Task Merging for Dynamic Power Management of Cyclic Applications in Real-Time Multiprocessor Systems

Qinru Qiu, Shaobo Liu, and Qing Wu
Department of Electrical and Computer Engineering
Binghamton University, State University of New York
Binghamton, NY 13902 USA
{qqiu, sliu5, qwu}@binghamton.edu

***Abstract*—In this paper we propose the method of task merging and idle period clustering for dynamic power management (DPM) in a real-time system with multiple processing elements. We show that with good task scheduling, the energy and delay overheads due to power mode switching can be reduced significantly, while the opportunity for the system to switch to low power modes can be further improved. New on-line and off-line task scheduling algorithms are proposed that minimize the number of idle time intervals under the deadline and precedence constraints. A simple DPM policy is then used to save the energy dissipation during the idle time intervals. Experimental results show that, comparing to the DPM schemes without proper task scheduling, the proposed method reduces the number of power mode switching by 56% in average.**

***Index Terms*—dynamic power management, low power, multiprocessor, real-time**

I. Introduction

Multiprocessor systems, which were used only in the area of high performance computing, has become more and more popular in the sensor, mobile entertainment, and other real-time applications. In these systems, the processors collaborate with each other to process tasks with precedence and deadline constraints. For example, a system could consist of an embedded processor and a digital signal processing (DSP) FPGA [1]. A simple flow of sensing and cognition application running on this system involves both processors. At first, the data need to be preprocessed by the embedded processor, and then the results are given to the DSP FPGA for advanced processing. The DSP outputs are sent back to the embedded processor for knowledge rule generation.

Dynamic power management (DPM) is one of the most effective system-level power reduction techniques [7]. For any DPM technique, there are always energy and delay overheads associated with power mode switching. For example, it is shown that [8], to power up an XScale processor, it takes 10ms precharge time and 500mW power drains. These are significant overheads considering that the processor's active power is only 200mW when running at full speed. As another example, the cost to power on and off an FPGA chip is also very high if SRAM based LUT is used [9]. Such overheads are normally not negligible, especially in long-term mobile applications where the system consumes little active power. The goal of an optimal DPM policy is to maximize the system sleeping time while minimize switching overhead.

Most of the previous power management techniques focus on the stochastic behavior of the system. Various DPM techniques have been proposed, from predictive based heuristic techniques [1][3] to stochastic optimization based techniques [4][5]. However, there are a large number of systems whose behavior is deterministic. For example, a sensor node senses, processes, communicates and records information periodically. Because of the deterministic behavior, we know exactly how long the next idle period will be. In such system, the greedy DPM policy, which switches the system into low power mode if the next idle period is longer than the break even time (T_{be}), is the most simple and effective approach. Because the application is deterministic, the overall idle time and active time of the system is fixed. The only variable that affects the performance of power management is how these idle and active intervals are distributed. Obviously, proper application scheduling that merges tasks together or clusters the idle periods will facilitate the power management because it increases the chances to put the system into low power modes and reduces the number of power mode switching.

For a real-time multiprocessor system and a set of cyclic deterministic tasks, the problem of low-power task scheduling that is considered in this paper, can be stated as follows. Given any task graph, assuming that the mapping between tasks and processors is determined, find the optimal task scheduling combined with power management techniques, such that the system power consumption is minimized while meeting the deadline and precedence constraints.

Only few existing DPM research works consider scheduling and DPM at the same time. Among them, the authors of [10] proposed a scheduling algorithm with the consideration of min/max timing and min/max power constraints. In reference [11], an on-line scheduling algorithm is proposed to reduce the number of power mode switching in a multiprocessor system. However, precedence and deadline constraints are not considered during the optimization. The authors of [12] solve the task scheduling and hierarchical power management problems at the same time by using the continuous-time Markov decision process. The model is constructed based on the architecture with single processor and multiple peripherals. No task precedence constraints can be incorporated in the model.

In this paper, we propose new on-line and off-line task scheduling algorithms that consider task merging to facilitate power management. The characteristics of the proposed work are described as follows.

1. Compared with minimum latency task scheduling, the proposed scheduling algorithms significantly reduces the number of idle intervals and extends the length of each idle interval. Therefore, they produce a sequence of scheduled system activities that are more suitable for power management.
2. The proposed problem formulation considers the general constraints in a real-time multiprocessor system, such as the task arrival time, task deadline, data/control dependency, and buffer size. The mapping between the tasks and the processors is assumed to be given.
3. The proposed method is targeted at periodic task graphs. The on-line algorithm buffers and burst processes the tasks. The off-line algorithm first unrolls the given task graph for a number of times, then schedules the unrolled task graph using a faster heuristic algorithm called *slack-based task merging* (STM).
4. We apply a simple DPM policy to reduce power consumption by turning off the processors if their idle time intervals are longer than T_{be}.

The rest of this paper is organized as follows. Section II gives a motivational example that demonstrates the importance of task scheduling to DPM. Section III presents the formal problem definition. The two proposed task scheduling algorithms are introduced in Section IV and V. Sections VI and VII provide the experimental results and summaries.

II. A MOTIVATIONAL EXAMPLE

Consider a sensing and data processing application that is implemented on a two-processor system. The sensor acquires data every 1ms and each data goes through three steps of operations. The first and the third operations are executed on processing element 1 (PE1), while the second operation is executed on processing element 2 (PE2). The execution times of the three operations are 0.1ms, 0.2ms, and 0.1ms respectively. Because of the nature of the sensing application, the sampled data do not need to be processed immediately, however, their processing cannot be delayed excessively either. Otherwise the system cannot detect the abnormal event in time. The deadline for data sampled at time t to go through the ith operation is defined as $t + td_i$. The value of td_i, i= 1, 2, 3 are 3ms, 4ms, and 5ms respectively. When an operation is completed, a request is sent to its successor. If the request cannot be processed then it will be buffered and be noted as *pending request*. We assume that the request is serviced in first-in-first-out order.

The simplest scheduling for the above system is to process each operation as soon as the input data is available. Figure 1 shows the activities of both processors when the operations are scheduled for minimum latency. The light blue (light grey), red (medium grey) and dark blue (dark grey) areas indicate the time when the processors are busy processing op_1, op_2, and op_3 respectively. The white area indicates the time when the processor is idle. A number is associated with each active interval, which indicates the index of the sampled data that is currently under processing.

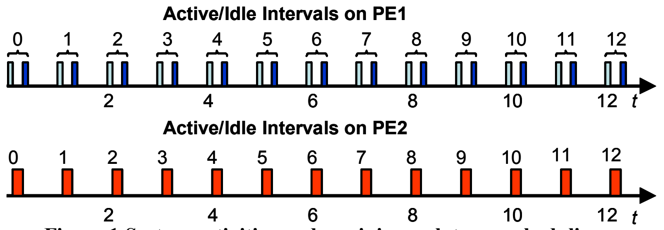


Figure 1 System activities under minimum latency scheduling.

A simple DPM policy is applied to each processor such that, if the idle time is less than some break even time T_{be}, then the processor will be turned off during the idle period. Obviously, the minimum latency scheduling does not work well with dynamic power management because it breaks the idle period into many small intervals. This will either prohibit the processor to switch to low power mode because the idle intervals are long enough or lead to a lot of on/off power mode switching.

The optimal scheduling algorithm minimizes the number of idle intervals by merging tasks together. At the same time, it guarantees the task deadline and precedence constraints. Figure 2 gives the system activities under the optimal scheduling. Compared with Figure 1, such active and idle interval distribution gives more power saving opportunities and less number of on/off power model switching.

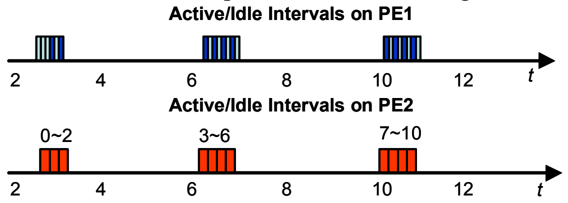


Figure 2 System activities under optimal scheduling.

III. PROBLEM DEFINITION

In this section we are going to give the formal definition of the scheduling problem. The hardware system that we are interested is a multiprocessor system with point-to-point communication. A control/data flow graph (CDFG) $G(V, E)$ is given that models the application precedence and timing constraints. V is the set of operations which is also denoted as OP. An edge from vertex i to vertex j with weight $w_{i,j}$ indicates that t_j-$t_i \geq w_{i,j}$, where t_i and t_j are the starting time of op_i and op_j. Associated with operation i there is a queue (q_i). When all of the input data of i is ready, a request for op_i is generated and is buffered in the queue. The capacity of the queue is denoted as B_i. The number of pending requests in q_j is denoted as β_j.

Some of the operations do not have any predecessor. These operations are triggered periodically and the cycle time is denoted as T. These operations are called the *triggering operations*. For example, the data acquisition in a sensor system and the carrier detection in a wireless communication system are both triggering operations. The start time of the triggering operation is fixed. Whenever the triggering operation is active, there is a *triggering event* (*trg*) generated. The triggering event will propagate through the CDFG and drives the system operation.

The deadline of an operation is defined relative to the time when the triggering event is generated. The triggering events and deadlines are anchors in the CDFG.

We assume that the mapping of the operations to the processors is provided. In a heterogeneous multiprocessor system, such mapping is usually based on the different functionality of the processors. We use $proc(op_i)$ to denote the processor to which the op_i is mapped. The processing time and the deadline of op_i is denoted as tp_i and td_i respectively. The value of td_i may be larger than the cycle time T. The rth initiation of the triggering operation starts at rT, $0<r<\infty$.

The scheduling problem is to find the starting time ($t_{i,r}$) for the rth initiation of operation i, $0<r<\infty$. The objective is to minimize average number of idle intervals in each cycle while satisfying general system constraints such as the deadline constraints, buffer size constraints, precedence constraint and sequencing constraints. These constraints can be formulated as the following:

$$t_{i,r} + tp_i \le r \cdot T + td_i, \quad 0<r<\infty$$

$$\beta_{i,j} \le B_{i,j}, \quad (i,j) \in E$$

$$t_{i,r} + tp_i \le t_{j,r}, \quad (i,j) \in E, \text{ and } 0<r<\infty$$

$$(t_{i,r}, t_{i,r} + tp_i) \cap (t_{j,r'}, t_{j,r'} + tp_j) = \phi,$$

$$proc(op_i)=proc(op_j),\ 0<r',r<\infty, \text{ and } r' \ne r$$

We assume that the utilization ratio of each processor $U_p = \frac{\sum_{proc(op_i)=p} tp_i}{T}$ is less than 1. Otherwise no feasible scheduling solution can be found. In the formulation, we assume that the system uses a DPM policy in which each processor will switch from "on" to "off" if its idle time exceeds T_{be}.

IV. On-line task merging Based on burst processing with EDF task selection

An ad-hoc way to cluster active and idle intervals is to buffer and burst process the requests. Based on this motivation, we developed two ad-hoc task merging algorithms.

A processor using Burst Processing I (BP-I) scheduling algorithm buffers the requests for each operation. If the deadline of the first pending request for op_i approaches, then the processor will start processing this request. It will not stop until all pending requests for op_i are processed.

The Burst Processing II (BP-II) is similar as the BP-I, except that each time after processing all pending requests for one operation, the processor will not stop if there is any other pending requests. The processor will stop only if it finishes processing all pending requests in the system.

As an example, we use BP-I and BP-II to schedule the applications that is described in the motivational example. Figure 3 shows the resulting system activities. As we can see, the BP-II gives a better way to cluster the active/idle intervals on PE1, however, it fragments the active/idle intervals on PE2.

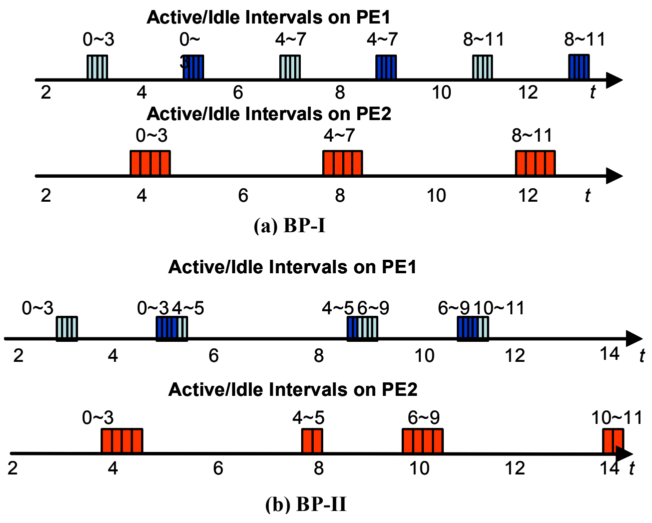


Figure 3 System activities under ad-hoc task scheduling.

Compared with the system activities after the optimal scheduling, the system under BP-I or BP-II scheduling has more idle intervals during the same period of time. On the other hand, the duration of each idle interval is shorter. Another serious problem with BP-I and BP-II is that they cannot guarantee deadline. This is because the ad-hoc scheduling algorithms do not consider the overall deadline and precedence constraints.

A notable difference between the optimal scheduling and the ad-hoc scheduling is that, instead of simply buffering and burst processing the incoming tasks, the optimal scheduling algorithm interleaves the processing of different operations on the same processor to achieve better clustering. Motivated by this observation, we propose the third on-line task merging method, which is based on burst processing with the earliest deadline first task selection (BP-EDF.)

The BP-EDF method is similar as BP-II. However, instead of processing all the pending requests for one operation after another, the processor always selects the pending request that has the earliest deadline first. Compared with BP-I and BP-II, BP-EDF significantly reduces the ratio of deadline miss, while having almost the same or sometime even less average number of idle intervals in the same period of time.

Although the BP-EDF has a reduced deadline miss ratio compared with BP-I and BP-II, it still cannot guarantee deadline. To fix this problem, a scaled deadline is used to determine the latest time that a request must be processed. More specifically, the latest starting time of the rth request for op_i is calculated as $r \cdot T + td_i \cdot \alpha - tp_i$. Here α is a scaling factor, $r \cdot T + td_i$ is the deadline of the request and tp_i is the processing time of the request. It is easy to see that decreasing the value of α reduces the number of deadline misses. However, it also reduces the chances of task merging.

Table 1 Scaling Factor vs. Quality of Scheduling.

# of PEs	3								2							
#of tasks	12				15				12				15			
Utilization ratio	41%		12%		74%		22%		76%		22%		68%		20%	
(P, D) / α	P	D (%)	P	D (%)	P	D (%)	P	D (%)	P	D (%)	P	D (%)	P	D (%)	P	D (%)
0.1	5.0	0	5.0	0	4.5	0	6.0	0	3.0	0	4.0	0	3.0	0	3.0	0
0.2	5.0	0	5.0	0	4.5	0	6.0	0	3.0	0	4.0	0	3.0	0	3.0	0
0.3	3.0	0	4.0	0	4.5	0	5.0	0	3.0	0	4.0	0	3.0	0	3.0	0
0.5	2.5	0	4.0	0	4.3	0	4.0	0	3.0	0	4.0	0	2.5	0	2.0	0
0.7	2.0	0	4.0	0	3.0	0	4.0	0	3.0	0	4.0	0	2.0	0	2.0	0
0.8	2.0	0	3.5	0	2.7	0	4.0	0	3.0	0	4.0	0	2.0	0	2.0	0
0.9	2.0	0	2.5	0	1.6	2	4.0	0	3.0	10	2.5	0	2.0	0	2.0	0
0.95	2.0	0	2.5	0	1.6	2	3.9	0	2.8	16	2.5	0	1.5	0	2.0	0
1.0	2.0	17	2.3	0	1.5	1	3.3	0	2.4	15	2.5	4	1.5	0	2.0	0

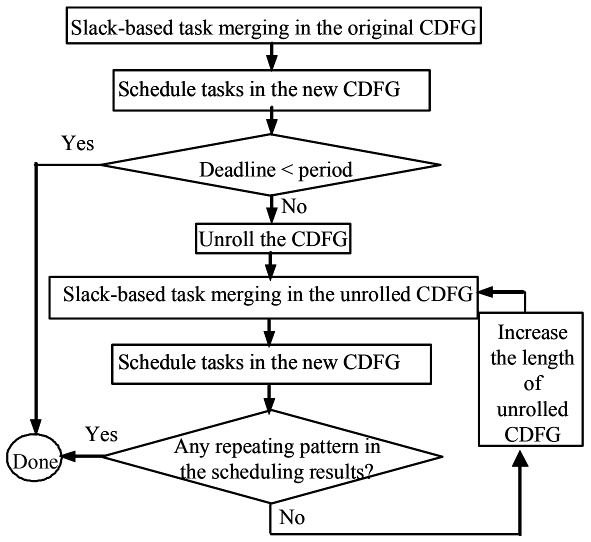
We measure the quality of task merging and scheduling algorithm using two parameters, the average number of idle intervals per triggering cycle and the average deadline miss ratio. Here we need to emphasize again that the overall idle and active time of each processor is fixed. The only variable is how these idle and active intervals are distributed. The first parameter associates with the average power consumption of the system, since the less number of idle intervals (or longer idle intervals) corresponds to less on/off switching and more opportunities to go to low power mode. The second parameter associates with the performance and delay of the system. Therefore, we denote the quality of scheduling using a pair of symbols (P, D), where P represents the average number of idle intervals and D represents the deadline miss ratio.

Simulations have been carried out to evaluate the relation between the scaling factor (α) and the quality of BP-EDF. Our simulation setup consists of systems with 2 or 3 PEs. Several different task graphs are randomly generated using TGFF [12]. We vary α from 0.1 to 1.0 and collected a set of (P, D) values as the measurement of the quality of BP-EDF. Table 1 gives our simulation results. The first three rows give the experiment setup information, which include the number of PEs, the number of tasks in the CDFG and the average processor utilization ratio of the overall system. The last 11 rows give the (P, D) value for different setups when α varies from 0.1 to 1.0. The results show that as the value of α increases, the average number of idle intervals in a triggering cycle decreases while the deadline missing ratio increases.

Because the BP-EDF requires very little computation, it can be used for on-line scheduling and task merging.

V. OFF-LINE TASK SCHEDULING WITH THE CONSIDERATION OF TASK MERGING

In this section, we are going to introduce an off-line algorithm that considers task merging and idle period clustering during task scheduling. The algorithm utilizes task slack time to merge them together. Therefore, in the rest of the paper, we refer to this algorithm as slack-based task merging (STM) algorithm. Compared with BP-EDF, the STM algorithm guarantees deadline and merges tasks more efficiently when the processor utilization is relatively low.



Slack-based task merging in the original CDFG
Schedule tasks in the new CDFG
Yes
Deadline < period
No
Unroll the CDFG
Slack-based task merging in the unrolled CDFG
Increase the length of unrolled CDFG
Schedule tasks in the new CDFG
Yes
Any repeating pattern in the scheduling results?
Done
No

Figure 4 The STM algorithm.

Figure 4 gives the flow of STM algorithm. The algorithm first works on the original CDFG to merge tasks together. A cluster of merged tasks is called *composite task* and the original single task is called *atomic task*. After merging, new task graph is generated that consists of both composite and atomic tasks. A heuristic algorithm is used to schedule the tasks in the new CDFG so that those tasks that are mapped to the same processor will be executed sequentially. If the task deadlines are less than one initiation cycle, then the algorithm stops. Otherwise cross-boundary task merge should be performed to further reduce the number of idle intervals. The STM algorithm will unroll the CDFG and merge tasks that belong to different triggering cycles. After that task scheduling is performed again. If a repeating pattern is found in the scheduling result then we will stop, otherwise we increases the length of the unrolled CDFG and repeat the previous steps. In all our experiments, we are able to find repeating pattern by unrolling the CDFG 100 times.

The timing relations between composite tasks are very complex. Sometime, a composite task cannot start too early or too late relative to the starting time of another composite task. Both maximum and minimum timing constraints [13] may exist in the CDFG after task merging. Given two operation i and j. The maximum timing constraint $u_{ij} \geq 0$ requires $t_j \leq t_i + u_{ij}$. It is represented as an edge from vertex j to i with negative weight $-u_{ij}$. The minimum timing constraint $l_{ij} \geq 0$ requires $t_j \geq t_i + l_{ij}$. It is represented as an edge from vertex i to j with positive weight l_{ij}.

A. *Task Merging Algorithm*

The key of STM algorithm is slack based task merging. The slack of an operation is the difference between its earliest and latest starting time obtained from the ASAP and ALAP scheduling. The slack defines the mobility of the scheduling of an operation. If the slack of two operations that are running on the same processor allows them to be scheduled back to back then these two operations can be merged to form a bigger operation.

Let (est_i, lst_i) denotes the slack of an operation i, where est_i is the earliest starting time and lst_i is the latest starting time. Given an unrolled CDFG, two operations i and j can be merged if and only if the following conditions are true:

$C1$: i and j are both executed on the same processor

$C2$: The slacks of the two operations are overlapping, i.e. $(est_i, lst_i + tp_i) \cap (est_j, lst_j + tp_j) \neq \phi$

$C3$: Let the $LD_{i,j}$ and $LD_{j,i}$ denote the distance of the longest path from i to j and from j to i respectively. (If j is not reachable from i then $LD_{i,j} = -\infty$. Similarly, $LD_{j,i} = -\infty$ indicates that i is not reachable from j.) The processing time of i and j must satisfy either of the following:

$$LD_{i,j} \leq tp_i \leq -LD_{j,i} \qquad (1)$$

$$\text{or } LD_{j,i} \leq tp_j \leq -LD_{i,j} \qquad (2)$$

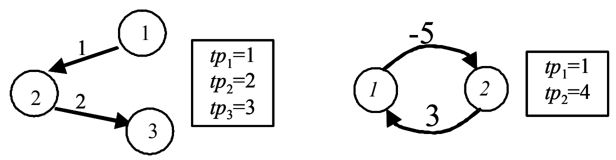
If (1) is satisfied then task j should follow task i after merge. If (2) is satisfied then task i should follow task j.

$C4$: To merge i and j with j following i, the following inequality must be true: $lst_k - est_k > tp_j$ if there is an edge from i to k and both i and k are mapped to the same processor. This condition guarantees that after merging, executing task i, j and k sequentially will not violate the timing constraint of k.

The necessity for condition C1, C2, and C4 is obvious. We will give some discussions about C3 using the following two examples.

Example 1 Consider the task graph given by Figure 5 (a), which models three sequentially executed tasks. Assume that all of three tasks are running on the same processor and there is no deadline constraint (i.e. the slack of each task is infinite). It is easy to see that task 1 and 3 cannot be merged. Checking conditions C1~C4, we will find that C3 is not satisfied. From the graph we know that $LD_{1,3} = 3$ and $LD_{3,1} = -\infty$. Therefore, neither (1) or (2) is true because $tp_1 < LD_{1,3}$ and $tp_3 > LD_{3,1}$. □

Example 2 Consider the task graph given by Figure 5 (b), which models two tasks with minimum and maximum timing relations. Task 1 must start at least 3 cycles but no more than 5 cycles after task 2 started. It is not difficult to find that task 1 and 2 can be merged with task 1 following task 2. Indead, condition C3 is satisfied for this example. Because $LD_{1,2} = -5$ and $LD_{2,1} = 3$, we have $LD_{2,1} \leq tp_2 \leq -LD_{1,2}$. □



(a) Task merging in a CDFG with minimum timing constraints

(b) Task merging in a CDFG with min/max timing constraints

Figure 5 Meeting condition *C*3.

After two operations i and j are merged, the corresponding two vertices and all the edges associated with them are removed from the graph. Assume that i is placed in front of j, a new vertex i-j is created for the composite task. The graph is updated based on the following rules.

U1: If there is an edge (k, i) in the original graph with weight w, add an edge (k, i-j) and set its weight to w.

U2: If there is an edge (k, j) in the original graph with weight w, add an edge (k, i-j). If k is mapped to a different processor other than i and j, the weight of the new edge is equal to w-tp_i; otherwise it is equal to w.

U3: If there is an edge (i, k) in the original graph with weight w, add an edge (i-j, k). If k is mapped to a different processor other than i and j, the weight of the new edge is equal to w; otherwise it is equal to w+tp_j.

U4: If there is an edge (j, k) in the original graph with weight w, add an edge (i-j, k) and set its weight to w+tp_i.

The four updating rules preserve the relative timing constraints of the original CDFG. They also enforce the sequencing constraints among the tasks that are mapped to the same PE. It can also be proved that if the original CDFG does not have any positive cycle, then there will be no positive cycle in the new CDFG. Therefore, a feasible schedule can be found for the new CDFG.

```
Update_Slack(i-j) begin
    FIFO-EST = FIFO-LT = {i-j};
    While (FIFO-EST is not empty) begin
        k = pop(FIFO-EST);
        For each g ∈ {successor of k} begin
            If $est_g < est_k + w_{k,g}$
            $est_g = est_k + w_{k,g}$ and push g into FIFO-EST;
        End
    End
    While (FIFO-LST is not empty) begin
        k = pop(FIFO-LST);
        For each g ∈ {predecessor of k} begin
            If $lst_g > lst_k - w_{g,k}$
            $lst_g = lst_k - w_{g,k}$ and push g into FIFO-LST;
        End
    End
End
```

Figure 6 Pseudo code of Update_slack( ).

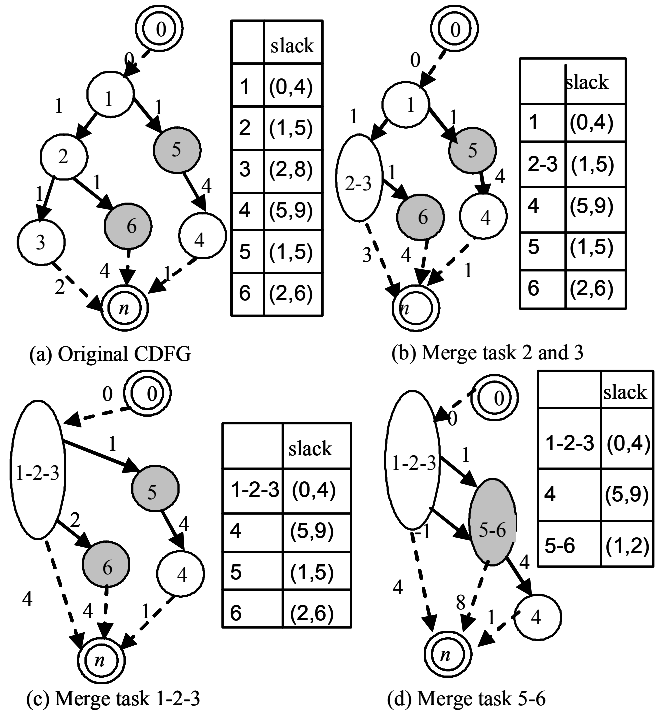
```
Slack Based Task Merging:
M = {Operations pairs that can be merged in the CDFG};
While (M≠Φ) loop begin
    Merge the operation pair (i, j) that has the highest priority;
    Remove vertex i and j as well as the associated edges;
    Create new vertex i-j;
    Update graph;
    Update_Slack(i-j );
    M = {Operations pairs that can be merged in the updated graph};
End
```

Figure 7 Pseudo code of Slack Based Task Merging.

After merging, the slack of the composite task i-j is set to be (max(est_i, est_j-tp_i), min(lst_i, lst_j-tp_i)). The slack of other tasks should also be updated. This is done in the update_slack procedure. In this procedure, two FIFOs are maintained. They are denoted as FIFO-EST and FIFO-LST. FIFO-EST is used to store the operations whose earliest starting time has been changed and FIFO-LST is used to store the operations whose latest starting time has been changed. After operation merge, each of the FIFO has only one element, i-j. While the FIFO-EST is not empty, its first entry k will be popped out. If the earliest starting time of k's successors g is smaller than t_k+$w_{k,g}$, where $w_{k,g}$ is the weight of the edge (k, g), then the est_g is updated and g is pushed into FIFO-EST. Similar operations are performed on FIFO-LST. The pseudo code of Update_slack( ) procedure is given in Figure 6.

When several task pairs are available for merging, we only merge one of them at each time. The two operations that have the longest slack after they merge will have the highest priority. By doing so, we increase the possibility to further merge the composite operations. The overall algorithm for slack based task merging is given in Figure 7.

Example 3 This example shows how slack based task merging works on the original CDFG. Figure 8 (a) gives the original CDFG. The system consists 6 tasks and 2 PEs. We assume that task 1~4 are mapped to PE0 and task 5~6 are mapped to PE1. tp_1~tp_4 are 1 and tp_5~tp_6 are 4. Vertex 0 and n are anchors that represent the triggering operation and deadline. Assume that the triggering event is generated at time 0 and the deadline is at time 10. An edge from vertex 3 to n indicates that the latest starting time of task 3 is 8 (i.e. the deadline of task 3 is 9.) The slack of each task is listed beside the CDFG.



(a) Original CDFG

	slack
1	(0,4)
2	(1,5)
3	(2,8)
4	(5,9)
5	(1,5)
6	(2,6)

(b) Merge task 2 and 3

	slack
1	(0,4)
2-3	(1,5)
4	(5,9)
5	(1,5)
6	(2,6)

(c) Merge task 1-2-3

	slack
1-2-3	(0,4)
4	(5,9)
5	(1,5)
6	(2,6)

(d) Merge task 5-6

	slack
1-2-3	(0,4)
4	(5,9)
5-6	(1,2)

Figure 8 Example of slack based task merging.

Checking the CDFG conditions C1~C4 we found several task pairs that can be merged. They include task 1 and 2, task 2 and 3, task 2 and 4 and task 3 and 4. Because task 2 and 3 gives the largest slack after merging, these two are selected and merged first. We then update the CDFG based on rule U1~U4. The new CDFG is given in Figure 8 (b). After that task 1-2-3, 5-6 are further merged. The resulting CDFGs are given in Figure 8 (c) and (d) respectively. Note that in the last CDFG, from composition task 1-2-3 to composition task 5-6, there are two edges, because 5 must start at least one cycle after 1-2-3 and 6 must start at least 2 cycles after 1-2-3. In this case, only the longest edge will be kept. We cannot merge composite task 1-2-3 with atomic task 4, because condition C3 is not satisfied. □

B. Task Scheduling

Task scheduling in the STM algorithm is a relative easy step because we are not interested in finding the optimal scheduling for minimum resource or latency. The only goal is to find an execution order that satisfies relative timing constraints and sequencing constraints.

It can be proved that, if two composite tasks x and y that are mapped to the same processor cannot be merged, then either there is a directed path between them or their slacks are non-overlapping. Therefore, a simple ASAP algorithm can be used to schedule the tasks. For example, for the task graph given in Figure 8 (d), composite task 1-2-3 will start at time 0, 5-6 will start at time 1 and task 4 will start at time 5.

C. CDFG Unrolling

CDFG unrolling is performed to merge tasks that belong to different triggering cycles. The original CDFG is copied L times and the new task graph $G_u = (V_u, E_u)$ is generated. The lth copy corresponds to the system behavior triggered by the lth triggering event. op_i^l denotes the ith operation in the lth copy and its starting time is denoted as $t_{i,l}$. The E_u contains all the edges that are in the original CDFG and some additional edges. The following edges are added to specify the dependency of the operations in different copies.

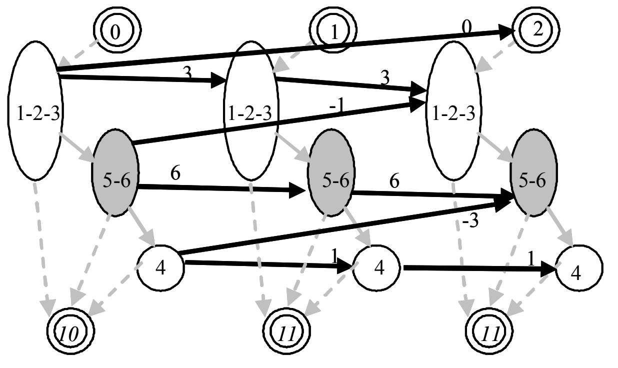


Figure 9 Example of unrolled task graph.

- An edge from op_i^l to op_i^{l+1} is added, $0 \le l < L-1$. Its weight is equal to tp_i. This edge is added because the request queue is serviced at first-in-first-out order, the lth request for op_i must be executed before the (l+1)th request.
- If there is an edge (i, j) in the original CDFG, then an edge is added from op_j^l to $op_i^{l+B_j}$, $0 \le l < L-B_j$. Its weight is equal to $-w_{i,j}$, where $w_{i,j}$ is the weight of edge (i, j) in the original graph. This edge is added because the request queue of task j can only hold up to B_j pending requests. The lth request for op_j must be processed before the (B_j+l)th request is generated. Therefore we must have $t_{i,(l+B_j)} + w_{i,j} \ge t_{j,l}$ which is equivalent to $t_{i,(l+B_j)} - t_{j,l} \ge -w_{i,j}$.

Figure 9 gives the unrolled task graph for example 3. We assume that the application is triggered in every unit time and buffer size for each operation is 2. Note that there are multiple anchors in the unrolled graph and they specify different triggering times and deadlines for different copies. An edge is connecting from the first copy of composite task 1-2-3 to the anchor of time 2. It indicates that the first initiation of composite task 1-2-3 must start before time 2. An edge is connecting from the first copy of composite task 5-6 to the third copy of composite task 1-2-3 and from the first copy of task 4 to the third copy of task 5-6. It indicates that the maximum starting time of the first request for operation 5 and 6 is 1ns after the third initiation of composite task 1-2-3.

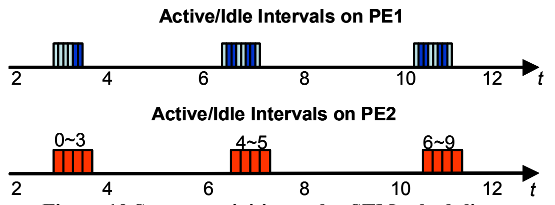


Figure 10 System activities under STM scheduling.

After CDFG unrolling, tasks belongs to different iterations can be merged. As an example, we apply the STM algorithm to schedule the applications specified in the motivational example. Figure 10 shows the system activities. As we can see for this example, the STM algorithm has almost the same task merging quality as the optimal scheduling algorithm.

VI. EXPERIMENTAL RESULTS

To evaluate the performance of the proposed algorithm, two hardware systems are used in the experiments with two and three PEs respectively. In both systems, the processors use point-to-point communication with each other. We assume that each processor has two power modes, active and sleep.

Table 2 Summary of Task Graphs.

# of PEs	3						2					
work load	High		Medium		Low		High		Medium		Low	
TG	γ	σ	γ	σ	γ	σ	γ	σ	γ	σ	γ	σ
1	0.72	0.08	0.43	0.05	0.21	0.03	0.68	0.08	0.41	0.05	0.20	0.03
2	0.48	0.14	0.29	0.09	0.14	0.04	0.60	0.14	0.36	0.09	0.17	0.04
3	0.56	0.30	0.34	0.18	0.17	0.09	0.47	0.30	0.42	0.18	0.21	0.09
4	0.83	0.26	0.50	0.16	0.24	0.08	0.70	0.26	0.41	0.16	0.20	0.08
5	0.57	0.12	0.34	0.08	0.16	0.04	0.86	0.12	0.51	0.08	0.25	0.04
6	0.41	0.17	0.25	0.10	0.12	0.05	0.77	0.17	0.46	0.10	0.23	0.05
7	0.74	0.22	0.44	0.13	0.22	0.07	0.68	0.22	0.40	0.13	0.20	0.07

In the experiments, we compare the performance of STM algorithm, minimum latency scheduling (MLS) algorithm, BP-I, BP-II and BP-EDF algorithms. As we have mentioned in Section IV, the quality of a scheduling algorithm is measured using two parameters, the average number of idle intervals in a triggering cycle (*P*) and deadline miss ratio (*D*). The parameter *P* is proportional to the energy overhead of power mode switching and the parameter *D* is related to system performance. An event driven simulator is developed that simulates systems that are running under different scheduling algorithms. For all the burst processing on-line algorithms, the deadline scaling factor α is set to 0.8.

Table 3 Quality of Scheduling (High Workload, 3 PEs).

Algorithms	MLS		STM		BP-I		BP-II		BP-EDF	
TG	P	D (%)	P	D (%)	P	D (%)	P	D (%)	P	D (%)
1	2.99	0	0.37	0	0.77	0	0.77	0	0.77	0
2	3.00	0	1.37	0	1.70	19.2	0.86	8.0	1.23	0
3	3.00	0	1.00	0	1.31	10.8	0.86	6.0	0.86	0
4	2.99	0	0.16	0	0.62	7.7	0.27	9.4	0.38	0
5	5.26	0	3.14	0	3.33	36.2	3.00	0	3.00	0
6	5.00	0	1.51	0	2.24	24.6	2.01	8.3	2.01	0
7	4.66	0	4.02	0	5.45	13.1	2.52	9.7	2.67	0

Table 4 Quality of Scheduling (Medium Workload, 3 PEs).

Algorithms	MLS		STM		BP-I		BP-II		BP-EDF	
TG	P	D (%)	P	D (%)	P	D (%)	P	D (%)	P	D (%)
1	3.00	0	0.51	0	0.96	0	0.96	0	0.96	0
2	4.00	0	2.26	0	1.47	9.6	1.33	0.4	1.11	0
3	3.00	0	0.75	0	1.96	0	1.23	0	1.47	0
4	3.00	0	0.33	0	0.71	10.3	0.49	0	0.37	0
5	7.00	0	3.00	0	4.66	11.0	3.00	0	3.00	0
6	5.00	0	2.00	0	6.00	0	2.51	8.2	2.51	0
7	7.00	0	2.55	0	7.91	0	3.98	0	3.98	0

Table 5 Quality of Scheduling (Low Workload, 3 PEs).

Algorithms	MLS		STM		BP-I		BP-II		BP-EDF	
TG	P	D (%)	P	D (%)	P	D (%)	P	D (%)	P	D (%)
1	3.00	0	0.75	0	0.80	0	0.80	0	0.80	0
2	4.00	0	1.51	0	1.24	0	1.45	0	1.37	0
3	3.00	0	0.75	0	1.96	0	0.99	0	0.99	0
4	3.00	0	0.49	0	0.78	0	0.48	0	0.48	0
5	7.00	0	2.02	0	5.79	0	3.00	0	3.00	0
6	5.00	0	1.51	0	6.98	0	3.50	0	3.50	0
7	7.00	0	2.52	0	8.60	0	3.96	0	3.96	0

Table 6 Quality of Scheduling (High Workload, 2 PEs).

Algorithms	MLS		STM		BP-I		BP-II		BP-EDF	
TG	P	D (%)	P	D (%)	P	D (%)	P	D (%)	P	D (%)
1	1.51	0	0.40	0	0.75	08.0	0.46	0. 7	0.46	0
2	2.69	0	1.89	0	0.98	0	0.99	0	1.47	0
3	2.01	0	0.38	0	1.65	0	0.50	3.8	0.53	0
4	3.00	0	1.01	0	1.23	13.7	0.50	9.0	0.66	0
5	3.00	0	7.00	0	1.36	39.6	0.68	11.0	0.62	0
6	3.05	0	2.96	0	5.03	13.5	3.01	0	3.00	0
7	3.00	0	3.00	0	5.00	3.3	2.01	6.6	2.01	0

Table 7 Quality of Scheduling (Medium Workload, 2 PEs).

Algorithms	MLS		STM		BP-I		BP-II		BP-EDF	
TG	P	D (%)	P	D (%)	P	D (%)	P	D (%)	P	D (%)
1	3.00	0	1.01	0	1.00	8.3	1.76	0.3	0.76	0
2	4.00	0	2.02	0	1.31	0	1.22	0	1.31	0
3	2.01	0	0.99	0	2.74	0	0.75	0	0.87	0
4	4.00	0	1.01	0	1.97	0	0.50	3.6	0.79	0
5	3.00	0	1.50	0	3.17	1.8	2.00	0	2.00	0
6	5.00	0	2.01	0	6.98	2.7	4.00	0	4.00	0
7	3.00	0	1.50	0	7.48	0	2.01	0	2.01	0

Table 8 Quality of Scheduling (Low Workload, 2 PEs).

Algorithms	MLS		STM		BP-I		BP-II		BP-EDF	
TG	P	D (%)	P	D (%)	P	D (%)	P	D (%)	P	D (%)
1	3.00	0	1.01	0	1.32	0	1.00	0	1.00	0
2	4.00	0	2.00	0	1.32	0	1.63	0	1.62	0
3	2.01	0	1.00	0	3.12	0	0.99	0	0.99	0
4	4.00	0	0.75	0	1.97	0	0.98	0	0.99	0
5	3.00	0	0.70	0	4.79	0	2.00	0	2.00	0
6	5.00	0	2.03	0	7.63	0	3.99	0	3.99	0
7	3.00	0	1.50	0	7.97	0	3.00	0	2.00	0

Table 3~8 report the quality of five scheduling algorithms when applied to schedule different applications in the different hardware systems. In most of the cases, STM, BP-I, BP-II and BP-EDF algorithms result in smaller *P* value (i.e. the number of idle time intervals) than the MLS algorithm. Figure 11 gives the average reduction of the *P* value for STM, BP-I, BP-II and BP-EDF algorithms compared to the minimum latency scheduling. The BP-II and BP-EDF always have very similar *P* value. The STM algorithm outperforms these two when the workload is low or medium. More specifically, when the workload is low, the average *P* value reduction of STM algorithm is 24% and 31% more than that of BP-EDF and BP-II algorithms, respectively. When the workload is medium, the average *P* value reduction of STM algorithm is 10% and 14% more than that of BP-EDF and BP-II algorithms, respectively. Overall, comparing to the MLS algorithm, the STM, BP-II and BP-EDF have 56% reduction of the *P* value while the BP-I has only 16% reduction, in average.

Experimental results also show that, neither STM nor BP-EDF algorithm causes any task deadline misses. The BP-I and BP-II scheduling have task deadline miss in most of the test cases when the workload is high or medium. The task deadline miss ratio increases when the workload increases. Figure 12 gives the average deadline miss ratio for these two algorithms in different scenarios.

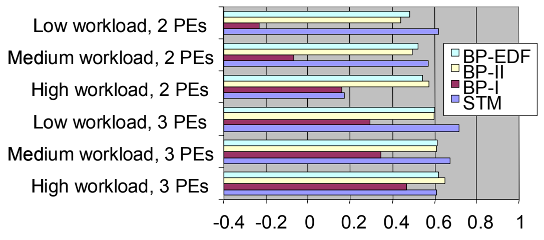


Figure 11 Average reduction of number of idle intervals.

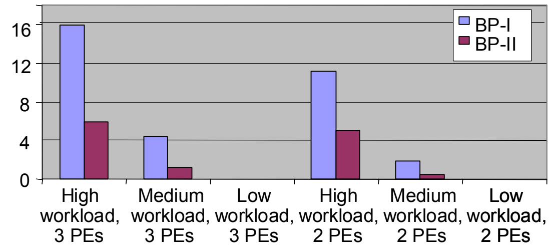


Figure 12 Average deadline miss ratio (%).

For the STM scheduling, we unroll the task graph 100 times. We are able to find repeating scheduling pattern without increasing the length of the unrolled DFG. Table 9 gives the length of the repeating period for the low workload task graphs in the unit of triggering cycles.

Table 9 Repeating period of STM scheduling (Low Workload).

TG	1	2	3	4	5	6	7
2 PEs	6	2	4	4	2	2	2
3 PEs	4	4	4	8	4	10	6

VII. CONCLUSIONS

We have proposed a method of task merging for dynamic power management in a real-time system with multiple processing elements. It shows that, with good task scheduling, the energy and delay overheads due to power mode switching in the DPM scheme can be reduced significantly. Two new task scheduling algorithms are proposed to minimize the number of idle time intervals under the deadline and precedence constraints. A simple DPM policy is then used to reduce the energy during the idle intervals. Experimental results show that, comparing to the DPM schemes without proper task scheduling, the proposed method reduces the number of power mode switches by 56% at average.

REFERENCES

[1] http://pasta.east.isi.edu/

[2] M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Transactions on VLSI Systems*, Vol. 4, pp. 42–55, March 1996.

[3] C-H. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *Proceeding of International Conference on Computer-Aided Design*, November 1997.

[4] L. Benini, G. Paleologo, A. Bogliolo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Transactions on Computer-Aided Design*, Vol. 18, pp. 813–33, June 1999.

[5] Q. Qiu, Q Wu and M. Pedram, "Stochastic modeling of a power-managed system-construction and optimization," *IEEE Transactions on Computer-Aided Design*, Vol. 20, pp. 1200-1217, October 2001.

[6] B. Zhai, D. Blaauw, D Sylvester, and K Flautner, "Theoretical and practical limits of dynamic voltage scaling," *Proceedings of Design Automation Conference*, 2004.

[7] L. Benini, A. Bogliolo and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration Systems*, Volume 8, Issue 3, pp. 299-316, June 2000.

[8] B. Schott, M. Bajura, J. Czarnaski, J. Flidr, T. Tho and L. Wang, "A modular power-aware microsensor with >1000X dynamic power range," *International Symposium on Information Processing in Sensor Networks*, April 2005.

[9] K. Morris, "Power: suddenly, we care," *FPGA and Programmable Logic Journal*, April 2005.

[10] J. Liu, P. Chou, N. Bagherzadeh, and F. Kurdahi, "A constraint-based application model and scheduling techniques for power-aware systems," *International Conference on Hardware Software Codesign*, April 2001.

[11] Y. Lu, L. Benini, and G. D. Micheli, "Low-power task scheduling for multiple devices," *International Workshop on Hardware/Software Codesign*, 2000.

[12] P. Rong and M. Pedram, "Hierarchical dynamic power management with application scheduling," *Proc. of Symp. on Low Power Electronics and Design,* Aug. 2005.

[13] G. D. Micheli, "Synthesis and optimization of digital circuits," *McGraw-Hill, Inc*, 1994.

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