Real-Time Task Scheduling on Island-Based Multi-Core Platforms

Che-Wei Chang, *Member, IEEE*, Jian-Jia Chen, Tei-Wei Kuo, *Fellow, IEEE*, and Heiko Falk

Abstract—With the increasing number of cores in a computing system, how to coordinate the computing units and heterogeneous memory resources has soon become extremely critical for real-time systems. This paper explores the joint considerations of memory management and real-time task scheduling over island-based multi-core architecture, where the local memory module of an island offers shorter access time than the global memory module does. The objective of this work is to minimize the number of needed islands to successfully schedule real-time tasks. When the required amount of the local memory space is specified for each task, a scheduling algorithm is proposed to provide an asymptotic $\frac{29}{9}$-approximation bound. When there is flexibility in determining the needed local memory space for each task, we propose an algorithm with an asymptotic 4-approximation bound to jointly manage memory resources and allocate computing cores. In addition to the worst-case approximation analysis, the proposed algorithms are also evaluated with 82 real-life benchmarks with the support of a worst-case execution time analyzer. Moreover, extensive evaluations are conducted to show the capability of the proposed approaches when being used with various computing cores and memory resources.

Index Terms—Real-time system, multi-core architecture, heterogeneous memory, task scheduling, memory allocation

1 Introduction

In order to address the demands in processor performance, vendors had been focusing on the technologies in increasing the processor clock rate, whereas these technologies had led to critical design problems in recent years, due to extremely high power consumption and heat dissipation [20]. Although the adoption of multiple cores has been proven as an effective way to resolve the power-consumption and thermal problems [15], system engineers now face serious challenges in developing effective memory architecture [14], [21], [23], [36] for systems with a rapidly increasing number of cores. For multi-core embedded systems, there is an advanced architecture design which includes off-chip DRAM as the global memory and adopts on-chip SRAM as the fast local memory [18]. Cluster computers, such as blade servers [31], also serve as a good example in the throughput improvement of large-scale applications by deploying both local and remote memory modules in the cost and performance tradeoff. With the popular heterogeneous-memory design and multi-core architecture in mind, the goal of this work is to jointly consider task scheduling and memory allocation for multi-core real-time system synthesis.

Traditionally, there have been two major approaches to the multiprocessor real-time task scheduling, which are *global* [7] and *partitioned* [6], [12] scheduling schemes. In global scheduling, all task instances are put into a global queue, and a global scheduler fetches a task instance from the queue to an available processor according to some criteria, such as the deadlines of task instances. Regarding partitioned scheduling, tasks are statically assigned onto processors such that any instance of a task is only executed on the designated processor. There are also some hybrid approaches, such as *semi-partitioned* scheduling [28], in which a task could be statically partitioned into subtasks and assigned to processors under some constraints, such that the subtasks of a task must execute at different time slots. For more details of multiprocessor scheduling, we refer readers to a comprehensive survey in [15]. However, most of the traditional scheduling algorithms assume the (worst-case) execution time of each real-time task is given and fixed, and do not consider the memory architecture with heterogeneous memory modules. Thus, the emergence of heterogeneous memory designs has renewed the research of multiprocessor real-time scheduling.

To analyze the response time of a real-time task on systems with heterogeneous memory, system developers have to consider not only task scheduling but also the impact of memory allocation on the worst-case execution time (WCET) of the task. When cache is included, the work in [10] analyzes all potential cache conflicts to quantify the possible swapping overheads during the task execution on single and multi-core platforms, and the work in [33] discusses the preemption overheads for the early-deadline-first

- *C.-W. Chang is with the Department of Computer Science and Information Engineering, School of Electrical and Computer Engineering, College of Engineering, Chang Gung University, No.259, Wenhua 1st Rd., Guishan, Taoyuan 33302, Taiwan, and with the Research Center for Information Technology Innovation, Academia Sinica, Taiwan. E-mail: chewei@mail.cgu.edu.tw.*
- *J.-J. Chen is with the Department of Informatics, Karlsruhe Institute of Technology, Germany. E-mail: j.chen@kit.edu.*
- *T.-W. Kuo is with the Department of Computer Science and Information Engineering, National Taiwan University, No. 1, Roosevelt Rd., Sec. 4, Taipei 106, Taiwan, with the Research Center for Information Technology Innovation, Academia Sinica, Taiwan, with the Graduate Institute of Networking and Multimedia, National Taiwan University, Taiwan, and with the College of Information and Communication Engineering, Sungkyunkwan University, Korea. E-mail: ktw@csie.ntu.edu.tw.*
- *H. Falk is with the Institute of Embedded Systems/Real-Time Systems, Ulm University, Germany. E-mail: Heiko.Falk@uni-ulm.de.*

Manuscript received 5 Feb. 2013; revised 6 Dec. 2013; accepted 16 Dec. 2013. Date of publication 15 Jan. 2014; date of current version 9 Jan. 2015.
Recommended for acceptance by X.-H. Sun.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.2297308

(EDF) scheduling scheme on a single processor system. To achieve better predictability of the execution time of each real-time task, scratchpad memory (SPM) [25] is usually included in the single processor real-time embedded system with deterministic memory allocation. The work in [40] further produces the WCETs of tasks with the consideration of dynamic partitioning of SPM in a preemptive multitasking system with one processor.

On the Intel many integrated core (MIC) architecture, the work in [39] customizes the data layout of the studied application bounding volume hierarchies (BVHs) to optimize the throughput of the application. Based on general purpose operating system environments, such as the Linux kernel with the X Window system, the work in [26] proposes two protocols which enable application tasks to share the graphics processing units (GPUs) in the X Window System, and the protocols can favor high-priority tasks to acquire more GPU time. The system implementation in [27] further allows GPU contexts to communicate with each other by implementing a runtime GPU memory management in the operating system instead of the user space [27]. However, most of the above implementations concentrate on the average performance of systems and produce the completion time of each task by experiments. None of the above implementations provides analysis to test whether a hard real-time task can meet its deadline in the worst case.

Regarding multiprocessor systems with local memory modules, Baruah [5] assumes that each processor consists of exactly one private memory module with a limited capacity, and each real-time task would occupy a given size of the private memory while it is scheduled on a processor. The work proposes a partitioned scheduling algorithm with the considerations of both the timing and private memory space requirements of tasks, and an analysis scheme is also provided to check whether all the tasks can meet their deadlines. The work in [35] considers different memory footprints for tasks on different processors and provides a partitioned scheduling algorithm to pack tasks onto heterogeneous processors without violating the timing and memory space constraints, where all processors share only one common memory pool. The latest result in [9] considers platforms with heterogeneous memory, but all memory devices are shared among all cores. Even though the joint consideration of memory management and real-time scheduling is an important and critical topic for advanced multi-core systems, none of the above results has considered the impact of allocating local and global memory modules on the worst-case execution time of real-time tasks.

This paper studies the partitioned scheduling on island-based multi-core platforms, where an island-based multi-core platform consists of a global memory pool and multiple islands. All cores are grouped by islands, and the cores in the same island share a fast local memory module. Examples for such platforms are blade servers [31], computers with hierarchical memory architecture [21], and embedded systems with shared scratchpad memory designs [22], [36], where scratchpad memory is small SRAM with much shorter access latency compared to DRAM. Different from cache, which is another popular application of SRAM, scratchpad memory can be mapped into the address space of processors at predefined address ranges so that software can explicitly access scratchpad memory with the support of compilers or operating systems. However, the WCET of a task depends on how it uses the scratchpad memory, in which the scratchpad memory has to be shared by multiple tasks. Thus, one of the major contributions of this work to derive a memory allocation scheme and memory-allocation-aware task scheduling algorithms, such that the memory allocation scheme can co-work with our task scheduling algorithms to satisfy the timing requirements of all real-time tasks with minimized hardware resource costs.

For island-based multi-core platforms, this paper explores the joint considerations of task scheduling over homogeneous multiple cores and memory allocation with heterogeneous memory modules. For an *implicit-deadline* sporadic task set, our objective is to provide a solution with the minimum number of allocated islands without violating the timing and memory-space constraints, where a sporadic task is said to have an *implicit deadline* if its relative deadline is equal to its minimum inter-arrival time. When the scheduler has the flexibility to decide the number of blocks in the fast local memory pool for each task, our algorithms provide an asymptotic 4-approximation bound (a formal definition of an asymptotic approximation bound will be given at the end of Section 2.2). When the number of blocks in the fast local memory pool for each task is fixed in advance, we propose a two-phase algorithm: The first phase partitions tasks into task groups under the capacity constraint of the fast local memory by adopting any polynomial-time algorithm for the traditional bin packing problem.[1] The second phase greedily allocates islands based on the task groups. Specifically, if the first-fit-decreasing[2] procedure for the bin packing problem is adopted, the algorithm provides an asymptotic $\frac{29}{9}$-approximation bound. To evaluate the performance of the proposed algorithms, 82 real-life benchmarks are analyzed by using the worst-case execution time analyzer aiT [1], and the profiling results are included for two case studies of the proposed algorithms. Extensive evaluations are also conducted to illustrate the performance of the proposed algorithms with different numbers of cores in an island and with different sizes of the fast local memory.

The rest of this paper is organized as follows: Section 2 presents the task and platform models and provides the problem definitions. Section 3 presents our algorithms to allocate memory and to schedule tasks. If the memory allocation is fixed and given as a part of the input instance, Section 4 provides an algorithm to minimize the number of required islands. Section 5 evaluates the capability of the proposed algorithms with benchmark suites, and Section 6 concludes this work.

1. Given a bin size and a list of sizes of the items to pack, the bin packing problem is to find an integer number of bins and a partition of the items such that all items are packed into the bins [19] under the bin-size constraint.

2. Given an instance of the bin packing problem, the first-fit-decreasing approach would sort all items in a non-increasing order by the sizes and sequentially pack each item into the first bin which can accommodate the item [17].

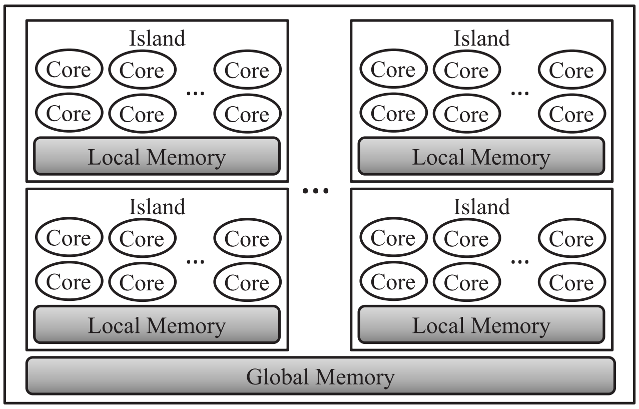

Fig. 1. The island-based multi-core system architecture.

2 System Model and Problem Definition

This section presents the system model. The problem definition is then provided, while its computational complexity is also analyzed.

2.1 System Model

This paper considers a platform consisting of multiple islands, where an island L_i consists of M homogeneous multiple cores $\mathbf{P}_i = \{C_{1,i}, C_{2,i}, \ldots, C_{M,i}\}$. For each island, there is a fast local memory pool with B blocks, which is shared among the cores in the island, where a block is the unit for memory allocation. For a platform, there is a large global memory pool, which is shared among all cores of all islands in the platform. The schematic diagram of the considered platform is shown in Fig. 1.

A set $\mathbf{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$ of N independent implicit-deadline sporadic tasks is scheduled onto cores to meet the timing constraints with the considerations of different memory access latencies. Each implicit-deadline sporadic task τ_i is characterized by its minimum inter-arrival time T_i, where the relative deadline is equal to T_i according to the definition of an implicit deadline.

In this work, we consider the different access latencies of heterogeneous memory modules. Thus, the execution of each task could be sped up if some proper blocks of the task are mapped to the fast local memory. When the blocks in the fast local memory are allocated to host a task scheduled on a core in the corresponding island, such a configuration would be fixed so that we can derive the worst-case execution time of the task. For a task τ_i, the set of accessed memory blocks is denoted as $\mathbf{B}_i$. For any set $\mathbf{X}$, let $|\mathbf{X}|$ be the cardinality of $\mathbf{X}$ for the simplicity of presentation. In order to derive the WCETs W_i^j of each task $\tau_i \in \mathbf{T}$ when $j = 0, 1, 2, \ldots, |\mathbf{B}_i|$ available blocks in the fast local memory are assigned to τ_i, a state-of-the-art static WCET analyzer aiT [1] is adopted to perform the static analysis for each task. An example is the WCET-aware C compiler presented in [18]. The basic idea behind the WCET-aware C compiler is to capture the current worst-case execution path and the possible switches of the analyzed task. For a given number j of the available fast local memory blocks, W_i^j (and the corresponding memory allocation) of a task τ_i is selected from several candidates with using no more than j blocks of the fast local memory. For the rest of this paper, we assume that W_i^j for $j = 0, 1, \ldots, |\mathbf{B}_i|$, $i = 1, 2, \ldots, |\mathbf{T}|$ are given, and the analysis and optimization are both based on the given WCETs. For the simplicity of notations, we implicitly denote W_i^j as $W_i^{|\mathbf{B}_i|}$ if $j > |\mathbf{B}_i|$ and $j \leq B$ because the local memory blocks used by task τ_i is no more than the total memory blocks of τ_i. Moreover, W_i^j is set to ∞ if $j > B$ or $j < 0$ to eliminate infeasible memory allocations.

2.2 Problem Definition

For an island-based multi-core platform, to meet the timing constraints, we have to (1) decide the number of available fast local memory blocks for each task and (2) assign each task onto a core of an island for the execution without violating real-time constraints. For each island, tasks are assigned to the island under the following constraints: (1) The number of the required fast local memory blocks of tasks in the island does not exceed the size B of the fast local memory, and (2) all tasks could be scheduled onto the cores without violating the real-time requirements (arrival times and deadlines). On each core, tasks can be scheduled by applying intra-core real-time scheduling algorithms, such as the earliest-deadline-first (EDF) and rate monotonic (RM) strategies [32], and the utilization of a task is defined as its WCET divided by its minimum inter-arrival time. As a result, if the number of available blocks j in the fast local memory of task τ_i is decided, the utilization u_i of task τ_i is $U_i^j = \frac{W_i^j}{T_i}$, where the meaning of U_i^j is the worst-case utilization of task τ_i by using j blocks of the fast local memory. The EDF policy assigns the task instance with an earlier absolute deadline a higher scheduling priority, and the RM policy uses the reciprocal of the task period (minimum inter-arrival time) as the static priority of the task. Liu and Layland have proven that the *achievable utilization factor* [32] of EDF and RM are 100 and 69.3 percent, respectively, where the achievable utilization factor provides an efficient strategy for verifying the real-time guarantee of a given scheduling scheme. With any intra-core scheduling policy, all tasks in a core can meet real-time requirements if the total utilization of the real-time tasks is no more than the corresponding achievable utilization factor. In the rest of this paper, if not specified, we assume EDF is adopted for the intra-core scheduling.

When the required number of blocks in the fast local memory is *flexible* for each task, the first target problem is defined as follows:

Definition 1. (The Island-Based Real-Time Scheduling (IBRT) Problem). *Consider an island-based multi-core platform with a large global memory pool, and each island consists of M homogeneous cores and B blocks of the fast local memory. An implicit-deadline sporadic task set $\mathbf{T}$ is given along with the WCETs W_i^j for mapping $0 \leq j \leq |\mathbf{B}_i|$ blocks of τ_i in $\mathbf{T}$ to the fast local memory. The problem is to derive the number of the available fast local memory blocks for each task and to assign the task onto a core of an island such that the number of allocated islands is minimized, the total number of the required* *fast local memory blocks of tasks on each island does not exceed the fast local memory size, and all tasks meet their real-time constraints.*

For some application scenarios, the number of required fast local memory blocks might be fixed for a task due to some application-specific constraints. When the number of the blocks in the fast local memory is *fixed* for each task, the second target problem is defined as follows:

Definition 2. (The Island-Based Real-Time Scheduling with Fixed Memory Allocation (IBRTA) Problem) *In addition to the input M and B in the IBRT problem, each task τ_i in $\mathbf{T}$ has a fixed required amount m_i of the fast local memory blocks and its WCET $W_i^{m_i}$. The problem is to assign each task onto a core of an island such that the number of allocated islands is minimized, the number of the required fast local memory blocks of the tasks on each island does not exceed the fast local memory size, and all tasks meet their real-time constraints.*

For each task, to derive a feasible memory allocation and a task scheduling result, we have to make sure the required processor utilization of the task is no more than 100 percent (where the adopted intra-core scheduling algorithm is EDF), otherwise there is no feasible solution. Therefore, any W_i^j with $\frac{W_i^j}{T_i} > 100$ percent is reset to ∞, as such a setting is not eligible for feasible solutions. For the rest of this paper, we assume that the capacity of the global memory pool is larger than or equal to the required memory space for the tasks, i.e., $\sum_{\tau_i \in \mathbf{T}} |\mathbf{B}_i|$. The following theorem shows the computational complexity of the problems.

Theorem 1. *The IBRT and IBRTA problems are $\mathcal{NP}$-hard in the strong sense.*

Proof. For a special case in which the memory allocation has no influence on the worst-case execution time, the fast local memory is sufficiently large, i.e., $B \geq \sum_{\tau_i \in \mathbf{T}} |\mathbf{B}_i|$, and there is only one core in each island, both problems are equivalent to the bin packing problem, which is $\mathcal{NP}$-hard in the strong sense [19]. □

As the studied problems are $\mathcal{NP}$-hard, instead of deriving optimal solutions, we will look for polynomial-time approximation algorithms for these problems. For the studied problems, an algorithm is said to be with an α-approximation bound if it guarantees to derive a solution that uses at most $\alpha \cdot OPT$ islands when the optimal solution requires OPT islands. Similarly, an algorithm is said to be with an asymptotic α-approximation bound if it guarantees to derive a solution that uses at most $\alpha \cdot OPT + \beta$ islands when the optimal solution requires OPT islands, where β is a positive constant.

3 Our Algorithms for the IBRT Problem

This section presents our solutions for the IBRT problem with joint considerations of task scheduling and memory allocation. The lower bound of any instance of the IBRT problem is first analyzed in Section 3.1, which also shows the rationale behind our memory allocation strategy. Two algorithms are then presented for the special and general cases of the IBRT problem, respectively. Their approximation bounds are also discussed.

3.1 Lower Bound for An Input Instance

In this section, we first establish the lower bound of the optimal solution for any instance of the IBRT problem so that we can use this lower bound as a base to derive the approximation bound of our algorithms later. There are two resource constraints in the IBRT problem: (1) the limited processor utilization of a core and (2) the limited fast local memory space of an island. In order to balance the usage of the processor utilization and the fast local memory space (which have different units and scales), the required quantities of the two resources of each task are *normalized* to better understand the resource usage. That is, for a task τ_i with m_i blocks mapped to the fast local memory, we define that the *normalized required processor utilization* is $\frac{U_i^{m_i}}{M}$, where $U_i^{m_i}$ and M are the utilization of τ_i and the number of cores of an island, and the *normalized required memory space* is $\frac{m_i}{B}$ which is the required fast local memory space m_i divided by the size B of the fast local memory in an island. For each task τ_i, we define the *normalized resource usage* as the sum of the normalized required processor utilization and the normalized required memory space, and λ_i is denoted as the minimum normalized resource usage of τ_i. That is $\lambda_i = \min_{m_i=0}^{|\mathbf{B}_i|}\{\frac{U_i^{m_i}}{M} + \frac{m_i}{B}\}$.

The following lemma shows that $\Lambda = \frac{1}{2}\sum_{\tau_i \in \mathbf{T}} \lambda_i$ is the lower bound of any optimal solution for the IBRT problem, and that shows the rationale behind the definition of the minimum normalized resource usage λ_i of each task τ_i. The lower bound also provides the base for the approximation analysis.

Lemma 1. *The lower bound of any input instance for the IBRT problem is $\Lambda = \frac{1}{2}\sum_{\tau_i \in \mathbf{T}} \lambda_i$.*

Proof. For an input instance, we assume that the optimal solution uses OPT islands. As the solution is feasible, each of the OPT islands provides at most 100 percent normalized processor utilization and at most 100 percent normalized memory space. Therefore, the total normalized resource usage of the tasks of the optimal solution is no more than $2 \cdot OPT$. As $\sum_{\tau_i \in \mathbf{T}} \lambda_i$ is the minimum normalized resource usage of all tasks, we know that $2 \cdot OPT \geq \sum_{\tau_i \in \mathbf{T}} \lambda_i$. As a result, we prove that $\Lambda = \frac{1}{2}\sum_{\tau_i \in \mathbf{T}} \lambda_i \leq OPT$. □

3.2 An Algorithm for Single-Core Islands

We first consider a special case of the IBRT problem, in which the fast local memory is private for exactly one core, i.e., $M = 1$. An example is the programmable private SRAMs for the Synergistic Processing Elements (SPEs) of IBM Cell architecture [24]. The general case of the IBRT problem (i.e., $M \geq 1$) is going to be studied in Section 3.3.

Algorithm 1 illustrates the pseudo code of our single-core-island (Sci) algorithm. In Algorithm 1, $\bar{\mathbf{L}}$ is the list of the allocated islands and initialized as $\emptyset$ (an empty list), where a list is a set with ordering. We denote p_k as the processor utilization of the core in island L_k and denote s_k as the number of the used fast local memory blocks in island L_k. The Sci algorithm first determines the utilization u_i and the number m_i of the required fast local memory blocks for each task τ_i, as shown in Line 3 of Algorithm 1, which minimizes the normalized resource usage of task τ_i. This initialization is motivated by Lemma 1 and would provide a good property for proving the approximation bound. Then, for any task ordering of $\mathbf{T}$, a first-fit approach is applied to assign τ_i to an island based on the configuration of u_i and m_i with the minimum normalized resource usage of τ_i, as shown from Lines 4 to 7. If there is no island which can accommodate the considered task τ_i, a new island is allocated and then queued at the end of the allocated-island list $\vec{\mathbf{L}}$ from Lines 8 to 11.

Algorithm 1: Single Core Island

```
Input: an IBRT problem instance with a task set $\mathbf{T}$, and there
       is only one core in each island;
1  the allocated-island list $\vec{\mathbf{L}} \leftarrow \emptyset$;
2  forall the $\tau_i \in \mathbf{T}$ do
3      $u_i \leftarrow U_i^j$ and $m_i \leftarrow j$, such that $(U_i^j + \frac{j}{B})$ is minimized;
4      forall the $L_k \in \vec{\mathbf{L}}$ sequentially do
5          if $u_i + p_k \leq 100\%$ and $m_i + s_k \leq B$ then
6              assign $\tau_i$ on the core of island $L_k$;
7              $p_k \leftarrow u_i + p_k$; $s_k \leftarrow m_i + s_k$; break;
8      if $\tau_i$ has not been assigned then
9          assign $\tau_i$ on the core of a new island $L_\ell$;
10         $p_\ell \leftarrow u_i$; $s_\ell \leftarrow m_i$;
11         queue $L_\ell$ to the end of list $\vec{\mathbf{L}}$;
12 return the memory allocation and task partition results;
```

The time complexity of Line 3 is bounded by the number $|\mathbf{B}_i|$ of the candidates of the memory allocation of task τ_i, and let $\Upsilon = \sum_{\tau_i \in \mathbf{T}} |\mathbf{B}_i|$. The time complexity of the **forall** loop from Line 4 to Line 7 is bounded by the number of the allocated islands, which is limited by the number $|\mathbf{T}|$ of tasks. As a result, the time complexity of the Sci algorithm is $O(|\mathbf{T}|^2 + \Upsilon)$. The following theorem further shows the approximation bound of the presented algorithm.

Theorem 2. *The Sci algorithm is a polynomial-time 4-approximation algorithm for the IBRT problem when $M = 1$.*

Proof. For an instance of the IBRT problem, let the optimal solution and the Sci algorithm use OPT and $|\vec{\mathbf{L}}|$ islands, respectively. By Line 3 of Algorithm 1, we know that total normalized resource usage of the tasks is minimized as $\sum_{\tau_i \in \mathbf{T}} \lambda_i$. For the $|\vec{\mathbf{L}}|$ islands derived from the Sci algorithm, if we merge the tasks in any two of the $|\vec{\mathbf{L}}|$ islands into one island, either the processor utilization constraint or the fast local memory space constraint would be violated because a first-fit approach is adopted to pack tasks into islands. This can be proved by contradiction. Therefore, the total normalized resource usage of the tasks in any two of the $|\vec{\mathbf{L}}|$ islands is more than 100 percent. Because each island can provide at most 100 percent of processor utilization and 100 percent of the normalized fast local memory space, by pigeonhole principle, we have that

$$\frac{|\vec{\mathbf{L}}|}{2} < \sum_{\tau_i \in \mathbf{T}} \lambda_i. \tag{1}$$

By Lemma 1 and Equation (1), we know that

$$|\vec{\mathbf{L}}| < 2 \cdot \sum_{\tau_i \in \mathbf{T}} \lambda_i \leq 4 \cdot OPT.$$

The time complexity $O(|\mathbf{T}|^2 + \Upsilon)$ is polynomial in the input size. Therefore, we reach the conclusion. □

The analysis in the proof of Theorem 2 provides an upper bound for the approximation bound. We, unfortunately, do not have a tight example. The lower bound of the approximation bound of Algorithm 1 can be at least 3 with the following example: There are $3N$ tasks (N is a positive integer), and each task τ_i consists of two feasible memory configurations that are $(\tilde{u}_i = \frac{5}{9}, \tilde{m}_i = 0)$ and $(\tilde{u}_i = \frac{1}{3}, \tilde{m}_i = \frac{1}{3})$, where $\tilde{u}_i$ and $\tilde{m}_i$ are the normalized required processor utilization and memory space. The optimal solution would choose the second memory configuration and pack the tasks into N islands. However, the Sci algorithm would choose the first memory configuration with the minimum normalized resource usage and require $3N$ islands to accommodate all tasks. Thus, the tight approximation bound of the Sci algorithm for the single-core IBRT problem is between 3 and 4.

3.3 An Algorithm for Multi-Core Islands

This section presents the multi-core-island (Mci) algorithm, illustrated in Algorithm 2, which has an asymptotic 4-approximation bound for the IBRT problem. Similar to the Sci algorithm, for each task τ_i, the Mci algorithm determines the worst-case utilization u_i and the number m_i of required fast local memory blocks to minimize the normalized resource usage of the task. For the general case when $M > 1$, instead of any arbitrary ordering adopted in the Sci algorithm, all tasks are then sorted in a non-increasing order by the required fast local memory space m_i. In Algorithm 2, s_k is denoted as the used fast local memory space of island L_k, and $C_{r,k}$ and $p_{r,k}$ are denoted as the rth core of L_k and the current utilization of $C_{r,k}$, respectively. For assigning a task, all the allocated islands are tested by a first-fit approach as shown in Lines 6 to 14 in Algorithm 2. For testing the feasibility of assigning a task τ_i on an island, the Mci algorithm has to make sure that the remaining space of the fast local memory is large enough to accommodate the required blocks m_i in Line 7, and all cores in the island are tested in a first-fit scheme to check that if there is a core to feasibly serve the required processor utilization u_i (From Lines 8 to 12). If there is no allocated island to feasibly accommodate task τ_i under τ_i's minimum normalized resource usage, a new island is then allocated to accommodate τ_i.

The time complexity of the memory setting in Lines 1 and 2 is bounded by $O(\Upsilon)$, where Υ is set as $\sum_{\tau_i \in \mathbf{T}} |\mathbf{B}_i|$, and the sorting in Line 3 is bounded by $O(|\mathbf{T}|\log|\mathbf{T}|)$. The **for** loop from Line 5 to Line 18 has $O(|\mathbf{T}|)$ iterations, while the complexity for testing each task is bounded by the number of tasks $|\mathbf{T}|$. As a result, the time complexity of the Mci algorithm is $O(|\mathbf{T}|^2 + \Upsilon)$.

In the following analysis, let $\vec{\mathbf{L}} = \{L_1, L_2, \ldots, L_{|\vec{\mathbf{L}}|}\}$ be the resulting list of the allocated islands by the Mci algorithm. Similar to the normalized resource usage of tasks, for an island L_k, the *normalized resource utilization* R_k is defined as $\frac{\sum_{r=1}^{M} p_{r,k}}{M}+\frac{s_k}{B}$. Lemma 2 shows the relation among the allocated islands in terms of the summation of the normalized resource utilization. It also provides the base for proving the asymptotic approximation bound of the MCI algorithm in Theorem 3.

Algorithm 2: Multi-Core Island

```
Input: an IBRT problem instance with a task set $\mathbf{T}$ and with
       $M$ cores in each island;
1  forall the $\tau_i \in \mathbf{T}$ do
2      $u_i \leftarrow U_i^j$ and $m_i \leftarrow j$, such that $(\frac{U_i^j}{M}+\frac{j}{B})$ is minimized;
3  sort all tasks in a non-increasing order by $m_i$;
4  the allocated-island list $\vec{\mathbf{L}} \leftarrow \emptyset$;
5  for $i = 1$ to $|\mathbf{T}|$ do
6      forall the $L_k \in \vec{\mathbf{L}}$ sequentially do
7          if $m_i + s_k \le B$ then
8              for $r = 1$ to $M$ do
9                  if $u_i + p_{r,k} \le 100\%$ then
10                     assign $\tau_i$ on core $C_{r,k}$;
11                     $p_{r,k} \leftarrow u_i + p_{r,k}$; $s_k \leftarrow m_i + s_k$;
12                     break;
13         if $\tau_i$ has been assigned then
14             break;
15     if $\tau_i$ has not been assigned then
16         assign $\tau_i$ on the first core $C_{1,\ell}$ of a new island $L_\ell$;
17         $p_{1,\ell} \leftarrow u_i$; $s_\ell \leftarrow m_i$;
18         queue $L_\ell$ into the end of list $\vec{\mathbf{L}}$;
19 return the memory allocation and task partition results;
```

Lemma 2. *For any instance of the IBRT problem, suppose that the* MCI *algorithm derives a result of allocated-island list* $\vec{\mathbf{L}}$. *Then,* $R_{k-1}+R_k$, *as the summation of the normalized resource utilizations of the* $(k-1)^{th}$ *and* kth *islands, is greater than* 100 *percent for* $1 < k < |\vec{\mathbf{L}}|$.

Proof. Because the MCI algorithm uses a first-fit approach to assign tasks onto islands, while a task τ_i is assigned onto an island L_k, at least one of the following two cases holds: (1) $m_i + s_{k-1} > B$, which means that the remaining fast local memory space of L_{k-1} is not enough to accommodate τ_i, or (2) $m_i + s_{k-1} \le B$ but $u_i + p_{r,k-1} > 100\%,\ \forall 1 \le r \le M$, which means that there is no core in L_{k-1} having enough available processor utilization to serve τ_i. If the first case is true, we know that

$$\frac{s_{k-1}}{B}+\frac{s_k}{B} > 100\%. \tag{2}$$

For the second case, we know that there is at least one task assigned to each core in L_{k-1}, because no task could be with the required processor utilization more than 100 percent. Thus, there are at least M tasks assigned to L_{k-1} because there are M cores in an island. Since the MCI algorithm sorts all tasks in a non-increasing order by the required fast local memory space before assigning them to islands, we know that the fast local memory of L_k can accommodate at least M tasks while each core can support at least one task. As L_k is not the last allocated island ($k < |\vec{\mathbf{L}}|$), there are at least M tasks assigned to L_k. For each core $c_{r,k-1}$ in L_{k-1} and each task τ_i in L_k, we have $p_{r,k-1}+u_i > 100$ percent due to the statement in this case. Since there are at least M tasks assigned to L_k, we know that

$$\frac{\sum_{r=1}^{M} p_{r,k-1}+\sum_{r=1}^{M} p_{r,k}}{M} > 100\%. \tag{3}$$

Because at least one of Equations (2) and (3) would be true, we have that

$$\begin{aligned}&\left(\frac{\sum_{r=1}^{M} p_{r,k-1}}{M}+\frac{s_{k-1}}{B}\right)+\left(\frac{\sum_{r=1}^{M} p_{r,k}}{M}+\frac{s_k}{B}\right)\\ &\quad = R_{k-1}+R_k > 100\%,\end{aligned} \tag{4}$$

$\forall 1 < k < |L|$. Thus, this lemma is proved. □

By using the result of Lemma 2, the approximation bound of the MCI algorithm is proved in Theorem 3, which provides an insight for real-time system synthesis for island-based multi-core platforms.

Theorem 3. *The* MCI *algorithm is a polynomial-time asymptotic* 4*-approximation algorithm for the IBRT problem.*

Proof. For any instance of the IBRT problem, let $\vec{\mathbf{L}}$ be the allocated-island list derived by the MCI algorithm, and the optimal solution exactly uses OPT islands. By Lemma 2, we have

$$\sum_{k=2}^{|\vec{\mathbf{L}}|-1}(R_{k-1}+R_k) > |\vec{\mathbf{L}}|-2, \tag{5}$$

where R_k is the normalized resource utilization of each allocated island L_k by the MCI algorithm. As the MCI algorithm selects the memory configuration with the minimum normalized resource usage $\min_{m_i=0}^{|\mathbf{B}_i|}\{\frac{U_i^{m_i}}{M}+\frac{m_i}{B}\}$ which is denoted by λ_i for each task τ_i, by Lemma 1, we have

$$\frac{1}{2}\sum_{\tau_i\in\mathbf{T}}\lambda_i \le OPT. \tag{6}$$

By Equations (5) and (6), we have

$$\begin{aligned}|\vec{\mathbf{L}}| &< \sum_{k=2}^{|\vec{\mathbf{L}}|-1}(R_{k-1}+R_k)+2 < 2\sum_{k=1}^{|\vec{\mathbf{L}}|}R_k+2\\ &= 2\sum_{\tau_i\in\mathbf{T}}\lambda_i+2 \le 4\cdot OPT+2,\end{aligned} \tag{7}$$

where the reason for the "=" is that the total normalized resource usage of the tasks must be equal to the total normalized resource utilization of the islands, because all tasks are assigned on islands. The time complexity of the MCI algorithm is $O(|\mathbf{T}|^2+\Upsilon)$ which is polynomial in the input size. Therefore, we reach the conclusion. □

4 Our Scheme for the IBRTA Problem

This section presents a polynomial-time algorithm for the IBRTA problem. For an instance of the IBRTA problem, each task τ_i has the required processor utilization $u_i = \frac{W_i}{T_i}$ and the required fast local memory space m_i. For the rest of this section, we assume $u_i \leq 100\%$ and $m_i \leq B$ for each $\tau_i \in \mathbf{T}$, otherwise there is no feasible solution for the input instance.

Before stepping into our results, we first summarize a related problem, called the *two-dimensional vector packing problem* [41]: Given a set $\mathbf{V}$ of vectors $< v_1, v_2, \ldots, v_N >$ with two dimensions, where $0 \leq v_{i,j} \leq 1$ is the element in the jth dimension of vector v_i, the problem is to partition $\mathbf{V}$ into L subsets such that L is minimized and the summation of the vectors in each subset is no more than 1 in both dimensions. The two-dimensional vector packing problem is a generalization of the bin packing problem which has only one dimension in the vectors [19], and the two-dimensional vector packing problem is proved to be $\mathcal{APX}$-hard [41]. When there is only one core in an island ($M = 1$), the IBRTA problem is equivalent to the two-dimensional vector packing problem. For the two-dimensional vector packing problem, the state-of-the-art polynomial-time (randomized with high probability) asymptotic approximation bound is $(\ln 2 + 1)$ by adopting a randomized algorithm [4].

Algorithm 3: MCI with Fixed Memory Allocation

```
Input: an IBRTA problem instance with a task set $\mathbf{T}$, and there
       are $M$ cores in each island;
1  partition $\mathbf{T}$ into $g$ task groups, $\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_g$, by using a
   polynomial-time algorithm with an asymptotic
   $\alpha_{BIN}$-approximation bound for the bin packing problem by
   considering only the memory constraint that $\sum_{\tau_i \in \mathbf{G}_x} m_i \leq B$
   for each task group $\mathbf{G}_x$, for all $x = 1, 2, \ldots, g$;
2  foreach $\mathbf{G}_x$ with $x = 1, 2, \ldots, g$ do
3      a list of allocated islands $\vec{\mathbf{L}} \leftarrow \emptyset$;
4      forall the $\tau_i \in \mathbf{G}_x$ do
5          forall the islands $L_k \in \vec{\mathbf{L}}$ do
6              for $r = 1$ to $M$ do
7                  if $p_{r,k} + u_i \leq 100\%$ then
8                      assign $\tau_i$ on core $C_{r,k}$;
9                      $p_{r,k} \leftarrow p_{r,k} + u_i$; break;
10             if $\tau_i$ has been assigned then
11                 break;
12         if $\tau_i$ has not been assigned then
13             allocate and queue a new island $L_\ell$ to the end of $\vec{\mathbf{L}}$;
14             assign $\tau_i$ on $C_{1,\ell}$; $p_{1,\ell} \leftarrow u_i$;
15 return the task assignment of $\mathbf{T}$;
```

The rest of this section discusses only the case with $M > 1$ by presenting and analyzing our algorithm which is denoted by multi-core-island with fixed fast local memory allocation (MCIF). The pseudo code of the MCIF algorithm is presented in Algorithm 3. This algorithm has two phases: (1) In the first phase, it considers only the limited fast local memory space constraint to pack tasks into groups, and (2) in the second phase, it considers only the processor utilization constraint to assign tasks onto cores and to allocate new island if it is necessary.

The first phase could be achieved by adopting any existing polynomial-time bin packing algorithm by considering only the constraint of the limited fast local memory space. For example, it has been shown in [17] that the first-fit-decreasing procedure (in polynomial time) uses at most $\frac{11}{9} \cdot OPT_{BIN} + \frac{6}{9}$ bins, in which OPT_{BIN} is the optimal solution for the instance of the bin packing problem. It has also been shown in [16] that the bin packing problem can admit an asymptotic polynomial-time approximation scheme that uses at most $(1 + \epsilon) \cdot OPT_{BIN} + \beta$ bins, where β is a positive constant, and $\epsilon > 0$ is a user-specified parameter. We do not restrict ourselves for any polynomial-time algorithm of the bin packing problem. Suppose that the first phase partitions the tasks into g task groups, i.e., with g sets $\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_g$ of tasks, in which $\sum_{\tau_i \in \mathbf{G}_x} m_i \leq B$ for any $x = 1, 2, \ldots, g$.

In the second phase (from Lines 2 to 14), for each task group $\mathbf{G}_x$, each task τ_i in $\mathbf{G}_x$ is assigned to a core by a first-fit approach from Lines 5 to 11, where $p_{r,k}$ denotes the current utilization of the rth core of island L_k. If no core can accommodate the tasks, a new island would be allocated for the task group as shown in Lines 12 to 14. Note that the fast local memory space constraint is never violated in the second phase because the required total fast local memory space of the tasks in a group is no more than B. Now, we show the property of the MCIF algorithm in Theorem 4.

Theorem 4. *The MCIF algorithm is a polynomial-time asymptotic $(\alpha_{BIN} + 2)$-approximation algorithm for the IBRTA problem, where the asymptotic approximation bound of the polynomial-time approximation algorithm adopted in the first phase is α_{BIN}.*

Proof. The time complexity of the second phase in the MCIF algorithm is bounded by $O(|\mathbf{T}|^2)$. Provided that the first phase is also with polynomial-time complexity, we know that the MCIF algorithm has polynomial-time complexity.

For the rest of this proof, we focus on the optimality. Suppose that the optimal solution allocates OPT islands to serve all tasks. The first phase groups tasks by considering only their *fixed* numbers of required blocks in the fast local memory. Thus, it is clear that based on any feasible solution for the IBRTA problem, grouping all tasks in each island as a task group is also a feasible way to group tasks by only considering the fast local memory constraint which is $\sum_{\tau_i \in \mathbf{G}_x} m_i \leq B$ for any task group $\mathbf{G}_x$. Let the minimum number of task groups be OPT_{BIN} (the optimal solution of the bin packing problem instance in the first phase), and the polynomial-time asymptotic approximation algorithm (for the bin packing problem) adopted in the first phase produces the number of task groups g, such that $g \leq \alpha_{BIN} \cdot OPT_{BIN} + \beta_{BIN}$, where α_{BIN} and β_{BIN} are defined from the adopted asymptotic approximation algorithm. Therefore, we know that

$$g \leq \alpha_{BIN} \cdot OPT_{BIN} + \beta_{BIN} \leq \alpha_{BIN} \cdot OPT + \beta_{BIN}, \quad (8)$$

With the derived bound of the number of task groups in Equation (8), now we move to the second phase to illustrate the bound of the number of allocated islands. Let the first phase partition task into g task groups. Suppose that there are x task groups, in which each of these groups uses only one island in the second phase of the MCIF algorithm. Therefore, each of the remaining y task groups uses more than one island in the second phase of the MCIF algorithm, where $y = g - x$. For these y task groups, we further suppose these tasks require z islands in the second phase of the MCIF algorithm. For each of these y task groups, it is not difficult to see that the processor utilization for any two used cores in the second phase is more than 100 percent, otherwise the tasks on the two cores should be packed into one core, because the MCIF algorithm adopts a first-fit strategy to assign tasks onto cores from Lines 5 to 11. Based on the first-fit strategy, we also know that all cores of the allocated islands would be used, except the cores in the last allocated island for each of the y task groups. Therefore, more than $(z - y)M$ cores are used for assigning the tasks in the y task groups, and the total processor utilization for these used cores is more than $\frac{(z-y)M}{2}$. This leads to the lower bound of the optimal solution, in which $OPT > (z - y)/2$ because the average utilization among the cores in these $(z - y)$ islands in more than 50 percent. As a result, we know that the number of the allocated islands of the MCIF algorithm is $x + z$, in which

$$\begin{aligned} x + z &= x + y + (z - y) \leq \alpha_{BIN} OPT + \beta_{BIN} + 2 \cdot OPT \\ &= (\alpha_{BIN} + 2) OPT + \beta_{BIN}, \end{aligned} \tag{9}$$

where the inequality comes from the definition $x + y = g$, Equation (8), and $OPT > (z - y)/2$. Thus, this theorem is proved. □

Corollary 1. *When the first phase in the MCIF algorithm uses the* First-Fit-Decreasing *procedure, the MCIF algorithm has an asymptotic $\frac{29}{9}$-approximation bound for the IBRTA problem, with time complexity $O(|\mathbf{T}|^2)$.*

Proof. First-Fit-Decreasing procedure is proved to use at most $(\frac{11}{9} \cdot OPT_{BIN} + \frac{6}{9})$ bins in the bin packing problem [17], where OPT_{BIN} is the optimal solution of the bin packing problem. The algorithm and analysis is shown asymptotically tight in [17]. The time complexity for the first-fit-decreasing procedure is $O(|\mathbf{T}| \log |\mathbf{T}|)$. Therefore, the time complexity is dominated by the second phase, i.e., $O(|\mathbf{T}|^2)$, and the asymptotic approximation bound is $\frac{29}{9}$ based on Theorem 4. □

This paper explores how to assign a set of independent implicit-deadline sporadic tasks onto an island-based multi-core platform to meet the timing constraints with the considerations of different memory access latencies. It is notable that this work considers static memory allocation to support the execution of tasks, i.e., the memory allocation will be derived and fixed by our memory allocation algorithm before the task execution. Thus, the switching overheads of local memory are not considered in this paper. Thus, the WCET of each real-time task can be derived when the allocation of the local memory is given. In this work, we have to count the worst-case latency caused by memory bus contention to the WCET of each task to safely meet the timing requirements of all real-time tasks. To further provide tighter WCETs for our task scheduling algorithms, our work can be extended by considering timing predictable memory bus arbitrators, such as the time division multiple access (TDMA) policy [29], [37] for the bus sharing of multiple tasks and the bandwidth reservation scheme [42] of memory bus. For the implementation of memory bus arbitrators, the MERASA project [38] develops a package of system software and proposes a program model to limit the extra latency for memory bus access, which can be applied to ensure the timing behavior for communications.

In this paper, all of the proposed polynomial-time algorithms are with approximation bounds which are proven by assuming that the adopted intra-core scheduling is EDF. If RM is adopted in our algorithms for the intra-core scheduling, we have to make sure that the utilization of each core is no more than $n_{r,k}(2^{\frac{1}{n_{r,k}}} - 1)$ to safely meet all timing requirements of real-time tasks [32], where $n_{r,k}$ is the number of the tasks on the rth core of the kth island. For example, Step 7 of Algorithm 3 (the MCIF algorithm) "**if** $p_{r,k} + u_i \leq 100\%$ **then**" will be changed into "**if** $p_{r,k} + u_i \leq n_{r,k}(2^{\frac{1}{n_{r,k}}} - 1)$ **then**" if RM is used to replace EDF for the intra-core scheduling. Moreover, we know that $\lim_{n_{r,k}\to\infty} n_{r,k}(2^{\frac{1}{n_{r,k}}} - 1) = ln2 < 69.3\%$. Thus, the proven approximation bound $\alpha_{BIN} + 2$ in Theorem 4 will become $\frac{\alpha_{BIN}+2}{69.3\%}$.

5 PERFORMANCE EVALUATION

This section presents the setting of our experiments and the experimental results to evaluate the performance of the proposed algorithms.

5.1 Environment Setup

In order to schedule all real-time tasks without violating their real-time timing requirements, we have to derive the WCETs of all tasks and verify whether all tasks could meet their relative deadlines even with the WCETs. To derive the WCETs of each task by using different numbers of the fast local memory blocks, a WCET-aware C compiler [18] is used to derive the values W_i^j for all possible numbers j of the used fast local memory blocks of each task τ_i, where the WCET-aware C compiler is based on the Infineon TriCore TC1797 architecture with optimization level *-O2*. For the Infineon TriCore TC1797 platform, 1 Mega-Byte half-word addressable Flash is installed as the global memory module, and 40 Kilo-Bytes on-chip scratchpad memory is included as the fast local memory, where scratchpad memory is small SRAM that are mapped onto the address space of cores at predefined address ranges so that software can explicitly access scratchpad memory. The latency cycles for accessing the fast local memory (scratchpad memory) and the global memory (half-word addressable Flash memory) are 1 cycle and 7 cycles, respectively. In the WCET-aware C compiler, the size of a block is set to 128 Bytes.

The rationale behind the WCET-aware C compiler is to capture the current worst-case execution path (which is

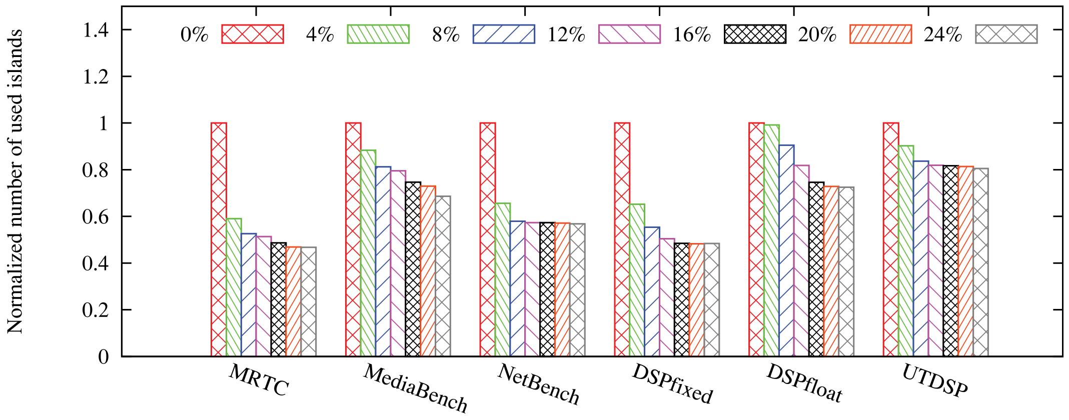

Fig. 2. The performance evaluation of the MCI algorithm with benchmarks for small local memory.

the path with the current WCET) of a task during the compiling and check possible switches to derive the WCET with the limited size of the fast local memory. Therefore, by iteratively applying all possible sizes of the fast local memory to the WCET-aware C compiler, all the WCETs W_i^j of τ_i with $0 \leq j \leq |\mathbf{B}_i|$ can be derived. In the following evaluations, 82 different real-life benchmarks from MRTC [2], MediaBench [30], UTDSP [3], NetBench [34], and DSPstone [43] are included, where NetBench is with encoders, and DSPstone is partitioned into fixed-point and floating-point parts.

5.2 Case Studies with Benchmarks

In the first experiment, benchmarks are grouped into six benchmark groups, as shown in Fig. 2, and there are four cores in each island. For each benchmark group, 40 tasks are created for each benchmark with the WCETs obtained from the WCET-aware C compiler. The minimum inter-arrival time of each task is then properly set such that the worst-case utilization forms a uniform distribution between 1 and 100 percent (by normalizing to the number of cores in an island, that are 0.25 and 25 percent) when using no scratchpad memory. For each benchmark group, the size of the fast local memory (the scratchpad memory) in an island is configured as 0, 4, 8, 12, 16, 20, and 24 percent of the total size of the program codes of all tasks. With the above setting, the MCI algorithm is then applied to derive the results with 1,000 tests for each configuration, and the average of the required numbers of islands is reported.

For each benchmark group, the evaluation results are normalized to the number of required islands when using no scratchpad memory (0 percent). The results in Fig. 2 show that the number of the required islands can be significantly reduced for most of the benchmark groups by using the fast local memory with the size only 8 percent of the total size of the program codes. It is because that the MCI algorithm would allocate proper fast local memory blocks for each task to provide the minimum lower bound of the number of required islands. However, some applications, such as the benchmarks in UTDSP and DSPfloat, do not have their performance bottlenecks on memory access, i.e., increasing the number of the fast local memory blocks does not significantly reduce the WCETs of the applications. Thus, the reduced numbers of required islands of those two benchmark groups are less than the results of the other benchmark groups. To further evaluate the MCI algorithm with larger local memory space, we further conduct a simulation set, as shown in Fig. 3. In this experiment, the size of the fast local memory (the scratchpad memory) in an island is configured as 0, 20, 40, 60, 80, and 100 percent of the total size of the program codes of all tasks. According to the results in Fig. 3, we can see that the numbers of required islands are saturated when the size is larger than 60 percent. The reason is that all of the

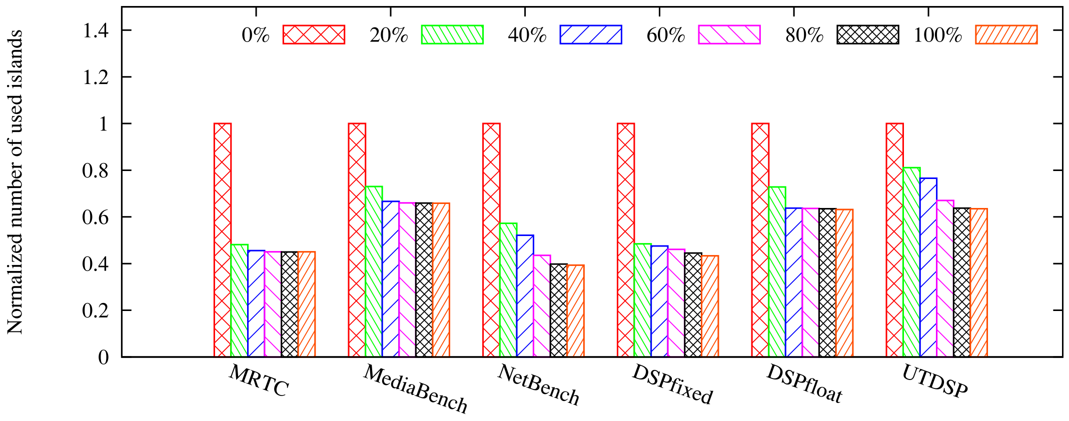

Fig. 3. The performance evaluation of the MCI algorithm with benchmarks for large local memory.

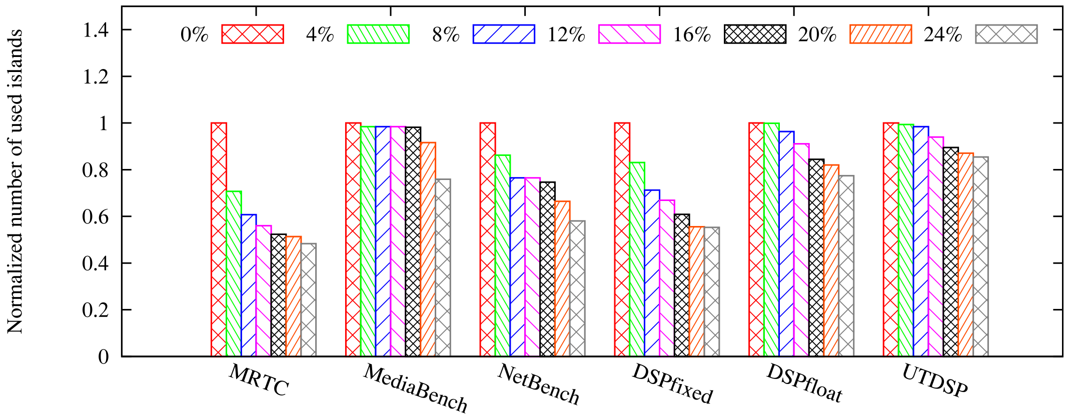

Fig. 4. The performance evaluation of the MCIF algorithm with benchmarks.

frequently accessed parts of each benchmark are already in the local memory, and the performance bottleneck is moved to the computing power of cores when the size of local memory is larger than 60 percent.

In the second experiment, as shown in Fig. 4, we evaluate the performance of the MCIF algorithm. When the memory requirement of the fast local memory space is fixed for each task, the MCIF algorithm could schedule tasks onto cores with joint considerations of the processor utilization and the limited local memory size. For each benchmark group, 40 tasks are created for each benchmark in the group with the obtained WCETs. The setting of the minimum inter-arrival time of each task is the same with the setting of the previous experiment. For each benchmark group, the size of the fast local memory in an island is configured as 0, 4, 8, 12, 16, 20, and 24 percent of the total size of the program codes of all tasks. For the memory allocation, we use the same amount of the fast local memory as that used by the MCI algorithm in the previous experiment for each experimental set, and the fast local memory space is randomly distributed among the tasks. That is, the number of the required fast local memory blocks m_i would be fixed for each task τ_i before we apply the MCIF algorithm, and the WCET $W_i^{m_i}$ is derived from the WCET-aware C compiler. With the fixed memory allocation results, the MCIF algorithm is applied to derive the results with 1000 tests for each evaluation set, and the average of the required numbers of islands is then reported.

The evaluation results of the second experiment are also normalized to the number of required islands when using no scratchpad memory. As shown in Fig. 4, the MCIF algorithm could also reduce the number of required islands while the size of the fast local memory is increasing. However, the performance of the MCIF algorithm is not as good as the performance of MCI algorithm (in Fig. 2) because the memory allocation is not optimized in this experiment. From Figs. 2 and 4, we can see that the MCI algorithm outperforms the MCIF algorithm when the size of local memory is small because memory allocation is a critical issue when the size of the local memory is relatively small. When the size of the local memory is large, the bottleneck of the system resource co-management is moved from the remaining local memory space to the available processor utilization. Thus, the performance of the MCIF algorithm is similar to that of the MCIF algorithm when the size of the local memory is large because the MCIF algorithm also carefully partition tasks onto cores of islands to efficiently use the computing utilization of all cores.

5.3 Extensive Evaluations

In the third experiment, as shown in Fig. 5, we present a case study by combining the 82 different real-life benchmarks from MRTC, MediaBench, UTDSP, NetBench, and DSPstone into a task set, where each benchmark maps to a task with the corresponding WCETs. The minimum inter-arrival time of each task is then set such that the worst-case utilization is 40 percent when using no scratchpad memory. In this experiment, *First-Fit* is the solution which uses the first-fit approach to assign tasks to cores without the consideration of the fast local memory. *5 percent-MCI* (resp. *10 percent-MCI* and *20 percent-MCI*) is the solution of the MCI algorithm by using the fast local memory with the size of 5 percent (resp. 10 and 20 percent) of the total size of the program codes. The number of the cores in an island is varied from 1 to 8, where all cores are identical (having the same computing power) in all configurations.

From this case study, we can see that using the fast local memory can help system designers to reduce the number of required islands because some frequently used blocks of tasks can be mapped to the fast local memory. While the number of cores in an island is relatively small, the available processor utilization is the critical resource. Therefore, for

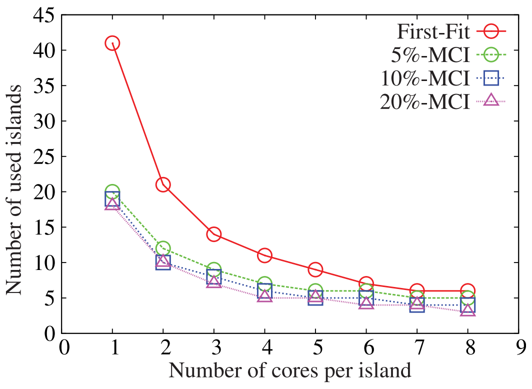

Fig. 5. The evaluation with different numbers of cores per island.

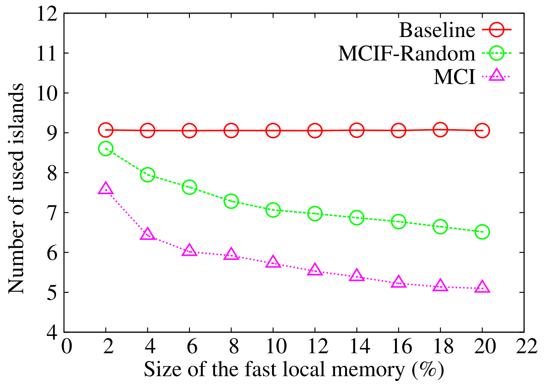

Fig. 6. The evaluation with different sizes of the fast local memory.

an extreme case that there is only one core in each island, First-Fit, 5 percent-MCI, 10 percent-MCI, and 20 percent-MCI require 41, 20, 19, and 18 islands, respectively. For this case, a small-size fast local memory can provide a significant improvement. For another extreme case that each island consists of 8 cores, First-Fit, 5 percent-MCI, 10 percent-MCI, and 20 percent-MCI require six, five, four and three islands, respectively. For this case, since the resource bottleneck is moved to the fast local memory, using more blocks of the fast local memory can further reduce the number of required islands.

In the last experiment, as shown in Fig. 6, we would like to show the performance of the MCIF algorithm and analyze the memory allocation strategy of the MCI algorithm. For this experiment, each island consists of 4 cores, and the 82 benchmarks are included, where two tasks are created for each benchmark with the derived WCETs. The minimum inter-arrival time of each task is set such that the worst-case utilization forms a uniform distribution between 0.4 and 40 percent when using no scratchpad memory. The size of the fast local memory in each island is configured as 2 to 20 percent of the total size of the program codes of all tasks.

In this experiment, we consider three solutions as follows: (1) *Baseline* uses only the first-fit approach to assign each task on a core by using no fast local memory for the WCET analysis. (2) *MCI* represents the MCI algorithm. (3) *MCIF-Random* uses the MCIF algorithm to partition tasks onto cores. For the memory allocation, MCIF-Random uses the same amount of the fast local memory as that used by MCI, and the fast local memory space is randomly distributed among the tasks. The results shown in Fig. 6 illustrate that MCIF-Random always outperforms Baseline, and MCI outperforms MCIF-Random. The performance improvement of MCIF-Random compared to baseline is from the using of the fast local memory and the proposed task assignment algorithm. The performance improvement of MCI compared to MCIF-Random is due to the intelligent memory allocation algorithm which minimizes the lower bound of the number of the required islands.

6 Conclusions and Future Work

In this paper, we explore the joint considerations of task scheduling and memory allocation for real-time tasks on island-based multi-core platforms. For an implicit-deadline sporadic task set, our algorithms provide asymptotic 4-approximation and $(\alpha_{BIN} + 2)$-approximation bounds for minimizing the numbers of required islands for the IBRT and IBRTA problems, respectively, in which α_{BIN} is the asymptotic approximation bound of the algorithm adopted for the bin packing problem. The performance of the presented algorithms is evaluated with 82 real-life benchmarks with different numbers of the cores in an island and with different sizes of the fast local memory.

When *constrained-deadline* sporadic tasks are further considered, it is no more guaranteed that all timing constraints can always be satisfied if the utilization of each core is no more than the achievable utilization factor, where a sporadic task is said to have a *constrained deadline* if its relative deadline is no longer than its minimum inter-arrival time. In this extension, the *density* of a task is defined as the WCET divided by the relative deadline, and we can use the density of a task to replace the utilization of the task if the ratio of the minimum inter-arrival time to the relative deadline is bounded by a constant γ. Within this patch, the proposed algorithms can still work for constrained-deadline sporadic tasks, but the approximation bounds will be increased by γ times.

To provide better approximation algorithms for constrained-deadline sporadic tasks, we should look into the required computing time of each task at any time point (it is also called as *demand bound function* of the task in [6], [12], [13]) for the future work of this paper. Specifically, the recent result [11] has studied how to allocate the local memory to minimize the required size of the scratchpad memory to meet the timing constraints for constrained-deadline tasks by using EDF and RM in uniprocessor systems.

Acknowledgments

This work was partially supported by Baden Württemberg MWK Juniorprofessoren-Program, by the National Science Council of Taiwan, R.O.C., under grant 100-2221-E-002-120-MY3, by the Excellent Research Projects of National Taiwan University under grant 102R890822, and by Deutsche Forschungsgesellschaft (DFG) under grant FA 1017/1-1 and EU COST Action IC1202: Timing Analysis On Code- Level (TACLe). This work was an extended version of the paper [8] that appeared in the Asia and South Pacific Design Automation Conference (ASP-DAC) 2013.

References

[1] AbsInt Angewandte Informatik GmbH, “aiT: Worst-Case Execution Time Analyzers, ” http://www.absint.com/ait, 2013.
[2] Malardalen WCET Research Group, “WCET Benchmarks,” http://www.mrtc.mdh.se/projects/wcet, 2013.
[3] UTDSP Benchmark Suite, http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/UTDSP.tar.gz, 2013.
[4] N. Bansal, A. Caprara, and M. Sviridenko, “Improved Approximation Algorithms for Multidimensional Bin Packing Problems,” *Proc. IEEE 47th Ann. Symp. Foundations of Computer Science (FOCS)*, 2006.
[5] S. Baruah, “Partitioning Sporadic Task Systems Upon Memory-Constrained Multiprocessors,” *ACM Trans. Embedded Computing Systems*, vol. 12, article 78, 2013.

[6] S. Baruah and N. Fisher, "The Partitioned Multiprocessor Scheduling of Sporadic Task Systems," *Proc. IEEE 26th Int'l Real-Time Systems Symp. (RTSS)*, pp. 321-329, 2005.

[7] M. Bertogna, M. Cirinei, and G. Lipar, "Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms," *IEEE Trans. Parallel and Distributed Systems*, vol. 20, no. 4, pp. 553-566, Apr. 2009.

[8] C.-W. Chang, J.-J. Chen, T.-W. Kuo, and H. Falk, "Real-Time Partitioned Scheduling on Multi-Core Systems with Local and Global Memories," *Proc. 18th Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2013.

[9] C.-W. Chang, J.-J. Chen, W. Munawar, T.-W. Kuo, and H. Falk, "Partitioned Scheduling for Real-Time Tasks on Multiprocessor Embedded Systems with Programmable Shared SRAMs," *Proc. ACM 10th Int'l Conf. Embedded Software (EMSOFT)*, 2012.

[10] S. Chattopadhyay and A. Roychoudhury, "Scalable and Precise Refinement of Cache Timing Analysis via Model Checking," *Proc. IEEE 32nd Real-Time Systems Symp. (RTSS)*, 2011.

[11] J.-J. Chen, "Task Set Synthesis with Cost Minimization for Sporadic Real-Time Tasks," *Proc. IEEE Real-Time Systems Symp. (RTSS)*, 2013.

[12] J.-J. Chen and S. Chakraborty, "Resource Augmentation Bounds for Approximate Demand Bound Functions," *Proc. IEEE Real-Time Systems Symp. (RTSS)*, pp. 272-281, 2011.

[13] J.-J. Chen and S. Chakraborty, "Partitioned Packing and Scheduling for Sporadic Real-Time Tasks in Identical Multiprocessor Systems," *Proc. 24th Euromicro Conf. Real-Time Systems (ECRTS)*, pp. 24-33, 2012.

[14] Q. Chen, M. Guo, and Z. Huang, "Adaptive Cache Aware Bi-Tier Work-Stealing in Multi-Socket Multi-Core Architectures," *IEEE Trans. Parallel and Distributed Systems*, vol. 24, no. 12, pp. 2334-2343, Dec. 2013.

[15] R.I. Davis and A. Burns, "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems," *ACM Computing Surveys*, vol. 43, no. 4, article 35, Oct. 2011.

[16] W.F. de la Vega and G.S. Lueker, "Bin Packing Can Be Solved Within 1+Epsilon in Linear Time," *Combinatorica*, vol. 1, no. 4, pp. 349-355, 1981.

[17] G. Dosa, "The Tight Bound of First Fit Decreasing Bin-Packing Algorithm is $FFD(I) \leq \frac{11}{9}OPT(I) + \frac{6}{9}$," *Proc. First Int'l Conf. Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pp. 1-11, 2007.

[18] H. Falk and J.C. Kleinsorge, "Optimal Static WCET-Aware Scratchpad Allocation of Program Code," *Proc. ACM/IEEE 46th Design Automation Conference (DAC)*, 2009.

[19] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[20] P. Gepner and M.F. Kowalik, "Multi-Core Processors: New Way to Achieve High System Performance," *Proc. Int'l Symp. Parallel Computing in Electrical Engineering (PARELEC)*, 2006.

[21] J. Guo, M. Lai, Z. Pang, L. Huang, F. Chen, K. Dai, and Z. Wang "Hierarchical Memory System Design for a Heterogeneous Multi-Core Processor, *Proc. ACM Symp. Applied Computing (SAC)*, 2008.

[22] Y. Guo, Q. Zhuge, J. Hu, M. Qiu, and E.H.-M. Sha, "Optimal Data Allocation for Scratch-Pad Memory on Embedded Multi-Core Systems," *Proc. Int'l Conf. Parallel Processing (ICPP)*, pp. 464-471, 2011.

[23] Z. Huang, M. Zhu, and L. Xiao, "LvtPPP: Live-Time Protected Pseudo-Partitioning of Multi-Core Shared Caches," *IEEE Trans. Parallel and Distributed Systems*, vol. 24, no. 8, pp. 1622-1632, Aug. 2013.

[24] J.A. Kahle, M.N. Day, H.P. Hofstee, C.R. Johns, T.R. Maeurer, and D. Shippy, "Introduction to the Cell Multiprocessor," *IBM J. Research and Development*, vol. 49, no. 4.5, pp. 589-604, 2005.

[25] S. Kang and A.G. Dean, "Leveraging Both Data Cache and Scratchpad Memory through Synergetic Data Allocation," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2012.

[26] S. Kato, K. Lakshmanan, Y. Ishikawa, and R. Rajkumar, "Resource Sharing in GPU-Accelerated Windowing Systems," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2011.

[27] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt, "Gdev: First-Class GPU Resource Management in the Operating System," *Proc. USENIX Conf. Ann. Technical Conf. (ATC)*, 2012.

[28] S. Kato, N. Yamasaki, and Y. Ishikawa., "Semi-Partitioned Scheduling of Sporadic Task Systems on Multiprocessors," *Proc. 21st Euromicro Conf. Real-Time Systems (ECRTS)*, pp. 249-258, 2009.

[29] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury, "Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds," *Proc. Euromicro Conf. Real-Time Systems (ECRTS)*, 2011.

[30] C. Lee, M. Potkonjak, and W.H. Mangione-Smith, "Mediabench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," *Proc. IEEE/ACM Ann. Int'l Symp. Microarchitecture (MICRO)*, 1997.

[31] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S.K. Reinhardt, and T.F. Wenisch, "Disaggregated Memory for Expansion and Sharing in Blade Servers," *Proc. 36th Ann. Int'l Symp. Computer Architecture (ISCA)*, pp. 267-278, 2009.

[32] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46-61, 1973.

[33] W. Lunniss, S. Altmeyer, C. Maiza, and R.I. Davis, "Integrating Cache Related Pre-Emption Delay Analysis Into Edf Scheduling," *Proc. IEEE 19th Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2013.

[34] G. Memik, W.H. Mangione-Smith, and W. Hu, "Netbench: A Benchmarking Suite for Network Processors," *Proc. IEEE/ACM Int'l Conf. Computer Aided Design (ICCAD)*, 2001.

[35] M. Niemeier, A. Wiese, and S. Baruah, "Partitioned Real-Time Scheduling on Heterogeneous Shared-Memory Multiprocessors," *Proc. 23rd Euromicro Conf. Real-Time Systems (ECRTS)*, 2011.

[36] O. Ozturk, M. Kandemir, and I. Kolcu, "Shared Scratch-Pad Memory Space Management," *Proc. Seventh Int'l Symp. Quality Electronic Design (ISQED)*, 2006.

[37] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Timing Analysis for TDMA Arbitration in Resource Sharing Systems," *Proc. IEEE 16th Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2010.

[38] T. Ungerer, F.J. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, H. Casse, C. Rochange, E. Quinones, S. Uhrig, M. Gerdes, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, J. Mische, M. Paolieri, and J. Wolf, "Mersa: Multicore Execution of Hard Real-Time Applications Supporting Analyzability," *Micro*, vol. 30, pp. 66-75, 2010.

[39] I. Wald, "Fast Construction of Sah Bvhs on the Intel Many Integrated Core (MIC) Architecture," *IEEE Trans. Visualization and Computer Graphics*, vol. 18, no. 1, pp. 47-57, Jan. 2012.

[40] J. Whitham, R.I. Davis, N.C. Audsley, S. Altmeyer, and C. Maiza, "Investigation of Scratchpad Memory for Preemptive Multitasking," *Proc. IEEE 33rd Real-Time Systems Symp. (RTSS)*, 2012.

[41] G.J. Woeginger, "There is no Asymptotic PTAS for Two-Dimensional Vector Packing," *Information Processing Letters*, vol. 64, no. 6, pp. 293-294, 1997.

[42] H. Yunz, G. Yaoz, R. Pellizzoni, M. Caccamo, and L. Sha, "Leveraging Both Data Cache and Scratchpad Memory through Synergetic Data Allocation," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2013.

[43] V. Zivojnovic, J.M. Velarde, C. Schlager, and H. Meyr, "DSPstone: A DSP-Oriented Benchmarking Methodology," *Proc. Int'l Conf. Signal Processing and Technology (ICSPAT)*, 1994.

Che-Wei Chang received the BS degree in computer science and information engineering from National Chiao Tung University, Taiwan, in 2006 and the PhD degree in computer science and information engineering from National Taiwan University, Taiwan and won the PhD Dissertation Award of the Institute of Information and Computing Machinery (IICM) in 2012. He is an assistant professor of the Department of Computer Science and Information Engineering, Chang Gung University, Taiwan. He was a visiting junior researcher in the Karlsruhe Institute of Technology (KIT), Germany, in 2011. He has been a postdoctoral research fellow at the Research Center for Information Technology Innovation, Academia Sinica, Taiwan from 2012 to 2013. His research interests include energy-efficient scheduling, fast-booting designs and multi-core management. He is a member of the IEEE.

Jain-Jia Chen received the BS degree from the Department of Chemistry, National Taiwan University, in 2001, and the PhD degree from the Department of Computer Science and Information Engineering, National Taiwan University, Taiwan, in 2006. He is a junior professor at the Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany. After finishing the compulsory civil service in Dec. 2007, between Jan. 2008 and April 2010, he was a postdoc researcher at Computer Engineering and Networks Laboratory (TIK) in ETH Zurich, Switzerland. He joined KIT in May 2010. His research interests include real-time systems, embedded systems, energy-efficient scheduling, power-aware designs, temperature-aware scheduling, and distributed computing. He received Best Paper Awards from ACM Symposium on Applied Computing (SAC) in 2009 and IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) in 2005 and 2013.

Tei-Wei Kuo received the BSE and PhD degrees in computer science from the National Taiwan University and the University of Texas at Austin, in 1986 and 1994, respectively. He is currently a distinguished professor of the Department of Computer Science and Information Engineering, National Taiwan University, where he was the department chair between 2005 and 2008. He is also the executive director and a research fellow of the Intelligent and Ubiquitous Computing Thematic Center of the Research Center of the IT Innovation, Academia Sinica, Taiwan. He chairs the Embedded Systems Group of the National Networked Communication Program office, Taiwan, since 2010, and serves as the Program Director of the Computer Science Division of the Taiwan National Science Council from 2013. His research interests include embedded systems, real-time systems, and non-volatile memory. He has published a number of papers in top journal and conferences with two best paper awards so far and owns more than 15 patents in flash memory storage systems and real-time operating systems. He received the Distinguished Research Award from the National Science Council of Taiwan, the Young Scholar Research Award from the Academia Sinica, and the 10 Outstanding Young Persons Award of Taiwan. He was in the editorial board of the *Journal of Real-Time Systems, IEEE Embedded Systems Letters*, and *IEEE Transactions on Industrial Informatics.* He was a program chair and a general chair of the *IEEE Real-Time Systems Symposium* and serves as a program committee members of many top conferences in his fields, such as DAC, RTAS, EMSOFT, CODES+ISSS, ICDCS, etc. He is a fellow of the IEEE.

Heiko Falk received the PhD degree in computer science from the University of Dortmund, Germany, in 2004. From 2004 until 2011, he worked as assistant professor in the embedded systems group at the Technical University of Dortmund. Since 2011, he is full professor for embedded systems and real-time systems at Ulm University (Germany). His PhD focused on high-level source code optimizations. Typical embedded multimedia applications only use a small fraction of their execution time to compute audio or video data. Most of the execution time is used to evaluate complex control flow. Motivated by this observation, he developed novel techniques for control flow optimization at the source code level. In the last years, the focus of his work is on code generation and optimization for performance and predictability of safety-critical real-time systems. The WCC compiler initially established by him and developed by the research teams led by him is the currently only known compiler which is able to systematically reduce the worst-case execution time (WCET) of programs by tightly integrating static timing analyses into the code generation and optimization stage. He works on novel concepts to handle parallelism during real-time analysis and optimization. Mutual interferences between tasks running preemptively on the same CPU, or between tasks running on different cores and using shared resources are the key challenges of his current work. In the future, WCC's optimizations will be applied to all software components of complex multitask and multicore systems in order to globally optimize the entire system's real-time capabilities.