2009 International Conference on Parallel and Distributed Computing, Applications and Technologies

A Task Selection Based Power-aware Scheduling Algorithm for Applying DVS

Yuichiro MORI
Department of Systems and Social Informatics, Graduate School of Information Science, Nagoya University, Nagoya, JAPAN
ymori@watanabe.ss.is.nagoya-u.ac.jp

Koichi ASAKURA
Department of Information Systems, School of Informatics, Daido University, Nagoya, JAPAN
asakura@daido-it.ac.jp

Toyohide WATANABE
Department of Systems and Social Informatics, Graduate School of Information Science, Nagoya University, Nagoya, JAPAN
watanabe@is.nagoya-u.ac.jp

Abstract—**Recently, power consumption of server computers is one of the most important topics. Although DVS (Dynamic Voltage Scaling) can reduce power consumption, this method is only used at idle time or low load computation time, generally. In this paper, we propose a task scheduling algorithm for reducing power consumption especially at high load computation time. DVS is applied to tasks that are not in critical path, which can reduce more power consumption by introducing a task selection mechanism without increasing the makespan of task graphs. Experimental results show that our algorithm can reduce about 9.1% and 12.8% of power consumption on 4 and 8 processors respectively on average in comparison with the existing method.**

Index Terms—**task scheduling; dynamic voltage scaling**

I. Introduction

Recently, power consumption of server computers is one of the most important topics[1,2]. Especially, to reduce power consumption of processors is very essential since the processor is the most power-consumed part in servers. Thus, in order to reduce power consumption of servers, it is effective to reduce power consumption of processors.

Dynamic Voltage Scaling (DVS) is one of the methods for reducing power consumption of processors[3]. In DVS, combinations of an operating voltage and an operating frequency, which are called P-States, are changed dynamically at run time. Coolin'n'Quiet and PowerNow! by AMD, and SpeedStep by Intel are examples of implementation for DVS. Generally, DVS is adopted for reducing power consumption of processors at idle time. Namely, DVS does not pay any attention to power consumption of heavy-load processors.

In this paper, we propose a task scheduling algorithm to reduce power consumption of multi-processor servers. In our algorithm, we apply DVS to tasks of a scheduling result generated by traditional algorithms which do not take power consumption into account. We consider application of DVS at heavy load time as well as at idle time without increasing the makespan of the task graph.

The rest of this paper is organized as follows. Section 2 describes preliminaries of a task scheduling algorithm and DVS. Section 3 describes our algorithm. Section 4 describes related work. Section 5 shows experimental results by simulation. Section 6 concludes this paper and mentions our future work.

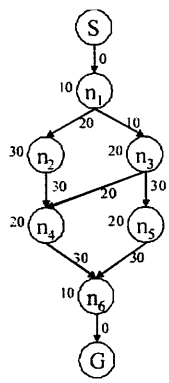

Fig. 1. A taskgraph

II. Preliminaries

A. Task scheduling

A task scheduling for parallel systems is the process in which each task in a task graph is allocated to a processor. A task graph is represented as a directed acyclic graph. Figure 1 shows an example of a task graph. Each node represents a task and each edge represents a partial order among the tasks. A computation cost of the task n_i is represented as $comp(n_i)$, and a communication cost between tasks n_i and n_j is represented as $comm(n_i, n_j)$.

A task scheduling process can be divided into two aspects: spatial and temporal assignments. The former is described as allocation of each task to a processor. The latter is described as assignment of a start time to each task. A result of a task scheduling is called a schedule. Figure 2 depicts a schedule in Gantt chart representation. In this figure, each scheduled task is drawn as a rectangle. The horizontal axis represents time. Time scale is represented as unit time. Each communication between two processors is denoted as an arrow.

B. DVS

DVS is a technology in which combinations of an operating voltage and an operating frequency of a processor can be changed dynamically at run time. These combinations are

978-0-7695-3914-0/09 $26.00 © 2009 IEEE
DOI 10.1109/PDCAT.2009.36

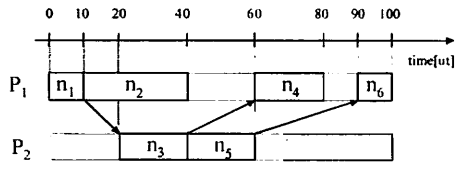

Fig. 2. A schedule

TABLE I
SPECIFICATION OF A COMPUTER

parts	specification
Motherboard	Gigabyte GA-K8 VM800M
CPU	AMD Athlon64 3000+
Clock frequency	2.00GHz
Number of P-States	3
RAM	1.00GB
OS	Windows XP Professional SP3

called P-States. Recently, many processors equip this technology and processors generally have several P-States discretely. By DVS, the processing speed as well as power consumption can be changed dynamically by transiting to the other P-State because the processing speed is proportional to the operating frequency and power consumption is proportional to the product of the operating frequency and square of the operating voltage in CMOS circuits. Namely, DVS can reduce power consumption by slowing down the processing speed. Thus, there is a trade-off problem between power consumption and the processing speed. For P-States s_1 and s_2, the operating frequencies f_{s_1} and f_{s_2}, and elapsed times for task execution T_{s_1} and T_{s_2} have the following relationship:

$$\frac{f_{s_1}}{f_{s_2}} = \frac{T_{s_2}}{T_{s_1}}. \quad (1)$$

In order to make an effect of DVS clear, we conduct a preliminary experiment. We measure power consumption of a computer described in Table 1. Since this processor has three P-States, we measure power consumption of the processor for each P-State. The experimental result is shown in Table 2. This experimental result makes it clear that DVS can reduce power consumption at both idle time and heavy loaded time by stepping down the operating voltage and the operating frequency although the processing speed becomes slower.

III. POWER-AWARE SCHEDULING ALGORITHM

In this section, an algorithm for reducing power consumption by utilizing DVS to parallel programs without increasing the makespan of the program is described. Our algorithm focuses on a schedule generated from a task graph represented as directed acyclic graph.

A. Overview of the algorithm

The input of our algorithm is a schedule which is produced by traditional non-power-aware scheduling algorithms such as ETF. Application of DVS to tasks makes the execution time of tasks longer. Thus, in order not to increase the makespan of the task graph. DVS must be applied to tasks whose completion time modified by DVS does never affect the start times of other tasks. For this purpose, we focus on idle time slots next to tasks.

If there is a long time slot enough to the application of DVS to the task, power consumption of processors can be decreased without incrementing the makespan of the task graph. In our algorithm, an idle time slot next to a task is called slack-time of the task. We give a detailed calculation algorithm of the slack-time in Section 3.2.

For application of DVS, the lower-frequency P-State has to be selected as long as the task does not influence the start times of other tasks. For an example, we consider consumed energies in each P-State described in Table 2 for the 50 unit-time-length task. It takes 100 unit times to execute the task in the lowest-frequency P-State. Thus, we compare the energies E from time 0 to 100[ut]. Figure 3 shows task execution in each P-State. In each P-State, power consumption is calculated as follows:

$$\begin{aligned} E_1 &= 100_{[W]} \cdot 50_{[ut]} + 59_{[W]} \cdot 50_{[ut]} &&= 7950_{[W \cdot ut]}, \\ E_2 &= 87_{[W]} \cdot 55_{[ut]} + 59_{[W]} \cdot 45_{[ut]} &&= 7440_{[W \cdot ut]}, \\ E_3 &= 61_{[W]} \cdot 100_{[ut]} &&= 6100_{[W \cdot ut]}. \end{aligned}$$

From this result, it is clear that we have to select the P-State as lower frequency as possible. Namely, we have to select the task, to which the lowest-frequency P-State can be applied, from a candidate task set. We give a task selection scheme in Section 3.4.

TABLE II
AN EFFECT OF DVS

P-States of the processor			Power consumption[W]	
P-State	Frequency [MHz]	Voltage [V]	at idle time	at heavy loaded
1	2000	1.50	87	100
2	1800	1.40	78	87
3	1000	1.10	59	61

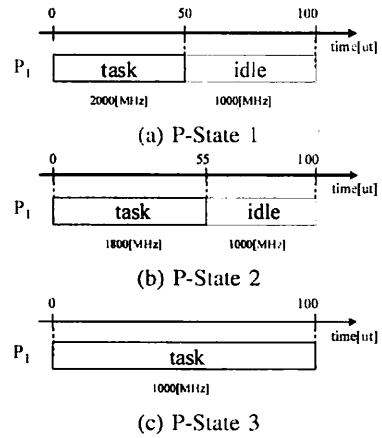

Fig. 3. Examples of P-States

Figure 4 shows our algorithm. In this algorithm, earliest start time and earliest completion time of the task n in the processor p is represented as $est(n,p)$ and $ect(n,p)$, respectively[4]. Generation of candidate task list for DVS application is shown in lines 5-9 and selection of the most effective task is shown in lines 10-15.

B. Computation of slack-time

For calculating the slack-time of the task n, $slacktime(n)$, the relationship between the task n and the successor tasks is taken into account. Figure 5 shows the computation method of slack-time of the task n. In Figure 5, $slacktime(n'_{s0})$ is assumed to be zero. In order not to increase makespan, we do not increase est of other processors. If the task n is allocated to a processor p and a successive task n_{s1} is allocated to the other processor p_{s1}, slack-time between tasks n and n_{s1} represented as $slacktime(n, n_{s1})$ is defined as follows:

$$slacktime(n, n_{s1}) = est(n_{s1}, p_{s1}) - ect(n, p) - comm(n, n_{s1}). \quad (2)$$

If a successive task n_{s0} is allocated to the same processor of the task n, $slacktime(n, n_{s0})$ is defined as follows by using $slacktime(n_{s0})$:

$$slacktime(n, n_{s0}) = est(n_{s0}, p) - ect(n, p) + slacktime(n_{s0}). \quad (3)$$

From these results, slack-time of the task n is calculated as follows:

$$slacktime(n) = \min_{n_s \in succ(n)} slacktime(n, n_s). \quad (4)$$

If there are no successive tasks, slack-time is defined as zero. If increase in execution time of a task by DVS is within the slack-time, the makespan of the task graph schedules is not changed. Therefore, the algorithm can reduce power consumption more effectively.

C. Generation of candidate task list for DVS application

Since calculation of $slacktime(n)$ requires the slack-time for successive task in the same processor, calculation of slack-time is performed from the tail task to the head task for each processor. If slack-time of the task n is positive, the task n is added to a candidate task list $nlist$. Next, a preceding task of the task n is focused on. If slack-time of the preceding task of n is also positive, the preceding task of n is added to $nlist$. Next, the task n is substituted for the preceding task of n. In a similar way, the preceding task is added to $nlist$ if the slack-time of the preceding task is positive. This step is repeated until there are no preceding tasks or slack-time of the preceding task becomes zero (lines 5-9).

Require: schedule: a scheduling result by non-power-aware scheduling algorithm
Ensure: schedule: a scheduling result
1: **begin**
2: **for** each processor p **do**
3: $n \leftarrow$ the tail task of p
4: **while** ($n \neq$ head task of p) **do**
5: $nlist \leftarrow \phi$.
6: **while** ($slacktime(n) > 0$) **do**
7: add n to $nlist$.
8: $n \leftarrow$ a preceding task of n.
9: **end while**
10: **while** ($nlist \neq \phi \vee$ all tasks in $nlist$ cannot be applied DVS) **do**
11: $n_{target} \leftarrow argmax(n \text{ in } nlist)\ f(n)$, where $f(n)$ is the amount of power reduction for n.
12: apply DVS to n_{target}.
13: adjust est and compute slack-time for all tasks of p.
14: remove n_{target} from $nlist$.
15: **end while**
16: $n \leftarrow$ a preceding task of n.
17: **end while**
18: **end for**
19: **end**

Fig. 4. Our algorithm

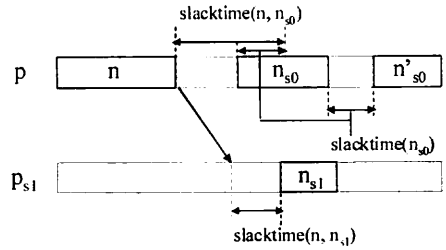

Fig. 5. A computation method of slacktime

D. Selection of task for application of DVS

In this phase, the task is selected for application of DVS from the candidate task list. First, the amount of power consumption is calculated on a P-State which can reduce the most power consumption without exceeding slack-time of each task in $nlist$. A task of the largest amount of power consumption to reduce in $nlist$ is selected. The selected task n_{target} is applied by DVS on the best P-State. Next, earliest start time, slack-time, and power consumption of every task in $nlist$ is updated because these parameter are changed by applying DVS. Then, task selection is activated repeatedly. This procedure is performed until no task is selected for application of DVS (lines 10-15).

E. Discussion

In this section, we explain that our algorithm is more effective than Kimura's algorithm[5]. A schedule is generated by applying Kimura's algorithm on processors described in Table 1 to a schedule shown in Figure 2. The task which is applied by DVS is depicted as a gray rectangle. In Kimura's algorithm, because interprocessor communication costs are not considered, we improve the algorithm so that interprocessor communication costs can be dealt with.

In Kimura's algorithm, if slack-time of a task is positive and slack-time of a task next to the task is also positive, these tasks are applied by DVS on average. That is, DVS is applied to all tasks in connective tasks. However, because P-State of processors are various operating frequency and operating voltage, the nonbiased DVS approach does not always make great contributions to reducing the amount of power consumption. In addition, because operating frequency and operating voltage of P-State is discrete, that of P-State cannot set arbitrary value. Therefore, as shown in Figure 6(a), there is a case in which slack-time leaves after application of Kimura's algorithm.

We reduce as longer slack-time as possible after our algorithm is applied by introducing the features of task selection in order to reduce more the amount of power consumption than Kimura's algorithm. As described in Section 3.4, in our algorithm, if tasks, which each of them has slack-time, are connective, a task of the most amount of power consumption to reduce in a task list is selected and DVS is applied to the task. If tasks which can applied by DVS leave in the task list after DVS is applied to the target task, the phase are repeated until there are no tasks in the candidate task list or DVS cannot be applied to any tasks in the task list. Figure 6(b) shows a result of applying our proposal algorithm on processors described in Table 1 to a schedule shown in Figure 2. In Figure 6(b), task n_2 is processed at lower operating frequency than that in Figure 6(a) and task n_4 is processed at higher operating frequency than that in Figure 6(a). When the amount of power consumption to surrounded part of Figure 6(a) and that of Figure 6(b) are calculated on processors described in Table 1, the former is 6,770[W·ut] and the latter is 5,270[W·ut]. The result shows that our algorithm is more effective than Kimura's algorithm.

IV. Related work

There are some researches in which the amount of power consumption is reduced by applying DVS[5-8]. Chen et al. have proposed a power-aware scheduling algorithm[6]. Input of the algorithm is a schedule generated from tree task graphs. This algorithm can reduce power consumption without increasing the makespan by applying DVS to tasks which are not in the critical path. However, this method deals with only tree task graphs and does not consider interprocessor communication costs. Our proposed algorithm deals with directed acyclic graph and we consider interprocessor communication cost based on Chen's method. In addition, our algorithm can

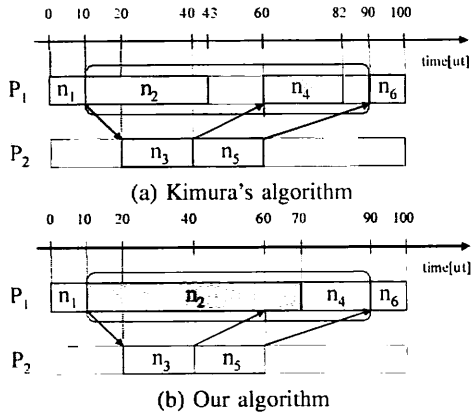

Fig. 6. Gantt charts for two methods

more power consumption by introducing task selection method described in Section 3.4.

Kimura et al. have proposed a power-aware scheduling algorithm based on Chen's method[5]. Kimura's method is improvement of Chen's method. This method can deal with arbitrary directed acyclic graphs as well as the tree task graphs. This method focuses on waiting time among tasks. Effectiveness of DVS is proved accurately by experiments in real machines. However, there is a problem that this method also does not consider interprocessor communication costs. Therefore, this method cannot assure proper power-aware scheduling on condition that communication costs are not zero for interprocessor communication. Our method can assure a proper power-aware scheduling even if interprocessor communication costs are not zero. Additionally, our algorithm can achieve good power consumption on task graphs having various CCRs.

Varatkar et al. have proposed a task scheduling algorithm based on a power consuming model according to communications traffic in interprocessor communication[7]. In this method, firstly a allowable deadline of makespan is defined preliminary. Secondly, interprocessor-communication-minimized task scheduling is performed so that the makespan does not exceed the deadline in order to reduce the amount of power consumption. However, this method focuses on only reducing interprocessor communications, and applying method of DVS is not discussed in detail. In addition, this method does not consider interprocessor communication time generated from interprocessor communication cost. Our approach deals with application of DVS mainly in order to reduce more power consumption and consider interprocessor communication volume and interprocessor communication time.

Hotta et al. have proposed an algorithm in which after a target program is divided into several fragments, each fragment is processed at proper operating frequency, respectively[8]. In this method, profiling information of programs for each P-State must be achieved. Then, processing time and power consumption of each fragment of the application are obtained

and the optimal operating frequency is decided. However, this method requires preprocessing because data of each fragment of the application must be obtained by processing at various operating frequencies.

V. Evaluation

We evaluate how effective our algorithm is to reduce the amount of power consumption by simulation. As input, our algorithm deals with schedules by ETF algorithm.

We use Standard Task Graph set (STG) for generating examples of task graphs for the experiments[9]. For experiments, we use 180 task graphs with 50 tasks and also 180 task graphs with 100 tasks in STG. Communication cost is assigned for each task graph according to Communication to Computation Ratio (CCR)[4]. The CCR is proposed for an index that represents properties of a task graph. The CCR is defined as follows:

Let task graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, w, c)$.

$$CCR(\boldsymbol{G}) = \frac{\sum_{e \in \boldsymbol{E}} c(e)}{\sum_{n \in \boldsymbol{V}} w(n)}. \tag{5}$$

For each task graph in STG, we generate three task graphs whose CCRs are about 0.1, 1 and 10, which means the task graphs have low, middle and high frequency of communication, respectively[4]. In addition, for each task graph, 10 kinds of communication cost are generated randomly. Thus, in total, we use 21600 task graphs for experiments.

For the target processor, we assume the AMD Athlon64 3000+ processor described in Table 1, and the target machine assumes to have 4 or 8 processors.

A. Power consumption model

A relationship of power consumption P, an operating frequency f and an operating voltage V is represented as follows:

$$P \propto f \cdot V^2 \tag{6}$$

From the results of Table 2 and the equation (6), power consumption model is defined in this section. When we define power consumption except processor as $x_{[W]}$, there is an equation described as follows:

$$\frac{P}{f \cdot V^2} = \frac{59 - x}{1000 \cdot 1.1^2} = \frac{87 - x}{2000 \cdot 1.5^2} \tag{7}$$

From the equation (7), power consumption except processors at idle time is calculated and this value is about 49[W]. We also assume that power consumption ratio of a processor is 1 : 0.75 : 0.25 at 2000MHz, 1800MHz and 1000MHz, respectively. From this consideration, we can define the power consumption model of the processor in Table 3. In Table 3, the item of "Idle" represents power consumption at the idle state. The item of "Execution" represents power consumption of the state of processing a task.

In general, there is an overhead in changing among P-States in DVS. The overhead is defined as some dozens of micro seconds generally[8]. For example, the overhead is 10[μs] in Intel Speed Step. This overhead is less than 1% since average computation costs of task graphs of STG are 3.1[ms]. Therefore, in these experiments, the overhead in changing among P-States can be ignored.

TABLE III
POWER CONSUMPTION MODEL OF PROCESSOR

Frequency[MHz]	Voltage[V]	Idle[W]	Execution[W]
2000	1.50	40	52
1800	1.40	30	39
1000	1.10	10	13

B. Experimental results

Based on the above power consumption model, Table 4-6 show the result of applying our algorithm to the original schedule generated by ETF algorithm. The amount of power consumption is shown in Table 4 and Table 5. Percentages of reducing the amount of power consumption compared with that of power consumption before applying power-aware algorithm in average are shown in Table 6. Both the effects of Kimura's algorithm and our algorithm are shown.

From the experimental results, we can observe that our algorithm can reduce the more amount of power consumption than Kimura's algorithm in all cases.

Table 6 shows that our algorithm can also reduce the most amount of power consumption in task graphs which have high CCR. This is because waiting time of communication increases as communication costs increase. Although in the case of 8 processors shown in Table 6, our algorithm looks as if the amount of reduced power consumption does not depend on CCR, this is because percentages of the amount of power consumption at idle time in 8 processors are larger than that in 4 processors. Our algorithm is not applied to idle time.

In addition, when the number of processor is large, the percentages of reducing the amount of power consumption are large. When the number of processors is large, much communication occurs. This makes waiting time of communication increase. Therefore, we consider that this is because slack-time of each task increases as well as the case of task graphs which have high CCR.

VI. Conclusion

In this paper, we proposed a power-aware task scheduling algorithm by applying DVS which is electrical power saving technology of processors based on features of task selection. The input of our algorithm is a schedule generated from a task graph represented as directed acyclic graph by applying task scheduling algorithm such as ETF algorithm. The output is a schedule in which the amount of power consumption is reduced. In order not to spoil advantage of parallel processing, DVS is applied to tasks without increasing makespan of the task graph. From the experimental results by simulation, our algorithm can reduce about 9.1% and 12.8% of power consumption on 4 and 8 processors respectively on average.

In our experiments, the overhead in changing among P-States in DVS is regarded as very small and the effect assumes to be ignored. However, there is the case that the overhead

TABLE IV
EXPERIMENTAL RESULTS 1 (AMOUNT OF POWER CONSUMPTION)

Number of Tasks : 50

		The amount of power consumption [W · s]		
Processor	CCR	Original	Kimura's method	Our method
4	0.1	2271.8	2110.7	2016.8
	1	2287.2	2129.9	2042.8
	10	2550.1	2287.9	2191.8
8	0.1	2633.1	2349.6	2241.4
	1	2675.4	2397.7	2300.3
	10	3299.1	2946.7	2872.8

TABLE V
EXPERIMENTAL RESULTS 2 (AMOUNT OF POWER CONSUMPTION)

Number of Tasks : 100

		The amount of power consumption [W · s]		
Processor	CCR	Original	Kimura's method	Our method
4	0.1	4322.0	4105.1	3902.4
	1	4334.5	4106.9	3918.1
	10	4565.3	4173.1	3977.5
8	0.1	4807.6	4367.8	4114.8
	1	4846.8	4401.6	4172.5
	10	5476.5	4870.6	4664.8

TABLE VI
EXPERIMENTAL RESULTS (RATIO OF REDUCING POWER CONSUMPTION)

Number of Tasks		50		100	
Processor	CCR	Kimura's method	Our method	Kimura's method	Our method
4	0.1	7.1%	11.3%	4.8%	9.4%
	1	6.9%	10.8%	5.0%	9.3%
	10	10.3%	14.1%	9.0%	13.8%
8	0.1	11.3%	15.7%	8.9%	14.3%
	1	10.8%	14.7%	10.6%	16.5%
	10	10.8%	13.3%	11.2%	15.1%

cannot be ignored because granularity of tasks of a task graph is very small. Even if a schedule is generated from a task graph whose granularity of tasks is low, we should construct a power-aware algorithm considering the overhead.

In addition, experimental results are obtained by simulation. We do not verify the effect of our algorithm in real machine. In order to verify that our algorithm is effective in real machine, there is a method that DVS routine is inserted to ranges between MPI routine in real parallel processing program[10]. To evaluate in real machine by utilizing such method is also our future work.

REFERENCES

[1] L.A.Barroso: "The Price of Performance", *Queue*, Vol. 3, No. 7, pp. 48–53 (2005).

[2] X.Fan, W.D.Weber and L.A.Barroso: "Power Provisioning for a Warehouse-sized Computer", *Proc. of the 34th Annual Int'l. Symp. on Computer Architecture*, pp. 13–23 (2007).

[3] Y.Zhang, X.S.Hu and D.Z.Chen: "Task Scheduling and Voltage Selection for Energy Minimization", *Proc. of the 39th Conf. on Design Automation*, pp. 183–188 (2002).

[4] O.Sinnen: *"Task Scheduling for Parallel Systems"*, Wiley-Interscience (2007).

[5] H.Kimura, M.Sato, Y.Hotta, S.Matsuoka, T.Boku and D.Takahashi: "Empirical Study on Reducing Energy of Parallel Programs using Slack Reclamation by DVFS in a Power-scalable High Performance Cluster", *Proc. of 8th IEEE Int'l. Conf. on Cluster Computing (CLUSTER)*, CD-ROM, (2006)

[6] G.Chen, K.Malkowski, M.Kandemir and P.Raghavan: "Reducing Power with Performance Constraints for Parallel Sparse Applications", *Proc. of Workshop on High-Perfomance, Power-Aware Computing (IPDPS)*, pp. 231–250 (2005).

[7] G.Varatkar and R.Marculescu: "Communication-Aware Task Scheduling and Voltage Selection for Total Systems Energy Minimization": *Proc. of 2003 Int'l. Conf. on Computer-aided design (ICCAD)*, pp. 510–517 (2003).

[8] Y.Hotta, M.Sato, H.Kimura, S.Matsuoka, T.Boku and D.Takahashi: "Profile-based Optimization of Power Performance by using Dynamic voltage Scaling on a PC cluster", *Proc. of Workshop on High-Perfomance, Power-Aware Computing (IPDPS)*, CD-ROM, (2006).

[9] T.Tobita and H.Kasahara: "Performance Evaluation of Minimum Execution Time Multiprocessor Scheduling Algorithms Using Standard Task Graph Set", *Proc. of 2000 Int'l. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pp. 745–751 (2000).

[10] L.Choy, S.G.Petiton and M.Sato: "Toward Power-Aware Computing with Dynamic Voltage Scaling for Heterogeneous Platforms", *Proc. of 2007 Int'l. Conf. on Cluster Computing*, pp. 550–557 (2007).