OpenVX and Real-Time Certification: The Troublesome History

Tanya Amert, Sergey Voronov, and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill

***Abstract*—Many computer-vision (CV) applications used in autonomous vehicles rely on historical results, which introduce cycles in processing graphs. However, existing response-time analysis breaks down in the presence of cycles, either by failing completely or by drastically sacrificing parallelism or CV accuracy. To address this situation, this paper presents a new graph-based task model, based on the recently ratified OpenVX standard, that includes historical requirements and their induced cycles as first-class concepts. Using this model, response-time bounds for graphs that may contain cycles are derived. These bounds expose a tradeoff between responsiveness and CV accuracy that hinges on the extent of allowed parallelism. This tradeoff is illustrated via a CV case study involving pedestrian tracking. In this case study, the methods proposed in this paper enabled significant improvements in both analytical and observed response times, with acceptable CV accuracy, compared to prior methods.**

I. Introduction

In semi- or fully autonomous advanced driver-assist systems (ADASs), computer-vision (CV) algorithms are often used to provide much of the safety-critical sensor-based processing. To facilitate the development of these algorithms, the OpenVX standard was ratified in 2014 [37]. OpenVX, which specifically targets heterogeneous embedded hardware, allows programmers to specify CV algorithms as dataflow graphs by interconnecting high-level CV primitives. While such an approach eases the design of CV algorithms, the OpenVX API has a glaring omission: it completely ignores real-time concerns. This omission has led to recent work directed at applying real-time scheduling principles to OpenVX graphs and producing response-time bounds for such graphs [14, 42–44].

Unfortunately, prior OpenVX response-time analysis breaks down in the presence of cycles, either by failing completely or by drastically sacrificing parallelism or CV accuracy. This is a critical shortcoming, because actual ADAS CV processing graphs often have cycles due to historical dependencies. For example, pedestrian tracking entails predicting future pedestrian positions from their prior trajectories. In order to be able to certify CV applications as used in ADASs, response-time analysis for cyclic OpenVX graphs is needed. *If this problem is not addressed, these workloads cannot be certified.*

In this paper, we address this problem by presenting the first ever response-time analysis for cyclic OpenVX graphs that does not require conservative methods that obviate cycles in simplistic ways at the price of degrading CV accuracy or schedulability. Our work specifically targets multicore platforms augmented with graphics processing units (GPUs)—arguably the most commonly considered type of hardware platform in work involving OpenVX. Before describing what we mean by "conservative methods," and how we avoid them, we first provide an overview of prior work.

Prior work. A number of methods exist for modeling dataflow applications [5, 7, 8, 21, 22, 36, 40]. Generally, these methods specify computations as processing graphs, with tasks corresponding to graph nodes, and edges indicating precedence relationships between tasks. The real-time scheduling and analysis of such graphs, both on uniprocessors and multiprocessors, has been extensively studied; representative publications include [1–4, 11, 13, 14, 17–20, 25–29, 31–35, 39, 41–44].

Of the just-cited papers, three [14, 42, 44] warrant further scrutiny: they are the only ones to consider OpenVX graphs, and one of them [42] is the only prior work to consider response-time bounds for cyclic multicore graphs. Two of these papers, by Elliott *et al.* [14] and by K. Yang *et al.* [42], are companion papers, focusing on implementation and analysis, respectively. K. Yang *et al.* proposed two techniques for breaking cycles. First, they noted that any back edge in a graph that feeds history information to its target task that is so "old" that cycle-oblivious real-time scheduling ensures the precedence constraint anyway can simply be removed. However, in any CV algorithm that provides reasonable accuracy, such "old" history information would likely be of little use. Second, they showed that a given cycle can be broken by combining all of its nodes into a single sequential *supernode*. This technique can be applied to convert any OpenVX graph into a DAG. K. Yang *et al.* showed that a response-time bound can be computed for such a DAG by transforming it to an "equivalent" set of independent sporadic tasks, as done in earlier work by Liu and Anderson on DAGs generally [25]. However, this transformation process requires the utilization of each node (*i.e.*, task) to be at most 1.0, a restriction that can be easily violated by a supernode.

More recently, M. Yang *et al.* [44] proposed altering the transformation process above by converting to a sporadic task set that allows *intra-task parallelism* (*i.e.*, multiple jobs of the same task may execute concurrently), as done in earlier work [15, 43] not pertaining to OpenVX. M. Yang *et al.* showed that such parallelism enables much lower response-time bounds for OpenVX graphs. However, parallel node execution breaks the supernode idea, so they expressly considered cycles to be out of scope.

Contributions. We extend the prior transformation-based methods discussed above [14, 42, 44] to enable the real-time

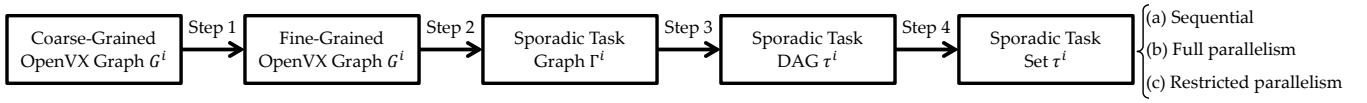

Fig. 1: The transformation from an OpenVX graph to a sporadic task set; different models of sporadic task sets provide varying intra-task parallelism (labeled (a)-(c)). Steps 1, 2-3, and 4 come from [44], [42], and [32], respectively.

certification of arbitrary OpenVX graphs on multicore+GPU platforms. We make three key contributions.

First, we extend the transformation process of M. Yang *et al.* [44] to deal with cyclic graphs. Our key insight here is based upon a property of the sporadic task model with intra-task parallelism: under this model, per-task response-time bounds can be computed *without requiring task utilizations to be at most 1.0*. This fact suggests a way forward for handling arbitrary supernodes. While (as noted earlier) parallelism breaks the supernode idea, we show that it can be allowed if back edges can supply slightly older history information. In fact, we will show that, for any schedulable system of graphs, the degree of parallelism that can be allowed, the age of history information, and the response-time bounds that can be guaranteed are all closely linked. Loosely speaking, older history information allows for increased parallelism and lower response-time bounds; insisting on the most recent possible history information can kill parallelism and result in an unschedulable graph. *The designers of CV algorithms should be aware of these tradeoffs when constucting OpenVX graphs.* In particular, they should set history age requirements so that both CV accuracy and response-time bounds are acceptable.

From a schedulability point of view, setting history age requirements equates to specifying an allowed degree of parallelism in processing a cycle. Thus, we need as the end point of our transformation process a sporadic task model wherein the allowed intra-task parallelism is a per-task *settable parameter*. Our second key contribution involves defining such a task model, namely the *rp-sporadic task model* (restricted parallelism), and presenting response-time analysis for it.

Though analytically interesting, it remains to be seen whether the parallelism/accuracy/response-time tradeoffs enabled by our work are worthwhile to consider from the perspective of a CV algorithm designer. As a final contribution, we present an assessment of this issue via a case study involving pedestrian tracking. In this study, we consider an OpenVX graph that is actually unschedulable as originally specified and show the effects of increasing parallelism. We found that we were able to bound response times for this graph if intra-task parallelism is enabled, with only a minor accuracy drop compared to the original unschedulable graph (which has the highest accuracy but unbounded response times).

Generality. Although we focus on OpenVX as our motivation, the rp-sporadic task model and the derived response-time analysis are applicable to any application that can be specified as a sporadic task graph containing cycles, as Fig. 1 (discussed in Sec. II) implies. Such graphs may arise in many contexts, such as control, motion planning, and recurrent neural networks; if the utilization of a cycle is greater than 1.0, prior work cannot provide response-time bounds for these graphs.

Organization. In the rest of this paper, we describe our new transformation process (Sec. II), present the rp-sporadic task model (Sec. III) and response-time analysis under it (Sec. IV), discuss our case study (Sec. V), and conclude (Sec. VI).

II. Transformation Process

Prior work has shown how to transform an OpenVX graph into an "equivalent" set of independent sporadic tasks [32, 42, 44], for which response-time analysis exists [10, 15, 16, 23–25]. This process is depicted in Fig. 1. However, Step 3, as originally proposed [42], requires that the utilization of each cycle is at most 1.0.

In this section, we illustrate the existing transformation steps and discuss the implications of no or full intra-task parallelism (choices (a) and (b) in Fig. 1). We then describe how we augment Steps 2-4 to allow restricted parallelism, enabling this approach for graphs containing cycles of any utilization.

A. OpenVX

In OpenVX, primitives and the data objects upon which they operate comprise a bipartite graph [38]. An OpenVX graph $\mathcal{G}^i$ contains data objects $D^i_1, \ldots, D^i_{y_i}$ and nodes $N^i_1, \ldots, N^i_{z_i}$. An edge $\left(N^i_v, D^i_w\right)$ corresponds to a data object D^i_w that is written by node N^i_v, and $\left(D^i_w, N^i_v\right)$ corresponds to a data object read by node N^i_v. Data objects can optionally be *delay objects*, indicating that the data from prior time steps must be buffered for later use. Associated with each delay object is a value indicating the age, in time steps, of the data.

To simplify analysis, we assume that each graph has a single source node and a single sink node. (If this is not the case, then a single "virtual" source and/or sink can be added.) For all graphs we consider, we assume that the first indexed node (N^i_1 for an OpenVX graph $\mathcal{G}^i$) is the source.

Ex. 1. An example OpenVX graph $\mathcal{G}^1$ is shown in Fig. 2. In this figure, rectangles correspond to data objects, $D^1_1, \ldots, D^1_8$, and round nodes indicate primitives, $N^1_1, \ldots, N^1_4$, that act on them. There are three delay objects, D^1_3, D^1_5, and D^1_6, with delay values 1, 3, and 2, respectively. $\Diamond$

The OpenVX standard specifies a series of rules for processing graphs [38]. The rules relevant to our work are:

1) *Single Writer*: Every data object has at most one incoming edge.
2) *Broken Cycles*: Every cycle in $\mathcal{G}^i$ must contain at least one input edge $\left(D^i_w, N^i_v\right)$ where D^i_w is a delay object.

Ex. 1 (cont'd). In $\mathcal{G}^1$, every data object has a single incoming edge (although D^1_2 has two outgoing edges). Additionally, there are two cycles, containing edges from delay objects D^1_5 and D^1_6 to node N^1_3. $\Diamond$

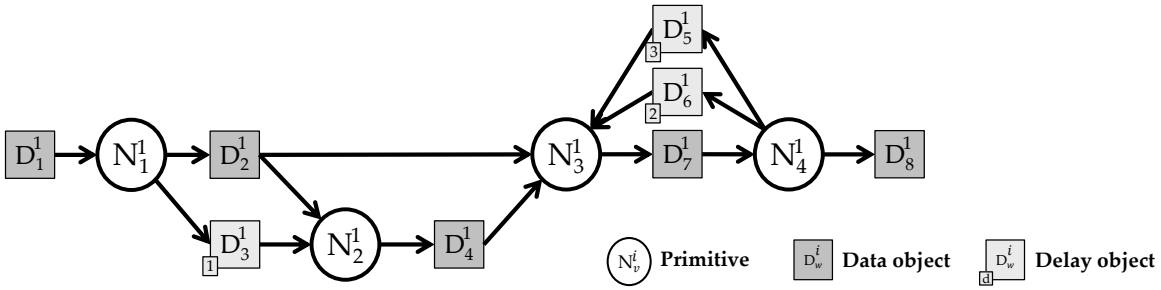

Fig. 2: An OpenVX graph $\mathcal{G}^1$ of four primitives and eight data objects, including three delay objects. Delay values are inset in the delay object boxes.

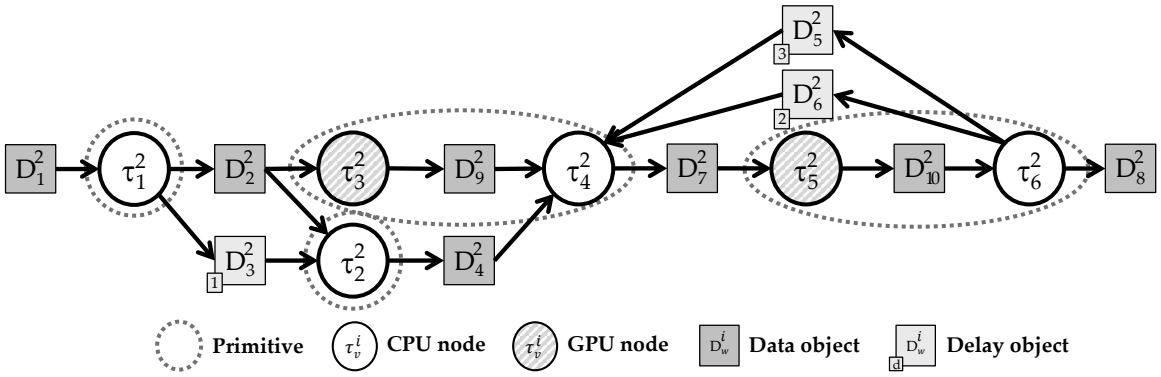

Fig. 3: A fine-grained OpenVX graph $\mathcal{G}^2$ corresponding to the coarse-grained graph in Fig. 2. $\mathcal{G}^2$ contains six nodes (four CPU nodes and two GPU nodes) and ten data objects. N_3^1 and N_4^1 have each been expanded to separate CPU and GPU nodes, and new data objects have been added.

B. Transforming OpenVX Graphs to Sporadic Task Sets

The transformation process depicted in Fig. 1 must be performed for each OpenVX graph $\mathcal{G}^i$ in a system. We now illustrate each step in detail.

Step 1: From a coarse- to a fine-grained OpenVX graph. The OpenVX standard specifies little about the concurrent execution of primitives within a graph. M. Yang *et al.* [44] showed that treating each primitive as a schedulable entity is often too coarse-grained to guarantee bounded response times. Rather, primitives should be split into multiple nodes, with each executing on either a CPU or a GPU.[1]

Ex. 2. We illustrate the transformation process with a continuing example. Fig. 3 depicts a fine-grained OpenVX graph $\mathcal{G}^2$ corresponding to the coarse-grained OpenVX graph $\mathcal{G}^1$ from Fig. 2. Primitives N_3^1 and N_4^1 have been decomposed into nodes $\{\tau_3^2, \tau_4^2\}$ and $\{\tau_5^2, \tau_6^2\}$, respectively, with additional data objects D_9^2 and D_{10}^2 added between the new nodes. Additionally, each node in Fig. 3 is shaded based on whether that node executes on a CPU or a GPU. ◊

Step 2: From a fine-grained OpenVX graph to a sporadic task graph. A sporadic task graph Γ^i is comprised of z_i nodes, $\tau_1^i, \ldots, \tau_{z_i}^i$, with each node corresponding to a task. A task τ_v^i releases a potentially infinite sequence of jobs $J_{v,1}^i, J_{v,2}^i, \ldots$. Edges in Γ^i indicate producer/consumer relationships between tasks: a job must wait to begin execution until the corresponding job of each task from which it consumes data (*i.e.*, for each edge for which it is a consumer) has completed.

Given a fine-grained OpenVX graph $\mathcal{G}^i$, we can perform a simple transformation to obtain a sporadic task graph Γ^i:

- Each node τ_v^i in $\mathcal{G}^i$ becomes a node τ_v^i in Γ^i.
- Each input edge $\left(D_w^i, N_v^i\right)$ other than that into the source τ_1^i becomes a directed edge $\left(\tau_u^i, \tau_v^i\right)$, where τ_u^i is the single writer of data object D_w^i.
- An edge is a *delay edge* if its corresponding data object D_w^i is a delay object, and a *regular edge* otherwise.
- Multiple edges of the same type between the same pair of nodes are merged into a single edge of that type.

Note that delay edges can be either forward or backward edges, depending on whether they result in a cycle in the graph. For each delay edge $\left(\tau_v^i, \tau_u^i\right)$, we include a range $[p, q]$,[2] $p \leq q$, corresponding to the range of delay values for that edge. Thus, a delay edge $\left(\tau_v^i, \tau_u^i\right)$ with range $[p, q]$ indicates that a job $J_{u,j}^i$ relies on the outputs of $\{J_{v,j-q}^i, \ldots, J_{v,j-p}^i\}$.

Ex. 2 (cont'd). Fig. 4 shows the sporadic task graph Γ^2 corresponding to the fine-grained OpenVX graph from Fig. 3. The three delay objects are represented here as two delay edges, one forward and one backward. The delay values are encapsulated in the p and q values for the delay edges. ◊

[1] We assume the mapping of primitives to processor types is decided by the application designer.

[2] For simplicity of notation, we will omit subscripts and superscripts for delay edges' ranges when the edge in question is clear.

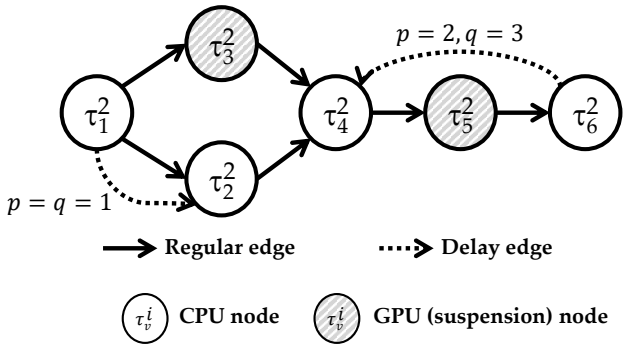

Fig. 4: A sporadic task graph Γ^2 derived from the fine-grained OpenVX graph from Fig. 3.

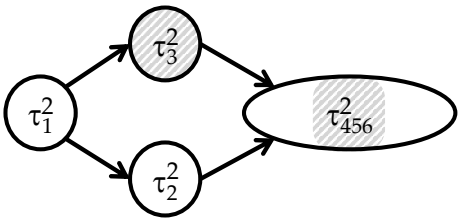

Fig. 5: A sporadic task DAG τ^2 derived from the cyclic graph from Fig. 4.

Step 3: From a sporadic task graph to a sporadic task DAG. K. Yang *et al.* [42] provided a series of rules for removing delay edges from graphs, resulting in a DAG. They showed that forward delay edges can simply be replaced by regular edges, and they proposed to break cycles by combining all nodes in a given cycle in a graph into a single *supernode*.

Ex. 2 (cont'd). Fig. 5 shows the DAG τ^2 derived from the cyclic graph Γ^2 in Fig. 4. The forward delay edge from τ_1^2 to τ_2^2 has been removed because a regular edge between these nodes already exists, and nodes τ_4^2, τ_5^2, and τ_6^2 comprising the cycle have been combined into a single supernode τ_{456}^2. ◊

Step 4: From a sporadic task DAG to a sporadic task set. Given a sporadic task DAG τ^i, it is straightforward to consider each node as an independent sporadic task. Each task τ_v^i has a worst-case execution time given by C_v^i and a relative deadline given by D_v^i. All tasks belonging to τ^i share a period T^i. Jobs of the source task τ_1^i are assumed to be released sporadically, at least T^i time units apart. For non-source tasks, Liu *et al.* [32] showed how response-time bounds R_u^i (explained in detail below) of tasks τ_u^i that produce data consumed by τ_v^i can be used to set an offset Φ_v^i. This offset specifies the release time of a job $J_{v,j}^i$ relative to that of its graph's corresponding source job $J_{1,j}^i$.[3] Note that task deadlines are used here to define priorities rather than strict (hard) timing constraints, so R_v^i may exceed D_v^i, *i.e.*, jobs may complete after their deadlines.

Ex. 2 (cont'd). Fig. 6 depicts an example schedule for the task set derived from the sporadic task DAG τ^2 in Fig. 5. In this example, we assume response-time bounds of the four DAG tasks have been computed to be $R_1^2 = 9$, $R_2^2 = 7$, $R_3^2 = 5$, and $R_{456}^2 = 9$. As described in [32], $\Phi_1^2 = 0$, $\Phi_2^2 = \Phi_3^2 = R_1^2 = 9$, and $\Phi_{456}^2 = \max\{\Phi_2^2 + R_2^2, \Phi_3^2 + R_3^2\} = 16$. ◊

[3]Liu *et al.* [32] also showed that early releasing [9] can be used to improve response times by releasing a job as soon as its consumed data is available, potentially before its actual release time, as long as its deadline is unchanged.

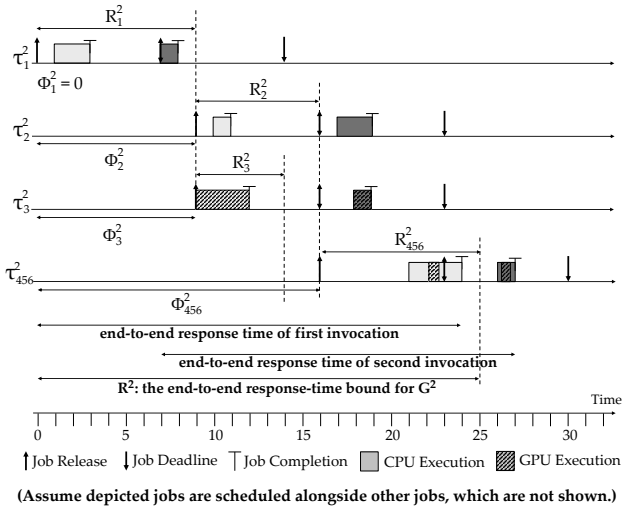

Fig. 6: A possible schedule of two sets of jobs of the sporadic tasks in τ^2. The second job of each task is shaded darker than the first.

We define the *utilization* of τ_v^i to be $u_v^i = C_v^i/T^i$. The utilization of the entire system is given by $U = \sum_{\tau^i \in \tau} \sum_{\tau_v^i \in \tau^i} u_v^i$. We can define the utilization of a cycle similarly: $\sum_{\tau_v^i \in \tau'} u_v^i$, where τ' is the set of tasks in the cycle.

C. Response-Time Analysis

For a job $J_{v,j}^i$ of task τ_v^i, let $r_{v,j}^i$ denote its release time and let $f_{v,j}^i$ denote its completion time (or *finish* time). We define $J_{v,j}^i$'s *response time* as $f_{v,j}^i - r_{v,j}^i$ and the *end-to-end response time* of a sporadic task graph Γ^i as $\max_j\{f_{z_i,j}^i - r_{1,j}^i\}$.

We seek to calculate a response-time bound R_v^i for each task τ_v^i. Such bounds can be propagated back to the original graph(s) to give end-to-end response-time bounds of all graphs. The available response-time analysis depends upon the choice of parallelism in the sporadic task model.

Existing sporadic task models. The conventional sporadic task model requires jobs of the same task to execute sequentially, *i.e.*, a job $J_{v,j}^i$, $j \geq 2$, is not ready unless $J_{v,j-1}^i$ has completed execution. This model has been the subject of much prior work on response-time analysis under global schedulers [10, 16, 23, 24], which will be our focus here.

Ex. 3. Fig. 7 depicts possible schedules for jobs of τ_{456}^2 from Fig. 5 on a platform with four CPUs and one GPU, assuming $T^2 = 5$, $C_{456}^2 = 6$, and $R_{456}^2 = 21$. The schedule begins at time 100, when job $J_{456,21}^2$ is released.

In schedule (a), the jobs execute sequentially. Due to jobs of other tasks (not shown), $J_{456,21}^2$ is not scheduled until time 114. This postponement impacts the subsequent jobs; $J_{456,24}^2$ has a response time of 7.4. However, the $p = 2$ requirement is met, *e.g.*, $J_{456,21}^2$ completes before $J_{456,23}^2$ begins. ◊

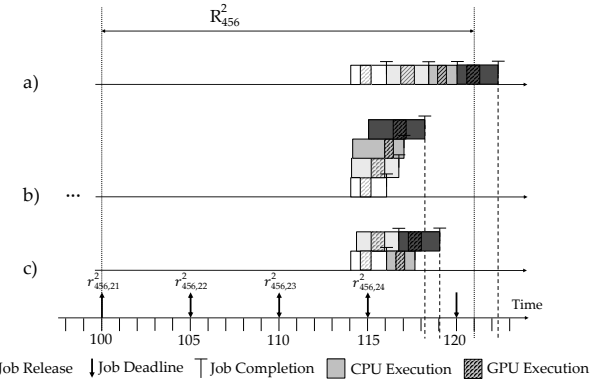

Fig. 7: Scheduling repercussions of the degree of intra-task parallelism, assuming GPU computations are FIFO scheduled on a single GPU. Successive jobs $J^2_{456,21}$, $J^2_{456,22}$, $J^2_{456,23}$, and $J^2_{456,24}$ are shaded progressively darker.

Later work considered a model that allows full intra-task parallelism, *i.e.*, any number of unfinished jobs of the same task may execute concurrently. This model enables much smaller response-time bounds to be ensured [15].

Ex. 3 (cont'd). Schedule (b) in Fig. 7 shows the result of allowing full intra-task parallelism. We assume GPU computations are FIFO scheduled, which causes three of the four jobs' execution times to increase. However, the response time of $J^2_{456,24}$ is reduced to 3.2 time units. ◊

Unfortunately, unrestricted intra-task parallelism creates two problems. First, the jobs of a task can complete out of order; however, this can be simply resolved by buffering job outputs, as discussed in [14]. Second, and more importantly, such parallelism can violate the dependencies required by backward delay edges. In fact, sequential execution was originally assumed for the transformation to a DAG (Step 3) [42]. Theorem 3 in [42] showed that if $p = 1$ for some backward delay edge, then no two jobs of any task in that cycle may execute in parallel. This proof can be generalized to show that if more than p jobs of a task in a cycle execute concurrently, then a precedence constraint must be violated.

Ex. 3 (cont'd). The supernode τ^2_{456} was created from a cycle with $p = 2$. Thus, job $J^2_{456,23}$ requires output from job $J^2_{456,21}$. However, in schedule (b) of Fig. 7, jobs $J^2_{456,21}$ and $J^2_{456,23}$ execute concurrently, violating this precedence constraint. ◊

The troublesome history. Response-time analysis for sequential sporadic tasks requires $u^i_v \leq 1.0$ for all tasks. This requirement extends to supernodes in [42]: the utilization of each cycle must be at most 1.0. However, if smaller bounds are desired or if the cycle has higher utilization, no existing analysis can be applied. Furthermore, cycles with utilization exeeding 1.0 can easily occur in actual CV graphs. When full intra-task parallelism is enabled, $u^i_v \leq 1.0$ is no longer required, but historical requirements may not be met.

Ex. 3 (cont'd). If jobs execute sequentially as in Fig. 7(a), response times can be unbounded for τ^2_{456}, as $u^2_{456} = 6/5$. ◊

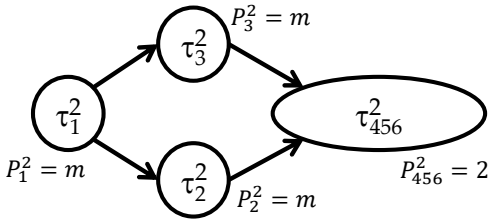

Fig. 8: Intra-task parallelism for nodes of τ^2 from Fig. 5.

D. A New Hybrid Approach

Our work bridges this parallelism divide, resulting in response-time bounds for sporadic task graphs (and thus OpenVX graphs) that prior work deemed infeasible. We provide a new restricted-parallelism sporadic task model that specifies intra-task parallelism on a per-task basis. A key feature of our approach is that *per-task utilizations are allowed to exceed 1.0, yet parallelism (and hence accuracy) is controlled.*

Ex. 3 (cont'd). Restricted intra-task parallelism is shown in schedule (c) of Fig. 7. The response time of $J^2_{456,24}$ is increased to 4.0, but the history requirements are respected, as only $p = 2$ jobs of τ^2_{456} execute concurrently. ◊

Abstracting GPU computations. Although M. Yang *et al.* [44] suggested considering CPU and GPU tasks separately in response-time analysis, their results hold only for DAGs. Instead, as in K. Yang *et al.* [42], we arbitrate access to the GPU with a locking protocol, such as GPUSync [12]. Thus, we henceforth assume that all graph nodes are CPU nodes, with their worst-case execution times inflated to include GPU blocking and execution time, and that tasks can contain non-preemptive regions due to said locking protocol. In Sec. IV-C, we briefly discuss the complications that arise in our setting if CPU access is not arbitrated via a locking protocol.

Transforming cycles, revisited. We leverage the supernode concept from [42] to transform a sporadic task graph Γ^i into a sporadic task DAG τ^i. We supplement each node τ^i_v of the DAG with a value P^i_v indicating the allowed intra-task parallelism for the jobs of that task. All tasks within a cycle are combined into a single supernode τ^i_u, with P^i_u defined to be the smallest p of any forward or backward delay edge contained in the cycle (we do not use q, as it is does not limit the parallelism of the cycle). A task τ^i_v that is not part of any cycle has $P^i_v = m$, the number of CPU processors, *i.e.*, unrestricted intra-task parallelism, as in [44].

Ex. 4. Fig. 8 depicts the DAG that results from our parallelism-aware supernode transformation.[4] The nodes correspond to those in Fig. 5, and are labeled with their intra-task parallelism values P^i_v. For tasks that are not supernodes, the intra-task parallelism is m. Task τ^2_{456} is a supernode derived from a cycle with $p = 2$ in Fig. 4, so it has $P^2_{456} = 2$. ◊

Offset computation for forward delay edges. In prior work, forward delay edges were either deemed as out of scope [44]

[4]Note that, while we ended up with the same number of compute nodes as in the original coarse-grained graph in Fig. 2, this will generally not be the case. We are somewhat constrained here to consider small graphs.

or supported assuming only sequential task execution [13, 42]. We propose a different method for handling such edges here.

Consider a forward delay edge (τ_v^i, τ_u^i) with delay value p. Denote the offset of τ_u^i computed in a DAG without the delay edge as $\Phi_u'^i$. The forward delay edge adds the requirement that a job $J_{u,j}^i$ must not start earlier than the completion of $J_{v,j-p}^i$, p DAG periods prior. Thus, we require $\Phi_u^i \geq \Phi_v^i + R_v^i - p \cdot T^i$. At the same time, we require $\Phi_u^i \geq \Phi_u'^i$. Combining both expressions, we have $\Phi_u^i = \max(\Phi_u'^i, \Phi_v^i + R_v^i - p \cdot T^i)$.

Note that, because offsets are determined from source to sink [32], by the definition of a forward delay, Φ_v^i is available when Φ_u^i is determined. Note also that the method above can be generalized for the case wherein forward delay edges are directed from several nodes to the node τ_u^i.

K. Yang *et al.* [42] proposed instead to replace each forward delay edge with a regular forward edge. Effectively, such a replacement is equivalent to the computation of Φ_u^i with $p = 0$, so our approach generalizes theirs.

To this point, we have explained how to adapt prior work to transform a coarse-grained OpenVX graph into an "equivalent" sporadic task set with restricted parallelism. What remains is to formally define this sporadic task-model variant and to derive response-time bounds under it. This we do next in Secs. III and IV, respectively.

III. The RP-Sporadic Task Model

We now introduce the *rp-sporadic task model*, which permits per-task allowed parallelism to be specified. Under this model, the i^{th} task is specified as $\tau_i = (\Phi_i, T_i, C_i, P_i)$, where Φ_i, T_i, C_i, and P_i are as defined in Sec. II (but omitting the graph index, as it is not relevant to us here). We assume that tasks have implicit deadlines, *i.e.*, $D_i = T_i$. We denote τ_i's utilization as $u_i = C_i/T_i$, total utilization as U, the j^{th} job of τ_i as $J_{i,j}$, its release time as $r_{i,j}$, its deadline as $d_{i,j} = r_{i,j} + T_i$, the maximal length of a single non-preemptive section as B_{max} (recall the earlier discussion about using locking protocols to arbitrate GPU access), and the maximal worst-case execution time (WCET) of any task as C_{max}.

Scheduler. We consider a platform with m CPUs (recall that, with GPU access arbitrated using locking protocols, we can focus on a CPU-only system in our analysis). Global earliest-deadline-first (G-EDF) scheduling guarantees bounded response times without undue utilization restrictions [10, 15], so we assume G-EDF scheduling with deadline ties broken arbitrarily but consistently (*e.g.*, by task index). We let $J_{i,j} \prec J_{k,l}$ denote that job $J_{i,j}$ has higher priority than job $J_{k,l}$.

Feasibility conditions. As in existing response-time analysis, we require $U \leq m$, or the entire system can become overutilized, with response times being unbounded. Additionally, at most P_i jobs of a task τ_i can execute at once, so we require

$$\forall i: \; u_i \leq P_i. \tag{1}$$

In particular, with τ_i restricted to execute on at most P_i processors at any time, if $u_i > P_i$ and τ_i releases jobs as early as possible, its response times will grow without bound.

IV. Response-Time Bounds

In this section, we prove that every task of a feasible rp-sporadic task set τ has bounded response times under G-EDF. In proving this result, we assume time to be continuous.

A. Basic Bound

Throughout this section, we consider a job of interest; as the proven response-time bound holds for any job of interest, it inductively applies to all jobs of all tasks in the task system. We consider an analysis window, and bound the amount of work that conflicts with the job of interest within this window. Initially, we consider a simpler edge case (Lemma 2). For the more complex case, we first show that non-preemptive sections of lower-priority jobs can affect the execution of higher-priority jobs only if such sections are scheduled at the start of the analysis window (Lemma 3). To bound the response time for the job of interest, we first bound the total workload of high-priority jobs given their maximal response times (Lemma 4). Then, we show that the inductively assumed response-time bounds of high-priority jobs ensure the same bound for the job of interest if it is big enough (Lemma 5). Finally, we present our full response-time theorem (Theorem 1) and its closed-form version (Corollary 1).

Def. 1. *At a time instant t, job $J_{i,j}$ is* unreleased *if $t < r_{i,j}$ and* released *otherwise; $J_{i,j}$ is* complete *if it is completed by t; $J_{i,j}$ is* pending *if it is released but not completed; and $J_{i,j}$ is* ready *if it is pending and job $J_{i,j-P_i}$ is complete (i.e., $J_{i,j}$ can be scheduled at t).*

Job of interest. We consider an arbitrary job $J_{k,l}$ of a task $\tau_k \in \tau$. Let t_d be the absolute deadline of $J_{k,l}$, *i.e.*, $t_d = r_{k,l} + T_k$. Let t_f be the completion time of $J_{k,l}$. We will show inductively with respect to $\prec$ that the response time of τ_k is bounded by $x + T_k + C_k$ for any positive x that is large enough (as formalized later in (9)). We assume $t_d \leq t_f$, for otherwise the response time of $J_{k,l}$ is less than T_k.

Def. 2. *We let Ψ (resp., $\overline{\Psi}$) denote the job set consisting of all jobs that have higher (resp., lower) priority than $J_{k,l}$.*

Def. 3. *We say that a time instant t is* busy *if m jobs of $\Psi \cup \{J_{k,l}\}$ are scheduled, or there is a ready job in $\Psi \cup \{J_{k,l}\}$ that is not scheduled at t, and* non-busy *otherwise. Both busy conditions imply that every processor executes a job. We say that a time interval $[t, t')$ is* busy *if all instants in it are busy.*

Lemma 1. *For any task τ_i the number of its ready jobs in $\Psi \cup \{J_{k,l}\}$ does not increase after t_d.*

Proof. All jobs in $\Psi \cup \{J_{k,l}\}$ are released within $[0, t_d]$. A pending job $J_{i,j}$ in this set can become ready after t_d only at the time instant when $J_{i,j-P_i}$ completes (and is no longer ready). Thus, the total number of ready jobs of τ_i in $\Psi \cup \{J_{k,l}\}$ does not increase after t_d. □

There are two cases for t_d: it is either a busy or a non-busy time instant. We will consider the non-busy case in Lemma 2 first and then the busy case in Lemmas 3–5.

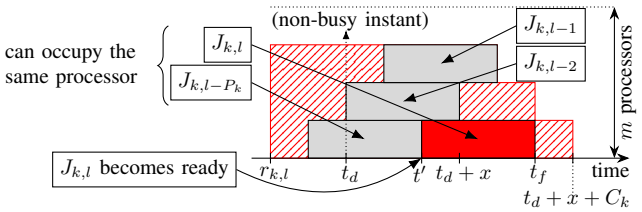

Fig. 9: Lemma 2 illustration ($P_k = 3$).

Lemma 2. *If t_d is a non-busy time instant, and the response time of each job of τ_k released before $J_{k,l}$ is at most $x+T_k+C_k$, then the response time of $J_{k,l}$ is bounded by $x + T_k + C_k$. (No conditions on x except $x \geq 0$ are implied in this lemma.)*

Proof. By Lemma 1, the number of ready jobs in $\Psi \cup \{J_{k,l}\}$ does not increase after t_d. Therefore, if t_d is not a busy time instant, then any later time instant is not busy, as jobs from $\Psi \cup \{J_{k,l}\}$ occupy fewer than m processors.

Thus, $J_{k,l}$ is scheduled at the first time instant $t' \geq t_d$ when it is ready. As shown in Fig. 9, if $t' > t_d$, then $J_{k,l}$ becomes ready upon completion of $J_{k,l-P_k}$, which was released by time $t_d - T_k - P_k \cdot T_k$. By the lemma statement, $J_{k,l-P_k}$ must complete by time $t_d - T_k - P_k \cdot T_k + x + C_k + T_k = t_d + x + C_k - P_k \cdot T_k$. By (1), $C_k \leq P_k \cdot T_k$, so $t' \leq t_d + x$. As $J_{k,l}$ is scheduled immediately upon becoming ready, it completes by time $t_d + x + C_k$, within $x + T_k + C_k$ time units from $r_{k,l}$. □

We now consider the case when t_d is busy.

Def. 4. *Let t_0 denote the first busy instant such that $[t_0, t_d)$ is a busy interval. Let t_b denote the last time instant such that $[t_d, t_b)$ is a busy interval.*

The following lemma limits the number of lower-priority jobs in $\overline{\Psi}$ that can affect the execution of higher-priority ones.

Lemma 3. *A non-preemptive section of a job $J_{i,j}$ in $\overline{\Psi}$ may block the execution of ready jobs in $\Psi \cup \{J_{k,l}\}$ within $[t_0, t_f)$ only if that section is scheduled at t_0. Moreover, such blocking may occur only within $[t_0, t_b)$.*

Proof. Consider the interval $[t_0, t_f)$, depicted in Fig. 10 for two cases, (a) $t_b > t_f$ and (b) $t_b \leq t_f$ (note that t_{av} is defined later in Lemma 5). We begin by showing, in both cases, that all time instants after t_b are non-busy. By Def. 3, at t_b, the at most $m-1$ ready jobs in $\Psi \cup \{J_{k,l}\}$ are scheduled. By Lemma 1, the number of ready jobs in $\Psi \cup \{J_{k,l}\}$ does not increase after t_d. Thus, if a job $J_{g,h} \in \Psi \cup \{J_{k,l}\}$ becomes ready at some time $t > t_d$, then $J_{g,h-P_g}$ must have completed, and the processor upon which it executed is available at t. Additionally, as jobs in $\Psi \cup \{J_{k,l}\}$ have higher priority than those in $\overline{\Psi}$, they remain scheduled until they complete, so no time instant after t_b is busy.

By Def. 3, if $J_{i,j} \in \overline{\Psi}$ blocks a job in $\Psi \cup \{J_{k,l}\}$ at $t' \in [t_0, t_f)$, then t' is a busy instant. As no time instant after t_b is busy, $t' \in [t_0, t_b)$. $J_{i,j}$ has lower priority than any job in $\Psi \cup \{J_{k,l}\}$, so it must therefore execute non-preemptively at every instant in $[t_0, t']$, or else it would be preempted. Thus, the non-preemptive section scheduled at t' must also be scheduled at t_0, and blocking by $J_{i,j}$ occurs only within $[t_0, t_b)$. □

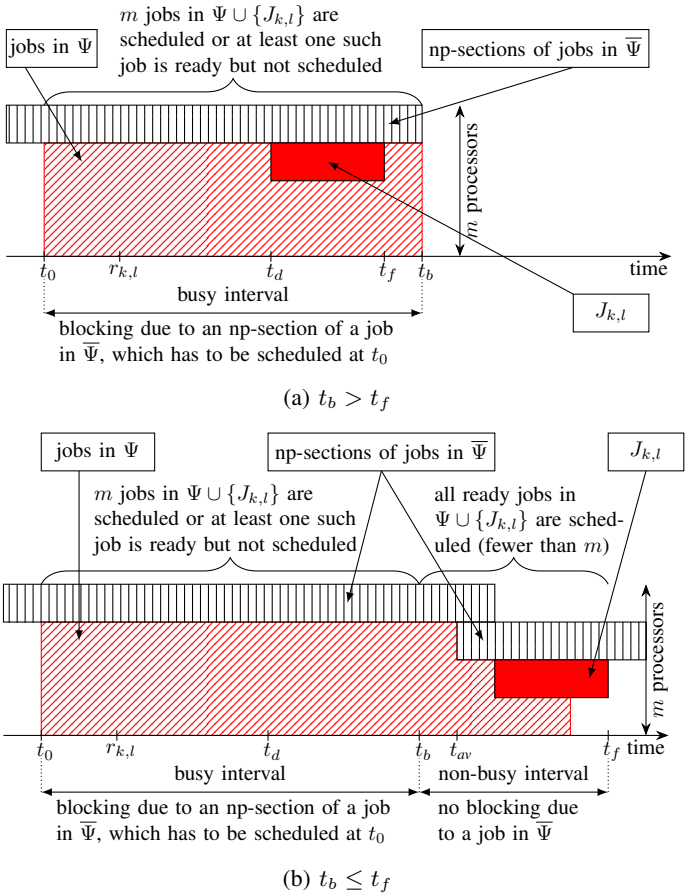

Fig. 10: Important time points in the analysis.

Let W_d be C_k plus the total workload that can potentially prevent the execution of $J_{k,l}$. By Lemma 3, W_d includes the workload of non-preemptive sections of jobs in $\overline{\Psi}$ that are scheduled at t_0 and the workload of all jobs in $\Psi \cup \{J_{k,l}\}$.

By Lemma 4, given below, $L(x)$, defined next, is an upper bound for W_d.

$$L(x) = (m-1)C_{max} + B_{max} + \max_{\substack{\tau^* \subseteq \tau \text{ s.t.} \\ \sum_{\tau_i \in \tau^*} P_i \leq m-1}} \left(\sum_{\tau_i \in \tau^*} (u_i x + 2C_i) \right) \tag{2}$$

Lemma 4. *If t_d is a busy time instant, and the response time of each job $J_{i,j} \in \Psi$ is at most $x + T_i + C_i$, then $W_d \leq L(x)$.*

Proof. Let t_0^- be $t_0 - \varepsilon$ for an arbitrarily small $\varepsilon > 0$ such that $[t_0^-, t_0)$ is a non-busy interval, as illustrated in Fig. 11. (If $t_0 = 0$, then we can conceptually view $[-\varepsilon, 0)$ as an interval where no work is scheduled.) Because ε is arbitrarily small, no scheduling events (jobs completions or releases) occur within $[t_0^-, t_0)$. To upper bound W_d, we first bound the workload at t_0 of jobs released before t_0^- in Claims 1 and 2 (all jobs in $\Psi \cup \{J_{k,l}\}$ that are ready at t_0^- are scheduled). Then we bound the workload of jobs released within $[t_0, t_d)$ in Claim 3. Finally, we bound the workload completed over $[t_0, t_d)$ in Claim 4. (For clarity, claim proofs end with ■ while other proofs end with □.)

Let a (resp., b) be the number of jobs in $\Psi \cup \{J_{k,l}\}$ (resp., $\overline{\Psi}$) that are scheduled at t_0^-.

Claim 1. *Consider the jobs that are scheduled at t_0^-. Their total non-completed workload at t_0 is at most $aC_{max} + bB_{max}$.*

Proof. By Lemma 3, only non-preemptive sections of jobs in $\overline{\Psi}$ can block the execution of jobs in $\Psi \cup \{J_{k,l}\}$. The maximal length of a non-preemptive section is B_{max}, and the number of such sections is b. The maximal workload of any job in $\Psi \cup \{J_{k,l}\}$ is bounded by C_{max}, and the number of such jobs scheduled at t_0^- is a. The total non-completed workload due to these jobs is upper bounded by $aC_{max} + bB_{max}$. ■

Let τ^* be the set of all tasks that have jobs in $\Psi \cup \{J_{k,l}\}$ that are pending but not ready at t_0^-.

Claim 2. *Consider the pending jobs in $\Psi \cup \{J_{k,l}\}$ that are not ready at t_0^-. Their total workload at t_0 is at most $\sum_{\tau_i \in \tau^*} (u_i x + 2C_i)$.*

Proof. Let s_i be the number of jobs of a task $\tau_i \in \tau^*$ that are pending at t_0^-. By the definition of τ^*, some jobs of τ_i are pending but not ready at t_0^-. Thus, certain preceding jobs of τ_i are not completed at t_0^-. By the definition of P_i and job readiness, the first P_i pending jobs of τ_i are ready, because P_i jobs of τ_i can be scheduled in parallel. Thus, $s_i > P_i$. Note that the first P_i of these jobs are scheduled at t_0^- (t_0^- is a non-busy instant). Let $J_{i,j}$ be the earliest pending job of τ_i at t_0^-. Then $J_{i,j}$ is ready at t_0^-, and $J_{i,j} \neq J_{k,l}$, or else $\tau_k \notin \tau^*$ (as all pending jobs of τ_k in $\Psi \cup \{J_{k,l}\}$ would be ready). Thus, $J_{i,j} \in \Psi$. Also, because s_i jobs of τ_i are pending at t_0^-,

$$r_{i,j} \leq t_0^- - (s_i - 1)T_i. \tag{3}$$

Since $J_{i,j} \in \Psi$, it is completed by time $r_{i,j} + x + T_i + C_i$. Because $J_{i,j}$ is pending at t_0^-, $r_{i,j} + x + T_i + C_i \geq t_0^-$, or

$$r_{i,j} \geq t_0^- - x - C_i - T_i. \tag{4}$$

Combining (3) and (4), $t_0^- - x - C_i - T_i \leq t_0^- - (s_i - 1)T_i$, which implies $s_i - 2 \leq (x + C_i)/T_i$, which in turn implies

$$s_i \leq x/T_i + u_i + 2. \tag{5}$$

As the first P_i pending jobs of τ_i at t_0^- are ready, the total workload at t_0 of the jobs pending but not ready at t_0^- is

$$\begin{aligned}(s_i - P_i)C_i &\leq \{\text{by (5)}\} \\ &\quad (x/T_i + u_i + 2 - P_i)C_i \\ &= \{C_i/T_i = u_i\} \\ &\quad u_i x + 2C_i + (u_i - P_i)C_i \\ &\leq \{\tau \text{ is feasible, so by (1), } u_i \leq P_i\} \\ &\quad u_i x + 2C_i.\end{aligned}$$

Combining over all tasks in τ^*, we have a total workload of at most $\sum_{\tau_i \in \tau^*} (u_i x + 2C_i)$, as claimed. ■

Claim 3. *Consider the jobs in $\Psi \cup \{J_{k,l}\}$ that are not released at t_0^-. Their total generated workload over $[t_0, t_d)$ is at most $U(t_d - t_0)$.*

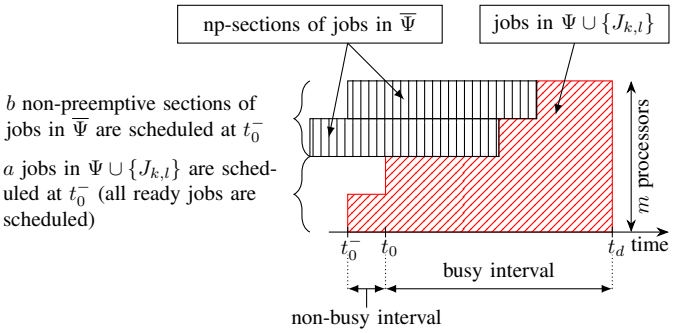

Fig. 11: Lemma 4 illustration.

Proof. All jobs in $\Psi \cup \{J_{k,l}\}$ have deadlines at or before t_d. The jobs of a task τ_i with releases and deadlines within $[t_0, t_d)$ generate a workload of at most $\lfloor (t_d - t_0)/T_i \rfloor C_i \leq u_i(t_d - t_0)$. Summing over all such jobs of all tasks in τ yields the claim. ■

Claim 4. *The workload completed in $[t_0, t_d)$ is $m(t_d - t_0)$.*

Proof. By Def. 4, $t_0 \leq t_d$ and $[t_0, t_d)$ is a busy interval, so the total completed workload is $m(t_d - t_0)$. ■

Now we can finally bound W_d:

$$\begin{aligned}W_d = &\quad \text{Workload at } t_0 \text{ of jobs scheduled at } t_0^- \\ &+ \text{Workload at } t_0 \text{ of jobs pending but} \\ &\quad \text{not ready at } t_0^- \\ &+ \text{Workload at } t_d \text{ of jobs released after } t_0^- \\ &- \text{Workload completed within } [t_0, t_d) \\ \leq &\ \{\text{by Claims 1-4}\} \\ &\quad aC_{max} + bB_{max} + \sum_{\tau_i \in \tau^*} (u_i x + 2C_i) \\ &\quad + U(t_d - t_0) - m(t_d - t_0) \\ \leq &\ \{\tau \text{ is feasible, so } U \leq m\} \\ &\quad aC_{max} + bB_{max} + \sum_{\tau_i \in \tau^*} (u_i x + 2C_i)\end{aligned} \tag{6}$$

Note that, by the definition of t_0^-, at least one processor is not occupied with a job from $\Psi \cup \{J_{k,l}\}$ at t_0^-, so $a \leq (m-1)$. Additionally, the total number of scheduled jobs at t_0^- cannot exceed m. Thus, because $B_{max} \leq C_{max}$, we have

$$aC_{max} + bB_{max} \leq (m-1)C_{max} + B_{max}. \tag{7}$$

Also, any task $\tau_i \in \tau^*$ has exactly P_i ready jobs scheduled at t_0^-, while their total number is at most $(m-1)$. Thus,

$$\sum_{\tau_i \in \tau^*} P_i \leq m - 1. \tag{8}$$

Combining (6), (7) and (8), and recalling (2), we get $W_d \leq (m-1)C_{max} + B_{max} + \sum_{\tau_i \in \tau^*} (u_i x + 2C_i) \leq L(x)$. □

Lemma 5. *If t_d is a busy time instant, and the response time of each job $J_{i,j} \in \Psi$ is at most $x + T_i + C_i$, where*

$$mx \geq L(x), \tag{9}$$

then the response time of $J_{k,l}$ is bounded by $x + T_k + C_k$.

Proof. Note that under G-EDF, $J_{k,l}$ cannot be preempted after its deadline t_d (which is T_k time units after $J_{k,l}$'s release). Thus, it is enough to prove that $J_{k,l}$ is scheduled at some point within $[t_d, t_d + x]$.

Let t_{av} ("av" means a processor is available—see Fig. 10(b)) denote the first time instant after t_d such that some processor exists that is not executing a job in $\Psi \cup \{J_{k,l}\}$ or any non-preemptive section of a job in $\overline{\Psi}$ that is scheduled at time t_0 (and hence executes continually in $[t_0, t_{av}]$). Note that $t_b \leq t_{av}$. We consider three cases, depending on how much processor allocation $J_{k,l}$ receives within $[t_0, t_{av})$.

Case 1. $J_{k,l}$ is completed before t_{av}.

In this case, the response time of $J_{k,l}$ is bounded by $t_{av} - r_{k,l} = t_{av} - t_d + T_k$. Note that $t_{av} \leq t_d + W_d/m$ (W_d is the workload that keeps all processors busy), so by Lemma 4, $t_{av} - t_d \leq L(x)/m$, which, by (9), implies $t_{av} - t_d \leq mx/m = x$. This ensures a response-time bound of $x + T_k + C_k$ for $J_{k,l}$.

Case 2. $J_{k,l}$ is ready at t_{av}.

Let δ denote the remaining amount of execution for $J_{k,l}$ at t_{av}. Because the total remaining workload from jobs in $\Psi \cup \{J_{k,l}\}$ at t_d is W_d, at most $W_d - \delta$ of this workload can be completed within $[t_d, t_{av})$. Hence, $t_{av} - t_d \leq (W_d - \delta)/m$. By Lemma 4, $W_d \leq L(x)$, so $t_{av} - t_d \leq (L(x) - \delta)/m$. By Lemma 3, $J_{k,l}$ cannot be blocked by jobs or non-preemptive sections that do not contribute to W_d, so $J_{k,l}$ is scheduled in $[t_{av}, t_{av} + \delta)$, and $t_{av} + \delta - r_{k,l} = t_{av} + \delta - t_d + T_k$ is the response time of $J_{k,l}$. Because

$$\begin{aligned} t_{av} - t_d + \delta &\leq (L(x) - \delta)/m + \delta \\ &= L(x)/m + \delta(1 - 1/m) \\ &\leq \{\text{by (9)}\} \\ &\quad mx/m + \delta(1 - 1/m) \\ &\leq \{\delta \leq C_k\} \\ &\quad x + C_k, \end{aligned}$$

the response time of $J_{k,l}$ is at most $x + C_k + T_k$.

Case 3. $J_{k,l}$ is not ready at t_{av}.

In this case, $J_{k,l-P_k}$ (which is in $\Psi \cup \{J_{k,l}\}$) is not finished by t_{av}. This predecessor is released at the latest by time $t_d - (P_k+1)\cdot T_k$. By the lemma statement, $J_{k,l-P_k}$ completes at the latest by $t_d - (P_k+1)\cdot T_k + x + T_k + C_k = t_d + x - P_k \cdot T_k + C_k$. By (1), $C_k \leq P_k \cdot T_k$, so $J_{k,l}$ is ready at the latest by $t_d + x$. By Lemma 3, $J_{k,l}$ is not blocked by any job at t_{av}, because $t_b \leq t_{av}$. That ensures the response-time bound. □

We now can conclude both the busy and the non-busy t_d cases in the following theorem.

Theorem 1. *Every job $J_{i,j}$ of every task $\tau_i \in \tau$ completes within $x + T_i + C_i$ time units after its release for any $x > 0$ such that x satisfies (9).*

Proof. Follows by induction over $\prec$, applying Lemma 2 or Lemma 5. □

We now introduce some terminology that is used in obtaining a closed-form expression for x that it is of relevance in the context of the processing graphs that motivate this work.

Def. 5. *Call a task τ_i* p-restricted *(parallelism-restricted) if $P_i < m$, and* non-p-restricted *if $P_i \geq m$. Also, let*

$$U^b_{res} = \sum_{\substack{b \text{ largest values} \\ \tau_i \text{ is p-restricted}}} u_i \quad \text{and} \quad C^b_{res} = \sum_{\substack{b \text{ largest values} \\ \tau_i \text{ is p-restricted}}} C_i,$$

and let $U_{res} = U^n_{res}$ and $C_{res} = C^n_{res}$.

Corollary 1. *The response time of any task $\tau_i \in \tau$ is bounded by $x + T_i + C_i$, where*

$$x = \frac{(m-1)C_{max} + B_{max} + 2C_{res}}{m - U_{res}}. \tag{10}$$

Furthermore, if there exists $P_{min} \geq 1$ such that for every p-restricted task τ_i, $P_i \geq P_{min}$, then U_{res} and C_{res} in (10) can be replaced with U^ℓ_{res} and C^ℓ_{res}, where $\ell = \lfloor (m-1)/P_{min} \rfloor$.

Proof. Note that the task subset τ^* in (2) consists of only p-restricted tasks, because $\sum_{\tau_i \in \tau^*} P_i \leq m - 1$ (see (8)), while $P_i \geq m$ for any non-p-restricted task. Thus,

$$\begin{aligned} &\max_{\substack{\tau^* \subseteq \tau \text{ s.t.} \\ \sum_{\tau_i \in \tau^*} P_i \leq m-1}} \left(\sum_{\tau_i \in \tau^*} (u_i x + 2C_i) \right) \\ &= \max_{\substack{\tau^* \text{ consists of p-restricted tasks} \\ \sum_{\tau_i \in \tau^*} P_i \leq m-1}} \left(\sum_{\tau_i \in \tau^*} (u_i x + 2C_i) \right) \\ &\leq \sum_{\tau_i \text{ is a p-restricted task}} (u_i x + 2C_i) \\ &= U_{res} x + 2C_{res}. \end{aligned}$$

Hence, by (2), $L(x) \leq (m-1)C_{max} + B_{max} + U_{res}x + 2C_{res}$. Because, by (10), $mx = (m-1)C_{max} + B_{max} + U_{res}x + 2C_{res} \geq L(x)$, x satisfies (9). Therefore, by Theorem 1, $x + T_i + C_i$ is a response-time bound for any task τ_i.

If for every p-restricted task τ_i, $P_i \geq P_{min}$, then $|\tau^*| \leq \lfloor (m-1)/P_{min} \rfloor$, as $\sum_{\tau_i \in \tau^*} P_i \leq m - 1$. In this case, only the $\lfloor (m-1)/P_{min} \rfloor$ p-restricted tasks with the highest corresponding values have to be considered in U_{res} and C_{res}. □

Recall that we are interested in rp-sporadic tasks obtained via our graph-transformation process. Tasks corresponding to supernodes will generally be p-restricted, while other tasks will not. Hence, the corollary above is useful in our context.

The results of this section provide clear tradeoffs. For example, if an OpenVX graph has a cycle with utilization exceeding 1.0 that must execute sequentially, then bounded response times for that graph cannot be ensured. Our analysis shows that, by allowing parallelism within such a cycle, this result can be reversed. Furthermore, Corollary 1 shows that response-time bounds can be lowered by increasing P_i values, *i.e.*, by sacrificing some accuracy.

B. Improved Bounds

The basic bound just derived can be improved via several techniques that we omitted above due to space constraints. We briefly mention those techniques here.

Improved definition of a busy time instant. We could replace m with $m^+ = \lceil U \rceil$. This change would yield a significant improvement for low-utilization task sets.

Accurate accounting of ready jobs. In Claim 1 of Lemma 4, we bounded the maximal workload of any ready job at t_0^- as C_{max}. However, this could be reduced with a more precise accounting of ready jobs, yielding an improvement for task sets for which the highest-WCET tasks are p-restricted.

Compliant-vector analysis. We considered every task to have the same value x. We could instead apply compliant-vector analysis [15, 16], which assigns a distinct x_i to each task τ_i.

GEL schedulers. The provided analysis easily extends to any GEL (G-EDF-like) scheduler. Such a scheduler prioritizes each job by a *priority point*, a point in time a constant distance from its release. Under arbitrary GEL scheduling, response times can be lowered by determining priority points via linear optimization [43]. Also, as FIFO is a GEL scheduler, the same analysis can be applied for FIFO-scheduled GPUs.

C. GPUs as Schedulable Entities

The final comment above suggests the possibility of considering GPUs as schedulable entities instead of synchronization objects as we have done. However, the former creates some surprising analysis difficulties, as illustrated next.

Ex. 5. Consider the cycle depicted in Fig. 12 in a system with one CPU and one GPU. Observe that the total utilization of this cycle is 1.0. However, both the CPU and the GPU are not fully utilized. Thus, there could exist other GPU work on the same platform that causes some amount of blocking for the GPU task in the figure. When considering this cycle from a CPU point of view, where time spent accessing the GPU (including both execution and blocking) is viewed as suspension time away from the CPU, the GPU blocking results in an overloaded system and unbounded response times. ◊

As this example suggests, it turns out that, with GPUs considered as schedulable entities, we must consider a given cycle from both a CPU perspective—in which case time accessing a GPU is suspension time away from CPU execution—and from a GPU perspective—in which case time executing on a CPU is suspension time away from GPU execution. Determining such suspension times requires determining GPU and CPU response times, respectively. Thus, we have a circularity: in order to determine CPU and GPU response times, we need to know CPU and GPU response times! Note that this circularity is unique to nodes within cycles—other nodes are not so affected.

While this circularity may seem rather devastating, we have actually devised several workarounds to it, but we lack sufficient space to explain them. In any event, we mention this issue here to provide some indication as to why we opted for the simpler synchronization-based approach in this first work on dealing with arbitrary cycles in OpenVX graphs.

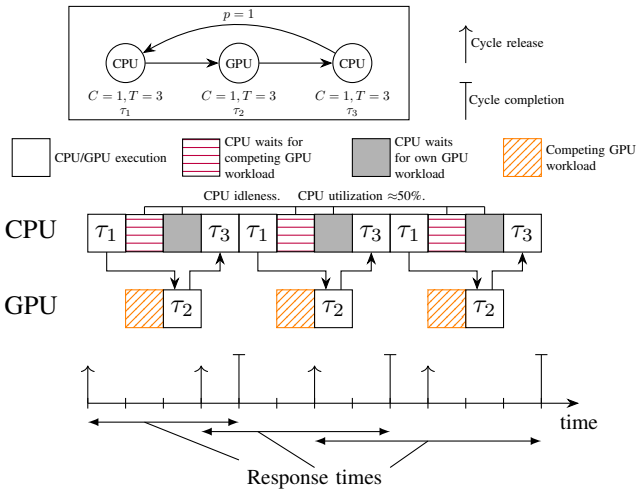

Fig. 12: A schedule of the cycle from Ex. 5.

V. Case Study Evaluation

We evaluated our approach via a case study of a CV pedestrian-tracking application for which the graph contains a cycle. In this section, we describe our pedestrian-tracking experimental setup, and then present the results of varying the minimum history requirement for the cycle induced by tracking, and discuss the effects on analytical and observed response times and on the tracking accuracy.

A. Experimental Setup

We chose for our case study a pedestrian-tracking application using the *Histogram of Oriented Gradients (HOG)* method for detecting pedestrians from camera image frames. This type of application would be important in an ADAS, as it enables the car to take action depending on the trajectories of pedestrians or other dynamic obstacles.

Pedestrian tracking via HOG. HOG computes gradients within the image at a range of different scales, and classifies potential detections at each scale. The computational cost increases with the number of image scales, but each scale enables detection of a pedestrian at a different distance from the camera. We used as a starting point the HOG implementation evaluated in prior work by our group [44]. As in [14] and [44], we used PGM$^{\mathrm{RT}}$ [13] to handle data passing, and employed schedulers provided by LITMUS$^{\mathrm{RT}}$ [30].

The features computed by HOG are provided to a classifier such as a support vector machine, which determines whether a potential detection is a pedestrian. The output is a series of rectangles of varying sizes and positions. Over time (*i.e.*, frames of the video), detections of a given pedestrian can be matched to form a *track* of positions. This process requires matching a current-frame detection with a track based on the prior frame (or older, if $p > 1$), resulting in a cycle.

The graphs involved in our case study are depicted in Fig. 13. As discussed later, we chose to execute HOG on the CPU as a single non-p-restricted node. The cycle introduced by tracking results in a single p-restriced supernode. In order to achieve intra-task parallelism at runtime, we replicated the

	Sequential	$p = 1$	$p = 2$	$p = 3$
Analytical Bound (ms)	N/A	N/A	927.27	928.37
Observed Maximum Response Time (ms)	25250.67	572.81	713.53	537.60
Observed Average Response Time (ms)	11765.23	293.63	280.86	293.07

TABLE I: Analytical and observed end-to-end response times. A bound of N/A indicates a violated feasibility condition.

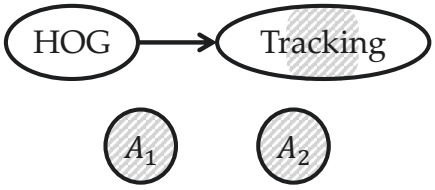

Fig. 13: Graphs comprising the case study. The tracking, A_1, and A_2 tasks all use the GPU.

HOG node $P_i = m$ times and the tracking node $P_i = p$ times in the PGM$^{\mathrm{RT}}$ graph. Each frame of the video was passed to only one of the HOG replicas, in round-robin order. Similarly, only one of the p tracking supernodes processed the resulting detections for a given frame.

Test platform. We performed our experiments on a platform with two eight-core Intel CPUs and 32 GB of DRAM. The CPU cores each have a 32-KB L1 data cache, a 32-KB L1 instruction cache, and a 1-MB L2 cache. All eight cores on a socket share an 11-MB L3 cache. The platform additionally has an NVIDIA 1070 GPU, and was configured to run Ubuntu 16.04 with the 2017.1 LITMUS$^{\mathrm{RT}}$ kernel [30].

Competing workloads. We chose as competing workloads two synthetic GPU-using tasks (A_1 and A_2 in Fig. 13) with $p = m$ that increase the blocking suffered by the tracking supernode. To measure the full effect of this contention, we ran HOG on the CPU, and configured tracking to perform computations on both the CPU and the GPU (see Fig. 13).

The HOG and tracking tasks were given a 25-ms period, corresponding to camera frames being processed at 40 frames per second (FPS) (CV applications typically target 30–60 FPS). Each competing task was given a 50-ms period and accessed the GPU for 2 ms, resulting in worst-case blocking of $B_{max} = 32$ ms for 16 processors. The number of competing tasks was chosen to be the maximum such that $U < m$.

B. Results

Our goal was to measure the impact, in terms of response times and accuracy, of varying p for a given graph in the presence of resource contention that results in a higher utilization for that graph's supernode. We compare varying values of p for just the tracking supernode to *sequential scheduling*, in which all tasks (not just tracking) have $p = 1$.

Impact of p on response times. We used FeatherTrace [6] to measure the worst-case execution times of each task, and took the 99th percentile value over $10,000$ samples.

We computed the response-time bound of each task using Corollary 1 in Sec. IV-A. The utilization constraints are violated for sequential scheduling and for $p = 1$, so no bound could be computed. The resulting end-to-end response-time bounds are listed in Table I for varying p, along with the observed worst- and average-case end-to-end response times. The response-time distributions are plotted in Fig. 14.

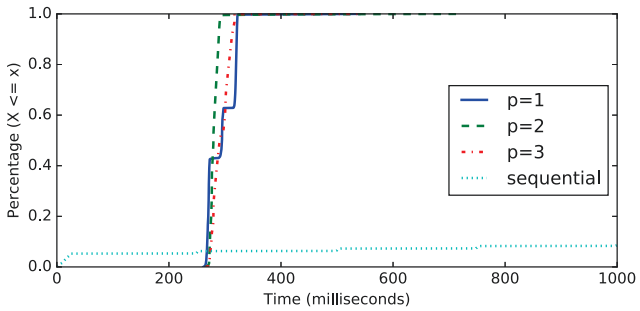

Fig. 14: CDF of observed response times for varying p.

Obs. 1. The system is unschedulable if the supernode is not replicated (p = 1) or if graph is scheduled sequentially.

Under sequential scheduling, both HOG and tracking have $p = 1$. HOG in particular has a high worst-case execution time, so the end-to-end response time of the graph far exceeded its period, and in fact grew without bound. This is evident in the observed response time in both Table I and Fig. 14. When $p = 1$, the observed response time was much better, but the inflation due to potential GPU blocking caused the tracking node to have a utilization higher than 1.0, violating the feasibility condition (1) in Sec. III.

Obs. 2. The analytical response-time bounds for $p \geq 2$ are almost identical.

This is expected behavior; due to space constraints, the bound we presented in Corollary 1 is somewhat conservative, and remains the same if p increases but the number of p-restricted tasks remains significantly smaller than m (this case study includes a single p-restricted task).

Obs. 3. The analytical response-time bounds upper-bounded the observed response times for $p \geq 2$.

This is demonstrated in Table I.

Obs. 4. For $p \geq 2$, as p increases, observed maximum (resp., average) response times decrease (resp., increase).

This trend is shown in Table I. Although intra-task parallelism allows for shorter response times in the worst case, the number of jobs competing with the job of interest at a given time increases, resulting in worse average-case behavior.

Impact of p on accuracy. Bounded response times for (previously unschedulable) cycles come at a price: accuracy drops as p increases. To fully assess the impact on accuracy, a study of multiple CV workloads with varying p values would

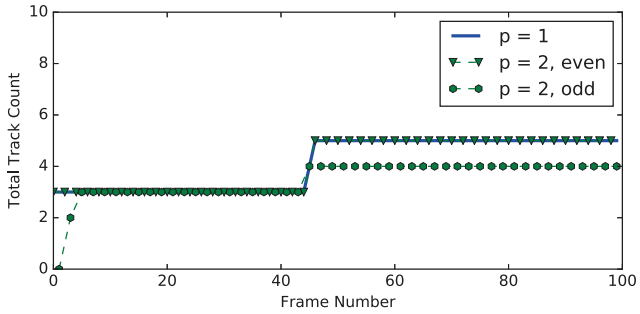

Fig. 15: Total tracks per frame for $p = 1, 2$.

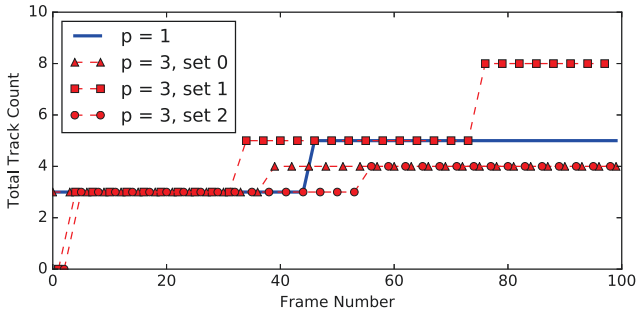

Fig. 16: Total tracks per frame for $p = 1, 3$.

be required. In this paper, we instead seek to demonstrate that allowing a small amount of restricted parallelism does not necessarily translate to a large drop in accuracy.

As p increases, the distance between the "current" position of a pedestrian and their last-tracked position increases. As a result, the track might be lost, to be started anew at a later frame. Therefore, we expect the total number of tracks maintained throughout the video (including tracks abandoned when pedestrians are "lost") to increase with increasing p.

In practice, p might represent the *maximum* age of historical results available in a given cycle, *i.e.*, newer results could be used, if available. In our experiments, however, we assume that p corresponds to the *actual* age of the historical results available in order to demonstrate the worst-case accuracy. This worst-case behavior effectively partitions the frames into distinct sets. For example, if $p = 2$, then data produced by frames $0, 2, 4, 6, \ldots$ will never be available to frames $1, 3, 5, 7, \ldots$ and vice versa; in this case, a given pedestrian corresponds to two separate tracks, one for each set of frames.

We chose as a metric for accuracy the total number of tracks maintained throughout the video, including tracks abandoned when pedestrians are lost. Given the divisions of frames based on p, we consider this total on a per-frame-set basis. Figs. 15 and 16 depict the total tracks for 100 frames of the video. The solid line indicates the total track count for $p = 1$. Fig. 15 depicts the total track counts for the two frame sets for $p = 2$, and Fig. 16 depicts the three frame sets for $p = 3$.

Obs. 5. Accuracy is comparable for $p = 1$ and $p = 2$.

This is supported by Fig. 15. For $p = 2$, the two sets of even and odd frames effectively result in two different video sequences, each with half the frame rate of the original. The even frame sequence for $p = 2$ maintains the same number of tracks as the "ground truth" of $p = 1$, and after the first few frames, the odd frame sequence tracks only one fewer pedestrian. Additionally, the two sequences for $p = 2$ only differ by at most one tracked pedestrian.

Obs. 6. Accuracy significantly decreases for $p = 3$.

This can be seen in comparing Figs. 15 and 16. For $p = 3$, pedestrians effectively move three times as far as $p = 1$ between "consecutive" frames of a given sequence. As a result, pedestrians are more frequently lost, as evidenced by the higher total track count for one of the $p = 3$ sets in Fig. 16. Furthermore, the three sequences corresponding to $p = 3$ in Fig. 16 differ greatly from each other, indicating that the results are much less stable as p increases.

As mentioned above, the results presented here assume that p represents the *exact* age of historical results available in a given cycle. If p instead represented the *maximum* age of results, then we expect that higher values of p could be used without significant impact on the accuracy. We plan to explore such implications in future work.

VI. Conclusion

We have presented the first ever work on guaranteeing response-time bounds for OpenVX graphs that have arbitrary cycles. Such graphs are crucial to consider in real-time certification processes applicable to autonomous vehicles due to the prevalence of uses cases where historical information must be tracked. Our results reveal interesting tradeoffs pertaining to graph cycles that hinge on response times, allowed parallelism, and CV accuracy. We discussed an approach to enable such tradeoffs to be explored that involves transforming an OpenVX graph to an "equivalent" sporadic task set for which allowed intra-task parallelism is a settable per-task parameter. We introduced the rp-sporadic task model to enable the formal study of such task sets, and derived response-time bounds that are applicable to any feasible task set under this model. Additionally, our work can be applied to any graph that contains a cycle, including those from motion planning and machine-learning applications.

This paper opens up many avenues for future work. First, as discussed in Secs. IV-B and IV-C, we made certain simplifying assumptions in our analysis due to space constraints; we intend to fully explore all of the options mentioned there for easing these assumptions. Second, like in prior work, our approach does not allow specifying desired response-time bounds (doing so would introduce utilization constraints). We will explore system design choices and their impacts on resulting bounds. Third, we intend to extend our experimental efforts to consider higher-level notions of accuracy in autonomous driving, such as missed obstacles when engaged in actual driving scenarios, and to perform a large-scale study of the tradeoff between response times and accuracy for a broad set of autonomous-driving applications. Finally, we intend to develop a tool that will enable CV programmers to graphically specify OpenVX programs that are then automatically transformed to fine-grained implementations with response-time analysis.

VII. Acknowledgements

The authors would like to thank Catherine Nemitz and Clara Hobbs for stimulating discussion and helpful feedback. Additionally, this work was supported by NSF grants CNS 1409175, CNS 1563845, CNS 1717589, and CPS 1837337, ARO grant W911NF-17-1-0294, and funding from General Motors.

References

[1] H. I. Ali, B. Akesson, and L. M. Pinho, "Generalized extraction of real-time parameters for homogeneous synchronous dataflow graphs," in *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2015, pp. 701–710.

[2] M. Bamakhrama and T. Stefanov, "Hard-real-time scheduling of data-dependent tasks in embedded streaming applications," in *Proceedings of the 9th ACM International Conference on Embedded Software*, 2011, pp. 195–204.

[3] S. Baruah, "Federated scheduling of sporadic DAG task systems," in *Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 179–186.

[4] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software synthesis from dataflow graphs*. Springer Science & Business Media, 2012, vol. 360.

[5] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cycle-static dataflow," *IEEE Transactions on signal processing*, vol. 44, no. 2, pp. 397–408, 1996.

[6] B. Brandenburg and J. H. Anderson, "Feather-Trace: A lightweight event tracing toolkit," in *Proceedings of the 3rd International Workshop on Operating Systems Platforms for Embedded Real-Time applications*, 2007, pp. 19–28.

[7] S. Chatterjee and J. Strosnider, "Distributed pipeline scheduling: A framework for distributed, heterogeneous real-time system design," *The Computer Journal*, vol. 38, no. 4, pp. 271–285, 1995.

[8] S. Chatterjee and J. Strosnider, "A generalized admissions control strategy for heterogeneous, distributed multimedia systems," in *Proceedings of the ACM Multimedia*, 1995, pp. 345–356.

[9] U. Devi, "Soft real-time scheduling on multiprocessors," Ph.D. dissertation, University of North Carolina at Chapel Hill, 2006.

[10] U. Devi and J. H. Anderson, "Tardiness bounds under global EDF scheduling on a multiprocessor," *Real-Time Systems*, vol. 38, no. 2, pp. 133–189, 2008.

[11] Z. Dong, C. Liu, A. Gatherer, L. McFearin, P. Yan, and J. H. Anderson, "Optimal dataflow scheduling on a heterogeneous multiprocessor with reduced response time bounds," in *Proceedings of the 29th Euromicro Conference on Real-Time Systems*, 2017, pp. 15:1–15:22.

[12] G. A. Elliott, "Real-time scheduling of GPUs, with applications in advanced automotive systems," Ph.D. dissertation, University of North Carolina at Chapel Hill, 2015.

[13] G. A. Elliott, N. Kim, J. P. Erickson, C. Liu, and J. H. Anderson, "Minimizing response times of automotive dataflows on multicore," in *Proceedings of the 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2014, pp. 1–10.

[14] G. A. Elliott, K. Yang, and J. H. Anderson, "Supporting real-time computer vision workloads using OpenVX on multicore+GPU platforms," in *Proceedings of the 36th IEEE Real-Time Systems Symposium*, 2015, pp. 273–284.

[15] J. P. Erickson and J. H. Anderson, "Response time bounds for G-EDF without intra-task precedence constraints," in *Proceedings of the 15th International Conference On Principles Of Distributed Systems*, 2011, pp. 128–142.

[16] J. P. Erickson, N. Guan, and S. Baruah, "Tardiness bounds for global EDF with deadlines different from periods," in *Proceedings of the 14th International Conference On Principles Of Distributed Systems*, 2010, pp. 286–301.

[17] S. Goddard, "On the management of latency in the synthesis of real-time signal processing systems from processing graphs," Ph.D. dissertation, University of North Carolina at Chapel Hill, 1998.

[18] J. P. Hausmans, M. H. Wiggers, S. J. Geuns, and M. J. Bekooij, "Dataflow analysis for multiprocessor systems with non-starvation-free schedulers," in *Proceedings of the 16th International Workshop on Software and Compilers for Embedded Systems*, 2013, pp. 13–22.

[19] C.-J. Hsu and S. S. Bhattacharyya, "Cycle-breaking techniques for scheduling synchronous dataflow graphs," Institute for Advanced Computer Studies, University of Maryland, Tech. Rep., 2007.

[20] X. Jiang, N. Guan, X. Long, and W. Yi, "Semi-federated scheduling of parallel real-time tasks on multiprocessors," in *Proceedings of the 38th IEEE Real-Time Systems Symposium*, 2017, pp. 80–91.

[21] K. Lakshmanan, S. Kato, and R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *Proceedings of the 31st IEEE Real-Time Systems Symposium*, 2010, pp. 259–268.

[22] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[23] H. Leontyev and J. H. Anderson, "Generalized tardiness bounds for global multiprocessor scheduling," in *Proceedings of the 28th IEEE Real-Time Systems Symposium*, 2007, pp. 413–422.

[24] H. Leontyev and J. H. Anderson, "Tardiness bounds for FIFO scheduling on multiprocessors," in *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, 2007, pp. 71–80.

[25] H. Leontyev and J. H. Anderson, "Generalized tardiness bounds for global multiprocessor scheduling," *Real-Time Systems*, vol. 44, no. 1-3, pp. 26–71, 2010.

[26] J. Li, "Parallel real-time scheduling for latency-critical applications," Ph.D. dissertation, Washington University in St. Louis, 2017.

[27] J. Li, K. Agrawal, C. Gill, and C. Lu, "Federated scheduling for stochastic parallel real-time tasks," in *Proceedings of the 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2014, pp. 1–10.

[28] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," *Real-Time Systems*, vol. 53, no. 5, pp. 760–811, 2017.

[29] J. Li, Z. Luo, D. Ferry, K. Agrawal, C. Gill, and C. Lu, "Global EDF scheduling for parallel real-time tasks," *Real-Time Systems*, vol. 51, no. 4, pp. 395–439, 2015.

[30] LITMUS$^{\mathrm{RT}}$ Project, http://www.litmus-rt.org/.

[31] C. Liu and J. H. Anderson, "Supporting soft real-time DAG-based systems on multiprocessors with no utilization loss," in *Proceedings of the 31st IEEE Real-Time Systems Symposium*, 2010, pp. 3–13.

[32] C. Liu and J. H. Anderson, "Supporting graph-based real-time applications in distributed systems," in *Proceedings of the 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2011, pp. 143–152.

[33] C. Liu and J. H. Anderson, "Supporting soft real-time parallel applications on multicore processors," in *Proceedings of the 18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2012, pp. 114–123.

[34] Y. Liu, X. Zhang, H. Li, and D. Qian, "Allocating tasks in multi-core processor based parallel system," in *Proceedings of the 4th IFIP International Conference on Network and Parallel Computing Workshops*, 2007, pp. 748–753.

[35] S. Niknam, P. Wang, and T. Stefanov, "Hard real-time scheduling of streaming applications modeled as cyclic csdf graphs," in *The 23rd Design, Automation & Test in Europe Conference & Exhibition*, 2019, pp. 1549–1554.

[36] Naval Research Laboratory, "Processing graph method specification," 1987.

[37] The Khronos Group, "OpenVX: Portable, Power Efficient Vision Processing," Online at https://www.khronos.org/openvx/.

[38] The Khronos Group, "The OpenVX Specification," Online at https://www.khronos.org/registry/OpenVX/specs/1.2.1/OpenVX_Specification_1_2_1.html#sub_graphs_rules.

[39] H. Rihani, M. Moy, C. Maiza, R. I. Davis, and S. Altmeyer, "Response time analysis of synchronous data flow programs on a many-core processor," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 67–76.

[40] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *Proceedings of the 32nd IEEE Real-Time Systems Symposium*, 2011, pp. 217–226.

[41] N. Ueter, G. von der Brüggen, J.-J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *Proceedings of the 39th IEEE Real-Time Systems Symposium*, 2018, pp. 482–494.

[42] K. Yang, G. A. Elliott, and J. H. Anderson, "Analysis for supporting real-time computer vision workloads using OpenVX on multicore+GPU platforms," in *Proceedings of the 23th International Conference on Real-Time Networks and Systems*, 2015, pp. 77–86.

[43] K. Yang, M. Yang, and J. H. Anderson, "Reducing response-time bounds for DAG-based task systems on heterogeneous multicore platforms," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 349–358.

[44] M. Yang, T. Amert, K. Yang, N. Otterness, J. H. Anderson, F. D. Smith, and S. Wang, "Making OpenVX really 'Real Time'," in *Proceedings of the 39th IEEE Real-Time Systems Symposium*, 2018, pp. 80–93.