An Energy Conservation DVFS Algorithm for the Android Operating System

Wen-Yew Liang* and Po-Ting Lai
Department of Computer Science and Information Engineering
National Taipei University of Technology
Taipei 106, Taiwan
*wyliang@mail.ntut.edu.tw

Che Wun Chiou
Department of Computer Science and Information Engineering
Ching Yun University
Chung-Li 320, Taiwan

Abstract—**Typically, when a user wishes to minimise the energy consumption for an application running on a handheld device, he/she may choose to set the processor speed to its slowest level. However, our study indicated that due to the processes involved in memory accesses, decreasing the CPU frequency may not always reduce the energy consumption. A critical speed has been defined as the CPU frequency, at which energy consumption can be minimised for a program. It can be used when the user wants to maximise energy saving for the device if performance is a less important issue. In this paper, an energy conservation DVFS algorithm is proposed to achieve this goal. It predicts and applies the critical speed as the target CPU frequency during the program's execution time. The algorithm relies on a prediction equation that is constructed based on the correlation between the critical speed and the memory access rate. We have implemented the algorithm on the Android operating system. Our results show that both the energy consumption and the performance can be improved than the situation of simply selecting the lowest frequency.**

Keywords- Embedded Systems; Low Power Software Design; DVFS; Energy-conservation; Android; Linux

I. Introduction

DVFS (Dynamic voltage and frequency scaling) and DPM (dynamic power management) are two methods commonly used to reduce energy consumption and to extend battery life for mobile devices. DPM has been widely used in portable devices to save power. Components can be put into low-power states when they are not in use. For example, the processor and the peripherals can change to the idle mode or the power saving mode when they are not in use. Different from DPM, DVFS provides an efficient energy saving mechanism for components when they remain in active states. DVFS is now usually supported by processors designed for mobile applications such as laptop computers or handheld devices, in which multiple voltage and frequency levels can be utilised by the system software in different conditions to save on energy consumption. For example, when an application does not need to be run at the highest performance, it may reduce the frequency and voltage so as to reduce the power consumption.

Many DVFS researches have proposed methods to reduce power consumption for applications while trying to maintain their performance. However, occasionally, users may wish to minimise the energy consumption instead of taking into account the performance issue. In such cases, the performance-oriented power saving algorithms may not be applicable. For example, when a user has used a smart-phone to record a video and then wants to convert the file format so as to upload the video to a website for sharing with friends, the user may wish to do the file format conversion with minimum energy consumption, because this kind of job does not usually need to be finished immediately. As a result, a method that provides maximum energy saving will be preferred, at the expense of performance, so as to extend the battery life.

Traditionally, users may choose to use the lowest CPU frequency for this purpose. However, from real measurements, we have observed that reducing the CPU frequency may not always reduce the energy consumption. The lowest energy consumption usually appears at some operating frequency other than the lowest one supported by the processor. This is mainly caused by memory accesses, which require an accessing latency independent of the CPU frequency. The frequency that can induce the lowest energy consumption is defined as the critical speed. One advantage of the critical speed is that it consumes less energy while providing better performance, compared to the slowest clock rate which users may typically consider for the minimum energy consumption. As a result, if the user wants to maximise the energy saving, the critical speed can be a better solution.

In this paper, we propose an energy conservation algorithm which predicts the critical speed during the task execution time, for use in the cases where the user wants to minimise the energy consumption. From experiments, we have found that a relationship exists between the critical speed and the memory access behaviour, which is represented by an index called the memory access rate (MAR). A correlation equation can thus be constructed to describe the relationship between MAR and the critical speed. It can be used in the algorithm to predict the critical speed during the run time for maximum energy saving. We have implemented an energy conservation DVFS algorithm in the Android operating system. The hardware counters that are provided by the processor for counting events, such as the number of cache misses, are used to collect the run-time MAR information, which is then applied in the correlation equation to predict the critical speed.

The remainder of this paper is organised as follows. In Section II, related works are described. Section III introduces the concept of the critical speed, discusses its relationship with the memory access rate, and then explains how the energy conservation algorithm works based on predicting the critical speed through a prediction equation which can be deduced from the relationship. The implementation of our DVFS algorithm on the Android operating system is introduced in

Section IV. In Section V, results from the experiment and a comparison with the traditional method for minimising the energy consumption are reported. Finally, the conclusions are given in Section VI.

II. Related Works

The dynamic power consumed due to the switching of gates contributes greatly to the total power dissipated in CMOS circuits. The dynamic power consumption can be stated by the following equation.

$$P_{dyn} = N_{sw} C_L V_{dd}^2 f \qquad (1)$$

In the equation, N_{sw} is the switching activity, C_L is the load capacitance, V_{dd} is the supply voltage, and f is the operating frequency. From (1), we can see that the power consumption is proportional to the product of the frequency and the square of the supply voltage. As a result, decreasing the voltage has a quadratic effect on the reduction of the power consumption.

Many previous works on DVFS have considered real-time systems, in which the job execution time can be extended to approach the deadline so that the frequency can be scaled down. This in turn saves the energy consumption. However, most of the handheld devices in use today are not real-time systems. In addition, the source of energy consumption not only includes the processor, but also some other parts such as the main memory, the storage and the peripheral devices.

In [1] and [2], Choi et al. proposed a DVFS technique called workload decomposition, in which the CPU workload is decomposed in two parts: on-chip and off-chip. This is based on some run-time statistics reported by the hardware counters. The on-chip workload means the execution cycles of the instructions in CPU operations, and the off-chip part represents the cycles for external memory accesses. Rajan et al. [7] [8] defined the memory access rate (MAR) according to the information provided by the hardware counters to evaluate the effect of external memory accesses. They proposed an online algorithm to achieve maximum energy saving by selecting the optimum frequency-voltage combination based on the system workload. Jejurikar and Gupta [5] indicated that the minimum energy consumption of a system may not appear at the slowest operating speed, and defined the critical speed of a task as the one which can assure the minimum energy consumption. We have also observed a similar phenomenon [6].

Based on these previous works, we have further observed that there is an interesting relationship between the memory access rate and the critical speed. In this paper, a DVFS energy-conservation algorithm for maximum energy saving is proposed. It predicts the critical speed based on this relationship. A correlation equation, called the MAR - Critical Speed Equation (MAR-CSE), is first conducted to describe the relationship for the target platform, and is then used in the algorithm for the critical speed prediction.

Our algorithm has been implemented in the Android operating system [12] as a power manager. Android is an open source operating system. It has been used extensively in smart phones and some other systems since it was released in 2008. It is built on top of the Linux kernel and contains a novel user level software stack for mobile devices. The software stack includes three layers: the Application layer in which the Android applications exist, the Application Framework layer which comprises components supported by Android for application developments, and the Libraries layer which contains native processes and shared libraries.

Android provides a simple power management framework. It provides application developers with a set of power management interfaces through the PowerManager class. The Android power manager basically utilises the DPM techniques. For example, when an application needs to use the CPU or the back light for a period of time, it has to allocate a WakeLock and acquire the lock so as to keep them powered on. If no application needs to hold the resource anymore, the CPU will enter a sleep mode and the back light will be turned off once the Android operating system thinks it can do so. While Android has provided an aggressive dynamic power management scheme, it is mainly used to reduce the energy consumption when some of the system components are idle or not being used. The Android operating system basically relies on the underlying Linux kernel support for dynamic voltage and frequency management. However, the default method, called the Ondemand governor implementation [11], was not designed for the case that we address, in which the user wants to minimise energy consumption when tasks are running.

The implementation of our energy conservation algorithm involves three parts, corresponding to the Android Application layer, the Android Libraries layer, and the Linux kernel-level driver. The MAR-CSE prediction equation is implemented in the power manager, which runs as an Android service to predict the critical speed which tends to minimise the energy consumption. During task execution time, the MAR information is retrieved and calculated from the hardware counters in the Performance Monitor Unit (PMU) [13] of the Intel XScale PXA270 processor that we used in our experimental environment. The power manager then uses the information as an input to the MAR-CSE equation to get the corresponding critical speed. Once the critical speed is obtained, it needs to be applied to the processor as the target frequency. However, the predicted critical speed is usually not one of the discrete frequencies supported by the processor. We have used a method based on the dual-speed proposed in [8] to find a pair of neighbouring frequencies so as to approximate the critical speed.

In addition to retrieving task-related information, the hardware counters can also be used to estimate power consumption. For example, Contreras and Martonosi [3] demonstrated a linear power estimation model for the processor by using the counters. The model provides an easy way to get power consumption information without using external hardware equipments for the measurement. In [9] and [10], Snowdon et al. built a time model and an energy model, in which the hardware counters were also used to estimate performance degradation and energy consumption. Although these methods are convenient for power estimation without the need of extra equipments and complicated setup for the measurement, to get more accurate data, however, we still chose to use standalone measurement hardware instead of the estimation methods.

III. Predicting the Frequency for Minimum Energy Consumption

In this section, the existence of the critical speed is first introduced. The relationship of the memory access rate and the critical speed is then discussed. After that, the core method of the proposed energy conservation algorithm – prediction of the critical speed by a prediction equation based on the relationship – is then explained.

A. *The Critical Speed*

In real applications, the CPU usually needs to stall and wait for memory accesses. Consequently, the memory operations also influence the total energy consumption. From previous studies [5][6], it has been found that reducing the frequency may not always induce lower energy consumption. We have observed that the lowest energy consumption usually appears at some operating speed other than the slowest clock rate. The frequency at which the lowest energy can be obtained is defined as the critical speed for the executed code on the running machine.

Fig. 1 illustrates how the energy consumption changes for two benchmark programs when different frequencies were used. The data were collected on our target platform through real measurements. The measured values only counted the energy consumed by the CPU and the memory. In the figure, we can see that the minimum energy consumption appears at a frequency higher than the lowest frequencies for both programs. This phenomenon is caused by the reason that as the CPU frequency is decreased, the total execution time is lengthened. Extra energy consumption will be introduced by the memory subsystem during the extended execution time, because the memory subsystem needs to be kept in active mode while the processor is working. The total energy consumption may thus be increased inversely.

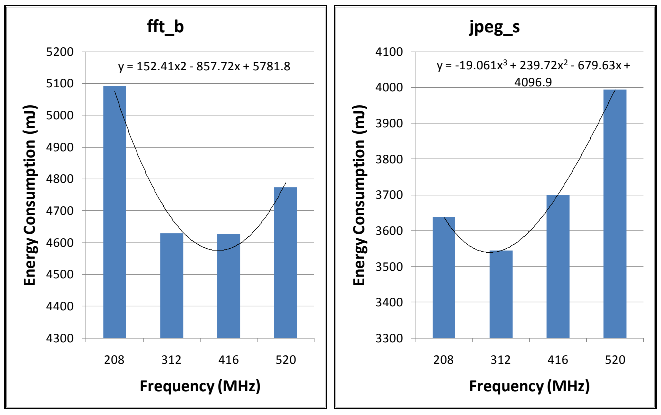
From the measured data, an approximation curve equation describing the relationship between the energy consumption and the frequency can be created by regression analyses. From the equation, the frequency which can induce the minimum energy value, i.e. the critical speed, can be obtained. The approximation equations computed from the measured data for fft_b and jpeg_b and the corresponding curves are also shown in Fig. 1. The local minimums of the curves indicate their critical speeds. As described in the first section, our goal is to find the critical speed during the execution time for the maximum energy conservation.

B. *Relationship between MAR and Critical Speed*

Since energy consumption is affected by the memory access behaviour of the running program, we use the memory access rate (MAR) index as an indicator for the memory access property. It is defined as the ratio of the total number of data and instruction cache misses (N_{cache_miss}) to the number of instructions executed (N_{instr_exec}). The formula of MAR is as follows.

$$MAR = \frac{N_{cache\_miss}}{N_{instr\_exec}} \tag{2}$$

The statistical numbers can be retrieved from the hardware counters such as the PMU counters provided by the Xscale PXA270 processor that we used. From this definition, we can see that a program with a lower MAR value implies that it tends to be a CPU-bound program, and a higher value implies that it tends to be a memory-bound program.



(a) fft_b (b) jpeg_b

Figure 1. Energy consumption for different frequencies.

We have chosen the MiBench [4] benchmark suite in our study. The selected benchmark programs include basicmath, bitcount, fft, sha, susan, jpeg, and mad. In considering that the size of the problem may affect how a program behaves, each benchmark program was tested with two sizes of input data: large and small size. The critical speed and the MAR value were measured for each of these programs. The Xscale PXA270 development board was used for the measurements. The frequency and voltage combinations that we used on the platform are listed in Table I. The CPU frequency of 104MHz, which is actually supported by processor, was not considered in our study because the corresponding bus frequency is different from the one that is used for other CPU frequencies. Since our study does not yet consider different bus frequencies, we have decided to exclude this configuration.

Table II shows the measured results for all the benchmark programs. In the table, the critical speeds have been normalised with respect to the highest frequency, 520MHz. The table contents have been sorted according to the order of the MAR values, from the smallest to largest number. Note that the critical speeds are theoretical numbers calculated from the approximation curve equations of the measured data. The critical speed is typically not one of the discrete frequencies supported by the CPU. For example, the critical speed of fft_b is 0.7628, which is about 396MHz and is not one of the supported frequencies listed in Table I.

From the sorted result in Table II, it can be easily observed that there is a very interesting relationship between the MAR and the critical speed. That is, MAR is inversely proportional to the critical speed. As the value of MAR increases, the corresponding critical speed value goes down. That is, a program with a lower MAR (which tends to be CPU-bound) will have a higher critical speed, whereas a program with a higher MAR (which tends to be memory-bound) will have a lower critical speed.

TABLE I. THE FREQUENCY/VOLTAGE USED IN OUR STUDY

CPU frequency (MHz)	Bus frequency (MHz)	Memory frequency (MHz)	CPU voltage (V)
208	208	104	1.15
312	208	104	1.25
416	208	104	1.35
520	208	104	1.45

TABLE II. CRITICAL SPEED AND MAR FOR THE BENCHMARKS

Benchmark Programs	MAR	Normalised Critical Speed
bitcount_b	0.000171	0.785
fft_b	0.001489	0.7628
bitcount_s	0.00182	0.7377
fft_s	0.00236	0.754
basicmath_s	0.002597	0.7201
sha_b	0.003209	0.6685
mad_b	0.003319	0.6567
susan_b	0.003708	0.6421
basicmath_b	0.005748	0.6888
mad_s	0.00848	0.5885
susan_s	0.009569	0.5806
sha_s	0.010571	0.5305
jpeg_b	0.013905	0.5854
jpeg_s	0.015367	0.56
Note: Program names appended with _b and _s are for big and small data size, respectively.		

This means that the energy consumption of a CPU-bound program may be raised as soon as the frequency is decreased. For example, in Table II, fft_b has a relatively low MAR value and is most likely a CPU-bound program. It has a higher normalised critical speed of 0.7628, which is about 396MHz. From Fig. 1(a), we can see that once the clock rate was set to a frequency lower than the critical speed, say 312MHz, its energy consumption started to increase.

In contrast, a memory-bound program will be able to further reduce the energy consumption with an even lower frequency. For example, from Table II, we can see that the MAR of jpeg_s has a relatively high value and is more like a memory-bound program. Its normalised critical speed is then a lower value 0.56, which is about 291MHz. As a result, in Fig. 1(b), we can see that when the frequency was changed from 416MHz to 312MHz, which is still higher than its critical speed, the energy consumption was able to continue to be decreased.

C. Construction of the MAR-CSE Equation

Since the critical speed, which by definition consumes the least energy, is usually higher than the lowest frequency provided by a processor, it can be used as the target frequency when the user wants to minimise the energy consumption. The major advantage of the critical speed is that not only can the energy saving be maximised, but also a better performance can be achieved. As a result, it can be used to replace the traditional method, which typically chooses to use the lowest frequency, for the purpose of maximum energy saving.

Our goal is to predict and use the critical speed during the task execution time. Based on the inversely proportional relationship between the MAR and the critical speed, a regression equation can be created from the measured data in Table II by the least square curve fitting method. This equation is called the MAR-based Critical Speed Equation (MAR-CSE). When a program's run-time MAR information can be obtained, the MAR-CSE equation can be used to predict the critical speed.

The MAR-CSE equation and the approximation curve for the measured data are both illustrated in Fig. 2. To get the run-time MAR information, some counter values must be retrieved from the PMU at task execution time. The MAR value can then be mapped to the corresponding critical speed through the MAR-CSE equation. At the critical speed, we may then have the programs running with minimum energy consumption but at a better performance level. In the following section, details of the implementation are described.

IV. IMPLEMENTATION OF THE ENERGY CONSERVATION DVFS ALGORITHM

In the Linux kernel under the Android operating system, the CPUfreq subsystem provides a modularised interface to manage the CPU frequencies. The policy manager for power management is called a Governor in Linux, which controls the CPU frequency through the interface of CPUfreq. Fig. 3 illustrates the CPUfreq infrastructure, in which the CPUfreq subsystem decouples the driver of the CPU-specific hardware from the management policies.

Several kernel-level governors have been supported by Linux for CPU frequency management. For example, the Performance governor maintains the CPU frequency at the highest frequency, the Ondemand governor manages the frequency according to the CPU utilisation, and the Powersave governor sets the CPU frequency to the lowest clock rate. Linux also provides the Userspace governor to export the available frequency information to the user space and allows the user-level governors to control the CPU frequency through the Linux sysfs interface. Among the governors, the Powersave governor is typically chosen when the user wants to minimise the energy consumption.

We have implemented the proposed energy conservation DVFS mechanism as an Android service, which is a type of Android program, in the user space. It is called the AD-DVFS governor. The AD-DVFS governor is responsible for obtaining the current MAR value and predicting the critical speed according to the MAR-CSE equation. The execution flow of the AD-DVFS governor is shown in Fig. 4. Once the AD-DVFS governor has been started, a Java thread is created to periodically perform the algorithm. According to our energy conservation algorithm, at the beginning of the execution flow, the AD-DVFS governor gets the statistical information from the PMU counters to compute the MAR value. Then, the critical speed is calculated from MAR-CSE. The frequency that will be applied to the CPU will be selected based on the critical speed. The whole process is repeated periodically with an execution interval of 500ms.

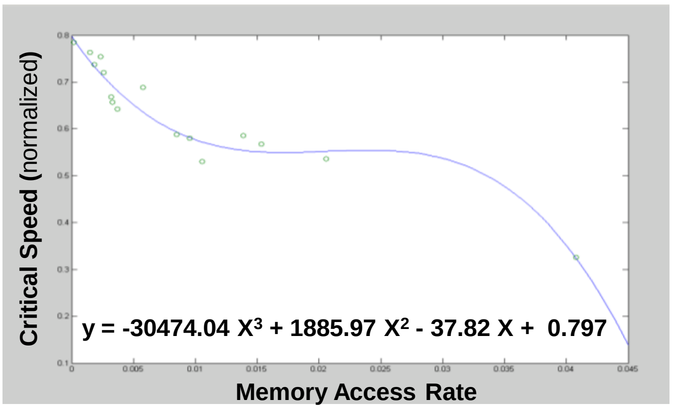


Figure 2. Curve of the MAR-CSE equation.

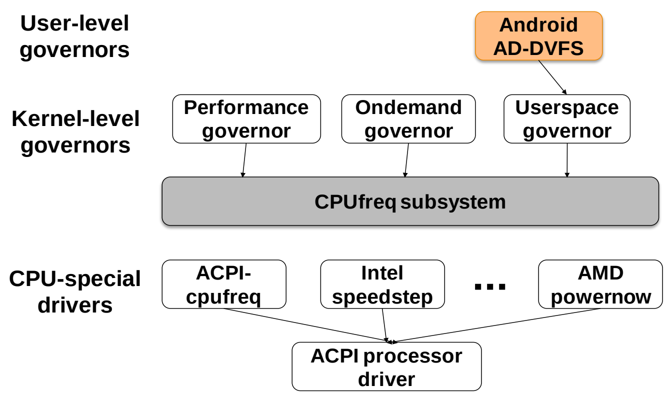


Figure 3. The infrastructure of CPUfreq.

The AD-DVFS governor was written as an Android service using the Java programming language. It runs on the Dalvik Virtual Machine, which is the Android's implementation of the Java Virtual Machine. The Java Native Interface (JNI) must be used to implement some native codes for the AD-DVFS governor to access the PMU counters and set the CPU frequency. The software stack for the AD-DVFS governor on the Android operating system is illustrated in Fig. 5. The native code under the JNI interface was built as a shared library. Several native functions have been implemented in the library for the AD-DVFS governor to control the CPU frequency and the PMU through the sysfs interfaces of the Linux kernel.

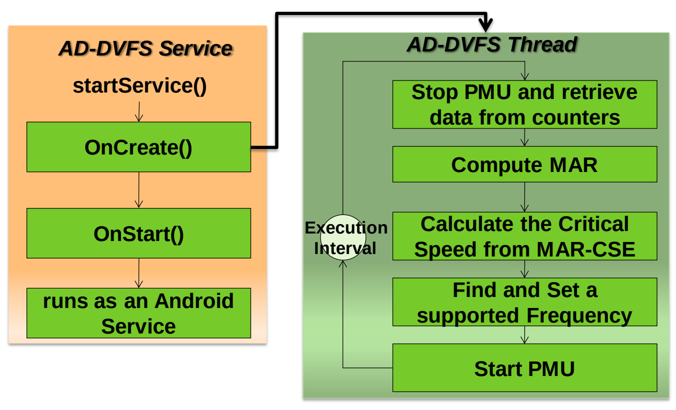

AD-DVFS Service
startService()
OnCreate()
OnStart()
runs as an Android Service
AD-DVFS Thread
Stop PMU and retrieve data from counters
Compute MAR
Calculate the Critical Speed from MAR-CSE
Find and Set a supported Frequency
Start PMU
Execution Interval

Figure 4. Execution flow and the algorithm of AD-DVFS.

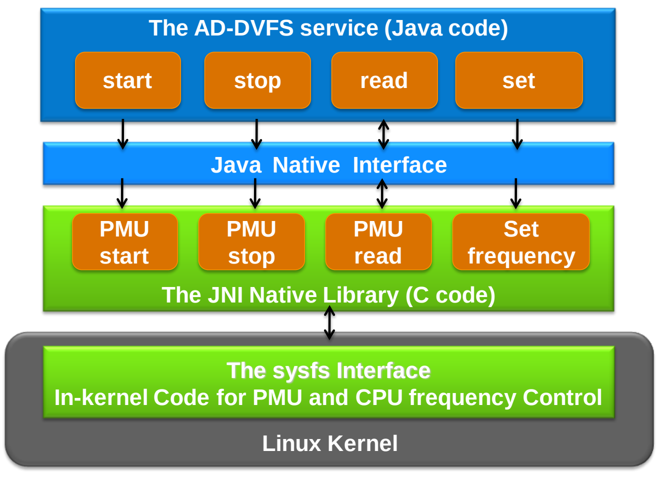

The AD-DVFS service (Java code)
start
stop
read
set
Java Native Interface
PMU start
PMU stop
PMU read
Set frequency
The JNI Native Library (C code)
The sysfs Interface
In-kernel Code for PMU and CPU frequency Control
Linux Kernel

Figure 5. Architecture of AD-DVFS.

For example, the library function PMU_Start() and PMU_Stop() are used to start and stop the PMU counters. The function PMU_Read() is used to read the values of the PMU counters. The other function Set_Frequency() is required to set the CPU frequency. Beneath the sysfs interface, we have also implemented the necessary kernel-level supporting code to control the hardware. The structure of the AD-DVFS implementation has one major advantage. That is, the policy manager of the AD-DVFS governor which is implemented in the Android application layer can be easily ported to machines running the Android operating system, leaving hardware-dependent parts to lower-level codes.

In the execution flow of the AD-DVFS governor, the predicted critical speed will be calculated from the MAR-CSE equation, which is a cubic equation involving floating-point operations. This equation cannot be efficiently computed in the Android application layer as a Java program, mainly because the Java program needs to be run on the Java virtual machine by way of interpretation. This is especially important when the program is designed to be run on an embedded system with less computing power and with a concern of extra energy consumption. To shorten the computation time, in our implementation, the MAR-CSE equation has first been converted into a pre-built lookup table, so that an approximated solution can be rapidly calculated by the interpolation method. Fig. 6 shows that the curve is divided into ten levels for ease of computation, and at the same time to reduce the size of the lookup table.

Once the critical speed has been calculated, the target frequency will be chosen in the next step of the execution flow. Since the critical speed is typically not one of the available frequencies supported by the processor, a dual speed method is instead used to find two neighbouring frequencies to approximate the critical speed. Fig. 7 shows how this method works. During the execution interval, a pair of neighbouring frequencies, including the least highest frequency $Freq_{high}$ above the critical speed $Freq_{cs}$ and the maximum frequency $Freq_{low}$ below $Freq_{cs}$, and a switch point , are calculated and applied. The processor first runs with the frequency $Freq_{high}$ until the switch point, then switches to the frequency $Freq_{low}$ and uses this until the end of the execution interval.

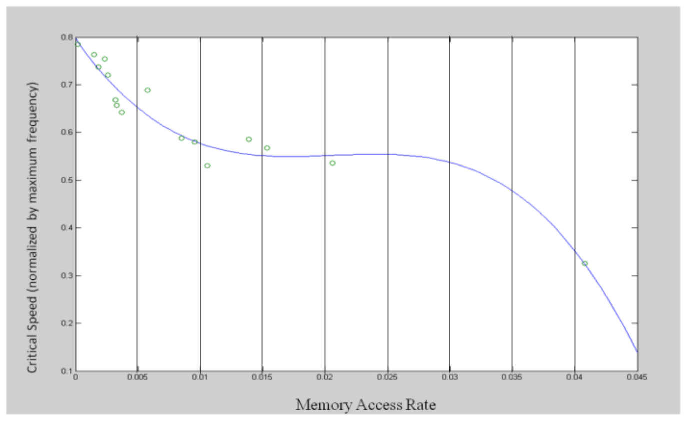


Figure 6. Partitioning of the curve for table lookup.

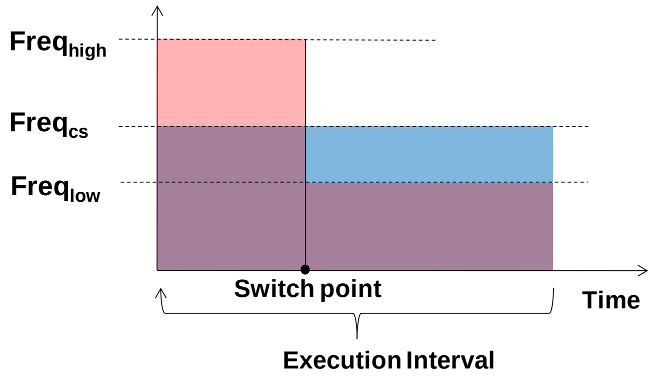


Figure 7. The dual speed method.

V. Evaluation

In this section, the experimental set-up is introduced, and the evaluation results are presented.

A. The Experimental Set-up

The experiments were performed on a real platform, the Creator PXA270 development board, on which we have ported the Android operating system version 1.0 using the Linux kernel 2.6.25 [15]. The LP3971 PMIC is used to support dynamic voltage adjustment for the platform. When the frequency is changed, the corresponding voltage will be changed accordingly. The NI USB-6251, a high performance data acquisition instrument (DAQ), was used in the measurements with the sampling rate set to 1000 samples per second. The voltage and the current of the CPU and the SDRAM were measured to compute their power consumption. The experiment set up is shown in Fig. 8. The MAR and the critical speed data listed in Table II were also measured on this platform. In the following subsection, the results collected from the experiments are reported.

B. Results of the Benchmark Programs

The AD-DVFS implementation of the proposed energy conservation algorithm has been evaluated with respect to the benchmark programs. It is compared to the cases where the clock rate was fixed at the highest frequency (520MHz) and the lowest frequency (208MHz) that we used in the target platform. A comparison of the results are presented in Fig. 9 and Fig. 10.

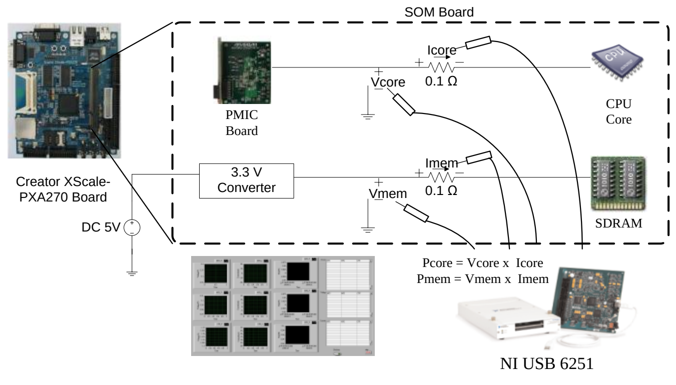


Figure 8. The measurement environment.

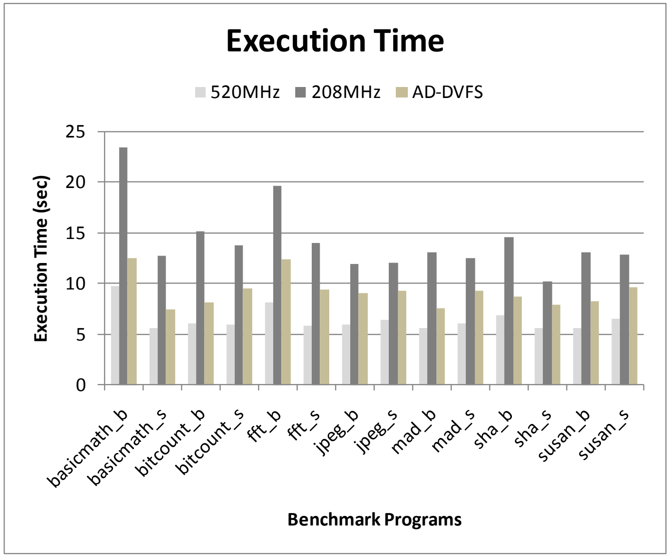


Figure 9. Execution time of the benchmark programs.

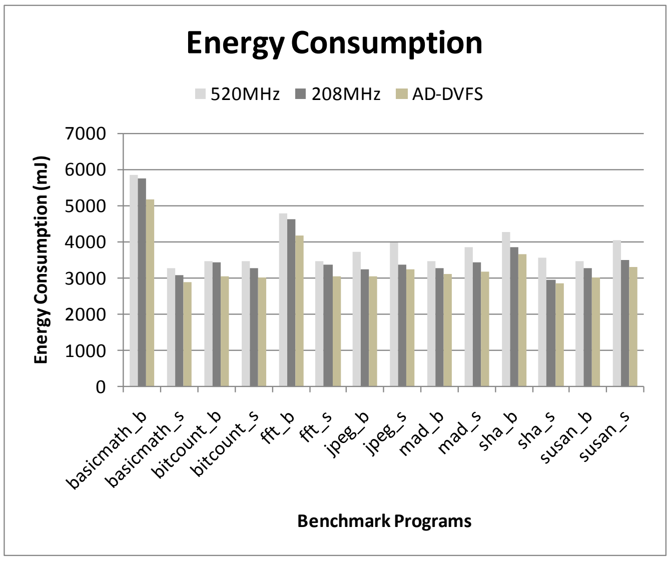


Figure 10. Energy consumption of the benchmark programs.

Fig. 9 shows the execution time for the benchmark programs. It is obvious that 520MHz results in the shortest execution time and 208MHz results in the longest time. For AD-DVFS, since a frequency between the highest and the lowest frequencies was selected as the target frequency (i.e. the critical speed), the execution time was basically between those two cases. Note that some programs, such as basicmath and bitcount, ran much faster under AD-DVFS than under 208MHz. This is because during the execution time, these programs were determined to be more like CPU-bound programs due to lower MAR values, and hence higher critical speeds were predicted. As a result, the programs were run with higher frequencies.

In Fig. 10, the results of the energy consumption for all of the benchmark programs are shown. Among the cases compared, 520MHz consumed the most energy for all programs. 208MHz consumed less energy, however it was not the least. Instead, the energy consumed by AD-DVFS is the lowest, since the algorithm was able to predict the critical speed and use it to set the target frequency.

The most interesting point from the results is that while AD-DVFS consumed less energy than the lowest frequency, its performance was better. More specifically, when compared to 208MHz, AD-DVFS saved more energy, from 2.6% to 11.1%, but ran faster, by 22.9% to 46.6%. These results show that the slowest CPU speed may not be a suitable frequency to use when the user wants to maximise energy saving. The critical speed not only can be a replacement for the lowest clock rate for minimum energy consumption, but can also provide better performance.

In addition to the benchmark programs, we have also tested the AD-DVFS implementation with respect to an open source application, named mpg123 [14], from SourceForge. In the test, a 2.4Mbyte music file was converted from MP3 format to a WAV format. The energy consumption and execution time are shown in Fig. 11 and Fig. 12, respectively. From the figure, we can see the AD-DVFS allows the program to run faster, while lower energy consumption can be achieved.

VI. Conclusions

Typically, when a user wants to minimise the energy consumption for those programs whose performance is not an important issue, the user may choose to set the lowest frequency for the CPU. However, we have observed that the least energy consumption may in fact appear at some operating speed other than the slowest clock rate. This operating speed is defined as the critical speed, and is closely related to the memory access behaviour of the program. In this paper, an energy conservation DVFS algorithm, based on the prediction of the critical speed, was introduced to adjust the frequency and the voltage during the run time so as to maximise the energy saving when energy consumption is the most important issue for the user.

The energy conservation algorithm has been implemented on the Android operation system as a user space service, called the AD-DVFS governor. The algorithm uses a prediction equation, namely MAR-CSE, to find the critical speed which can minimise the energy consumption. The equation can first be determined from an analysis of the relationship between MAR and the critical speed over benchmark programs on the target platform. During the task execution time, a dynamic MAR value can be obtained from the performance counters supported by the processor to predict the critical speed, through the MAR-CSE prediction equation.

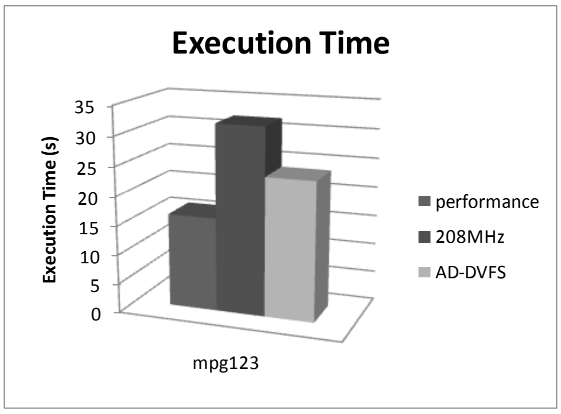


Figure 11. Execution time of other programs.

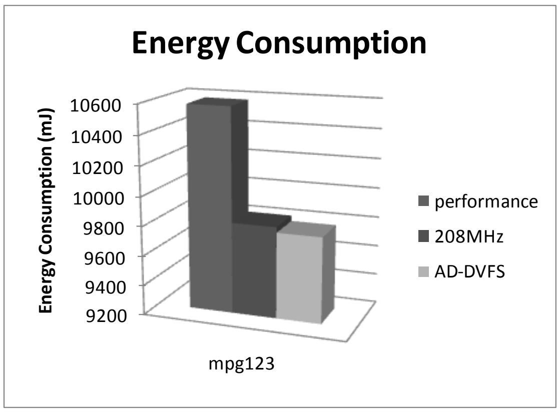


Figure 12. Energy consumption of other programs.

The AD-DVFS governor was realized on an Intel PXA270 XScale embedded platform on which we have ported the Android operating system, with the Linux kernel version 2.6.25. Results show that the AD-DVFS governor could save more energy than in the case where the slowest clock rate was used. In addition to the advantage of lower energy consumption, the AD-DVFS governor also provides better performance, since the critical speed is typically higher than the slowest speed supported by the CPU. As a result, the proposed energy conservation DVFS algorithm can be used to set the critical speed as the target frequency for minimum energy consumption, instead of simply adopting the lowest frequency, when the user needs to maximise energy conservation for applications running on the device.

Acknowledgment

This research was partially supported by the National Science Council, Taiwan under grant NSC 99-2221-E-027-047. The authors would like to thank the reviewers for their valuable comments.

References

[1] K. Choi, R. Soma, and M. Pedarm, “Dynamic Voltage and Frequency Scaling based on Workload Decomposition,” in Proc. 2004 Int. Symp. Low Power Electronics and Design, Aug. 2004, pp. 174-179, 2004.

[2] K. Choi, R. Soma, and M. Pedram, “Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-Off Based on the Ratio of Off-Chip Access to On-Chip Computation Times,” in Proc. 2004 Design, Automation and Test in Europe, vol. 1, p. 10004, 2004.

[3] D. Rajan, R. Zuck, and C. Poellabauer, “Workload-Aware Dual-Speed Dynamic Voltage Scaling,” in Proc. 12th IEEE Int. Conf. Embedded and Real-Time Computing Systems and Applications, pp. 251-256. 2006..

[4] D. Rajan, R. Zuck, and C. Poellabauer, “A Dual Speed Approach to Workload-Aware Voltage Scaling,” Technical Report TR-2006-05, University of Notre Dame, 2006..

[5] R. Jejurikar and R. Gupta, “Dynamic Voltage Scaling for Systemwide Energy Minimization in Real-Time Embedded Systems,” in Proc. of 2004 Int. Symp. Low Power Electronics and Design, pp. 78-81, 2004.

[6] W.-Y. Liang; S.-C. Chen; Y.-L. Chang; J.-P. Fang, “Memory-Aware Dynamic Voltage and Frequency Prediction for Portable Devices,” in Proc. 14th IEEE Int. Conf. Embedded and Real-Time Computing Systems and Applications, pp. 229-236, 2008.

[7] Android Developers website, [Online]. Available at: http://developer.android.com [Accessed: Dec. 2010].

[8] P. Venkatesh and S. Alexey, “The Ondemand Governor” in Proc. Linux Symposium, vol. 2, pp. 223-238, 2006, Available at: http://www.linuxinsight.com/files/ols2006/pallipadi-reprint.pdf.

[9] Intel XScale® Technology Overview (Intel® I/O Processors), [Online]. Available at: http://www.intel.com/design/intelxscale [Accessed: Dec. 2010].

[10] G. Contreras and M. Martonosi, "Power prediction for Intel XScale® processors using performance monitoring unit events," in Proc. 2005 Int. Symp. Low Power Electronics and Design, pp. 221-226, 2005.

[11] D.C. Snowdon, S.M. Petters, and G. Heiser, "Accurate on-line prediction of processor and memory energy usage under voltage scaling," in Proc. 7th ACM & IEEE Int. Conf. Embedded Software, pp. 84-93, 2007.

[12] D.C. Snowdon, G.V.D. Linden, S. Petters, and G. Heiser, "Accurate Run-Time Prediction of Performance Degradation under Frequency Scaling," in Proc. 2007 Workshop on Operating System Platforms for Embedded Real-Time Applications, pp. 58-64, 2007.

[13] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A Free Commercially Representative Embedded Benchmark Suite," in Proc. IEEE Int. Workshop on Workload Characterization, pp. 3-14, 2001..

[14] mpg123Source and Document of the Android Porting for Creator PXA270, [Online]. Available at http://www.ntut.edu.tw/~wyliang [Accessed: Dec. 2010].

[15] mpg123, [Online]. Available from: http://sourceforge.net/projects/mpg123/. [Accessed: Dec. 2010].