IEICE Electronics Express, Vol.*, No.*, 1–7

A Powersaving DVFS algorithm based on Operational Intensity for embedded systems

Seong Jin Cho, Seung Hyun Yun, and Jae Wook Jeon[1a)]
[1] *The College of Information and Communication Engineering Sungkyunkwan University, Suwon, 440746, Korea*

a) *jwjeon@yurim.skku.ac.kr*

Abstract: Energy use is currently an important issue in mobile embedded systems, and this will be continue in the future because of battery problems. We focused on a Dynamic Voltage Frequency Scaling (DVFS) algorithm to save as much energy as possible, and investigated the correlation of operations between memory and CPU. We introduce an Operational Intensity (OI)-based DVFS algorithm to follow the optimal frequency to conserve energy in embedded systems. Our proposed method was verified using FPMark of EEMBC, and conserved energy up to 17% in a memory-bound application.

Keywords: energy, DVFS, operational intensity, embedded systems
Classification: Electron devices, circuits, and systems

References

[1] M. Gries: The VLSI J. **38** (2004) 131.
[2] M.A. Awan, and S.M. Petters: ECRTS (2011) 92.
[3] R. Jejurikar, and R. Gupta: ISLPED (2004) 78.
[4] Y. Liang, P. Lai, and C. Chiou: Journal of Convergence **1** (2010) 93.
[5] FPMark: The Embedded Industry's First Standardized Floating-Point Benchmark Suite http://www.eembc.org/benchmark/fp_sl.php
[6] EEMBC: Embedded Microprocessor Benchmark Consortium http://www.eembc.org/
[7] S.W. Williams, A. Waterman and D.A. Patterson: Communications of the ACM **52** (2009) 65.
[8] STREAM: Sustainable Memory Bandwidth in High Performance Computers. http://www.cs.virginia.edu/stream/

1 Introduction

The latest embedded systems, such as in smart phones, and tablet PCs, with performance comparable to that of desktop PCs can now serve consumers who want to operate highly complex applications. Many mobile companies have now released recent smart phones with state-of-the-art applications processors (APs), consisting of a multi-core, high-performance GPU, and many

DOI: 10.1587/elex.12.20141128
Received December 1, 2014
Accepted January 5, 2015
Publicized January 16, 2015

dedicated IPs. Usually, an embedded system has millions of design possibilities [1]. However, the growth of technology regarding batteries has been relatively very low with regard to the growth of system performance. Thus, energy consumption by mobile systems is a very hot issue both in terms of hardware and software, and many related technologies exist to address this, including a DVFS.

We focused on a DVFS algorithm for reducing energy consumption by the system, because the powersave governor that apply to the lowest frequency in linux does not save the maximum amount of energy. The main reason for this problems is that many resources are operated for longer times due to the lower frequency. To address this problem, a race-to-halt strategy that runs as fast as possible is the best way to save energy [2]. However, it is not applicable in all situations. Even if a system is set to the highest frequency for finishing work, the memory is still operated at a slow speed in memory-bound applications that includes more memory operations than CPU operations because of the slower memory speed versus CPU speed. Thus, it is important to find an optimal frequency for conserving energy in the system. In efforts to conserve energy, considering memory, recent studies [3][4] have introduced various approaches. However, these studies still have limitations. Jejurikar et al. [3] presented a task slowdown algorithm for tasks that affect the leakage current of the processor. It has the ability to calculate a slowdown factor of tasks in a real-time OS using a simulation program. Liant et al. [4] proposed an AD-DVFS based on Memory Access Rate (MAR) that calculates memory activity using a cache miss parameter on the Performance Monitoring Unit (PMU). However, neither method considers state-of-the-art embedded system architectures, including a read-write-allocate cache or a hardware prefetch unit.

In this letter, we introduce a new DVFS algorithm based on Operational Intensity (OI) for embedded systems and our experiment explains the relationship between OI and energy consumption. Additionally, we verified the performance of our proposed method using the FPMark [5] benchmark program of the EEMBC consortium [6].

2 OI-based DVFS algorithm

2.1 Conventional method

Liang et al. [4] exploited the MAR method to conserve power consumption on the Intel PXA270. The MAR is calculated between the number of cache misses and the number of total instructions, and each counter can obtain the numbers based on the PMU during run-time. Thus, the MAR, to assess memory activity, is determined by:

$$MAR = \frac{N_{cache\_miss}}{N_{instruction\_exec.}} \tag{1}$$

However, this MAR formula does not apply to state-of-the-art embedded system architectures for the following reasons. Recent embedded system

architectures consist of multi-level memory systems to improve memory operation speed, which is slower than CPU operation speeds. Additionally, well-known techniques, such as a hardware prefetch unit or read-write-allocate, which reduce the effect of memory latency, have been used due to increase overall system speed.

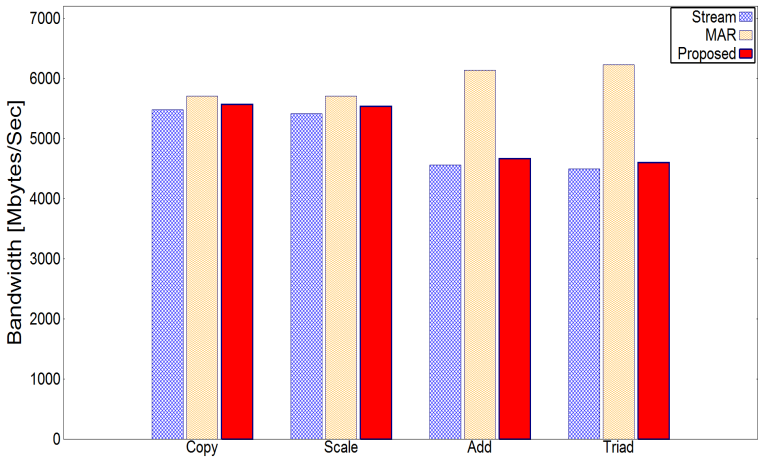

Fig. 1. The evaluation of memory bandwidth

In such cases, the parameter of cache misses causes errors and does not sufficiently represent memory activity shown in Fig. 1. Additionally, the DVFS algorithm that uses a MAR calculates a high-order equation because the energy of the system has non-linear characteristics. Thus, this method is not suitable for embedded systems.

2.2 OI-based DVFS algorithm

We propose a new power-saving strategy using the *operational intensity* of an application. The meaning of 'operational intensity,' proposed Williams et al. [7], is that when a kernel is executed on the system, it can be characterized by operations per byte access. Also, the bandwidth of the kernel that accesses DRAM, represents the peak performance per OI. Hence, the precise memory bandwidth must be calculated to evaluate OI precisely.

To evaluate memory bandwidth, we investigated all the PMU registers supported by an embedded system and finally found a bus parameter appropriate for that. Thus, the actual memory bandwidth and OI are determined by:

$$Memory\ Bandwidth = \frac{BUS\ access \times Byte}{Second} \tag{2}$$

$$Operational\ Intensity = \frac{The\ number\ of\ Operations}{BUS\ access \times Byte} \tag{3}$$

where *Bus access* represents the memory cycle that includes the activity of read and write operations between the last level cache and memory. That is, it can capture memory activity generated by a hardware prefetch unit or

read-write-allocate, whereas MAR does not. Thus, MAR causes measuring error, because it ignores a hardware techniques that reduce memory latency because it only considering memory activity based on cache misses. On the other hand, the OI has the ability to measure precise memory traffic because it only counts bus activity that was made by memory.

Fig. 1 shows that the observed memory bandwidth shows similar results to the stream benchmark [8] results. Stream benchmark program is widely used in various papers for measuring real peak memory bandwidth between the CPU and DRAM. It consists of four small programs – copy, scale, add, and triad – that constitute most memory operations, instead of computing operations. Thus, the results of the stream benchmark program may represent the attainable peak memory bandwidth compared with real-world applications that have a mix of memory and computing operations. To verify the relationship between OI and energy, we experimented with every OI using a micro-benchmark program, as shown in Fig. 2.

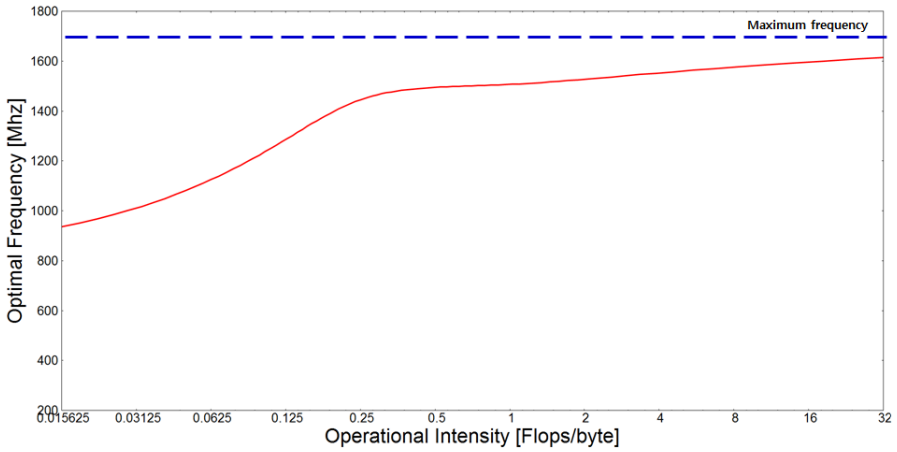

Fig. 2. The relationship between OI and Energy (ARM Cortex-A15)

The results show that higher OI, CPU-bound applications have a high optimal frequency, and lower OI, memory-bound applications have a low optimal frequency. The reason for the difference in optimal frequency is the amount of memory activity compared with total operations. That is, a higher OI that rarely includes a memory operation should finish work quickly, saving energy and a lower OI that includes many memory operations has an advantage, supposing the operation frequency is low. In the lower OI case, the lowest frequency supported by the system is not the optimal frequency. In fact, the lowest frequency consumes low power. However, more energy will be consumed because the total execution time of the application is extended.

To verify its real-world application, we experimented with an embedded system using the FPMark of EEMBC. The FPMark represents the floating-point performance of the embedded system and evaluates both single-precision and double-precision floating-point performance. Additionally, many workloads including FFT, Linear algebra, Ray Tracer, and others are con-

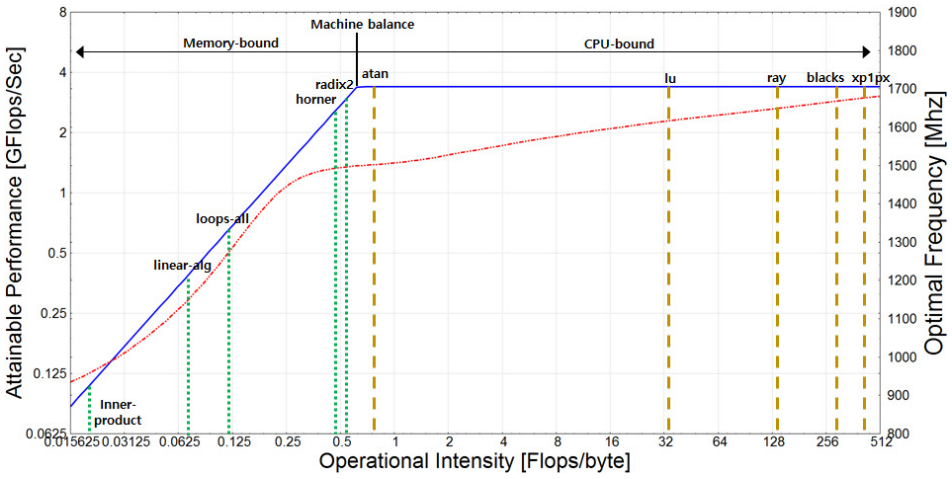

Fig. 3. The optimal frequency of FPMark on the roofline model

tained. The OIs of all workloads in the FPMark are shown in Fig. 3.

Fig. 3 shows the roofline model that provides insightful performance for applications on the system. The slope line of the roofline model corresponds to the memory-bound and the top line of the model corresponds to the CPU-bound. That is, the left side of the point that crosses over the slope-line and peak-line, called the machine balance, represents memory-bound and the right side represents CPU-bound. For example, inner-product, linear algebra, horner, and radix sort are memory-bound application and arctan, matrix manipulation, and ray tracer are CPU-bound applications. The roofline model is created only once per system, not once per kernel. Additionally, each workload has an optimal frequency following its own OI.

We implemented a new DVFS algorithm that had a 200-ms interval based on OI and followed the optimal frequency using calculated data.

3 Evaluation

3.1 Experiment environment

Experiments were performed on an embedded system with a 1.7GHz ARM Cortex-A15 dual core processor and 2GB of physical memory with Android OS 4.1. In Algorithm 1, we describe the pseudo-code of an OI-based DVFS algorithm. This algorithm is called periodically by a timer function that is registered in the work queue, and then it will set the target frequency, based on the calculated OI. To measure the numbers of the hardware counters, we made a kernel device driver for the performance-monitoring unit (PMU), which can best monitor the overall exact status of the system. Also, the energy of system was evaluated with the ARM Energy Probe. The experiment was executed with workload 10 times. All results are averages of the measured data. Also, we uninstalled applications and drivers that were not used to reduce experimental system variables.

Algorithm 1 The OI-based DVFS algorithm

```
1: procedure OI-DVFS(c: the number of core)
2:     disable_PMU_counting();
3:     VFP ← VFP − old_vfp;
4:     Bus_access ← Bus_access − old_bus;
5:     OI ← VFP / (Bus_access ∗ Byte);
6:     Optimal_Frequency ← lookup_table(OI);
7:     if Optimal_Frequency ≥ maximum_frequency then
8:         Optimal_Frequency ← maximum_frequency;
9:     else if Optimal_Frequency ≤ minimum_frequency then
10:        Optimal_Frequency ← minimum_frequency;
11:    end if
12:    err ← cpu_target_frequency(Optimal_Frequency);
13:    old_bus ← Bus_access;
14:    old_vfp ← VFP;
15:    enable_PMU_counting();
16:    start_timer(200ms);
17: end procedure
```

3.2 Results and Discussion

Fig. 4(a) and FIg. 4(b) show the results of energy consumption and performance in the FPMark 10 benchmarks between each governor, including performance, ondemand, powersave, and the proposed OI-based DVFS. The graph represents normalized results, based on the performance governor.

The ondemand governor that is used frequently can conserve about 1% in performance. Also, the powersave result indicates that the lowest frequency does not conserve energy over the all of the benchmarks. Our proposed OI-based DVFS can conserve, on average, energy of 9% versus performance, and 8% versus ondemand. In particular, where the application has a lower OI, inner-product of memory-bound, the reduction reaches a maximum of about 17% in energy, whereas with an application that has a higher OI, xp1px of CPU-bound, there is only a 3% reduction in energy compared to performance governor. Thus, our OI-based DVFS has the advantage for memory-bound applications.

In the results, both performance and ondemand have similar results and powersave shows a decrease of 10times versus performance due to the low frequency. The OI-based DVFS also decreases energy by about 12% versus performance, because we focused only on energy consumption.

4 Conclusions

We propose an OI-based DVFS algorithm for ARM embedded systems. It controls the optimal frequency based on information that includes memory activity of the applications. To implement the proposed algorithm, we used bus information, based on PMU, instead of cache-related monitoring registers. Unfortunately, bus information is only supported by the latest ARM cores. We explained the relationship between OI and energy, and implementation of the algorithm on the basis of the information based on the OI.

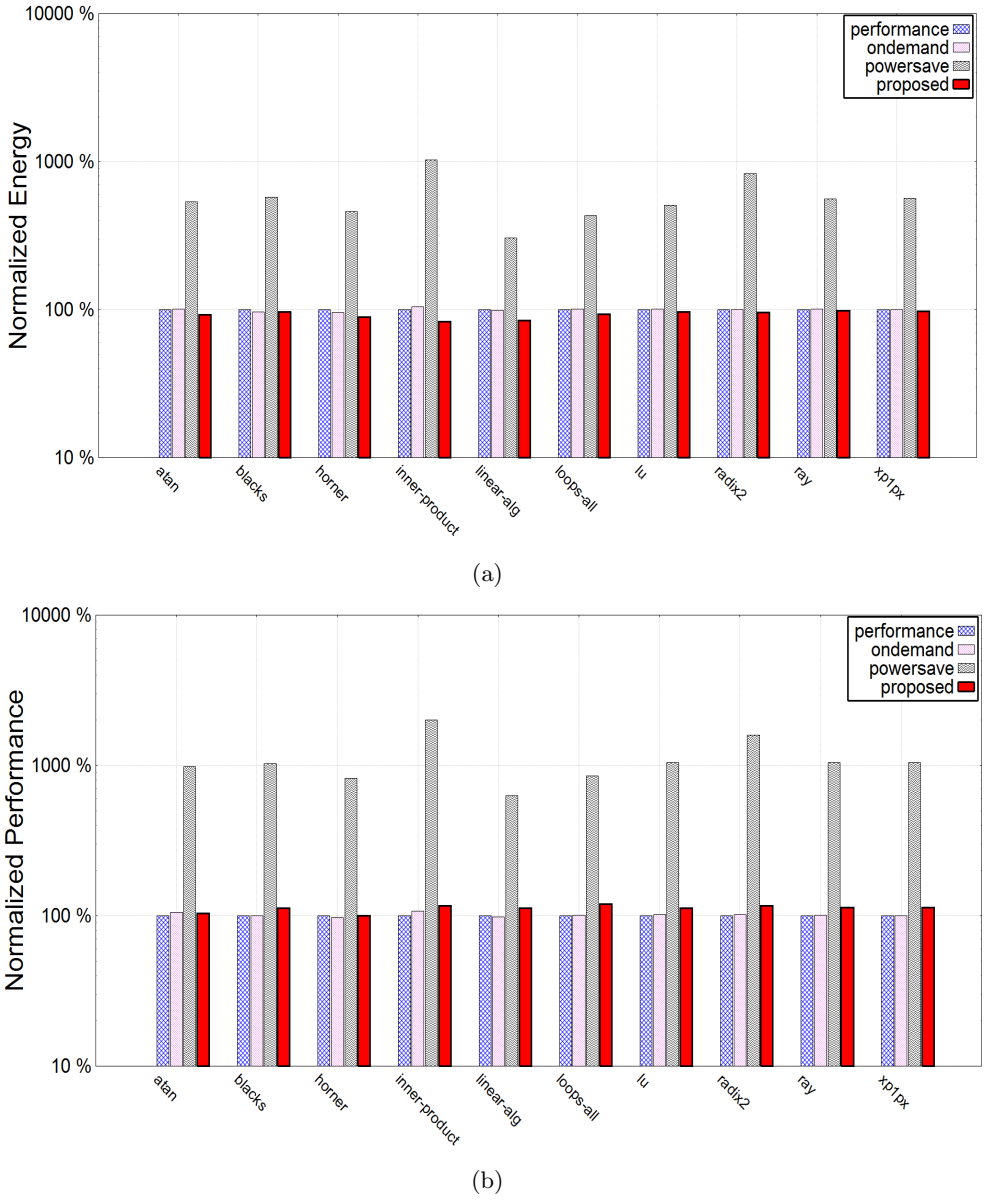

Fig. 4. The result of proposed OI-based DVFS algorithm. (a) Energy, (b) Performance

The OI-based DVFS algorithm shows that the energy consumption was, on average, 9% lower compared with a performance governor. In the case of a memory-bound application, the difference can be up to 17% in energy.

Acknowledgments

This research was supported by Basic Science Research Program through NRF of Korea, funded by MOE(NRF-2010-0020210)