LETTER

Processor-Minimum Scheduling of Real-Time Parallel Tasks*

Wan Yeon LEE†a), Kyungwoo LEE††, *Nonmembers*, Kyong Hoon KIM†††, *and* Young Woong KO†, *Members*

SUMMARY We propose a polynomial-time algorithm for the scheduling of real-time parallel tasks on multicore processors. The proposed algorithm always finds a feasible schedule using the minimum number of processing cores, where tasks have properties of linear speedup, flexible preemption, arbitrary deadlines and arrivals, and parallelism bound. The time complexity of the proposed algorithm is $O(M^3 \cdot \log N)$ for M tasks and N processors in the worst case.

key words: *scheduling algorithm, real-time task, parallel task, multicore*

1. Introduction

Multicore processors have become increasingly popular in embedded systems as well as server systems. In the multicore platform, processing cores dominate the entire energy consumption and thus implementing effective energy saving technologies for the processing cores has become a critical goal. In fact, many real-time tasks do not need to be run on all available processing cores to meet their deadlines. If we know the minimum number of processing cores required for the completion of the given real-time tasks, we may achieve potential energy-saving by deactivating the unneeded cores [1], [2]. To do so, we propose a polynomial-time algorithm which schedules all real-time tasks so as to complete their execution before their respective deadlines using the minimum number of cores (or processors). The time complexity of the proposed algorithm is $O(M^3 \cdot \log N)$ for M tasks and N processors in the worst case.

The task model considered in this paper has the properties of *flexible preemption*, *linear speedup*, *parallelism bound*, arbitrary deadlines and arbitrary arrivals. In flexible preemption, it is assumed to suspend and restart tasks among processors without incurring any additional costs [3], [4]. In linear speedup, the speedup is linearly proportional to the number of allocated processors [5]. In parallelism bound, the speedup of parallel tasks can be maintained only up to some bounded number of processors [6].

There have been many researches for the scheduling of real-time parallel tasks on multiprocessors. Caramia and Drozdowski [3] studied an optimal scheduling problem of real-time parallel tasks having properties of flexible preemption, linear speedup and parallelism bound. They studied the problem of minimizing the mean completion time of all tasks, but did not consider the problem of minimizing the number of processors. Burchard *et al.* [7] studied to find a feasible schedule on the minimum number of processors. Burchard's algorithm works only for non-parallel tasks but not for parallel tasks. Some previous studies [4], [8], [9] with optimal criteria are similar to our algorithm. Even though these studies can find a feasible schedule for real-time tasks having properties of linear speedup, flexible preemption and parallelism bound, they enforced some specific constraints on tasks' deadlines or arrivals, such as the same deadline [4], the same arrival time [8], or a particular order of Last Come First Served (LCFS) between arrival times and deadlines [9]. Contrarily, our algorithm allows tasks to have *arbitrary arrivals* and *arbitrary deadlines*. Hence, our algorithm is more practical than the previous theoretical studies.

Although this paper focuses on the processor-minimum scheduling of real-time tasks, the proposed algorithm is also applicable to any minimization problem of resources with like model of multiple requests and their parallel consumption, *e.g.*, allocating the bandwidth (computation) of simultaneous channels (real-time tasks) using the minimum number of communication links (processors) of routers [3].

2. Proposed Algorithm

This paper deals with the problem of scheduling a set of M tasks on N identical processors. To formulate the problem, processor n is denoted as P_n and task m is denoted as T_m with a quadruplet (a_m, d_m, c_m, b_m). a_m, d_m, c_m and b_m denote the arrival time, the deadline, the computation amount and the parallelism bound of T_m, respectively. The case of $c_m > b_m \cdot (d_m - a_m)$ is excluded because it is impossible to execute more than $b_m \cdot (d_m - a_m)$ for the time $(d_m - a_m)$. M tasks are given as a set $TS = \{T_1, \cdots, T_M\}$. We assume that a_m, d_m, c_m and b_m of each task T_m are known to the scheduler in advance.

Now we describe how the proposed algorithm schedules M tasks on N processors. We define at most $(2M - 1)$ intervals using each a_m and d_m of M tasks. All a_m and d_m of M tasks are sorted in the increasing order and labeled with another index e^x when their corresponding ranking is the x-th place. An interval I^x is then defined as $[e^x, e^{x+1})$. The number of interval I^xs is at most $(2M-1)$, because the number is $(2M - 1)$ when all a_m and d_m are distinct. We also

Manuscript received December 16, 2008.

†The authors are with Hallym Univ., South Korea.
††The author is with Intel Corporation, U.S.A.
†††The author is with Gyeongsang National Univ., South Korea.

*This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD, Basic Research Promotion Fund) (KRF-2007-313-D00634).

a) E-mail: wanlee@hallym.ac.kr

DOI: 10.1587/transinf.E92.D.723

define the minimal computation amount that must be allocated at I^x as *mandatary computation* of the task. In other words, the mandatary computation is the partial computation amount that remains when T_m is assumed to be executed with its maximum parallelism b_m in the next intervals, i.e., $c_m - b_m \cdot (d_m - e^{x+1})$. An interval is called *overloaded* if its computation capacity is less than the total mandatory computation assigned to this interval. The excess of the total mandatory computation over the available computation capacity is called *surplus workload* of the overloaded interval.

We schedule tasks within each interval from I^1 to I^{2M-1} sequentially. For each I^x, our algorithm first selects all tasks T_m which are executable at I^x (i.e., T_m satisfying $a_m \le e^x < e^{x+1} \le d_m$). Next, it allocates the mandatory computations of the tasks executable at I^x. When all mandatory computations can not be completely allocated due to lack of available computation capacity, the algorithm simply fails to schedule the task. Finally, if all mandatory computations at I^x are completely allocated and there still remains available computation capacity at I^x, the algorithm utilizes this computation capacity to execute the mandatory computations at the next intervals. It preferentially executes the surplus workload of the nearest overloaded interval. After executing all surplus workloads of overloaded intervals, it executes the mandatory computations at the next intervals from I^{x+1} to I^{2M-1}.

Figure 1 shows the Feasible-Scheduling algorithm which finds a feasible schedule of M tasks in TS using given N' processors, where the following notations are used:

SL^x: a set of all tasks executable at I^x
δ_m^x: the mandatory computation of T_m in SL^x at I^x
Φ^x: the available computation capacity at I^x
λ_m^x: the computation amount allocated to T_m at I^x

Feasible-Scheduling(TS, N')
1 $c'_m \leftarrow c_m$ **for each** T_m in TS;
2 **for each** x from 1 to $2M-1$
3 $\Phi^x \leftarrow N' \cdot (e^{x+1} - e^x)$; make a subset SL^x of TS;
4 **for each** T_m in SL^x
5 $\delta_m^x \leftarrow max(c'_m - b_m \cdot (d_m - e^{x+1}),\ 0)$; $c'_m \leftarrow c'_m - \delta_m^x$;
6 **endfor**
7 **endfor**
8 **for each** x from 1 to $2M-1$
9 **if**($\Phi^x < \sum \delta^x$), **return** *FAIL*;
10 $\Phi^x \leftarrow \Phi^x - \sum \delta^x$; $\lambda_m^x \leftarrow \delta_m^x$ and $\delta_m^x \leftarrow 0$ **for each** $\delta_m^x > 0$;
11 **while**($\Phi^x > 0$ and $SL^x \ne \{\ \}$)
12 select the nearest I^y such that $\Phi^y < \sum \delta^y$ and $\delta_m^y > 0$ for some $T_m \in SL^x$;
13 **if**($\Phi^y \ge \sum \delta^y$ or $\delta_m^y = 0$ for any y and any $T_m \in SL^x$), select the nearest I^y such that $\delta_m^y > 0$ for some $T_m \in SL^x$;
14 **for each** positive δ_m^y such that $T_m \in SL^x$
15 **if**($\Phi^y \ge \sum \delta^y$), $\lambda' \leftarrow min(\delta_m^y, \Phi^x, b_m \cdot (e^{x+1} - e^x) - \lambda_m^x)$;
16 **else**, $\lambda' \leftarrow min(\sum \delta^y - \Phi^y, \delta_m^y, \Phi^x, b_m \cdot (e^{x+1} - e^x) - \lambda_m^x)$;
17 increase λ_m^x by λ'; decrease Φ^x, δ_m^y and c_m by λ';
18 **if**($\lambda_m^x = b_m \cdot (e^{x+1} - e^x)$ or $c_m = 0$), remove T_m from SL^x;
19 **if**($\Phi^x = 0$ or $SL^x = \{\ \}$), **go to** line 22;
20 **endfor**
21 **endwhile**
22 $n \leftarrow 1$;
23 **Task-Allocation**(λ_m^x, I^x) **for each** $\lambda_m^x > 0$;
24 **endfor**
25 **return** *TRUE*;

Procedure **Task-Allocation**(λ_m^x, I^x)
26 **while**($\lambda_m^x > 0$)
27 $\tau_s \leftarrow$ the earliest available time of P_n within $[e^x,\ e^{x+1})$;
28 $\tau_e \leftarrow min((\tau_s + \lambda_m^x),\ e^{x+1})$; $\lambda_m^x \leftarrow \lambda_m^x - (\tau_e - \tau_s)$;
29 reserve P_n for the execution of T_m from τ_s to τ_e;
30 **if**($\tau_e = e^{x+1}$), $n \leftarrow n + 1$;
31 **endwhile**

Fig. 1 Description of Feasible-Scheduling.

The algorithm initializes Φ^x, SL^x and δ_m^x for each I^x at lines 1–7, and schedules tasks from I^1 to I^{2M-1} sequentially at lines 8–21. When scheduling SL^x within each I^x, it allocates the mandatory computation of all tasks T_m in SL^x at line 10. If the available computation capacity is less than the total mandatory computation (i.e., $\Phi^x < \sum \delta^x$), it immediately fails at line 9. If there remains available computation capacity (i.e., $\Phi^x > 0$) after completely allocating all mandatory computations of I^x, the mandatory computations δ_m^y at some behind interval I^y for T_m executable at both I^x and I^y (i.e., $T_m \in SL^x$ and $T_m \in SL^y$) is allocated to the remaining computation capacity of I^x at lines 11–21. Preferentially, the surplus workload of the nearest overloaded interval I^y is allocated at lines 12 and 16. If there is no overloaded interval including any positive δ_m^y allocatable at I^x, the mandatory computation of the next interval from I^{x+1} to I^{2M-1} is allocated at lines 13 and 15. The additional allocation amount λ' associated with δ_m^y is limited by the remaining computation capacity of I^x (i.e., Φ^x), the available execution capacity of T_m at I^x (i.e., $b_m \cdot (e^{x+1} - e^x) - \lambda_m^x$), and the surplus workload of the selected overloaded interval (i.e., $\sum \delta^y - \Phi^y$). Their minimum value determines the additional allocation amount λ' at line 15 or 16. This allocation procedure is repeated until the available computation capacity is exhausted (i.e., $\Phi^x = 0$) or each T_m in SL^x is allocated up to its parallelism bound or completely allocated (i.e., $\lambda_m^x = b_m \cdot (e^{x+1} - e^x)$ or $c_m = 0$).

Assigning each λ_m^x to available processors within I^x is performed in the Task-Allocation() procedure at line 23. The Task-Allocation() procedure searches the starting time τ_s and the ending time τ_e for P_n to execute T_m at lines 27–28. P_n is reserved for the execution of T_m from τ_s to τ_e at line 29. Until λ_m^x is completely assigned (i.e., $\lambda^x = 0$), the processor number n is increased by one and the next P_n is reserved for the execution of T_m at I^x.

A schedule completing all tasks before their respective deadlines is called *feasible*. Then Feasible-Scheduling always finds a feasible schedule. If Feasible-Scheduling finishes at line 25, the total computation of each task is completely allocated and the execution time of each task does not exceed its deadline. The number of processors allocated to tasks is not larger than N' at any time. The number of processors executing each task does not exceed its parallelism bound at any time.

Figure 2 shows the MinProc-Scheduling algorithm, which finds a feasible schedule using the minimum number of processors. MinProc-Scheduling utilizes the binary search operation. This binary search operation finds the

MinProc-Scheduling(TS, N **)**
1 $L \leftarrow 1$; $H \leftarrow min(N, \sum_{m=1}^{M} b_m)$; $N' \leftarrow \lfloor (L+H)/2 \rfloor$;
2 **while(** $N' < H$ **)**
3 **if(** **Feasible-Scheduling(** TS, N' **)** **)**, $H \leftarrow N'$;
4 **else**, $L \leftarrow (N'+1)$;
5 $N' \leftarrow \lfloor (L+H)/2 \rfloor$;
6 **endwhile**

Fig. 2 Description of MinProc-Scheduling.

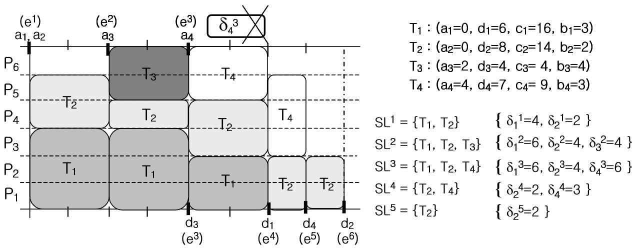

Fig. 3 A scheduling example of the Feasible-Scheduling algorithm.

median number of processors and performs the Feasible-Scheduling algorithm with the median number of processors. Depending on whether the Feasible-Scheduling algorithm succeeds, the algorithm searches either the lower or the upper half of processor numbers. This procedure is repeated until $N' \geq H$. When the algorithm completes the while loop at line 6, N' is the minimum number of processors to satisfy the deadlines of all tasks in TS.

The total time complexity of MinProc-Scheduling is $O(M^3 \cdot \log N)$ in the worst case. The innermost loop body at the lines 14–20 of Fig. 1 determines the time complexity of Feasible-Scheduling. This loop body performs $O(M^3)$ iterations because the loops at lines 8, 11, and 14 iterate at most $(2M-1)$, $2 \cdot (2M-2)$ and M, respectively. The *while* loop at line 26 performs at most N iterations for all λ_m^x in I^x. Usually $M^2 > N$ and thus the total time complexity of Feasible-Scheduling is $O(M^3)$. The binary search at lines 2–6 of Fig. 2 is a logarithmic algorithm and always performs $\lceil \log min(N, \sum_{m=1}^{M} b_m) \rceil$ iterations.

Figure 3 shows a scheduling example of Feasible-Scheduling, where four tasks are scheduled on six processors. By ordering all a_m and d_m of tasks T_m, each interval $I^x = [e^x, e^{x+1})$ is assigned. Set of executable tasks SL^x and all mandatory computations δ_m^x are initialized. They are shown in the right side of Fig. 3. When scheduling tasks T_1 and T_2 in SL^1, the mandatory computations δ_1^1 and δ_2^1 are allocated to λ_1^1 and λ_2^1 respectively (line 10 in Fig. 1). Because there still remains available computation capacity (i.e., $\Phi^1 = (e^2 - e^1) \cdot N' - (\lambda_1^1 + \lambda_2^1) = 6 > 0$), the algorithm utilizes the remaining computation capacity to allocate the surplus workload of the nearest overloaded interval I^2. It selects δ_2^2 arbitrarily among δ_1^2 and δ_2^2, and allocates $min(\ \sum \delta^2 - \Phi^2,\ \delta_2^2,\ \Phi^1,\ b_2 \cdot (e^2 - e^1) - \lambda_2^1\ ) = 2$ to λ_2^1 (line 16 in Fig. 1). Then the interval I^2 is not overloaded any more and the interval I^3 becomes the nearest overloaded interval. Next it allocates $min(\ \delta_1^3,\ \Phi^1,\ b_1 \cdot (e^2 - e^1) - \lambda_1^1\ ) = 2$ to λ_1^1. Then the assigned values at I^1 are $\lambda_1^1 = 6$ and $\lambda_2^1 = 4$. The mandatory computations $\delta_2^2 = 4$ and $\delta_1^3 = 6$ are updated with $\delta_2^2 = 2$ and $\delta_1^3 = 4$. Because $\Phi^2 = \sum \delta^2$ when scheduling SL^2, λ_m^2 becomes equal to δ_m^2 ($\lambda_1^2 = \delta_1^2 = 6$, $\lambda_2^2 = \delta_2^2 = 2$ and $\lambda_3^2 = \delta_3^2 = 4$). Because $\Phi^3 < \sum \delta^3$ when scheduling SL^3, the algorithm fails to schedule T_1, T_2, and T_4 within I^3.

3. Properties of the Proposed Algorithm

For clarity, we use the following notation:

- η_m^x: the smallest number of processors allocated to T_m within I^x.
- η^x: the smallest number of processors allocated to $T_1, T_2, \cdots, T_M$ within I^x.

Lemma 1: For any $T_m \in SL^y$ and I^x such that $a_m \leq e^x < e^y$, if $\eta^x < N'$ and $\eta_m^y > 0$ after scheduling SL^x at I^x, then $\eta_m^x = b_m$.

proof: Whenever available processors remain within I^x (i.e., $\Phi^x > 0$), Feasible-Scheduling uses these processors to execute δ_m^y for any T_m in SL^x such that $a_m \leq e^x < e^y$ until $\eta_m^x = b_m$ or $c_m = 0$. If $c_m = 0$ after scheduling SL^x at I^x, then $\eta_m^y = 0$ for any I^y such that $e^x < e^y$. Hence, if $\eta_m^y > 0$ after scheduling I^y, then $\eta_m^x = b_m$ for $\eta^x < N'$ and $T_m \in SL^y$. □

Lemma 2: If $\Phi^y < \sum \delta^y$ when scheduling SL^y at I^y, then the remaining computation of any $T_m \in SL^y$ cannot be allocated to the available processors in any I^x such that $a_m \leq e^x < e^y$ or in any I^z such that $e^y < e^z < d_m$.

proof: It is clear that a positive δ_m^y cannot be allocated when there is no available processors in I^x (i.e., $\eta^x = N'$). When there are available processors in any I^x such that $a_m \leq e^x < e^y$ (i.e., $\eta^x < N'$), the remaining computation of any T_m cannot be allocated to these processors because $\eta_m^x = b_m$ by Lemma 1. Also, $c_m \geq b_m \cdot (d_m - e^y)$ for any $T_m \in SL^y$ because Feasible-Scheduling does not decrease any δ^z in I^z such that $e^y < e^z$ until $\Phi^y = \sum \delta^y$. Hence, the remaining computation c_m of T_m cannot be allocated completely to the processors in any I^z such that $e^y < e^z < d_m$, when $\Phi^y < \sum \delta^y$. □

Lemma 3: If $\eta_k^x > 0$ and $\Phi^y < \sum \delta^y$ for any $T_k \in SL^x$ such that $a_k \leq e^x < \min(e^y, d_k)$ when scheduling SL^y at I^y, then $\eta_k^w = b_k$ for any I^w such that $e^x < e^w < \min(e^y, d_k)$ and $\eta^w < N'$, and $c_k = b_k \cdot (d_k - e^y)$ when $e^y < d_k$.

proof: If $\Phi^y < \sum \delta^y$ and $e^y < d_k$ for any $T_k \in SL^y$ when scheduling SL^y at I^y, then $c_k = b_k \cdot (d_k - e^y)$ because Feasible-Scheduling does not decrease δ_k^z in any I^z such that $e^y < e^z$ until $\Phi^y = \sum \delta^y$. Also, if $\Phi^y < \sum \delta^y$ when scheduling SL^u at I^u such that $e^x \leq e^u < e^y$, Feasible-Scheduling does not decrease any δ^w in any I^w such that $e^u < e^w < e^y$ and $\Phi^w > \sum \delta^w$ until $\Phi^y = \sum \delta^y$. If $\Phi^w < \sum \delta^w$ when scheduling SL^u, then $\Phi^w = \sum \delta^w$ when scheduling SL^w. Hence, $\eta_k^w = b_k$ or $\eta^w = N'$ after scheduling SL^w at I^w such that $e^x \leq e^w < \min(e^y, d_k)$. □

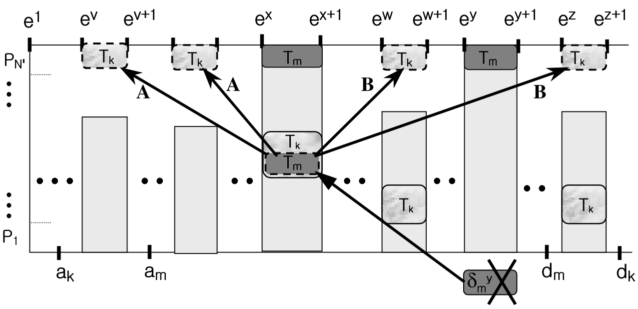

Fig. 4 The case of failing to schedule T_m within I^y.

Theorem 1: Feasible-Scheduling always finds a feasible schedule if there are feasible schedules on given N' processors.

proof: If $\Phi^y < \sum \delta^y$ when scheduling SL^y at I^y, then Feasible-Scheduling fails to schedule the tasks $T_m \in SL^y$ because the remaining computation of any T_m cannot be allocated to available processors in any interval by Lemma 2. We assume that there is a feasible schedule which satisfies the deadlines of all tasks simultaneously. This feasible schedule is referred to as *New Schedule* and the failed schedule of Feasible-Scheduling is referred to as *Original Schedule*. Compared with the Original Schedule, the New Schedule must additionally use some processors reserved for the execution of a previously scheduled task T_k in I^x such that $a_k, a_m \le e^x \le e^y$, in order to schedule T_m successfully. Only when $\eta_k^x > 0$, $\eta_m^x < b_m$ and $\eta^x = N'$ such that $a_k, a_m \le e^x \le e^y$, T_m is allowed to use some processors reserved for T_k within I^x. If $\eta^x < N'$, then $\eta_m^x = b_m$ by Lemma 1. When $\eta_m^x = b_m$ or $\eta_k^x = 0$, b_m processors are already allocated to T_m within I^x or there is no processor reserved for T_k within I^x, respectively.

If T_m uses some processors reserved for T_k in I^x, then T_k must use some processors available between a_k and d_k, in order to compensate for the additional processors required for T_m. Then let us check whether both T_k and T_m can be scheduled in the New Schedule. We first check the case that T_k uses available processors in any I^v such that $a_k \le e^v < e^x$ in order to compensate for its loss, which is described as arrow A in Fig. 4. Because $\eta_k^v = b_k$ by Lemma 1, the lost computation amount of T_k in I^x cannot be allocated to available processors in I^v. Next, we check the case that T_k uses available processors in any I^w such that $e^x < e^w < d_k$ in order to compensate for its loss, which is described as arrow B in Fig. 4. If $\eta_k^x > 0$ and $\Phi^y < \sum \delta^y$ for any $T_k \in SL^x$ such that $a_k \le e^x \le \min(e^y, d_k)$ when scheduling SL^y at I^y, then $\eta_k^w = b_k$ for any I^w such that $e^x < e^w < \min(e^y, d_k)$ and $\eta^w < N'$, and $c_k = b_k \cdot (d_k - e^y)$ when $e^y < d_k$ by Lemma 3. Hence, the lost computation amount of T_k in I^x cannot be allocated to available processors in any I^w such that $e^x < e^w < d_k$.

From the above-mentioned facts, the New Schedule cannot satisfy the deadlines of both T_k and T_m. In order to schedule both T_k and T_m successfully, the New Schedule must use some additional processors reserved for the execution of another task T_j. However, all of T_j, T_k and T_m cannot be scheduled in the New Schedule by the same reason. Consequently, the assumption on feasibility of the New Schedule is a contradiction. This means that Feasible-Scheduling always finds a feasible schedule if there are feasible schedules on given N' processors. □

Theorem 2: MinProc-Scheduling always finds a feasible schedule using the minimum number of processors.

proof: When MinProc-Scheduling finds a feasible schedule using N' processors, let us assume that there is a feasible schedule using N'' processors such that $N'' < N'$. If Feasible-Scheduling finds a feasible schedule on N'' processors, then Feasible-Scheduling can find a feasible schedule on N^* processors for any $N^* \ge N''$. If MinProc-Scheduling selects a feasible schedule using N' processors, then this means that Feasible-Scheduling failed to find the feasible schedule on N^* processors such that $N' > N^* \ge N''$. Then it is a contradiction of Theorem 1. □

References

[1] M. Nikitovic and M. Brorsson, "An adaptive chip-multiprocessor architecture for future mobile terminals," Int'l Conf. Compilers, Architecture, and Synthesis for Embedded Systems, pp.43–49, 2002.

[2] L. Benini, A. Bogliolo, and G.D. Micheli, "A survey of design techniques for system-level dynamic power management," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.8, no.3, pp.299–316, 2000.

[3] M. Caramia and M. Drozdowski, "Scheduling malleable tasks for mean flow time criterion," Tech. Rep. RA-008/05, Poznam Univeriesty, 2005.

[4] M. Drozdowski, "Real-time scheduling of linear speedup parallel tasks," Inf. Process. Lett., vol.57, no.1, pp.35–40, 1996.

[5] J. Blazewicz, M.Y. Kovalyov, M. Machowiak, D. Trystram, and J. Weglarz, "Malleable tasks scheduling to minimize the makespan," Annals of Operations Research, vol.129, pp.65–80, 2004.

[6] K.C. Sevcik, "Application scheduling and processor allocation in multiprogrammed parallel processing systems," Perform. Eval., vol.19, no.2-3, pp.107–140, 1994.

[7] A. Burchard, J. Liebeherr, Y. Oh, and S.H. Son, "New strategies for assigning real-time tasks to multiporcessor systems," IEEE Trans. Comput., vol.44, no.12, pp.1429–1442, 1995.

[8] W.Y. Lee and H. Lee, "Optimal scheduling for real-time parallel tasks," IEICE Trans. Inf. & Syst., vol.E89-D, no.6, pp.1962–1966, June 2006.

[9] W.Y. Lee and Y.W. Ko, "Real-time scheduling of parallel tasks on fewest processors," IEEE Int'l Conf. Hybrid Information Technology, pp.562–567, 2006.