Power Estimation for Interactive 3D Game Using an Efficient Hierarchical-Based Frame Workload Prediction

Da-Jing Zhang-Jian, Chung-Nan Lee, Ing-Jer Huang and Shiann-Rong Kuang
Department of Computer Science and Engineering, National Sun Yat-Sen University, Taiwan, R.O.C.
E-mail:salmoner.tw@yahoo.com.tw, cnlee@mail.nsysu.edu.tw, ijhuang@cse.nsysu.edu.tw, and srkuang@cse.nsysu.edu.tw

***ABSTRACT*— In this paper we propose a novel hierarchical-based workload prediction algorithm that integrates the proportion control, integral control, derivative control, frame structure, and analysis of a game or benchmark to make game workload predictions and perform power estimation. By hierarchical analysis of a game or benchmark, one can know the complexity of a future frame of interactive 3D games in advance. Hence, the prediction error is small. Experimental results show that the proposed algorithm requires only 41.2 ms of extra workload, but provides an improvement of more than 15.7% in cycle count estimation as compared to competing algorithms.**

I. INTRODUCTION

With the growing popularity of mobile devices in recent years, interactive 3D games are increasingly being transferred from desktops to mobile devices (e.g., PDAs, cell phones, and portable game consoles). However, because of limitations in computational capacity and power limitations, 3D graphic (3DG) applications need to support 3D real-time rendering and minimize power consumption to be run on embedded systems. These requirements have led to growing interest in power management and to efforts toward enhancing battery life.

Though different applications have different performance and power workload, most 3DG applications are computationally intensive and power-consuming. For mobile devices, it is necessary to provide users with high-quality 3DG experience under limited power and processing. Unfortunately, 3DG application workloads vary in significantly scene change. Dynamic Voltage and Frequency Scaling (DVFS) [1][2] and Dynamic Frequency Scaling (DFS) are usually used to reduce power consumption and make power management by changing the processor frequency based on the requirements of an application, e.g. for fix-duration tasks, in particular during periods of low utilization as waiting, the results in a proportional reduction of energy use in mobile devices. In addition to the process frequency the processor voltage can also be changed to reduce power consumption.

Recently many workload prediction techniques have been proposed such as history predictor, Proportion Integral Derivative (PID) predictor [3], frame structure-based predictor [4], signature table predictor [5], and hybrid predictor [6]. The history-based predictor uses the average of the workload of certain amount of frames to estimate the workload of next frame, and it is the simplest one that can be easily implemented into hardware. Based on the feedback from recent prediction errors, PID consists of three components (Proportional control, Integral control and Derivative control), where the proportional control measures the variation in predicted workload and actual workload. The integral control measures the variation based on the sum of the recent workload difference. Finally, the derivative control measures the change of frame rate in the process. According to the workload variation between the actual workload and the predicted workload from the proportional, integral and derivative control, one can predict the next estimated frame workload via the PID predictor.

The 3DG applications have a minor difference in adjacent frames but scene change, the signature table predictor needs to recognize whether the current frame is similar to the previous frame and then predict the workload of current frame. The signature table predictor uses four parameters such as average triangle area, triangle counts, average triangle height and vertex counts obtained from the triangles in the signature buffer to construct signatures. The signature concatenates these four parameters into a string and needs to find the best matching signature from the signature table with the smallest distance metric [5].

Based on Intel® VTune™ [3][4], Gu and Chakraborty used a theory-based DVS to scale the operating frequency and voltage of the processor to match a varying computational workload as closely as possible. Moreover,

they proposed a hybrid DVS scheme which combines frame structure-based and PID predictor to estimate the frame workload. The frame structure-based workload in rendering a game frame is roughly linearly correlated with its rasterization workload which is generated by processing the different objects, e.g. brush models and alias models. The existing techniques can be used to do frame workload prediction, but they often have more error when the 3DG application workloads vary in scene change.

Some power modeling analysis [7][8][9] for systems components provide trade-off between power estimation accuracy and computation workload, e.g. Nam et al. use extra hardware circuit to predict the workload [8]. In order to get better understand the workload/complexity of an application, one can use a suitable tool such as NVperfHUD [10], gDEBugger [11] or Intel®VTune™ [3] to find the bottleneck of graphics applications and game workload. Unfortunately, many performance tools [12][13] and benchmarks[14][15] are only developed for desktop computers and OpenGL [16], e.g. SPMark04 [17] and 3DMarkMobile06 [18]. Furthermore, these tools just support special hardware namely the platform dependent (e.g. Intel®VTune is for Intel's CPU) and they can not analyze power consumption to come up with a power management plan. In order to overcome these problems and conduct on-line power management, the proposed tool [19], Graphics Performance Tuning Tool (GPTT), is defined to be the extension functions of OpenGL ES [20] and is embedded in standard graphics library for measuring the performance of GPU on embedded systems. GPTT solves the problem that specific performance tool is only supported by a specific platform. Moreover, for doing power management GPTT can catch and visualize the statistics information of each part of rendering pipeline, then developers could easily get the performance information they need without conforming to a specific platform.

In this paper, we propose a hierarchical-based workload prediction algorithm to yield a better prediction for scene change in an interactive game. The rest of this paper is organized as follows. Section II presents the proposed system architecture for power management. The hierarchical-based workload prediction and performance evaluation are discussed in Section III and Section IV, respectively. Finally, conclusion and future work are given in Section V.

II. SYSTEM OVERVIEW

The overall of system architecture for power management as shown in Fig. 1 includes application, frame workload prediction, power management and software/hardware layers.

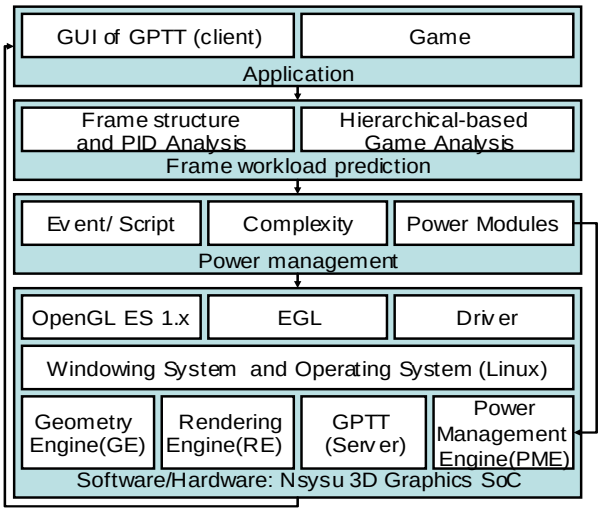


Fig. 1 System architecture for power management

Application layer consists of the game and graphics user interface (GUI) of GPTT, the analysis of game including scene change, variable camera etc., and the hierarchical tree of a game that will be discussed in Section III. Additionally, the GUI of GPTT displays the run time frame information such as triangle counts, pro-triangle counts from the server for the frame workload prediction.

In order to get better analysis for frame workload prediction layer, one can analyze the game and store the hierarchical tree information of the game into a database. The variations of frame workload can be predicted by the hybrid algorithm including PID and frame structure-based. Moreover, according to the complexity of the different events in the power management layer one can find the corresponding module in advance and select appropriate voltage (or frequency). The detailed information about the frame workload prediction and power management layers will be discussed in Section III.

In the software/hardware layer, the system collects information via GPTT during the run time, such as triangle counts from the Geometry Engine (GE), pixel counts from the Rendering Engine (RE), and the current frame workload, and uses PID and frame structure-based workload prediction algorithm in the frame workload prediction layer, then checks position of the user in the hierarchical tree to estimate the next frame workload in advance and finally selects the corresponding power module in the power management layer.

Functions of GPTT is a unique feature in our Nsysu 3DG core [21] during development stage which is integrated within the SoC for high-end 3DG products. It is a cross-platform to facilitate real-time profiling, debugging and performance measurement/tuning and collects real-time

statistics of a 3DG application running in software or hardware. Via such on-time information the developer can investigate the interaction between applications, workload prediction algorithms, GE and RM components, and come up with a power management plan for low-power systems.

The system architecture of GPTT can be divided into two components: One is the debugging capability in the embedded system (server) and the other is GUI for performance (client) observing of each part of rendering pipeline, e.g. transformation, lighting or fragment operations. The debugging capability is defined to be the extension functions of standard graphics library, gl.h, to realize OpenGL ES. Developers could insert these functions as using extension functions in OpenGL ES to their source codes, then display performance information via GPTT. Currently, not only one can print the performance information in the text file but also one can use the client-server framework for decreasing unnecessary burden to send the performance statistics to client, and display the different information. The GUI is responsible to collect and arrange information coming from rendering pipeline and to display some information, e.g. triangle counts, pro-triangle counts or pixel counts.

The items to be measured as listed in Table 1. Main information such as processed triangles from GE and processed pixels from RE. Fig. 2 shows the screenshot of a downtown benchmark and the performance results respectively, one can observe some performance information which displays triangle counts, pixel counts, computing time ratio, and understand the complexity benchmark from frame to frame by the variation of the curve as illustrated in the right part of Fig. 2. In addition, the information are helpful in workload prediction and power management. The detailed frame workload prediction will be discussed in Section III.

Table 1
Measurement items for software

1. Primitives	12.Lighting-module processed time
2. Processed primitives	13.Pixel processed time
3. Pixels	14.Utility rate of each per-fragment
4. Processed pixels	15.Buffer bandwidth
5. Triangles per second	16.History of functions call
6. Pixels per second	17.Video memory utility rate
7. Frames per second	18.GPU idle time (hardware only)
8. Screen resolution	19.Driver idle time (hardware only)
9. Texture size	20.Batch
10.Frame processed time	21.# of draw functions call
11.Transformation processed time	

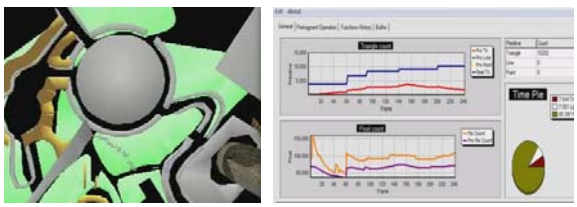


Fig. 2 Screenshot of benchmark and performance result for the Downtown benchmark

III. THE HIERARCHICAL-BASED WORKLOAD PREDICTION FOR POWER MANAGEMENT

In this section, we introduce a hierarchical-based workload prediction algorithm to predict the next frame workload by collecting frame workload information, including the frame workload, triangle counts and pixel counts and further reduce prediction error. Based on the complexity of the applications in the on-line game or mobile game can be obtained from analysis in advance, and by detecting the avatar position in the run time. The complexity of the current event one can predict the frame workload trend and select a corresponding module in next fame/second. The detailed description about the hierarchical-based game analysis and the prediction of frame workload will be discussed in Subsection III.A. and Subsection III.B, respectively.

A HIERARCHICAL-BASED GAME ANALYSIS

A game engine is a software system designed for the creation and development of computer games. Its core functionalities typically include a rendering engine for 2D/3D graphics, physics engine, animation, characters, 2D/3D sprite and 3D scene graph and so on. The task to process a frame is to process geometry throughput, lighting, texture and rendering etc. As illustrated in Fig. 3, a game system can be divided into the game player layer and game engine layer. According to the analysis of different game player layers (e.g. combat, story or script, AI and trading system), one can obtain the workload of each script. In addition, the different element of game engine layer often has different workload in a scene, e.g. shot change, moving object, avatar position and variable camera, by using these information will lead to a better frame workload prediction.

For experiment convenience we create a game alike benchmark, which consists of four stories and seven events as illustrated in Fig. 4**.** The different nodes of an n-branch hierarchical structure contain n different events which can be observed according to the avatar position or player information in 3D environment.

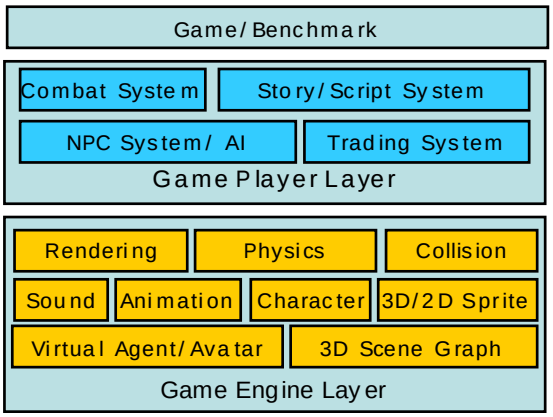


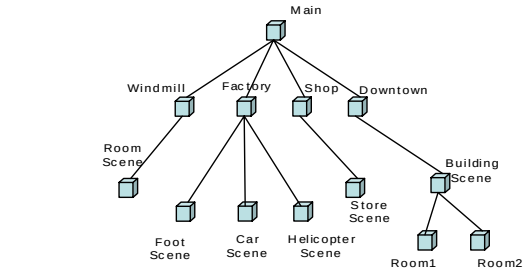
Fig. 3 The block diagram of a game system

Fig. 4 Different stories and events: the windmill (story1), factory (story2), shop (story3) and downtown (story4)

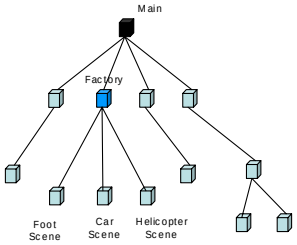
The hierarchical structure can represent story relations and topological properties in 3D space. In other words, the current scene is detected to be corresponded to a known story node.

By using the knowledge of the hierarchical structure one can predict the next event or the next path in the database. Furthermore, one can record complexity and coordinates of current event of 3D environment, when the event is enabled by a player. Based on the hierarchical structure it offers an insight into many stories as illustrated in Fig. 5(a). According to an avatar's position in 3D space, one can further predict which story in the future the avatar will be close to.

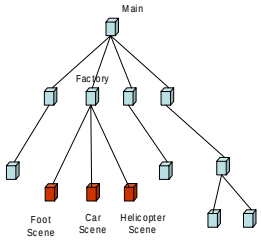
Fig. 5 shows windmill, factory, shop and downtown stories in the second level and room, foot, car, helicopter, store and building scene in the third level of a hierarchical structure. When the avatar walks in a certain space, one can check the current complexity (story) in the database as listed in Table 2 and predict next story. For example, If the distance in 3D space between the avatar position (A_P) and coordinates of factory story (C_S) is less than a threshold (ε) which is set to 1/400 of the scene, $|A_P(x,y,z) - C_S(x,y,z)| < \varepsilon$, one can predict that the avatar may enable the factory event in the next several frame/seconds from this hierarchical structure as illustrated in Fig. 5(b)-(c). The user may select a certain type of transportation in the factory story, e.g. foot, car and helicopter. Moreover, one can check whether the prediction is correct by checking the complexity in the next frame.



(a)



(b)



(c)

Fig. 5 Hierarchical structure of the benchmark: (a) hierarchical structure, (b) current event (black block) and predicted event (blue block) and (c) predicted scene

Table 2
Analysis of scene complexity from game

Current Complexity	Next Complexity	Coordinate (XYand Z)
0	3906	0.000000 0.000000 0.000000
3906	6830	0.000000 0.000000 0.000000
6830	8418	0.000000 0.000000 0.000000
8418	9122	0.000000 0.000000 0.000000
9122	10202	0.000000 0.000000 0.000000
10202	4886	7.350000 0.000000 0.000000
4886	0	7.4250000 0.000000 0.000000
4886	11267	2.800000 0.000000 -0.938000
11267	6400	4.550000 -0.050000 0.000000
6400	13487	4.000000 0.000000 6.000000
13487	3020	-6.460000 4.900000 0.000000
3020	7358	-2.250000 -0.350000 0.000000
7358	5864	-2.184000 1.250000 0.000000
5864	13487	-3.258000 4.250000 0.000000

In order to do power management one can define different power modules for different complexities, e.g. one can define 1~1500 triangles per second as module 1 and 1501~3000 as module 2. According to the difference of triangle counts between foot scene (triangle counts: 2400) and car scene (triangle counts: 3644), when the player chooses the type of transportation from foot to car one can know that the CPU cycle counts or power needed have better to increase from power module 2 to 3. According to the transportation selection a corresponding complexity can be known, and then one can select power needed for running the game via analyzing the complexity of the virtual scene as listed in Table 3.

Table 3
Analysis of scene complexity for power management

	Triangle Count	Power Module
Windmill Scene	3240	Module 3
Foot Scene	2400	Module 2
Car Scene	3644	Module 3
Helicopter Scene	5138	Module 4
Store Scene	913	Module 1
Room1 Scene	1362	Module 1
Room2 Scene	2364	Module 2

The different events and corresponding module are shown in Fig. 6, according to the event, one can find a corresponding module by searching the original complexity of stories in the database, and then select an appropriate voltage (or frequency) based on the hierarchical-based workload prediction algorithm for power management.

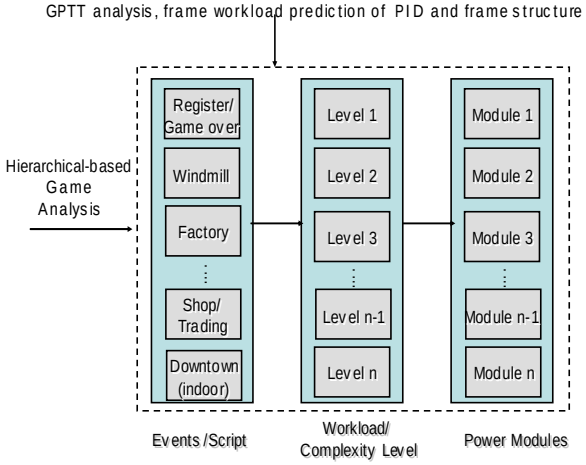


Fig. 6 Different event and corresponding module

B THE PREDICTION OF FRAME WORKLOAD

To do power management we need to predict the frame workload, if the error between predicted workload and actual workload is closer to zero, one will get a better power module and further adjust the voltage and frequency of FPGA/hardware or software. The different story often has the different complexity (triangle counts) when the avatar selects from the current story into another, one can predict the frame workload of the next story according to the hierarchical-based analysis in Subsection III.A. In fact, the frame workload in two adjacent frames F_i and F_{i+1} are usually utmost similar to each other because their original (pre-process) triangle counts $\#Tricount(F_i)$ and $\#Tricount(F_{i+1})$ are equal in the same scene. If the difference, ΔF, of triangle counts of two adjacent frames is equal to zero, the two frames are belong to the same scene, otherwise they have different scene. Hence, in order to do power management the scene change of the benchmark is needed to be detected. The definition of scene change is given as follows:

$$\Delta F = \#Tricount(F_{i+1}) - \#Tricount(F_i) \quad (1)$$

$$\begin{aligned} &If\ \Delta F = 0\ then\ the\ same\ scene \\ &else\ if\ \Delta F > 0 \\ &\quad then\ scene\ change\ and\ complexity\ increases \\ &\quad else\ scene\ change\ and\ complexity\ decreases \end{aligned} \quad (2)$$

Moreover getting the information of scene change, one can predict the variation of the next frame based on a simple analysis of scene animation, e.g. zoom-in and zoom-out. For example, in order to detect zoom-in, one can extract the variations of eye coordinates in frames F_i and F_{i+1} from the gluLookAt function i.e. gluLookAt function creates a viewing matrix derived from an eye point, a reference point indicating the center of the scene. If the camera behavior is zoom-in, z value coordinates of eye point, ΔL_Z, will decrease, the whole workload will increase. The variation of gluLookAt function are given as follows:

$$\Delta L_z = lookat_z(F_{i+1}) - lookat_z(F_i) \quad (3)$$

$$\begin{aligned} &If\ \Delta L_z = 0\ then\ the\ camera\ does't\ activate\ zoom\ function \\ &else\ if\ \Delta L_z < 0\ then\ zoom\text{-}in\ else\ zoom\text{-}out \end{aligned} \quad (4)$$

Moreover, we also use the CPU workload C_i in frame i and average CPU workload of previous several frames, $\overline{C_p}$, from frame (i-1-T) to frame i-1. T is set to be five frames. If the difference, ΔC, of CPU workload is smaller than zero, we know that CPU workload may decrease in frame i+1. The CPU workload trend of previous prediction is given as follows:

$$\Delta C = C_i - \overline{C_p} \quad (5)$$

$$where\ \overline{C_p} = \sum_{x=(i-1)-T}^{i-1} C_x / T \quad (6)$$

$$\begin{aligned} &If\ \Delta C = 0\ \ then\ \ no\ changes\ in\ the\ workload \\ &else\ if\ \Delta C > 0\ the\ workload\ increases \\ &\quad else\ the\ workload\ decreases \end{aligned} \quad (7)$$

By using the variations of ΔF, ΔLz and ΔC, one can predict the CPU workload trend. For example, when the adjacent frames are belong to the same scene and if the $\Delta C > 0$ and $\Delta Lz < 0$, one can know the whole workload will increase. The prediction of CPU workload trend is given as follows:

$$
\begin{aligned}
&If\ \Delta F = 0\ then \\
&\quad if(\Delta C = 0\ \&\ \Delta L_Z = 0)\ then\ no\ change\ in\ the\ workload \\
&\quad else\ if\ ((\Delta C \geq 0)|(\Delta L_Z \leq 0))\ then\ the\ workload\ increases \\
&\qquad else\ if\ ((\Delta C \leq 0)|(\Delta L_Z \geq 0))\ then\ workload\ decreases \\
&\qquad\quad else\ no\ change\ in\ the\ workload \\
&else\ if\ \Delta F > 0\ \ then\ \ the\ workload\ increases \\
&\quad else\ the\ workload\ decreases
\end{aligned}
\tag{8}
$$

After the trend of CPU workload is known, one needs to calculate the frame workload variations $\Delta\overline{fw_i}$ that is measured by the hybrid algorithm including PID and frame structure-based. PID predictor is a generic control loop feedback mechanism based on the feedback from recent prediction difference, PID includes three components (Proportional control, Integral control and Derivative control). The proportional control measures the variation $\varepsilon(t)$ of the system in predicted workload $\overline{w_i}$ and actual workload w_i, where by using GPTT one can receive the actual workload w_i. The integral control based on recent variation is used to measure the sum of the recent errors $\sum_{T_i}\varepsilon(t)$ from frame (i-T_I) to frame I and T_I is set to the frame interval. Finally, the derivative control measures the change of frame rate in the process $(\varepsilon(t) - \varepsilon(t - T_D))/T_D$, where T_D is set to be equal to the frame execution time. The frame workload variation of the PID predictor is given as follows:

$$\Delta PIDw_i = K_p \cdot \varepsilon(t) + \frac{1}{I} \cdot \sum_{T_i} \varepsilon(t) + D \cdot \frac{\varepsilon(t) - \varepsilon(t - T_D)}{T_D} \tag{9}$$

where K_p and I are the proportional and integral coefficients, and D is derivative coefficient. For experiment convenience one can obtain the best result by manually tuning these parameter values when K_p =0.9. I=28 and D=0.0033. In addition, we also need to calculate the predicted workload $\overline{FSw_i}$ using the frame structure-based predictor. Let ΔFSw_i denote the frame workload variation of the frame structure-based predictor, and its workload variation is given by $\Delta FSw_i = w_i - \overline{FSw_i}$, where we suggest that the predicted frame workload $\overline{FSw_i}$ which is almost linearly correlated with its rasterization workload, therefore one can predict the total frame workload via estimating the rasterization workload of a frame.

Based on the frame workload variations $\Delta PIDw_i$ and ΔFSw_i, the frame workload variation we select the minimum of them as the output $\Delta\overline{fw_i} = min(|\Delta PIDw_i|, |\Delta FSw_i|)$ and use the CPU workload trend from Eq.(8), one can do the frame workload prediction. If the whole workload will increase in frame i+1, the next estimated frame workload is given by $\overline{fw_{i+1}} = \overline{fw_i} + \Delta\overline{fw_i}$, otherwise the frame workload is given by $\overline{fw_{i+1}} = \overline{fw_i} - \Delta\overline{fw_i}$. The prediction of frame workload is given as follows:

$$\overline{fw_{i+1}} = \begin{cases} \overline{fw_i} + \Delta\overline{fw_i}, & if\ the\ workload\ \ increases \\ \overline{fw_i} - \Delta\overline{fw_i}, & if\ the\ workload\ \ decreases \\ \overline{fw_i}, & otherwise \end{cases} \tag{10}$$

IV. PERFORMANCE EVALUATION

The evaluation environment is built around a game alike benchmark as illustrated in Fig. 5(a). The experiments were run in the PC environment, PC with an AMD Athlon 1400+ and 512megabytes main memory. By using GPTT one can receive the real CPU cycle counts from the software implementation. Performance evaluation is conducted on the proposed algorithm, the traditional history-based method and the Gu et al's hybrid algorithm [6] .

As illustrated in Fig. 7(a), the proposed algorithm performs much better from frame 87 to 88. This is due to the proposed algorithm can predict the scene change in frame 88 according to the game analysis in advance as illustrated in Fig. 7(b), but the competing algorithms don't consider scene variation and may get more error. To give a better comparison, the CPU cycle counts for each algorithm are given in Fig. 8, and the proposed algorithm yields more than (8152-6872)/8152 * 100% = 15.7% improvement in cycle count estimation. Hence, it is a feasible way via game analysis to help us to predict the frame workload and match the diagrammatic curve of real CPU workload as illustrated in Fig. 9.

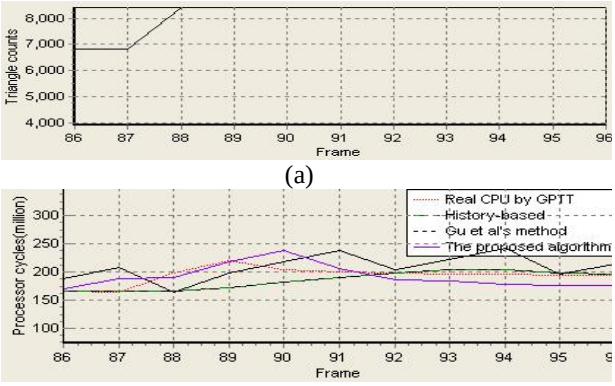


Fig. 7 Triangle counts and CPU cycle counts: (a) Triangle counts (b) Red small-dot line is real cycles by GPTT, green dash-dot line is the prediction by history-based predictor, black dotted line is the prediction by the Gu et al's algorithm, and purple line is the prediction by the proposed algorithm

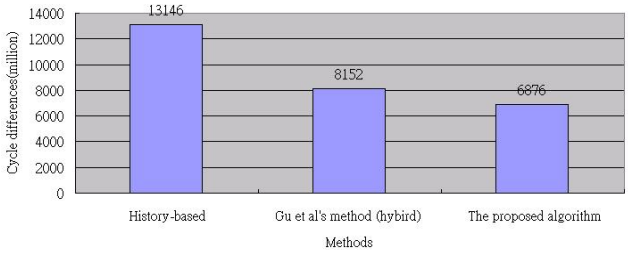


Fig. 8 Comparisons of cycle difference for different algorithms

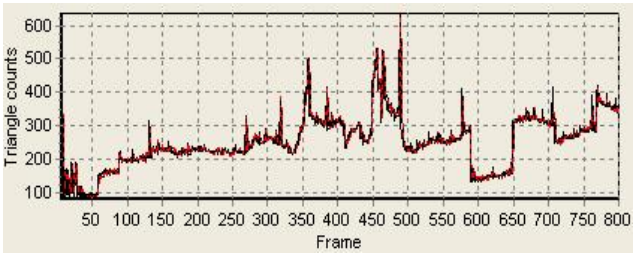


Fig. 9 Workload predictions: Black line is real CPU cycles and red dotted line is the prediction by the proposed algorithm

Currently, the proposed algorithm for game workload prediction only spends 171.2 microseconds to finish the work as illustrated in Fig. 10. Originally, each frame takes 0.0333333 seconds to finish. After using the hierarchical-based workload prediction algorithm, each frame will take 0.0335045 seconds, which is 0.51 % longer than the original period. Each frame takes 0.0344907 seconds to receive data after enabling GPTT (0.000903 seconds). In other words, there are 28.99 frames displayed within one second. The overhead is very insignificant. It almost doesn't affect the performance of CPU or graphics chip.

Moreover, the proposed algorithm provides an improvement of more than 1.028 frames and requires only 0.037 frame of extra workload as compared to the competing algorithms.

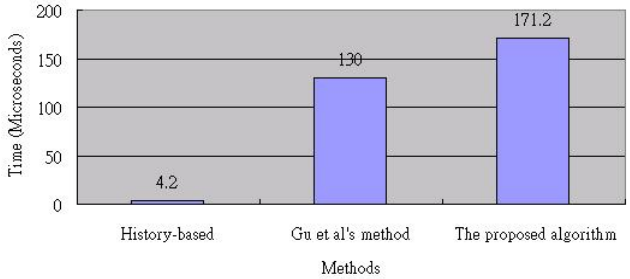


Fig. 10 Calculation workload for different algorithms

V. CONCLUSIONS

We have proposed a novel hierarchical-based workload prediction algorithm for power management. Based on measurement of GPTT, the proposed algorithm for game workload prediction spends only 41.2 ms of more than the competing algorithm to finish the work, but achieves an improvement of more than 15.7 % in cycle count estimation.

ACKNOWLEDGMENT

The authors would like to thank the National Science Council of Taiwan, for financially supporting this research under contract no. NSC96-2220-E-110-003.

REFERENCES

[1] C. J. Hughes, and S. V. Adve, "A Formal Approach to Frequent Energy Adaptations for Multimedia Applications," *In International Symposium on Computer Architecture (ISCA)*, pp. 138–149, Munich, Germany, 2004.

[2] C. Im, H. Kim, and S. Ha, "Dynamic Voltage Scheduling with Buffers in Low-Power Multimedia Applications," *ACM Transactions in Embedded Computing Systems*, pp. 686–705, 2004.

[3] Y. Gu, and S. Chakraborty, "Control Theory-Based DVS for Interactive 3D Games," *Design Automation Conference (DAC)*, Anaheim, CA, USA, 8-13 June, 2008.

[4] Y. Gu, and S. Chakraborty, "Power Management of Interactive 3D Games Using Frame Structures," *21st International Conference on VLSI Design (VLSID)*, Hyderabad, India, January 2008.

[5] B. Mochocki, K. Lahiri, S. Cadambi, and X. S. Hu, "Signature-Based Workload Estimation for Mobile 3D Graphics," in Proc. *Design Automation Conference (DAC)*, pp. 592-597, July 2006.

[6] Y. Gu, and S. Chakraborty, "A Hybrid DVS Scheme for Interactive 3D Games," *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 3-12, USA, April 2008.

[7] N. Bansal, K. Lahiri, A. Raghunathan, and S. T. Chakradhar, "Power Monitors: A Framework for System-Level Power Estimation Using Heterogeneous Power Models," in Proc. *The 18th international Conference on VLSI Design Held Jointly with 4th international Conference on Embedded Systems Design*, pp. 579-585, 2005.

[8] B. G. Nam, J. Lee, K. Kim, S. J. Lee, and H. J. Yoo, "A 52.4mW 3D Graphics Processor with 141Mvertices/s Vertex Shader and 3 Power Domains of Dynamic Voltage and Frequency Scaling," *In Digest of the 2007 IEEE International Solid-State Circuits Conference (ISSCC'07)*, 2007.

[9] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno, "Efficient Power Co-Estimation Techniques for System-on-Chip Design," *in Proceedings of Design, Automation and Test in Europe (DATE '00)*, pp. 27–34, Paris, France, March 2000.

[10] NVIDIA Corporation, "NVperfHUD," http://developer.nvidia.com/object/nvperfhud_home.htm

[11] graphicREMEDYCorporation,"gDEBugger," http://www.gremedy.com.

[12] Hawk Software, "GLTrace Programming Utility,"

http://www.hawksoft.com/gltrace/
[13] N. Duca, K. Niski, J. Bilodeau, M. Bolitho, Y. Chen, and J. Cohen, “A Relational Debugging Engine for the Graphics Pipeline,” *Proc. of the ACM Transactions on Graphics (TOG)*, Vol. 24, Issue 3, pp. 453-463, 2005.
[14] SPEC,“SPECvireperf 8.1,” http://www.spec.org/gpc/opc.static/vp81information.htm
[15] Futuremark Corporation, “3D Mark series benchmarks,” http://www.futuremark.com/products/
[16] Silicon Graphics, “OpenGL,”http://www.opengl.org/
[17] Futuremark Corporation, “SPMark04,” http://www.futuremark.com/products/spmark04/
[18] Futuremark Corporation, “3DMark Mobile06,” http://www.futuremark.com/products/3dmarkmobile06/
[19] C. N. Lee, D. J. Zhang-Jian and K. Y. Lin, “A New FPGA-Based Cross-Platform Graphics Performance Tuning Tool for ARM Versatile Platform,” *Computer Graphics Workshop (CGW)*, Taiwan, 2007.
[20] Khronos Group, “OpenGL ES Specification,” http://www.khronos.org/opengles
[21] L. B.i Chen, T. Y. Ho, I. J. Huang, Y. N. Chang, S. W. Haga, J. H. Hong, S. F. Hsaio, S. R. Kuang, K. C. Kuo and C. N. Lee, "The Development of an Energy-Awared Mobile 3D Graphics SoC with Real-Time Performance/Energy Monitoring and Control," *IEEE International SoC Design Conference (IEEE ISOCC'08)*, Busan, Korea, Vol. I, pp.234-237, Nov. 2008.