

Simultaneous Multithreading and Common-Period Sporadic Tasks in Hard Real-Time

Sims Hill Osborne
 Department of Computer Science
 Elon University
 Elon, North Carolina, U.S.A.
 sosborne3@elon.edu

Abstract—Simultaneous multithreading (SMT) can significantly improve hard real-time scheduling, but existing methods are limited to scenarios with pre-determined job release times. Here, a scheduling algorithm and polynomial-time schedulability test targeting sporadic, common-period systems is given. The challenge of using SMT here is that job costs are dependent on when other jobs release and are executed. The schedulability test given here uses the maximum-weight matching problem from graph theory to upper-bound the execution costs even given the worst possible release pattern. Schedulability studies show that with this algorithm, systems with utilizations exceeding 1.2 can be scheduled without deadline misses on a single core, a 20% increase compared to the best case without SMT.

Index Terms—real-time systems, simultaneous multithreading, hard real-time, scheduling algorithms

I. INTRODUCTION

Simultaneous Multithreading (SMT) allows multiple programs to execute at the same time on a single computing core. With careful planning, this technique allows for safe scheduling of otherwise unschedulable real-time task systems [1]. Doing so has particular advantages for embedded and cyber-physical systems, where size, weight, and power (SWaP) restrictions may limit both the number and speed of computing cores. Unfortunately, existing work on applying SMT to hard real-time (HRT) systems [2]–[4] relies on advance knowledge of job release times and so cannot support sporadic systems. However, sporadic tasks are common in real-time systems [5]. To support such systems, SMT-aware schedulers need to account for the worst possible job release times.

Contribution and organization. In this paper, we give a scheduling algorithm and SMT-aware schedulability test for common-period sporadic HRT systems. The challenge is that with SMT, every job’s cost is dependent on what other job is executing at the same time. Our $O(n^5)$ schedulability test (n denotes task count) accounts for every possible job cost without prior knowledge of which jobs will be scheduled at the same time. This test can be included in toolchains that work to determine hardware requirements at design-time.

Presently, we only consider single-core systems in which all tasks share a common period. This analysis is a first step towards using SMT to support systems with many periods that require many cores to be correctly scheduled. We suggest possible next steps towards this larger goal in Sec. V.

Secs. II and III discuss background information and our theoretical analysis, respectively. In Sec. IV, we present schedu-

lability studies that show, given the assumptions of our model and favorable but plausible conditions, that our method allows us to consistently schedule systems with utilizations of 1.2 or more on a single core; normally, task systems with utilization greater than 1.0 require multiple cores. In Sec. V, we conclude and discuss directions for future work.

II. BACKGROUND

A. Task and Platform Model

We consider scheduling a common-period sporadic hard real-time task system consisting of n independent tasks without precedence constraints, critical sections, or shared resource requirements. All tasks share a common period, T . Each task τ_i releases a single job at most once every T time units. The earliest possible release is time 0, but we do not assume synchronous start times. Each job has a maximum cost when not using SMT of C_i time units. C_i and the SMT-specific costs defined in Sec. II-B below are assumed to include overheads.

We use *implicit deadlines*: every job of a task τ_i must complete within T time units of its release. The simplest way to denote a task is $\tau_i = (C_i, T)$; additional parameters related to SMT are discussed in Sec. II-B below. Each task τ_i has a *utilization* given by $u_i = \frac{C_i}{T}$, with the total utilization of all tasks without SMT given by U . Without SMT, no system with $U > 1$ is schedulable on a single core [6], [7]. Our goal is to use SMT to schedule systems with $U > 1$ on a single core.

The j^{th} job of task τ_i is denoted as $\tau_{i,j}$. We use the job-level parameters $r_{i,j}$, $s_{i,j}$, $f_{i,j}$, and $d_{i,j}$ to denote the release, start, finish, and deadline times, respectively, of job $\tau_{i,j}$. Since $d_{i,j} = r_{i,j} + T$ and T gives the minimum inter-arrival time between jobs of the same task, i.e. $r_{i,j+1} \geq r_{i,j} + T$, it follows that $r_{i,j+1} \geq d_{i,j}$ must hold.

The system is scheduled correctly if and only if it can be shown that no job will ever miss a deadline, i.e., $f_{i,j} \leq d_{i,j}$ holds for all i and j . We use the terms *pending* and *age* to describe a job’s current status.

Def. 1. Job $\tau_{i,j}$ is pending at all times in $[r_{i,j}, f_{i,j})$. The age of $\tau_{i,j}$ at time t is equal to $t - r_{i,j}$. ◀

B. Overview of Simultaneous Multithreading

In modern processors, each core uses instruction-level parallelism within jobs to execute multiple instructions per cycle. With SMT, this behavior is expanded to allow multiple jobs to

execute instructions simultaneously. The result is that individual jobs will have increased execution times, but total throughput is increased. In most cases, the execution time for two tasks in parallel with SMT is less than the time required to execute the two sequentially [2], [3], [8]–[10]. Exactly how much execution times increase, and thus whether SMT is useful, is dependent on what other work is scheduled on the same core.

Foundational work on SMT was done by Tullsen et al. and Eggers et al. [11]–[13]. Practical factors that can affect SMT were discussed by Bulpin [8], [9]. Additional work on SMT to support real-time scheduling can be found in [1], [4], [14]–[21]. None of these works attempt to upper-bound worst-case response times when any job can potentially be scheduled with any other job. More recently, [22] gives scheduling and measuring techniques that use SMT to increase throughput and reduce scheduling latency, although without the strict restrictions of real-time scheduling. In our model, computing cores can support two execution threads; these conditions match those of both Intel and AMD processors that support SMT [23].

Simultaneous co-scheduling. One way to use SMT in a real-time context is to view each hardware thread as a separate core. This approach works for soft real-time systems that allow some deadline misses [1], [10], but worst-case execution time analysis to support HRT systems would need to consider all combinations of tasks and start times. If a timing analysis assumes jobs of two tasks will always begin simultaneously, then the timing analysis may not hold when they begin separately. Consequently, any deadline guarantees based on such an assumption would be invalid. However, undertaking timing analysis for every possible starting-time offset between every pair of jobs is impossible.

To make timing analysis feasible, we require *simultaneous co-scheduling* as developed in [3], which shows that with simultaneous co-scheduling, the reliability of timing analysis with SMT is comparable to reliability without SMT. This work also gives a more detailed explanation of SMT, including illustrated examples.

Def. 2. We say that two jobs are simultaneously co-scheduled, or paired, if both begin execution simultaneously on threads of the same core, and when one job completes, the remaining job continues on the same core, without an additional paired job, until complete. ◀

The practicality of simultaneous co-scheduling on real hardware has been previously demonstrated in case studies in [2], [20]. In these works, starting times of simultaneously co-scheduled jobs were synchronized within nanoseconds. More importantly, predicted execution times upper-bounded execution times observed during runtime, allowing for all jobs to meet their deadlines on real hardware.

Paired jobs require additional definitions for costs.

Def. 3. [3] Let $C_{i(k)}$ be the maximum cost of a job of τ_i given that it is simultaneously co-scheduled with a job of τ_k , and let $C_{k(i)}$ be the maximum cost of a job of a job of τ_k given that it is simultaneously co-scheduled with a job of τ_i . We use $i = k$

Algorithm 1

Require: Common period; non-preemptive scheduling.

```

1: while true do
2:   if any SMT-ineligible jobs are pending then
3:     Schedule max. age SMT-ineligible job.
4:   else if two or more SMT-eligible jobs are pending then
5:     Schedule two max. age SMT-eligible jobs as a pair.
6:   else if one SMT-eligible job is pending then
7:     Schedule job as a solo job.
8:   end if
9:   If no job is scheduled, wait until a new job releases.
10:  Else wait until scheduled job(s) complete.
11: end while

```

to denote jobs not using SMT, i.e., $C_{i(i)} = C_i$. The pair cost is

$$C_{i:k}^* = C_{k:i}^* = \max(C_{i(k)}, C_{k(i)}) \blacktriangleleft .$$

III. SCHEDULING ALGORITHM AND TEST

In this section we give our scheduling algorithm (Alg. 1) and the associated schedulability test. We distinguish between *SMT-eligible* and *SMT-ineligible* tasks.

Def. 4. An SMT-eligible task will use SMT whenever possible. An SMT-ineligible task will never use SMT. Both designations are made prior to runtime. ◀

This distinction improves scheduling in systems where some tasks do not gain from using SMT. For example, if $C_{i(k)} > C_i + C_k$, then making at least one of τ_i or τ_k SMT-ineligible would likely improve schedulability. The schedulability test we give holds for any given partition of τ into SMT-eligible and SMT-ineligible tasks. However, τ may be schedulable given some partitions and unschedulable given others. We discuss a heuristic for forming this partition in Sec. IV.

Def. 5. Let C^{no-smt} be equal to the total cost of all SMT-ineligible tasks, i.e. $C^{no-smt} = \sum_{\forall \tau_i \in \tau: \tau_i \text{ is SMT-ineligible}} C_i$. ◀

Our analysis of Alg. 1 is built on *idle instants* and *near-idle instants*. Idle instants were first formalized for schedulability analysis by Davis and Burns [24]; we add the near-idle instant as a complementary concept. Our definition of an idle instant is inspired by but differs from that of Davis and Burns.

Def. 6. An idle instant occurs whenever no pending job has an age greater than zero. ◀

Def. 7. A near-idle instant $s_{i,j}^*$ occurs at when the only pending job is the unstarted SMT-eligible job $\tau_{i,j}$. Under Alg. 1, $s_{i,j}^*$ will also be the start time of $\tau_{i,j}$. $s_{i,j}^*$ cannot coincide with the release of another job. ◀

A. Starting from an Idle Instant

To produce a schedulability test, we give conditions that guarantee every idle or near-idle instant will be followed by another idle or near-idle instant and that no deadline misses will occur between consecutive idle and near-idle instants. We first focus on conditions needed to meet deadlines starting from an

idle instant t_0 . We rely on the following property of our system, which follows from the definition of sporadic tasks and from T being the common period.

Prop. 1. *At most one job of each task can be released in any interval $[s, s + T)$.*

To upper-bound execution time of all SMT-eligible jobs, we model possible task interactions as a graph and then find the graph's *maximum-weight matching*.

Def. 8. *Given a weighted graph $G = (V, E)$, where V denotes a vertex set and E an edge set, a maximum-weight matching (MWM) is a set of edges $E' \subset E$ that maximize the total weight of edges in E' while subject to the restriction that every vertex $v \in V$ is incident on at most one edge in E' . We use $M(G)$ to denote the total weight of all edges in the set. ◀*

Prop. 2. *Let G' be a subgraph of graph $G = (V, E)$. $M(G') \leq M(G)$ must hold.*

MWMs can be found in $O(V^2E) = O(V^4)$ time using Edmonds' blossom algorithm [25].

We define a series of graphs that upper-bound SMT-eligible execution costs during specified intervals. We first use the MWM to upper-bound the maximum total execution cost of SMT-eligible jobs released in any interval $[t_0, t_0 + T)$ where t_0 is an idle instant. In Sec. III-B, we consider intervals beginning with near-idle instants.

Def. 9. *Let $G1$ be a graph constructed as follows: 1) Define a vertex v_i for one job of each SMT-eligible task $\tau_i \in \tau$; 2) For every unordered pair of vertices v_i and v_k , define a weight $C_{i,k}^*$ (Def. 3) edge connecting the two vertices; 3) Define a solo vertex s representing possible solo execution and connect every vertex v_i to s by a weight C_i edge. Including only one solo vertex ensures that $M(G1)$ models scenarios where at most one SMT-eligible job experiences solo execution. ◀*

Corollary 1. *$M(G1)$ upper-bounds the total execution time for a set of jobs consisting of one job per SMT-eligible task provided that (A) at most one such job experiences solo execution and (B) no such job is co-scheduled with a job not belonging to the set.*

Proof: This follows from the definition of $G1$ above. ■

Lemma 1. *Let t_0 be an idle instant. If*

$$C^{\text{no-smt}} + M(G1) < T \quad (1)$$

holds, then at least one of the following is true: (A) there exists an additional idle instant t_1 , $t_0 < t_1 < t_0 + T$ such that no job released in $[t_0, t_1)$ misses its deadline or; (B) there exists a near-idle instant $s_{y,z}^$, $t_0 \leq s_{y,z}^* < t_0 + T$ such that no job released in $[t_0, s_{y,z}^*)$ misses its deadline.*

Proof: At most one of each job can be released in $[t_0, t_0 + T)$ (Prop. 1). Thus total execution time of SMT-ineligible jobs released in $[t_0, t_0 + T)$ is upper-bounded by $C^{\text{no-smt}}$ (Def. 5).

It follows from Prop. 2 and Cor. 1 that total execution time of SMT-eligible jobs released in $[t_0, t_0 + T)$ can exceed $M(G1)$

only if multiple SMT-eligible jobs experience solo execution or at least one job released in $[t_0, t_0 + T)$ is co-scheduled with a job released no sooner than $t_0 + T$. We consider three possibilities.

First, if there is an interval in $[t_0, t_0 + T)$ in which no job executes, then the beginning of that interval is an idle instant t_1 . Furthermore, no job released in $[t_0, t_0 + T)$ can have a deadline before $t_0 + T$ and a lack of executing jobs implies that all jobs released in $[t_0, t_1)$ complete no later than t_1 . Thus all jobs released in $[t_0, t_1)$ meet their deadlines. Jobs released after t_1 are outside the scope of this Lemma; (A) holds.

Second, if an SMT-eligible job $\tau_{y,z}$ does execute without SMT, then its start time is a near-idle instant $s_{y,z}^*$ (Def. 7).

If (1) holds, $\tau_{y,z}$ is the first solo job within $[t_0, t_0 + T)$, and jobs have been executing at all times in $[t_0, s_{y,z}^*)$, then time elapsed between t_0 and $s_{y,z}^*$ must be less than $C^{\text{no-smt}} + M(G1)$. It follows that

$$s_{y,z}^* < t_0 + C^{\text{no-smt}} + M(G1) < t_0 + T$$

holds and the first part of (B)— $s_{y,z}^*$ exists in $[t_0, t_0 + T)$ —is true. $s_{y,z}^* < t_0 + T$ implies that all jobs released in $[t_0, t_0 + T)$ that complete before $s_{y,z}^*$ meet their deadlines. As $M(G1)$ accounts for the execution of $\tau_{y,z}$ as a solo job, (1) holding implies that $\tau_{y,z}$ also meets its deadline. Jobs released after $s_{y,z}^*$ are outside the scope of this lemma. (B) holds.

Third, if jobs execute at all times in $[t_0, t_0 + T)$, no SMT-eligible job experiences solo execution, and (1) holds, then all jobs released in $[t_0, t_0 + T)$ will finish no later than $t_0 + T$. An idle instant occurs at $t_0 + T$. No deadline for these jobs can fall before $t_0 + T$, so no deadlines are missed; (A) holds. ■

B. Starting from a Near-Idle Instant

Here we show when timely completion of jobs that were preceded by a near-idle instant can be guaranteed. Again, our first step is to model possible jobs and costs as a graph.

Def. 10. *Let $G2$ be a graph constructed as follows: 1) Define a vertex v_i for one job of each SMT-eligible task $\tau_i \in \tau$; 2) For every unordered pair of vertices v_i and v_k , define a weight $C_{i,k}^*$ edge connecting the two vertices. ◀*

Lemma 2. *Let $s_{i,j}^*$ be a near-idle instant for $\tau_{i,j}$. Assume that $\tau_{i,j}$ completes on time. If*

$$C_i + C^{\text{no-smt}} + M(G2) < T \quad (2)$$

holds, then at least one of the following holds: (A) there exists an idle instant t_1 , $s_{i,j}^ \leq t_1 < s_{i,j}^* + T$ such that no job released in $[s_{i,j}^*, t_1)$ misses its deadline¹ or; (B) there exists a near-idle instant $s_{y,z}^*$, $s_{i,j}^* < s_{y,z}^* < s_{i,j}^* + T$ such that all jobs released in $[s_{i,j}^*, s_{y,z}^*)$ apart from SMT-eligible job $\tau_{y,z}$ complete on time.*

Proof: The proof is similar to that for Lem. 1. Per Def. 7, no other job can release at $s_{i,j}^*$. $\tau_{i,j}$ will therefore execute as a solo job and complete no later than $s_{i,j}^* + C_i$.

At most one additional job per task can be released in $[s_{i,j}^*, s_{i,j}^* + T)$ (Prop. 1). Total execution time of SMT-ineligible jobs released in $[s_{i,j}^*, s_{i,j}^* + T)$ is thus upper-bounded by $C^{\text{no-smt}}$.

¹If t_1 is equal to $s_{i,j}^*$, then $s_{i,j}^*$ is both an idle and near-idle instant.

Every SMT-eligible job released in $[s_{i,j}^*, s_{i,j}^* + T)$ corresponds to a vertex in $G2$. Unlike $G1$, $G2$ does not have an s node to allow for a solo job. Thus $M(G2)$ gives the maximum total execution time of SMT-eligible jobs other than $\tau_{i,j}$ released in $[s_{i,j}^*, s_{i,j}^* + T)$ assuming that no job released in $[s_{i,j}^*, s_{i,j}^* + T)$ is co-scheduled with a job outside of that interval and no additional SMT-eligible job executes as a solo job. It is possible that $\tau_{i,j}$ was released prior to $s_{i,j}^*$. We again consider three cases.

First, as in Lem. 1, if there is an interval in $[s_{i,j}^*, s_{i,j}^* + T)$ in which no job executes, then the beginning of that interval is an idle instant t_1 . All jobs released no sooner than $s_{i,j}^*$ but before t_1 must meet their deadlines. Any jobs released no sooner than t_1 are outside the scope of this Lemma, and (A) holds.

Second, if an SMT-eligible job $\tau_{y,z}$ is executed solo—which can happen, per Alg. 1, only if there are no other pending jobs²—then the start of $\tau_{y,z}$ is the near-idle instant $s_{y,z}^*$ (Def. 7). As there can be no other pending jobs at $s_{y,z}^*$, all jobs released in $s_{y,z}^*$ other than $\tau_{y,z}$ must have completed before their deadlines; (B) holds.

As for $\tau_{y,z}$, the time required to complete $\tau_{y,z}$, and whether or not it meets its deadline is thus outside the scope of this lemma and will be addressed in Lem. 3 below.

Third, if jobs execute at all times in $[s_{i,j}^*, s_{i,j}^* + T)$, no SMT-eligible job other than $\tau_{i,j}$ experiences solo execution prior to $s_{i,j}^* + T$ and (2) holds, then all jobs released in $[s_{i,j}^*, s_{i,j}^* + T)$ will finish no later than $s_{i,j}^* + T$. An idle instant occurs at $s_{i,j}^* + T$. No deadline for jobs released after $s_{i,j}^*$ can fall before $s_{i,j}^* + T$, so no deadlines are missed; (A) holds. ■

Finally, we consider a job $\tau_{y,z}$ released at the second of two consecutive near-idle instants.

Def. 11. Let $G3_i$ be a graph constructed as follows: 1) Define a vertex v_k for one job of each SMT-eligible task τ_k such that $k \neq i$; 2) For every unordered pair of vertices v_k and v_ℓ , define an edge of weight $C_{k:\ell}^*$ connecting the vertices. 3) Define a vertex s representing possible solo execution and connect every vertex v_k to s by a weight C_k edge. ◀

The only difference between $G3_i$ and G_i is that $G3_i$ excludes vertices corresponding to τ_i .

Lemma 3. Let $s_{i,j}^*$ and $s_{y,z}^*$ be consecutive near-idle instants such that $s_{y,z}^* < s_{i,j}^* + T$ holds. Assume $\tau_{i,j}$ completes on time. If (1) and

$$C_i + C^{no-smt} + M(G3_i) < T \quad (3)$$

are true, then $\tau_{y,z}$ will complete on time.

Proof: At $s_{y,z}^*$, only $\tau_{y,z}$ is pending (Def. 7). Hence all other jobs released in $[s_{i,j}^*, s_{y,z}^*)$ must have completed by $s_{y,z}^*$ and $\tau_{y,z}$ must execute alone.

We consider two scenarios. **First**, $\tau_{i,j+1}$ releases in $[s_{i,j}^*, s_{y,z}^*)$. Given that $\tau_{y,z}$ is only pending job at $s_{y,z}^*$, it must

²If $\tau_{y,z}$ is the released when no other jobs are pending, the $s_{y,z}^*$ is both an idle and near-idle instant.

have released no sooner than $\tau_{i,j+1}$. We have

$$\begin{aligned} r_{y,z} &\geq r_{i,j+1} \\ \rightarrow \{T \text{ sets both deadline and minimum inter-release time.}\} \\ r_{y,z} &\geq d_{i,j} \\ \rightarrow \{\text{Assumption that } \tau_{i,j} \text{ completes on time.}\} \\ r_{y,z} &\geq f_{i,j}, \end{aligned}$$

which means that $\tau_{y,z}$ will not need to wait on $\tau_{i,j}$.

No SMT-eligible job other than $\tau_{y,z}$ can undergo solo execution while $\tau_{y,z}$ is pending; if it did, that other job, rather than $\tau_{y,z}$, would mark a near-idle instant. Thus the solo vertex s in $G1$ accounts for the cost of $\tau_{y,z}$ itself and $M(G1)$ upper-bounds the total cost to execute all SMT-eligible jobs that can possibly be released in $(s_{i,j}^*, s_{y,z}^*)$ including $\tau_{i,j+1}$ and $\tau_{y,z}$. If (1) holds, then $\tau_{y,z}$ will complete within T time units of its release, i.e., on time.

Second, $\tau_{i,j+1}$ does not release in $[s_{i,j}^*, s_{y,z}^*)$. $\tau_{y,z}$ may wait on $\tau_{i,j}$, which we account for with C_i in (3), but it cannot wait on $\tau_{i,j+1}$ or any pair that includes $\tau_{i,j}$. We can exclude $\tau_{i,j+1}$ from the graph of SMT-eligible costs; hence the use of $G3_i$, which upper-bounds execution time of jobs belonging to tasks *excluding* τ_i . If (3) holds, then $\tau_{y,z}$ will complete on time. ■

Theorem 1. If (1) and (2) hold and (3) holds for every SMT-eligible task τ_i , then no job in τ will miss its deadline.

Proof: Per Lem. 1, if (1) holds, then every idle instant is followed by either an idle instant or a near-idle instant. Per Lem. 2, if (2) holds, then every near-idle instant is followed either by an idle instant or a near-idle instant. As the beginning of the schedule will always be an idle instant, every moment in the schedule will therefore be either an idle instant, a near-idle instant, or will fall between two such instants.

Lem. 1 also guarantees that if (1) holds, then all jobs released in $[t_0, t_1)$, where t_0 is an idle-instant and t_1 the next idle or near-idle instant will complete on time.

Likewise, Lem. 2 guarantees that if (2) holds, then every job released in $[s_{i,j}^*, t_0)$, where t_0 is the next idle instant, or $[s_{i,j}^*, s_{y,z}^*)$, where $s_{y,z}^*$ is the next near-idle instant, will complete on time with the possible exception of job $\tau_{y,z}$, the job that defines the second near-idle instant.

The only jobs they do not have timely completion guaranteed by Lemmas 1 and 2 are jobs such as $\tau_{y,z}$ that define the second of two consecutive near-idle instants.

Given this characterization of $\tau_{y,z}$, Lem. 3 guarantees its timely completion provided that (1) and (3) hold, the previous near-idle instant $s_{i,j}^*$ is such that

$$s_{y,z}^* - s_{i,j}^* < T \quad (4)$$

holds, and $\tau_{i,j}$, which defines near-idle instant $s_{i,j}^*$, also completed on time.

Per Lem. 2, (2) holding implies $s_{i,j}^* < s_{y,z}^* < s_{i,j}^* + T$; thus (4) holds. The only remaining condition for $\tau_{y,z}$ to complete on time is to show that $\tau_{i,j}$ completed on time. Proving that $\tau_{i,j}$ completes on time will therefore show that all jobs will finish on time, completing the proof.

If $r_{i,j} = s_{i,j}^*$, then $s_{i,j}^*$ is also an idle instant; the timeliness of $\tau_{i,j}$ is guaranteed by Lem. 1. Otherwise, $s_{i,j}^*$ must be preceded by either an idle instant or a near-idle instant. If it is preceded by an idle instant, $\tau_{i,j}$'s timeliness can be guaranteed by Lem. 1.

If $s_{i,j}^*$ is preceded by an earlier near-idle instant $s_{g,h}^*$, then we can show inductively that $\tau_{i,j}$ would not miss its deadline unless $\tau_{g,h}$ missed its deadline, which could not have happened unless an earlier job missed its deadline. As we induct backwards, we will eventually reach a job that was preceded by an idle instant; note the beginning of the schedule is always an idle instant. Timeliness of this early job is guaranteed by Lem. 1, thus guaranteeing timely completion of all subsequent jobs. ■

Corollary 2. *The worst-case time to determine the schedulability of τ is $O(n^5)$.*

Proof: Schedulability analysis is dominated by finding the MWMs of the graphs discussed above. For each graph, the MWM can be found in time $O(V^2E) = O(V^4)$ [25]. If all tasks are SMT-eligible, V is equal to $n + 1$ for $G1$ and n for $G2$ and each $G3_i$ graph. Testing schedulability requires one graph each for $G1$ and $G2$, plus up to n graphs for $G3_i$. ■

IV. SCHEDULABILITY STUDIES

To evaluate our approach, we conducted a schedulability study in which we created 245 scheduling scenarios. For each scenario, we created 2000 synthetic task systems and then used Thm. 1 to evaluate the schedulability of each system. All code used in this study is available in [26].

A. Experimental Setup

Each scenario was defined by a per-task utilization range, SMT interaction model, and a rule for designating which tasks would and would not use SMT.

Per-task utilization. Per-task utilizations were randomly drawn from the uniform ranges with midpoints 0.025, 0.05, 0.10, 0.15, and 0.20. For each midpoint, there were two distributions, *narrow* and *wide*. The narrow distributions had a minimum equal to 80% of the midpoint and a maximum equal to 120% of the midpoint; the wide distributions had minimums and maximums equal to 40% and 160% of the midpoint, respectively. Both wide and narrow distributions are included to build on previous findings [10] that range of per-task utilization is as important as average per-task utilization. As all tasks use a common period, specifying costs would not add to the results; if a period were specified, defining a task's utilization would also define its cost as $C_i = T \cdot u_i$.

Modeling SMT Costs. Our modeling of SMT costs is based on observations of the TACLeBench [27] sequential benchmarks made in [21]. For every pair of tasks τ_i and τ_k , we defined $C_{i(k)}$ as a function of C_i , C_k , and a *pair multithreading score* $M_{i(k)}$.

Def. 12. *Given tasks τ_i and τ_k , let the pair multithreading score $M_{i(k)}$ be defined such that*

$$C_{i(k)} = C_i + M_{i(k)} \cdot \min(C_i, C_k). \quad \blacktriangleleft \quad (5)$$

$M_{i(k)}$ can be thought of as how much τ_k increases the cost of τ_i for each instant the jobs are co-scheduled. If $M_{i(k)}$ is

close to zero, the co-scheduling jobs of τ_i and τ_k will cause only a small increase to the cost of $\tau_{i,k}$. If $M_{i(k)} > 1$ holds, then co-scheduling jobs of τ_i and τ_k will increase the cost of τ_i by more than the cost of τ_k .

$M_{i(k)}$ is itself a function of the *task multithreading score* M_i , which quantifies the effect of SMT on τ_i independently of the how τ_i is co-scheduled. For each task τ_i , we first determine M_i by drawing a value from the exponential distribution, which has the probability distribution function $f(M_i, \beta) = \frac{1}{\beta} \cdot e^{-\frac{M_i}{\beta}}$, giving $E[M_i] = \beta$. For β we use the values 0.35 (optimistic), 0.55 (medium), and 0.75 (pessimistic).

In the *low-variance* approach, $M_{i(k)} = M_i$ holds for all tasks τ_k . This approach models SMT having a constant effect on each task τ_i regardless of the identity of its partner τ_k .

In the *high-variance* approach, we take the additional step of determining $M_{i(k)}$ as a value drawn from the exponential distribution with $\beta = M_i$ where M_i is first determined as described above. This approach allows for τ_k to have a greater impact on the value of $C_{i(k)}$. Additional motivations for using these approaches, and further details on applying them, are available in [21].

Scenario creation. Each scenario included task systems with total utilizations ranging from 1 to 1.5. 100 task systems were created for each 0.025 size utilization interval (i.e. $[1, 1.025)$, $(1.025, 1.05]$, etc.). Within each system, tasks were created by first drawing a utilization u_i from the scenario's utilization distribution until the total utilization fell in the desired range, second determining the solo cost as C_i as $u_i \cdot T$, third determining $M_{i(k)}$ by one of the methods described above, fourth applying (5) to set costs with SMT, and finally defining $C_{i:k}$ per Def. 3.

SMT-eligibility. A task τ_i was SMT-eligible only if using SMT would increase its cost by at most a threshold value h , i.e., $C_{i(k)} \leq C_i \cdot h$ held for all tasks τ_k not previously determined to be SMT-ineligible; this choice is implemented in the code included in [26]. For h , we used 1.25, 1.5, 2, and ∞ . $h = \infty$ means that SMT will always be allowed and gave very poor results, showing the importance of support for SMT-ineligible tasks. All graphs presented here use $h = 1.5$, which generally produced better results than other tested h values.

B. Results

To summarize scenarios, we created graphs showing utilization intervals on the horizontal axis and the *schedulability ratio*—the proportion of systems schedulable within each utilization interval—on the vertical axis; each point on a graph shows how many systems within a single utilization interval were schedulable. Error bars show the 95% confidence interval as calculated using the Wilson score interval [28]. Recall that without SMT, no task system with $U > 1$ is schedulable on a single core. Our graphs have $U = 1.0$ as the left side of the horizontal axis; even the weakest of our schedulability curves indicate schedulability that would be impossible without SMT.

We show a subset of our graphs that highlight key strengths and weaknesses of here. The full set of graphs is in [26].

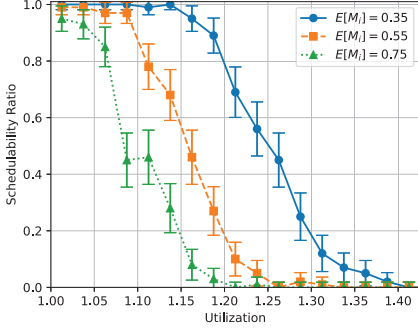


Fig. 1: util. $U(0.04, 0.06]$, low-variance

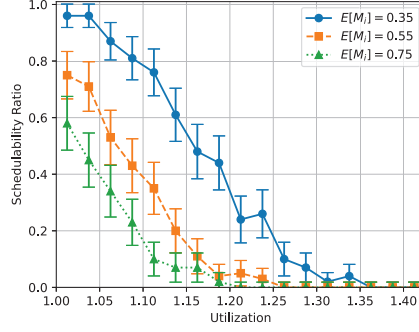


Fig. 2: util. $U(0.08, 0.12]$, low-variance

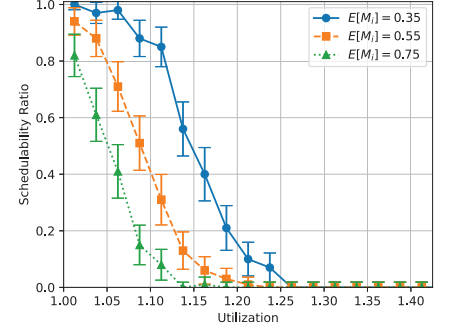


Fig. 3: util. $U(0.02, 0.08]$, low-variance

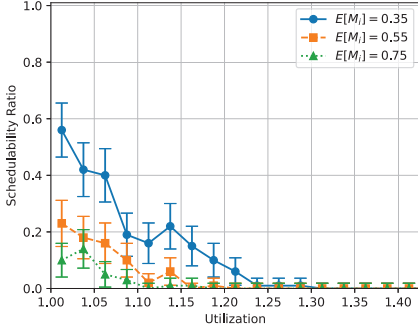


Fig. 4: util. $U(0.16, 0.24]$, low-variance

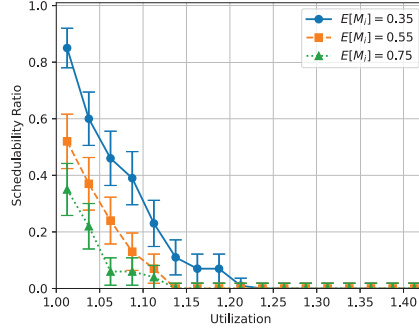


Fig. 5: util. $U(0.08, 0.12]$, high-variance

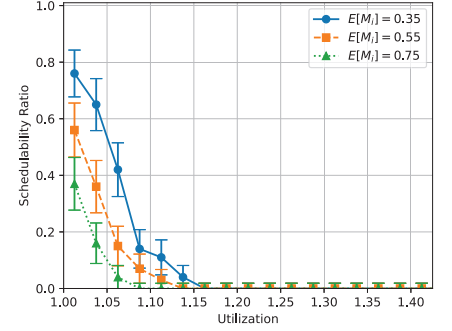


Fig. 6: util. $U(0.02, 0.08]$, high-variance

Obs. 1. The best scenarios used the low-variance SMT model and low per-task utilizations drawn from the narrow range, as in Fig. 1. Here, we scheduled over 80% of task systems with total utilization of 1.2 given $E[M_i] = 0.35$ (optimistic SMT behavior) and almost 50% of task systems with utilization of 1.10 with $E[M_i] = 0.75$ (pessimistic SMT behavior).³

Obs. 2. Increasing the expected per-task utilization while still using the narrow range, or using the wide range with the same expected per-task utilization, both decreased the effectiveness of SMT, as seen in Figs. 2 and 3.

Obs. 3. Scheduling was less effective when using the high-variance approach to SMT. See Figs. 5 and 6, which use the high-variance model but otherwise have identical parameters to Figs. 2 and 3, with which they are vertically aligned.

Obs. 4. There were benefits with average per-task utilizations of 0.20, provided task utilizations fell within a narrow range and the low-variance method was used; see Fig. 4. Systems with higher per-task utilization, or per task utilizations of 0.20 within a wide range, generally saw little benefit from using SMT.

Even when the majority of tasks in τ could benefit from SMT, rare tasks that are unfavorable for SMT (e.g., high utilization, $C_{i(k)}$ much higher than C_i) harm schedulability. Work on isolating harmful tasks may be helpful in the future.

Threats to validity and limitations. Our schedulability studies are based on synthetic data; successes observed here

³We observed stronger results with the range $[0.02, 0.03]$, but do not wish to over-focus on the lightest tasks.

may not transfer to systems where SMT's effects on costs differ too much from our model. Additionally, single-period systems are rare in practice; enhancing our work to support multi-period systems would allow for many more use cases.

V. CONCLUSIONS

We have shown that under our assumptions, SMT can be used to increase the schedulability of HRT task systems without compromising deadline guarantees. Our approach works best on systems of light-weight tasks where task utilizations fall within a narrow range, but struggles for systems of heavier or more varied tasks.

In future work, we plan to enable support for multi-period systems by partitioning tasks into same-period subsystems. The simplest approach would be to schedule each subsystem on a different core. A more complex approach would be to schedule tasks of each subsystem on a shared core using a variation of Alg. 1 in which each set of common-period tasks was allowed to form pairs only with tasks within the same period. From the perspective of each subsystem, all tasks belonging to other subsystems would appear as SMT-ineligible tasks. This approach would require a scheduling algorithm with more support than Alg. 1 for tasks with statically different priorities, thus complicating the scheduling analysis.

In addition, we wish to explore more effective means for determining which tasks should and should not use SMT and to expand our work to better support systems with wider ranges of per-task utilizations.

REFERENCES

- [1] R. Jain, C. Hughes, and S. Adve, "Soft real-time scheduling on simultaneous multithreaded processors," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2002.
- [2] J. Bakita, S. Osborne, S. Ahmed, S. Tang, J. Chen, F. Smith, and J. Anderson, "Simultaneous multithreading in mixed-criticality real-time systems," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, 2021.
- [3] S. Osborne and J. Anderson, "Simultaneous multithreading and hard real time: Can it be safe?" in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2020.
- [4] S. Osborne, S. Ahmed, S. Nandi, and J. Anderson, "Exploiting simultaneous multithreading in priority-driven hard real-time systems," in *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2020.
- [5] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis, "A comprehensive survey of industry practice in real-time systems," *Real-Time Systems*, vol. 58, no. 3, pp. 358–398, 2022.
- [6] J. Leung and M. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," *Information Processing Letters*, vol. 11, no. 3, pp. 115–118, 1980. [Online]. Available: [https://doi.org/10.1016/0020-0190\(80\)90123-4](https://doi.org/10.1016/0020-0190(80)90123-4)
- [7] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Massachusetts Institute of Technology, 1983, <https://dspace.mit.edu/bitstream/handle/1721.1/15670/10728774-MIT.pdf>.
- [8] J. Bulpin, "Operating system support for simultaneous multithreaded processors," Ph.D. dissertation, University of Cambridge, King's College, 2005, <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-619.pdf>.
- [9] J. Bulpin and I. Pratt, "Multiprogramming performance of the Pentium 4 with hyperthreading," in *Third Annual Workshop on Duplicating, Deconstruction and Debunking*, 2004.
- [10] S. Osborne, J. Bakita, and J. Anderson, "Simultaneous multithreading applied to real time," in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2019.
- [11] D. Tullsen, S. Eggers, and H. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings of the International Symposium on Computer Architecture*, 1995.
- [12] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm, "Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor," *SIGARCH Computer Architecture News*, vol. 24, no. 2, p. 191–202, may 1996. [Online]. Available: <https://doi.org/10.1145/232974.232993>
- [13] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen, "Simultaneous multithreading: a platform for next-generation processors," *IEEE Micro*, vol. 17, no. 5, 1997.
- [14] F. J. Cazorla, P. M. W. Knijnenburg, R. Sakellariou, E. Fernandez, A. Ramirez, and M. Valero, "Predictable performance in SMT processors: synergy between the OS and SMTs," *IEEE Transactions on Computers*, vol. 55, no. 7, pp. 785–799, July 2006.
- [15] S. Lo, K. Lam, and T. Kuo, "Real-time task scheduling for SMT systems," in *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2005.
- [16] J. Mische, S. Uhrig, F. Kluge, and T. Ungerer, "Using SMT to hide context switch times of large real-time tasks," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, 2010.
- [17] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "FlexPRET: A processor platform for mixed-criticality systems," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, 2014.
- [18] T. Gomes, S. Pinto, P. Garcia, and A. Tavares, "RT-SHADOWS: Real-time system hardware for agnostic and deterministic OSes within soft-core," in *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation*, 2015.
- [19] T. Gomes, P. Garcia, S. Pinto, J. Monteiro, and A. Tavares, "Bringing hardware multithreading to the real-time domain," *IEEE Embedded Systems Letters*, vol. 8, no. 1, 2016.
- [20] S. Osborne, J. Bakita, J. Chen, T. Yandrofski, and J. Anderson, "Minimizing DAG utilization by exploiting SMT," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, 2022.
- [21] S. Osborne, "Using simultaneous multithreading to support real-time scheduling," Ph.D. dissertation, UNC Chapel Hill, 2023, <https://www.cs.unc.edu/~anderson/diss/simsdiss.pdf>.
- [22] A. Pi, X. Zhou, and C. Xu, "Holmes: SMT interference diagnosis and CPU scheduling for job co-location," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, 2022.
- [23] A. Fog, "The microarchitecture of Intel, AMD, and VIA CPUs: an optimization guide for assembly programmers and compiler makers," 2018, available at <https://www.agner.org/optimize/microarchitecture.pdf>.
- [24] R. I. Davis and A. Burns, "Response time upper bounds for fixed priority real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2008.
- [25] J. Edmonds, "Maximum matching and a polyhedron with 0, 1-vertices," *Journal of research of the National Bureau of Standards—Mathematics and Mathematical Physics*, vol. 69, no. 125-130, pp. 55–56, 1965.
- [26] S. Osborne, "Simultaneous Multithreading and Common-Period Sporadic Tasks in Hard Real-Time: Additional Materials," 2025. [Online]. Available: <https://github.com/shosborn/SMT-DATE26>
- [27] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research," in *Proceedings of the International Workshop on Worst-Case Execution Time Analysis*, 2016.
- [28] E. B. Wilson, "Probable inference, the law of succession, and statistical inference," *Journal of the American Statistical Association*, vol. 22, no. 158, pp. 209–212, 1927.