2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)

Strict Partitioning for Sporadic Rigid Gang Tasks

Binqi Sun*, Tomasz Kloda†, Marco Caccamo*
*Technical University of Munich, Germany
†LAAS-CNRS, Université de Toulouse, INSA, Toulouse, France
Email: binqi.sun@tum.de, tkloda@laas.fr, mcaccamo@tum.de

***Abstract*—The *rigid gang* task model is based on the idea of executing multiple threads simultaneously on a *fixed* number of processors to increase efficiency and performance. Although there is extensive literature on global rigid gang scheduling, partitioned approaches have several practical advantages (*e.g.*, task isolation and reduced scheduling overheads). In this paper, we propose a new partitioned scheduling strategy for rigid gang tasks, named *strict partitioning*. The method creates disjoint partitions of tasks and processors to avoid inter-partition interference. Moreover, it tries to assign tasks with similar volumes (*i.e.*, parallelisms) to the same partition so that the intra-partition interference can be reduced. Within each partition, the tasks can be scheduled using any type of scheduler, which allows the use of a less pessimistic schedulability test. Extensive synthetic experiments and a case study based on Edge TPU benchmarks show that strict partitioning achieves better schedulability performance than state-of-the-art global gang schedulability analyses for both preemptive and non-preemptive rigid gang task sets.**

***Index Terms*—Real-time scheduling, Gang parallel task model, Partitioned scheduling, Tensor processing unit**

I. Introduction

Gangs are fine-grained parallel jobs that have to start at the same time and execute in parallel across multiple processing units. Gang scheduling has been widely used in high-performance computing [1], [2], distributed cloud [3], and containers [4]. In recent years, it is also increasingly gaining popularity in embedded systems. Especially, gang scheduling is well-suited for emerging deep-learning applications [5], [6] deployed on highly parallel hardware accelerators [7], [8] that require significant computational resources and may involve complex inter-task communication and dependencies.

Most of the approaches for gang scheduling in real-time systems are *global* [5], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], wherein tasks are not pinned to any particular processor and can start execution on any available processors. Despite the relatively lesser interest in *partitioned* scheduling [19] (*i.e.,* tasks are statically allocated to individual processors), such type of scheduling can offer benefits in specific scenarios where task isolation and reduced scheduling overheads [20], [21], [22] are essential requirements.

The migration cost is one of the major drawbacks of global scheduling. Parallel tasks are often deployed on hardware accelerators where the migration and setup times are usually high (*e.g., FPGA* dynamic partial reconfiguration [23] or *Edge TPU* model parameter loading [12], [24]). Figure 1 shows the configuration and the net execution times of seven representative deep neural networks (DNNs) on an AI Accelerator card integrated with 16 Edge TPUs. Each Edge TPU has 8

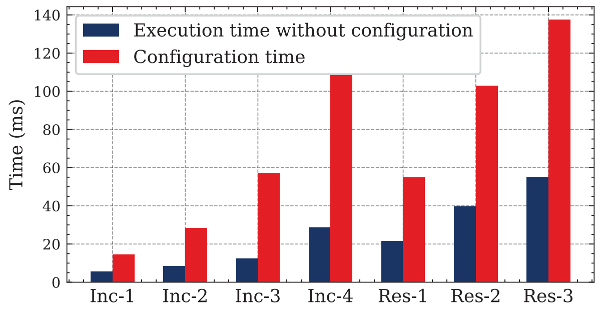


Fig. 1: Configuration time vs net execution time of DNN inferences on 16 Edge TPUs. "Inc-1,2,3,4" denote Inception-v1 [25], v2 [26], v3 [26], v4 [27], respectively; "Res-1,2,3" denote ResNet-50,101,152 [28], respectively.

MB on-chip memory and can be used to store DNN model weights. Partitioned scheduling reserves processing units for each model so that the model weights can be cached in the internal memory statically, eliminating the need for frequent and costly memory allocation during runtime.

The current state-of-the-art multiprocessor analyses for sequential tasks under global scheduling rely primarily on sufficient-only conditions [29]. Their pessimism [30] is due to the fact that synchronous arrival sequence is not necessarily the worst-case scenario [31], [32]. Global gang scheduling, besides the same old problems, is burdened with interference overestimation resulting from the difference in gang tasks' parallelism levels (*e.g.,* finding the set of interfering jobs that may exactly fit on the available processors [10] and unbounded priority-inversion in non-preemptive gang scheduling [33]).

While partitioned scheduling can help reduce runtime overheads, the task allocation is equivalent to the bin-packing problem [34] and is hence highly intractable (NP-hard in the strong sense [35]). As different gang tasks may have different parallelism levels, the gang partitioning is closely related to the two-dimensional bin packing problem (*i.e.,* packing a set of rectangular items into rectangular bins) [36]. Thus, the tractability challenge extends to the gang partitioning problem. Moreover, partitioned systems of sequential tasks can be easily analyzed using exact uniprocessor schedulability tests. However, for parallel tasks, more complicated tests may be necessary to account for the anomalies that may occur in partitioned gang systems where different gang task groups share some, but not all, processors (see Section IV) [19].

This paper proposes a simple yet effective method, named *strict partitioning*, for scheduling rigid gang tasks (*i.e.,* gang tasks with fixed parallelism levels) on identical multiprocessor platforms. It builds disjoint partitions of tasks and processors so that the tasks in different partitions do not interfere with each other. Within the boundaries of each partition, tasks can run under any type of online scheduler. Moreover, strict partitioning tries to group tasks with similar parallelism levels onto the same partition so that uniprocessor scheduler (*e.g., Deadline Monotonic (DM) [37], Earliest Deadline First (EDF)* [38]) and exact schedulability tests are applicable.

The contributions are summarized as follows.

- We propose a new *strict partitioning* strategy for scheduling rigid gang tasks.
- We propose a *first-fit decreasing volume (FFDV)* heuristic to partition rigid gang tasks and multiprocessor platforms.
- We present two strict partitioning variants `SP-U` and `SP-G` by combining FFDV with different online schedulers. For `SP-U`, we prove utilization bounds. For `SP-G`, we improve FFDV to achieve better performance.
- We evaluate the proposed strategy and algorithms by comparing them with state-of-the-art preemptive and non-preemptive gang scheduling techniques on synthetic task sets and a case study based on Edge TPU benchmarks.

The remainder of the paper is organized as follows. Relevant previous works are presented in Section II. Section III describes the proposed strict partitioning strategy, and Section IV compares strict partitioning with existing gang scheduling methods through illustrative examples. We propose strict partitioning algorithms in Section V and present evaluation results in Section VI. Section VII concludes the paper.

II. Related Work

Three main paradigms are adopted to schedule real-time tasks on a multiprocessor [39], [40]: *global* (each job can execute upon any processor), *partitioned* (each job is mapped to an individual processor), and *semi-partitioned* (several tasks are split into subtasks assigned to different processors).

Partitioned scheduling for sequential tasks. The partitioned schedulers incur low runtime overheads but require solving a task-to-processor bin-packing problem. Since the partitioning problem is NP-hard in the strong sense [41], many approximation methods (based on bin-packing heuristics) for sequential preemptive tasks were proposed [42], [43], [44], [45], [46], [47], [48]. Other works formulate the partitioning as an integer linear programming (ILP) problem [49], [50]. The partitioning methods for non-preemptive scheduling [51], [52], [53] try to cluster on the same processor the tasks with similar period ranges to reduce the blocking factor. *Semi-partitioned* approaches reclaim spare processing capacity in partitioned systems by splitting certain tasks among processors [54], [55].

Partitioned scheduling for parallel tasks. Casini et al. [56] and Wu et al. [57] propose partitioning algorithms for parallel tasks modeled as a *directed acyclic graph (DAG)*. Zahaf et al. [58] consider the allocation of conditional DAGs to remove unnecessary preemptions. *Federated scheduling* categorizes parallel tasks, according to their utilization factor, into heavy tasks and light tasks [59], [60]. Each heavy task is assigned a number of dedicated processors. The remaining processors are assigned to all light tasks, which are executed as sequential tasks. In [61], [62], heavy tasks are allowed to share processors, and in [63], they can partially execute on the processors from the light tasks' pool. However, unlike DAG tasks, whose subtasks can run on a different number of processors and do not have to start execution synchronously, the parallelism level of a rigid gang task cannot be changed at runtime. Thus, these partitioning methods designed for DAG tasks cannot be applied to rigid gang scheduling. The major concern of partitioning rigid gang tasks is how to effectively group tasks with fixed parallelism levels that are not necessarily the same.

Gang scheduling. The problem of rigid gang scheduling was shown to be NP-hard [64]. Several works have exploited the idea of *proportionate progress* [65], [66] to derive the optimal scheduling algorithms for periodic implicit deadline *rigid* [17] and *malleable* [67] (*i.e.,* job parallelism level can be adjusted at runtime) gang tasks. Other streams of works studied rigid gang tasks under *global fixed-priority* preemptive [68], [13] and non-preemptive [12], [11], [10], as well as *global EDF* preemptive [15], [5], [18], [69] and non-preemptive [33], [9] scheduling. Some recent works have also considered moldable gang tasks [14]. In this work, we apply a partitioned approach and compare its schedulability performance with the above state-of-the-art global policies in Section VI.

Gang partitioning. Ueter et al. [19] proposed the concept of stationary scheduling and a task-to-processor allocation heuristic for rigid gangs. In stationary scheduling, each gang task is mapped to a set of processors such that the number of processors equals the task parallelism level. The schedulability problem is equivalent to the uniprocessor schedulability of self-suspending tasks [70]. If tasks τ_i and τ_j run on the same processors but τ_j runs also on other processors where another task τ_k is allocated to, then τ_k can interfere indirectly with the τ_i's execution ($\tau_k \rightarrow \tau_j \rightarrow \tau_i$). In this work, we propose a different partitioning approach where tasks cannot run on the processors allocated to the tasks from different partitions. This eliminates the interference between the partitions and simplifies the scheduling problem, enabling the use of exact schedulability tests. On the other hand, more constrained processor allocation can lead to less efficient utilization of processing capacity. In Section VI, we compare both approaches and identify their respective advantages.

III. Strict Partitioning for Rigid Gang Tasks

A. Task and Platform Model

We consider a multiprocessor platform with a set Π of M identical processors. A task set τ is composed of n independent sporadic *rigid gang* tasks executing on Π. Each task $\tau_i \in \tau$ ($1 \leq i \leq n$) gives rise to an infinite sequence of jobs with consecutive jobs' invocations (arrivals) separated by at least T_i time units (*i.e., sporadic* task). We use J_i to denote a job of task τ_i. Job J_i released at time (arrival time) r_i has an absolute deadline $r_i + D_i$ and must complete its execution by that time where $D_i \leq T_i$ (*i.e., constrained* deadlines). Each job of task τ_i executes simultaneously on m_i processors for at most C_i time units (m_i is called *task volume* or *parallelism* interchangeably). As a result, a gang task can be characterized by a 4-tuple $\tau_i = (C_i, T_i, D_i, m_i)$. We assume that all the above parameters are non-negative integers. The tasks do not self-suspend and cannot be blocked by other tasks other than due to contention on processors. Additionally, we define the *sequential utilization* of task τ_i as $u_i = C_i/T_i$ and its utilization as $U_i = m_i \cdot C_i/T_i = m_i \cdot u_i$. The task set utilization is the sum of task utilizations, $U = \sum_{i=1}^{n} U_i$. Moreover, $\overline{m}$ and $\underline{m}$ denote the maximum and minimum task volume among all the tasks in the task set.

B. Strict Partitioning Strategy

The strict partitioning strategy includes two aspects: *offline partitioning* and *online scheduling*.

1) Offline Partitioning: Offline partitioning involves dividing processors into disjoint partitions and statically allocating tasks to these partitions.

Definition III.1 (Strict Partitioning)**.** Given a set of identical processors Π and a set of rigid gang tasks τ, its *strict partitioning* is such assignment where processors Π are divided into disjoint subsets $\rho = \{\rho_j \subseteq \Pi, \forall j\}$ and each task $\tau_k \in \tau$ is assigned to one and only one partition.

The set of tasks assigned to processor partition ρ_j is denoted by $\tau(\rho_j) \subseteq \tau$ and the volume of partition ρ_j (*i.e.*, the number of processors allocated to ρ_j) by $|\rho_j|$.

2) Online Scheduling: Once the offline partitioning is determined, the tasks allocated to each processor partition will be scheduled by an *online* real-time scheduler at runtime. Both *uniprocessor task schedulers* (*e.g.*, uniprocessor *DM* [37] and *EDF* [38]) and *global gang task schedulers* (*e.g.*, global *FP* [10], [12] and *EDF* [13] gang schedulers) can be applied. When a uniprocessor task scheduler is used, only one job is allowed to execute on the processor partition at each time instant (*i.e.*, different jobs cannot run in parallel). By contrast, a global gang task scheduler allows tasks to start their execution whenever a sufficient number of processors are available within the partition.

IV. Illustrative Examples: Comparative Study

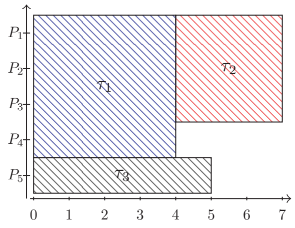
We will illustrate the tradeoffs that arise when considering different approaches to rigid gang scheduling. It includes the pessimism of global fixed-priority scheduling, a comparison between *stationary* and *strict* partitioning of rigid gangs upon multiprocessor, and priority-inversion in non-preemptive global policies. In what follows, we assume that the tasks are scheduled by a fixed-priority scheduler and indexed in decreasing priority order (*i.e.,* task τ_1 has the highest priority).

A. Pessimisms in Global Gang Schedulability Analyses

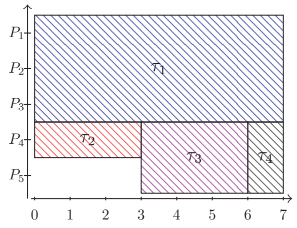
We recall two phenomena in global gang schedulability analyses identified by Lee et al. [13].

1) Non-parallel Execution: Classic multiprocessor frameworks for response time analysis of sequential tasks [71], [72] consider that every higher-priority task can interfere with the execution of the task under analysis, regardless of the execution of other interfering tasks. In rigid gang scheduling, certain tasks cannot run at the same time if there are not enough processors for all of them [13].

Example IV.1. Let us consider three sporadic rigid gang tasks $\tau_1 = (4, 10, 10, 4)$, $\tau_2 = (3, 10, 10, 3)$, and $\tau_3 = (5, 10, 10, 1)$ running on five, M=5, identical processors P_{1-5}. We check task τ_3's schedulability. Although tasks τ_1 and τ_2 have higher priorities, they cannot interfere with τ_3's execution. Given their parallelism levels, τ_1 and τ_2 cannot execute at the same time, $m_1 + m_2 = 4 + 3 = 7 > M = 5$, and there is always at least one processor for τ_3. Figure 2a shows a sample schedule.



(a) Non-parallel execution



(b) Overestimation of interference

Fig. 2: New sources of pessimism in global gang scheduling.

In general, identifying all sets of non-parallel executing jobs can be solved as the subset sum problem (*i.e.*, find a subset of the integers that sum precisely to a given target value) [73]. The problem is known to be NP-hard. A heuristic is proposed in [13] to address such non-parallel execution constraints.

2) Interference Overestimation: Classic multiprocessor frameworks for response time analysis of sequential tasks [71], [72] consider the interference when all M processors are occupied by the interfering tasks. In gang scheduling, task τ_k cannot execute when at least $M - m_k + 1$ processors are busy. This complicates the interference estimation as a part of the interfering tasks can be executed on a different number of processors. Such surplus interference can be reduced by deducting the overestimation by a heuristic proposed in [13]. However, computing the maximum overestimation deduction (*i.e.*, finding the optimal τ' in Theorem 5 of [13]) requires solving a knapsack problem, which is again NP-hard.

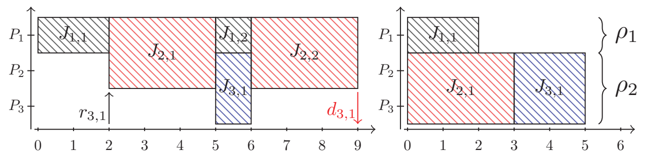
Example IV.2. Let us consider four sporadic rigid gang tasks $\tau_1 = (7, 10, 10, 3)$, $\tau_2 = (3, 10, 10, 1)$, $\tau_3 = (3, 10, 10, 2)$, and $\tau_4 = (1, 10, 10, 2)$ running on five, M=5, identical processors P_{1-5}. We check task τ_4's schedulability. To do that, we need to quantify the interference of higher-priority tasks. These tasks interfere with τ_4 when they run on at least four processors. Figure 2b illustrates the interference computation. When τ_1 and τ_2 run concurrently in interval $[0, 3)$, four processors are busy, and the workload of interfering tasks is $4 \cdot 3$=12. When τ_1 and τ_3 run concurrently in interval $[3, 6)$, five processors are busy, and the workload of interfering tasks is $5 \cdot 3$=15. Both intervals have equal lengths, and bigger interference during $[3, 6)$ does not affect the task under study τ_4 more than smaller interference during interval $[0, 3)$.

While the heuristics proposed in [13] can mitigate the aforementioned pessimisms, the experimental results in Section VI suggest that these pessimisms still exist in the analyses since it is not computationally tractable to solve them optimally.

B. Stationary Scheduling and Strict Partitioning

In *stationary scheduling* [19], task τ_k with parallelism level m_k is allocated to m_k processors. Some of these processors can be allocated to different tasks. If these tasks are also allocated to processors other than those shared with τ_k, then they may transfer the interference from these processors to the processors shared with τ_k. Such interfering behavior can be modeled and over-approximated by the corresponding sequential tasks with dynamic *self-suspension* [70]. In contrast, the strict partitioning proposed in this paper forbids overlapping in processor allocation and thus avoids scheduling anomalies related to the self-suspension task model.

Example IV.3. Consider a system comprising three sporadic rigid gang tasks $\tau_1 = (2, 5, 5, 1)$, $\tau_2 = (3, 6, 6, 2)$, and $\tau_3 = (2, 7, 7, 2)$ running on three identical processors P_1, P_2, and P_3. Figure 3 shows task-to-processor assignments done by (a) stationary gang heuristic and (b) strict partitioning heuristic. To avoid the combinatorial explosion, each heuristic uses a specific strategy. The stationary heuristic assigns the tasks to processors in a greedy way to have possibly the best processor utilization. The strict partitioning favors volume homogeneity within the same partition.



(a) Stationary gang (b) Strict partitioning

Fig. 3: Strict partitioning better than stationary scheduling.

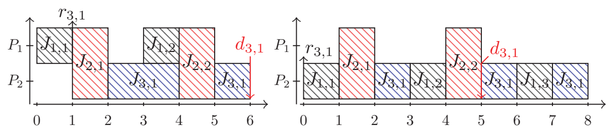
In the presented example, the stationary heuristic would allocate τ_1 to P_1 and then τ_2 to P_1 and P_2, as τ_1 and τ_2 can be both schedulable on P_1. In contrast, the strict partitioning, to avoid mixing different parallelism levels, would open a new partition $\rho_2 = \{P_2, P_3\}$ for τ_2 (τ_2 has a different volume as τ_1 running on P_1). This choice will be critical when assigning τ_3.

In the stationary gang assignment (a), since the utilization of P_1 is at 90% and it cannot accommodate τ_3 having sequential utilization of u_3=28.5%, the heuristic allocates τ_3 to P_2 (current utilization of 50%) and P_3 (idle). However, τ_3 is not schedulable on P_2 and P_3 as τ_2 exhibits self-suspending behavior [70] due to τ_1 preempting τ_2 on P_1. Task τ_3's first job, $J_{3,1}$, can miss its deadline when released at $r_{3,1}$=2 while tasks τ_1 and τ_2 release their first jobs, $J_{1,1}$ and $J_{2,1}$, at 0. The τ_2's second instance, $J_{2,2}$, is released at 6 and, if the τ_1's second instance, $J_{1,2}$, completes sooner than its worst-case execution time, $J_{2,2}$ will start its execution immediately on P_1 and P_2 occupying both processors until 9. Consequently, $J_{3,1}$ will not meet its deadline.

In the strict partitioning assignment (b), τ_3 is allocated to the partition $\rho_2 = \{P_1, P_2\}$ which, at the moment of assignment, is comprised of τ_2 only (each processor has utilization of 50%). Tasks τ_2 and τ_3 have both a volume of two and thus processors P_2 and P_3 are seen as a single resource. The tasks behave like on a traditional uniprocessor platform and are schedulable, showing no anomalies.

However, strict partitioning might lead to processor underutilization when tasks have very different parallelism levels.

Example IV.4. Consider a system comprising three sporadic rigid gang tasks $\tau_1 = (1, 3, 3, 1)$, $\tau_2 = (1, 4, 4, 2)$, and $\tau_3 = (3, 5, 5, 1)$ running on two identical processors P_1 and P_2. Figure 4 shows task-to-processor assignments done by (a) stationary gang heuristic and (b) strict partitioning heuristic. In stationary gangs, sequential tasks are assigned to two different processors. Despite the self-suspension phenomenon still present, all tasks are schedulable (task τ_3's critical instant is at $r_{3,1}$=1 when other tasks start at 0). Strict partitioning, because of the τ_2's volume, creates only one partition for all the tasks. The problem simplifies to uniprocessor scheduling and the potential parallelism cannot be exploited. Task τ_3 released at $r_{3,1}$=0 misses its deadline at $d_{3,1}$=5.



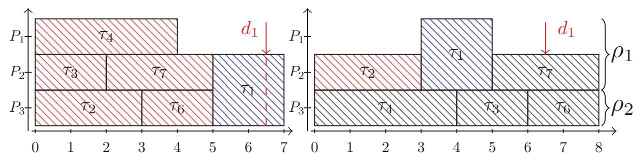
(a) Stationary gang (b) Strict partitioning

Fig. 4: Stationary scheduling better than strict partitioning.

C. Global and Partitioned for Non-preemptive Gangs

Non-preemptive scheduling of rigid gang tasks poses additional challenges. It has been demonstrated that global policies for rigid gangs can lead to long priority-inversion [33]. To mitigate this issue, static partitioning can be employed to separate the tasks that cause priority inversion.

Example IV.5. Consider seven tasks τ_{1-7} that execute non-preemptively on three identical processors. Task τ_1 must run on two processors simultaneously while all other tasks can run on any single processor. Tasks worst-case execution times are as follows: C_1=C_3=C_6=2, C_2=C_7=3, and C_4=4. A priority inversion is when a higher-priority job must wait for the completion of a lower-priority job. For example, all lower-priority tasks τ_{2-7} can be released one clock tick before τ_1. A global work-conserving scheduler always keeps the processors busy, and whenever a processor becomes idle, the next pending job that can run on idle processors is scheduled. While it results in good processor utilization, it can also lead to a long priority inversion. As shown in Figure 5a, when the start and end times of old and new tasks overlap, the new tasks with smaller volumes can get ahead of pending big-volume tasks (*i.e.,* τ_1 cannot start as long as two processors are not idle) and cause so-called *2D-blocking* [33].



(a) Global gang (b) Strict partitioning

Fig. 5: Blocking in non-preemptive global and strict gang.

Applying strict partitioning with a sequential scheduler on each partition makes it possible to find a compromise between parallelism and reduced blocking. Figure 5b shows a strict partitioning with two partitions: $\rho_1 = \{P_1, P_2\}$ and $\rho_2 = \{P_3\}$. While it leads to processor P_1 underutilization, the priority inversion experienced by τ_1 can be reduced to the duration of the longest lower-priority task in ρ_1.

V. Strict Partitioning Algorithms

A. *Offline Partitioning Heuristic*

The strict partitioning strategy requires making two offline decisions: *(i) how to divide processors into disjoint partitions* and *(ii) how to assign each gang task to one of the processor partitions.* It is shown in [41] that the task partitioning problem is NP-hard in the strong sense, even for sequential tasks. Hence, the complexity extends to gang tasks.

We propose an offline strict partitioning heuristic *First-fit Decreasing Volume (FFDV)* in Algorithm 1, inspired by the first-fit decreasing-height algorithm for the 2-D *strip packing* problem [74]. The heuristic first sorts the tasks by a non-increasing order of their volumes with ties broken by a non-decreasing order of task periods (line 2). Then, it assigns the sorted tasks one by one to form a sequence of partitions $\rho_1, \rho_2, ..., \rho_t$. Each task will be assigned to the first partition (*i.e.*, with the lowest index) such that it is schedulable together with other tasks already assigned to this partition (lines 4 to 6). If none of the partitions can accommodate this task, a new partition will be created with the same volume as the task, given that there are sufficient processors left (lines 7 to 9). The algorithm terminates if no remaining tasks are pending to be assigned (*i.e.*, return a success in line 12) or if there are no sufficient processors to create a new partition for the pending-to-be-assigned tasks (*i.e.*, return a failure in line 11).

We note that various schedulability tests can be used in line 4 to check the schedulability of the tasks assigned to each partition, depending on which online scheduler is used. We will present different variants combined with different types of online schedulers (*i.e.*, uniprocessor task schedulers and global gang task schedulers) used for each partition in Section V-B.

The time complexity of the proposed FFDV heuristic is analyzed as follows. There are two loops in Algorithm 1. The

Algorithm 1: First Fit Decreasing Volume (FFDV)

```
Input: $\tau$: a gang task set to be scheduled;
       $M$: number of available processors;
Output: $\rho, \tau(\rho)$: a set of partitions and the corresponding
        tasks assigned to each partition if schedulable;
        False otherwise.
1  $M' \leftarrow M, \rho \leftarrow \varnothing$;
2  for $\tau_i \in \tau$ (sorted by non-increasing order of volumes) do
3      for $\rho_j \in \rho$ do
4          if $\tau(\rho_j) \cup \{\tau_i\}$ is schedulable on $\rho_j$ then
5              $\tau(\rho_j) \leftarrow \tau(\rho_j) \cup \{\tau_i\}$;
6              Continue to the next iteration in the outer loop;
7      if $m_i \leq M'$ then
8          Create a partition $\rho_{new}$ with $\tau(\rho_{new}) = \{\tau_i\}$ and
           $|\rho_{new}| = m_i$;
9          $\rho \leftarrow \rho \cup \{\rho_{new}\}, M' \leftarrow M' - m_i$;
10     else
11         return False;
12 return $\rho$, $\tau(\rho) = \{\tau(\rho_j) \mid \rho_j \in \rho\}$;
```

outer loop iterates the task set, and the inner loop iterates the partitions that have been created and checks the schedulability. In the worst case, the numbers of tasks and partitions that need to be iterated are n and M, respectively, leading to an overall time complexity of $\mathcal{O}(nM\Omega)$, where $\mathcal{O}(\Omega)$ denotes the time complexity of the schedulability test used in line 4.

B. *Two Strict Partitioning Variants:* `SP-U` *and* `SP-G`

We propose two variants of the strict partitioning heuristic by considering two different types of online schedulers: `SP-U` standing for the strict partitioning with *uniprocessor online schedulers* and `SP-G` standing for the strict partitioning with *global gang online schedulers*. The advantage of a global gang scheduler lies in its full utilization of processors by allowing different jobs to execute in parallel. In contrast, classical uniprocessor schedulers may lead to processor underutilization, but they have lower runtime overhead and well-established exact schedulability tests. Both variants use FFDV as a framework to make offline partitioning decisions. For `SP-U`, we apply the FFDV heuristic as is by incorporating corresponding exact uniprocessor schedulability tests and prove its performance bounds in Section V-C. For `SP-G`, we propose modifications to the FFDV heuristic to improve further its performance in Section V-D.

C. *Performance Bounds for* `SP-U`

We prove three performance bounds for the FFDV heuristic when using uniprocessor online schedulers (*i.e.*, `SP-U`). The first bound (Theorem V.1) is a weighted utilization bound defined by a weighting function widely used in bin-packing literature [75]. The last two bounds (Theorem V.2 and V.3) are general utilization bounds inspired by the classical results in 2D strip packing literature [74]. In what follows, we denote by u_b the utilization bound of a uniprocessor task scheduler (*e.g.*, $u_b = 1$ for preemptive EDF [38]). Moreover, we denote by $FFDV(\tau)$ the number of processors used by the FFDV algorithm to schedule the task set τ, and the resulting partitions as $\rho_1, \rho_2, ..., \rho_t$ (*i.e.*, t is the total number of partitions). It suffices to prove the schedulability of τ under the FFDV algorithm on M processors by showing $FFDV(\tau) \leq M$.

Theorem V.1. *A gang task set τ is schedulable on M processors by* `SP-U` *if*

$$\sum_{\tau_i \in \tau} W(U_i) \leq (M - \overline{m}) \cdot u_b, \tag{1}$$

where the weighting function $W : [0, u_b] \to [0, \frac{5}{8}u_b]$ is defined as:

$$W(U_i) = \begin{cases} \frac{6}{5}U_i, & \text{if } 0 \leq U_i \leq \frac{1}{6}u_b; \\ \frac{9}{5}U_i - \frac{1}{10}, & \text{if } \frac{1}{6}u_b < U_i \leq \frac{1}{3}u_b; \\ \frac{6}{5}U_i + \frac{1}{10}, & \text{if } \frac{1}{3}u_b < U_i \leq \frac{1}{2}u_b; \\ \frac{6}{5}U_i + \frac{4}{10}, & \text{if } \frac{1}{2}u_b < U_i \leq u_b. \end{cases}$$

Proof: We prove the theorem by showing that $FFDV(\tau) \leq M$ given inequality (1). It is proved in Theorem 2 of [74] that:

$$W(U) = \sum_{\tau_i \in \tau} W(U_i) \geq (FFDV(\tau) - \overline{m}) \cdot u_b. \tag{2}$$

By combining (1) and (2), we have:

$$(FFDV(\tau) - \overline{m}) \cdot u_b \leq \sum_{\tau_i \in \tau} W(U_i) \leq (M - \overline{m}) \cdot u_b. \tag{3}$$

It follows that $FFDV(\tau) \leq M$. ■

Theorem V.2. *A gang task set τ is schedulable on M processors by* `SP-U` *if*

$$U \leq \frac{(M - \overline{m} + \underline{m}) \cdot u_b}{2}. \tag{4}$$

Proof: We denote the total utilization and sequential utilization of the tasks assigned to partition ρ_i as $U(\rho_i)$ and $u(\rho_i)$, respectively. We also denote the sequential utilization of the first task assigned to ρ_i as u_i^0. For each $i \geq 2$, the first task in ρ_i cannot fit into the previous partition ρ_{i-1}; therefore, we have $u(\rho_{i-1}) + u_i^0 > u_b$. Since the tasks in ρ_{i-1} have at least volume $|\rho_i|$, and the first task in ρ_i has volume $|\rho_i|$, we have $U(\rho_{i-1}) + U(\rho_i) \geq (u(\rho_{i-1}) + u_i^0)|\rho_i| > |\rho_i|u_b$. Therefore,

$$\begin{aligned} FFDV(\tau) = \sum_{i=1}^{t} |\rho_i| &< |\rho_1| + \frac{1}{u_b}\left(\sum_{i=1}^{t-1} U(\rho_i) + \sum_{i=2}^{t} U(\rho_i)\right) \\ &= \overline{m} + \frac{2}{u_b}\sum_{i=1}^{t} U(\rho_i) - \frac{1}{u_b}(U(\rho_1) + U(\rho_t)). \end{aligned}$$

Since $U(\rho_1) + U(\rho_t) > \underline{m} \cdot u_b$, it follows

$$FFDV(\tau) < \overline{m} + \frac{2}{u_b}\sum_{i=1}^{t} U(\rho_i) - \underline{m}.$$

Since all the tasks are successfully assigned to t partitions, we have $\sum_{i=1}^{t} U(\rho_i) = \sum_{\tau_i \in \tau} U_i = U$, thus $FFDV(\tau) < M$ follows given inequality (4). ■

Theorem V.3. *A gang task set τ is schedulable on M processors by* `SP-U` *if $\exists p \in \mathbb{Z}^+ \wedge p \geq 2$:*

$$u_i \leq \frac{u_b}{p}, \forall \tau_i \in \tau, \tag{5}$$

and

$$U \leq \frac{p}{p+1}(M - \overline{m})u_b. \tag{6}$$

Proof: We need to show $FFDV(\tau) \leq M$ given inequalities (5) and (6). By the definition of the FFDV algorithm, we know $\rho_1 = \overline{m}$, and thus:

$$FFDV(\tau) = \sum_{i=1}^{t} |\rho_{i+1}| + \overline{m}, \tag{7}$$

where $|\rho_{t+1}| = 0$. Therefore, it suffices to show that

$$\frac{p}{p+1}\sum_{i=1}^{t} |\rho_{i+1}|u_b \leq U, \tag{8}$$

because (6) and (8) $\implies \frac{p}{p+1}\sum_{i=1}^{t} |\rho_{i+1}|u_b \leq \frac{p}{p+1}(M - \overline{m})u_b \implies \sum_{i=1}^{t} |\rho_{i+1}| + \overline{m} = FFDV(\tau) \leq M$.

The complete proof of inequality (8) is in Appendix A. ■

D. *Modifications of FFDV for* `SP-G`

We propose two modifications to the FFDV algorithm presented in Algorithm 1 based on two observations.

Observation V.1. *Given a gang task set $\tau(\rho_j)$ assigned to a processor partition ρ_j, if*

$$\forall \tau_x, \tau_y \in \tau(\rho_j) : m_x + m_y > |\rho_j| \ (x \neq y), \tag{9}$$

and $\tau(\rho_j)$ is deemed unschedulable on ρ_j by an exact schedulability test for a uniprocessor task scheduler, $\tau(\rho_j)$ cannot be deemed schedulable on ρ_j by any schedulability tests for a global gang task scheduler of the same type[1].

The statement in Observation V.1 is true since if (9) holds, any two jobs in $\tau(\rho_j)$ cannot run in parallel due to an insufficient number of processors in ρ_j. Therefore, the global gang task online scheduler degrades to a uniprocessor task scheduler of the same type, and the exact analysis for the uniprocessor scheduler is thus an optimal schedulability test for such task sets.

Inspired by the above observation, we propose checking the condition in Inequality (9) before running the schedulability test. If the condition is satisfied (*i.e.*, no tasks can run in parallel), we will use an exact analysis for the uniprocessor task scheduler as our schedulability test. Otherwise, we will use the sufficient (but not necessary[2]) schedulability analysis for the global gang task scheduler. Specific procedures of this modification are presented in Algorithm 2, where `uniTest` and `globalTest` denote the exact analysis for uniprocessor

[1]For example, a global gang EDF scheduler is of the same type as a uniprocessor EDF scheduler.

[2]To the best of our knowledge, there has been no exact schedulability analysis for global gang task schedulers.

Algorithm 2: Schedulability Test Procedures for `SP-P` (to replace lines 4 to 6 of Algorithm 1)

```
1 if $\forall \tau_x, \tau_y \in \tau(\rho_j) \cup \{\tau_i\} : m_x + m_y > |\rho_j| \ (x \neq y)$ then
2     Test ← uniTest;
3 else
4     Test ← globalTest;
5 if $\tau(\rho_j) \cup \{\tau_i\}$ is schedulable on $\rho_j$ according to Test then
6     $\tau(\rho_j) \leftarrow \tau(\rho_j) \cup \{\tau_i\}$;
7     Continue to the next iteration in the outer loop;
```

Algorithm 3: Volume Increase Procedures for `SP-P` (to replace lines 10 to 11 of Algorithm 1)

```
1 else if $M' > 0$ and $\tau(\rho_j) \cup \{\tau_i\}$ is schedulable with volume
      $|\rho_j| + M'$ then
2     Increase $\rho_j$'s volume to $|\rho_j| + M'$;
3     $\tau(\rho_j) \leftarrow \tau(\rho_j) \cup \{\tau_i\}, M' \leftarrow 0$;
4     Continue to the next iteration in the outer loop;
5 else
6     return False;
```

scheduler and the sufficient analysis for global gang task scheduler, respectively.

The second observation is that we can leverage the unused processors to increase the volume of an existing partition when a task cannot be scheduled on any existing partitions and the remaining processors are insufficient to create a new partition with the same volume as the unschedulable task. Here, we propose to increase the volume of the last partition in the partition list since the task volumes on the last partition are the closest to those of the tasks that are yet to be assigned.

Observation V.2. *In `SP-G`, if a task τ_i cannot be assigned to any existing partitions $\rho_1, \rho_2, ..., \rho_t$ and the number of remaining processors M' satisfies $0 < M' < m_i$, there is a possibility for τ_i to be schedulable on the last partition ρ_t by increasing its volume by M'.*

Based on this observation, we introduce the volume increase procedures in Algorithm 3 to replace lines 10 to 11 of Algorithm 1 for improving the FFDV performance. We note that this modification is not necessarily applied to the `SP-U` variant since increasing the volume of existing partitions cannot make unschedulable tasks become schedulable due to the nature of uniprocessor task schedulers (*i.e.*, all the processors in a partition are regarded as a single resource).

VI. Performance Evaluation

This section evaluates the schedulability performance of the proposed strict partitioning strategy by comparing it with state-of-the-art schedulability analyses for gang task sets. We perform two sets of experiments. First, from Section VI-A to VI-C, we experiment with comprehensive synthetic task sets generated by standard task set generation tools. Second, in Section VI-D, we conduct a case study using Edge TPU benchmarks of representative DNNs widely used in computer vision applications. For synthetic task sets, we consider both preemptive and non-preemptive task sets with both FP and EDF scheduling policies. For the Edge TPU case study, neural network inference tasks are modeled as non-preemptive gangs due to the software and hardware characteristics of Edge TPU.

A. Synthetic Evaluation Settings

Task set generation. We generate synthetic task sets based on a standard task set generation tool `DRS` [76]. For a systematic evaluation, we vary four task set parameters: (i) number of processors $M \in \{8, 16\}$; (ii) number of tasks $n \in \{M, 2M\}$; (iii) task volume m_i range $\in$ $\{[1, \lceil 0.3M \rceil]$ (low volume), $[1, \lceil 0.6M \rceil]$ (medium volume), $[1, M)$ (large volume)$\}$; (iv) normalized task set utilization $U/M \in \{0.1, 0.2, ..., 1.0\}$.

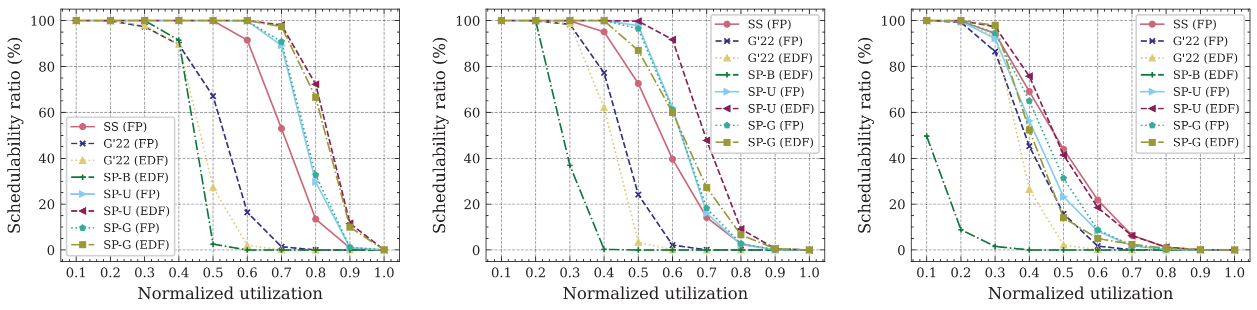
For each parameter combination, we generate 1,000 task sets. The generation of each task set includes the following procedures. First, we use `DRS` to generate task utilization U_i such that the sum equals the target task set utilization ($U = \sum_{\tau_i \in \tau} U_i$). Note that we set the maximum utilization for each task to be not greater than the task volume upper bound to avoid infeasible sequential task utilization (*i.e.*, $u_i > 1$). Second, we uniformly sample task periods T_i and volumes m_i from $[10, 1000]$ ms and the target volume range, respectively. Similarly, we set the minimum volume as the ceil of task utilization (*i.e.*, $\lceil U_i \rceil$) to avoid infeasible sequential utilization. Finally, we calculate the task WCETs by $C_i = \lfloor U_i \cdot T_i / m_i \rfloor$.

Comparison Algorithms. We compare the schedulability ratio of both preemptive and non-preemptive scheduling techniques on the same generated task sets. For preemptive scheduling, we consider both FP and EDF schedulers and compare the proposed strict partitioning with the following state-of-the-art schedulability analyses for gang tasks.

- `SS` (preemptive FP): stationary scheduling for preemptive FP in [19].
- `G'22` (preemptive FP/EDF): global schedulability analyses for preemptive FP/EDF in [13].
- `SP-B` (preemptive EDF): strict partitioning utilization bound for preemptive EDF (combination of Theorem V.1, V.2, and V.3, *i.e.*, a task set is deemed schedulable if any of the utilization bound conditions is satisfied).
- `SP-U` (preemptive FP/EDF): strict partitioning with uniprocessor online schedulers for preemptive FP/EDF. The exact preemptive FP response time analysis [77] and EDF utilization bound [38] are used for checking the schedulability.
- `SP-G` (preemptive FP/EDF): strict partitioning with global gang online schedulers for preemptive FP/EDF. We use `G'22` as the `globalTest` in Algorithm 2.

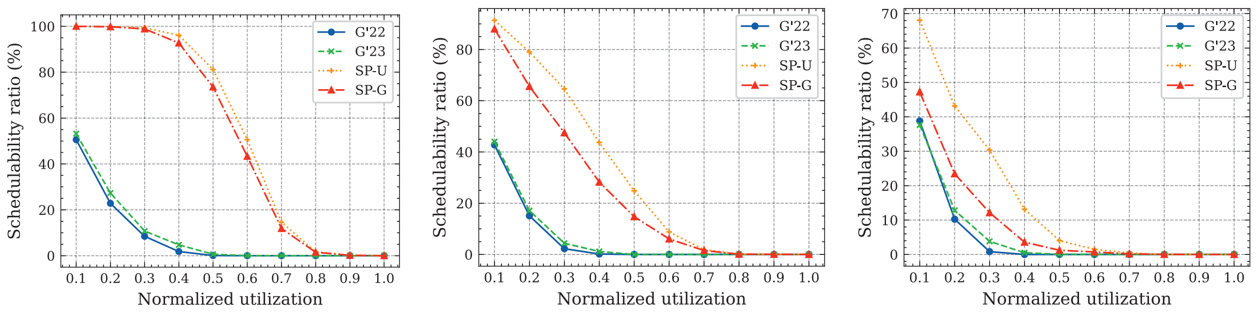
For non-preemptive scheduling, we focus our comparison on FP policies. The comparison algorithms include:

- `G'22` (non-preemptive FP): global schedulability analysis for non-preemptive FP in [10].
- `G'23` (non-preemptive FP): global schedulability analysis for non-preemptive FP in [12].
- `SP-U` (non-preemptive FP): strict partitioning with uniprocessor online schedulers for non-preemptive FP.



(a) Low volume level (b) Medium volume level (c) High volume level

Fig. 6: Schedulability ratio (*i.e.*, $\frac{\text{\#schedulable task sets}}{\text{\#total task sets}} \times 100$ %) of preemptive gang task sets.



(a) Low volume level (b) Medium volume level (c) High volume level

Fig. 7: Schedulability ratio (*i.e.*, $\frac{\text{\#schedulable task sets}}{\text{\#total task sets}} \times 100$ %) of non-preemptive gang task sets.

The exact non-preemptive FP response time analysis in [78] is used for checking the schedulability.

- `SP-G` (non-preemptive FP): strict partitioning with global gang online schedulers for non-preemptive FP. We use `G'23` as the `globalTest` in Algorithm 2.

For a fair comparison of all schedulability tests with FP policy, we apply DM [37] as the priority assignment algorithm. For each schedulability test, we calculate its schedulability ratio and present the results in Figure 6 and Figure 7 for the preemptive and the non-preemptive cases, respectively[3].

B. Evaluation Results of Preemptive Gang Scheduling

The evaluation results of preemptive rigid gang scheduling are shown in Figure 6, supporting the following observations.

Global scheduling vs partitioned scheduling. In all scenarios, partitioned scheduling (*i.e.*, `SS`, `SP-U`, and `SP-G`) achieved higher schedulability ratio than the state-of-the-art global schedulability analysis (*i.e.*, `G'22`). Specifically, for FP, the best performing partitioned scheduling method achieved up to 89.4%, 73.7%, and 28.2% more schedulable task sets than `G'22`, respectively. For EDF, the better performing strict partitioning method achieved up to 98.0%, 96.4%, and 49.3% more schedulable task sets than `G'22`, respectively. The results verify that the schedulability tests for partitioned scheduling are less pessimistic than the global analyses (as discussed in Section IV-A), and this advantage compensates for the loss due to processor under-utilization. Moreover, the utilization bound test `SP-B` performs better than `G'22` (EDF) for low-volume and low-utilization task sets. We note that the utilization bound test has linear time complexity.

Stationary scheduling vs strict partitioning. For all scenarios with *low or medium* task volume levels, all strict partitioning variants (*i.e.*, `SP-U` and `SP-G` with FP/EDF) achieved a higher schedulability ratio than stationary scheduling. This observation demonstrates the effectiveness of avoiding inter-partition interference in strict partitioning, as discussed in Section IV-B. For scenarios with *large* task volume levels, stationary scheduling achieved a higher schedulability ratio than the strict partitioning variants except for `SP-U` (EDF). This is expected since, for task sets with high-volume tasks, very few partitions can be created in strict partitioning. As a result, the interference between tasks cannot be effectively reduced by building boundaries through disjoint partitions.

`SP-U` vs `SP-G`. For scenarios with low task volume levels, `SP-U` and `SP-G` achieved similar schedulability ratios. This is expected since, for task sets with low task volume levels, there is a high chance that the tasks assigned to the same partition have a similar volume to the volume of the partition. As a result, tasks cannot execute in parallel on most partitions,

[3]The figures show the average schedulability ratio over different M and n.

and thus SP−G will use the same schedulability tests as SP−U according to Algorithm 2. For scenarios with high task volume levels, SP−G achieved a higher schedulability ratio than SP−U by up to 8.7% for the FP scheduling policy while a lower schedulability ratio by up to 27.4% for the EDF. This demonstrates that, compared to their corresponding uniprocessor exact analyses, there is less pessimism in global FP gang analysis than in global EDF.

FP vs EDF. The global gang schedulability tests (*i.e.*, G′22) achieved a higher schedulability ratio for FP than EDF, while SP−U achieved a higher schedulability ratio for EDF than FP. This observation reveals that the uniprocessor preemptive EDF test achieves higher schedulability than FP, which is a well-known result. However, the results of the corresponding global gang analyses are exactly the opposite. For SP−G, it achieved a higher schedulability ratio for EDF than FP with low-volume tasks, while a higher schedulability ratio for FP than EDF with high-volume tasks. This is reasonable since, for low-volume tasks, SP−G benefits from using uniprocessor exact tests on most partitions. In contrast, for high-volume tasks, SP−G uses global analysis more often since tasks are more likely to execute in parallel.

C. Evaluation Results of Non-preemptive Gang Scheduling

The schedulability results of non-preemptive gang scheduling are shown in Figure 7.

Global scheduling vs strict partitioning. In all scenarios, both strict partitioning variants (*i.e.*, SP−U and SP−G) achieved a higher schedulability ratio than global scheduling analyses. Specifically, for low-, medium-, and high-volume task sets, the bettering performing strict partitioning method achieved up to 91.4%, 62.0%, and 30.4%, respectively. This demonstrates the effectiveness of strict partitioning in reducing the pessimism of schedulability analyses (*e.g.*, 2D-blocking [33]) by assigning tasks with similar volumes to the same partitions and avoiding inter-partition interference, as discussed in Section IV-C.

SP−U vs SP−G. For scenarios with a low task volume level, SP−U and SP−G achieved similar schedulability ratios. For scenarios with a *medium or large* task volume level, SP−U achieved a higher schedulability ratio than SP−G, and the advantage increases with the task volume level. The first observation is explained in the same way as the preemptive case. The second observation reveals that the non-preemptive global gang analyses are so pessimistic that it is better to use the uniprocessor exact test even when tasks can run in parallel.

D. Edge TPU Case Study

We perform a case study based on the benchmarks of DNN workloads in computer vision applications on an AI hardware accelerator (*i.e.*, Edge TPU) to evaluate the performance of the proposed strict partitioning strategy on real systems.

1) Hardware and Software Specifications: In our case study, we use the COTS ASUS AI Accelerator cards CRL-G18U-P3D and CRL-G116U-P3D[4], integrated with 8 and 16

[4]https://iot.asus.com/products/AI-accelerator/AI-Accelerator-PCIe-Card/

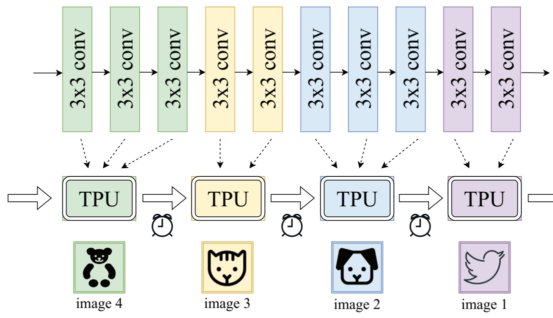


Fig. 8: Inllustration of Edge TPU pipelining.

Edge TPUs, respectively. Each Edge TPU has an on-chip scratchpad memory of 8 MB, which can be used for storing DNN weights. If the weights of a DNN model are larger than 8 MB, some weights cannot be fit into the on-chip memory and thus need to be loaded from the external memory (*i.e.*, DRAM) at runtime. This will incur a large runtime overhead (see Figure 1) and increase DRAM bandwidth contention. To avoid these issues, we can use the *Edge TPU Pipelining* technique. It splits a large DNN model into several consecutive segments and assigns each segment onto one Edge TPU such that the weights of each segment do not exceed the on-chip memory limit. This way, we can use multiple Edge TPU segments to form a pipeline. Figure 8 illustrates the Edge TPU pipelining technique, where four images are processed on four different Edge TPU segments in a pipeline, and we use different colors to show the mapping between DNN and Edge TPU segments.

There are two main advantages of using the Edge TPU pipelining technique. First, it avoids the aforementioned issues of loading weights from DRAM at runtime. Second, it enables pipeline parallelism since different inputs can be processed on different Edge TPU segments at the same time. The downside is the additional overhead caused by the communication time between two consecutive Edge TPUs. Therefore, small models benefit less from running on a large number of Edge TPUs than large models, considering the tradeoff between communication overhead and on-chip caching. Figure 9 shows the tradeoff by presenting the speedup factors of DNNs with different sizes running on Edge TPU pipelines with different volumes.

Once we decide the number of Edge TPUs to be used for a DNN model, we compile the DNN model into executable files using the Edge TPU Compiler[5] and execute the model using the libcoral runtime library[6]. The libcoral runtime spawns one thread for each Edge TPU segment and allocates the on-chip memory in all threads simultaneously. After the DNN inferences are finished and the results are retrieved, all the threads will be terminated at the same time. Since a DNN model occupies multiple Edge TPU segments, with all the threads starting and finishing synchronously, the DNN inference on Edge TPUs can be modeled as a gang task. Moreover,

[5]https://coral.ai/docs/edgetpu/compiler/
[6]https://coral.ai/docs/reference/cpp/

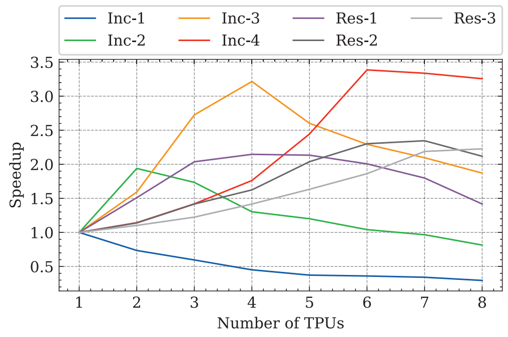


Fig. 9: Speedup factors of benchmarked DNN models w.r.t. different number of Edge TPUs on CRL-G18U-P3D.

TABLE I: Edge TPU benchmarks

Model	Inc-1	Inc-2	Inc-3	Inc-4	Res-1	Res-2	Res-3
Size (MB)	5.72	10.19	21.56	40.90	23.40	42.46	57.53
Volume	1	2	4	6	4	7	9
WOET (ms)	6	10	15	31	24	44	55

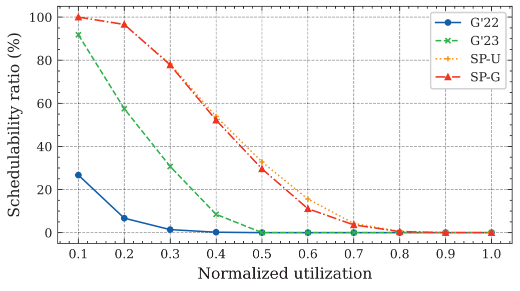
the execution is non-preemptive since the `libcoral` runtime does not support preemption within a DNN inference.

2) Workloads: We benchmarked seven representative DNN models widely used in computer vision applications. For each DNN model, we compile it with all possible numbers of Edge TPUs on the card. For each Edge TPU number, we measure its corresponding worst observed execution time (*WOET*) by running the model 1,000 times with different input images. Then, we set the task volume as the Edge TPU number that leads to the minimum worst observed execution time. Table I summarizes the specifications of the benchmarked workloads.

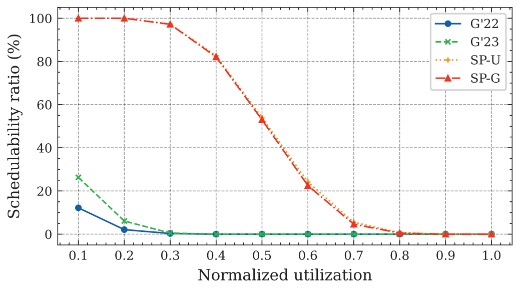
After we get the Edge TPU benchmarks, we generate two test suites for evaluation. The first test suite consists of the first six DNNs shown in Table I and considers CRL-G18U-P3D with 8 Edge TPUs as the hardware platform, while the second suite contains all the benchmarked DNNs and uses CRL-G116U-P3D with 16 Edge TPUs since the last DNN ResNet-152 requires a volume of 9. For each test suite, we vary the normalized task set utilization from 0.1 to 1.0 with a step of 0.1, and generate random task utilization for each DNN workload using `DRS`. The task periods are calculated by $T_i = C_i \cdot m_i / U_i$ accordingly.

We generate 1,000 task sets for each target normalized task set utilization and compare the schedulability ratio of different schedulability tests. Since the DNN tasks are modeled as non-preemptive gang tasks, only non-preemptive gang schedulability tests are used for evaluation.

3) Evaluation Results: Figure 10 depicts the schedulability ratio of different schedulability tests. The observations are similar to those of the synthetic non-preemptive task sets shown in Figure 7, *i.e.*, the strict partitioning strategy outperforms global non-preemptive gang analyses, and `SP-U` achieves similar schedulability ratio to `SP-G` for relatively low volume levels (Figure 10b). It demonstrates the effectiveness of the proposed strict partitioning strategy on real systems and the potential of using a classical uniprocessor scheduler for each partition to achieve low overhead and high schedulability performance.



(a) $n = 6, M = 8$



(b) $n = 7, M = 16$

Fig. 10: Schedulability ratio (*i.e.*, $\frac{\text{\#schedulable task sets}}{\text{\#total task sets}} \times 100$ %) of non-preemptive gang DNN task sets on Edge TPU pipelines.

VII. Conclusion

This paper proposed a new partitioned scheduling strategy, named *strict partitioning*, for gang tasks. Its main idea is to divide tasks and processors into disjoint subsets and assign tasks statically onto processor partitions. Strict partitioning has four main advantages. (i) It builds boundaries between different partitions so there is no inter-partition interference. (ii) It tries to assign tasks with similar volumes to the same partition so that the intra-partition interference is reduced. (iii) The tasks within each partition can be scheduled by any type of scheduler, which allows the use of a tighter schedulability test. (iv) The tasks are statically assigned to processor partitions so that there is less overhead caused by task migration. Experiments on synthetic task sets and a case study based on Edge TPU benchmarks demonstrated the effectiveness of the proposed strategy and algorithms by comparing them with state-of-the-art gang scheduling techniques.

In future works, we plan to study the hybrid of strict partitioning and global sequential task scheduling [71], [72], [79] for rigid gang tasks. Semi-partitioned gang scheduling is also an interesting strategy to research.

Appendix A
Proof of Inequality (8) in Theorem V.3

Let us define U_{ij} and u_{ij} as the total task utilization and sequential task utilization, respectively, of the tasks assigned to partition ρ_j when the last task is assigned in ρ_i before the next partition ρ_{i+1} is created. Moreover, we define u_i^0 as the sequential utilization of the first task assigned to partition ρ_i and $\delta_i = \max(0, p/(p+1)u_b - u_{ii})$. Before proving inequality (8), we first present the following lemmas that are essential to our proof.

Lemma A.1. *The following inequality holds.*

$$U_{ii} \geq u_{ii}|\rho_{i+1}| + u_i^0(|\rho_i| - |\rho_{i+1}|). \tag{10}$$

Proof: We denote by τ_i^0 the first task assigned to partition ρ_i and by $\tau(\rho_i)$ the set of tasks assigned to ρ_i when the last task is assigned to ρ_i before the next partition ρ_{i+1} is created. By the definition of U_{ii}, the LHS of (10) is the total utilization of $\tau(\rho_i)$. The RHS of (10) $= u_i^0|\rho_i| + (u_{ii} - u_i^0)|\rho_{i+1}|$. Since u_i^0 and $|\rho_i|$ are the sequential utilization and the volume of τ_i^0, respectively, $u_i^0|\rho_i|$ is the utilization of τ_i^0 (denoted as $U(\tau_i^0)$). Moreover, by the definition of u_{ii}, $(u_{ii} - u_i^0)$ is the total sequential utilization of $\tau(\rho_i) \setminus \{\tau_i^0\}$. Since the tasks are assigned in the decreasing order of volumes, the volumes of the tasks in $\tau(\rho_i)\setminus\{\tau_i^0\}$ are not smaller than $|\rho_{i+1}|$. Therefore, the total utilization of $\tau(\rho_i)\setminus\{\tau_i^0\}$ (denoted as $U(\tau(\rho_i)\setminus\{\tau_i^0\})$) is not smaller than $(u_{ii} - u_i^0)|\rho_{i+1}|$. Thus, $U_{ii} = U(\tau_i^0) + U(\tau(\rho_i) \setminus \{\tau_i^0\}) \geq u_i^0|\rho_i| + (u_{ii} - u_i^0)|\rho_{i+1}|$. ■

Lemma A.2. *The following inequality holds.*

$$u_{ii} \geq \left(\frac{p}{p+1}u_b - \delta_i\right). \tag{11}$$

Proof: For (11), there are two cases to consider: $\delta_i = 0$ and $\delta_i > 0$. If $\delta_i = 0$, by the definition of δ_i, we know that $u_{ii} \geq p/(p+1)u_b$. Thus, (11) holds. If $\delta_i > 0$, we know $\delta_i = p/(p+1)u_b - u_{ii}$, (11) still holds. ■

Lemma A.3. *The following inequality holds.*

$$U_{ii} \geq \left(\frac{p}{p+1}u_b - \delta_i\right)|\rho_{i+1}|. \tag{12}$$

Proof: Take (11) into (10), we have $U_{ii} \geq \left(\frac{p}{p+1}u_b - \delta_i\right)|\rho_{i+1}| + u_i^0(|\rho_i| - |\rho_{i+1}|)$. Since $|\rho_i| \geq |\rho_{i+1}|$ and $u_i^0 \geq 0$, $U_{ii} \geq \left(\frac{p}{p+1}u_b - \delta_i\right)|\rho_{i+1}|$. ■

Lemma A.4. *If $\delta_{i-1} > 0$, the following inequality holds.*

$$u_i^0 \geq \frac{1}{p+1}u_b + \delta_{i-1}. \tag{13}$$

Proof: Since $\delta_{i-1} > 0$, $\delta_{i-1} = p/(p+1)u_b - u_{i-1,i-1} > 0$. Moreover, since $u_i^0 \geq u_b - u_{i-1,i-1}$, (13) follows. ■

Now, we prove (8) based on Lemmas A.1-A.4.

Proof of Inequality (8) **in Theorem V.3.**

Since all the tasks in τ are successfully assigned by the FFDV algorithm to partitions $\rho_1, \rho_2, ..., \rho_t$, we know $U = \sum_{j=1}^{t} U_{tj}$. Replace U by $\sum_{j=1}^{t} U_{tj}$ in (8), it suffices to prove:

$$\sum_{i=1}^{t} U_{tj} \geq \frac{p}{p+1}\sum_{i=1}^{t}|\rho_{i+1}|u_b. \tag{14}$$

We prove (14) by induction. In what follows, we will prove

$$\forall i \in [1,t] : \sum_{j=1}^{i} U_{ij} \geq \frac{p}{p+1}\sum_{j=1}^{i}|\rho_{j+1}|u_b - \delta_i|\rho_{i+1}|, \tag{15}$$

which implies (14) by taking $i = t$.

For $i = 1$, (15) is reduced to

$$U_{11} \geq \left(\frac{p}{p+1}u_b - \delta_1\right)|\rho_2|,$$

which is proved by setting $i = 1$ in (12).

For $1 < i \leq t$, we prove (15) by supposing it holds for $i-1$. There are two cases to consider.

Case 1. $\delta_{i-1} = 0$. Since (15) is true for $i-1$, we have

$$\sum_{j=1}^{i-1} U_{i-1,j} \geq \frac{p}{p+1}\sum_{j=1}^{i-1}|\rho_{j+1}|u_b.$$

Since $\sum_{j=1}^{i-1} U_{ij} \geq \sum_{j=1}^{i-1} U_{i-1,j}$, we have

$$\sum_{j=1}^{i} U_{ij} = \sum_{j=1}^{i-1} U_{ij} + U_{ii} \geq \frac{p}{p+1}\sum_{j=1}^{i-1}|\rho_{j+1}|u_b + U_{ii}$$
$$= \frac{p}{p+1}\left(\sum_{j=1}^{i}|\rho_{j+1}| - |\rho_{i+1}|\right)u_b + U_{ii}.$$

Take (12) into the above inequality, (15) follows.

Case 2. $\delta_{i-1} > 0$. There are two subcases to consider.

Case 2.1. $\forall \tau_k \in \tau(\rho_i) : u_k \geq u_b/(p+1)$. By (5), there must be at least p tasks assigned to ρ_i. Additionally, by (13), the sequential utilization of the first task assigned to ρ_i is at least $u_b/(p+1) + \delta_{i-1}$. Therefore, we have

$$u_{ii} \geq \frac{p}{p+1}u_b + \delta_{i-1}.$$

Take the above inequality and (13) into (10), we have

$$U_{ii} \geq \left(\frac{p}{p+1}u_b + \delta_{i-1}\right)|\rho_{i+1}| + \left(\frac{u_b}{p+1} + \delta_{i-1}\right)(|\rho_i| - |\rho_{i+1}|)$$
$$\geq \frac{p}{p+1}|\rho_{i+1}|u_b + \delta_{i-1}|\rho_i|.$$

Combine this with (15) for $i-1$, we have

$$\sum_{j=1}^{i} U_{ij} \geq \sum_{j=1}^{i-1} U_{i-1,j} + U_{ii} \geq \frac{p}{p+1}|\rho_{i+1}|u_b + \delta_{i-1}|\rho_i|$$
$$+ \frac{p}{p+1}\sum_{j=1}^{i-1}|\rho_{j+1}|u_b - \delta_{i-1}|\rho_i|u_b = \frac{p}{p+1}\sum_{j=1}^{i}|\rho_{j+1}|u_b.$$

Therefore, (15) holds for Case 2.1.

Case 2.2. $\exists \tau_k \in \tau(\rho_i) : u_k < u_b/(p+1)$. This can happen only if $u_{i,i-1} + u_k > u_b$. By combining the above two inequalities, we have $u_{i,i-1} + u_b/(p+1) > u_b$, and thus

$$u_{i,i-1} > \frac{p}{p+1}u_b.$$

Combine it with $\delta_{i-1} = p/(p+1)u_b - u_{i-1,i-1}$, we have

$$u_{i,i-1} - u_{i-1,i-1} \geq \frac{p}{p+1}u_b - u_{i-1,i-1} = \delta_{i-1}.$$

Since $U_{i,i-1} - U_{i-1,i-1} \geq (u_{i,i-1} - u_{i-1,i-1})|\rho_{i+1}|$, it follows

$$U_{i,i-1} - U_{i-1,i-1} \geq \delta_{i-1}|\rho_{i+1}|. \quad (16)$$

Take (11) and (13) into (10), we have

$$U_{ii} \geq \left(\frac{p}{p+1}u_b - \delta_i\right)|\rho_{i+1}| + \left(\frac{u_b}{p+1} + \delta_{i-1}\right)(|\rho_i| - |\rho_{i+1}|). \quad (17)$$

Combine (16) and (17) with (15) for $i-1$, we have

$$\sum_{j=1}^{i} U_{ij} = U_{ii} + \sum_{j=1}^{i-1} U_{i,j} = U_{ii} + U_{i,i-1} + \sum_{j=1}^{i-2} U_{i,j}$$

$$\geq U_{ii} + \sum_{j=1}^{i-1} U_{i-1,j} + (U_{i,i-1} - U_{i-1,i-1})$$

$$\geq \left(\frac{p}{p+1}u_b - \delta_i\right)|\rho_{i+1}| + \left(\frac{u_b}{p+1} + \delta_{i-1}\right)(|\rho_i| - |\rho_{i+1}|)$$

$$+ \frac{p}{p+1}\sum_{j=1}^{i-1} |\rho_{j+1}|u_b - \delta_{i-1}|\rho_i| + \delta_{i-1}|\rho_{i+1}|$$

$$\geq \frac{p}{p+1}\sum_{j=1}^{i} |\rho_{j+1}|u_b - \delta_i|\rho_{i+1}|.$$

So (15) holds for Case 2.2. This concludes that (14) follows.

Acknowledgement

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

References

[1] J. K. Ousterhout, "Scheduling techniques for concurrent systems," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, vol. 82, 1982, pp. 22–30.

[2] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "Improving parallel job scheduling by combining gang scheduling and backfilling techniques," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2000, pp. 133–142.

[3] I. Moschakis and H. Karatza, "Evaluation of gang scheduling performance and cost in a cloud computing system," *The Journal of Supercomputing*, vol. 59, pp. 975–992, 2010.

[4] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[5] Z. Dong, K. Yang, N. Fisher, and C. Liu, "Tardiness bounds for sporadic gang tasks under preemptive global EDF scheduling," *IEEE Transactions on Parallel Distributed Systems*, vol. 32, no. 12, pp. 2867–2879, 2021.

[6] J. Bian, A. A. Arafat, H. Xiong, J. Li, L. Li, H. Chen, J. Wang, D. Dou, and Z. Guo, "Machine learning in real-time internet of things (IoT) systems: A survey," *IEEE Internet of Things Journal*, vol. 9, no. 11, pp. 8364–8386, 2022.

[7] K. Seshadri, B. Akin, J. Laudon, R. Narayanaswami, and A. Yazdanbakhsh, "An evaluation of edge tpu accelerators for convolutional neural networks," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2022, pp. 79–91.

[8] M. Verucchi, G. Brilli, D. Sapienza, M. Verasani, M. Arena, F. Gatti, A. Capotondi, R. Cavicchioli, M. Bertogna, and M. Solieri, "A systematic assessment of embedded neural networks for object detection," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2020, pp. 937–944.

[9] Z. Dong and C. Liu, "A utilization-based test for non-preemptive gang tasks on multiprocessors," in *IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 105–117.

[10] S. Lee, N. Guan, and J. Lee, "Design and timing guarantee for non-preemptive gang scheduling," in *IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 132–144.

[11] E. Kim, J. Lee, L. He, Y. Lee, and K. G. Shin, "Offline guarantee and online management of power demand and supply in cyber-physical systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2016, pp. 89–98.

[12] B. Sun, T. Kloda, J. Chen, C. Lu, and M. Caccamo, "Schedulability analysis of non-preemptive sporadic gang tasks on hardware accelerators," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2023, pp. 147–160.

[13] S. Lee, S. Lee, and J. Lee, "Response time analysis for real-time global gang scheduling," in *IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 92–104.

[14] G. Nelissen, J. Marcè i Igual, and M. Nasri, "Response-Time Analysis for Non-Preemptive Periodic Moldable Gang Tasks," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2022, pp. 12:1–12:22.

[15] S. Kato and Y. Ishikawa, "Gang EDF scheduling of parallel task systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2009, pp. 459–468.

[16] J. Goossens and V. Berten, "Gang FTP scheduling of periodic and parallel rigid real-time tasks," *arXiv preprint arXiv:1006.2617*, 2010.

[17] J. Goossens and P. Richard, "Optimal scheduling of periodic gang tasks," *Leibniz Transactions on Embedded Systems*, vol. 3, no. 1, pp. 04:1–04:18, 2016.

[18] Z. Dong and C. Liu, "Analysis techniques for supporting hard real-time sporadic gang task systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2017, pp. 128–138.

[19] N. Ueter, M. Günzel, G. von der Brüggen, and J.-J. Chen, "Hard real-time stationary gang-scheduling," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2021, pp. 10:1–10:19.

[20] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers," in *IEEE Real-Time Systems Symposium (RTSS)*, 2010, pp. 14–24.

[21] J. Lelli, D. Faggioli, T. Cucinotta, and G. Lipari, "An experimental comparison of different real-time schedulers on multicore systems," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2405–2416, 2012, automated Software Evolution.

[22] B. B. Brandenburg and M. Gül, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," in *IEEE Real-Time Systems Symposium (RTSS)*, 2016, pp. 99–110.

[23] A. Biondi and G. Buttazzo, "Timing-aware FPGA partitioning for real-time applications under dynamic partial reconfiguration," in *NASA/ESA Conference on Adaptive Hardware and Systems*, 2017, pp. 172–179.

[24] C. Han, H. S. Chwa, K. Lee, and S. Oh, "SPET: Transparent sram allocation and model partitioning for real-time DNN tasks on edge TPU," in *ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[26] S. Christian, V. Vincent, S. Ioffe, S. Jon, and W. Zbigniew, "Rethinking the Inception architecture for computer vision," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826.

[27] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence (AAAI)*, 2017.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[29] Y. Sun and M. Di Natale, "Assessing the pessimism of current multicore global fixed-priority schedulability analysis," in *Annual ACM Symposium on Applied Computing*, 2018, pp. 575–583.

[30] T. P. Baker, "A comparison of global and partitioned EDF schedulability tests for multiprocessors," in *International Conference on Real-Time and Network Systems (RTNS)*, 2006, pp. 119–127.

[31] S. Lauzac, R. Melhem, and D. Mosse, "Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multi-

processor," in *EUROMICRO Workshop on Real-Time Systems*, 1998, pp. 188–195.

[32] S. Baruah and K. Pruhs, "Open problems in real-time scheduling," *Journal of Scheduling*, vol. 13, no. 1, pp. 577–582, 2010.

[33] Z. Dong and C. Liu, "Work-in-progress: Non-preemptive scheduling of sporadic gang tasks on multiprocessors," in *IEEE Real-Time Systems Symposium (RTSS)*, 2019, pp. 512–515.

[34] V. Sarkar, "Partitioning and scheduling parallel programs for execution on multiprocessors," 1 1987.

[35] D. S. Johnson, "Fast algorithms for bin packing," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 272–314, 1974.

[36] B. S. Baker, E. G. Coffman, Jr., and R. L. Rivest, "Orthogonal packings in two dimensions," *SIAM Journal on Computing*, vol. 9, no. 4, pp. 846–855, 1980.

[37] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," *IFAC Proceedings Volumes*, vol. 24, no. 2, pp. 127–132, 1991.

[38] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, jan 1973.

[39] S. Baruah, M. Bertogna, and G. Buttazzo, *Multiprocessor Scheduling for Real-Time Systems*. Springer Publishing Company, Incorporated, 2015.

[40] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, vol. 43, no. 4, 2011.

[41] P. Ekberg and S. Baruah, "Partitioned scheduling of recurrent real-time tasks," in *IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 356–367.

[42] A. Burchard, J. Liebeherr, Y. Oh, and S. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1429–1442, 1995.

[43] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Operations Research*, vol. 26, no. 1, p. 127–140, 1978.

[44] J. Lopez, M. Garcia, J. Diaz, and D. Garcia, "Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2000, pp. 25–33.

[45] J. M. López, J. L. Díaz, and D. F. García, "Utilization bounds for EDF scheduling on real-time multiprocessor systems," *Real-Time Systems*, vol. 28, pp. 39–68, 2004.

[46] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of sporadic task systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2005.

[47] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems," *IEEE Transactions on Computers*, vol. 55, no. 7, pp. 918–923, 2006.

[48] N. Fisher, S. Baruah, and T. Baker, "The partitioned scheduling of sporadic tasks according to static-priorities," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2006, pp. 10 pp.–127.

[49] S. Baruah and E. Bini, "Partitioned scheduling of sporadic task systems: an ILP-based approach," in *Conference on Design and Architectures for Signal and Image Processing*, 2008.

[50] W. Zheng, Q. Zhu, M. D. Natale, and A. S. Vincentelli, "Definition of task allocation and priority assignment in hard real-time distributed systems," in *IEEE International Real-Time Systems Symposium (RTSS)*, 2007, pp. 161–170.

[51] N. Fisher and S. Baruah, "The partitioned multiprocessor scheduling of non-preemptive sporadic task systems," in *International Conference on Real-Time and Network Systems (RTNS)*, 2006, pp. 99–108.

[52] B. Berna and I. Puaut, "PDPA: Period driven task and cache partitioning algorithm for multi-core systems," in *International Conference on Real-Time and Network Systems (RTNS)*, 2012, pp. 181–189.

[53] J. Mayank and A. Mondal, "Non-preemptive multiprocessor scheduling for periodic real-time tasks," in *7th International Symposium on Embedded Computing and System Design (ISED)*, 2017, pp. 1–6.

[54] A. Burns, R. Davis, P. Wang, and F. Zhang, "Partitioned EDF scheduling for multiprocessors using a C=D scheme," in *Proceedings of 18th International Conference on Real-Time and Network Systems (RTNS)*, 2010, pp. 169–178.

[55] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2006, pp. 322–334.

[56] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Partitioned fixed-priority scheduling of parallel tasks without preemptions," in *IEEE Real-Time Systems Symposium (RTSS)*, 2018, pp. 421–433.

[57] Y. Wu, W. Zhang, N. Guan, and Y. Ma, "TDTA: Topology-based real-time DAG task allocation on identical multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 11, pp. 2895–2909, 2023.

[58] H.-E. Zahaf, G. Lipari, S. Niar, and A. E. Hassan Benyamina, "Preemption-aware allocation, deadline assignment for conditional DAGs on partitioned EDF," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2020, pp. 1–10.

[59] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2014, pp. 85–96.

[60] S. Baruah, "The federated scheduling of constrained-deadline sporadic DAG task systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 1323–1328.

[61] X. Jiang, N. Guan, H. Liang, Y. Tang, L. Qiao, and Y. Wang, "Virtually-federated scheduling of parallel real-time tasks," in *IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 482–494.

[62] N. Ueter, G. von der Brüggen, J.-J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *IEEE Real-Time Systems Symposium (RTSS)*, 2018, pp. 482–494.

[63] X. Jiang, N. Guan, X. Long, and W. Yi, "Semi-federated scheduling of parallel real-time tasks on multiprocessors," in *IEEE Real-Time Systems Symposium (RTSS)*, 2017, pp. 80–91.

[64] M. Kubale, "The complexity of scheduling independent two-processor tasks on dedicated processors," *Information Processing Letters*, vol. 24, no. 3, pp. 141–147, 1987.

[65] S. Baruah, "Fairness in periodic real-time scheduling," in *IEEE Real-Time Systems Symposium (RTSS)*, 1995, pp. 200–209.

[66] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," in *Annual ACM Symposium on Theory of Computing (STOC)*, 1993, pp. 345–354.

[67] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," *Information Processing Letters*, vol. 106, no. 5, pp. 180–187, 2008.

[68] V. Berten, P. Courbin, and J. Goossens, "Gang fixed priority scheduling of periodic moldable real-time tasks," in *5th Junior Researcher Workshop on Real-Time Computing*, 2011, pp. 9–12.

[69] P. Richard, J. Goossens, and S. Kato, "Comments on" gang edf schedulability analysis"," *arXiv preprint arXiv:1705.05798*, 2017.

[70] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley, R. Rajkumar, D. Niz, and G. Brüggen, "Many suspensions, many problems: A review of self-suspending tasks in real-time systems," *Real-Time Systems*, vol. 55, no. 1, pp. 144–207, 2019.

[71] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *IEEE Real-Time Systems Symposium (RTSS)*, 2007, pp. 149–160.

[72] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 4, pp. 553–566, 2009.

[73] J. Kleinberg and E. Tardos, *Algorithm Design*. USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[74] E. G. Coffman, Jr, M. R. Garey, D. S. Johnson, and R. E. Tarjan, "Performance bounds for level-oriented two-dimensional packing algorithms," *SIAM Journal on Computing*, vol. 9, no. 4, pp. 808–826, 1980.

[75] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C.-C. Yao, "Resource constrained scheduling as generalized bin packing," *Journal of Combinatorial Theory, Series A*, vol. 21, no. 3, pp. 257–298, 1976.

[76] D. Griffin, I. Bate, and R. I. Davis, "Generating utilization vectors for the systematic evaluation of schedulability tests," in *IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 76–88.

[77] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority preemptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[78] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[79] T. P. Baker, "Multiprocessor EDF and deadline monotonic schedulability analysis," in *IEEE Real-Time Systems Symposium (RTSS)*, 2003, pp. 120–129.