# Optimizing Resource Allocation for DAGs of Gang Tasks on Heterogeneous Platforms

Veronica Rispo*, Geoffrey Nelissen†, Federico Aromolo*, Daniel Casini*, Alessandro Biondi*
* *Scuola Superiore Sant'Anna*, Pisa, Italy
† *Eindhoven University of Technology*, Eindhoven, the Netherlands

***Abstract*—Real-time applications are increasingly compute- and data-intensive, often incorporating machine learning and AI workloads. These applications typically execute on heterogeneous platforms that combine multicore processors with GPUs and, in some cases, additional hardware accelerators. While gang task models effectively capture the computation on GPUs and other accelerators, Directed Acyclic Graphs (DAGs) are well-suited to represent the dependencies among different application components running across the multicore platform and accelerators.**

**In this paper, we study DAGs of gang tasks scheduled under a federated approach on heterogeneous platforms. We introduce a method to optimize the allocation of cores and streaming multiprocessors (SMs) for each DAG. Our results show that the proposed method significantly reduces resource requirements per application compared to existing state-of-the-art approaches.**

***Index Terms*—federated scheduling, GPU, multicore, response-time analysis, schedulability, gang, DAG**

## I. Introduction

Modern real-time applications often involve compute- and data-intensive parallel tasks executed on heterogeneous platforms composed of CPUs and GPUs. The tasks executed across CPUs and GPUs often exhibit complex dependencies that dictate the order in which they must execute. These *precedence constraints* can be effectively modeled using Directed Acyclic Graphs (DAGs), where nodes represent tasks and edges capture the precedence relations. Analyzing the timing properties of parallel DAG tasks distributed across different computing elements (the CPUs and GPUs) is known to be a complex problem [1], [2]. To add to that complexity, GPUs execute tasks by dividing them into blocks of threads, where each block requires access to a predefined number of cores before execution can begin. The number of GPU cores required by a block is task-specific and determined by the application programmer. This execution paradigm is commonly referred to as *rigid-gang scheduling*. Gang scheduling has been shown to improve the efficiency and performance of parallel applications on both CPUs and GPUs by reducing communication and context-switch overheads [3] but increases the complexity of the tasks' worst-case response times analysis [4].

Several works have analyzed the gang paradigm under global scheduling [5]–[7] or partitioned scheduling [1], [2], [8] using variations of the classical closed-form response-time analysis (RTA) approach [9].

Federated scheduling [10], a scheduling paradigm in which "heavy" tasks are allocated to dedicated processors while "light" tasks share the remaining processors under a global scheduler, may be a way to reduce pessimism due to the reduced overhead and the absence of interference for heavy tasks. Sun et al. [8], [11] proposed a response-time analysis for strict partitioning, where disjoint partitions of rigid gang tasks and processors were created to avoid interference. In a similar direction, Ahmed et al. [4] proposed an RTA for federated scheduling of DAGs of gang tasks.

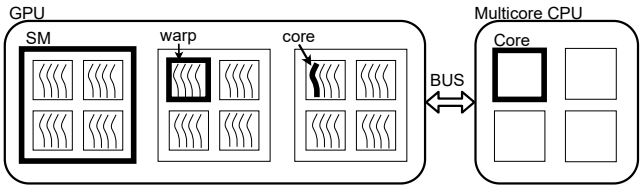



Fig. 1: Simplified representation of the system architecture.

Analyzing complex task models like DAGs or gang tasks using classical RTA may induce pessimistic results, as shown in [12]–[14]. The schedule abstraction graph (SAG) framework, proposed initially in [15] and extended in numerous works, is an alternative to the classic RTA and has been shown to be more accurate. However, no previous work leveraged it to analyze DAGs of gang tasks on heterogeneous platforms. Furthermore, methods to assign the resources (processors and SMs) to the DAGs so as to guarantee schedulability are still to be studied.

To fill this gap, this work proposes a method to optimize the allocation of resources for each DAG using the schedule abstraction framework. We first extend the analysis presented in [16] to support the analysis of DAGs of gang tasks on clustered systems. Then, we show how to use the schedule abstraction framework to optimize the number of cores and SMs assigned to each application. We show that despite the simplicity of the proposed method, the results obtained with respect to the state-of-the-art are significantly improved, increasing the number of task sets deemed schedulable by 84 percentage points when the CPU is used at 80% of its capacity. We also show that, for systems that were already deemed schedulable by the state-of-the-art method, our method reduces the number of SMs allocated to each application by 4.7 in average when the CPU is used at 70% of its capacity.

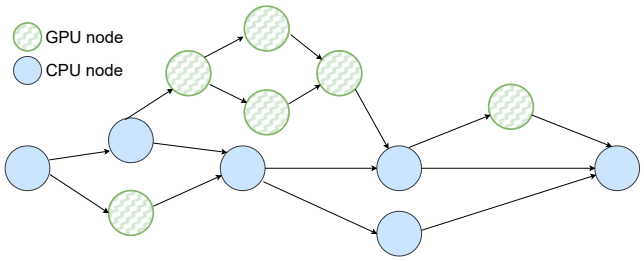



Fig. 2: Example of application represented as a DAG. Each node is a rigid-gang task that represents a function that must execute on CPU (solid blue circle) or a kernel block that must execute on GPU (striped green circle).

## II. System Model

This work considers a task set $\Gamma = \{G_1, G_2, \ldots, G_N\}$ consisting of $N$ applications. Each application $G_i$ is modeled as a DAG of gang tasks to be scheduled on a heterogeneous platform composed of a set $\mathcal{P}$ of CPU cores and one GPU, under non-preemptive federated scheduling.

A simplified representation of the computing platform is shown in Fig. 1. The main processing units of a GPU are the Streaming Processor (SP) cores (referred to as cores in Fig. 1), which are organized into warps (i.e., groups of cores, see Fig. 1). Warps are, in turn, organized into Streaming Multiprocessors (SMs). We denote the set of SMs as $\mathcal{G}$. Each SM in $\mathcal{G}$ has the same number of cores, and each core can execute one thread. The number of warps in an SM is denoted by $w$. Only one task can access the cores of a warp at any time (inter-task parallelism is not allowed on a warp).

A function that must be executed on the GPU is known as a kernel. A kernel is executed in parallel by multiple threads. The threads of a kernel are organized into blocks. All threads of a block must start their execution at the same time on the GPU. The size of a block (i.e., its number of threads) must be a multiple of the size of a warp. Blocks of different kernels can have different sizes.

We represent each application $G_i$ as a DAG of gang tasks released periodically with a period $T_i$, with a constrained deadline $D_i \leq T_i$. We assume that the scheduler prioritizes tasks of the same application $G_i$ arbitrarily but consistently (for example, using the task identifier). We denote the set of tasks of $G_i$ with higher priority than another task $\tau_j \in G_i$ as $\text{hp}_j$. An example of a DAG is shown in Fig. 2. Each DAG $G_i$ is represented by a tuple $(V_i, E_i)$ where $V_i$ is the set of nodes composing the DAG and $E_i$ is the set of edges connecting those nodes. Each node is a rigid-gang task $\tau_j$. A task $\tau_j$ represents either a function that must be executed on the CPU or a *kernel block* that must be executed on the GPU (see Fig. 2). The task $\tau_j$ is characterized by an affinity $\mathbf{a}_j \in \{cpu, gpu\}$ that encodes whether the node must execute on the CPU or GPU, a best-case execution time (BCET) $C_j^{min}$, a worst-case execution time (WCET) $C_j^{max}$, and a parallelism level $m_j$. The parallelism level represents the number of simultaneously available cores needed by a CPU task or the number of warps required by a kernel block. A directed edge between task $\tau_j$ to $\tau_k$ represents a precedence constraint between the two tasks. In case an edge exists between $\tau_j$ and $\tau_k$, we call $\tau_j$ an immediate predecessor of $\tau_k$, and $\tau_k$ an immediate successor of $\tau_j$. We denote the set of immediate predecessors of $\tau_j$ as $pred_j$.

In this paper, we consider federated scheduling [10]. Since applications access both the CPU and the GPU, they can never be transformed into a single equivalent sequential task. Therefore, we do not differentiate between "light" and "heavy" applications, and we assign an exclusive set of resources to each application. In particular, each application $G_i$ is assigned an exclusive set of $\mathcal{P}_{G_i} \subseteq \mathcal{P}$ CPUs and $\mathcal{G}_{G_i} \subseteq \mathcal{G}$ SMs to execute, such that $\bigcap_{i \neq j} \left(\mathcal{P}_{G_i} \cap \mathcal{P}_{G_j}\right) = \emptyset$ and $\bigcap_{i \neq j} \left(\mathcal{G}_{G_i} \cap \mathcal{G}_{G_j}\right) = \emptyset$.

Note that the model presented in this paper can be generalized to a heterogeneous platform composed of multiple hardware accelerators and CPUs.

## III. SAG for DAGs of Gang Tasks in Clustered Systems

In this work, we use the schedule-abstraction graph framework to analyze the response time of DAGs of gang tasks where each DAG executes on an exclusive set of CPUs and SMs. This can be seen as a clustered system in which the jobs of a DAG $G_i$ are executed on two clusters, one made of CPU cores and one made of warps.

Since each DAG is assigned exclusive clusters of cores on the CPU and GPU (i.e., we use federated scheduling), each DAG can be analyzed independently. Furthermore, since an application $G_i$ has a constrained deadline and all tasks of an application share that deadline, we only need to analyze a single job of each task in $G_i$ to obtain the application's worst-case response time, which is relative to the release of the whole DAG $G_i$. Therefore, in the rest of this paper, we use the terms job and task interchangeably.

The SAG framework analyzes the response time of an application by exploring all possible states in which the system may be when executing that application. It does so by building a graph $(\mathcal{S}, \mathcal{E})$ in which each node $s_k \in \mathcal{S}$ represents a system state, and each directed edge in $\mathcal{E}$ represents a transition from one state to another caused by the execution of a task. SAG-based solutions were proposed to analyze the execution of DAG tasks on clustered platforms and the analysis of independent gang tasks on multicore platforms in [16] and [13], respectively. However, no solution to analyze DAGs of rigid-gang tasks on clustered platforms has been proposed yet. We present such a solution by unifying the results of [13] and [16].

We represent a system state $s_k \in \mathcal{S}$ with the following information:

- The set of jobs that have already been dispatched until reaching system state $s_k$.
- For every dispatched job $J_j$ with a non-dispatched successor, a lower bound $EFT_j(s_k)$ and an upper bound $LFT_j(s_k)$ on $J_j$'s finish time in state $s_k$.

- The set of jobs that are certainly running on the CPU and the GPU in system state $s_k$.
- For every number of CPU cores $p$ (with $1 \leq p \leq |\mathcal{P}|$), a lower bound $A_p^{\min}(cpu, s_k)$ and an upper bound $A_p^{\max}(cpu, s_k)$ on when $p$ CPU cores become available to execute new jobs. That is, there are certainly less than $p$ cores free on the CPU before time $A_p^{\min}(cpu, s_k)$, and there are *certainly* at least $p$ cores free on the CPU at or after time $A_p^{\max}(cpu, s_k)$.
- A lower bound $A_p^{\min}(gpu, s_k)$ and an upper bound $A_p^{\max}(gpu, s_k)$ on when $p$ warps become available to execute new thread blocks on the GPU.

A job is said to be ready in state $s_k$ if all its predecessors have been dispatched. We denote the set of ready jobs on the CPU and the GPU as $\mathcal{R}(cpu, s_k)$ and $\mathcal{R}(gpu, s_k)$, respectively. For every ready job $J_j$ of a task $\tau_j$, a lower bound on its start time can be computed as follows:

$$EST(J_j, s_k) = \max\{A_{m_j}^{\min}(\mathbf{a}_j, s_k), \max_{\forall \tau_p \in pred_j}\{EFT_p(s_k)\}\},$$

where $EFT_p(s_k)$ is a lower bound on when the predecessor $\tau_p$ of $\tau_j$ finished executing, and $A_{m_j}^{\min}(\mathbf{a}_j, s_k)$ is a lower bound on when enough cores/warps become available to execute $\tau_j$.

We can also compute an upper bound on when one of the ready jobs will certainly start executing on the CPU as follows:

$$t_{wc}(cpu, s_k) = \min_{\forall \tau_r \in \mathcal{R}(cpu, s_k)}\{\max\{A_{m_r}^{\max}(cpu, s_k), \rho_r^{\max}(s_k)\}\},$$

where $\rho_r^{\max}(s_k)$ is an upper bound on the time at which all predecessors of $\tau_r$ finished executing in state $s_k$, i.e., $\rho_r^{\max}(s_k) = \max_{\forall \tau_p \in pred_j}\{LFT_p(s_k)\}$.

A similar upper bound for the GPU is given by:

$$t_{wc}(gpu, s_k) = \min_{\forall \tau_r \in \mathcal{R}(gpu, s_k)}\{\max\{A_{m_r}^{\max}(gpu, s_k), \rho_r^{\max}(s_k)\}\}.$$

Finally, an upper bound on when a job with higher priority than $J_j$ becomes ready on the same computing element (i.e., CPU or GPU) as $J_j$ is given by

$$t_{high}(J_j, s_k) = \min\{\rho_h^{\max}(s_k) \mid \mathbf{a}_h = \mathbf{a}_j \wedge \tau_h \in \text{hp}_j \cap \mathcal{R}(\mathbf{a}_j, s_k)\}.$$

A ready job $J_j$ may be the next job dispatched by the scheduler only if it can start before any other job is dispatched or any higher-priority job becomes ready. Thus, we have that $\tau_j$ may generate a transition from $s_k$ to a new state $s'_k$ only if $EST(J_j, s_k) \leq t_{wc}(cpu, s_k)$ and $EST(J_j, s_k) \leq t_{wc}(gpu, s_k)$ and $EST(J_j, s_k) < t_{high}(J_j, s_k)$. For each ready job $J_i$ that can generate a transition from a state $s_k$ to a new state $s'_k$, the new state $s'_k$ can be computed using the method shown in [13], [14].

The SAG recursively explores all possible state transitions starting from an initial state where no job has been dispatched yet. Details on how the exploration of all reachable system states can be done efficiently and on how to avoid exploring redundant states using the notion of independent scheduling decisions can be found in [16]. An implementation of the schedule abstraction framework for the analysis of DAGs of gang tasks executed on heterogeneous platforms can be found at https://github.com/SAG-org/sag-clustered.

## IV. Optimizing the Allocation of Computational Resources

The goal of the proposed method is to allocate the system's computational resources efficiently, ensuring that the task set remains schedulable while minimizing the resources dedicated to each individual DAG. To this end, we determine and optimize the number of exclusive CPUs $|\mathcal{P}_{G_i}|$ and exclusive SMs $|\mathcal{G}_{G_i}|$ assigned to each DAG $G_i$.

The method is described in the following steps:

1) For each DAG $G_i \in \Gamma$ we start assigning $|\mathcal{P}_{G_i}| = |\mathcal{P}|$ and $|\mathcal{G}_{G_i}| = |\mathcal{G}|$;
2) We analyze that DAG $G_i$ using the SAG analysis presented in Section III. We pass the DAG $G_i$, the number of CPU cores $|\mathcal{P}_{G_i}|$, and the number of warps $\mathcal{W} = |\mathcal{G}_{G_i}|w$ assigned to the DAG as input to the analysis. If the DAG is deemed schedulable on $|\mathcal{P}_{G_i}|$ cores and $\mathcal{W}$ warps, we incrementally reduce the number of SMs $|\mathcal{G}_{G_i}|$ assigned to $G_i$ by steps of 1 and repeat the analysis at each step until the DAG becomes unschedulable or when we have reached the minimum number of warps needed by any task $\tau_j \in V_i$ that needs to be executed on the GPU, i.e, when $\mathcal{W} = mw$, where $m = \min_{\tau_j | \mathbf{a_j} = gpu \wedge \tau_j \in G_i} m_j$.
3) Once the minimum number of SMs is found for $G_i$, we similarly reduce the number of CPU cores $|\mathcal{P}_{G_i}|$ assigned to $G_i$ by steps of 1. We analyze if the DAG $G_i$ remains schedulable using the SAG and we stop when the DAG becomes unschedulable or when we have reached the minimum number of CPUs needed by any task $\tau_j \in V_i$ that needs to be executed on the CPUs, i.e, when $|\mathcal{P}_{G_i}| = m$, where $m = \min_{\tau_j | \mathbf{a_j} = cpu \wedge \tau_j \in G_i} m_j$.
4) Finally, if $\sum_{G_i \in \Gamma} |\mathcal{P}_{G_i}| \leq |\mathcal{P}|$ and if $\sum_{G_i \in \Gamma} |\mathcal{G}_{G_i}| \leq |\mathcal{G}|$, then the entire task set is schedulable with the computed amount of resources allocated to each DAG; otherwise, the task set is deemed unschedulable.

Despite the apparent simplicity of the proposed method, we show in the next section that it yields significantly better results than the state of the art. The runtime of the overall method is also relatively low (less than a minute) because only a single application is analyzed at a time.

## V. Evaluation

In this section, we present the results of the experiments performed to evaluate our method. First, we compare the schedulability ratio of our method with respect to the solution proposed in [4]. Then, for task sets deemed schedulable by both, we compare the number of SMs needed with each.

### A. Experimental Setup

We generate DAGs similarly to the method proposed by Ahmed et al. [4] for their experiment on Multicore+GPU with multi-DAG systems (Section VII.B in [4]). In particular, we considered platforms consisting of 8 processors ($|\mathcal{P}|$ = 8) and 16 SMs ($|\mathcal{G}|$ = 16) with a total number of 256 warps. DAGs were generated with a number of nodes in $[20, 120]$. For each pair of nodes in a DAG, an edge was generated with

probability 0.6. We considered a high degree of parallelism, for which $m_i$ values were in $[1, 0.7 \cdot |\mathcal{P}|]$. The WCET $C_i^{max}$ was chosen from $[50, 100]$, while the BCET $C_i^{min}$ was set to 50% of $C_i^{max}$. Initially, DAGs with only CPU nodes with $m_i = 1$ are generated. DAGs are generated until a desired normalized utilization $U_d = U/|\mathcal{P}|$, where $U = \sum_{G_i \in \Gamma} \sum_{\tau_j \in G_i} C_j^{max}/T_i$, i.e., the sum of all DAG utilizations over the processor count, is reached. Then, for each DAG $G_i$, some CPU nodes with a heavy ratio in $[50, 80]\%$ are selected as GPU-accessing tasks. The maximum GPU-access length was chosen in $[0.1 \cdot C_i^{max}, 0.7 \cdot C_i^{max}]$. Each GPU-accessing node was split into two CPU nodes and multiple GPU nodes. For each GPU node, the number of warps ($m_i$ value) is selected from the set $\{8, 4, 2, 1\}$. Finally, edges were added from one CPU node to the GPU nodes to another CPU node to model a CPU task that submits a kernel to the GPU and waits for kernel completion.

The normalized utilizations vary from 0.1 to 1 with a step size of 0.1. For each utilization value, we generate 100 task sets. All experiments were executed on a system featuring an Intel Core i7-13700 CPU (2.1 GHz, 16 physical cores, 24 logical processors) and 16 GB of RAM.

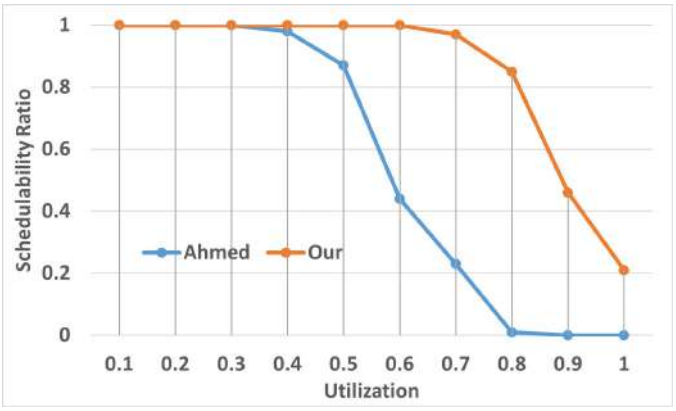
The analysis in Ahmed et al. [4] only returned whether a task set was deemed schedulable or not. Since our goal is to minimize the resources assigned to each DAG, we extended the ILP proposed in [4] with an objective function that minimizes the number of SMs allocated to each DAG. This enables a fair comparison between their results and ours.

### B. Experiments

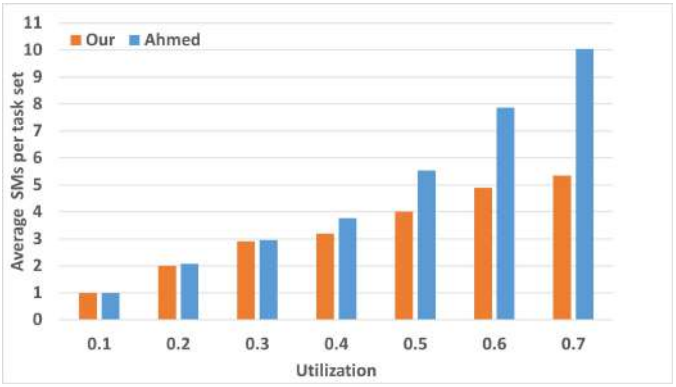
**Schedulability Ratio.** This experiment compares the schedulability ratio, i.e., the ratio of schedulable task sets over the number of analyzed task sets, obtained using our method (**Our**) and the one obtained by [4] (**Ahmed**).

The results of this experiment are shown in Fig. 3a. Our approach performs significantly better compared to [4], showing an improvement of 84 percentage points in schedulability at 80% system utilization. This is due to the fact that the SAG analysis introduces less pessimism, i.e., is more accurate, when analyzing the task set in comparison to closed-form equations. Additionally, we observed that for task sets deemed unschedulable by our method, the reason was that the number of CPU cores assigned to the applications exceeded the number of available CPU cores. We believe that for those task sets, it would be beneficial to try to reduce first the number of CPUs and then the number of SMs, i.e., inverting steps 2 and 3 in Section IV.

**SMs Allocations Optimization.** This experiment compares the average amount of SMs allocated in total, for each task set deemed schedulable by both methods, using our method (**Our**) and [4] (**Ahmed**). We show only the utilization percentage where there are at least 20 data points; otherwise, it is not significant. The results of this experiment are shown in Fig. 3b. The difference between the allocated SMs becomes larger when system utilization increases, showing the efficiency of our method in optimizing the allocated resources. The absolute number of allocated SMs reduced by 4.7 in average at 70% of system utilization.




(a)




(b)

Fig. 3: Comparison in terms of schedulability ratio (Fig. 3a) and optimization of the SMs resource allocated (Fig. 3b) between our method and Ahmed et al. [4].

## VI. Conclusions

In this work, we considered DAGs of gang tasks scheduled under a federated approach on heterogeneous platforms. We presented a method to allocate the system's computational resources in heterogeneous systems efficiently. Moreover, we presented an extension for DAGs of gang tasks of the SAG for clustered systems [16]. We have shown the effectiveness and utility of our method compared to the state-of-the-art by evaluating both schedulability and resource requirements. In future work, we will investigate how SAG in federated scheduling works for a heterogeneous platform composed of multiple hardware accelerators and CPUs, and, in the case where multiple implementations are available, how to decide on which hardware accelerators a task should execute. Additionally, we will compare federated with non-federated scheduling, where multiple applications can access the same computational resources.

**Acknowledgments.** We would like to thank the authors of [4] for sharing their tools. This was instrumental in performing the experiments presented in this paper.

## REFERENCES


[1] V. Rispo, F. Aromolo, D. Casini, and A. Biondi, "Response-time analysis of bundled gang tasks under partitioned fp scheduling," *IEEE Transactions on Computers*, 2024.

[2] N. Ueter, M. Günzel, G. von der Brüggen, and J.-J. Chen, "Hard real-time stationary gang-scheduling," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021, pp. 10–1.

[3] M. A. Jette, "Performance characteristics of gang scheduling in multiprogrammed environments," in *Proceedings of the 1997 ACM/IEEE conference on Supercomputing*, 1997, pp. 1–12.

[4] S. Ahmed, D. Massey, and J. H. Anderson, "Scheduling processing graphs of gang tasks on heterogeneous platforms," in *2025 IEEE 31st Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2025, pp. 362–374.

[5] B. Sun, T. Kloda, J. Chen, C. Lu, and M. Caccamo, "Response time analysis and optimal priority assignment for global non-preemptive fixed-priority rigid gang scheduling," *IEEE Transactions on Parallel and Distributed Systems*, 2025.

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

[7] S. Wasly and R. Pellizzoni, "Bundled scheduling of parallel real-time tasks," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 130–142.

[8] B. Sun, T. Kloda, C.-g. Wu, and M. Caccamo, "Partitioned scheduling and parallelism assignment for real-time dnn inference tasks on multi-tpu," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[9] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[10] 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 *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 2014, pp. 85–96.

[11] B. Sun, T. Kloda, and M. Caccamo, "Strict partitioning for sporadic rigid gang tasks," *arXiv preprint arXiv:2403.10726*, 2024.

[12] M. Nasri, G. Nelissen, and B. B. Brandenburg, "Response-time analysis of limited-preemptive parallel dag tasks under global scheduling," in *31st Conference on Real-Time Systems*, 2019, pp. 21–1.

[13] G. Nelissen, J. M. i Igual, and M. Nasri, "Response-time analysis for non-preemptive periodic moldable gang tasks," in *34th Euromicro Conference on Real-Time Systems, ECRTS 2022*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022, pp. 12–1.

[14] S. Srinivasan, M. Günzel, and G. Nelissen, "Response-time analysis for limited-preemptive self-suspending and event-driven delay-induced tasks," in *2024 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2024, pp. 29–42.

[15] M. Nasri and B. B. Brandenburg, "An exact and sustainable analysis of non-preemptive scheduling," in *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2017, pp. 12–23.

[16] G. Nelissen, "Work-in-progress: Response-time analysis of partitioned and clustered systems with the schedule-abstraction framework," in *2024 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2024, pp. 459–462.