LETTER

Processor-Minimum Scheduling of Real-Time Parallel Tasks*

Wan Yeon LEE†, 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 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 $TS = \{T_1, \ldots, T_M\}$. 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 $d_m$ and $c_m$ 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 $I^x$ 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: wanllee@hallym.ac.kr
DOI: 10.1587/transinf.E92.D.723
define the minimal computation amount that must be allocated at \( I^i \) as mandatory computation of the task. In other words, the mandatory 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., \( c_m - b_m \cdot (d_m - e^{i+1}) \). 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 \( I^i \), our algorithm first selects all tasks \( T_m \) which are executable at \( I^i \) (i.e., \( T_m \) satisfying \( a_m \leq e^i < e^{i+1} \leq d_m \)). Next, it allocates the mandatory computations of the tasks executable at \( I^i \). 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 \( I^i \) are completely allocated and there still remains available computation capacity at \( I^i \), 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 \( I^{i+1} \) 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:

- \( S \): a set of all tasks executable at \( I^i \)
- \( \delta_m^i \): the mandatory computation of \( T_m \) in \( SL^i \) at \( I^i \)
- \( \Phi^i \): the available computation capacity at \( I^i \)
- \( \lambda_m^i \): the computation amount allocated to \( T_m \) at \( I^i \)

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

Assigning each \( \lambda_m^i \) to available processors within \( I^i \) is performed in the Task-Allocation() procedure at line 23. The Task-Allocation() procedure searches the starting time \( t_x \) and the ending time \( t_x \) for each \( T_m \) from \( t_x \) to \( t_x \) at line 29. Until \( t_x \) is completely assigned (i.e., \( t_x = t_x \)), \( n \) is increased by one and the next \( P_n \) is reserved for the execution of \( T_m \) at any time.

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

![Fig. 2](image)

Description of MinProc-Scheduling.

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 \(O(M^2 \cdot \log N)\) in the worst case. The innermost loop body operates \(O(M^3)\) 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 each \(\lambda_m\) in \(I^\delta\). 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 \(\lfloor \log \min(N, \sum_{m=1}^{N} b_m) \rfloor\) 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 \(I^s\) is assigned. Set of executable tasks \(SL^s\) and all mandatory computations \(\delta_m^s\) are initialized. They are initializes. They are initialized. They are

\[
\Phi = \left\{ a_m \mid b_m = 0, c_m < \min(e^i, d^i) \right\}
\]

\[
\Phi^2 = \left\{ a_m \mid b_m = 0, c_m < \min(e^i, d^i) \right\}
\]

\[
\Phi^3 = \left\{ a_m \mid b_m = 0, c_m < \min(e^i, d^i) \right\}
\]

The mandatory computations \(\delta_2^6 = 4\) and \(\delta_3^6 = 6\) are updated with \(\delta_2^6 = 2\) and \(\delta_3^6 = 4\). Because \(\Phi^2 = \sum \delta^2\) when scheduling \(SL^2\), \(\lambda_2^2\) becomes equal to \(\delta_2^6\) \((\lambda_1^1 = \delta_1^6 = 6, \lambda_2^2 = \delta_2^6 = 2\) and \(\lambda_3^6 = \delta_3^6 = 4\)). Because \(\Phi^3 < \sum \delta^3\) when scheduling \(SL^3\), the algorithm fails to schedule \(T_1, T_2,\) and \(T_4\) within \(I^7\).

3. Properties of the Proposed Algorithm

For clarity, we use the following notation:

- \(\eta_m^s\): the smallest number of processors allocated to \(T_m\) within \(I^s\).
- \(\eta^\Phi\): the smallest number of processors allocated to \(T_1, T_2, \ldots, T_M\) within \(I^\Phi\).

**Lemma 1:** For any \(T_m \in SL^{I^s}\) and \(I^s\) such that \(a_m \leq e^i < e^i\), if \(\eta_m^s \leq \eta^\Phi\) and \(\eta_m^s > 0\) after scheduling \(SL^s\) at \(I^s\), then \(\eta_m^s = b_m\).

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

\(\square\)

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

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

\(\square\)

**Lemma 3:** If \(\eta_m^s > 0\) and \(\Phi^\Phi < \sum \delta^s\) for any \(T_k \in SL^s\) such that \(d_k \leq e^i < \min(e^i, d_k)\) when scheduling \(SL^s\) at \(I^s\), then \(\eta_k^s = b_k\) for any \(I^s\) such that \(e^i < e^i < \min(e^i, d_k)\) and \(\eta^\Phi > \eta^s\) and \(c_k = b_k \cdot (d_k - e^i)\) when \(e^i < d_k\).

**proof:** If \(\Phi^\Phi < \sum \delta^s\) and \(e^i < d_k\) for any \(T_k \in SL^s\) when scheduling \(SL^s\) at \(I^s\), then \(c_k = b_k \cdot (d_k - e^i)\) because Feasible-Scheduling does not decrease \(\delta_m^s\) in any \(I^s\) such that \(e^i < e^i\) until \(\Phi^\Phi = \sum \delta^s\). Also, if \(\Phi^\Phi < \sum \delta^s\) when scheduling \(SL^s\) at \(I^s\) such that \(e^i < e^i < e^i\), Feasible-Scheduling does not decrease any \(\delta^s\) in any \(I^s\) such that \(e^i < e^i < e^i\) and \(\Phi^\Phi > \sum \delta^s\) until \(\Phi^\Phi = \sum \delta^s\). If \(\Phi^\Phi < \sum \delta^s\) when scheduling \(SL^s\), then \(\Phi^\Phi = \sum \delta^s\) when scheduling \(SL^s\). Hence, \(\eta_k^s = b_k\) or \(\eta^s = \eta^\Phi > \eta^\Phi> \eta^s\) after scheduling \(SL^s\) at \(I^s\) such that \(e^i < \min(e^i, d_k)\).

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

\[ \text{proof:} \quad \text{If } \Phi^w < \sum \delta^w \text{ when scheduling } SL^w \text{ at } P, \text{ then Feasible-Scheduling fails to schedule the tasks } T_{m} \in SL^w \text{ because the remaining computation of any } T_{m} \text{ cannot be allocated to available processors in any interval by Lemma 2. We assume that there is a feasible schedule which satisfies the deadlines of all tasks simultaneously. This feasible schedule is referred to as New Schedule. Compared with the Original Schedule, the New Schedule must additionally use some processors reserved for the execution of a previously scheduled task } T_{k} \text{ in } I^{i} \text{ such that } a_{k}, a_{m} \leq e^{i} \leq e^{w}, \text{ in order to schedule } T_{m} \text{ successfully. Only when } \eta_{k}^{w} > 0, \eta_{m}^{w} < b_{m} \text{ and } \eta^{w} = N^{w} \text{ such that } a_{k}, a_{m} \leq e^{i} \leq e^{w}, T_{m} \text{ is allowed to use some processors reserved } \text{for } T_{k} \text{ within } I^{i}. \text{If } \eta^{i} < N^{i}, \text{ then } \eta_{m}^{w} = b_{m} \text{ by Lemma 1. When } \eta_{m}^{w} = b_{m} \text{ or } \eta_{k}^{w} = 0, b_{m} \text{ processors are already allocated to } T_{m} \text{ within } I^{w} \text{ or there is no processor reserved for } T_{k} \text{ within } I^{i}, \text{ respectively.} \]

If \( T_{m} \) uses some processors reserved for \( T_{k} \) in \( I^{i} \), then \( T_{k} \) must use some processors available between \( a_{t} \) 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 \( I^{w} \) such that \( a_{k} \leq e^{w} < e^{i} \) in order to compensate for its loss, which is described as arrow \( A \) in Fig. 4. Because \( \eta_{k}^{w} = b_{k} \) by Lemma 1, the lost computation amount of \( T_{k} \) in \( I^{i} \) cannot be allocated to available processors in \( I^{w} \). Next, we check the case that \( T_{k} \) uses available processors in any \( I^{w} \) such that \( e^{i} < e^{w} < d_{k} \) in order to compensate for its loss, which is described as arrow \( B \) in Fig. 4. If \( \eta_{k}^{w} > 0 \) and \( \Phi^{w} < \sum \delta^{w} \) for any \( T_{k} \in SL^{w} \) such that \( a_{k} \leq e^{i} \leq \text{min}(e^{w}, d_{k}) \) when scheduling \( SL^{w} \) at \( P \), then \( \eta_{k}^{w} = b_{k} \) for any \( I^{w} \) such that \( e^{i} < e^{w} < \text{min}(e^{w}, d_{k}) \) and \( \eta^{w} < N^{w} \), and \( c_{k} = b_{k} \cdot (d_{k} - e^{w}) \) when \( e^{i} < d_{k} \) by Lemma 3. Hence, the lost computation amount of \( T_{k} \) in \( I^{i} \) cannot be allocated to available processors in any \( I^{w} \) such that \( e^{i} < e^{w} < d_{k} \).

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 \( T_{j} \) and \( T_{k} \) 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.

\[ \blacksquare \]

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

\[ \text{proof:} \quad \text{When MinProc-Scheduling finds a feasible schedule using } N^{i} \text{ processors, let us assume that there is a feasible schedule using } N^{w} \text{ processors such that } N^{w} < N^{i}. \text{ If Feasible-Scheduling finds a feasible schedule on } N^{w} \text{ processors, then Feasible-Scheduling can find a feasible schedule on } N^{w} \text{ processors for any } N^{w} \geq N^{i}. \text{ If MinProc-Scheduling selects a feasible schedule using } N^{w} \text{ processors, then this means that Feasible-Scheduling failed to find the feasible schedule on } N^{w} \text{ processors such that } N^{w} > N^{w} \geq N^{i}. \text{ Then it is a contradiction of Theorem 1.} \]

\[ \blacksquare \]

References


