Virtual Gang Scheduling of Parallel Real-Time Tasks

Waqar Ali
University of Kansas, Lawrence, USA
wali@ku.edu

Rodolfo Pellizzoni
University of Waterloo, Ontario, CA
rpellizz@uwaterloo.ca

Heechul Yun
University of Kansas, Lawrence, USA
heechul.yun@ku.edu

***Abstract*—We consider the problem of executing parallel real-time tasks according to gang scheduling on a multicore system in the presence of shared resource interference. Specifically, we consider sets of gang-tasks with precedence constraints in the form of a DAG. We introduce the novel concept of a virtual gang: a group of parallel tasks that are scheduled together as a single entity. Employing virtual gangs allows us to tightly bound the effect of shared resource interference. It also transforms the original, complex scheduling problem into a form that can be easily implemented and is amenable to exact schedulability analysis, further reducing pessimism. We present and evaluate both optimal and heuristic methods for forming virtual gangs based on a known interference model and while respecting all precedence constraints among tasks. When precedence constraints are not considered, we also compare our approach against existing response-time analysis for globally scheduled gang-tasks, as well as general parallel tasks. The results show that our approach significantly outperforms state-of-the-art multicore schedulability analyses when shared-resource interference is considered. Even in the absence of interference, it performs better than the state-of-the-art for highly parallel tasksets.**

***Index Terms*—real-time, gang scheduling, precedence constraints, safety critical, parallel tasks**

I. Introduction

High-performance multicore embedded computing platforms are increasingly being used in safety-critical real-time applications, such as avionics, robotics and autonomous vehicles. However, such use brings significant challenges due to the difficulties in ensuring predictable timing on these platforms.

In a multicore platform, tasks running concurrently can experience high timing variations due to the shared hardware resource contention. The effect of contention highly depends on the underlying hardware architecture which can often show extremely poor worst-case behaviors [1]. Furthermore, which tasks are co-scheduled at a time instant depends on the OS scheduler's decision and can vary over time. In other words, timing of a task is *coupled* with the rest of the tasks, the OS scheduling policy, and the underlying hardware. For this reason, in the use-cases where hard real-time guarantees are a must, such as avionics, it is recommended to disable all but one core of a multicore processor [2].

Gang scheduling was originally proposed in high-performance computing to maximize performance of parallel tasks [3], and many real-time varieties have since been studied [4]–[8]. In gang scheduling, threads of a parallel task are scheduled only when there are enough cores available to schedule all of them simultaneously. Therefore, gang scheduling reduces scheduling induced timing variations and synchronization overhead [7]. However, most prior gang scheduling studies do not consider the co-runner dependent timing variations due to contention in the shared hardware resources, and instead simply assume that WCETs already account for such effects, which may introduce severe pessimism in their analysis [9].

This research is supported by NSF CNS 1718880, CNS 1815959, and NSA Science of Security initiative contract #H98230-18-D-0009.

To address this problem, a more restrictive gang scheduling policy was recently proposed [10], which simply schedules one real-time gang task at a time; any remaining cores are allowed to schedule best-effort tasks but their shared resource usages are regulated so that their impact to the real-time gang task can be strictly bounded. From the analysis point of view, this approach effectively transforms the problem of parallel real-time task scheduling into a well understood unicore scheduling problem, which greatly simplifies analysis and reduces pessimism. However, the downside is that it can also greatly reduce real-time task schedulability as many cores can be left idling even when there are other real-time tasks ready to be scheduled.

In this paper, we propose a new gang scheduling approach, which is based on the novel concept of a *virtual gang*: a group of gang tasks that are scheduled together as a single entity. In our approach, parallel real-time tasks are grouped into a set of virtual gangs, which are then scheduled one virtual gang at a time using a gang scheduler as in [10]. All member tasks of a virtual gang share the same period and are synchronously released. Employing virtual gangs allows us to tightly bound the effect of shared resource interference while minimizing resource under utilization because the members of a virtual gang are determined a priori and do not change at run-time. Furthermore, it allows us to use the same unicore schedulability analysis [11], further reducing analysis pessimism. In forming virtual gangs, we consider precedence constraints among the gang tasks, which are expressed in the form of DAGs. This allows us to handle complex inter-task dependencies that are commonly observed in real world applications [12], [13].

We present and evaluate both optimal and heuristic methods for forming virtual gangs, which utilize an interference model and respect the given precedence constraints. When precedence constraints are not considered, we also compare our approach against existing response-time analysis for globally scheduled gang-tasks, as well as general parallel tasks. The results show that our approach significantly outperforms state-of-the-art multicore schedulability analyses when shared-resource interference is considered. Even in the absence of interference, it performs better than the state-of-the-art for highly parallel tasksets. To the best of our knowledge, this is the first work which enables schedulability analysis of real-time gang tasks with precedence constraints.

The rest of the paper is organized as follows. We discuss related work in Section II. We then define our system model in Section III. We present the proposed virtual gang formation algorithms and their schedulability analysis results in Section IV and in Section V, respectively. We conclude in Section VI.

II. Related Work

Parallel real-time tasks are typically modeled as following: Fork-join model [14], DAG model [15] and gang task model [4]–[6]. In the fork-join model, a task alternates between parallel (fork) and sequential (join) phases over time. In the DAG model, a task is represented as a directed acyclic graph with a set of associated precedence constraints, which allows more flexible scheduling as long as the constraints are satisfied. In both models, threads of a parallel task are scheduled independently. Lastly, in the gang task model, a task is characterized by a number of threads, which are scheduled simultaneously only if there are enough available cores. In literature, the gang task model is further distinguished by rigid, moldable, and malleable varieties depending on whether the number of threads of a gang task can vary over time [5], [6]. More recently, the bundled gang model [7] was proposed, which extends the rigid gang task model to support gang scheduling of different number of threads in different phases, called bundles, which are sequentially executed within a job. In this work, we consider a more general gang task model, in which each task is a rigid gang, but precedence constraints between tasks with the same period can be expressed in the form of a DAG.

In scheduling gang tasks, both fixed-priority and dynamic priority real-time versions of gang scheduling algorithms, namely Gang FTP and Gang EDF, respectively, are studied and analyzed [5], [6], [8]. Gang FTP [5] schedules rigid and periodic real-time gang tasks as follows: At each scheduling event, it schedules the highest priority task τ_i on the first h_i available cores (if exist) among the ready tasks. The process repeats for the remainder of the active tasks on any available cores. Gang EDF [6] works similarly except a task's priority is determined dynamically at runtime. However, these prior methods do not consider interference caused by shared hardware resources in multicores as they allow any gang tasks to be co-scheduled as long as there are available cores. Consequently, a gang task may need to consider every possible interleaving of co-scheduled tasks to determine its WCET, which leads to high pessimism. In contrast, RT-Gang [10] implements a more restrictive form of fixed priority gang scheduling policy, which limits only one gang task to be scheduled at any time, to address this problem, but at the cost of significant utilization loss.

III. System Model

We consider a multicore processor based platform π, which contains m unit-speed CPU cores. We consider a system comprising a set Γ of n periodic, rigid gang real-time tasks with implicit deadlines: $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task $\tau_i = (c_i, h_i, r_i, T_i)$ is characterized by its WCET c_i in isolation, the number of cores $h_i \leq m$ needed to execute, the shared resource demand factor r_i in the range $[0, 1]$, and the period T_i. The system also comprises a set of k DAGs $\{G_1, G_2, ..., G_k\}$. Each DAG $G_i = (v_i, e_i, T_i)$ expresses a set of precedence constraints among tasks. The node set $v_i \subseteq \Gamma$ consists of a subset of the tasks in Γ; we assume that all tasks in v_i must have the same period T_i, and no task in Γ can belong to the node set of more than one DAG. The edge set $e_i : v_i \times v_i$ consists of ordered pairs of the form (τ_p, τ_q) describing the precedence constraints among the tasks in v_i: formally, this means that the j-th job of τ_p must finish before the j-th job of τ_q can start executing.

A. Virtual Gangs and Scheduler

We assume that the number of distinct periods q within Γ is small relative to n. In other words, multiple tasks may share the same period, which is common in practice (e.g., [13]). All tasks that share the same period T forms a candidate-set $\Delta_T = \{\forall \tau_i \in \Gamma \mid T_i = T\}$. A virtual gang w_l is a subset of the tasks within the candidate-set Δ_T that are statically grouped together as a schedulable unit. Each virtual gang $w_l = (C_l, H_l, R_l, T_l)$ is characterized by its WCET C_l, the core requirement H_l and the resource demand R_l; the latter two are equal to the sum of the respective parameters of all of its member tasks. For the virtual gangs of Δ_T, there must exist a linear ordering between them such that all precedence constraints $\{G_1, G_2, ..., G_k\}$ are satisfied. The virtual gangs of all candidate sets are scheduled according to a preemptive fixed-priority gang scheduling scheme, which schedules one virtual gang at a time on π, subject to the linear ordering of each candidate set. We require all tasks in a candidate-set to be synchronously released and all member tasks of a virtual gang to be scheduled in parallel under the gang scheduler.

B. Interference Model

As noted earlier, the member tasks of a virtual gang are scheduled simultaneously on π. As such, they can suffer from interference on shared hardware resources (e.g., cache, memory bandwidth). In general, it is difficult to precisely model the impact of interference on a COTS hardware platform. For analysis purpose, we use a simple interference model in which the impact of interference to a virtual gang w_l is incorporated in its WCET C_l by scaling the length of its longest constituent task as follows: $C_l = \max_{\forall \tau_k \in w_l}\{c_k\} \times \max(R_l, 1)$; intuitively, we assume that w_l suffers no interference until the resource is over-utilized, after which we apply a linear scaling. We note that this simple interference model is based on our experimental evaluation on two real embedded platforms (a NVIDIA Jetson Nano and a Raspberry Pi 4) using a set of synthetic benchmarks where they compete for memory bandwidth of the evaluated platforms. We do not, however, claim the general correctness of the interference model. If a different interference model exists for a given hardware platform, it can be used instead. Furthermore, we stress that regardless of the used interference model and its accuracy, it stands to reason that the static nature of virtual gangs enables low timing variability, effective shared resource partitioning (e.g, [16], [17]), and accurate WCET estimations.

IV. Virtual Gang Formation

Problem Statement: For a given candidate-set Δ_T of N tasks with the same period T and a given multicore platform with m unit-speed CPU cores with a known interference model, we want to partition the N tasks into a set of virtual gangs such that the total completion time of the virtual gangs is minimized, while respecting all the precedence constraints among the original tasks. In the following, we first describe a satisfiability modulo theories (SMT) based optimal algorithm for finding the virtual gang configuration with minimum completion time and then explain a heuristic solution.

A. Optimal Virtual Gang Formation via SMT

In the SMT based solution for virtual gang formation, we write the constraints for our optimization problem in a form that can be understood by an SMT solver; based on the candidate-set. The resulting SMT script is then fed to the SMT solver which declares the problem as either satisfiable or unsatisfiable. For a satisfiable problem, we also obtain a model for the input parameters that satisfies the constraints of the problem. We use quantifier free linear integer arithmetic logic (QF-LIA) of SMT. In the following, we describe the parameters of our problem and the constraints for a feasible solution.

Parameters: For each task τ_i in the considered candidate-set Δ_T, we use the variable x_i to denote the index of the virtual gang that τ_i is assigned to in a feasible solution. Note that for a candidate-set with N tasks, at-most N virtual gangs can be formed. We assume that the virtual gangs are indexed in the linear order in which they are required to execute. Consistent with our system model, we use the variable C_i to denote the length (WCET) and the variable R_i to denote the resource demand of each virtual gang.

Constraints: The parameters x_i, C_i and R_i are subject to the following constraints:

Constraint 1: $\forall \tau_i \in \Delta_T : 1 \leq x_i \leq N$

The value of each x_i must be between 1 and N; because we can have at-most N virtual gangs.

Constraint 2: $\forall j = 1...N : \sum_{\forall \tau_i \in \Delta_T | x_i = j} h_i \leq m$

The combined core demand of all the tasks assigned to each virtual gang must not exceed the total number of cores m.

Constraint 3: $\forall G_p \mid T_p = T, \forall(\tau_i, \tau_j) \in e_p : x_i < x_j$

If τ_i has a precedence constraint with τ_j, the virtual gang x_i containing τ_i must execute before the virtual gang x_j.

Constraint 4: $\forall j = 1...N : R_j \geq \sum_{\forall \tau_i \in \Delta_T | x_i = j} r_i$

The combined resource demand R_j of the j-th virtual gang must be greater than or equal to the sum of the resource demands of its constituent tasks.

Constraint 5: $\forall j = 1...N, \forall \tau_i \in \Delta_T \mid x_i = j : C_j \geq c_i \wedge C_j \geq c_i \times R_j$

The length C_j of the j-th virtual gang must be greater than or equal to the length of each of its constituent tasks, as well as the length of each constituent task multiplied by the combined resource demand R_j. In essence, this means that C_j must be at least equal to the length of the longest constituent task multiplied by $\max(1, R_j)$; the above formulation ensures that the constraints are expressed in linear arithmetic by removing the $\max$.

Constraint 6: $\sum_{j=1}^{N} C_j = \mathbb{C}$

Finally, the combined length of all virtual gangs must be equal to a specified value $\mathbb{C}$ whose minimum possible assignment needs to be found.

Since we use quantifier free logic, in our SMT formulation, we remove the universal quantifier ($\forall$) by repeating each constraint for every task τ_i, index j and/or edge (τ_i, τ_j) in the corresponding constraint formula. To find the virtual gang combination with minimum collective length, we conduct binary search on the combined gang length value $\mathbb{C}$; starting with the sum of the length of all the tasks in the candidate-set i.e., $\mathbb{C}_{init} = \sum_{\forall \tau_i \in \Delta_T} c_i$. In each step of the binary search, the SMT script is re-run with a new value of $\mathbb{C}$; to check if a model of input parameters (x_i, R_j and C_j) can be found which satisfies the constraints. If a satisfiable solution is found, the maximum combined gang length is reduced (search down); otherwise it is increased (search up). The process is repeated until the maximum combined gang length cannot be changed any further; in which case, the last solution, that was found satisfiable, is taken as the optimal solution. Note that a satisfiable solution can be found for any value of $\mathbb{C}$ equal to or greater than the optimal because Constraint 5 requires the values of C_j to be greater than or equal, rather than exactly equal, to the adjusted length of the longest constituent task. This allows the SMT solver to find feasible solutions quicker; note that the same applies for R_j in Constraint 4. The values of x_i in the optimal solution are used to create a new taskset of virtual gangs by combining the tasks in the candidate-set.

B. Virtual Gang Formation Heuristic

Due to the combinatorial nature of the optimization problem of virtual gang formation, the time required for obtaining the optimal solution via SMT quickly becomes intractable with increasing candidate-set size; as can be seen in Sec V-C. For this reason, we design a fast running heuristic for virtual gang formation, which is shown in Algorithm 1.

At the high-level, the algorithm tries to group tasks with similar WCET values so long as their combined shared resource utilization is not too high; to make the virtual gang's WCET as small as possible while fully utilizing the cores. The algorithm begins by sorting tasks in Δ_T by their WCETs (*line-4*). It then removes the longest task τ_i from the sorted queue and identifies all tasks τ_j which can be paired with τ_i; under the following constraints: 1) The combined core demand of τ_i and τ_j must be less than m. 2) τ_i and τ_j must not be related by precedence constraints (*lines-10:12*).

To check for precedence constraints, we introduce the notion of the $family$ of a node τ_i in our DAG which comprises all nodes τ_k that are connected with τ_i in an ancestor or descendant relationship i.e., $family(\tau_i) = \{\tau_k \mid \tau_k \in ancestor(\tau_i) \vee \tau_k \in descendant(\tau_i)\}$. With this, the precedence constraint check between τ_i and τ_j becomes $\tau_j \notin family(\tau_i)$ i.e., τ_j cannot be paired with τ_i if it is a member of τ_i's family.

Algorithm 1: Virtual Gang Formation Heuristic

```
1  Input: Candidate Set ($\Delta_T$), Number of Cores ($m$)
2  Output: Taskset comprising virtual gangs
3  function gang_formation($\Delta_T$, $m$)
4      $pq = sort\_tasks\_by\_wcet(\Delta_T)$
5      $virtualGangs = ()$
6      while $not\_empty(pq)$ do
7          $\tau_i = pq.pop()$
8          $f_i = family(\tau_i)$
9          $partners = ()$
10         for $\tau_j \in pq$ do
11             if $\tau_i.h + \tau_j.h \leq m \ \wedge \ \tau_j \notin f_i$ then
12                 $partners \leftarrow partners \cup \{\tau_j\}$
13         $pq_i = score\_partners(partners)$
14         while $not\_empty(pq_i)$ do
15             $\tau_p = pq_i.pop()$
16             $\tau_i = merge(\tau_i, \tau_p)$
17             $pq.remove(\tau_p)$
18             $update\_partners(\tau_i, pq_i)$
19         $virtualGangs \leftarrow virtualGangs \cup \{\tau_i\}$
20 return $virtualGangs$
```

In the list of potential corunners of τ_i, we score each task τ_p based on the *net advantage* that can be obtained by pairing it with τ_i using the following idea. The *advantage* obtained by pairing τ_p with τ_i is equal to the length of τ_p since it is the shorter running task in the potential virtual gang. The *disadvantage* of this pairing is the potential *increase* in the length of τ_i if $\tau_i.r + \tau_p.r > 1$. The *net advantage* is then equal to the difference between these two values; which we use to score all the potential partners of τ_i (*line-13*).

Once all the partners are scored, we keep removing the partner with the currently highest score from the partner list and merge it with τ_i to create a virtual gang; until no more pairing is possible (*lines-15:16*). In each step, we keep track of the precedence constraint in the DAG from forming virtual gang (since merging tasks can change the precedence constraint relationships in the DAG) and update the partner list to remove any tasks that can no longer be paired due to the new precedence constraint requirements (*line-18*). A task that is paired off is removed from the priority queue as well. Once a virtual gang is finalized, we put it in a separate list and start the process again by selecting the next τ_i from priority queue until the queue is empty. The final virtual gangs and the transformed DAGs of the candidate-set are returned once the heuristic finishes.

V. Schedulability Analysis

The schedulability analysis of a taskset comprising virtual gangs $\{w_1, w_2, \cdots, w_l\}$ under the rate-monotonic priority assignment scheme [11] can be done by performing fixed point iteration on the response time equation:

$$\mathbb{R}_i^{k+1} = \overline{C_i} + \sum_{\forall w_j \in hp(w_i)} \left\lceil \frac{\mathbb{R}_i^k}{T_j} \right\rceil C_j, \tag{1}$$

where: $\mathbb{R}_i^k$ is the response-time of w_i at the k-th iteration; $\overline{C_i}$ is the sum of the WCET of w_i itself and all the virtual gangs with the same period which come before w_i in the linear execution order; and $hp(w_i)$ represents the set of all virtual gangs which have higher priority than w_i (i.e., smaller period). The taskset is deemed schedulable if for each w_i, the final response time $\mathbb{R}_i$ is less than the period T_i [1].

In the following, we compare schedulability results with virtual gang formation with other parallel real-time task scheduling approaches with synthetically generated tasksets.

A. Simulation Study

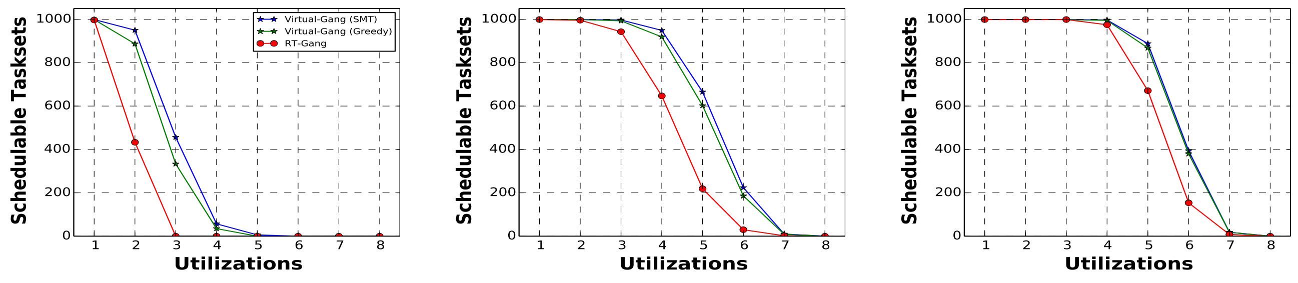
Taskset Generation: For the real-time taskset generation, we first uniformly select a period T_i in the range $[10, 1500]$. For each T_i, N tasks $\tau_{i,j}$, where N is randomly picked from the interval $[2, m]$, are generated by selecting a WCET $c_{i,j}$ in the range $[T/10, T/5]$, a resource demand factor $r_{i,j}$ in the interval $[0, 1]$ and a parallelism level $h_{i,j}$. The utilization $u_{i,j}$ of each $\tau_{i,j}$ is then calculated using the relation: $u_{i,j} = (c_{i,j} \times h_{i,j})/T_i$. If $u_{i,j}$ is less than the remaining utilization for the taskset, $c_{i,j}$ is adjusted so that $\tau_{i,j}$ fills the remaining utilization. Otherwise, taskset generation continues until the desired level of utilization is reached. We generate 1000 tasksets for each data point.

Precedence Constraints: Once a taskset is generated, we model precedence constraint by adding edges among tasks, which have the same period T_i, based on an edge probability value $P(e)$ which represents the average chance of an outgoing edge from one task to another. To simplify the creation of a DAG without explicitly checking for a cycle, we assume that an edge can only exist between $\tau_{i,j}$ and $\tau_{i,k}$ if $j < k$; hence, task $\tau_{i,N}$ has no outgoing edges. Under this scheme, the tasks with smaller index values have potentially more *neighbors*, to have an edge with, than the tasks with larger index values. To have a balanced edge generation scheme, we divide $P(e)$ value by the number of potential neighbors of a task.

Taskset Types: Similarly to [7], we consider three types of tasksets in our simulation, based on the allowed level of parallelization $h_{i,j}$ for the tasks in the taskset. For a *lightly-parallel* taskset, $h_{i,j}$ is uniformly selected in the range $[1, \lceil 0.3 \times m \rceil]$. For a *heavily-parallel* taskset, the value of $h_{i,j}$ is picked from the range $[\lceil 0.3 \times m \rceil, m]$. Finally, for *mixed* taskset, $h_{i,j}$ is selected randomly from the interval $[1, m]$.

Scheduling Policies: We conduct two separate experiments to understand the impact of virtual gang formation on system schedulability. In the first experiment, we consider precedence constraints among tasks using an edge probability $P(e) = 0.25$. Due to the absence of an existing schedulability analysis for gang tasks that can handle precedence constraints to the best of our knowledge, we only compare schedulability with virtual gangs in this experiment against RT-Gang. Concretely, we consider three scheduling scenarios in this experiment. Under the *RT-Gang* scheme, the unicore response time analysis using Equation 1 is applied to calculate schedulability of the taskset under the one-gang-at-a-time scheduling. For *Virtual Gang*

[1]It suffices to check the last virtual gang in the linear order for each period.



(a) Lightly Parallel (b) Mixed (c) Heavily Parallel

Fig. 1: Schedulability plots for tasksets with precedence constraints ($P(e) = 0.25$) on 8 cores

(SMT), we first form virtual gangs from the given taskset using the optimal SMT algorithm and then use Equation 1 to calculate schedulability of the new taskset comprising the virtual gangs. Under *Virtual Gang (Heuristic)*, we use the heuristic from Sec IV-B to form virtual gangs and then calculate the schedulability results.

In the second experiment, we assume that there are no precedence constraints among the tasks i.e., $P(e) = 0$ which allows us to consider more multicore scheduling policies for comparing against virtual-gang scheduling. In this case, for each taskset type, we calculate schedulability results under four scheduling policies. We retain the RT-Gang and virtual-gang (SMT) schemes from the first experiment. In addition, we consider *Gang-FTP* policy and use the analysis in [7] to calculate schedulability of the taskset under gang fixed-priority scheduling. We also consider the *Threaded* scheme which models the scheduling of parallel tasks under vanilla Linux real-time scheduler, where the $h_{i,j}$ threads of each task $\tau_{i,j}$ are independently scheduled. In this case, we assess schedulability based on the state-of-the-art analysis for fixed-priority scheduling of DAG tasks in [15]; here $\tau_{i,j}$ is simply modeled as a DAG of $h_{i,j}$ nodes with the same execution time.

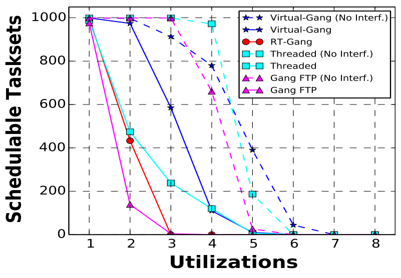
Interference Model: For each scheduling policy considered in the second experiment, we calculate schedulability with and without taking the interference between corunning tasks into account. In virtual gang scheduling, the gang formation algorithms already incorporate the interference model described in Sec III-B in creating virtual gangs. For *Gang-FTP*, for each $\tau_{i,j}$, we enumerate all possible sets of co-running tasks based on the remaining number of cores $m - h_{i,j}$, and pick the set with the maximal combined resource demand $R_{i,j}$ which we then use to scale the execution time $c_{i,j}$ of $\tau_{i,j}$ by multiplying it with $\max(1, R_{i,j})$. For *Threaded*, we assume that each independently scheduled thread of $\tau_{i,j}$ has a resource demand of $r_{i,j}/h_{i,j}$, and pick the $m-1$ other threads (either of the same or different task) with maximal demands. While the described procedure for *Gang-FTP* and *Threaded* can be pessimistic, we are not aware of any better mechanism to safely account for the effect of resource interference under such scheduling policies. Furthermore, we point out that results for *Threaded* can still be optimistic, since we do not account for the extra synchronization overheads that could be incurred when scheduling threads independently rather than as a gang. Finally, in creating plots without interference, we set the resource demand of each task to zero and redo all our calculations (e.g., SMT virtual gang formation) before calculating schedulability.

B. *Schedulability Results*

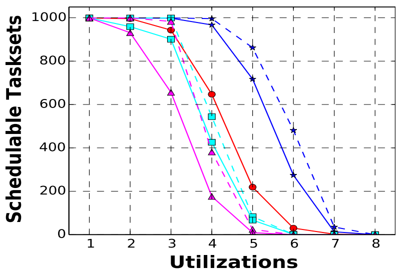
Figure 1 shows the schedulability plots for the first experiment with $P(e) = 0.25$ from our simulation for 8 cores ($m = 8$). For all taskset types, virtual gang formation provides noticeable improvement in schedulability as compared to RT-Gang. Moreover, the virtual gang formation heuristic does a good job in giving comparable performance to the optimal SMT algorithm; in terms of total number of schedulable tasksets under both schemes. It can also be seen from this figure that the difference in the performance between the heuristic and the optimal virtual gang formation decreases as the parallelization level of the tasksets increases. This is expected since for lightly parallel tasksets, there are much greater possibilities for virtual gang formation which can be missed by the greedy local optimization criteria of the heuristic.

When the tasksets comprise independent tasks (i.e., $P(e) = 0$), Fig 2 shows the schedulability results for the considered scheduling policies of the second experiment. For lightly parallel tasksets, the *Threaded* and *Virtual Gang* schemes give the best schedulability results, followed closely by the *Gang-FTP* policy, if interference model is not used (dashed lines). However, when interference is considered (solid lines), the schedulability under *Threaded* and *Gang FTP* policies deteriorates rapidly as compared to the *Virtual Gang* scheme. This is due to the fact that under the *Virtual Gang* scheme, only the tasks of the same virtual gang can possibly interfere with each other, while a lot more tasks must be considered in Gang-FTP and Threaded. As expected, *RT-Gang* suffers the most for lightly parallel tasks as it under-utilizes the cores.

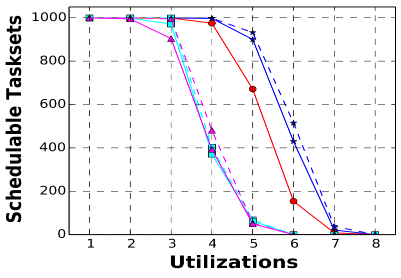
For mixed and heavily parallel tasksets, the *Virtual Gang* scheme outperforms the rest regardless whether interference is considered or not. For these taskset types, *RT-Gang* performs considerably better as well since a single parallel task can utilize more cores in the platform, though it still lags behind Virtual Gang scheme. On the other hand, *Gang-FTP* and *Threaded* are significantly worse than the Virtual Gang scheme and the RT-Gang for both mixed and heavily parallel tasksets. This can be attributed to the analysis pessimism needed to handle carry-in jobs in their schedulability tests [7], [15], which becomes more pronounced as the parallelism of the tasks



(a) Lightly Parallel



(b) Mixed



(c) Heavily Parallel

Fig. 2: Schedulability plots for tasksets containing independent tasks ($P(e) = 0$) on 8 cores.

increases. Because both Virtual Gang and RT-Gang can use exact unicore-based fixed-priority schedulability techniques, they do not suffer from such analysis pessimism.

Finally, in all cases, interference impact becomes less prominent as the parallelization of the taskset increases. This is because with highly parallel tasks, the opportunity of getting co-scheduled with other resource intensive tasks decreases, leading to improved schedulability.

In summary, our simulation results show that the Virtual Gang scheme significantly outperforms the rest when interference is considered, and is competitive even when interference is not considered.

C. SMT and Heuristic Gang Formation Runtime

In this experiment, we compare the time required to form virtual gangs from a given candidate-set using the SMT and the heuristic algorithms. We use the Z3 SMT solver [18]. We vary the candidate-set size (N) from 4 tasks up-to 9 tasks and measure the time taken by each algorithm in generating virtual gangs. For each N, we generate 75 candidate-sets and process them through the gang formation algorithms. We retain all the other simulation parameters from the first experiment in the previous section. We plot the time taken by each algorithm for all the candidate-sets in Fig 3. It can be seen that the runtime of obtaining the optimal solution via SMT increases with an exponential trend and quickly becomes unmanageable. The runtime of the heuristic, on the other hand, remains relatively stable (within 10-msec) for all candidate-set sizes.

VI. Conclusion

We introduced the novel concept of a virtual gang: a group of parallel tasks that are statically linked and scheduled together. We presented virtual gang formation algorithms and demonstrated how to tightly bound the effect of shared resource interference on COTS multicore platforms and how to transforms the original, complex scheduling problem into a form that is amenable to unicore schedulability analysis. In future, we plan to demonstrate the practical benefits of our approach in a real operating system with real-world workloads.

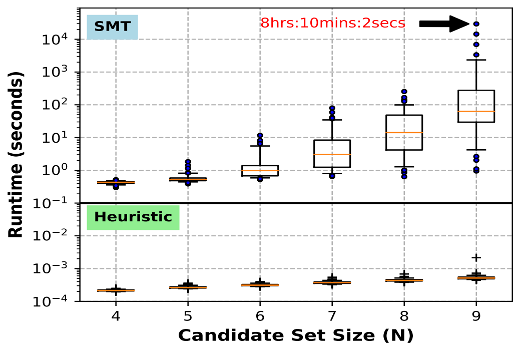


Fig. 3: Comparison of SMT and heuristic virtual gang formation runtime. In each box, the orange line represents the median value. The box represents the interquartile range (Q2-Q3). The lower and upper whiskers mark the 5 percentile and the 95 percentile values respectively.

References

[1] M. G. Bechtel and H. Yun, "Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention," in *RTAS*, 2019.

[2] Certification Authorities Software Team, "CAST-32A: Multi-core Processors," tech. rep., Federal Aviation Administration, 2016.

[3] D. G. Feitelson and L. Rudolph, "Gang Scheduling Performance Benefits for Fine-Grain Synchronization," *Journal of Parallel and distributed Computing*, vol. 16, no. 4, pp. 306–318, 1992.

[4] V. Berten, P. Courbin, and J. Goossens, "Gang Fixed Priority Scheduling of Periodic Moldable Real-Time Tasks," in *RTNS*, 2011.

[5] J. Goossens and V. Berten, "Gang FTP Scheduling of Periodic and Parallel Rigid Real-Time Tasks," in *RTNS*, 2010.

[6] S. Kato and Y. Ishikawa, "Gang EDF scheduling of Parallel Task Systems," in *RTSS*, 2009.

[7] S. Wasly and R. Pellizzoni, "Bundled Scheduling of Parallel Real-Time Tasks," in *RTAS*, 2019.

[8] A. Bhuiyan *et al.*, "Mixed-Criticality Multicore Scheduling of Real-Time Gang Task Systems," in *RTSS*, 2019.

[9] H. Yun, R. Pellizzon, *et al.*, "Parallelism-Aware Memory Interference Delay Analysis for COTS Multicore Systems," in *ECRTS*, 2015.

[10] W. Ali and H. Yun, "RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems," in *RTAS*, 2019.

[11] N. Audsley *et al.*, "Applying New Scheduling Theory to Static Priority Preemptive Scheduling," *Software Engineering Journal*, vol. 8, 1993.

[12] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *CVPR*, 2005.

[13] S. Kato *et al.*, "An Open Approach to Autonomous Vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[14] A. Saifullah *et al.*, "Multicore Real-Time Scheduling for Generalized Parallel Task Models," *Real-Time Systems*, no. 4, 2013.

[15] J. Fonseca, G. Nelissen, *et al.*, "Improved Response Time Analysis of Sporadic DAG Tasks for Global FP Scheduling," in *RTNS*, 2017.

[16] H. Yun *et al.*, "MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multicore Platforms," in *RTAS*, 2013.

[17] H. Yun *et al.*, "PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms," in *RTAS*, 2014.

[18] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *TACAS*, 2008.