Power Minimization Techniques on Distributed Real-Time Systems by Global and Local Slack Management

Shaoxiong Hua and Gang Qu
Department of Electrical & Computer Engineering
University of Maryland, College Park, MD 20742, USA
{shua, gangqu}@eng.umd.edu

Abstract— Recently, a static power management with parallelism (P-SPM) technique has been proposed to reduce the energy consumption of distributed systems to execute a set of real-time dependent tasks [7]. The authors claimed that the proposed P-SPM outperforms other known methods in energy reduction. However, how to take advantage of the local static slack for further energy optimization remains as an open problem.

In this paper, we propose the static power management with proportional distribution and parallelism scheme (PDP-SPM) that not only answers this open problem, but also exploits the parallelism. Simulations on task graphs derived for DSP applications and TGFF benchmark suite suggest that PDP-SPM achieves 64% energy saving over the system without power management, and 15% over the P-SPM scheme.

I. Introduction

With the advances in VLSI and communication technology, distributed real-time systems have emerged as a popular platform for applications such as multimedia applications, mobile computing and information appliances. For hard real-time systems, application deadlines must be met at all time. However, early completion (before the deadline) of the applications may not bring the systems extra benefits. In this case, we can trade this extra performance for other valuable system resources, for example, energy consumption. Low power (energy) consumption has become more and more important in the design of distributed real-time systems. Dynamic voltage scaling (DVS), which varies the processor speed and supply voltage according to the workloads at run-time, can achieve the highest possible energy efficiency for time-varying computational loads while meeting the deadline constraints [1]. Evidently, there are many recent works on the power management problem of distributed systems by using DVS [4, 5, 6, 7, 10].

In this paper, we consider distributed real-time systems executing a set of dependent tasks that are normally represented by a task graph. Our goal is to develop DVS schemes to reduce the energy consumption without missing any deadlines. Specifically, given the mapping of tasks to multiple processors, we propose a static power management (SPM) scheme that simultaneously considers both proportionally distributing the slacks among tasks and the degree of the parallelism presented in distributed systems. Extensive simulation shows that our proposed scheme outperforms all the known SPM algorithms such as P-SPM [7] and S-SPM [3] by using the worst case execution time (WCET). We discuss our algorithms on DVS systems that support either continuous speeds (voltages) or multiple discrete speeds (voltages).

The rest of the paper is organized as follows: Section II formulates the problem based on application model and DVS system model. The static scheduling algorithms have been discussed in Section III. Simulation results for different scheduling schemes to reduce energy consumption are given in Section IV. Finally, we conclude the paper in Section V.

II. Problem Formulation

We consider real-time distributed systems, which support DVS, executing frame based applications.

Application Model We assume that the distributed systems repeatedly executes frame based applications [5] that consist of a set of tasks that have to be completed before a common deadline. The applications in a frame can be modeled by a task graph $G = (V, E)$, where V is the set of vertices each of which represents one computation (task), and E is the set of directed edges that represent the data dependencies between vertices. For each vertex v_i we associate it a worst case execution time (WCET) at the reference (highest) voltage that can be obtained a priori by profiling. For each directed edge (v_i, v_j), there is a significant interprocessor communication (IPC) cost when the data from vertex v_i in one processor is transmitted to vertex v_j in another processor although the data communication cost in the same processor can be ignored. We are also given a *deadline* constraint $\mathcal{D}$, which must be met for the completion of any iteration.

DVS System Model We assume there are distributed real-time systems with N homogeneous processors each of which has its private memory. In each system the processor can support continuous or discrete voltage and speed changes. We assume that the energy consumption, when the processor is idle, is ignored [1].

We assume that the dynamic power dominates the processor power consumption. The relation between dynamic power P_d, processor speed (or operational frequency) f and

[1]Our proposed approaches reduce the processor's idle time, therefore, if one considers the energy consumption when the processor is idle, our approaches can save more energy.

supply voltage V_{dd} can be expressed as follows [8]:

$$P_d = C_{ef} \cdot V_{dd}^2 \cdot f \quad (1)$$
$$f = k \cdot (V_{dd} - V_t)^2 / V_{dd} \quad (2)$$

where C_{ef} is the effective switching capacitance, k is the constant and V_t is the threshold voltage. When V_t is small enough to be negligible or can be proportionally adjusted at run time [3], the power consumption is a cubic function (and the energy consumption is a quadratic function) of V_{dd} and f. Specifically, given a workload that requires computation time T at the highest voltage V_{max} and speed f_{max}, if the processor finishes this workload just in time $(T + l - \delta_O)$ at the reduced voltage and speed, then the energy consumption E can be given by

$$E = P_{max} \cdot T \cdot (\frac{T}{T + l - \delta_O})^2 + \Delta_O \quad (3)$$

where P_{max}, the power consumption at the highest voltage and speed, is $C_{ef} \cdot V_{max}^2 \cdot f_{max}$. δ_O is the delay switching overhead, and Δ_O is the energy switching overhead.

Energy-E cient Scheduling Problem Given a task graph where all the tasks have already been assigned to multiple processors [9], we consider the problem of *when and at which voltage should each task be executed in order to reduce the system's energy consumption while meeting the deadline constraint.* Our solution includes two phases: first we use SPM schemes based on WCET to statically assign a time slot to each task. Then we apply dynamic scheduling algorithm (DPM) to further reduce energy consumption by exploiting the slack arising from the run-time execution time variation. As our main contribution is the static part in distribution of slack, in this paper we focus on the static power management techniques.

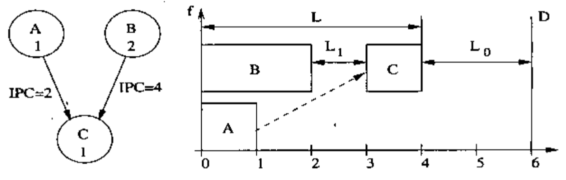
III. SPM: Static Power Management

In this section, we first introduce the current state-of-the-art SPM techniques by a small example. Then, using the same example, we illustrate how to achieve better (actually optimal in this case) solutions by exploiting slacks in the middle of a schedule, a problem known as "non-trivial" [7]. Finally, we propose our approach: SPM with proportional slack distribution and parallelism (or PDP-SPM for short).

A. A Motivational Example

For a scheduled task graph, let $\mathcal{L}$ be the length of its critical path (when all tasks have their WCET) and $\mathcal{D}$ be the deadline, the *global static slack* of such schedule is defined as $L_0 = \mathcal{D} - \mathcal{L}$. The *laxity* is the ratio $\frac{\mathcal{D}}{\mathcal{L}}$, which measures how loose the deadline is. On each processor, there may still exist idle time between the execution of two consecutive tasks due to the IPC cost and dis-synchronization among difference processors. We refer to such idle time as the *local static slack*.

Fig. 1 depicts a task graph with three vertices and a schedule that assigns them to two homogeneous processors, where task A is on one and tasks B and C are on the other.



(a)An application (b)A static schedule for the application

Fig. 1. A task graph and its static schedule on two processors.

The IPC cost between the two processors will be 2 and 4 if task C is scheduled to a processor different from the one for task A and B respectively. The critical path of this schedule in Fig. 1(b) is the path from task A to task C, which has length 4 (including the IPC cost). Therefore, the global static slack $L_0 = 6 - 4 = 2$. We also notice a local static slack $L_1 = 1$ between the execution of tasks B and C on the second processor.

Assuming, for simplicity, that running at the reference frequency f_{max} consumes one unit of energy in one unit time, and there is no energy consumed when the system is idle, this schedule consumes energy in the total of $E_{npm} = 1 + 2 + 1 = 4$ on computation, where $_{npm}$ stands for "no power management".

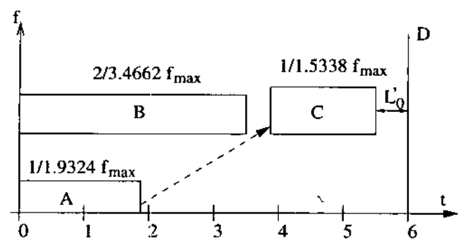
A simple SPM technique distributes the global static slack evenly over the length of the schedule and reduces the speed for all tasks accordingly [3]. In this example, all three tasks will run at $\frac{L}{D} f_{max} = \frac{2}{3} f_{max}$. This results in a total computation energy consumption of $\frac{4}{9} E_{npm}$.

A greedy SPM technique (G-SPM) assigns the global static slack to the first task on each processor and reduces the running speed and voltage on these tasks to save energy [7]. In this case, tasks A and B will get two more units of CPU time and can be run at $\frac{1}{3} f_{max}$ and $\frac{1}{2} f_{max}$, respectively. Task C will still run at the reference speed. This yields a total energy consumption of $\frac{1}{9} + \frac{1}{2} + 1 = \frac{29}{18}$, or $\frac{29}{72} E_{npm}$.

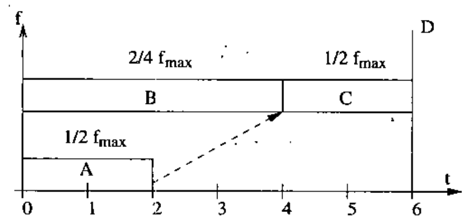
The static power management for parallelism scheme (P-SPM) transfers this problem as a constrained optimization problem and solves it numerically [7]. First, the total CPU time T_i when i processors are running at the same time is counted; next l_i units of CPU time will be added to T_i so the speed can be reduced to $\frac{T_i}{T_i + l_i} f_{max}$; then it seeks to minimize the total energy consumption $\sum_i i \cdot C_{ef} \cdot (\frac{T_i}{T_i + l_i} f_{max})^3 \cdot (T_i + l_i)$ under the constraint that the total CPU assigned $\sum_i l_i$ is no more than the global static slack L_0. This gives the static schedule as shown in Fig. 2 (a) with a total energy consumption of $0.3464 E_{npm}$, slightly less than $\frac{25}{72} E_{npm}$.

We observe that none of these SPM techniques take advantage of the local static slack and this consequently creates new global static slack even when all the original L_0 is distributed as one can see from L_0' in Fig. 2(a). This prevents such techniques from finding the best solution. For example, a better solution shown in Fig. 2(b) is reported as a case when P-SPM fails [7].

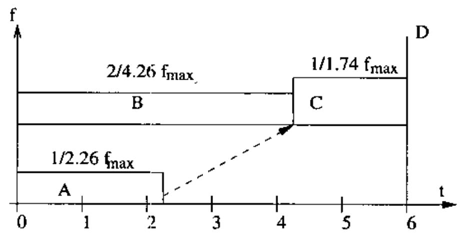
We propose an iterative algorithm that proportionally



(a) Static Power Management with Parallelism



(b) Static Power Management with Proportional Distribution



(c) Static Power Management with Proportional Distribution and Parallelism

Fig. 2. P-SPM scheduler and its improvements.

```
Algorithm: Proportional_Slack_Distribution()
1.  for each vertex $v_i$
2.    { $t_q^i = WCET_i$;  $done(v_i) = 0$; }        /* allocate
      $WCET_i$ to vertex $i$ and prepare to allocate it slack */
3.  compute the current schedule's completion time $L$;
4.  if ($L > \mathcal{D}$)    exit;    /* cannot complete before $\mathcal{D}$ */
5.  while ($done(v_i) = 0$ for some vertex $v_i$)
6.  { compute the current schedule's completion time $L$;
7.    while ($L < \mathcal{D}$)
8.    { for each vertex $v_i$ with $done(v_i) = 0$
9.          $t_q^i = t_q^i \cdot (1 + \delta)$;    /* proportionally distribute
                                  slack to all vertex by $\delta > 0$*/
10.     compute the current schedule's $L$ with the new $t_q^i$;
11.   } /* now $L > \mathcal{D}$, we need to take back the assigned
                                               slack from $t_q^i$. */
12.   for each vertex $v_i$ with $done(v_i) = 0$
13.   { $t_q^i = t_q^i/(1 + \delta)$;
14.     if $v_i$ is on a critical path    $done(v_i) = 1$;
        /* there is no slack left for $v_i$ for the selected $\delta$ */
15.   }
16. } /* the while loop continues as the local slack exists
       in the middle of the schedule off critical paths. */
```

Fig. 3. Procedure to distribute slack proportionally among the vertices in the scheduled task graph.

distributes both the global and local static slack to the tasks under the deadline constraints. It quickly converges to the solution in Fig. 2(b) which has a total computation energy consumption is $\frac{1}{4}E_{npm}$.

Now there is no static slack left, however, we find that it is still possible to reduce energy by further exploiting the parallelism. Specifically, if we reduce the CPU time assigned to task C, then more energy will be consumed on executing C, but both tasks A and B will get more CPU time and can run at reduced speed to save energy. We win when the energy savings from A and B are more than the extra energy we spend on task C. In this case, we find a static schedule with $0.2417E_{npm}$ energy consumption, which is better than $\frac{1}{4}E_{npm}$, as shown in Fig. 2(c).

B. PDP-SPM: SPM with Proportional Distribution and Parallelism

The proposed PDP-SPM algorithm is based on the above observations. It consists of two phases, the global and local static slack distribution and the CPU time re-allocation for parallelism. Fig. 3 shows the *Proportional_Slack _Distribution*() procedure that iteratively distributes slack to tasks. The global slack is distributed evenly to all vertices in step 9 by expanding their allocated slot t_q^i by a factor of δ till the completion time L exceeds the deadline $\mathcal{D}$. Let L_{IPC} be the total IPC cost on any critical path when this inner *while* loop (steps 7-11) stops. It is easy to see that this loop will execute at most $\log_{1+\delta}\frac{\mathcal{D}}{L_{ini}-L_{IPC}}$ times, where L_{ini} is the initial completion time obtained in step 3.

Then we need to scale down the CPU time assigned to all the vertices so the deadline $\mathcal{D}$ can be made. For all the nodes that are on any critical paths, we mark them *done* because expanding such nodes' CPU time by the factor δ again will violate the deadline constraint (steps 12-15). Note that, following a (predetermined) topological order, one can compute the completion time and obtain the vertices on the critical paths for a given schedule in time linear to the N, the number of nodes in the task graph.

At this point, the completion time is within δ of the deadline (i.e., $\mathcal{L} > \frac{\mathcal{D}}{1+\delta}$). The remaining global static slack is not sufficient to be distributed proportionally among all the vertices by the factor δ again. However, we can still distribute this along with the local static slack to those nodes off critical path. This is conducted in the outer *while* loop (steps 5-16) till all the nodes are marked *done*. This loop will be executed no more than N times as each time there will be new nodes being marked *done*. For given parameters such as $\mathcal{D}$ and δ, we conclude that the *Proportional_Slack_Distribution*() algorithm has a computational complexity of $O(N^2)$.

In the second phase, PDP-SPM re-allocates the CPU time assigned to each task based on the degree of parallelism (that is, the number of processors running at the same time). The basic idea is to *create slack* by reducing the time assigned to the tasks with small degree of parallelism. Fig. 4 depicts in detail this phase, where we take away time

```
Algorithm: PDP-SPM()
1.  calculate $E_p$ for present time allocation;
2.  for each vertex $v_i$   $finish(v_i) = 0$;
                    /* time re-allocation for $v_i$ is not finished*/
3.  while ($finish(v_k)$= 0 for some vertex $v_k$)
4.  { for each vertex $v_i$ { $done(v_i) = finish(v_i)$;   $t_q^{i\prime} = t_q^i$; }
5.    select $v_j$ from the unfinished vertices with the
      smallest degree of parallelism;
6.    $t_q^j = t_q^j - \delta L$;
7.    run steps 5-16 in Proportional_Slack_Distribution()
      as shown in Fig. 3;
8.    calculate the energy consumption $E_n$ based on
      Equation (3) for the new setting;
9.    if ($E_p - E_n > \Delta$)    $E_p = E_n$;  /* a positive threshold
                                  $\Delta$ to speed up the iteration */
10.   else        /* no or little energy reduction, restore the
                  previous assignment and mark $v_j$ finished */
11.   { for each unfinished vertices $v_k$     $t_q^k = t_q^{k\prime}$;
12.       finish($v_j$) = 1;
13.   }
14. }
```

Fig. 4. Phase II of PDP-SPM: Re-allocate time to each task based on parallelism.

δL from task v_j that has the smallest degree of parallelism in step 6. This introduces local slack in the amount of δL on the processor which v_j is assigned to. Then part of the *Proportional_Slack_Distribution*() procedure is called to distribute such slack in step 7. If this gives us energy reduction, we continue the *while* loop (steps 3-14) by taking away time from the same task again. We repeat this until there is no or little energy saving. We mark v_j finished, meaning that it gets its final CPU assignment. However, the same *while* loop will continue running on the remaining unfinished tasks. Because that *Proportional_Slack_Distribution*() has an $O(N^2)$ run time, the complexity of PDP-SPM algorithm is $O(N^3)$.

C. Voltage and Speed Selection

After the SPM techniques distribute the slack to each task, the system need to select the appropriate voltage and speed to execute it. The provably most energy-efficient voltage scaling technique has been reported as follows [8]:

- for ideal voltage systems, select voltage that corresponds to speed $f_{ideal} = \frac{WCET_i}{t_q^i} f_{max}$, where t_q^i is the time slot allocated to vertex v_i.
- for discrete voltage (speed) systems, find the two consecutive speed f_1 and f_2 with $f_1 < f_{ideal} < f_2$, execute at f_1 for certain time and then switch to f_2 till v_i's completion.

Note that the above technique can be easily adapted to the case when there are non-negligible delay and energy overhead for voltage switching. We will switch the voltage only when the voltage scaling provides us sufficient energy saving to compensate the overhead. When voltage and speed are not allowed to change at any time as assumed in [8], we can apply the method proposed by Zhang et al. [11] to minimize the number of intra-task and inter-task transitions to achieve the maximal energy reduction.

IV. Simulation Results

A. Simulation Setup

We have implemented the two algorithms: P-SPM proposed originally in [7], and our proposed PDP-SPM. We simulate them over a variety of real-life graphs such as FFT (Fast Fourier Transform), Laplace (Laplace transform) and karp10 (Karplus-Strong music synthesis algorithm with 10 voices), and some random task graphs generated by a standard package TGFF [2]. Before we apply the algorithms to the benchmark graphs, we use the dynamic level scheduling (DLS) [9] method to schedule the tasks to different processors.

Each processor in the systems can support continuous or discrete voltage(speed) changes. For discrete case, the processors are modeled to support four different voltage and speed pairs ((1.75V,1000MHz), (1.40V, 800MHz), (1.20V, 600MHz) and (1.00V, 466MHz)).

Based on the WCET, we can get the length of the critical paths L for the task graph that indicates the minimum deadline we can set without any deadline missing. By varying the *laxity*, we model the inverse of the load imposed on the systems by the actual deadline $D = laxity \times L$, where $laxity \geq 1$. If not specify, the *laxity* is 1.5 in the simulation.

In order to compare the energy consumption for all algorithms, we normalize the energy consumption for no power management (NPM) to be 1.

B. Results and Discussion

We first analyze the sensitivity for the PDP-SPM algorithm. Then give the comparison of the energy consumption by using different algorithms such as P-SPM and PDP-SPM on different benchmarks, which use WCET in the case of continuous or discrete voltage and speed changes. We also investigate the impact of the laxity and number of processors to energy efficiency of different SPM algorithms.

Sensitivity analysis of PDP-SPM is done offline by determine the optimal value of δL (step 6 in Fig. 4), where $\delta L = \frac{L_0}{m_{max} K}$. L_0, which is equal to $D - L$, is the global static slack for the current setting. m_{max} is the maximum number of tasks assigned to one processor. K is the granularity of PDP-SPM. When we change the granularity K, δL will be changed. From the simulation we have seen that the energy consumption does not change much (within 1% error) with the variation of granularity from 1 to 10,000. Therefore, we set the granularity to be 100 for the rest of simulation.

Table I reports the average energy consumption per iteration by different SPM algorithms on different benchmarks. From the table we can see that in average P-SPM can save more than **58%** energy when the systems support continuous speed and voltage changes, and **48%** energy when they

TABLE I
AVERAGE ENERGY CONSUMPTION/SAVING PER ITERATION BY USING NPM, P-SPM AND PDP-SPM WITH DEADLINE CONSTRAINTS $\mathcal{D}$. (THE PERCENTAGES STAND FOR THE ENERGY SAVING; n: NUMBER OF VERTICES IN THE BENCHMARK TASK GRAPH; m: NUMBER OF PROCESSORS; ENERGY IS IN THE UNIT OF THE DISSIPATION IN ONE CPU UNIT AT THE REFERENCE VOLTAGE AND SPEED; E_{i1}, E_{i2} ARE THE ENERGY CONSUMPTION AT CONTINUOUS CHANGES; E_{d1}, E_{d2} ARE THE ENERGY CONSUMPTION AT DISCRETE CHANGES.)

Benchmark	n	m	No. IPCs	$\mathcal{D}$	NPM	P-SPM		PDP-SPM			
					energy E_0	E_{i1} vs. E_0	E_{d1} vs. E_0	E_{i2} vs. E_0	E_{i2} vs. E_{i1}	E_{d2} vs. E_0	E_{d2} vs. E_{d1}
FFT1	28	2	15	1404	1520	57.48%	48.39%	63.63%	14.47%	52.51%	7.98%
FFT2	28	2	16	1660	960	75.25%	63.92%	82.97%	31.20%	67.35%	9.51%
Laplace	16	2	13	3555	2880	60.56%	50.42%	76.33%	39.98%	64.25%	27.90%
karp10	21	2	12	1127	1188	58.08%	48.88%	61.00%	6.95%	50.53%	3.24%
TGFF1	39	2	20	5235	6548	56.48%	46.99%	60.82%	9.97%	50.22%	6.09%
TGFF2	51	3	36	4902	7658	56.93%	47.52%	66.03%	21.13%	54.43%	13.16%
TGFF3	60	3	51	6133	10522	56.77%	47.29%	65.69%	20.64%	54.40%	13.48%
TGFF4	74	2	49	9367	12170	55.96%	46.51%	57.80%	4.17%	47.79%	2.39%
TGFF5	84	3	74	7695	13694	56.77%	47.16%	63.02%	14.46%	51.87%	8.92%
TGFF6	91	2	59	12417	15122	56.58%	47.15%	62.42%	13.44%	51.63%	8.48%
TGFF7	107	3	89	9706	17256	56.47%	47.11%	62.59%	14.07%	51.54%	8.37%
TGFF8	117	3	111	10527	18516	56.75%	47.16%	63.86%	16.44%	52.67%	10.42%
TGFF9	131	2	85	15456	20242	55.80%	46.37%	57.09%	2.93%	47.29%	1.71%
TGFF10	147	4	163	10023	23364	56.33%	46.99%	63.66%	16.79%	52.63%	10.63%
TGFF11	163	3	159	13403	26014	55.79%	46.39%	58.11%	5.25%	48.04%	3.08%
TGFF12	174	4	169	12682	30190	56.54%	47.05%	61.55%	11.53%	50.91%	7.28%
average energy saving						58.03%	48.46%	64.16%	15.21%	53.00%	8.91%

support discrete changes, over NPM. More importantly our PDP-SPM scheme can save more energy than P-SPM on all benchmarks. Specifically, PDP-SPM can have nearly **15%** and **9%** energy saving over P-SPM in the case of continuous and discrete changes, respectively. And obviously the energy consumption in the case of continuous changes is much lower than that in the case of discrete changes for the same task graph and algorithm.

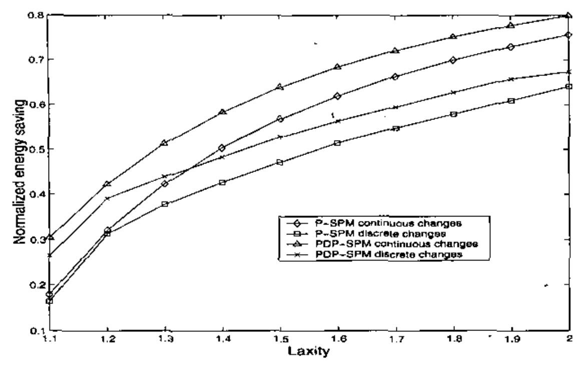
Fig. 5 depicts the laxity's impact to energy efficiency of P-SPM and PDP-SPM. PDP-SPM can save more energy than P-SPM with any laxity. The smaller is the laxity, the more energy efficient is the PDP-SPM in both continuous and discrete changes. The reason is that P-SPM only exploits the global slack while PDP-SPM exploits both global and local slack. With decreased laxity, the global slack will be reduced while the local slack due to task synchronization does not change, which results in the energy saving by PDP-SPM being reduced with its reduction percentage being less than P-SPM's. Similarly we can explain the scenario in Fig. 6 that shows the energy consumption per iteration by using NPM, P-SPM and PDP-SPM with different number of processors and different deadlines. For the same task graph, using more processors will reduce the length of its critical paths and increase the task parallelism. Therefore for the same deadline such as 6000, with the increase of the number of processors, both the global and local slack will increase. This results in the energy reduction by using either P-SPM or PDP-SPM scheme and that PDP-SPM achieving more energy saving compared with P-SPM. It is mentioned that when the deadline is too loose, such as 12000 in the case of discrete changes and the number of processors is larger than 4 (Fig. 6(b)), all tasks will be executed at the lowest voltage (1.0V) and speed (466MHz) and the energy consumption will keep constant regardless of the increase of the number of the processors. We conclude that our PDP-SPM algorithm can save significant energy over P-SPM, especially when the deadline is tight and the number of processors is large.

V. CONCLUSIONS

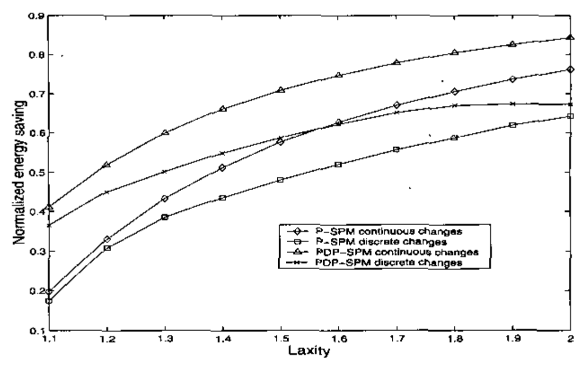
In this paper, we consider how to schedule a set of dependent tasks for distributed real-time systems in order to reduce the energy consumption without any deadline misses. Given the task assignment and ordering, we propose the PDP-SPM (Static Power Management with Proportional Distribution and Parallelism) algorithm that exploits both the global and local slack. It can save more energy than P-SPM (SPM with Parallelism), which at the best of our knowledge is the most energy-efficient SPM scheme. Specifically, compared with P-SPM, PDP-SPM can save more than **15%** and **8.9%** energy in the case of continuous and discrete voltage/speed changes, respectively. Similarly, when compared with NPM (No Power Management), PDP-SPM can save more than **64%** and **53%** energy.

REFERENCES

[1] T. D. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system", *IEEE J. Solid-State Circuits*, Vol. 35, pp. 1571–1580, 2000.

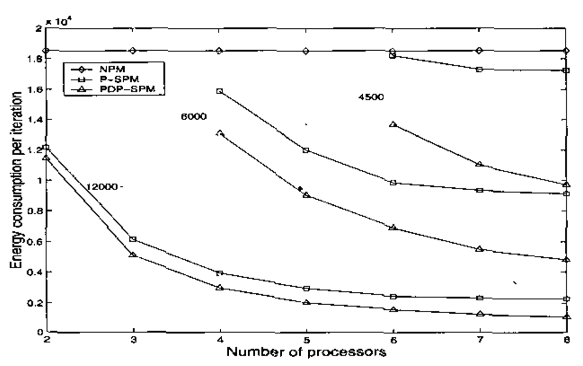


(a) energy vs. laxity for 3 processors

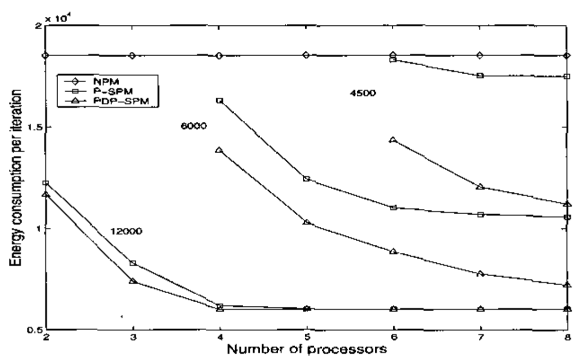


(b) energy vs. laxity for 6 processors

Fig. 5. The normalized energy saving by using different SPM schemes on benchmark TGFF8 with different laxity.



(a) continuous voltage and speed changes



(b) discrete voltage and speed changes

Fig. 6. The energy consumption per iteration by using different SPM schemes on benchmark TGFF8 with different number of processors and different deadlines (12000, 6000, and 4500).

[2] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free", *Proc. Int. Workshop Hardware/Software Codesign*, pp. 97–101, 1998.

[3] F. Gruian, "System-level design methods for low-energy architectures containing variable voltage processors", *Proc. of the Workshop on Power-Aware Computing Systems*, pp. 1–12, 2000.

[4] S. Hua, G. Qu, and S. S. Bhattacharyya, "Energy-efficient multi-processor implementation of embedded software", *3rd ACM International Conference on Embedded Software*, pp. 257–273, Oct. 2003.

[5] F. Liberato, S. Lauzac, R. Relhem, and D. Mosse, "Fault tolerant real-time global scheduling on multiprocessors", *Proc. of the 10th IEEE Euromicro Workshop in Real-Time Systems*, June 1999.

[6] J. Luo and N. K. Jha, "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems", *IEEE/ACM International Conference on Computer-Aided Design*, pp. 357–364, Nov. 2000.

[7] R. Mishra, N. Rastogi, D. Zhu, D. Mosse, and R. Melhem, "Energy aware scheduling for distributed real-time systems", *International Parallel and Distributed Processing Symposium*, pp. 1–9, 21b., 2003.

[8] G. Qu, "What is the limit of energy saving by dynamic voltage scaling?" *IEEE/ACM International Conference on Computer-Aided Design*, pp. 560-563, 2001.

[9] G.C. Sih and E.A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures", *IEEE Tran. on Parallel and Distributed Systems*, Vol 4, pp. 175–187, 1993.

[10] Y. Zhang, X. Hu, and D. Chen, "Task scheduling and voltage selection for energy minimization", *ACM/IEEE Design Automation Conference Proceedings*, pp. 183–188, 2002.

[11] Y. Zhang, X. Hu, and D. Chen, "Energy minimization of real-time tasks on variable voltage processors with transition energy overhead", *ACM/IEEE Asia and South Pacific Design Automation Conference*, pp. 65–70, 2003.