MEMORY CHARACTERIZATION TO ANALYZE AND PREDICT MULTIMEDIA PERFORMANCE AND POWER IN EMBEDDED SYSTEMS

Yu Bai, Priya Vaidya
Marvell Semiconductor, Inc., {yu.bai, priya.n.vaidya}@marvell.com

ABSTRACT

Main memory performance and power are becoming increasingly critical for embedded and general purpose computing systems. That is the primary motivation for us to focus on analyzing and modeling memory utilization for multimedia applications in embedded systems like cell-phones and PDAs with limited hardware resources and battery supply. In addition, the paper presents scenarios of utilizing the memory characteristic model to predict multimedia application performance, optimize embedded software, and adapt hardware resources to save system power while guaranteeing adequate performance. We present a simple, but effective indicator to identify the application's dynamic memory and computational composition. Using this, we can easily and accurately forecast application performance and help determine optimal resources needed by the multimedia applications.

Index/Keywords – Memory Characterization and Modeling, Performance Prediction and Power Management, Embedded Systems, Multimedia Applications

1. INTRODUCTION

To satisfy rapidly increasing performance and power requirements from the cellular and handheld markets, architectural designs of cellular phones and PDAs are becoming more and more complicated with increasing clock frequency, advanced techniques, additional hardware resources, denser and highly integrated circuits, and etc. With those trends, especially rapidly increasing clock frequency, the memory access is extremely costly in terms of the CPU core clock cycles and thus the memory system turns into the main bottleneck to system performance. This is mainly because the speed gap between the fast CPU core and the relatively slow memory widens. Memory is also expensive from the die size perspective. Choosing the appropriate memory configuration to satisfy the performance requirements within the die size budget is a challenging design decision. Moreover, low power is another important design consideration besides high performance, especially for those battery-supplied devices. Lower power means longer battery life time and user-experience time. Tremendous power management techniques have been investigated [3, 5, 7]. Memory is usually a significant power contributor in such embedded systems for popular use cases. In addition, its power dissipation tends to increase quickly with its increasingly complicated design and large memory sizes and hierarchies to keep up the system's fast CPU development. Figure 1 shows overall power distribution in a general cellular phone system[1] for a 3D gaming application, in which memory sub-system is responsible for 27% of the total system power dissipation. For more memory intensive applications, such as high-end video encoding, the memory could consume even more power. Therefore, reducing memory power could introduce significant battery life benefits.

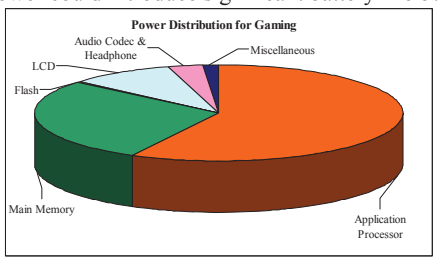

Figure 1: Power distribution for a 3D gaming application

The rest of the paper is organized as follows. Related work is summarized in Section 2. Section 3 introduces memory characterization and identification in details. Section 4 presents our experimental results, key findings, our proposal, and contributions. Section 5 offers conclusions and promising future work.

2. RELATED WORK AND MOTIVATION

To better understand various applications' intrinsic behaviors, it is essential to investigate memory characteristics and build memory model to determine if the application involves intensive memory accesses and even help predict the application's performance under different system configurations. An elaborate framework of computational and memory characterization mechanism was introduced in the paper [1]. Based on the memory operations, number of instructions retired etc., the characterization model decomposes the number of cycles consumed in a window into computation and memory cycles. Using this model, it is straightforward to identify if the application is more computational bound or more memory bound. Although the performance analysis and characterization methodology in [1] is accurate to predict performance as proved in the paper, it is quite complicated to implement. Quite a few Performance Monitoring Unit (PMU) events need to be probed to guarantee the model's prediction accuracy and this might require multiple times of PMU probing, since only limited number of PMU events can be monitored at the same time. This potential issue can be solved by probing different PMU events at different times, but such a method definitely introduces non-trivial overhead and inaccuracy. In other words, it is very expensive to fully incorporate this methodology in reality. Thus the purpose of this paper is to look for a simple and cheap way to dynamically characterize the application's computational and memory composition with the acceptable accuracy.

To take advantage of dynamic voltage and frequency management, adaptive power management methods are emerging as the key to performance and power scaling in real-time embedded systems. One of such adaptive power management technique was proposed for XScale™ Microarchitecture based platforms, which

[1] The system is built on Marvell's current generation XScale™ application processor running Linux-based operating system with the QVGA LCD display and 256MB low power DDR.

dynamically characterizes executing workloads, based on system level events and adapts frequency and voltage so as to save power dissipation [7]. Figure 2 shows the overall framework for this embedded software based power management framework. It is composed of two basic components: performance/idle profiler and policy manager. The performance/idle profiler is responsible for probing the system, collecting the CPU utilization and memory statistics from PMU and operating system, and making these information available to the policy manager in each window The policy manager uses these inputs and optimally chooses suitable system operating point and even different power modes to save power while still satisfying the application's dynamic needs in the next window. We seek to improve the power scaling solution by having a better and simpler memory analytical model to mend the performance prediction's capability and accuracy.

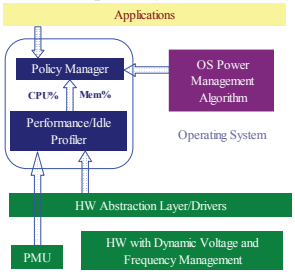

Figure 2: Architecture of the framework [7]

Some other memory work research has been studied from others perspectives. The paper [2] proposed a simple memory controller policy to manage memory power by entering the low power state whenever the memory is identified to be idle by the memory controller. The patent [6] managed the memory performance based on feedbacks collected from PMU. Our work is completely different from previous work by dynamically monitoring the system performance, estimating the memory loads, predicting performance for the near future, and optimizing the power.

3. MEMORY CHARACTERIZATION

This paper is intended to investigate how the application's performance is varied with memory related hardware performance monitors and identify which performance monitor or combination is effective and accurate to indicate the application's memory and computational composition. This investigation is useful to predict future CPU utilization based on memory characterization analysis and can therefore be used to improve policy manager in the power management framework.

3.1 Memory/Computationally Bound Applications

Apparently, without memory involved, the CPU utilization should scale linearly with the CPU core frequency. For example, if the CPU utilization is 50% at 104MHz with little or even no memory access, we can easily predict that the CPU utilization would be around 25% at 208MHz with the high confidence. But with memory access involvement, the CPU utilization would not scale linearly with the core frequency any more. At the higher frequency point, performance is usually more blocked by the memory since the CPU spends more CPU cycles in waiting for memory response. Here we assume that the memory frequency does not change at different CPU frequencies. In this sense, applications can be characterized by memory and computational behaviors into two types: computational bound and memory bound applications. For the computational bound applications, it is trivial to predict CPU utilization since it is closely proportional to the CPU frequency. For memory bound applications, it is much harder to predict. We need to analyze the memory and computational composition and then determine how skew the CPU utilization is with the CPU frequency. Typically, the multiplication of the CPU utilization and CPU frequency is defined as the metric of quantifying the application's cost in the CPU, memory, and other hardware resources. In this paper, we name it "Application Cost". If a workload involves with only computational operations with few or even no memory access, its application cost is approximately a constant regardless of the CPU frequencies, as illustrated in Figure 3 (the blue curve). Differently, if a workload accesses the memory heavily, its application cost changes with the CPU frequency and trends higher at the higher CPU frequency as shown in Figure 3 (the red curve). Obviously, if only considering the application cost metric, the lower CPU frequency is typically more effective as long as it can provide enough hardware resources and memory bandwidth for the workload. However, if considering more factors, like power dissipation, the conclusion might be different.

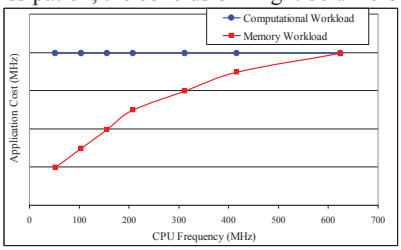

Figure 3: Typical application cost curves of computational and memory workloads

3.2 Hardware Performance Events

We focus on studying three different approaches to characterize memory composition and help determine the application's CPU utilization under different system configurations. Those hardware performance events are collected by probing PMU. Thus our methodology only applies to the system with the PMU hardware support. Event and indicator details are described as follows:

1. Overall Data Cache Miss Rate Intuitively, higher data cache miss rates imply heavier memory traffic. To obtain data cache miss rate, we need to monitor the total number of accesses and misses in the first level data cache and second level data cache if presented. So, two or four PMU event probes are needed depending on the availability of the second level cache.

2. Main Memory Access Rate Occupancy rate of external memory controller is a direct indicator of memory utilization. To get main memory access rate, two PMU events need to be collected: the total number of cycles that memory controller is occupied and the total number of cycles in the monitoring window.

3. Data Stall Rate Pipeline stalls are mainly due to data dependencies while the reason of data unavailability is caused by far slower memory accesses compared to the CPU speed. So the number of pipeline stall occurrence also reflects memory traffic situation. Moreover, this indicator implies memory access's criticality. Not every memory access is critical to the ultimate performance, thus it is quite useful to keep track of memory accesses that introduce data dependencies which impact performance. In this approach, the event occurrence is monitored

in which the pipeline is stalled due to the data dependency. In addition, the total number of cycles needs to be recorded for data stall rate calculation in each window.

Different approaches reflect memory characterization from different perspectives. We might use single approach or combinations for more effective performance analyses and more accurate prediction with reasonable overhead.

4. EXPERIMENTAL RESULTS AND ANALYSIS

4.1 Experimental Setup

We use Marvell's current generation XScale™ application processor[2] running Linux-based operating system with the QVGA LCD display as our test platform. This processor is a super-pipelined and RISC microarchitecture with high-performance and low power dissipation. It can provide a rich feature set optimized for cellular telephone handset and personal digital assistant applications. In addition, it includes two level cache structures to enhance system performance, in which the second level cache can be turned on and off by software for analytical purposes. Given the importance of the standard algorithms in the multimedia community, we choose to focus on popular standard audio (MP3/AAC+) and video (H.264) decoders in this study. Video streams cover two different resolutions (QVGA/QCIF). In all experiments, audio/video applications are downloaded as executables in the system and audio/video streams are running out of the file system.

4.2 Experimental Results and Prediction of CPU Utilization

Figure 4 gives the comparisons of three approaches described in Section 3.2 for MP3 decoder, AAC+ decoder, and H.264 decoder. Each graph includes two curves, one with the second level cache enabled and one disabled, and three CPU frequencies, 208MHz, 416MHz, and 624MHz. The "Load" is the application cost as the multiplication of CPU utilization and CPU frequency.

With no heavy memory accesses, the CPU utilization should be approximately linear to the core frequency and thus application cost/load curve should be flat with the core frequency. MP3 and AAC+ decoders are such good examples when L2 cache is on. This is because MP3 and AAC+ decoders only introduce a small amount of memory accesses and most of them are caught by L2 cache. Clearly, L2 cache can efficiently reduce memory accesses. In the cases of no L2 cache, application loads go up with the core frequency as discussed earlier. We also notice that cache miss rates do not change significantly with the CPU frequency for either L2 cache on case or L2 cache off case, which implies that overall data cache miss rate is an ineffective metric to indicate memory access situation and therefore predict the CPU utilization.

Intuitively, the memory access rate, DDR%, contains the similar amount of system information as the cache miss rate because cache misses directly introduce memory accesses. However, our experimental results indicate that this is not true. For example, H.264 QCIF decoder shows similar cache miss rate trend as H.264 QVGA decoder, but H.264 QVGA decoder spends much larger percentage of the application time in the memory accesses than H.264 QCIF decoder. This again proves that solely monitoring cache miss rate is not effective to indicate memory composition and therefore predict the CPU utilization for power management purposes. In fact, cache miss rates do not necessarily introduce performance drop if the total amount of cache accesses is trivial. MP3 decoder without L2 cache is such a good example. On the other hand, typical cache miss rate may introduce a great amount of memory accesses. H.264 QVGA decoder shows this trend.

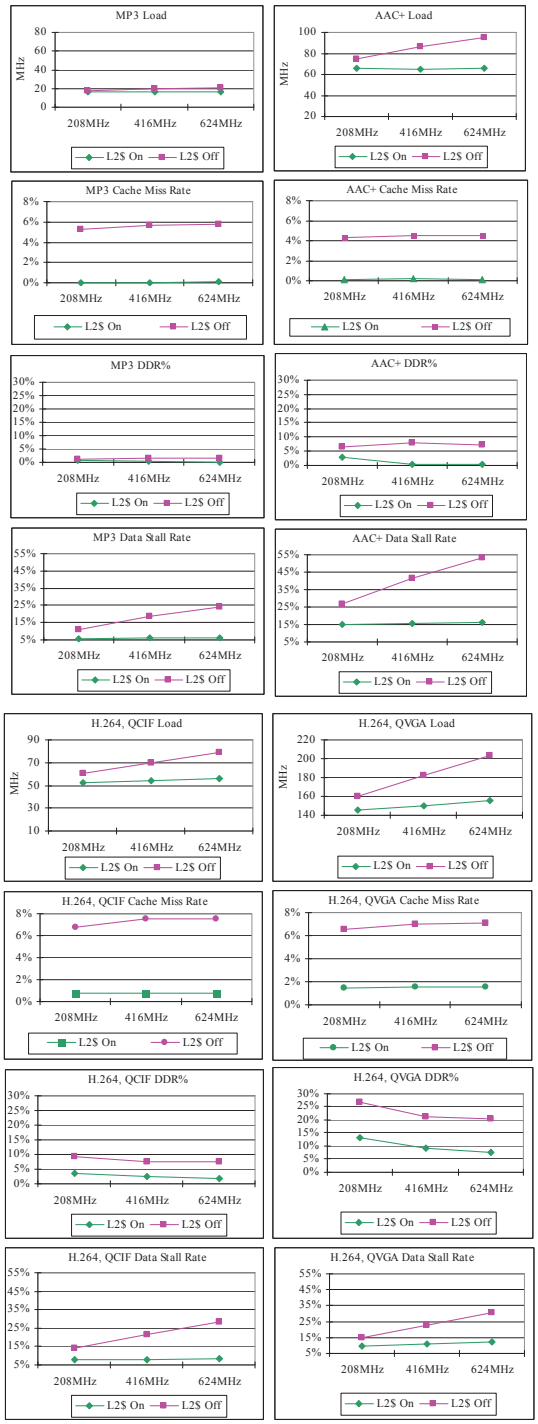

Figure 4: Three approaches of memory characterization

[2] There is no hardware video accelerator.

As discussed earlier, not all memory accesses contribute equally to the system performance. Some may be on the performance critical path, some may not be. So it is helpful to study the memory access criticality more carefully and distinguish them so as to better predict system performance. However, neither cache miss rate, the total amount of cache accesses, nor main memory access rate can differentiate if the memory accesses are on the critical path or not. Fortunately, our results show that data stall rate is a good indicator of the memory access criticality. Apparently, data stall rate curves are synchronous with application cost/load curves for all applications except MP3 decoder. Data stall rate is the best metric so far to predict application loads from our experiments. For MP3 decoder, memory access rate is extremely low. Therefore, even if there are some critical memory accesses among overall very few memory accesses, performance impact is still trivial.

In summary, the CPU utilization can be predicted to be proportional to the CPU frequency if overall memory access rate is negligible. Data stall rate should be used together for predicting the CPU utilization when the memory access rate becomes non-trivial. With more memory accesses, data stalls lead to larger application cost increases as the CPU frequency increases. The relation among memory access, data stalls, the CPU frequency, and the CPU utilization is dependent on the application and could be approximated by experiments. Figure 5 presents an algorithm example using the linear function to predict the CPU utilization based on memory characterization. In this figure, F is the CPU frequency for now and F_i is the CPU frequency that the system chooses to switch to in the future. CPU% is the CPU utilization measured for now and $CPU\%_i$ is the CPU utilization that needs to be predicted if the system runs at the frequency F_i in the future. DDR% denotes the main memory access rate for now. Stall% is the data stall rate for now. T_1, T_2, and T_3 are three thresholds depending on the applications and need to be defined and adjusted by experiments. Additionally, we can implement dynamically changing thresholds by feedbacks from the system. This algorithm first checks if the memory access rate is lower than the predefined threshold T_1. If it holds, we predict that the CPU utilization is linear to the CPU frequency, as shown in the yellow block in Figure 5; otherwise, the CPU utilization is predicted in two steps: (1) being proportional to the frequencies; (2) adjusting according to the data stall rate. In the second step, we introduce two more thresholds: T_2 and T_3, which are shown in the cyan block. To implement this algorithm, we need to keep track of both main memory access rate and data stall rate. Therefore, at most three PMU events need to be monitored: (i) the total number of cycles that external memory controller is occupied; (ii) the total number of occurrences that the pipeline is stalled due to data dependency; and (iii) the total number of cycles during the monitoring window. (i)/(iii) gives DDR% and (ii)/(iii) gives Stall%. Clearly, this algorithm is much easier to incorporate into the power management framework [7] than the one proposed in [1]. Additionally, we can also include a simple closed loop feedback system to dynamically adjust three thresholds to better predict the CPU utilization based on the difference between the predicted CPU utilization and actually measured CPU utilization from the system.

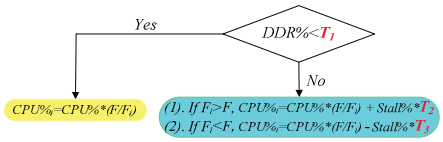

Figure 5: A simple algorithm of performance prediction

Our simulation results show that MP3 decoder does not benefit from this new algorithm at all while AAC+ decoder gets most benefit by increasing its battery life by 27%. H.264 QCIF and QVGA decoders are in the middle. Their battery life is increased by 15% and 7%, respectively. Because MP3 playback has quite few memory accesses, its performance prediction is straightforward and so the additional memory analysis is not beneficial. In this experiment, performance degradation due to the power saving technique is within the acceptable range.

5. CONCLUSION

This paper proposes a simple algorithm to improve the performance prediction by studying three memory indicators to help characterize memory composition. Our experiments show good battery life gains from incorporating our newly proposed algorithm into a power management framework. The approach proposed in this paper can be applied in any system with the PMU hardware support. In the future, many promising work could be investigated. For example, it is interesting to experiment with more multimedia and non-multimedia applications and optimize our algorithm by designing dynamically adaptive thresholds based on user inputs and/or more system feedbacks. Future work may also include incorporating the memory characterization and performance prediction presented in this paper to other software based power managements and memory performance/power managements, such as adaptive sampling window scheme and moving average modeling in [4].

REFERENCES

[1] Jia Bao, Bin Xiao, Priya Vaidya, and Yu Bai, "Memory Modeling Approach to Multimedia Applications Performance Analysis and Characterization", *Global Signal Processing Conference & Expos for the Industry*, *4th International Signal Processing Conference*, October 2006

[2] Xiaobo Fan, Carla S. Ellis, and Alvin R. Lebeck, "Memory Controller Policies for DRAM Power Management", *Proceedings of the International Symposium on Low Power Electronics and Design)*, August 2001

[3] Krisztián Flautner, David Flynn, and Mark Rives, "A Combined Hardware-Software Approach for Low-Power SoCs: Applying Adaptive Voltage Scaling and Intelligent Energy Management Software", DesignCon 2003, *System-on-Chip and ASIC Design Conference*

[4] Moinul H. Khan, Yu Bai, Bin Xiao, and Priya N. Vaidya, "Time Series Modeling Based Power and Performance Scaling Framework", *International Conference on Acoustics, Speech, and Signal Processing*, April 2007

[5] Youngsoo Shin, Kiyoung Choi, and Takayasu Sakurai, "Power Optimization of Real-Time Embedded Systems on Variable Speed Processors", *Proceedings of the International Conference on Computer-Aided Design*, November 2000

[6] Sreevinas Subramoney, Richard Hudson, Mauricio Serrano, and Ali-Reza Adl-Tabatabai, "Dynamic Performance Monitoring-Based Approach to Memory Management", *United States Patent*

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