Supplement of "Reducing Peak Power Consumption in Multi-Core Systems Without Violating Real-Time Constraints"

Jinkyu Lee, Buyoung Yun and Kang G. Shin

EXAMPLES IN SECTIONS 3, 4 AND 5

Example 1: [in Section 3] Consider the following multi-core chip and task set:

- A chip with 4 cores $\{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4\}$, and
- A set of 6 tasks to be executed on their designated cores $\Phi_1 = \{\tau_1, \tau_2\}$, $\Phi_2 = \{\tau_3\}$, $\Phi_3 = \{\tau_4, \tau_5\}$ and $\Phi_4 = \{\tau_6\}$, with their respective peak power consumption $B_1 = 20$, $B_2 = 15$, $B_3 = 9$, $B_4 = 17$, $B_5 = 8$, and $B_6 = 12$ (W).

Then, the first column of Table 1 shows Θ^* for this setting, where $\Theta^*(y)$ is the y-th element of Θ^*.

Example 2: [in Section 4.1] Consider a two-core chip $\{\mathcal{S}_1, \mathcal{S}_2\}$, and a set of five tasks $\Phi_1 = \{\tau_1, \tau_2, \tau_3\}$ and $\Phi_2 = \{\tau_4, \tau_5\}$. Suppose that $\Theta \triangleq \{(\tau_1, \tau_4), (\tau_2, \tau_4)\}$ and all tasks are active at t, i.e., $Q(t) = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$ at Step 1 in Algorithm 1. A global fixed priority is given to each task such that the smaller task index, the higher priority, i.e., $\tau_1 > \tau_2 > \tau_3 > \tau_4 > \tau_5$.

At t, τ_1 is chosen by Step 3 in Algorithm 1, and then τ_2 and τ_3, and τ_4 are removed from $Q(t)$ by Steps 6(a) and 6(b), respectively. In the next iteration, τ_5 is chosen by Step 3. Then, while the traditional partitioned FP scheduling algorithm chooses $\{\tau_1, \tau_4\}$ at t, FP_Θ chooses $E(t) = \{\tau_1, \tau_5\}$ at Step 8, due to the pair (τ_1, τ_4) in Θ.

Example 3: [in Section 4.2] We consider the same task set and two-core chip as Example 2. Each task's specification is given in Table 5. Suppose $\Theta \triangleq \{(\tau_1, \tau_4), (\tau_2, \tau_4)\}$. Then, $\Gamma_4 = \mathcal{H}_4 \cup \mathcal{F}_4 = \emptyset \cup \{\tau_1, \tau_2\} = \{\tau_1, \tau_2\}$ and $\Gamma_5 = \mathcal{H}_5 \cup \mathcal{F}_5 = \{\tau_4\} \cup \emptyset = \{\tau_4\}$. Therefore, $\Gamma_4 \subset \Gamma_5$ does not hold in spite of $\tau_4 \in \Gamma_5$, meaning that P1 does not hold.

If P2 holds, the maximum blocking time of τ_5 by tasks in $\Gamma_5 = \{\tau_4\}$ in an interval of length 2 is to be upper-bounded by $\left\lceil \frac{2}{5} \right\rceil \cdot 1 = 1$. This occurs when τ_4 and τ_5 are released synchronously.

However, if the release patterns of τ_4 and τ_5 (as well as τ_1 and τ_2) are given as in Fig. 2, the maximum blocking time of τ_5 by tasks in $\Gamma_5 = \{\tau_4\}$ in an interval of length 2 can be 2, meaning that P2 does not hold. This is because jobs of τ_1 and τ_2 do not directly block the job of τ_5, but they delay the execution of τ_4, increasing the blocking time of τ_5 by τ_4, as shown in Fig. 2.

Task	Designated core	T_i	C_i	D_i	P_i	B_i
τ_1	$\mathcal{S}_1$	5	2	5	1	20
τ_2	$\mathcal{S}_1$	3	1	3	2	15
τ_3	$\mathcal{S}_1$	6	1	6	3	9
τ_4	$\mathcal{S}_2$	5	1	5	4	17
τ_5	$\mathcal{S}_2$	4	1	4	5	10

TABLE 5
Task specification

Example 4: [in Section 4.2] We consider the same task set and two-core chip as Example 3 (see Table 5). Suppose that $\Theta \triangleq \{(\tau_1, \tau_4), (\tau_2, \tau_4)\}$.

We calculate $\{R_i\}_{i=1}^5$ from R_1 to R_5 according to their task priority as follows:

- $\Gamma_1 = \emptyset$; $R_1 = C_1 = 2$.
- $\Gamma_2 = \{\tau_1\}$ and $\delta_{(2,1)} = 0$; $R_2 = 3$.
- $\Gamma_3 = \{\tau_1, \tau_2\}$ and $\delta_{(3,1)} = \delta_{(3,2)} = 0$; $R_3 = 5$.
- $\Gamma_4 = \{\tau_1, \tau_2\}$ and $\delta_{(4,1)} = \delta_{(4,2)} = 0$; $R_4 = 5$.
- $\Gamma_5 = \{\tau_4\}$ and $\delta_{(5,2)} = 5 - 1 = 4$; $R_5 = 3$.

Then $R_i \leq D_i, \forall \tau_i$ holds. Therefore, the task set is schedulable by FP_Θ with $\Theta = \{(\tau_1, \tau_4), (\tau_2, \tau_4)\}$.

Example 5: [in Section 5] We consider the same task set and two-core chip as Example 3 (see Table 5).

In Step 1 in Algorithm 2, we construct Θ^* as shown in Table 6. In Step 2, we remove $\Theta^*(6)$ because $\Theta^*(6) = 19 < B_{max} = B_1 = 20$. Then, we check to find that this setting does not satisfy both Steps 3(a) and 3(b). By Steps 4–6, we set succ, fail and curr to 0, 6 and 3, respectively.

In the first iteration of Steps 7–10, Theorem 1 deems the task set unschedulable by FP_Θ with $\Theta = [\Theta^*(1), \Theta^*(2), \Theta^*(3)]$ because $\Gamma_5 = \{\tau_1, \tau_4\}$, so $R_5 > 4$. Then, fail and curr are set to 3 and 1. In the second iteration, the theorem deems the task set schedulable by FP_Θ with $\Theta = [\Theta^*(1)]$, and then succ and curr are set to 1 and 2. In the third iteration, the theorem also deems the task set schedulable by FP_Θ with $\Theta = [\Theta^*(1), \Theta^*(2)]$, and then succ and curr are both set to 2. Now, fail $>$ succ $+ 1$ does not hold, so the iteration halts. Finally, the algorithm return FEASIBLE with FP_Θ with $\Theta = [\Theta^*(1) = \{\tau_1, \tau_4\}, \Theta^*(2) = \{\tau_2, \tau_4\}]$, and in this case $\mathcal{B}$ (the design-time chip-level peak power consumption) is set to $B_1 + B_5 = 30$.

SOME DETAILS OF SECTION 6.1

We generated task sets based on the technique in [17], which has been widely used to generate real-time task sets. For each task, T_i is uniformly distributed in $[1, T_{max} = 1000]$; C_i is chosen according to the exponential distribution of C_i/T_i, whose probability density function is $\lambda \cdot \exp(\lambda \cdot x)$ with $1/\lambda = 0.3$; and D_i is set to T_i for implicit deadline task sets and is uniformly distributed within $[C_i, T_i]$ for constrained deadline task sets.

$\Theta^*(i) =$ (task on $\mathcal{S}_1$, task on $\mathcal{S}_2$)	an upper bound on the chip-level power consumption (W)
$\Theta^*(1) = (\tau_1, \tau_4)$	$20 + 17 = 37$
$\Theta^*(2) = (\tau_2, \tau_4)$	$15 + 17 = 32$
$\Theta^*(3) = (\tau_1, \tau_5)$	$20 + 10 = 30$
$\Theta^*(4) = (\tau_3, \tau_4)$	$9 + 17 = 26$
$\Theta^*(5) = (\tau_2, \tau_5)$	$15 + 10 = 25$
$\Theta^*(6) = (\tau_3, \tau_5)$	$9 + 10 = 19$

TABLE 6
An upper-bound of the chip-level power consumption when two tasks on different cores are executed at the same time

For each type of task sets (implicit or constrained), we repeat the following procedure and generate 20,000 per-core task sets (i.e., Φ_i).

1. Initially, generate a set of 2 tasks.
2. Check whether the generated task set can pass the exact uniprocessor schedulability test of FP [11].
3. If it fails to pass the test, discard the generated task set and return to Step 1. Otherwise, include this set for evaluation. Then, this set serves as a basis for the generation of a new set; create a new set by adding a new task into an already created and tested set, and return to Step 2.

Additional Related Work

There has been a line of work on how to achieve power efficiency of general-purpose systems when they run linear algebra applications that are known to yield high peak power [22], [25], [26]. However, since their approaches utilize task parallelism (sub-tasks can be executed concurrently on multiple cores), they cannot be applicable to our sequential task model in Section 2.

[25] H. Ltaief, P. Luszczek, and J. Dongarra, "Profiling high performance dense linear algebra algorithms on multicore architectures for power and energy efficiency," Computer Science - Research and Development, vol. 27, no. 4, pp. 277–287, 2012.

[26] G. Bosilca, H. Ltaief, and J. Dongarra, "Power profiling of cholesky and QR factorizations on distributed memory systems," To appear in Computer Science - Research and Development.