepted all calls made by the game to the Direct3D run-time library. This was done using a technique called dynamic link library (DLL) injection provided by the Microsoft Detours Library [30]. When the game starts up, it loads the Direct3D library. The DLL injection inhibits the loading of the original DLL and instead forces the game to load our own library into its context. Consequently, instead of the original Direct3D functions, our code is executed. This enabled us to profile the game, to run our workload prediction and DVFS scheme and to render a visualization on top of the game (see [2] for a video).

A game signals the beginning of a new frame to the GPU with the DirectX command BeginScene. At the end of the frame, the EndScene together with the Present command is called, thereby informing the GPU that the frame can be shown. The interarrival time between Present commands gives us the frame execution time. Based on these execution times the predictor estimates the workload of the next frame and computes the required processor’s frequency. To actually scale the frequency of the processor, the processor’s Model Specific Registers (MSRs) have to be set accordingly. As this is only allowed in kernel mode, a driver is loaded at creation time of the DirectX device which allows setting the MSRs from user space (see Figure 2).

Additionally, this interception based approach allowed us to add an online visualization of e.g., the power savings and an user interface, which enables the user to influence the DVFS algorithm and modify parameters like the desired frame rate (see Figure 3 and [2] for a video).

For the experiments we used a desktop PC with an Intel® Core™ 2 Quad QX6700 with 2.66GHz, 2048 MB of RAM and a NVIDIA Geforce® 8800 FX graphics card and running Microsoft Windows XP®. The processor supports five different frequency levels that correspond to 1.6 GHz, 1.86 GHz, 2.1 GHz, 2.4 GHz and 2.66 GHz. We used the Windows system API functions QueryPerformanceCounter() and QueryPerformanceFrequency() to measure the frame execution times. In this work the frequency and voltage of the four cores was always scaled equally as the focus of this paper is on the evaluation of suitable workload predictors for game applications. We plan to extend this work and use the prediction results for power management of multi-core processors.

**Selection of Games:** We have chosen three popular games, each from a different genre, to evaluate the performance of our power management scheme: a first person shooter named Crysis’ from Crytek, a racing game named Need for Speed™ - Shift (NFS) from Electronic Arts and Ubisoft’s strategy game World in Conflict™ (WIC). The average workload $C$, the workload’s standard deviation $\sigma$, $d_{\text{min}}$, and $d_{\text{max}}$ are given in Table 1 for six recorded game plays (two for each game). The workloads’ standard deviation for games that use hardware rendering are significantly higher compared to the one’s in Quake II. The major difference between the two architectural setups is the composition of the workloads. A game’s workload is in general composed of rendering, AI, physics and game logic. Computations like AI or physics are not necessarily frame based and therefore can change more abruptly if measured in cycles per frame. The rendering workload on the other hand is of course frame based, and therefore changes are likely to be slower. In case of Quake II, up to 90 percent of the CPU’s time is spent for rendering. For games with hardware-rendering the CPU time is instead used for complex AI or physics computations. As these computations are not necessarily done on a frame basis, the workload no longer changes in a continuous manner. This explains the observed difference between the workload’s standard deviations, and this will later influence the choice of appropriate predictors.

This paper targets power management schemes for the CPU. If a game is mainly GPU-bound for the available CPU frequencies, then it could always be run with the lowest available frequency and no DVFS or workload prediction would be required. To present meaningful results, we have chosen realistic graphics settings for each game and verified that the games are not always GPU-bound. Figure 4 shows the resulting average frame rate of Crysis for the available processor speeds. It can clearly be seen that the game is not GPU-bound for the chosen settings (Resolution $1024 \times 768$, no Anti-Aliasing, all detail levels set to medium). Note that there still might be scenes for which a game becomes GPU-bound in-between. Here, it would be beneficial to come up with an additional power management scheme for the GPU.
3.3 Simulation Setup

The highly dynamic nature of game workloads and additional variations introduced by the underlying OS necessitate an exhaustive exploration of the space of workload predictor parameters, as will be elaborated later in this paper. Towards this, we developed a simulation environment for a systematic evaluation of the performance of the PID controller, the LMS and the ARMA-based workload predictor.

The algorithm for DVFS (see Section 2) is replicated in the simulation, where the processing of frames and the workload measurement is replaced by a workload model. This workload model is based on recorded workload profiles.

In contrast to video applications, in games the content of every frame and its workload depends on the user action and the processor frequency that was used to render the past frames: Let the $i^{th}$ frame at time $t[i]$ require $c[i]$ clock cycles and $f_j[i] \in F = \{f_1, f_2, \ldots, f_n\}$ be the corresponding processor frequency used to render the $j^{th}$ frame. The time $\Delta t[i]$ taken for rendering the $i^{th}$ frame is then given by $c[i]/f_j[i]$. 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 $f_j[i]$ (when the frequency is higher, more frames are used to “fill” a certain time interval).

This dependency prohibits the direct usage of recorded workload profiles. To get around this, we assume “linear” behavior of the workload profiles. Our experiments showed that this assumption is valid over small time scales, as considered in our case. Thus, for each available processor frequency $f_j \in F$, the corresponding workload profile $C_{f_j}$ is recorded and transformed from the frame number to the time domain by interpolating the missing values. For each frame that is pseudo-processed in simulation with frequency $f_j$, the corresponding workload profile $C_{f_j}$ is evaluated.

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 in terms of the performance metrics (see Section 2). The runtimes of the simulation compared to concrete runtimes of the games are given for different runs in Table 2. A speedup of 5512× is achieved with a Mathworks MATLAB® implementation, which clearly shows the advantage of using a simulation-based approach for tuning the controller parameters (gain values) of our workload predictors, as will be explained later.

<table>
<thead>
<tr>
<th>No. of Simulation Runs</th>
<th>$t_{sim}$ [s]</th>
<th>$t_{game}$ [s]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>116.4</td>
<td>78.7e+03</td>
<td>676</td>
</tr>
<tr>
<td>1600</td>
<td>206.3</td>
<td>31.2e+04</td>
<td>1515</td>
</tr>
<tr>
<td>442401</td>
<td>15650</td>
<td>86.3e+06</td>
<td>5512</td>
</tr>
</tbody>
</table>

Fig. 5. Prediction error in case of an unstable predictor

4 PID Controller Based Prediction

Now that we have explained our experimental setup, in what follows we describe various workload predictors and their evaluations. We start with the PID controller based workload prediction which has been successfully applied to various power management problems. In the following we are going to describe the theoretical background and challenges of using PID controllers for game workload prediction.

The input signal computed by a PID controller consists of three components,

1) Proportional: $P_{comp}(t) = K_P \cdot e(t)$
2) Integral: $I_{comp}(t) = \frac{1}{T} \cdot \sum_{t=0}^{t-1} e(t)$
3) Derivative: $D_{comp}(t) = K_D \cdot e(t) - e(t-T_D)$

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 $PID_{output}(t) = P_{comp}(t) + I_{comp}(t) + D_{comp}(t)$. Let $c[i]$ and $\tilde{c}[i]$ 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 $\tilde{c}[i+1]$ of the $(i+1)^{th}$ frame by utilizing the actual workload $c[i]$ of the $i^{th}$ frame and the PID control signal $PID_{output}(t)$, i.e., $\tilde{c}[i+1] = c[i] + PID_{output}[i]$. Towards this, we compute $PID_{output}[i]$ with $e[i] = c[i] - \tilde{c}[i]$ being the error signal and

$$PID_{output}[i] = K_P \cdot e[i] + \frac{1}{K_I} \cdot \sum_{j=0}^{n} e[i-j] + K_D \cdot \frac{e[i] - e[i-1]}{\Delta t[i-1]}$$

4.1 PID Controller’s Stability

The choice of PID controller gains $K_P$, $K_I$ and $K_D$ is crucial for the performance of the predictor. The predictor directly influences the choice of the processor’s frequency. Ideally, it will choose a frequency sufficient to complete the frame just in time and therefore with the lowest possible power consumption. If the predictor’s parameters are chosen incorrectly the predictor may become unstable. A typical plot of an unstable controller-based prediction is shown in Figure 5. The prediction error and hence the predicted workload starts to oscillate and becomes infinitely large over time. In terms of processor frequency settings, this results in a periodic switching between highest and lowest frequency. This of
course is highly inefficient in terms of power consumption and will also lead to a considerable loss in game quality.

Figure 6 shows the distribution of stable controller gains for a particular game play (i.e., Shooting-2). It may be noted that only a small portion of the entire space of the controller gains ensures the predictor’s stability. The controller gains with reasonable prediction quality (i.e., smaller than 10% frame deadline miss) are indicated by the points. Note that the number of such points is limited and distributed over the entire space of stable controller gains. Hence, identifying gain values that lead to a controller with acceptable performance is a non-trivial task. In the following subsection we show how suitable gain values may be chosen.

4.2 PID Performance Space

As mentioned in Section 2, the performance of the PID controller and all other workload predictors is quantified using two metrics, the percentage of frames missing their deadline and the average power consumption. We obtain a performance space by plotting the values of the metrics corresponding to various sets of PID controller gains. For example, Figure 7 shows the resulting performance space of the PID controller for the Quake II game play Level-2. Every point corresponds to results obtained with a specific choice of PID controller gains. The Pareto-front marks the optimal choice of points. These are the points of interest and are found by an exhaustive simulation. It may be noted that the average power consumption, here given for the laptop, 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. The same behavior has been observed for the Desktop PC. 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 chose the 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.

Software-based Rendering: To investigate the influence of workload variations, we ran exhaustive simulations for the Quake II game plays listed in Table 1. 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 7. For each game play we selected the set of gains resulting in the lowest rate of frame deadline misses, i.e., \( \text{set}_{\text{Explore-1}} \) is the optimal set for game play Explore-1 (see Figure 8).

In a complex game it is very unlikely that a player generates the same workload twice. Hence, an important issue is the predictor’s robustness to changes in the workload characteristics. To evaluate this robustness we used the PID gains, which were optimized for one game play, for the workload prediction of the remaining game plays. As shown in Figure 8, we observed inferior performance if we used gain values that were optimized for one game play for other game plays. For example, using \( \text{set}_{\text{Explore-1}} \) for Level-2 will increase the percentage of frame deadline misses by 7.3%.

The PID gains can also be optimized with all the game plays taken together, i.e., \( \text{set}_{\text{ALL}} \) in Figure 8. This set has been computed by merging the performance spaces of all game plays. Nevertheless, the controller gains again need re-optimization in case a new game play is considered. For example, we optimized the PID gains by considering all the game plays listed in Table 1.
except for the game play Level-2. These PID gains were then used for Level-2, which resulted in an unstable predictor for Level-2.

**Hardware-based rendering:** The same measurements were also repeated for hardware rendering. Unfortunately, the option to record game plays like in Quake II is not offered by any recent games that use hardware rendering. To address this issue of reproducibility, the predictor's performance was simulated, first using very long pre-recorded workloads. Once a good setting was found the gains were evaluated online in the game and results were thereby verified.

Figure 9 shows the resulting percentages of frames missing their deadlines for the DirectX games listed in Table 1. The gains have been tuned individually for six game plays in total (two per game). A cross-validation was performed with the other game plays. As can be seen in the figure, the performance drops significantly when a set of PID gains, which has been tuned for one particular game play, is used for a different game or game play, e.g., using set_WIC−1(©) for NFS-1 increased the number of frame drops by 12.6%. Further, for all games the controller became unstable after a particular number of frames.

Based on the measurement results from both - the software- and hardware-rendering setup - we conclude that the PID-based predictor’s performance depends on the nature of the game play and hence the controller gains should be optimized for each game play individually. Since all game plays cannot be known in advance (because they depend on the player), we conclude that PID-based prediction, which was successfully applied to video power management, is not suitable for game applications in real-life settings. Instead, a robust predictor is required, that needs to be calibrated only once, and that thereafter provides good performance for a priori unknown game plays. These outcomes motivate the use of a self-tuning algorithm.

### 5 LMS Linear Predictor

The Least Mean Squares (LMS) Linear Predictor [21] 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 weights over several iterations.

We used it to estimate the workload \( \hat{c}[i + 1] \) of the \((i + 1)^{th}\) frame by utilizing the actual workloads of the previous frames. Towards this, we represented the predictor’s 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

\[
\hat{c}[i + 1] = \sum_{k=0}^{n-1} w[k]\cdot c[i-k] = W[i]^{T}C[i],
\]

where \( n \) is the predictor’s order, \( w_k \), for \( k = 0, \ldots, n-1 \), are unknown predictor coefficients and

\[
W[i] = [w_0[i], w_1[i], \ldots, w_{n-1}[i]]^T,
\]

\[
C[i] = [c[i], c[i-1], \ldots, c[i-n+1]]^T.
\]

The goal is to learn \( W[i] \) adaptively such that \( \hat{c}[i + 1] \) in Equation (1) results in the minimum error \( e[i] = c[i + 1] - \hat{c}[i + 1] \). Therefore, from Equation (1) we have, \( e[i] = c[i + 1] - W[i]^{T}C[i] \). The unknown predictor coefficients are initialized to 0 and after each prediction step, the coefficients are updated according to Equation (2)

\[
W[i + 1] = W[i] + \mu \cdot e[i] \cdot C[i],
\]

where \( \mu \) is the learning rate.

In order to reduce the sensitivity of the learning process to the choice of the learning rate \( \mu \), we used a normalized version of Equation (2) that is given by

\[
W[i + 1] = W[i] + \frac{\mu \cdot e[i]}{||C[i]||^2} \cdot C[i],
\]

with \( \mu \) being between 0 and 2.

For an accurate modeling of the system and prediction of the workload, the coefficients \( W[i] \) should converge after a sufficient number of iterations. However, this convergence is not guaranteed: If the prediction error \( |e[i]| \) goes repeatedly above the bound

\[
\frac{2||C[i]||^2}{\mu \cdot c[i-k]} \cdot |w_k[i]|,
\]

the LMS weights might not converge (see [1] for a derivation). Not converging and therefore continuously increasing weights will result in an unstable predictor and a high prediction error. Convergence, on the other hand, implies accurate approximation of the system using an LMS Linear Predictor.

**Software-based rendering:** The performance of LMS Linear Prediction is determined by two parameters, the prediction order \( n \) and the learning rate \( \mu \). The order \( n \) of the predictor indicates the number of workload values of the previous frames being utilized to model its output (see Equation (1)). If the order of the predictor is too low, e.g., \( n = 1 \), then its coefficients will not be able to accurately model the output and will not converge.
This in turn results in a significant reduction of the predictor’s performance (see Figure 10). However, an unnecessarily high order, i.e. \( n > 11 \), does not improve the performance any further. We experimentally found that an order of 10 guarantees a good performance for all Quake II game plays.

Using this order, the boundaries given by Equation (4) were seldom violated (at maximum by 13.12% for Level-2) thereby indicating a convergence of the weights. Figure 11 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 Quake II game plays during the simulation to verify that convergence is preserved under changing system dynamics. Therefore, Quake II system dynamics can be accurately approximated by an LMS Linear Predictor.

The second parameter that affects the quality of prediction is the learning rate \( \mu \). As seen in Figures 10 and 12, a very small learning rate \( \mu \) results in a high prediction error 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.

To evaluate the performance of LMS, we determined the \( \mu \) that resulted in the optimal performance for each game play (see Figure 13). Compared to the PID based approach, the percentage of frames missing their deadlines is slightly increased if the LMS predictor was used (at maximum by 1.8% for game play Massive-1). In order to evaluate the robustness of LMS, we then used these game-specific \( \mu \)'s for the prediction of the other game plays. Compared to the PID-based predictor, the robustness of LMS is significantly better, as (i) the maximum deviation from the optimal performance was observed to be 3.99%, whereas for the PID based approach a deviation of 7.3% was observed, and more importantly, (ii) for no combination of game plays the predictor became unstable. Based on these results we conclude that the LMS predictor is an improved choice for modeling Quake II workloads.

**Hardware-based rendering:** For Quake II workloads it was possible to observe convergence of the LMS weights. However, evaluating Equation (4) for hardware-rendering based workloads revealed that for nearly all game plays a significant percentage of weights did not satisfy this condition. Figure 14 shows how the weights evolved over time for the game play Crysis-1. Clearly, convergence of the weights can not be observed in this case, though the search space of the learning rate and the predictor’s order has been exhaustively evaluated.

As convergence is required for an accurate and stable workload prediction, the LMS Linear predictor is not the correct choice for highly varying workloads as in the case of hardware-based rendering. Using the predictor would lead to instability and ultimately our DVFS algorithm would only select the highest and lowest processing frequencies (see Section 4.1). To overcome this problem we now introduce a generalization of the LMS Linear Predictor, the Auto-regressive Moving Average Model.
6 Auto-regressive Model

In literature, an Autoregressive Moving Average (ARMA) model is a generalization of the LMS Linear Predictor given by Equation (1). Note that another generalization of LMS Linear Predictor could have been to use a non-linear model or a linear model with non-linear functions of past workloads as predictors. However, fitting such models is complicated and time consuming and in the following we will show that already the much simpler ARMA model and even a sub-class named AR models suffices to accurately and efficiently predict game workloads.

The ARMA model is appropriate when the system under consideration can be thought of as a stationary time series process whose output depends linearly on past values, as well as on independent inputs introduced to the system. We have tested our workload data for stationarity using the KPSS [23], the Dickey-Fuller (DF) and Augmented Dickey-Fuller (ADF) [14] tests. Due to space constraints we would like to refer to the supplemental material of this work [1] for more details about the different tests. All three of these tests suggest that the gaming workloads (both, software- and hardware-rendering based) are of stationary nature. Thus the first criterion for successfully applying ARMA to our problem is fulfilled.

The ARMA($n$, $m$) model is given by the following equation:

$$c[i + 1] = \sum_{k=0}^{n-1} w_k c[i - k] + \sum_{j=0}^{m-1} v_j c[i - j] + \epsilon[i + 1],$$  (5)

where $\epsilon[i]$ are the white noise error terms and $\epsilon[i + 1]$ is the error in the linear ARMA representation of the current frame’s workload $c[i + 1]$. We can compute a prediction based on the workloads $c[i]$, the previous prediction errors $\epsilon[i]$ and the model parameters: the autoregressive coefficients $w_k$, the autoregressive order $n$, the moving average coefficients $v_j$ and the moving average order $m$.

An Autoregressive (AR) process is a special case of an ARMA process where the model does not account for past pulses i.e., all $v_j$’s are equal to zero. AR processes are sometimes preferable as they are easier to interpret. Also, using AR models is not as restrictive (compared to using ARMA models) as it might seem, as a large class of ARMA models can be expressed as infinite order AR models, which are known as invertible ARMA models.

In the case of the LMS linear predictor, the gains $w_k$ change over time as they are learned online based on the learning rate $\mu$ and Equation (2). Under high workload variations, such online systems might adapt to the data too quickly and therefore can be misled (resulting in unstable behavior). In contrast, for ARMA and AR, the set of parameters is fitted offline only once as described in the following, and does not change over time.

6.1 Fitting ARMA and AR Models

Before the ARMA and AR models can be used to predict the game’s workload, suitable model parameters ($w_0, \ldots, w_{n-1}, v_0, \ldots, v_{m-1}$) and model orders $n$ and $m$ have to be determined that guarantee a good prediction performance.

Model parameters: The ARMA model parameters can be fitted using a maximum likelihood method that maximizes the joint probability function of the model given the data. We chose to perform this maximization through a subspace method using an iterative Gauss-Newton algorithm (using the system identification toolbox from MATLAB) [25]. For large data sets this approach is usually slower in implementation compared to the steepest gradient method employed by the LMS filter. However, determining these parameters needs to be done only once and is done offline based on pre-recorded game workloads. As we discuss later, once these parameters are determined, they remain constant not only across game plays, but also across different games.

A common approach to tune parameters of AR models is the non-iterative, least-squares method [29] which is as well used by the MATLAB System Identification Toolbox. All results presented in this paper are based on models which have been tuned using this toolbox.

Model Order: Using the methods described above, we trained the model parameters for each DirectX-based game/game play individually with $n, m \leq 100$. Figure 15 shows the gained results for an AR($n$) model. We observed that an order greater than 10 for both, $n$ and $m$ only slightly improved the predictor’s performance. Hence, for the sake of simplicity and to avoid an increased algorithm’s complexity, we restricted the remainder to orders $n, m \leq 10$. The best performance for $n, m \leq 10$ was achieved with an ARMA($10$, $9$) model. It could be observed that the performance of an AR($10$) model provides a very good fit which is close to the ARMA($10$, $9$) model, and their forecasts are also close. Hence, for the sake of simplicity and in view of the previous discussion, we restrict the remainder of our analysis in this paper to AR processes of order 10.

6.2 Evaluating AR models

Figures 16 and 17 depict the performance of the AR($10$) models for the software and hardware-rendering approaches respectively. For each game and game play, a model was individually derived with the MATLAB toolbox. The model was then tested with the corresponding
game play and with other games and game plays. It can be seen that the best performance achieved with AR(10) is approximately the same as when using an individually tuned PID controller. In case of an AR model, however, no instability has been observed as it can happen for the PID controller. As can be seen in Figure 17, only a small performance difference (at maximum 0.68%) can be noticed when the model, tuned using the workload from one game or game play, is used for the workload prediction of a different game or game play. In comparison, with the PID-based prediction a significant difference of up to 12.6% of more frames missing their deadline is observed (see Figure 9). The performance difference is close to zero if the models are tuned on a per game basis, i.e. using the individually tuned NFS-2(+) for NFS-2 itself results in 10.3% of deadline misses. Using NFS-1(−)-based models for NFS-2 results in 10.5%.

We conclude that AR models are robust enough to workload variations such that the model can be fitted only based on one sample game play. As shown, the resulting model will provide a good performance, even if used for game plays which it has not been optimized for. A further generalization (e.g., to non-linear models) as described at the beginning of Section 6 is not required.

7 Power Measurement Results

So far, the evaluation of the predictors’ performances focused on the prediction quality in terms of percentage of frames missing their deadline. The evaluation suggests the suitability of autoregressive models for game workload prediction. In addition to game quality i.e., minimizing the number of frame deadline misses, we are also concerned with minimizing the average power consumption of the processor. In this section we present the experimental setup and results of the overhead and power measurements based on the proposed DVFS algorithms. We show that power savings of up to 35.8% can be achieved while maintaining a desired gaming quality.

7.1 Power Management Overhead

Any power management technique involves overhead, which in our case consists of the time consumed for the application profiling, the computation time for the workload prediction and the settling time of the voltage and frequency regulator. In case of the software-rendering based setup, the source code has been directly instrumented and the profiling overhead merely consists of reading the time stamp counter (TSC) values. This overhead can be completely neglected when compared with a game frame’s workload. For the hardware-rendering based setup, all DirectX calls are intercepted. On an average we observed an overhead of 2.34 ms per frame. With a target frame rate of 40 fps this equals to 9.3% of the available time per frame. As we will show in the following, despite this overhead we can achieve considerable power savings. The overhead is mostly caused by the interception itself as every Direct3D command first calls our proxy library before being forwarded to the original Direct3D library. It could be reduced to negligible levels if the DVFS algorithm was incorporated directly into the DirectX graphics API or if the API offered an interface. As a consequence the power savings could be even higher.

<table>
<thead>
<tr>
<th>Predictor</th>
<th># of Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>15.0</td>
</tr>
<tr>
<td>LMS(10)</td>
<td>323.9</td>
</tr>
<tr>
<td>AR(10)</td>
<td>26.3</td>
</tr>
<tr>
<td>ARMA(10,10)</td>
<td>52.8</td>
</tr>
</tbody>
</table>

The overhead for the different workload prediction techniques is given in Table 3. Compared to the number of cycles that are at least available to the CPU per frame this overhead can be completely neglected.

The overhead for scaling an Intel® Core™2 Duo E6850 processor is available from literature with 30.3 μs at maximum [34]. If we assume frequency switches at 40 Hz this scaling overhead makes up 0.12% of a frames processing time.

7.2 Power savings with software rendering

In the software-based rendering setup (used in Quake II), the power measurements were performed at the output of the laptop’s AC Adaptor. A Texas Instruments MSP430™ microcontroller 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 \( P \) was then calculated according to 
\[
P = \frac{1}{T} \sum_{t=0}^{T} v(t)i(t),
\]

where \( l \) is the duration of the game and \( 1/T \) 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.

This setup provides measurements corresponding to the power consumption of the entire laptop. During the measurements the battery was removed from the laptop to avoid measuring the power consumed for recharging the battery as well. Additionally, we ensured that the laptop settings remained constant during all 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 4 shows the laptop’s average power consumption for all available frequencies of the processor. In our simulation, these recordings were used to approximate actual power consumption. As we measured the system’s total power consumption, our measurements included the power consumption of, for example, memory or front-side bus, which highly depends on the load of the system [39]. 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 total power (compared to maximum power consumption of the laptop) may be saved at the maximum by running the system at the lowest frequency at all times. This, however, will result in an unreasonably high percentage of deadline misses, significantly reducing the game’s quality.

We incorporated all the proposed predictors into the Quake II source code. Table 5 gives the measured average power consumption for each game play using the individually tuned PID-based predictor ($\overline{P}_{PID}$), the LMS Linear Predictor ($\overline{P}_{LMS}$) with a $\mu$ of 0.074 and individually tuned AR(10) models ($\overline{P}_{AR}$). The observed difference among the different predictors in terms of average power consumptions is negligible. The last column shows the achieved power savings of AR(10) compared to the power consumed if the laptop is clocked at the highest frequency. It may be noted that between 24.0% and 35.8% of power is saved depending on the characteristics of the game play.

Note that as presented in Section 4.2 and depicted in Figure 7, a non-optimal choice of PID gains affects the percentage of frames missing their deadline significantly more than the average power consumption. The cross-validation for PID and AR(10)-based predictors experimentally confirmed this behavior. Only small variations in respect to the optimal average power consumption were observed: 2.53% and 1.34% for PID and AR(10) respectively. For each game play, not only the power savings were similar, but also the game quality which can be seen from Figures 8 and 16. However, as shown in Section 6 it is sufficient to tune the AR model only once, whereas the PID gains need to be tuned for each game play individually to avoid large performance drops. This individual tuning is not practical in a real-life scenario.

### Table 4

<table>
<thead>
<tr>
<th>Frequency [MHz]</th>
<th>800</th>
<th>1066</th>
<th>1333</th>
<th>1600</th>
<th>1866</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power [Watt]</td>
<td>21.1</td>
<td>23.3</td>
<td>25.8</td>
<td>29.1</td>
<td>33.0</td>
</tr>
</tbody>
</table>

### Table 5

<table>
<thead>
<tr>
<th>Game Play</th>
<th>Average Power Consumption [W]</th>
<th>Savings using AR [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\overline{P}_{PID}$</td>
<td>$\overline{P}_{LMS}$</td>
</tr>
<tr>
<td>Explore-1</td>
<td>21.42</td>
<td>21.6</td>
</tr>
<tr>
<td>Explore-2</td>
<td>21.71</td>
<td>21.9</td>
</tr>
<tr>
<td>Shooting-1</td>
<td>23.1</td>
<td>23.6</td>
</tr>
<tr>
<td>Shooting-2</td>
<td>25.2</td>
<td>24.8</td>
</tr>
<tr>
<td>Massive-1</td>
<td>25.1</td>
<td>23.9</td>
</tr>
<tr>
<td>Level-2</td>
<td>23.7</td>
<td>23.2</td>
</tr>
</tbody>
</table>

#### 7.3 Default Linux Power Management

Linux is equipped with a widely-used Ondemand Governor [33] for power management. We ran the Quake II 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 AR-based Predictor achieves power savings of up to 35.8%. As Quake II is programmed as an endless loop, the Ondemand Governor will always detect high system utilization. Consequently, voltage/frequency scaling cannot be enabled during most of the time.

#### 7.4 Power savings in case of DirectX-based Games

For the DirectX-based approach we incorporated the proposed predictors into the proxy DLL (see Section 3.2). As both, the PID-based Predictor and the LMS Linear Predictor were not stable the measurement results are not shown here. Figure 18 gives the ratio at which the available frequencies of the processor were used for the three games Need for Speed, Crysis and World in Conflict. The results are given for the autoregressive model and an oracle predictor. The oracle predictor is a purely theoretical construct and is assumed to know the full future. Hence, it gives the maximum possible power savings with the smallest possible number of frames missing their deadlines. The AR(10) model in comparison selects the frequencies with almost the same distribution. This establishes that these policies result in a close to optimal energy consumption of the processor.
With the aim of DVFS-based power management, in this paper we have proposed workload prediction schemes for game applications whose parameters need to be tuned once during an offline phase and may then be used for both – game plays, as well as new games that are not a priori known. Hence, our focus has more been on the robustness of the prediction scheme, rather than its optimality. In other words, occasionally, a predictor that is hand-tuned for a particular game play might outperform our proposed offline-tuned predictor. However, the overall prediction quality of the offline-tuned predictor was comparable to a hand-tuned predictor.

Towards this, we studied several games – Quake II, whose source code is available, as well as more modern closed-source games like Crysis, Need for Speed (NFS) and World in Conflict (WIC). We showed that concerns about stability and tedious hand-tuning of parameters make known PID-based workload prediction schemes unsuitable for any of the considered games. We next studied an LMS Linear Predictor, which worked well for Quake II with software rendering, but not for Crysis, NFS or WIC that rely on hardware-rendering support. For these games, we showed that the Autoregressive Moving Average (ARMA) model and its simplified version, the AR model, address the concerns that arise in a real-life power management setup, viz., that not all game plays and game applications are a priori known. Our results are consistent across all evaluated games and particularly attractive because parameters of an AR-model, tuned using a set of game plays, provide good gaming quality as well as power savings when applied not only to different game plays, but also different game applications. The similarity between results obtained using AR and ARMA also show that more complex non-linear models or linear models with non-linear functions of past workloads are not required.

Further, we have exploited the software (in particular the graphics processing) architectures of modern games by intercepting API calls made by the game application to the graphics library. These were used to estimate the execution times of game frames, which were then used for workload prediction. This enabled us to apply our power management scheme to a variety of modern closed-source games, whereas all previously known techniques required modifications of the game’s source code which is not available for most modern games. The applicability of our scheme to closed-source games ensures its practical relevance. Further, our techniques differ from workload prediction and DVFS schemes known for video processing in two major ways. First, we showed that PID controllers that were successfully used for video applications, are not practical for game applications, which require different workload prediction schemes, such as the ones proposed in this paper. Second, we exploited the graphics processing architectures of games (through API call interception), that do not arise in video processing. It is possible to combine our results with frame-workload prediction schemes that analyze the contents/structure of game frames as proposed in [19], where a PID-based scheme was combined with frame structure analysis. However, workload prediction based on analyzing the contents of a frame is mostly relevant for software-based rendering schemes. This is because only the rendering (and not AI or game physics) workload may be estimated the contents of a frame. Hence, such schemes will not be relevant for more modern games that rely on hardware support for rendering.

As future work we plan to investigate power management of GPUs, and in particular the combined power management of both – the GPU and the CPU. Additionally, combining this with power management schemes for other subsystems of a mobile device such as its wireless interface and display will be meaningful.

**References**

[1] Appendix (supplemental material) to this paper.


* Other names and brands may be claimed as the property of others.
IEEE TRANSACTIONS ON COMPUTERS


Benedikt Dietrich received his Master’s in Electrical Engineering from TU Munich in 2009 and joined the Institute for Real-Time Computer Systems to pursue his PhD degree. Since then he has been working on power management algorithms with a focus on graphics-intensive interactive applications. Here, he studies workload prediction for closed-source games and how the game’s communication with the underlying OS can be leveraged to predict interactive workloads.

Dip Goswami obtained his PhD in Electrical Engineering from the National University of Singapore (NUS) in 2008. Subsequently, he was a Post-Doctoral Fellow in the School of Computing at NUS and then an Alexander von Humboldt Fellow at TU Munich, where he is now a senior researcher. His research interests span across embedded systems, control theory and robotics. He has more than 30 publications on these topics including Best Paper Awards at ASP-DAC 2011 and EUC 2010.

Apratim Guha received his PhD in Statistics from the University of California, Berkeley. He is currently an Associate Professor in the Indian Institute of Management, Ahmedabad. His research interests include time series analysis, categorical data analysis, information theory and application of statistics in various computer science areas. He has authored papers in various journals including Statistica Sinica, Neural Networks, Computational Neuroscience and Journal of Statistical Planning and Inference.

Samarjit Chakraborty is a Professor of Electrical Engineering at TU Munich, where he holds the Chair for Real-Time Computer Systems. Prior to joining TU Munich, he was an Assistant Professor of Computer Science at the National University of Singapore from 2003-2008. He obtained his PhD from ETH Zürich in 2003. He works on various aspects of system level design and analysis of embedded systems and has more than 150 publications in this area.

Matthias Gries received the Doctor of Technical Sciences degree from ETH Zürich in 2001. He was a post-doc in the Computer-Aided Design group at UC Berkeley from 2002 - 04. He is currently a research scientist at the Germany Microprocessor Lab of Intel Labs. His research interests include design for user experience, design automation, microarchitecture innovations and mitigating the memory wall. He has co-authored more than 40 papers and holds 6 patents. He is a member of the IEEE.