Conservative Modeling of Shared Resource Contention for Dependent Tasks in Partitioned Multi-Core Systems

Junchul Choi, Donghyun Kang and Soonhoi Ha
Department of Computer Science and Engineering, Seoul National University, Seoul, Korea,
{hinomk2, kangdongh, sha}@iris.snu.ac.kr

***Abstract*—In a multi-core system with shared resources, the accesses to the shared resources from several cores may experience non-deterministic arbitration delay due to resource contention. Such delay should be considered conservatively in the worst case response time (WCRT) analysis of multi-core systems. Recently, several techniques have been proposed to account for arbitration delay for shared resource contention, based on the event stream modeling of resource access. While they all assume independent tasks, in this paper, we propose a conservative modeling technique of shared resource contention supporting dependent tasks. To find a tight upper bound of arbitration delay, we derive a shared resource demand bound for each processing element, considering the task dependency. The proposed technique is not specific to a particular WCRT analysis method, and supports both preemptive and non-preemptive scheduling policy. In the experiments, the significance of considering data dependency of parallel applications and the performance of our technique are verified by synthetic examples and a real-life example.**

I. Introduction

Multi-core systems are being introduced into the real-time embedded systems, replacing the single-core systems that do not provide scalable performance increase anymore. The automotive systems are representative of the emerging multi-core real-time systems. Multi-core electronic control units (ECUs) are increasingly used for complicated automotive applications such as In-Vehicle Infotainment (IVI) and Advanced Driver Assistance Systems (ADAS) whose executions can be parallelized to achieve higher performance than sequential execution on a single core.

When running multiple real-time applications on a multi-core system, there are two types of scheduling policies: partitioned scheduling and global scheduling. When a partitioned scheduling is used, the task-to-core mapping is not changed at run-time. On the other hand, tasks can be migrated over cores when they are globally scheduled. Since a global scheduling system accompanies non-negligible overhead of task migration and scheduling, partitioned scheduling is more preferred in practical real-time embedded systems. In this paper, we are concerned with a partitioned multicore system.

For the design of real-time embedded systems, it is required to guarantee the satisfaction of real-time constraints. In the long history of research over two decades, various techniques have been proposed to formally analyze the schedulability of a single core system. Even though schedulability analysis of a multi-core system gains extensive research focus recently, the inter-core interference due to shared resource contention and dependency between tasks have not been fully addressed in the previous work. Since multi-core systems usually consist of shared resources such as shared L2 cache, memory, and peripheral devices, it is very likely for the applications to experience non-deterministic resource access delay due to shared resource contention caused by simultaneous accesses from several cores unless time division multiplexing or fully-static scheduling of resource accesses is used. Resource contention cannot be estimated in the conventional WCET analysis technique that considers each processor alone. Thus it is very challenging to conservatively but accurately estimate the non-deterministic arbitration delay of shared resources.

Several techniques have been proposed recently for considering the shared resource contention in the schedulability analysis [1][2][3]. Their techniques are based on the event stream model to capture the resource access pattern of a task. They all assume that independent tasks are scheduled in a partitioned multi-core system, implying that each task corresponds to a sequentially executed application. On the other hand, we incorporate parallelized execution of an application in the schedulability analysis.

In this paper, we model an application as a task graph scheduled in a partitioned multi-core system in which each core has either preemptive or non-preemptive fixed-priority scheduling policy. We propose a novel modeling technique for shared resource contention to find a tight and conservative upper bound of shared resource access delay. To the best of our knowledge, it is the first approach to consider task dependency in the conservative estimation of shared resource contention based on event stream model. Extensive experiments with synthetic examples and a practical example prove the significance of considering data dependency of parallel applications for the modeling of shared resource contention delay.

II. Related Work

The schedulability analysis of a partitioned multi-core system may use the traditional schedulability analysis of each core separately if shared resource contention does not exist. The schedulability analysis for a single core system is mature enough to consider various issues such as arbitrary deadline [4], release jitter [5], dynamic offset [6], non-preemptive scheduling policy [7], and precedence constraints [8].

The response time analysis with shared resources based on the Time Division Multiple Access (TDMA) protocol is

978-3-9815370-6-2/DATE16/

proposed in [9]. TDMA protocol simplifies the timing analysis since the response time of a resource access can be computed without considering the accesses from other cores, but may not be efficient in the view of resource utilization. On the other hand, the approaches in [1], [2], and [3] consider the shared resources that are arbitrated with a fixed priority non-preemptive policy. Using the event stream model, they compute an upper bound of resource access requests from the other cores that may contend with the task of interest. However, these approaches only consider independent tasks meaning that each application should be sequentially executed.

A few approaches have been proposed to consider task dependency to some extent. The technique proposed in [10] divides a task into several super-blocks that have different resource access patterns. Though super-blocks can be regarded as dependent sub-tasks, they assume simple scheduling policy that is static time slot assignment. There is a technique that analyzes the worst case cache access scenario of parallelized application modeled by Message Sequence Graph (MSG) [11]. This approach has huge time complexity since all cache references are analyzed considering the cache states and replacement policy. In addition, it assumes that the cache access behaviors are known and finite. Compared with those approaches, we use more general models, an event stream model for resource access and a task graph model for dependent tasks, in order to support a wide range of resource access patterns and parallelized execution of an application.

III. Problem Definition

A system consists of a set of processing elements (PEs) and a set of shared resources (SRs). We assume that all PEs use partitioned scheduling so that task mapping is not varying at run time. The scheduling policy of a PE can be either a fixed-priority preemptive scheduling ($\mathcal{P}$) or a fixed-priority non-preemptive scheduling ($\mathcal{N}$). SRs are globally accessed by the tasks executed in all PEs. We assume that the accesses to a shared resource are arbitrated in a non-preemptive fashion with priorities of issuing tasks and the task that attempts to access a shared resource waits without being preempted until the access is processed.

An input application, G_i, is represented as an acyclic task graph. If a task has more than one input edge, it is released after all predecessor tasks are completed. An application G can be initiated periodically or sporadically, characterized by a tuple (G^p, G^j) where G^p and G^j represent the period and the jitter, respectively. For sporadic activation, G^p denotes the minimum initiation interval. Task graph G is given relative deadline G^d to meet once activated.

A task is a basic mapping unit onto a processing element. We assume that task mapping is given and fixed. The processing element that the task τ is mapped to is denoted by τ^{proc}. For each task τ, the varying execution time is represented as a tuple $[\tau^{BCET}, \tau^{WCET}]$ indicating the lower and the upper bound on the mapped PE, which are given bounds analyzed with the ideal assumption that there is no shared resource contention. We assume that all tasks in the whole system have distinct priorities to make the scheduling order deterministic. The priority of the task τ is denoted by τ^{pri}. The task graph that task τ belongs to is denoted by G_τ. The accesses to a shared resource during the task execution are characterized by a tuple $(\tau^n_{i,j}, \tau^w_{i,j}, \tau^d_{i,j})$ where $\tau^n_{i,j}$, $\tau^w_{i,j}$, and $\tau^d_{i,j}$ represent the maximum number of accesses, the maximum access duration, and the minimum distance between two accesses to shared resource SR_j during the task τ_i execution, respectively.

With the given information of task mapping and task execution profiles for task execution times and shared resource accesses, we aim to compute a tight upper bound of resource access delay each task experiences during the execution.

IV. Proposed Technique

In this section, we propose a novel conservative modeling technique of SR contention supporting dependent tasks.

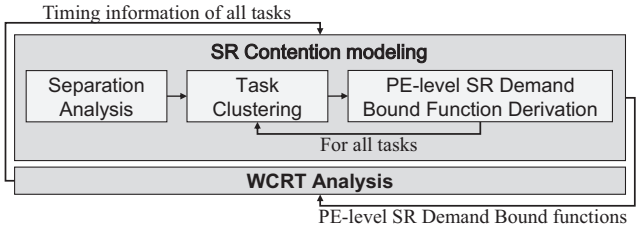


Fig. 1. The overall framework of the proposed technique

As displayed in Fig. 1, the proposed methodology consists of two modules: SR contention modeling and WCRT analysis. Since two modules are dependent on each other, the methodology forms an iterative loop until converged. The SR contention modeling module computes the upper bound of resource demand from each PE that may block the access of a task to each shared resource. This module needs two pairs of time bound information for each task, the earliest and latest release time pair (τ_i^{minR}, τ_i^{maxR}) and the earliest and latest finish time pair (τ_i^{minF}, τ_i^{maxF}), which are obtained from the WCRT analysis module. The WCRT analysis module computes the worst case response time of each task considering the shared resource contention using the SR demand bound functions generated from the SR contention modeling module. In this paper, we focus on the analysis of the worst case shared resource access delay, which corresponds to SR contention modeling module. Our methodology can be applied to any WCRT analysis technique if it supports dependent tasks and gives required time bound information [12][13].

Since tasks mapped to a same PE cannot be executed in parallel, we derive a SR demand bound function per PE by considering the mapped tasks all together to find a tight upper bound of arbitration delay.

Definition 1: $D_{S,i}(\Delta t)$ is the maximum amount of resource accesses that may be issued by tasks in a task set S to a shared resource SR_i within a time window of size Δt.

For each task τ_c, the SR contention modeling module derives $D_{S,i}(\Delta t)$ for a set of tasks that have higher priority than τ_c and have possibility to be executed in parallel with τ_c in each PE. It consists of three submodules: Separation Analysis, Task Clustering, and PE-level SR Demand Bound Function

Derivation, as shown in Fig. 1. In the Separation Analysis, we compute the minimum distance between two tasks in the same task graph and check whether they can be simultaneously executed or not. In case two tasks are completely separated, meaning that they cannot be simultaneously executed at all times, we need not consider the resource demand of the other task. In the Task Clustering, we make a cluster of tasks that will be executed sequentially to consider their resource demands all together. Finally PE-level SR Demand Bound functions are derived considering the task execution order of each cluster in the last sub-module. Now we explain each submodule in detail.

A. Separation Analysis

Two tasks may be completely separated due to dependency in a task graph. If two tasks have a parent-child relationship, they are definitely separated. Between two tasks without direct dependency, we still can check if they are completely separated based on two pairs of time bound information, (τ_i^{minR}, τ_i^{maxR}) and (τ_i^{minF}, τ_i^{maxF}), for each task τ_i. Separation analysis is accomplished by computing the minimum possible scheduling distance between two tasks in the same task graph.

We define $Dist_{i,j}$ as the minimum distance from the finish time of τ_i to the release time of τ_j. Note that the sign of $Dist_{i,j}$ means the direction. If it is non-negative, the finish time of τ_i is no later than the release time of τ_j. Otherwise, τ_i may be simultaneously executed with τ_j. Since the task release and finish times are computed based on the application release time, only time bounds of tasks in the same application are comparable. $Dist_{i,j}$ is formulated with the following equation:

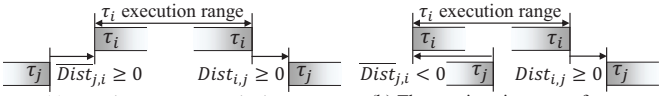
$$Dist_{i,j} = \begin{cases} \tau_i^{minR} - \tau_i^{minF}, & \text{if } i = j \\ \max\limits_{\tau_s \in suc[\tau_i]} \begin{pmatrix} \tau_s^{minF} - \tau_s^{minR} \\ +Dist_{s,j} \end{pmatrix}, & \text{else if } \tau_i \in anc[\tau_j] \\ \tau_j^{minR} - \tau_i^{maxF}, & \text{else if } G_{\tau_i} = G_{\tau_j} \\ -\infty, & \text{otherwise} \end{cases} \tag{1}$$

where $suc[\tau_i]$ and $anc[\tau_j]$ denote a set of immediate successors of τ_i and a set of ancestors of τ_j, respectively. $Dist_{i,i}$ is set to $\tau_i^{minR} - \tau_i^{minF}$, which is the negative value of the best case response time of τ_i. If task τ_i is an ancestor of τ_j, the minimum distance can be computed recursively by computing the minimum distance from each successor of τ_i. To compute the minimum distance, we add the best case response time of each successor to the distance from the successor to τ_j. In case there are multiple dependency paths from task τ_i to τ_j, we choose the maximum value. In case two tasks τ_i and τ_j belong to the same task graph and have no direct dependency, we compute the minimum distance by the difference of earliest release time of τ_j and the latest finish time of τ_i for conservative computation. If no aforementioned case is satisfied, the distance is set to a large negative value, implying that there is no dependency between two tasks if they belong to different task graphs.

In case the deadline is larger than the period of an application, the instances from different iterations may affect each other. Although a task instance has a positive distance from another task in the current iteration, its execution may overlap with the task of the previous iteration. In order to examine the possibility of overlap between task instances that belongs to different iterations, we define another distance variable $\overline{Dist}_{j,i}$ to indicate the minimum distance from the finish time of τ_j of the previous iteration to the release time of τ_i in the current iteration as follows:

$$\overline{Dist}_{j,i} = \begin{cases} \tau_i^{minR} - \left(\tau_j^{maxF} - G_{\tau_j}^p\right), & \text{if } G_{\tau_i} = G_{\tau_j} \\ -\infty, & \text{otherwise} \end{cases} \tag{2}$$

If both $Dist_{i,j}$ and $\overline{Dist}_{j,i}$ are non-negative, there is no overlap between two tasks as illustrated in Fig. 2 (a). If $\overline{Dist}_{j,i}$ is negative while $Dist_{i,j}$ is positive, the previous iteration may affect the execution of τ_i, which is the case of Fig. 2 (b).



(a) τ_i and τ_j are separated when $Dist_{i,j} \geq 0$ and $\overline{Dist}_{j,i} \geq 0$

(b) The previous instance of τ_j may overlap with τ_i when $\overline{Dist}_{j,i} < 0$

Fig. 2. $Dist_{i,j}$ and $\overline{Dist}_{j,i}$ computation for separation analysis

B. Task Clustering

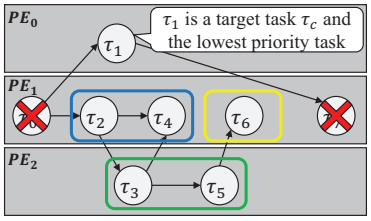
1: $Cand_{c,i,j} \leftarrow \left\{ \tau_s \;\middle|\; \begin{array}{c} \tau_s^{proc} = PE_i, \tau_s^{pri} > \tau_c^{pri}, \tau_{s,j}^n \neq 0, \\ \left(Dist_{c,s} < 0 \text{ or } \overline{Dist}_{s,c} < 0\right), \\ \left(Dist_{s,c} < 0 \text{ or } \overline{Dist}_{c,s} < 0\right) \end{array} \right\}$
2: Sort $Cand_{c,i,j}$ in topological order, $CS_{c,i,j} \leftarrow \emptyset$
3: **while** $Cand_{c,i,j} \neq \emptyset$ **do**
4: $\quad \tau_l \leftarrow$ leftmost task in a set $Cand_{c,i,j}$
5: $\quad \tau_r \leftarrow \tau_l$, $C \leftarrow \{\tau_l\}$, $Cand_{c,i,j} \leftarrow Cand_{c,i,j} - \{\tau_l\}$
6: $\quad$ **repeat**
7: $\quad\quad R \leftarrow \{\tau_s | \tau_s \in Cand_{c,i,j}, Dist_{r,s} \geq 0, \overline{Dist}_{s,l} \geq 0\}$
8: $\quad\quad \tau_r \leftarrow \tau_k$ that $\tau_k \in R$ and $Dist_{r,k} = \min_{\tau_s \in R} Dist_{r,s}$
9: $\quad\quad$ **if** $R \neq \emptyset$ **then**
10: $\quad\quad\quad$ Move τ_r from $Cand_{c,i,j}$ to the rightmost in C
11: $\quad\quad$ **end if**
12: $\quad$ **until** $R = \emptyset$
13: $\quad$ Add task cluster C to task cluster set $CS_{c,i,j}$
14: **end while**
15: **return** $CS_{c,i,j}$

Fig. 3. Task Clustering algorithm with three input parameters τ_c, PE_i, SR_j

In the task clustering module sketched in Fig. 3, we first collect tasks into a set $Cand_{c,i,j}$ that can be simultaneously executed with the target task τ_c in PE_i and have accesses to SR_j (line 1). If $\left(Dist_{c,s} \geq 0, \overline{Dist}_{s,c} \geq 0\right)$ or $\left(Dist_{s,c} \geq 0, \overline{Dist}_{c,s} \geq 0\right)$, τ_s is always separated from τ_c so that it does not contribute to the shared resource contention of τ_c.

Among the chosen tasks, we make a cluster of tasks that will be executed in order in any case; tasks in the cluster do not interfere with each other. By considering a cluster as a whole, we can derive a tighter bound on the overall resource demand than considering the resource demand of each task separately since we can analyze the distance between resource accesses issued in the cluster.

From the candidate task set $Cand_{c,i,j}$, we make an ordered task cluster C in the execution order of tasks. After a task is chosen and moved to the cluster C (lines 4 and 5), we find a task that is separated from the current ordered cluster C and has the closest distance from the rightmost task in cluster C (lines 7-8). If such a task is found, it is moved from $Cand_{c,i,j}$



(a) A task graph and PE mapping

	$(\tau_i^{minR}, \tau_i^{maxR})$	$(\tau_i^{minF}, \tau_i^{maxF})$
τ_0	(0,0)	(10,10)
τ_1	(10,10)	(60,60)
τ_2	(10,10)	(20,20)
τ_3	(20,20)	(30,30)
τ_4	(30,30)	(40,50)
τ_5	(30,30)	(35,70)
τ_6	(35,70)	(45,80)
τ_7	(60,60)	(70,80)

(b) Time bound information

Fig. 4. An example of Task Clustering

to cluster C (lines 9-11). This procedure is repeated until there is no separated task (line 12). Task Clustering procedure finishes when all candidate tasks cluster (line 3).

Consider the example of Fig. 4 where tasks are mapped to three PEs. For simple illustration, we assume that every task accesses the same resource SR_j and the deadline is not greater than the period so that $\overline{Dist}_{i,j}$ is always positive. Let task τ_1 be the target task τ_c and the lowest priority task. τ_0 and τ_7 are excluded from $Cand_{1,1,j}$ since they have parent-child relationship with τ_1. Then τ_2 is selected as the first element of cluster. $Dist_{2,4} = \max(\tau_3^{minF} - \tau_3^{minR} + Dist_{3,4}, \tau_4^{minF} - \tau_4^{minR} + Dist_{4,4}) = \max(10 + (10 - 10), 0) = 10$ and $Dist_{2,6} = \tau_6^{minR} - \tau_2^{maxF} = 35 - 20 = 15$ so that τ_4 and τ_6 are in R. Since τ_4 is closer than τ_6, τ_4 is added to the cluster. τ_6 constructs its own cluster since $Dist_{4,6} = \tau_6^{minR} - \tau_4^{maxF} = 35 - 50 = -15$. Similarly for PE_2, τ_3 and τ_5 cluster together because $Dist_{3,5} = 0$.

C. PE-level SR Demand Bound Function Derivation

After all task clusters are formed, we construct the PE-level SR demand bound function by distributing the time window Δt to each task cluster for execution.

Since one PE can be occupied by only one task at a time, there are plenty of task execution scenarios that fill a time window Δt. Then the problem of computing the maximum shared resource demand bound issued in a PE within a time window Δt becomes the problem that finds the combination of task cluster execution amounts to produce the maximum shared resource demand among all possible combinations. The PE-level SR Demand Bound function is formulated as follows:

$$D_{CS,j}(\Delta t) = \max \left\{ \sum_{C \in CS} D_{C,j}(t_C, \Delta t) \;\middle|\; \sum_{C \in CS} t_C = \Delta t \right\} \quad (3)$$

where CS, t_C, and $D_{C,j}(t_C, \Delta t)$ denote a task cluster set obtained from Task Clustering, a time amount allocated for execution of task cluster C, and a function that finds the maximum shared resource demand bound when task cluster C executes t_C amount of time in a time window Δt, respectively. $D_{CS,j}(\Delta t)$ can be obtained by the max-plus convolution of individual functions $D_{C,j}(t_C, \Delta t)$ per cluster in polynomial time since the max-plus convolution has associative property and commutative property. Then the remaining problem is to derive a function $D_{C,j}(t_C, \Delta t)$ for each cluster.

Let τ_{C_i} be the i-th task in cluster C and n be the number of tasks in cluster C. Since a cluster C includes a set of tasks that is executed sequentially, the worst case cluster execution can be represented as a weighted cyclic graph as shown in Fig. 5 (a). An edge has a weight that means the distance between two

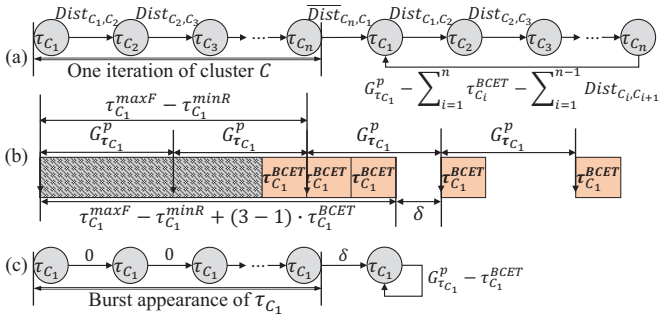


Fig. 5. The cluster execution flow graph with weighted edge

tasks. Note that we assume that two iterations of cluster execution have the minimum inter-iteration distance $\overline{Dist}_{C_n,C_1}$ to obtain the worst-case resource demand. The subsequent iterations appear periodically as indicated by the feedback arc from the rightmost task to the leftmost task in the cluster, with the edge weight $G^p_{\tau_{C_1}} - \sum_{i=1}^{n} \tau_{C_i}^{BCET} - \sum_{i=1}^{n-1} Dist_{C_i,C_{i+1}}$.

Note that the minimum inter-iteration distance can be negative. In this case, cluster C consists of only one task. Then some task instances may appear in a bursty fashion as shown in Fig. 5 (b). Let the number of burst instances x. Then the first positive distance, δ in Fig. 5 (b), can be computed as $G^p_{\tau_{C_1}} \cdot x - \{(\tau_{C_1}^{maxF} - \tau_{C_1}^{minR}) + (x-1) \cdot \tau_{C_1}^{BCET}\}$. Since it is positive value, the number of burst instances becomes $x = \lceil ((\tau_{C_1}^{maxF} - \tau_{C_1}^{minR}) - \tau_{C_1}^{BCET}) / (G^p_{\tau_{C_1}} - \tau_{C_1}^{BCET}) \rceil$. The subsequent instances appear periodically. Finally, the corresponding execution flow graph for burst execution is modeled as Fig. 5 (c).

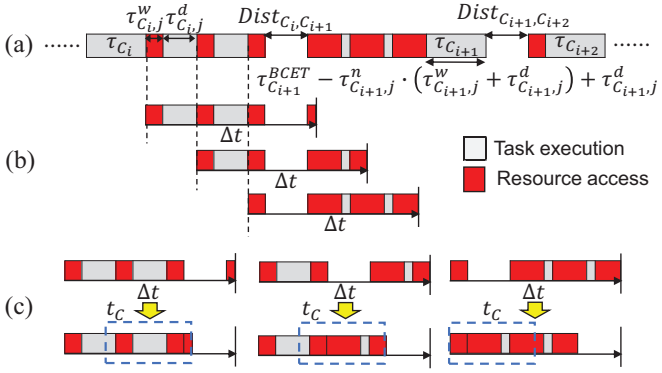


Fig. 6. Maximum resource demand computation for execution time amount t_C and a time window Δt

With the information of cluster execution flow graph described above, We can compute $D_{C,j}(t_C, \Delta t)$ that is the maximum resource demand issued by tasks in a cluster C when cluster C executes t_C time amount within a time window Δt. Since the absolute resource access times are not known in our application model, we assume the worst case pattern of shared resource accesses; all accesses of the first task in time window may be concentrated at the end of execution with the minimum access distance while the resource accesses in subsequent tasks are concentrated at the start of execution, as shown in Fig. 6 (a).

We set all resource access points as candidate starting points of time window, as illustrated in Fig. 6 (b). Consider task τ_{C_i}

in the cluster in Fig. 6. Since it has three resource accesses, there are three candidate starting points associated with task τ_{C_i}. For each time window, we should consider the amount of resource demand when cluster C executes the allocated time t_C that includes only the sum of the net execution times of cluster, ignoring the idle time between tasks, as illustrated in Fig. 6 (c). Finally, $D_{C,j}(t_C, \Delta t)$ is chosen as the maximum resource demand among all candidate time windows.

V. Experiments

We implemented the proposed SR contention modeling technique with our own WCRT analysis technique [14].

In the first set of experiments, we vary the configurations of the proposed technique as shown in Table I to verify the effectiveness of two key modules of the proposed technique: Task Clustering and PE-level SR Demand Bound function. If PE-level SR Demand Bound function is not used, the Task-level(or Cluster-level) SR Demand Bound functions are simply summated for computing resource contention. If Task Clustering is not used, the tasks are regarded as independent tasks so that each task constructs its own cluster.

TABLE I
Configurations for verification of the key techniques

Configuration	Task Clustering	PE-level SR Demand Bound function
Config1	Not used	Not used
Config2	Not used	Used
Config3	Used	Not used
Config4	Used	Used

We use 100 synthetic examples that have dependent tasks. The number of PEs varies from 3 to 5, and the number of SRs from 3 to 4. The scheduling policy of a PE is either $\mathcal{P}$ or $\mathcal{N}$. The number of task graphs varies from 2 to 3 and the number of total tasks varies from 20 to 30. τ^{BCET} and τ^{WCET} of each task are randomly selected in the range of $[1000, 1500]$ and $[\tau^{BCET}, \tau^{BCET} \times 1.5]$. SRs that a task accesses are also randomly selected. When a task accesses a shared resource, the number of maximum accesses varies from 10 to 40 and the maximum access duration from 1 to 5. The minimum access distances and the periods are randomly chosen.

The normalized shared resource contention for the generated 100 examples are displayed in Fig. 7, while the results are normalized by the result of Config1. The shared resource contention is computed by the difference of the estimated WCRT with resource contention and the estimated WCRT without resource contention. From the results of Config2 and Config3 that use only one of two techniques, we observe that one technique does not dominate the other since they reduce different parts of overestimation. The task clustering technique works well if the remote tasks that may induce contentions are serially executed or separated. On the other hand, the PE-level SR Demand Bound function reduces overestimation by considering the possible resource demand from PE. On average, however, the task clustering method gives more performance gain. Table II presents the best, worst, and average performance estimation ratio of Config2, Config3, and Config4, respectively, compared to Config1. Config2 and Config3 reduce the estimation by 25.42% and 36.72%, respectively on average. The proposed technique reduces the estimation of shared resource contention dramatically, on average by 54.32%.

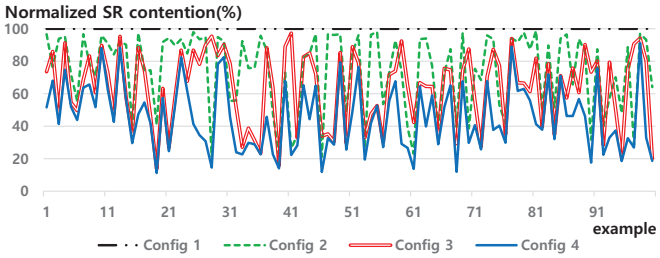


Fig. 7. Normalized SR contention graph for four configurations

TABLE II
The best, worst, and average performance

Configuration	Min	Max	Avg
Config2	21.05%	98.67%	74.58%
Config3	13.79%	97.20%	63.28%
Config4	11.19%	91.25%	45.68%

The next set of experiments is conducted to examine the effect of resource access pattern to the estimation performance by making memory access request more sparse but with longer duration. The number of maximum accesses varies from 1 to 2 and the maximum access duration from 100 to 250. Fig. 8 shows that the task clustering technique becomes the dominant source of performance improvement in almost all examples. Moreover the performance gap between Config1 and Config4 is reduced. If the resource accesses are infrequent and have long access duration, tasks in a PE may demand a resource continuously without pause. In that case, the overall resource demand from a PE becomes very close to the simple summation of the resource demands of all tasks, making the PE-level SR Demand Bound derivation meaningless.

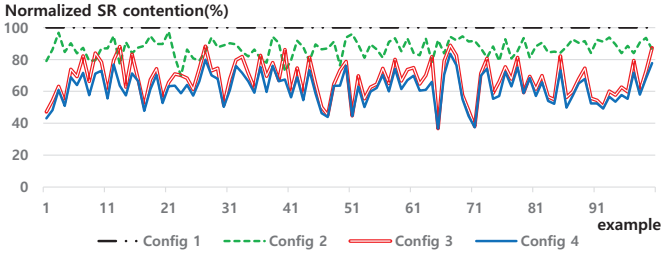


Fig. 8. Normalized SR contention graph for four configurations with sparse resource access pattern

In the last set of experiments, we analyze the worst case response time of parallelized viola-jones object detection as a real-life example. After the initialization step (τ_0), each detection task (τ_1 - τ_{17}) independently proceeds using cascaded classifiers on each downsampled integral image from 640x480 image resolution. After detection process finished, the results are combined in the finalization step (τ_{18}). We profile the task information of execution cycle and cache miss pattern using our own simulator [15] with a system that has a 32KB L1 cache. The information is summarized in Table III. $\tau^w_{i,0}$ and $\tau^d_{i,0}$ for all tasks are commonly 200 and 1, respectively.

With the profiled task information, we estimate the worst case response time when tasks are executed on a quad-core system. Fig. 9 shows the estimated WCRTs while varying

TABLE III
PROFILED TASK INFORMATION (UNIT OF BCET AND WCET:CYCLE)

Task	τ_i^{BCET}	τ_i^{WCET}	$\tau_{i,0}^{n}$	Task	τ_i^{BCET}	τ_i^{WCET}	$\tau_{i,0}^{n}$	Task	τ_i^{BCET}	τ_i^{WCET}	$\tau_{i,0}^{n}$
τ_0	28,119,576	28,120,030	8,536	τ_7	57,669,176	94,305,609	233,658	τ_{13}	4,650,264	6,989,643	11,785
τ_1	541,960,280	763,061,294	2,475,671	τ_8	38,615,013	63,730,841	146,647	τ_{14}	2,740,420	5,371,356	11,017
τ_2	372,602,070	531,171,589	1,248,607	τ_9	25,695,213	43,626,677	100,296	τ_{15}	1,714,636	3,546,796	14,028
τ_3	258,487,562	380,000,067	888,365	τ_{10}	17,566,664	28,655,096	60,764	τ_{16}	1,183,579	1,598,812	6,073
τ_4	181,527,732	284,875,102	727,991	τ_{11}	11,347,916	18,238,489	50,159	τ_{17}	889,162	946,998	4,199
τ_5	126,202,815	201,385,108	567,993	τ_{12}	7,136,979	11,560,672	48,977	τ_{18}	14,962,401	15,198,446	5,492
τ_6	85,613,575	139,997,737	522,943								

TABLE IV
TWO TASK MAPPING CONFIGURATIONS

Configuration		Tasks(index)	Configuration		Tasks(index)
Mapping1	PE_0	0, 1	Mapping2	PE_0	0, 1, 5, 12
	PE_1	2		PE_1	2, 7, 10, 14, 16, 17
	PE_2	3, 4		PE_2	3, 9, 11, 13, 15
	PE_3	5-17, 18		PE_3	4, 6, 8, 18

the task mapping. No SR contention means that the WCRT is estimated without considering shared resource contention. Two mapping configurations Mapping1 and Mapping2 are described in Table IV. The priorities are assigned by the following rule: $\tau_i^{pri} > \tau_j^{pri}$ if $(i < j, \tau_i^{proc} = \tau_j^{proc})$ or $(\tau_i^{proc} = PE_k, \tau_j^{proc} = PE_l, k < l)$. If a traditional analysis technique that does not consider shared resource contention is used, Mapping1 would be regarded as better than Mapping2 as shown in Fig. 9. On the other hand, the WCRT of Mapping2 considering shared resource contention is smaller than that of Mapping1. This implies the importance of considering shared resource contention when exploring the design space of a multi-core system. For both mapping configuration, the estimated WCRTs are almost half of the WCRT when the application is sequentially executed.

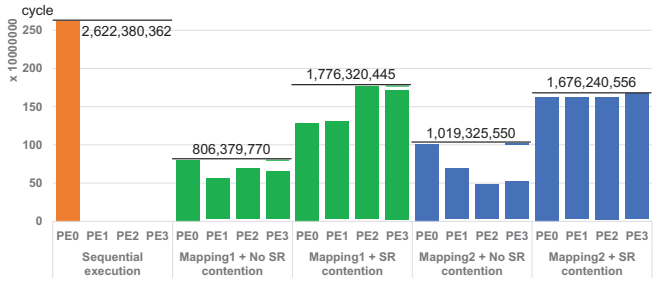


Fig. 9. Estimated WCRTs of a viola-jones object detection example while task mapping is varied

VI. CONCLUSION

In this paper, we propose a novel conservative modeling technique of shared resource contention supporting dependent tasks. We identify the tasks that will not incur any access contention in any case by analyzing the schedule distance between tasks. And we cluster tasks that will be scheduled in a fixed order in each processing element. Finally we obtain the PE-level SR demand bound by finding the worst combination of the SR demand from task clusters in each PE. The proposed methodology is not specific to a particular WCRT analysis method, and supports both preemptive and non-preemptive scheduling policy. Experimental results with synthetic examples and a real-life example confirm the significance of considering the data dependency in the modeling of shared resource contention.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning(NRF-2013R1A2A2A01067907). The ICT at Seoul National University provides research facilities for this study.

REFERENCES

[1] M. Negrean, S. Schliecker, and R. Ernst. (2009. Apr.) *Response-Time Analysis of Arbitrarily Activated Tasks in Multiprocessor Systems with Shared Resources*, pp. 524–529. Design, Automation and Test in Europe Conference and Exhibition (DATE).

[2] M. Negrean and R. Ernst. (2012. Jun.) *Response-Time Analysis for Non-Preemptive Scheduling in Multi-Core Systems with Shared Resources*, pp. 191–200. Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on.

[3] S. Schliecker and R. Ernst. (2010. Dec.) *Real-Time Performance Analysis of Multiprocessor Systems with Shared Memory* , Vol. 10. ACM Transactions on Embedded Computeting Systems.

[4] K. Tindell and J. Clark. (1994. Apr.) *Holistic Schedulability Analysis of Distributed Hard Real-Time Systems*, Vol. 40, pp. 117–134. Microprocessing and microprogramming.

[5] N. Audsley, A. Burns, K. Tindell, M. Richardson and A. Wellings. (1993. Sep.) *Applying New Scheduling Theory to Static Priority Preemptive Scheduling*, Vol. 8, pp. 284–292. Software Engineering Journal.

[6] J. C. Palencia and M. Gonzlez Harbour. (1998. Dec.) *Schedulability Analysis for Tasks with Static and Dynamic Offsets*, Vol. 8, pp. 26. Proceedings of the IEEE Real-Time Systems Symposium.

[7] E. Bini and G. C. Buttazzo. (2004. Nov.) *Schedulability Analysis of Periodic fixed Priority Systems*, Vol. 53, pp. 1462–1473. IEEE Transactions on Computers.

[8] J. C. Palencia and M. Gonzlez Harbour. (1999. Dec.) *Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-time Systems*, pp. 328–339. Proceedings of the 20th IEEE Real Time Systems Symposium.

[9] A. Schranzhofer, R. Pellizzoni, J. Chen, L. Thiele, and M. Caccamo. (2010, Jun.) *Worst-Case Response Time Analysis of Resource Access Models in Multi-Core Systems*, pp. 332–337. Proceedings of the 47th Annual Design Automation Conference.

[10] R. Pellizzoni, A. Schranzhofer, J. Chen, M. Caccamo, and L. Thiele. (2010. Mar.) *Worst Case Delay Analysis for Memory Interference in Multicore Systems*, pp. 741–746. Proceedings of Design, Automation & Test in Europe Conference & Exhibition.

[11] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury. (2009. Dec.) *Timing Analysis of Concurrent Programs Running on Shared Cache Multi-Cores*, pp. 57-67. Proceedings of the 30th IEEE Real-Time Systems Symposium.

[12] J. Kim, H. Oh, J. Choi, H. Ha, and S. Ha. (2013. Jun.) *A Novel Analytical Method for Worst Case Response Time Estimation of Distributed Embedded Systems*, pp. 1–10. Proceedings of the 50th Annual Design Automation Conference.

[13] T. Yen and W. Wolf. (1998, Nov.) *Performance Estimation for Real-Time Distributed Embedded Systems*, Vol. 9, pp. 1125–1136. IEEE Transactions on Parallel and Distributed Systems.

[14] J. Choi, H. Oh, and S. Ha. (2014. Jun.) *SWAT: A Suite of Worst Case Response Time Analysis Tools for Distributed Hard Real-Time Systems*, retrivable from http://iris.snu.ac.kr/xe/swat

[15] S. Kang, S. Ha, (2013. Jan.) *TQSIM: Timed QEMU-based Simulator*, retrivable from https://github.com/mseed2/tqsim