2012 Fifth International Symposium on Parallel Architectures, Algorithms and Programming

Design a Low-Power Scheduling Mechanism for a Multicore Android System

Slo-Li Chu
Department of Information and Computer Engineering
Chung Yuan Christian University
Chung Li, Taiwan
slchu@cycu.edu.tw

Shiue-Ru Chen, Sheng-Fu Weng
Department of Information and Computer Engineering
Chung Yuan Christian University
Chung Li, Taiwan
roger70366@gmail.com, fzn792@gmail.com

***Abstract*—Nowadays, multicore processors are widely adopted in the hand-held systems. Since the hand-held systems are powered by battery, the battery life will become the dominated limitation. An efficient low power scheduling mechanism for hand-held multicore system is become important today. This paper propose a novel low power scheduling mechanism, called Bounded-Power Multicore Dynamic Frequency Scaling (BPM-DFS), which integrates a system configuration selection algorithm and a task re-scheduling mechanism. According to the assigned power budget by the user, BPM-DFS can dynamically adjust the configuration of the multicore system to control the suitable alive core number, working frequency, and task reassignment, to achieve good performance and under the limitation of power consumption. The proposed BPM-DFS has been implemented on quad-core x86 Android system to compare the actual capabilities of Linux/Android build-in power managers. The experimental results reveal that BPM-DFS can save 25 % power consumption than Linux Performance mode.**

Keywords:Low Power Scheduling, Android, Linux, Task Scheduling, Multicore.

I. INTRODUCTION

The design of multicore architecture has become mainstream currently due to evolution of semiconductor technology. And how to reduce energy consumption of battery and battery life had become major research issue. Past research are in progress for this issue, increasing battery life by using better battery material and design, through improvement of process technology, using more energy saving electronic material or circuit design, clock gating, power gating, optimization of software...etc. methods to decrease system energy consumption. Due to system need high performance mode in the most part of status at some time. Even system's CPU computation ability becomes lower when the workload is very low or none. If CPU can decrease working frequency or voltage, energy consumption will be reduced. Thus the one of widely studied and discussed method is that CPU adjust working frequency and voltage to reduce energy consumption by task scheduling and Dynamic Voltage and Frequency Scaling (DVFS) [1][2][3] for the status of system at that time. In some situation, system turns on multiple cores and running in lower frequency saves more energy than system turns on single core and running in high frequency. Thus through core management is one of the important method of reducing energy consumption.

Android is a common system in major portable devices. Due to Android's OS kernel is Linux kernel, and Linux kernel don't manage core number and working frequency for multicore architecture, thus the status of using of remaining cores and high working frequency execution make the additional power consumption of system, and decreasing the battery life. However, it also means processor will handle the problem that become more serious along the energy consumption. At the same time, Android usually execute on smart phone or tablet PC, and these devices often stay in idle status or sleeping status in over one of three time. This time is proper to decrease power consumption through turning off CPU core and decreasing working frequency. Although Linux kernel don't manage core number and working frequency for multicore architecture, but Linux provides API to let us can control CPU directly base on Advanced Configuration and Power Interface (ACPI) [4] for x86 system, for example, the alive of cores and the scheme of frequency.

This paper proposes a new algorithm: Bounded-Power Multicore DFS Scheduling Algorithm. The algorithm can according system real workload at that time to adjust system configuration depend on the system of non-periodic non-fixed task set for every new joined and finished task, and at the user's assigned most highest energy consumption, as possible as hence system performance. In order to reach the target, our algorithm according system's situation to adjust working frequency, core alive status and performance.

The rest sections of this paper are arranged as following. Section 2 introduces the related works, Section 3 introduces BPM-DFS algorithm. Section 4 provides an example of comparing BPM-DFS, Linux Scheduling, and SCA-ICA. Section 5 discusses the results of experiments. At last, Section 6 will make a conclusion.

II. RELATED WORK

In this section, two low power scheduling mechanisms are reviewed and discussed.

2168-3034/12 $26.00 © 2012 IEEE
DOI 10.1109/PAAP.2012.12

- **SCA-ICA**

Sufficient-Cores Assignment and Insufficient-Cores Assignment (SCA-ICA), proposed by [9], is a heuristic task assignment algorithm for multicore architecture. This algorithm considers task's utilization to divide task from heavy task, medium task and light task. If core number is enough, light tasks and medium tasks are distributed to cores by First-Fit Decreasing mechanism; otherwise, they are distributed to cores by Worst-Fit Decreasing mechanism. And then heavy task will be parallelized to many medium tasks or light tasks, and will be distributed as above methods. This algorithm will turn off low utilization core of CPU when task moves from one to another. At last the algorithm will according EDF [10][11] to decide each core's frequency. This paper [9] is proper to periodic task and fixed task set for multicore architecture. General program is hard to completely parallel for multicore architecture, and tasks are distributed to low frequency cores if can reduce total energy consumption of finishing application depend on real situation.

- **Linux Power Modes**

Linux kernel of general portable device only provides basic function of DFS for user, although users can command the system to manage core or working frequency, but this method don't include the function of power management, for example, energy saving and adjusting performance, user can reach the target by through their trying. Linux provides a ACPI [4] on x86 to process DFS of power management. Thus, in order to compare with bounded-power of BPM-DFS algorithm, this research includes x86 Linux as one of the basic compared object. Linux power management in x86 base on ACPI[4][13][14] provided relational libraries to control system, user through API of Linux power management for special target to choose scheme and setting system file. ACPI will consider the setting scheme to adjust the working frequency, and ACPI provides five frequency adjustment schemes to set.

ACPI provides frequency adjustment scheme are as follows [15]:

1) cpufreq_performance

This frequency adjustment scheme is performance oriented, CPU will executes tasks by most highest working frequency when system is set to this scheme, normal mode and DFS Linux mode have same working frequency.

2) cpufreq_powersave

This frequency adjustment scheme will save energy as far as possible; CPU will execute tasks by lowest working frequency, in order to reach the lowest energy consumption.

3) cpufreq_ondemand

This scheme will schedule each workload of CPU to dynamically adjust working frequency. But this scheme is not for special target of energy management.

4) cpufreq_conservative

It is similar to adjustment of cpufreq_ondemand mode, but this scheme will step-by-step adjust working frequency.

5) cpufreq_userspace

This scheme doesn't provide initiative adjustment of working frequency, user must command to system files to adjust working frequency of each core.

As above description, we can find that only one scheme is relational to energy saving, and this scheme is useless for requirement of user's bounded-power. Android or Linux, their scheduler is not power oriented like BPM-DFS algorithm, their scheduler don't consider energy consumption in application scheduling.

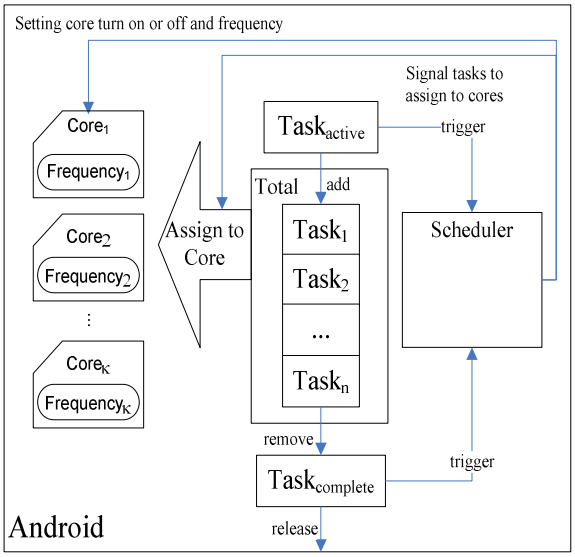


Figure 1. Bounded-Power Multicore DFS Scheduling Mechanism

III. BOUNDED-POWER MULTICORE DYNAMIC FREQUENCY SCHEDULING (BPM-DFS)

This research proposes a Bounded-Power Multicore DFS Scheduling mechanism, the executing method of the mechanism is like Figure 1. The task set of the execution base on Android of multicore operating system is T, $T = \{ T_1, T_2, \ldots, T_n \}$. And the usable core number of system is 喈, the configuration of core alive status is $cfs = \{ P_1, \ldots, P_\kappa \}$. We use P_i to present the alive status of core i, it presents cores are off when P_i is 0, and it presents cores are on when P_i is 1.

And the set of all cores alive status is $CFS = \{ cfs_1, \ldots, cfs_{2^{(\kappa-1)}} \}$, cfs_1 presents (1, 0, ..., 0), it tells that first core is turned on and other cores are turned off. $cfs_{2^{(\kappa-1)}}$ presents (1,

1, ..., 1), it tells that all cores are turned on. And first core of cfs is permanently turned on, thus core alive configuration number of CFS are $2^{\kappa-1}$.

The available working frequency of core i is $F_i = \{ f_{ij} \mid 1 \leqq j \leqq m , f_{i1} < f_{i2} < ... < f_{im} \}$. f_1 is the lowest degree frequency, f_m is the highest degree frequency, so the working frequency of all of the cores is $Freq=\{F_1, ..., F_g, ... , F_\kappa\}$, and $F_g \in \{f_1, ... , f_m \mid f_1 < ... < f_m \}$.

User can decide the PowerBudget, it presents that the upper bound of the user wanted power consumption. We propose a algorithm, the scheduler will be activated after the tasks are added or finished, and the algorithm can decide a configuration, Config = (Power, cfs, Freq, TS_Cur, L, C), that conform the power consumption can reach the highest performance, Power is power consumption, cfs is core live status, Freq is working frequency, TS_Cur is the status of current tasks are dispatched to cores, L is workload, C is thread migration number. T_i.C is that the tasks are dispatched to cores, for example, task1 is dispatched to core2, so T_i.C is T_1.C = 2. And T_i.L is the workload of task, for example, the workload of task2 is 70, thus T_i.L is T_2.L = 70. The set of all tasks are dispatched to cores is also the set of T_i.C, the set is TS.

This paper propose a Bounded-Power Multicore DFS Scheduling Algorithm, its mechanism makes user to decide a system configuration in a custom PowerBudget, and the system can gain maximum performance at same time. In order to reach the target, this scheduling mechanism must satisfy follow conditions:

a) PowerBudget is the upper bound of user defined power consumption, Config is completed system configuration, it must find out system configuration Config when power consumption is lower than PowerBudget.

b) The algorithm will do adjustment of workload balance, core workload will be decrease as far as possible in this process, the performance of the cores which's workload is decreased will increase.

c) Thread migration present task according workload to move from some core to another core, it calls thread migration, algorithm will find the configuration of smaller thread migration number.

BPM-DFS algorithm is listed as Algorithm 1. When Task T_{issue} is active or completely, our scheduler will be trigger. At first, the scheduler will estimate λ, ϖ, β and TS_Cur. And then it will assume first core is online, other cores is offline, and all cores use minimum working frequency. And then the scheduler will add T_{issue} into T_{total} or remove Tissue from T_{total}.

Through by CoreTaskMapping(), the scheduler will redistribute tasks of Ttotal to cores of cfs, and return a set of result. It can get predictive power consumption from Predictive Power Model. And then the scheduler conclude above results into one configuration, and add into configuration set if its predictive power consumption is smaller than power budget.

Algorithm 1: Bounded-Power Multicore DFS Scheduling Algorithm.

Bounded-Power Multicore DFS Scheduling Algorithm(BPM-DFS)

Input : $T = \{T_1, \dots , T_n\}$, PowerBudget, λ, ω, β,TS_Cur
Output : $Config_{out}$

if(T_{issue} is Complete) then
  $T_{total} := \{T\} - \{T_{issue}\}$;
else
  $T_{total} := \{T\} + \{T_{issue}\}$;
end

for ∀cfs_i ∈CFS

  for ∀freq∈Freq := { $F_1, F_2,\dots,F_g,\dots, F_\kappa$} ,
    $F_g \in \{ f_1, \dots, f_m \mid f_1 < \dots < f_m \}$, $g \in \{ 1,\dots, \kappa \}$

  do
    {TS_Tmp, L_{tmp}, C_{tmp}} :=
      CoreTaskMapping(T_{total}, cfs_i, TS_Cur);

    $Power_{tmp} = (\sum_{h=1}^{\kappa} P_h \times \lambda \times F_h \times \varpi \times Load(h, TS_{tmp})) + \beta$

    $Config_{tmp}$:= ($Power_{tmp}$ cfs_i, freq, TS_Tmp, L_{tmp}, C_{tmp}) ;

    if Powertmp PowerBudget
      ConfigSet := ConfigSet + $Config_{tmp}$;
    end

  end

end

if(ConfigSet = ∅) then

    ConfigSet :={ ($P_{tmp} = (\sum_{h=1}^{\kappa} P_h \times \lambda \times F_h \times \varpi \times Load(h, TS_{tmp})) + \beta$
      , $Core_{base}$, $freq_{base}$, CoreTaskMapping(T_{total}, cfs_{base}, TS_Cur)) }
end

for ∀ $Config_i$ ∈ ConfigSet
  L_{min} := min(L_{min}, Load($Config_i$))

σ:=sortByChange(ConfigSet);

CandidateConfigSet := ∅;

for ∀Configi ∈ σ from head to tail do

    if(ConfLoad($Config_i$) = L_{min}) then
      if(ConfC($Config_i$) = ConfC($Config_1$))
      CandidateConfigSet := CandidateConfigSet + $Config_i$;
    end

end

σ:=sortByAvgFreq(CandidateConfigSet);

$Config_{out}$:= $Config_1 \in \sigma$;

return $Config_{out}$

If no proper configuration smaller than power budget, the scheduler will add baseline configuration into configuration set. And then it will finds out the configuration that has better workload balance and minimum Thread Migration number. And return the configuration that their average frequency has maximum value.

CoreTaskMapping's algorithm is as shown in Algorithm 2. Tasks will be distributed to cores by Worst-Fit Bin Packing and First-Fit Bin Packing.

The scheduler will compare TS_Worst-Fit and TS_First-Fit with TS_Cur to get Thread Migration number. Through TS_Worst-Fit and TS_First-Fit, we can get workload balance value. Comparing workload balance value of TS_Worst-Fit with workload balance value of TS_First-Fit to decide output results.

Algorithm 2: CoreTaskMapping Algorithm.

```
CoreTaskMapping(CTM)
Input  : T_total, cfs_tmp, TS_Cur
Output : { TS_Tmp, L_tmp , C_tmp}

TS_Worst-Fit := Worst-FitBinPacking( T_total, cfs_tmp );

TS_First-Fit := First-FitBinPacking( T_total, cfs_tmp );

for i = 1 to n do
    if( T_i.C – t_i.C ≠0 where T_i ∈TS_Cur , t_i ∈TS_WorstFit) then
        C_Worst-Fit := C_Worst-Fit + 1;
    if( T_i.C – t_i.C ≠0 where T_i ∈TS_Cur , t_i ∈TS_FirstFit) then
        C_First-Fit := C_First-Fit + 1;
end

for i=1 to κ do
    L_Worse-Fit := max( Load( i, TS_Worst-Fit ), L_Worst-Fit );
    L_First-Fit := max( Load( i, TS_First-Fit ), L_First-Fit );
end

if( L_Worst-Fit< L_First-Fit ) then
    { TS_Tmp, L_tmp, C_tmp } := { TS_Worst-Fit, L_Worst-Fit, C_Worst-Fit };
else
    { TS_Tmp, L_tmp, C_tmp } := {TS_First-Fit, L_First-Fit, C_First-Fit };
end
return { TS_Tmp, L_tmp, C_tmp }
```

The meaning of utilized functions in these two algorithms is as below.

- Load(i, TS) will return workload of core i in TS..
- Worst-FitBinPacking(T, cfs_{tmp}) will return TS_Worst-Fit, T will be distributed to cores of cfs_{tmp} by Worst-Fit Bin Packing[16]
- First-FitBinPacking(T, cfs_{tmp}) will return TS_First-Fit, T will be distributed to cores of cfs_{tmp} by First-Fit Bin Packing[16].
- ConfLoad($Config_i$) will return workload balance value of $Config_i$.
- ConfC($Config_i$) will return Thread Migration number of $Config_i$.
- offlineComputing() will pre-calculate λ, ω, β and TS_Cur.
- CoreTaskMapping(T, cfs_i, TS_Cur) will distribute tasks of T to cores of cfs_i and get a new TS_tmp, comparing with TS_Cur to estimate Thread Migration number[7], at last, return TS_Tmp, L_{tmp} of workload balance value and C_{tmp} of Thread Migration number.
- sortByChange(ConfigSet) will sort Config of ConfigSet by Thread Migration number in ascending order.
- sortByAvgFreq(CandidataConfigSet) will sort Config of CandidataConfigSet by average frequency in descending order.

IV. The Experimental Results of BPM-DFS, SCA-ICA, and Linux Scheduling

This section will discuss the real scheduling results of proposed BPM-DFS, Linux Power Mode, and SCA-ICA. The power budgets of BPM-DFS are configured as 71 and 91, to demonstrate the low power and high performance scheduling mode. The adopted Linux power modes are "Linux Powersave" and "Linux Performance". The evaluated workload sequence is as shown in Section 4. The CPU of the experimental system is Intel Core 2 Quad Q6600, which consists of four cores. These four cores can be turn-on/shutdown, the corresponding working frequency can be configured individually. The available working frequency is 1603 MHz, 1870 MHz, 2136 MHz, and 2403MHz. The operating system of the evaluated computer system is Android 2.3.5 with the corresponding Linux kernel version 2.6.39. The power consumption of whole computer system is measured by a recordable digital power meter. The whole experiments are divided into two parts. The first experiment evaluates the scheduling results of the workload sequence that are introduced in Section 4, by using BPM-DFS, Linux Power Mode, and SCA-ICA. The background tasks are only the required daemons and services of Android and Linux kernel. The second experiment is the same as first experiment, except add a heavy load task that is executed in the background. The adopted heavy loading tasks are from SPEC2000 (64.gzip, 176.gcc, 181.gcc and 300.twolf) and SPECJVM98 (Raytrace).

A. The Scheduling Results Given Task Sequence by Three Scheduling Mechanisms.

In this section, we will discuss the mechanism of BPM-DFS by using power budget of 73, 91, 93, Linux Powersave Mode, Linux Performance Mode, and SCA-ICA. In the Figure 2, when power budget is 73, BPM-DFS algorithm compares with Linux Powersave mode can reduce 5% power consumption. When power budget is 91, BPM-DFS algorithm compares with SCA-ICA algorithm can reduce 1.3% power consumption. When power budget is 93, BPM-DFS algorithm compares with Linux performance mode can reduce 1% power consumption. Thus we can through power budget to reduce power consumption.

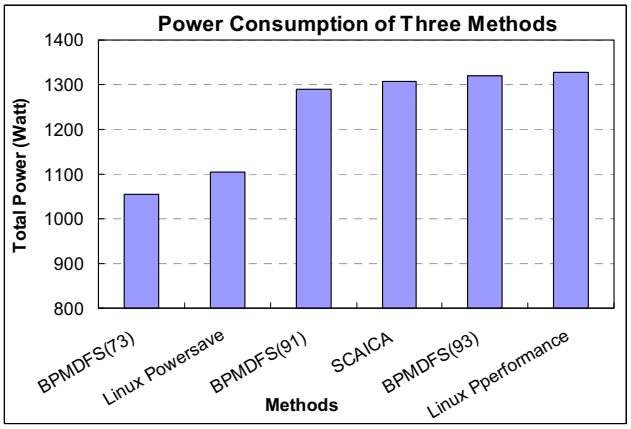


Figure 2. The results of BPM-DFS, SCA-ICA and Linux Scheduling.

B. The Scheduling Results Given Task Sequence by Three Scheduling Mechanisms witt Heavy Loading Backround Task.

In this section, we will discuss the mechanism of BPM-DFS by using power budget of 71, 91 and 95., Linux Powersave Mode, Linux Performance Mode, and SCA-ICA, as shown in Figure 3 and Figure 4. The adopted heavy loading tasks are from SPEC2000 (64.gzip, 176.gcc, 181.gcc and 300.twolf) and SPECJVM98 (Raytrace). As Figure 5, comparing with Linux Powersave mode in the lowest working frequency, BPM-DFS algorithm of power budget is lower than 71 that will reduce power consumption. But other algorithm's power consumption is more than power consumption of Linux Powersave mode. If we compare with Linux performance mode such as Figure 6, BPM-DFS algorithm of power budget is lower than 71 that still reduce the largest power consumption.

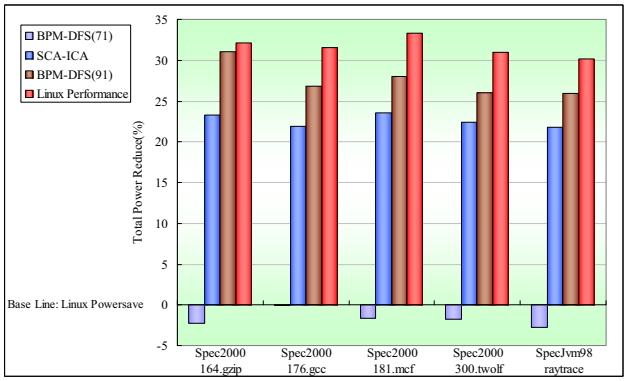


Figure 3. The total power reducing in Linux Powersave Mode.

The case of Raytrace is illustrated in Figure 5 and Figure 6. The Raytrace task is executed in background. The results of Linux performance mode and BPM-DFS power budget 91 are as Figure 7, x axis is step number of task sequence, y axis is power consumption. We can find out the power consumption of two methods is similar at each step. The total power consumption is as Figure 6(a), x axis is two methods, y axis is total power consumption, and BPM-DFS reduces 3.2% power consumption. Performance is as Figure 6(b), x axis is two methods, y axis is timing, and BPM-DFS increases 5% performance.

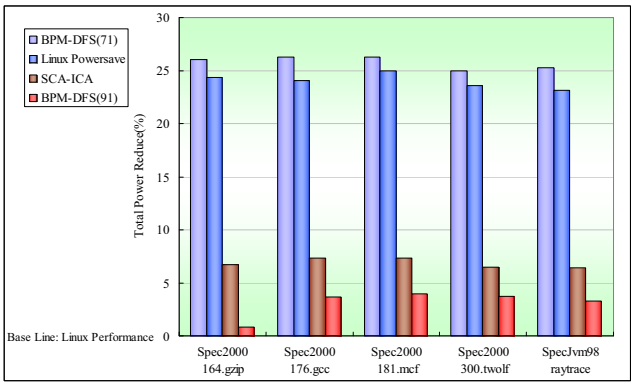


Figure 4. The total power reducing in Linux Performance Mode.

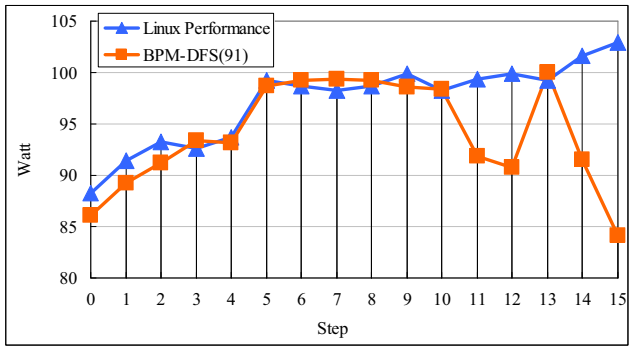
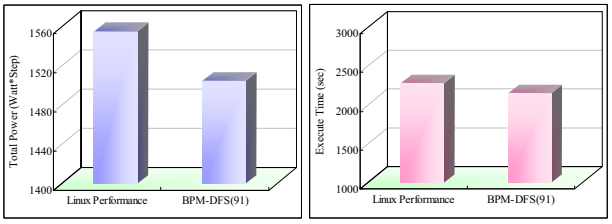


Figure 5. The power consumptions of Linux performance mode and BPM-DFS (91) with Raytrace.



(a) Total power consumption (b) Execution timing

Figure 6. The comparison of total power consumption and execution timing of Linux performance mode and BPM-DFS (91) with Raytrace.

Through the experiment, we can find out that BPM-DFS algorithm not only reduce power consumption but also improve performance in some situation. As Figure 6, when

power budget is 91, comparing with Linux Performance mode, BPM-DFS algorithm can reduce 3.3% power consumption and improve 6% performance.

V. CONCLUSION

Energy management or reducing energy consumption in many field gradually become the most important issue and be discussed and researched. This research proposes a new mechanism called Bounded-Power Multicore Dynamic Frequency Scaling Scheduling Algorithm, simply called BPM-DFS algorithm. The BPM-DFS algorithm will through core alive mechanism, dynamic adjust working frequency and adjust core/task assignment, and transform theoretical workload to real workload for Predictive Power Model to estimate predictive power consumption. The scheduler will find out the configuration that predictive power consumption under the power budget. The BPM-DFS algorithm will output proper configuration and setup the real system by results of the configuration. Through adjustment of power budget, comparing with Linux Performance mode, it can save 25% power consumption. Comparing with SCA-ICA algorithm, it can save 21% power consumption. Comparing with Linux Powersave mode, it can save 3% power consumption. So BPM-DFS algorithm really can reach the target of energy management and energy saving.

ACKNOWLEDGMENT

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] C. Xian and Y.-H. Lu, “Dynamic voltage scaling for multitasking realtime systems with uncertain execution time,” in ACM Great Lakes Symp. VLSI (GLSVLSI), 2006, pp. 392–397.

[2] E. Talpes and D. Marculescu, “Toward a multiple clock/voltage island design style for power-aware processors,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 13, no. 5, pp. 591–603, 2005.

[3] G. Magklis, G. Semeraro, D. H. Albonesi, S. G. Dropsho, S. Dwarkadas, and M. L. Scott, “Dynamic frequency and voltage scaling for a multipleclock-domain microprocessor,” IEEE Micro, vol. 23, no. 6, pp. 62–68, 2003.

[4] Advanced Configuration and Power Interface,[Available Online]:http://www.acpi.info/

[5] Goglin, B., LaBRI, INRIA, Talence, France, Furmento, N., “Enabling high-performance memory migration for multithreaded applications on LINUX” , Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, May 2009, pp. 23-29.

[6] Patrick Brady, “2008 Google I/O Anatomy & Physiology of an Android”, [Available Online]: http://sites.google.com/site/io/anatomy--physiology-of-an-android.

[7] Haisang Wu, Binoy Ravindran, E. Douglas Jenseny, and Peng Li, “CPU Scheduling for Statistically-Assured Real-Time Performance and Improved Energy Efficiency” , CODES+ISSS’04, September 8–10, 2004, Stockholm, Sweden.

[8] J. R. Lorch and A. J. Smith, “Improving dynamic voltage scaling algorithms with PACE,” in SIGMETRICS/Performance, 2001, pp. 50–61.

[9] Wan Yeon Lee, “Energy-saving DVFS Scheduling of Multiple Periodic Real-time Tasks on Multicore Processors” , DS-RT '09 Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, pp. 216–223.

[10] W. Horn. “Some simple scheduling algorithms.” , Naval Research Logistics Quarterly, 21:177–185, 1974.

[11] P. Pillai and K. G. Shin., “Real-time dynamic voltage scaling for low-power embedded operating systems.” , In ACM SOSP, pages 89–102, 2001.

[12] Don Domingo, “Power Management Guide”, Linux Whitepapers [Available Online]:https://www.linux.com/learn/whitepapers/

[13] IBM Developer Works, “Pwer Management” , [Available Online] : http://www.ibm.com/developerworks/cn/linux/l-power

[14] Patrick Mochel, “Linux Kernel Power Management” , Proceedings of the Linux Symposium, July 23th–26th, 2003, pp. 326-339.

[15] Arch Linux, “CPU Frequency Scaling” , [Available Online] : https://wiki.archlinux.org/index.php/CPU_Frequency_Scaling

[16] J. E. G. Coffman, M. R. Garey, and D. S. Johnson, “Approximation algorithms for NP-hard problems.” ,Boston, MA, USA: D. Hochbaum, PWS Publishing Co., 1996, ch. 2. Approximation algorithms for bin packing: a survey, pp. 46–93.

[17] T. Martin. Balancing Batteries, “Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing.” PhD thesis ,Carnegie Mellon ,University, August 1999.

[18] J. Wang, B. Ravindran, and T. Martin., ”A power aware best-effort real-time task scheduling algorithm” , In IEEE WSTFES/ISORC Workshop, pages 21–28, May 2003.

[19] SPEC, "SPEC CPU2000”, http://www.spec.org/cpu2000/.

[20] SPEC, "SPEC JVM98”, http://www.spec.org/jvm98/.