2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications

Energy Minimizing for Parallel Real-Time Tasks Based on Level-Packing

Huiting Xu Fanxin Kong Qingxu Deng
Dept. of Computer Science & Technology, Northeastern University, China
xuht_neu@hotmail.com, kongfx@ise.neu.edu.cn, dengqx@mail.neu.edu.cn

***Abstract*—While much work has addressed energy minimizing problem of *real-time sequential tasks*, little has been done for the *parallel real-time task* case. In this paper, based on level-packing, we study energy minimization problem for parallel task systems with discrete operation modes and under timing constraints. For tasks with fixed (variable) parallel degrees, we first formulate the problem as a 0-1 Integer Linear Program (0-1 ILP), and then propose a polynomial-time complexity two-step (three-step) heuristic to determine task schedule and frequency assignment (and the task parallel degree). Our simulation result shows that the heuristics consume nearly the same energy as do 0-1 ILPs.**

I. Introduction

To prolong the battery lifetime in battery-powered embedded systems and to cut the electricity bill of large-scale server systems, energy efficiency has ever been an important issue over the past decades in both academic and industry communities. Dynamic voltage scaling is recognized as one of the most effective and widely-used hardware methodologies, which dynamically regulates the system operation mode and the supply voltage/frequency for energy saving. In addition, multiprocessor/multicore platforms have been also adopted to balance the system performance and the power density that dramatically grows with higher voltage and frequency. As real-time constraints are required to maintain the system dependability, one of key issues in embedded systems is to minimize the energy consumption with timing guarantees.

The energy-efficient scheduling problem has been extensively studied for multiprocessor real-time systems, e.g., [1]–[9]. Despite adopting different assumptions on task and power models, all of these previous work focused on the *sequential task model* which restricts that an individual task can *not* run on more than one processor at the same time but on only one processor at a time. This model can not represent some key characteristics of multiprocessor real-time systems.

Due to the boosting demand for system performance, *parallel processing* allowing that an individual task executes in multiple processors simultaneously, has been widely used in high performance computing area, such as scientific computing which divides large amount of data/tasks across processors. Some researchers discussed the tread-off between performance and energy scalability in general-purpose parallel task systems. The authors in [10] addressed the interplay between parallelization, performance and energy consumption theoretically, while the authors in [11] conducted evaluations on energy scalability of parallel algorithms methodologically and experimentally while satisfying some performance requirements.

In recent, parallel processing has appeared in embedded real-time systems, such as robot arm dynamics [12] and video processing [13]. Unlike general-purpose systems, there are timing constraints such as release times and deadlines that must be guaranteed in real-time systems. Some research has addressed the scheduling problem that combines timing constraints and the task parallelization but no energy consumption issues, such as some research work which addressed the real-time scheduling problem of parallel tasks under gang scheduling policy[1] [14]–[18] and under multi-thread scheduling policy [18], and real-time divisible load theory [19], [20] where the computational loads can be arbitrarily divided.

Blindly adopting the sequential task energy minimization approaches without considering parallelization, or the parallel task scheduling approaches without considering processor voltage/ frequency optimization, can result in a waste of energy. This work aims at exploring energy-efficient scheduling for parallel real-time tasks. Specially, we study the energy minimization problem for parallel tasks when combining timing constraints and gang scheduling scheme. The only one closely related work is the research effort [21], which otherwise assumes processors regulating frequency continuously, and uses sub-linear speedup ratio model where the rate of increment in speedup is lower than the rate of increment in the number of the assigned processors. The Algorithm BS in [21], under the two assumptions, would derive the optimal solution for any level-packed schedule.

However, in this paper, we will address a more practical and tough case where processors run on discrete number of valid operation modes and make no constraints on processors' speedup model, i.e., arbitrary speedup ratio. Firstly, we adopt level-packing [22] as basic task scheduling policy for two reasons. It first has very good performance when minimizing schedule length, which tends to derive more slack time for task stretching or lowering frequency to save energy. Then, it can incorporate with frequency scaling scheme very well due to the concept of *level*, which also can be seen from our approaches. Secondly, we focus on the realistic case that a processor can only operate at discrete set of valid operational modes and do not make any restriction on the form of power function and task execution time function to the processor frequency. As technical contributions, for tasks with fixed (variable) parallel degrees, we first formulate the energy minimizing problem as a 0-1 Integer Linear Program (0-1 ILP), and then propose an efficient two-step (three-step) algorithm to determine the task schedule and frequency assignment (and the task parallel degree). Our simulation result shows that the heuristics consume nearly the same energy as 0-1 ILPs which derive optimal results for the level-packing based energy minimizing problem.

[1]Gang scheduling is one of the most efficient parallel processing schemes, in which processors execute a task in unison. In other words, the threads of one task should be the same time quantum, start at the same time and execute at the same pace.

978-0-7695-4824-1/12 $26.00 © 2012 IEEE
DOI 10.1109/RTCSA.2012.10

The rest of this paper is organized as follows. Section II provides the system models and defines the problem. Section III and Section IV propose 0-1 ILP formulations and efficient heuristics for two different parallel task models respectively. Section V presents the experimental results. Section VI concludes this paper and points out our future work.

II. Problem Setting

A. System Model

We study a real-time application consisting of a set of independent parallel tasks $\Gamma = \{\tau_1, ..., \tau_N\}$ with a deadline of T, i.e., frame based task, under a homogenous multiprocessor/multicore system of M identical processers/cores. Each processor can dynamically adjust its frequency and voltage on application requests, and has Q active states, i.e., a discrete set of Q valid frequency and voltage pairs denoted as $\{(F_1, V_1), ..., (F_Q, V_Q)\}$, $F_1 < ... < F_Q$. For each frequency/voltage pair, there is a power consumption associated with it, and thus we have a set of power values: $\{P_1, ..., P_Q\}$. We adopt inter-task DVS policy, i.e, each task τ_n has only one frequency assignment.

There is a vector of *scaling factor* $\{\Theta_n^q | q \in [1, Q]\}$ for each task τ_n, and its execution time is $\Theta_n^q C_n$ when τ_n runs at frequency F_q, where C_n is the task execution time at the maximum frequency F_Q. In fact, for each task τ_n, we only have to consider the frequency/voltage pair satisfying that lower frequency (voltage) requires less energy, i.e., $\Theta_n^q P_q < \Theta_n^{q+1} P_{q+1}$. If one pair has low frequency but high energy, the pair can be removed safely without harming energy saving. Without loss of generality, we assume that all tasks has Q such pairs where lower frequency requires less energy consumption.

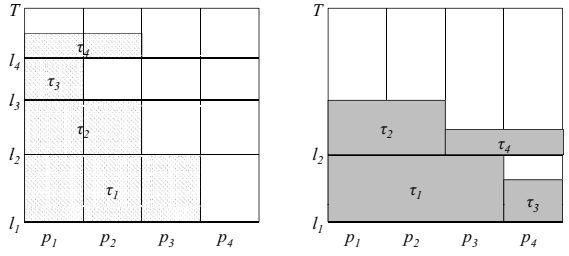
Another parameter associated with each task τ_n is a vector of $\{\Upsilon_n^m | m \in [1, M]\}$, which denotes the *speed up ratio* obtained by running τ_n on m processors. The number m of processors assigned to τ_n is also called *parallel degree* of the task.[2] In sum, when running τ_n on m processors simultaneously at frequency F_q, the execution time equals to $\frac{\Theta_n^q}{\Upsilon_n^m} C_n$ while the energy consumption is $\frac{\Theta_n^q}{\Upsilon_n^m} m P^q C_n$.

According to the task flexibility, a parallel task can be divided into three categories: (i) *rigid*, the parallel degree of the task is fixed a priori [28], [29]; (ii) *moldable*, the parallel degree of the task is determined when the task is to be activated, and can't change throughout its execution [15], [23], [25], [26]; (iii) *malleable*, the parallel degree of the task can change during its execution [24], [27], [30]. Since we use gang scheduling scheme, the execution of one rigid or moldable task forms a rectangle of processor×time space while the execution of one malleable task corresponds to a union of rectangles. We focus on the rigid and moldable tasks in this paper. Note that since each frame has the same task schedule, we have to determine the parallel degree of a moldable task only once.

[2]We make no assumption on the speedup ratio, such as linear [23], [24] or sub-linear [15], [25]–[27], i.e., our approach can adopt to arbitrary speedup ratio.

B. Level-Packing

As mentioned above, level-packing is an effective approach to schedule parallel tasks for schedule length minimization, which packs the tasks (rectangles) in levels. Each level is determined by the horizontal line drawn on the top of the rectangle with the maximum height packed on the previous level. The highest rectangle in one level also determines the height of the level. See Figure 1(b) has two levels: ℓ_1 and ℓ_2. There are some classical heuristics for level-packing: *First Fit Decreasing Height (FFDH)*, *Best Fit Decreasing Height (BFDH)* and *Next Fit Decreasing Height (NFDH)*. They all first sort tasks in a non-increasing order of the execution time h_i, and then pack tasks according to the different fit rules. Details of the three algorithms can be found in [22]. We adopt level-packing to schedule a parallel task set. It first has very good performance to minimize makespan, which tends to derive more slack time for stretching tasks, and then it can incorporate with frequency scaling scheme very well due to the concept of level.



(a) Assume 4 levels in ILP (b) 2 levels in solution

Fig. 1. An application of four tasks $\{\tau_1, \tau_2, \tau_3, \tau_4\}$ under a multiprocessor/multicore with four cores. Shadow rectangle: packing in assumption. Solid rectangle: packing in solution. The horizontal axis is the processor dimension and the vertical axis is the time dimension.

C. Problem Statement

We address energy-efficient scheduling for a real-time application consisting of N independent parallel tasks with a deadline of T in homogeneous multiprocessors/multicores with discrete operation modes. We aims at minimizing the energy consumption for parallel tasks without violating the timing constraints based on level-packing under gang scheduling policy. The optimization problem has two dimensions: task scheduling and frequency assignment for rigid tasks, while it has another additional dimension of the parallel degree allotted to each task for moldable tasks. Both of the two problems are NP-hard, since both the parallel task scheduling problem based on level-packing and the DVS problem targeting on the processor with discrete operation modes, are well known to be NP-hard in general. Hence, the objective of this paper is to formulate them as *0-1 Integer Linear Programming* (0-1 ILP) problems as well as develop efficient heuristics.

III. Rigid Tasks

The energy minimizing problem for rigid real-time tasks has two dimensions: task scheduling and frequency assignment. Now we will first formulate a 0-1 ILP to deal with the two dimensions together, and then present an efficient two-step heuristic to solve them respectively.

We say a task *initializes* one level if its execution time is the maximum in the level, and call the task as *key task*. In the worst case, each task initializes a level and there are N levels. Hence, we assume N levels in our 0-1 ILP formulation. There may be less than N levels in the solution, and if there is no task in one level, the height of the level equal to zero. Without loss of generality, if τ_n is in level ℓ_n, we can say τ_n initializes the level, since we can insert some levels without any task in-between levels with tasks to make the key tasks index equal the level index, i.e., τ_n to ℓ_n. For example, comparing Figure 1 (a) and (b), we can see that τ_1 and τ_2 are the key tasks of ℓ_1 and ℓ_2 respectively, and the heights of ℓ_3 and ℓ_4 both equals to zero. In all, there are 2 levels in the solution.

A. 0-1 ILP

For each task τ_n, we define several binary variables for determining the level and the frequency, at which τ_n executes.

- $x_{niq} = 1$ iff τ_n executes at level ℓ_i and frequency F_q; note that among these binary variables, x_{iiq} (where the first two indexes equal) is special, since it represents whether τ_i initializes level ℓ_i, i.e., $x_{iiq} = 1$ iff τ_i initializes level ℓ_i and runs at frequency F_q.

Each task τ_n can locate at only one level and run on only one frequency, whether τ_n initializes one level or not. Hence:

$$\forall n \in [1, N], \sum_{i=1}^{N} \sum_{q=1}^{Q} x_{niq} = 1 \tag{1}$$

The sum of parallel degrees of tasks in each level ℓ_i should be less than or equal to the number of processors M:

$$\forall i \in [1, N], \sum_{n=1}^{N} \sum_{q=1}^{Q} x_{niq} w_n \leq M \tag{2}$$

Note the parallel degree w_n of τ_n is constant for the rigid task.

If τ_i initializes level ℓ_i, the execution time of τ_i is always largest (highest) among all tasks in ℓ_i no matter before or after frequency assignment. Hence, we have:

$$\forall i, n \in [1, N], \sum_{q=1}^{Q} x_{niq} \frac{\Theta_n^q}{\Upsilon_n^{w_n}} C_n \leq \sum_{q=1}^{Q} x_{iiq} \frac{\Theta_i^q}{\Upsilon_i^{w_i}} C_i \tag{3}$$

To meet deadline, the sum of heights of levels or the sum of execution times of key tasks, is less than or equal to T:

$$\sum_{i=1}^{N} \sum_{q=1}^{Q} x_{iiq} \frac{\Theta_i^q}{\Upsilon_i^{w_i}} C_i \leq T \tag{4}$$

The optimizing objective is to minimize the total energy consumption:

$$\min \quad \sum_{n=1}^{N} \sum_{i=1}^{N} \sum_{q=1}^{Q} x_{niq} \frac{\Theta_n^q}{\Upsilon_n^{w_n}} w_n P_q C_n \tag{5}$$

Conditions (1) to (4) forms a constraint set, and combined with the optimization objective in Condition (5), it forms a 0-1 ILP problem.

B. Efficient Heuristics

The heuristic contains of two steps: schedule task and assign frequency. We first adopt the efficient level-packing algorithms, such as FFDH, NFDH or BFDH to schedule the parallel task set.

After task scheduling, there are I levels which may be less than N, and K_i tasks in each level ℓ_i where $\sum_{i=1}^{I} K_i = N$. We re-index tasks in each level ℓ_i as $\tau_{ik \in [1,K_i]}$ in this subsection. According to the packing strategies above, τ_{i1} always has the largest execution time, thus the height of ℓ_i equals to $t_i(z_{i1}) = \frac{\Theta_{i1}^{z_{i1}}}{\Upsilon_{i1}^{w_{i1}}} C_i$ when τ_{i1} runs at frequency $F_{z_{i1}}$. As mentioned before, we only consider the voltage/frequency pairs where lower frequency requires less energy. As the parallel degree is fixed for the rigid task case, to minimize energy, the remain task $\tau_{ik \in [2,K_i]}$ in ℓ_i should stretch as close as to the level height, and should run at the lowest frequency that makes the corresponding execution time just less than the level height. According to this, the frequencies and energy consumptions of other tasks in ℓ_i can be easily obtained, if z_{i1} is determined. This is also the explanation for Line 17 in Algorithm 1 and Line 12 in Algorithm 2. Hence, the total energy consumption of level ℓ_i can expressed by frequency of the key task τ_{i1}. We use $e_i(z_{i1})$ to denote the energy consumption of level ℓ_i. The $e_i(z_{i1})$ value for all levels and all voltage/frequency pairs can be calculated in $O(QN)$ time in the worst case when each task forms a level. It's easy to know $e_i(1) < ... < e_i(Q)$.

Then, for frequency assignment, we will present two greedy algorithms: *Stretch* and *Shrink*.

Algorithm 1 Stretch

Input: the derived schedule by FFDH, NFDH or BFDH;
Output: the frequency assignment of each task;

```
1:  L ⇐ {ℓ_1, ..., ℓ_I};
2:  if Σ_{i=1}^{I} t_{i1}(Q) > T then
3:      return no feasible solution;
4:  end if
5:  while L ≠ ∅ do
6:      find level ℓ_i with largest Δ_{iq} in L;
7:      z_{i1} ⇐ q − 1;
8:      if Σ_{i=1}^{I} t_{i1}(z_{i1}) > T then
9:          z_{i1} ⇐ q;
10:         L ⇐ L − ℓ_i;
11:         continue;
12:     end if
13:     if z_{i1} = 1 then
14:         L ⇐ L − ℓ_i;
15:     end if
16: end while
17: calculate all frequencies of the remain tasks τ_{ik} accordingly;
18: return the frequency assignment of each task;
```

1) Stretch: For the first heuristic, we begin with assigning all frequencies of $\tau_{i1}, i \in [1, I]$ to the maximum value of F_Q ($z_{i1} = Q$). If the total level height $\sum_{i=1}^{I} t_i(Q)$ is greater than the deadline T, no feasible solution exists. Otherwise, if $\sum_{i=1}^{I} t_i(Q) \leq T$, we have to *stretch* the schedule as close as to the deadline. For such purpose, we will iteratively increase the height of the level ℓ_i with the largest value of $\Delta_{iq} = \frac{e_i(q) - e_i(q-1)}{t_i(q-1) - t_i(q)}$, i.e., search for the level with the largest energy change but the least height change by changing z_{i1} from q

to $q-1$. Note that during the iteration, it may happen that some τ_{i1} already runs at F_1 ($z_{i1}=1$) and can not lower the frequency or increase the level height, so we will not consider the level any more. See Algorithm 1.

2) Shrink: Instead of assigning the frequency from the maximum value F_Q, the second heuristic starts from an opposite direction. We initialize the frequency of each task to F_1, and then *shrink* the schedule by decreasing the level height until $\sum_{i=1}^{I} t_i(z_{i1}) \leq T$. We will iteratively decrease height of the level ℓ_i with the least value of Δ_{iq}. See Algorithm 2.

Algorithm 2 Shrink

Input: the schedule by FFDH, NFDH or BFDH;
Output: the frequency assignment of each task;
1: $L \Leftarrow \{\ell_1, ..., \ell_I\}$;
2: **if** $\sum_{i=1}^{I} t_{i1}(Q) > T$ **then**
3: **return** no feasible solution;
4: **end if**
5: **while** $L \neq \emptyset$ & $\sum_{i=1}^{I} t_{i1}(z_{i1}) > T$ **do**
6: find level ℓ_i with least $\Delta_{i(q+1)}$ in L;
7: $z_{i1} \Leftarrow q+1$;
8: **if** $z_{i1} = Q$ **then**
9: $L \Leftarrow L - \ell_i$;
10: **end if**
11: **end while**
12: calculate all frequencies of the remain tasks τ_{ik} accordingly;
13: **return** the frequency assignment of each task;

The time complexity of the first step is $O(NlogN)$ due to the task sorting in the FFDH, NFDH or BFDH. Calculating $e_i(z_{i1})$ values for all levels and all voltage/frequency pairs need $O(QN)$ time. Both of Algorithm 1, 2 is $O(N)$. So, the time complexity of the overall two-step algorithm is $O(QN)$.

IV. Moldable Tasks

Compared to the rigid task case, the energy minimization of moldable tasks becomes even harder due to the freedom of the parallel degree allotted to each task, i.e., we have to determine the number of processors each task runs on, as well as task schedule and frequency assignment. In this section, we first formulate a 0-1 ILP to determine the three dimensions together, and then present an efficient three-step algorithm to solve them respectively.

A. 0-1 ILP

Similar to Subsection III-A, we also assume there are N levels. For each task τ_n, we define several binary variables for determining the level, the parallel degree and the frequency, at which τ_n executes.

- $x_{nimq} = 1$ iff τ_n executes at level ℓ_i, m processors and frequency F_q; also note that among these binary variables, x_{iimq} represents whether τ_i initializes level ℓ_i, i.e., $x_{iimq} = 1$ iff τ_i initializes level ℓ_i, and runs at m processors and frequency F_q.

Each task τ_n can execute at only one level, only one parallel degree and only one frequency, whether τ_n initializes one level or not. Hence, we have

$$\forall n \in [1,N], \sum_{i=1}^{N}\sum_{m=1}^{M}\sum_{q=1}^{Q} x_{nimq} = 1 \tag{6}$$

For the similar reasons of Conditions (2)(3)(4), we have

$$\forall i \in [1,N], \sum_{n=1}^{N}\sum_{m=1}^{M}\sum_{q=1}^{Q} x_{nimq} m \leq M \tag{7}$$

$$\forall i,n \in [1,N], \sum_{m=1}^{M}\sum_{q=1}^{Q} x_{nimq}\frac{\Theta_n^q}{\Upsilon_n^m}C_n \leq \sum_{m=1}^{M}\sum_{q=1}^{Q} x_{iimq}\frac{\Theta_i^q}{\Upsilon_i^m}C_i \tag{8}$$

$$\sum_{i=1}^{N}\sum_{m=1}^{M}\sum_{q=1}^{Q} x_{iimq}\frac{\Theta_i^q}{\Upsilon_i^m}C_i \leq T \tag{9}$$

Note that m is now the parallel degree to be chosen in the moldable task case.

The optimization objective is to minimize the total energy consumption:

$$\min \quad \sum_{n=1}^{N}\sum_{i=1}^{N}\sum_{m=1}^{M}\sum_{q=1}^{Q} x_{nimq}\frac{\Theta_n^q}{\Upsilon_n^m} m P_q C_n \tag{10}$$

Conditions (6) to (9) forms a constraint set, and combined with the optimization objective in Condition (10), it forms a 0-1 ILP problem.

B. Efficient Heuristics

A vector $\mathbf{W} = \{w_1, ..., w_N\}$ is defined as an *allotment* of processors to the N tasks, where w_n is the parallel degree allotted to τ_n. Since one rigid task set can be seen as an allotment of the corresponding moldable task set, we can use the algorithm in Section III-B as subroutines. The overall three-step algorithm consists of:

- ***Parallel Degree Allotment***: Each parallel degree w_n of task τ_n can be anyone of the M parallel degrees, so there are M^N candidate allotments totally. We propose in what follows an efficient heuristic to generate at most $N(M-1)+1$ candidate allotments instead of dealing with the exponential search space.
- ***Task Scheduling***: Pack each allotment by a level-packing heuristic algorithm, such as FFDH, BFDH or NFDH.
- ***Frequency Assignment***: For each allotment and the corresponding schedule, adopt the Algorithm 1 (Algorithm 2) to derive one solution.

The overall algorithm returns the allotment, task schedule and frequency assignment that derives the minimum energy consumption among all these candidate allotments.

1) Allotment Algorithm: To reduce the energy consumption, it may be better to derive allotments with small workload and short execution time for each task. However, in general, the task workload and execution time does not always vary with the parallel degree in the same direction, i.e., smaller workload does not accompany shorter execution time, and vice versa. Hence, we propose an algorithm to generate a set of candidate allotments that contains some with small task workload, some with short task execution time and some in-between. See Algorithm 3. The algorithm iteratively reduces

Algorithm 3 Allotment

Input: C_n and vector $\Upsilon_n^{w_n}$ of each task τ_n;
Output: a set A of candidate allotments;

1: $\Gamma = \{\tau_1, ..., \tau_N\}$, $i = 1$;
2: sort the workload of each task τ_n in an increasing order and re-denote it as $\{w_{n1} \times h_{n1}, ..., w_{nM} \times h_{nM}\}$;{$w_{nm}$ denotes the parallel degree and h_{nm} denotes the corresponding execution time, i.e., $\frac{C_n}{\Upsilon_n^{w_{nm}}}$}
3: $A_i \Leftarrow \{w_{11}, ..., w_{N1}\}$, i++;
4: **while** $\Gamma \neq \emptyset$ **do**
5: find the largest h_{nm} among those with corresponding w_{nm} in A_i and task τ_n in Γ;
6: find such t that $h_{nt} < h_{nm} \wedge m < t$;
7: **if** $t \neq \emptyset$ **then**
8: $w_{nm} \Leftarrow w_{n(min\{t\})}$;
9: $A_i \Leftarrow the\ new\ allotment$, i++;
10: **else**
11: $\Gamma \Leftarrow \Gamma - \tau_n$;
12: **end if**
13: **end while**
14: **return** A;

the execution time of the longest task in the current allotment according to the sorted workload list.

Recall that the time complexity is $O(QN)$ for the latter two steps, so the time complexity of the overall three-step algorithm is $O(QMN^2)$.

V. Simulation Results

We performed a series of simulations to evaluate the energy efficiency of our approaches. Since no comparison work has addressed the energy minimizing problem of rigid (moldable) tasks from all the two (three) dimensions: task scheduling, frequency assignment (and the parallel degree allotment) in multiprocessors with discrete operation modes, we mainly compared the heuristics with the 0-1 ILPs which derive the optimal solutions for the level-packing based energy-efficient scheduling problem.

In our simulation, we considered multiprocessor/multicore systems with M equal to 4, 8, 16 and 32 cores respectively. The realistic processor parameters (frequencies and power consumptions) of IBM PowerPC 750 and Intel XScale in [31] were employed, and we only present the simulation results for Intel XScale cores due to the similar results of IBM PowerPC 750. The period (deadline) of the application was set to 100 time units. For each task τ_n, its execution time C_n at maximum frequency F_Q was randomly generated in the range of $[0.01T, T]$. Recall that the proposed algorithms in this paper dose NOT restrict to any task execution time function to the frequency. However, for simplicity, the execution time at frequency F_q was then computed as $\frac{F^Q}{F^q}C_n$ ($\Theta_n^q = \frac{F_Q}{F_q}$), i.e., the execution time was assume to be inversely proportional with the frequency. Note that since the lowest frequency level of Intel XScale does not satisfy that lower frequency requires less energy due to this task setting, we focus on its four highest frequency levels. For speed up ratio vector, Υ_n^1, when τ_n runs at one processor, was set to 1, and then other ratios was randomly generated with $\Upsilon_n^{m-1} \leq \Upsilon_n^m \leq m,\ m \in [2, M]$ satisfied, which is realistic, due to the communication overheads between threads, but the proposed algorithms could deal with arbitrary speedup ratio. The task number N was set to range from $\frac{M}{2}$ to $\frac{3M}{2}$, due to the larger task number leading to infeasible solutions. We only plot the results of the approaches combining FFDH, since similar conclusions can be draw from other approaches combining BFDH or NFDH. The energy consumptions in each figure were normalized to that of the heuristics incorporating with algorithm *Shrink* for each setting, respectively. We record the energy consumption for each simulation run, and plot the results averaged over 500 runs. The 0-1 ILPs are solved by ILOG CPLEX solver. The experiments are performed on a Windows PC with an Intel Core2 2.83GHZ 32-bit processor and 2GB main memory.

A. Rigid Task Case

In this experiment, we evaluate the energy efficiency of different approaches for rigid real-time tasks. Each rigid task randomly chooses one of the M parallel degrees and also the corresponding execution time. As shown in Figure 2, although the energy difference between the heuristics and 0-1 ILP slightly increases as the number of task grows, the approach combining with Algorithm *Stretch* performs nearly the same as the 0-1 ILP solution, while the one with Algorithm *Shrink* performs slightly worse. The reason is that the heuristic *Stretch* aims at minimizing the energy consumption in each iteration, which complies with the goal of the initial problem. While the heuristic *Shrink* follows a different direction, which may lead to irreversible energy consumption jump.

B. Moldable Task Case

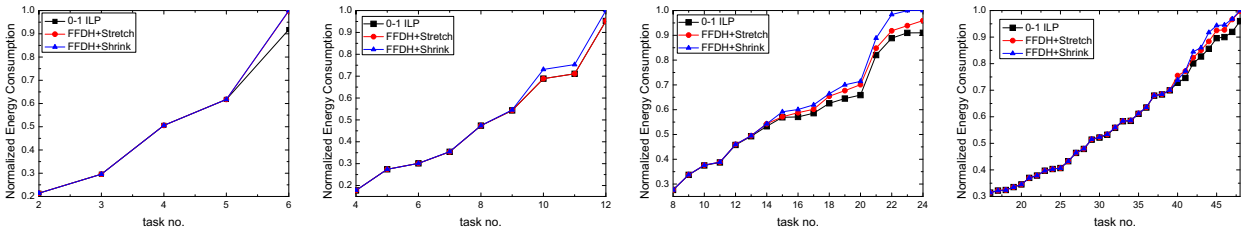
In this experiment, we evaluate the energy efficiency of different approaches for moldable real-time tasks. As shown in Figure 3, the two heuristics perform almost the same as the 0-1 ILP solution[3], which represents that the allotments derived by Algorithm 3 always contains some appropriate ones.

From the results of both Figure 2 and Figure 3, we can see that the energy difference between the heuristics and 0-1 ILP for moldable task case, is smaller than that for the rigid task case. The reason is that the moldable task case has larger allotment search space and the overall algorithm returns the minimum energy value among all those polynomial allotments, while the parallel degree is fixed for a rigid task set. According to this, we can also conclude that the parallel degree allotment dominates other problem dimensions.

VI. Conclusion

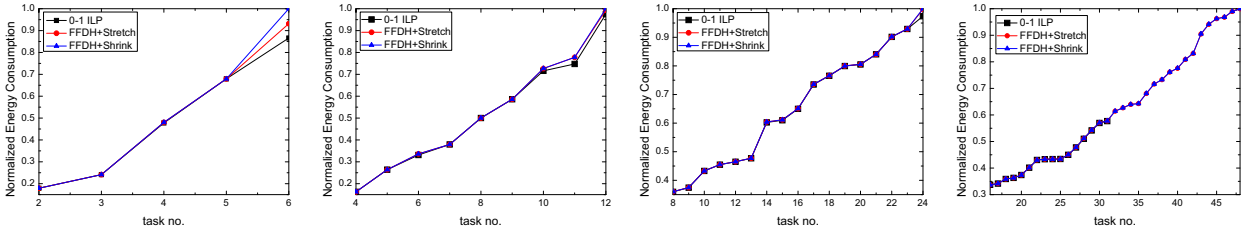
This is the first work addressing energy minimizing for parallel tasks by combining timing constraints and gang scheduling in multiprocessor/multicore systems with discrete

[3]Note that the 0-1 ILP curve is shorter than other two in Figure 3(b), since the CPLEX solver runs out of time when we use *node files* to store some parts of the branch & cut tree in disk due to the limited main memory.



(a) 4 cores (b) 8 cores (c) 16 cores (d) 32 cores

Fig. 2. Comparison of energy efficiency of 0-1 ILP and rigid tasks under multiprocessors with 4, 8, 16 and 32 cores using FFDH.



(a) 4 cores (b) 8 cores (c) 16 cores (d) 32 cores

Fig. 3. Comparison of energy efficiency 0-1 ILP and moldable tasks under multiprocessors with 4, 8, 16 and 32 cores using FFDH.

operation modes. Based one level-packing, for rigid tasks, we first presented 0-1 ILP to deal with the task scheduling and frequency assignment together, and then proposed an efficient two-step algorithm to solve them respectively. For moldable tasks, a 0-1 ILP was also provided, and then combining the new allotment heuristic, a three-step algorithm with polynomial-time complexity was proposed. Simulation experiments reported excellent results of the proposed heuristics. For the future work, we will explore energy saving for other general packing schemes with timing guarantees and compare to the approaches proposed in this paper.

Acknowledgement: We would like to thank the anonymous reviewers for their constructive comments. This work is partially supported by the NSF of China under Grant No. 60973017 and the Fundamental Research Funds for the Central Universities under Grant No. N100604010 and N110804003.

REFERENCES

[1] J. Chen, H. Hsu, K. Chuang, C. Yang, A. Pang, and T. Kuo, "Multiprocessor energy-efficient scheduling with task migration considerations," in *ECRTS'04*.

[2] J. Chen and T. Kuo, "Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics," in *ICPP'05*.

[3] C. Yang, J. Chen, and T. Kuo, "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor," in *DATE'05*.

[4] T. AlEnawy and H. Aydin, "Energy-aware task allocation for rate monotonic scheduling," in *RTAS'05*.

[5] J. Chen, T. Kuo, C. Yang, and K. King, "Energy-efficient real-time task scheduling with task rejection," in *DATE'07*.

[6] J. Chen, H. Hsu, and T. Kuo, "Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems," in *RTAS'06*.

[7] D. Zhu, R. Melhem, and B. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems," in *RTSS'01*.

[8] C. Xian, Y. Lu, and Z. Li, "Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time," in *DAC'07*.

[9] J. Chen, C. Yang, H. Lu, and T. Kuo, "Approximation Algorithms for Multiprocessor Energy-Efficient Scheduling of Periodic Real-Time Tasks with Uncertain Task Execution Time," in *RTAS'08*.

[10] S. Cho and R. Melhem, "Corollaries to Amdahls law for energy," *CAL*, 2008.

[11] V. Korthikanti and G. Agha, "Analysis of Parallel Algorithms for Energy Conservation in Scalable Multicore Architectures," in *ICPP'09*.

[12] A. Y. Zomaya, "Parallel processing for real-time simulation: A case study," *PDT*, 1996.

[13] S. Kwon, Y. Kim, W. Jeun, S. Ha, and Y. Paek, "A retargetable parallel-programming framework for MPSoC," *TODAES*, 2008.

[14] G. Manimaran, C. Murthy, and K. Ramamritham, "A new approach for scheduling of parallelizable tasks in real-time multiprocessor systems," *RTS*, 1998.

[15] G. Manimaran and C. Murthy, "An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems," *TPDS*, 1998.

[16] O. Kwon and K. Chwa, "Scheduling parallel tasks with individual deadlines," *TCS*, 1999.

[17] S. Kato and Y. Ishikawa, "Gang EDF Scheduling of Parallel Task Systems," in *RTSS'09*.

[18] I. Lupu and J. Goossens, "Scheduling of hard real-time multi-thread periodic tasks," *Arxiv preprint arXiv:1105.5080*, 2011.

[19] X. Lin, Y. Lu, J. Deogun, and S. Goddard, "Real-time divisible load scheduling for cluster computing," in *RTAS'07*.

[20] ——, "Real-time divisible load scheduling with different processor available times," in *ICPP'07*.

[21] F. Kong, N. Guan, Q. Deng, and W. Yi, "Energy-efficient scheduling for parallel real-time tasks based on level-packing," in *SAC'11*.

[22] E. Coffman Jr, M. Garey, D. Johnson, and R. Tarjan, "Performance bounds for level-oriented two-dimensional packing algorithms," *SICOMP*, 1980.

[23] Q. Wang and K. Cheng, "A heuristic of scheduling parallel tasks and its analysis," *SICOMP*, 1992.

[24] M. Drozdowski, "Real-time scheduling of linear speedup parallel tasks," *IPL*, 1996.

[25] J. Turek, J. Wolf, K. Pattipati, and P. Yu, "Scheduling parallelizable tasks: putting it all on the shelf," *SIGMETRICS*, 1992.

[26] G. Mounie and D. Trystram, "Efficient approximation algorithms for scheduling malleable tasks," in *SPAA'92*.

[27] C. Han and K. Lin, "Scheduling parallelizable jobs on multiprocessors," in *RTSS'89*.

[28] J. Blazewicz, M. Drabowski, J. Weglarz, I. Automatyki, and P. Poznańska, "Scheduling multiprocessor tasks to minimize schedule length," *TC*, 1986.

[29] O. Kwon, J. Kim, S. Hong, and S. Lee, "Real-time job scheduling in hypercube systems," in *ICPP'97*.

[30] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," *IPL*, 2008.

[31] R. Xu, D. Zhu, C. Rusu, R. Melhem, and D. Mossé, "Energy-efficient policies for embedded clusters," in *LCTES'05*.