Applied Mathematics & Information Sciences
An International Journal

CPPM: a Comprehensive Power-aware Processor Manager for a Multicore System

*Slo-Li Chu**, *Shiue-Ru Chen, and Sheng-Fu Weng*

Department of Information and Computer Engineering, Chung Yuan Christian University, Chung Li, 32023, Taiwan

Received: 7 Sep. 2012; Revised 21 Nov. 2012; Accepted 23 Dec. 2012
Published online: 1 Mar. 2013

Abstract: The growing functionality of mobile devices explains increasing system performance requirements and the subsequent wide adoption of multicore processors. As mobile systems are battery powered, battery life largely limits these high performing multicore mobile devices. Developing an efficient power-aware processor manager for mobile multicore systems has received considerable attention. The conventional processor management system of embedded systems e.g., the Linux kernel scheduler incorporates an automatic scheme to control peripheral operations and processor frequency. However, this mechanism fails to consider user requirements, task loading, and operating status of processors in the multicore system to satisfy operating requirements. Therefore, this work presents a novel power-aware multicore processor manager, referred to herein as a comprehensive power-aware processor manager (CPPM), which integrates a system configuration selection algorithm (BPM-DFS), task re-scheduling mechanism (CTM), and precise system power estimation mechanism (PPM). The CPPM manager can dynamically set system configurations and rearrange executed tasks among multiple cores to comply with the limitation of power consumption that is assigned by the user. Moreover, the proposed CPPM is implemented on quad-core x86 Android system to compare with the capabilities of other scheduling mechanisms.

Keywords: Power-aware scheduling, processor manager, Android, Linux, task scheduling, multicore

1. Introduction

The growing functionality of mobile devices increases the requirements of system performance. High frequency, multicore processors are widely adopted to comply with performance requirements of mobile devices. Many of these high performing mobile devices use Android as their operating system to provide extreme user experience and multimedia functionalities. However, higher working frequency and core numbers of processors consume more power and reduce the battery life of such devices. These products are seriously limited in satisfying user demand. Conventional power-aware management mechanisms, including dynamic voltage frequency scaling (DVFS) [1][2], can adjust the wording frequency and voltage to reduce the power consumption of processors when remaining idle. Additionally, the efficiency of power savings can not increase since the DVFS mechanism fails to consider the actual workloads of each processor in the multicore system and adjust to the appropriate working frequency. Another solution integrates the DVFS mechanism with an OS task scheduler. Since the fundamental feature of Android is Linux kernel, the task scheduler of Android is also Linux task scheduler. Integrated with the Linux task scheduler, the power managing mechanism of Android/Linux kernel can automatically control the working frequency of the processors, based on the workload of the corresponding processor. Owing to the inability of mobile devices to execute heavy loading programs continuously, performance of the programs is not considered when the devices are idle. Therefore, the unused cores and working frequency can be turned off to reduce power consumption and extend the battery life of mobile devices. Unfortunately, the five automatic power modes provided by the Linux processor manager can not turn off the unused cores; the frequency scaling policy is too conservative. This inability also limits the feasibility of Android/Linux-based mobile devices.

This work presents a novel power-aware processor manager, referred to herein as comprehensive power-aware processor manager (CPPM), to configure dynamically processor cores to satisfy the limitation requirements of power consumption comply with limitations in power consump-

* Corresponding author: e-mail: slchu@cycu.edu.tw

tion that is assigned by the user. The CPPM processor manager comprises three mechanisms, a system configuration selection algorithm referred to herein as bounded-power multicore dynamic frequency scaling (BPM-DFS), a task monitoring and rescheduling mechanism referred to herein as core task mapping (CTM), and a precise estimating mechanism of system power consumption referred to herein as predictive power model (PPM). By integrating BPM-DFS, CTM, and PPM, the proposed CPPM can adjust the status of the multicore processors and balance the workload of each processor to enhance the system performance under the constraint of power consumption. Moreover, the CPPM manager is implemented on quad-core x86 Android system to compare with the capabilities of other scheduling mechanisms e.g., Linux/Android build-in power manager and SCA-ICA scheduling mechanism in order to compare how these power-aware scheduling mechanisms differ in power consumption.

The rest of this paper is organized as follows. Section 2 introduces pertinent literature. Section 3 then introduces the proposed CPPM manager. Next, Section 4 summarizes the experimental results. Conclusions are finally drawn in Section 5, along with recommendations for future research.

2. Related Works

2.1. Linux CPUFreq Governor

The built-in processor manager of Linux system is Linux CPUFreq Governor [4], which is based on the dynamic frequency scaling (DFS) mechanism to adjust the frequencies of cores on-the-fly in order to reduce the power consumption of the idle cores. In addition to considering the workload of task queues and task schedulers of each core in order to determine an appropriate frequency automatically, Linux CPUFreq Governor provides five execution modes for different situations, including performance, powersave, ondemand, conservative, and userspace. Users can adjust the execution mode manually by the supported APIs. These five modes are explained briefly as follows.

- **–Performance:** When the Linux CPUFreq Governor is set at ”performance” mode, the working frequency of the processors is set as the highest statically. Therefore, Linux CPUFreq Governor consumes the highest power consumption, yet achieves the best performance. It is also the default mode of the Linux computer system.
- **–Powersave:** When the Linux CPUFreq Governor set as at ”powersave” mode, the working frequency of the processors is set as the lowest statically. The power consumption and the working frequency of the processors can be reduced maximally, subsequently diminishing the overall performance significantly.
- **–Ondemand:** When the Linux CPUFreq Governor is set at ”ondemand”, Linux power manager automatically adjusts the working frequencies of the cores, based on their runtime workloads. Therefore, the processors must be capable of changing the working frequency quickly. Also, this mode provides several configurable parameters to set the sampling frequency, workload statistics, and threshold values. This mode focuses on an improved performance rather than lower power consumption.
- **–Conservative:** While resembling ondemand mode, this mode also relies on the Linux power manager to dynamically adjust the working frequency, based on the runtime situation. However, this mode smoothly adjusts to the frequency, rather than jump to the maximum or minimum frequency immediately. Therefore, it is feasible for battery-powered devices to save additional power yet degrade system performance.
- **–Userspace:** This mode fails to adjust the frequency dynamically. Users wanting to adjust the working frequency of specific cores must use the corresponding proposed APIs to set the frequency manually.

According to the above discussion, the Linux CPUFreq Governor can dynamically adjust the working frequency of the multicore processors, yet fails to turn off unused processors to save additional power, making it impossible to achieve a workload balance between the processors. Therefore, a power-aware processor manager for multicore Android/Linux systems must be developed for multicore computer systems.

2.2. SCA-ICA Scheduling Mechanism

Lee [3] developed a heuristic task assignment mechanism, referred to as sufficient-cores assignment and insufficient-cores assignment (SCA-ICA), to rearrange tasks for a multicore architecture in order to reduce power consumption. SCA-ICA can reschedule the tasks for every core in the multicore system to reduce the peak workload of cores and lower the corresponding working frequencies. Also, SCA-ICA classifies the tasks into three groups, heavy, medium, and light ones. If the amount of tasks is less than the number of available cores, SCA-ICA schedules light tasks and heavy tasks for the distinct cores by using the first-fit decreasing algorithm. If the amount of tasks is larger than the number of available cores, the tasks are reassigned by using the first-fit decreasing algorithm. Next, the heavy tasks are parallelized by SCA-ICA and then divided into several portions. These subprograms are arranged for cores by using the above policy. The cores are shutdown if they are idle after scheduling by SCA-ICA. The work frequencies of each core are assigned based on their workload. However, this algorithm is designed for many-core architectures. The speedup is limited if SCA-ICA is applied on x86 multicore architectures. Moreover, the heavy tasks for actual computer systems, including Android or Linux systems, are difficult to parallelize. The task sets are not a periodically fixed task set in the computer systems of consumer electronics, making SCA-ICA infeasible for Android or Linux based computer systems.

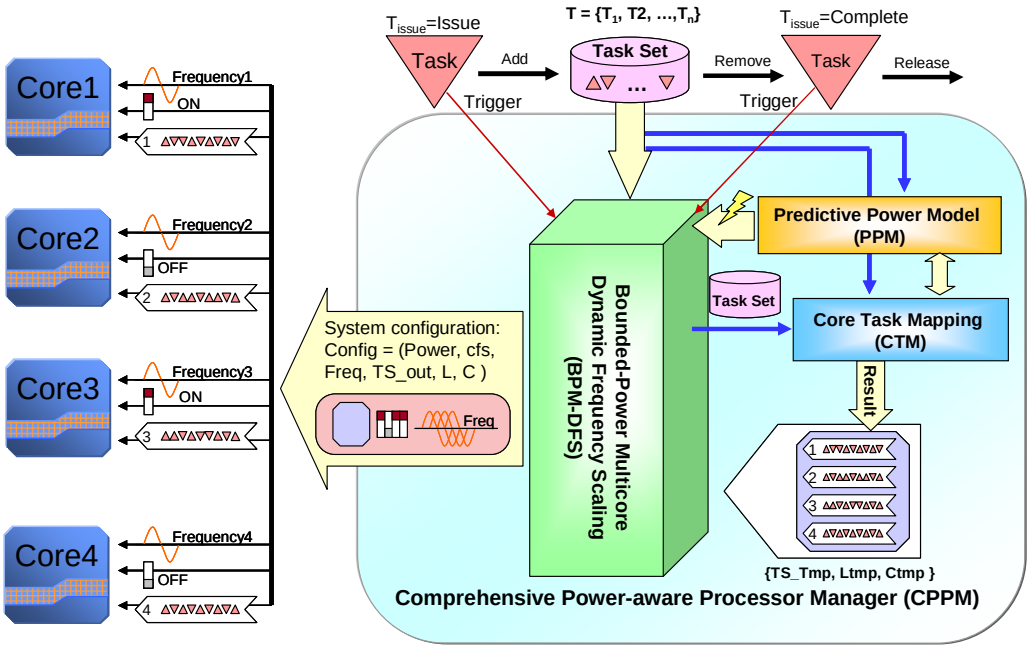

Figure 1 The fundamental architecture of proposed CPPM manager.

3. Details of Comprehensive Power-aware Processor Manager (CPPM)

This section discusses the proposed comprehensive power-aware processor manager ($CPPM$) in detail, which can configure the system dynamically when issuing or completing a task, based on the current status of the system, workload of each core, and loading of the new issued task. By managing the core shutdown/power-on, adjusting the working frequency, and rearranging the task queues of each core, the proposed CPPM can both comply with the "power budget" and achieve the best system performance, ultimately reducing the power consumption of the system. To achieve the above objective, this mechanism schedules tasks of the task set under three sub-objectives. First, power budget, which is assigned by the user, is the upper bound of the system power consumption. All possible system configurations are determined by CPPM using this power budget. Second, the tasks of each core are rescheduled to achieve an improved load balance. The total system performance can be improved when the peak workload of the core can be reduced. Third, the frequency of task migration among cores is minimized to reduce the additional power consumption. Finally, after the most appropriate system configuration and task schedule of each core are determined, CPPM sets the system configuration accordingly. To achieve the above design objective, CPPM comprises three major components: a system configuration selection algorithm referred to herein as bounded-power multicore dynamic frequency scaling ($BPM-DFS$), a task monitoring and rescheduling mechanism referred to herein as core task mapping (CTM), and a precise estimating mechanism of system power consumption referred to herein as predictive power model (PPM). Figure 1 illustrates the conceptual organization of CPPM. The following subsections introduce the details of these three mechanisms.

3.1. Bounded-Power Multicore Dynamic Frequency Scaling (BPM-DFS) Mechanism

In a multicore Android system with the CPPM manager, the task set is denoted as $T = \{T_1, T_2, \ldots, T_n\}$. The available core number of this multicore system is κ. The core enabling status of the multicore system is denoted as $cfs = \{P_1, ..., P_\kappa\}$; if $P_i = 1$, the corresponding core is power-on. If $P_i = 0$, the core is shutdown. The set of all possible combinations of the core enabling status is denoted as $CFS = \{cfs_1, ..., cfs_{2^{(\kappa-1)}}\}$, where $cfs_1 = (1, 0, .., 0)$ and $cfs_{2^{(\kappa-1)}} = (1, 1, ..., 1)$. Since the system requires at least one enabled core, the number of combinations of the core enabling status is $2^{(\kappa-1)}$. The set of possible working frequencies of core i is denoted as $F_i = \{f_{ij} | 1 \leq j \leq m, f_{i1} < f_{i2} < ... < f_{im}\}$, where f_1 represents the lowest frequency and fm refers to the highest frequency. Therefore, all working frequencies of the cores are denoted as $Freq = \{F_1, ..., F_g, ..., F_\kappa\}, F_g \in \{f_1, ..., f_m | f_1 < ... < f_m\}$. The workload and executed core number of task i denote as $T_i.L$ and $T_i.C$, respectively. The set of all tasks is denoted as $TS = \{T_1.C, T_2.C, ..., T_n.C\}$. Also, TS_Cur and TS_Tmp represent the current task set and temporary task set, respectively. The proposed system configuration selection

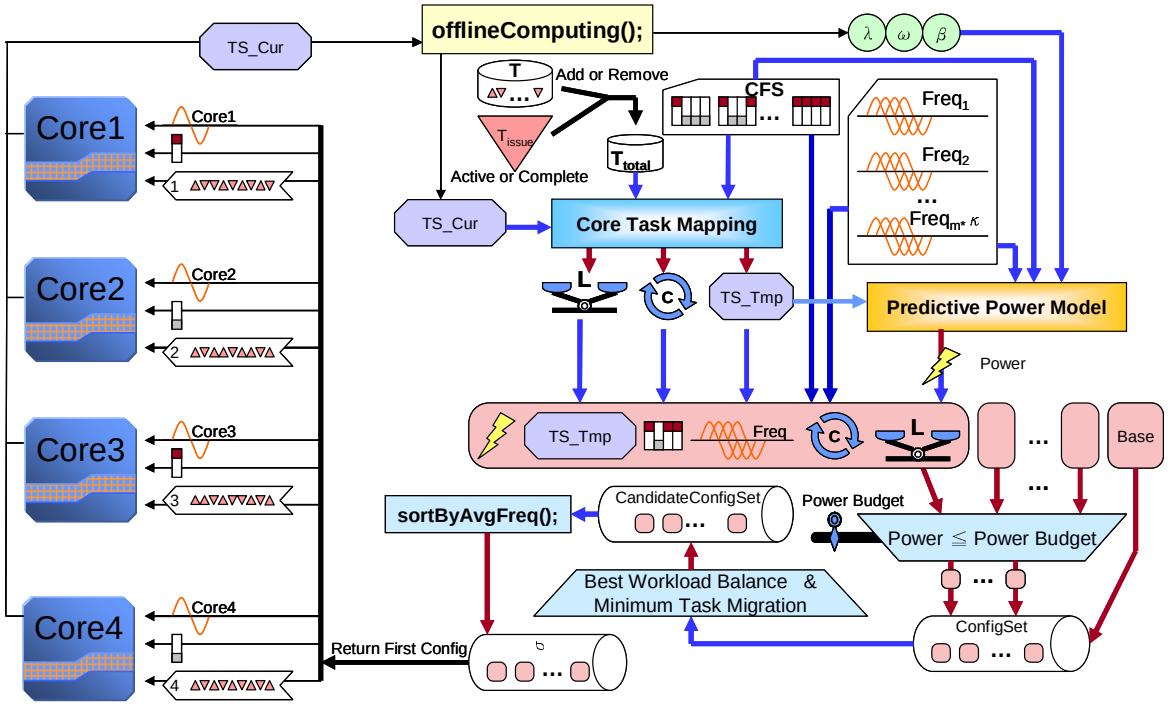

Figure 2 The architecture of BPM-DFS mechanism.

mechanism, Bounded-Power Multicore Dynamic Frequency Scaling ($BPM-DFS$), must be executed to assign a "Power Budget" by users, which refers to the upper bound of the system power consumption that users are allowed. BPM-DFS can be activated when a task is issued initiated ($T_{issue} = Issue$) or completed ($T_{issue} = Complete$). BPM-DFS can determine the possible system configurations, and select one of the most appropriate configurations, which can achieve the lowest task migration number, best load balance, and highest working frequency. The core/task scheduling can rearrange the task queue of each core to achieve an improved load balance and performance. The feasible system configuration of core enabling status (cfs), set of working frequencies of cores ($Freq$), and rearranged task set (TS_out) can be determined. Finally, the system configuration, denoted as $Config = (Power, cfs, Freq, TS\_out, L, C)$, can be generated by BPM-DFS, along with the multicore Android system set as well. The configuration consists of five tuples, where *Power* denotes the predictive power consumption of this configuration; L represents the highest workload in these cores; and C refers to the maximum task migration number. Figure 2 illustrates the detailed execution flow of BPM-DFS.

The auxiliary functions of CPPM manager are listed as follows.

–**offlineComputing()** evaluates and returns pertinent environmental parameters, λ, ω, β, and TS_Cur, which are required by BPM-DFS and Predictive Power Model.

–**CoreTaskMapping(T, cfs_i, TS_Cur)** reschedules and reassigns the given task set T under the system configuration of cfs_i by using the given core/task mapping algorithm. The rearranged task set, TS_Tmp, is then returned. The workload value of TS_Tmp is evaluated and stored into L_{tmp}. The task migration number, C_{tmp}, is also obtained by comparing the updated TS_Tmp and original TS_Cur. The next subsection discusses the detailed mechanism.

–**sortByAvgFreq(CandidataConfigSet)** sorts all of the configurations in the CandidataConfigSet set, based on their average working frequencies in a descending order and, then, returns the sorted CandidataConfigSet set in the σ.

–**Load(i, TS)** evaluates and returns the workload value of the TS task set on the core i, where the TS is scheduled and reassigned to improve the load balance. This value is also adopted to estimate the power consumption of the current configuration by Predictive Power Model, as discussed in the next subsection.

3.2. Core Task Mapping (CTM) Mechanism

Results of this study demonstrate that the workloads of cores significantly affect power consumption of cores. Also, the workload balance of the cores in the multicore system also affects the overall performance of the target computer system. Therefore CPPM consists of an effective task

rescheduling mechanism, referred to herein as core task mapping (CTM), to rearrange the tasks among the cores and improve the workload balance. Figure 3 shows the execution flow of CTM. CTM considers the current configurations of T_{total}, cfs_{tmp}, and TS_Cur, to schedule and rearrange the tasks by using the algorithms of Worst-Fit Bin Packing and First-Fit Bin Packing [6] and, then, returns the results of TS_Worst-Fit and TS_First-Fit, respectively. Since the task migration consumes additional power, the proposed CTM selects the scheduled task set with the optimum workload balance and minimal task migration number. The task migration number is obtained based on a comparison of TS_Cur, TS_First-Fit, and TS_Worst-Fit. C denotes the total thread migration number, which increases if a task is moved from another core. The maximal workload value of the cores, L, is then obtained. Since a smaller L value denotes an improved system performance and lower power consumption, the L values of scheduled task sets by using Worst-Fit Bin Packing and First-Fit Bin Packing algorithms are obtained. Finally, the task set with a smaller L and C is stored into the tuple of $\{TS\_Tmp, L_{tmp}, C_{tmp}\}$.

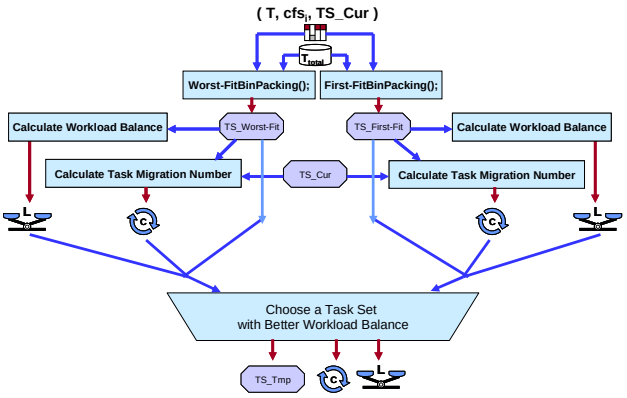

Figure 3 The scheduling flow of CTM mechanism.

3.3. Predictive Power Model (PPM)

As mentioned earlier, implementing the entire CPPM manager includes BPM-DFS and Core Task Mapping algorithms, need the power consumption of the current system. Actual power consumption of the evaluated computer can not be measured precisely and immediately by the power meter. Therefore, this work presents a novel power estimation mechanism, referred to herein as predictive power model (PPM). PPM can estimate the current power consumption of the computer system by identifying three major portions of the power consumption: power consumption of core execution, power consumption of the computer system except for cores, and baseline power consumption of the idle core, as shown in Eq. 1.

$$PredictivePowerModel = \varepsilon + \beta \quad (1)$$

In Eq.1, ε and β denote the power consumptions of cores and other components except for cores in the computer system, respectively. Notably, β can be treated as a constant when the configurations of the components in the computer system are the same. When the multicore system contains κ cores, the power consumption of a core is denoted as μ. Eq. 2 shows the total power consumption of the cores, ε.

$$\varepsilon = \sum_{h=1}^{\kappa} \mu_h \quad (2)$$

According to the results of [1][2], power consumption of the core is formulated as $P = kCV^2 f$, where k denotes the constant; C represents the effective capacitance of the core; V refers to the working voltage; and f is the working frequency. According to our results, while the system workload is increased, the power consumption of the corresponding cores is increased. Additionally, the enabling status (power-on/shutdown) of the core also affects the power consumption of the core. Therefore, Eq. 2 can be extended as in Eq. 3.

$$\mu_h = P_h \times C \times V_h^2 \times F_h \times \varpi \times Load_h \quad (3)$$

Where P_h denotes the enabling status of the core h; $P_h = 1$ refers to a situation in which the power of core h is turned on; $P_h = 0$ refers to a situation in which core h is shutdown; F_h represents the working frequency of core h; V_h denotes the working voltage of core h; and $Load_h$ refers to the workload of core h, which can be estimated by the auxiliary function, $Load()$, and the current TS of this core. Moreover, ω is a constant factor of workload and the power consumption of the core. Finally, the overall power consumption of the system can be represented as Eq. 4.

$$PredictivePowerModel = (\sum_{h=1}^{\kappa} P_h \times \lambda \times F_h \times \varpi \times Load_h) + \beta \quad (4)$$

Where ε, β, and ω can be obtained by using the auxiliary function, offlineComputing(), which varies with the varied computer system.

4. Experimental Results

This section discussed the experimental results of CPPM, which runs on an actual multicore Android system. The power-aware scheduling capabilities of CPPM, SCA-ICA [3] and Linux CPUFreq Governor [4] are also compared and adopted in this experiment. The multicore Android platform adopts Intel Core 2Quad Q6600 and Android 2.3.5 with Linux Kernel 2.6.39. Notably, Intel Core2Quad Q6600 is a quad cores x86 processor, in which the available working frequencies of each core is 1603MHz, 1870MHz, 2136MHz, and 2403MHz. The actual system power consumption is

measured by a recordable digital power meter. The experiments are based on a task list with sixteen individual tasks executed within the same period. This task list can represent the controlled case of executing tasks to evaluate the results of power consumption under three power management mechanisms. The task lists, task execution sequence, and three scheduling mechanisms are implemented to evaluate the power consumption and compare the scheduling results of CPPM, SCA-ICA, and Linux CPUFreq Governor. Next, a heavy loading benchmark application is executed in the background to represent the general situation in the real multicore Android system. The four heavy loading applications adopted in this section are **db**, **mtrt**, **gcc**, and **mcf**. The former two benchmarks are adopted from SPECJVM98 [5], the latter two benchmarks are adopted from SPEC2000 [5] to illustrate the execution by Dalvik Virtual Machine (DVM) and execution by native program (JNI), respectively. Since Linux CPUFreq Governor is not shut down or the power of the cores is turned on, the experiments of Linux scheduling turn on all of the cores.

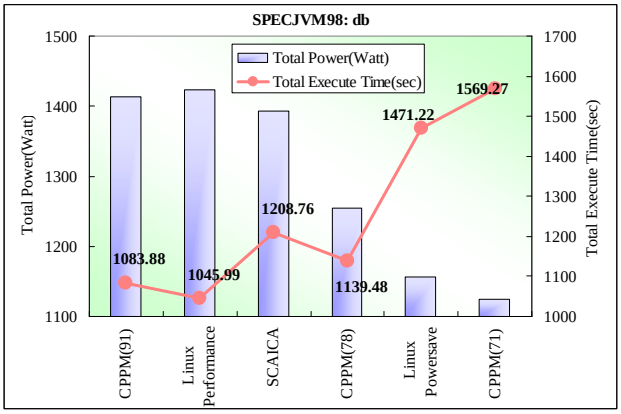

Figure 4 The experimental results of three scheduling mechanisms with **db** benchmark.

The general situation of a multicore Android system is demonstrated in the first experiment, which adopts a heavy loading program executed in the background. The heavy loading task, **db** benchmark as adopted from SPECJVM98, also functions as the performance metric to evaluate how the three scheduling mechanisms i.e. CPPM, Linux CPUFreq Governor, and SCA-ICA differ in performance. Figure 4 summarizes the experimental results, where the left Y-axis denotes the power consumption of the three scheduling mechanisms, and the right Y-axis represents the execution time of the **db** under three scheduling mechanisms. The Power budget of CPPM is set as 71, 78, and 91 to represent low power, general case, and high performance situations, respectively. The execution mode of Linux CPUFreq Governor is set at "powersave" and "performance". The power consumption of Linux Performance mode is 1423, while the execution time of **db** is 1046 sec. However, when the power budget of CPPM is sets as 91, the power consumption is only 1413, and the execution time of **db** is 1084 sec. The power consumption of CPPM is better than that of the Linux Performance mode, owing to the improved power management capability of CPPM under a heavy loading system. Also, the Linux performance mode is the default configuration of Linux/Android system. Experimental results indicate that CPPM can obtain a power savings of 1%. Consider a situation in which the power budget of CPPM is set as 71. Under this situation, CPPM can reduce power consumption by 21% over that of the Linux performance mode. Although CPPM with power budget set as 71 can save power by 20% over that of SCA-ICA, the former takes more time than the latter in completing the execution of **db**.

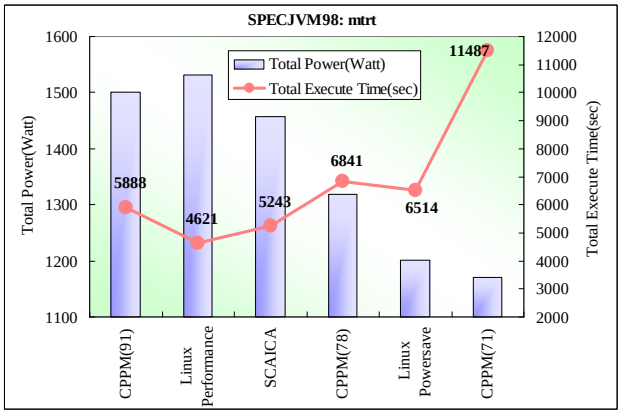

Figure 5 The experimental results of three scheduling mechanisms with **mtrt** benchmark.

Exactly how the three scheduling mechanisms differ in performance is evaluated in the second experiment, in which the heavy loading task, **mtrt** benchmark as adopted from SPECJVM98 are used to act as the Android application that is executed on Dalvik Virtual Machine. The experimental setting is the same as that of the first experiment. Figure 5 summarizes the experimental results. The power consumption of Linux performance mode is 1532, and the execution time of **mtrt** is 4621 sec. However, when the power budget of CPPM is set as 91, the power consumption is only 1413; in addition, the execution time of **mtrt** is 1084 sec. Although the power consumption of CPPM is better than that of Linux performance mode by 2%, the execution time of the former is slower than that of the latter by 22%. Consider a situation in which the power budget of CPPM is set as 71. Under this situation, CPPM can reduce power consumption by 24% over that of the Linux performance mode. CPPM with Power Budget set as 71 can save power consumption by 20% over that of, the former takes more time than the latter in executing **mtrt**.

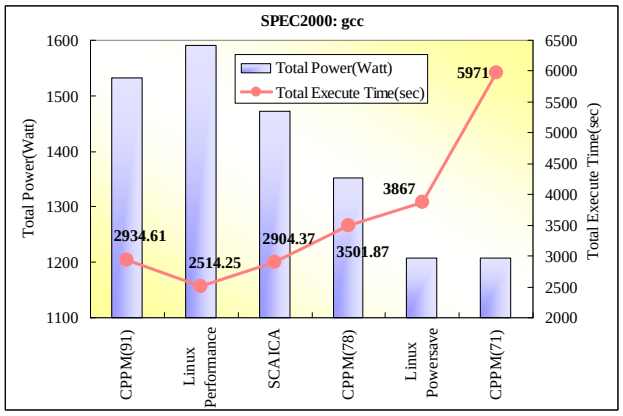

Figure 6 The experimental results of three scheduling mechanisms with **gcc** benchmark.

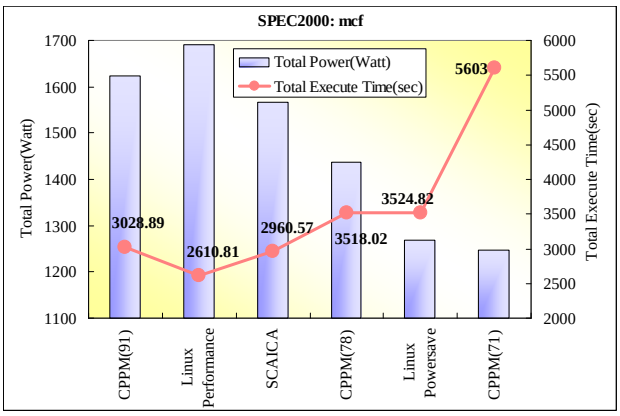

Figure 7 The experimental results of three scheduling mechanisms with **mcf** benchmark.

Exactly how three scheduling mechanisms CPPM, Linux CPUFreq Governor, and SCA-ICA differ in performance is evaluated in the third experiment, in which the heavy loading task, **gcc** benchmark as adopted from SPEC2000 act as the native application. The experimental setting is the same as that of the above two experiments. Figure 6 summarizes the experimental results. Power consumption of the Linux performance mode is 1590, while the execution time of **gcc** is 2514 sec. However, when the power budget of CPPM is set as 91, the power consumption is only 1532, and the execution time of **gcc** is 2935 sec. Although the power consumption of CPPM is lower than that of the Linux performance mode by 4%, the execution time of the former is slower than that of the latter by 14%. Consider a situation in which the power budget of CPPM is set at 71. Under this situation, CPPM can reduce power consumption by 24% over that of the Linux performance mode. Although CPPM with Power Budget set at 71 can save power consumption by 18% over that of SCA-ICA, but the former consumes more time than the latter in completing the execution of **gcc**. Additionally, CPPM (71) can save power consumption by 3% over that of the Linux powersave mode. This finding illustrates the advantage of CPPM mechanism in saving more power consumption than that of the power manager in the widely adopted Linux operating system.

Exactly how the three scheduling mechanisms differ in performance is evaluated in the fourth experiment, in which the heavy loading task and **mcf** benchmark adopted from SPEC2000 are used to function as the native application. The experimental setting is the same as that of the above two experiments. Figure 7 summarizes the experimental results. The power consumption of Linux performance mode is 1690, while the execution time of **mcf** is 2611 sec. However, when the power budget of CPPM is set at 91, the power consumption is only 1623, and the execution time of **mcf** is 3029 sec. Although CPPM consumes 4% more power than the Linux performance mode, the execution time of the former is slower than that of the latte by 14%. Consider a situation in which the power budget of CPPM is set as 71. Under this situation, CPPM can reduce more power consumption than the Linux performance mode by 26%. Although CPPM with power budget set as 71 can save more power consumption than SCA-ICA by 20%, the former consumes more time than the latter in completing the execution of **mcf**. Additionally, CPPM set at 71 can save more power consumption than the Linux Powersave mode by 2%. This finding illustrates the advantage of CPPM mechanism in saving more power than the power manager in the widely adopted Linux operating system. The above comparison reveals that the more power savings generally induces a greater execution time. Since CPPM allows users to assign the expected power budget, users can select the most feasible power budget for their specific purpose.

5. Conclusions

The growing functionality of mobile devices explains increasing system performance requirements and the subsequent wide adoption of multicore processors. The Android/Linux operating systems are widely adopted to satisfy the requirements of high performance multimedia applications. The conventional power management system of embedded systems such as the Linux kernel scheduler adopts an automatic scheme to control the usage of peripheral operations and processor frequency. These mechanisms fail to consider user requirements, task loading, and operational status of processors in the multicore system, to comply with actual operating conditions. Also, the multiple cores are not required since most of the idle time of Android devices is light loading. The unused idle cores can be shut down to save more power. Therefore, this work describes a novel power-aware scheduling mechanism referred to herein as comprehensive power-aware proces-

sor manager (CPPM), which can dynamically set system configurations, turn off the idle cores, adjust the working frequency, and rearrange executed tasks among multiple cores, to adhere the limitation of power consumption that is assigned by the user. Experimental results reveal that CPPM can save 26% and 18% more power consumption than the Linux performance mode, and SCA-ICA, respectively. Moreover, CPPM can save 3% more in power consumption than that of the Linux powersave mode. Above results demonstrate that CPPM can reduce system power consumption by using feasible system configuration decision and task rearrangement.

Acknowledgement

This work is supported in part by the National Science Council of Republic of China, Taiwan under Grant NSC 101-2221-E-033-049.

References

[1] E. Talpes and D. Marculescu, IEEE Transactions on Very Large Scale Integration Systems, **13**, 591-603 (2005).
[2] G. Magklis, G. Semeraro, D. H. Albonesi, S. G. Dropsho, S. Dwarkadas, and M. L. Scott, IEEE Micro, **23**, 62-68 (2003).
[3] W. Y. Lee, Proc. 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, 216-223 (2009).
[4] P. Mochel, Proc. Linux Symposium, 326-339 (2003).
[5] SPEC, SPEC Benchmark Suits, http://www.spec.org/.
[6] J. Wang, B. Ravindran, and T. Martin, Proc. IEEE Workshop on Software Technologies for Future Embedded Systems, 21-28 (2003).

Slo-Li Chu received his PhD degree in Electrical Engineering from National Sun Yat-sen University in 2002. He is currently an assistant professor of Department of Information and Computer Engineering, Chung Yuan Christian University, Taiwan. His research interests include computer architectures, parallelizing compilers, system level modeling, system-on-chip design, GPU architectures, and embedded system.

Shiue-Ru Chen received his MS degree in Information and Computer Engineering from Chung Yuan Christian University, Taiwan, in 2010. He is currently pursuing for his PhD degree in Electronic Engineering at Chung Yuan Christian University, Taiwan. His research interests include computer architectures and embedded system.

Sheng-Fu Weng received his MS degree in Information and Computer Engineering from Chung Yuan Christian University, Taiwan, in 2012. His research interests include computer architectures and embedded system.